Machine Cell Formation Using a Mathematical Model and a Genetic Algorithm Based Heuristic

Published in 2006 in the International Journal of Production Research, Vol. 44, 2421 -2444

Please cite this article as:

Defersha, F. M., and Chen, M., (2006). Machine Cell Formation Using a Mathematical Model and a Genetic Algorithm Based Heuristic. International Journal of Production Research. Vol. 44, No. 12, 2421–2444

The online version can be found at the following link:

http://dx.doi.org/10.1080/00207540500337833

Machine cell formation using a mathematical model and a genetic algorithm based heuristic

FANTAHUN M. DEFERSHA[†] and MINGYUAN CHEN [†‡]

This paper presents a comprehensive mathematical model and a genetic algorithm based heuristic for the formation of part families and machine cells in the design of cellular manufacturing systems. The model incorporates dynamic cell configuration, alternative routings, sequence of operations, multiple units of identical machines, machine capacity, workload balancing among cells, operation cost, subcontracting cost, tool consumption cost, setup cost and other practical constraints. To solve this model efficiently, a two-phase genetic algorithm based heuristic was developed. In the first phase, independent cells are formed which are relatively simple to generate. In the second phase, the solution found during the first phase is gradually improved to generate cells optimizing inter cell movement and other cost terms of the model. A number of numerical examples of different sizes are presented to demonstrate the computational efficiency of the heuristic developed.

Keywords: Cellular Manufacturing, Dynamic Cell Configuration, Alternate Routings, Workload Balancing, Integer Programming, Genetic Algorithm

1. Introduction

Cellular manufacturing involves processing a collection of similar parts (part families) on a dedicated cluster of machines (machine cells) (Singh, 1993). By grouping similar parts (same set-up, or processing, or routing, etc.), one can take advantage of their similarities in design and manufacturing. Similarly, by grouping machines together into relatively independent cells, inter-cell movement and material handling costs can be reduced. Furthermore, reductions in set-up time, manufacturing lead time, design variety, work-in-process inventory and labor can be achieved (Wemmerlöv U. and Johnson, 1997; Sule, 1994; Singh, 1996). Better supervision and improved quality are also among the reported advantages of the implementation of cellular manufacturing. In the last three decades, research in cellular

[†]Concordia University, Department of Mechanical and Industrial Engineering, 1455 de Maisonneuve Blvd., W. Montreal, PQ, Canada H3G 1M8

[‡]For correspondence: mychen@me.concordia.ca, Tel: (514) 848-2424 Ext. 3134; Fax: (514) 848-3175

manufacturing has been extensive and literature in this area is abundant. Comprehensive summaries and taxonomies of studies devoted to part-machine grouping problems are presented by various researchers (e.g., Wemmerlöve and Hayer (1986), Kusiak (1987), Vakharia and Slim (1994) and Selim *et. al.* (1998)). Typically, there are two types of methods developed for solving this problem: the binary machine-part matrix method and the clustering method. The binary machine-part matrix method was first developed by King (1980) and King and Nakornchai (1982) and is the most cited research work in cellular manufacturing. The clustering method is based on similarity coefficients of the parts which was first explored in McAuley (1972).

A number of factors are typically considered in manufacturing cell formation and part family identification problem. These factors include dynamic cell reconfiguration, sequence of operations, alternate part routings, operation time and cost, subcontracting cost, machine capacity, setup cost, tool consumption, workload balancing, cell size limits, machine separation constraints, among others. In recent years, machine cell formation methods addressing subsets of these factors were developed by several researchers separately. Some of these methods include Caux *et. al.* (1990), Chen (1998), Mungwattana (2000), Suresh *et. al.* (1999) and Wicks and Reasor (1999). In this paper, a mathematical programming model incorporating a majority of these factors is proposed. In practice, exact solutions of combinatorial optimization problems of larger sizes are often difficult to compute. Machine cell formation is a typical example of such problems. In the present study, heuristic method based on genetic algorithm is developed and employed to find optimal or near optimal solutions of the cell formation problem. Genetic algorithm (GA) introduced in 1970s by J. Holland (Holland, 1975) has gained increasing popularity in solving many combinatorial optimization problems. A number of works are also published reposting the development and use of genetic algorithm to solve the part family machine cell formation problem. Recent works include Wicks and Reasor (1999), Dimopoulos and Mort (2001), Gonçalves and Resendeb (2004), Solimanpury *et. al.* (2004), Rogers and Kulkarni (2005) among others. The distinguishing future of the genetic algorithm presented in this paper is that it is for solving a mathematical model addressing simultaneously several pragmatic issues in the design of a CMS. In general, genetic algorithm starts with an initial population of solutions. Genetic operators are applied to this population; offsprings are created from parents. The new population is constituted in select-

ing the best individuals. This process continues until a certain stoping criterion is met. The genetic algorithm presented in this paper follows this general procedure and has a number of problem specific genetic operators, heuristics and two searching phases. The remainder of this paper is organized as follows. Section 2 provides the problem description and the proposed mathematical model. Detailed discussion of the solution procedure developed is presented in Section 3. Section 4 presents three numerical examples to illustrate the heuristic solution procedure, and to evaluate the effectiveness and efficiency of the developed method. Finally discussion and conclusions are presented in Section 5.

2. The Proposed Mathematical Model

2.1. Problem Description

Consider a manufacturing system consisting of a number of machines. Each machine has a number of tools available. A part may require several operations in sequence. An operation of a part can be processed by a machine if the required tool is available on that machine. If a tool is available on more than one machine type then those types of machines are considered as alternative routings for processing that part. In addition, we consider the manufacturing system in a number of time periods t, where $t = 1, 2, ..., T$. One time period could be a month, a season, or a year. Each machine has a limited capacity expressed in hours during each period and can have one or more identical copies to meet capacity requirements. We assume that the demand for the parts varies with t in a deterministic manner and is known. To group machines into relatively independent cells, a mixed integer programming model is formulated. The notations used are presented below.

Indexes:

t - Time index, $t = 1, 2, ..., T$,

i - Part index, $i = 1, 2, ..., I$,

j - Index of operations of part i, $j = 1, 2, ..., J_i$,

k - Machine index, $k = 1, 2, ..., K$,

g - Tool index, $g = 1, 2, ..., G$,

l - Cell index, $l = 1, 2, ..., L$.

Input Data:

$d_i(t)$ - Demand for part i in time period t,

V_i - Unit cost to move part i between cells,

B_i - Batch size of part type i,

Φ_i - Cost of subcontracting part i,

λ_{jig} - A binary number indicating whether operation j of part i requires tool g or not,

δ_{gk} - A binary number indicating whether tool g is available on machine k or not,

h_{jik} - Processing time of operation j of part i on type k machine in minutes,

w_{jik} - Tool consumption cost of operation j of part i per unit of part i on machine type k,

μ_{jik} - Setup cost for operation j of part i on type k machine,

P_k - Procurement cost of type k machine,

H_k - Maintenance and other overhead cost of type k machine,

O_k - Operation cost per hour of type k machine,

C_k - Capacity of one unit of type k machine,

LB_l - Minimum number of machines in cell l,

UB_l - Maximum number of machines in cell l,

I_k^+ - Cost of installing one unit of type k machine,

I_k^- - Cost of removing one unit of type k machine,

q - $0 \leq q \leq 1$; a factor describing that the work load of a cell can be as low as $q \times 100\%$ from the average work load per cell,

M^∞ - Large positive number,

S - Set of machine pairs that cannot be placed in the same cell.

Decision Variables:

General Integer:

$N_{kl}(t)$ - Number of type k machines assigned to cell l at the beginning of period t,

$y_{kl}^+(t)$ - Number of type k machines added to cell l at the beginning of period t,

$y_{kl}^-(t)$ - Number of type k machines removed from cell l at the beginning of period t.

Continuous:

$\eta_{jikl}(t)$ - The proportion of the total demand of part i with its j^{th} operation performed by type k machine in cell l during period t,

$\bar{\eta}_i(t)$ - The proportion of the total demand of part i to be subcontracted in time period t.

Auxiliary Binary Integer Variables:

$$r_{kl}(t) = \begin{cases} 1, & \text{if type } k \text{ machines are assigned to cell } l \text{ during time period } t, \\ 0, & \text{otherwise.} \end{cases}$$

$$p_{jil}(t) = \begin{cases} 1, & \text{if operation } j \text{ of part } i \text{ is processed in cell } l \text{ during period } t, \\ 0, & \text{otherwise.} \end{cases}$$

2.2. Objective Function and Constraints

Following the problem description and notations given in Section 2.1, the comprehensive mixed integer programming model for cellular manufacturing system design is presented below.

Objective:

Minimize $Z =$

$$\sum_{t=1}^{T}\sum_{l=1}^{L}\sum_{k=1}^{K} N_{kl}(t)\cdot H_k + \sum_{t=1}^{T}\sum_{k=1}^{K} P_k \cdot \left(\sum_{l=1}^{L} N_{kl}(t) - \sum_{l=1}^{L} N_{kl}(t-1)\right)$$

$$+\frac{1}{2}\sum_{t=1}^{T}\sum_{l=1}^{L}\sum_{i=1}^{I}\sum_{j=1}^{J_i-1}\left(d_i(t)\cdot V_i\left|\sum_{k=1}^{K}\eta_{i,j+1,kl}(t) - \sum_{k=1}^{K}\eta_{ijkl}(t)\right|\right)$$

$$+\sum_{t=1}^{T}\sum_{l=1}^{L}\sum_{k=1}^{K}\sum_{i=1}^{I}\sum_{j=1}^{J_i} d_i(t)\cdot\eta_{ijkl}(t)\cdot h_{ijk}(t)\cdot O_k + \sum_{t=1}^{T}\sum_{l=1}^{L}\sum_{k=1}^{K}\sum_{i=1}^{I}\sum_{j=1}^{J_i} d_i(t)\cdot\eta_{ijkl}(t)\cdot w_{ijk}$$

$$+\sum_{t=1}^{T}\sum_{l=1}^{L}\sum_{k=1}^{K}\sum_{i=1}^{I}\sum_{j=1}^{J_i}\frac{d_i(t)\cdot\eta_{ijkl}(t)}{B_i}\cdot\mu_{ijk} + \sum_{t=1}^{T}\sum_{l=1}^{L}\sum_{k=1}^{K}\left(I_k^+\cdot y_{kl}^+(t) + I_k^-\cdot y_{kl}^-(t)\right)$$

$$+\sum_{t=1}^{T}\sum_{i=1}^{I}\Phi_i\cdot d_i(t)\cdot\bar{\eta}_i(t) \qquad (1)$$

Subject to:

$$d_i(t) \cdot \sum_{l=1}^{L} \sum_{k=1}^{K} \eta_{ijkl}(t) = d_i(t)(1 - \bar{\eta}_i(t)) \tag{2}$$

$$\eta_{ijkl}(t) \leq \lambda_{jig} \times \delta_{gk} \tag{3}$$

$$\sum_{k=1}^{K} \eta_{jikl}(t) = p_{jil}(t) \tag{4}$$

$$\sum_{l=1}^{L} p_{jil}(t) = 1 \tag{5}$$

$$C_k \cdot N_{kl}(t) - \sum_{i=1}^{I} \sum_{j=1}^{J_i} d_i(t) \cdot \eta_{ijkl}(t) \cdot h_{jik} \geq 0 \tag{6}$$

$$\sum_{l=1}^{L} N_{kl}(t) - \sum_{l=1}^{L} N_{kl}(t-1) \geq 0 \tag{7}$$

$$\sum_{k=1}^{K} \sum_{i=1}^{I} \sum_{j=1}^{J_i} d_i(t) \cdot \eta_{ijkl}(t) \cdot h_{ijk} \geq \frac{q}{L} \sum_{l=1}^{L} \sum_{k=1}^{K} \sum_{i=1}^{I} \sum_{j=1}^{J_i} d_i(t) \cdot \eta_{ijkl}(t) \cdot h_{ijk} \tag{8}$$

$$LB_l \leq \sum_{k=1}^{K} N_{kl}(t) \leq UB_l \tag{9}$$

$$N_{kl}(t) = N_{kl}(t-1) + y_{kl}^{+}(t) - y_{kl}^{-}(t) \tag{10}$$

$$N_{kl}(t) \leq M^{\infty} \cdot r_{kl}(t) \tag{11}$$

$$r_{kl}(t) \leq N_{kl}(t) \tag{12}$$

$$r_{k^1 l}(t) + r_{k^2 l}(t) \leq 1, \left(k^1, k^2\right) \in S \tag{13}$$

$$0 \leq \eta_{ijkl}(t) \leq 1 \tag{14}$$

$$0 \leq \bar{\eta}_i(t) \leq 1 \tag{15}$$

$y_{kl}^{+}(t),\ y_{kl}^{-}(t),\ N_{kl}(t)$ are general integers and

$$r_{kl}(t) \text{ is binary} \tag{16}$$

Model Objective Function: The objective function given in Eq. (1) comprises several cost terms. The first term is machine maintenance and overhead costs. The second term is machine procurement cost at the beginning of each period. available from the a new system. In this cost term, $N_{kl}(t)$ stands for the number of type k machines assigned to cell l at the begining of period t, with $N_{kl}(0) = 0, \forall k$. The third term of the objective function represents the inter-cell material handling cost. The forth, fifth, six and the seventh terms stand

for machine operating cost, tool consumption cost, setup cost and machine relocation cost, respectively. The last term is the cost of subcontracting parts.

Model Constraints: The constraint in Eq. (2) ensures that if a part is not subcontracted, each operation of the part is assigned to a machine. Eq. (3) permits the assignment of an operation to a machine if and only if a tool required by the operation is available on that particular machine. Eqs. (4) and (5) allow the processing of operation j of part i in at most one cell in time period t. Eq. (6) guarantees that machine capacities are not exceeded. Eq. (7) implies that the number of type k machines used any period is greater than or equal to that of the previous period. This means that the model is not going to remove extra machines of any type if that type of machines happen to be in excess in a certain time period. The presence of extra machines in the system increases system flexibility and reliability by providing alternative routes during machine breakdown. Eq. (8) enforces workload balance among cells. Eq. (9) specifies the lower and upper bounds of cell sizes. Eq. (10) states that the number of type k machines in the current period in a particular cell is equal to the number of machines in the previous period, adding the number of machines being moved in and subtracting the number of machines being moved out of the cell. Eqs. (11) and (12) set the value of $r_{kl}(t)$ equal to 1 if at least one unit of type k machine is placed in cell l during period t or 0 otherwise. Eq. (13) ensures that machine pairs included in S are not placed in the same cell. Eqs. (14) and (15) limit the values of $\eta_{ijkl}(t)$ and $\bar{\eta}_i(t)$, respectively, within $[0, 1]$. Eq. (16) is the integrality constraint.

3. The Genetic Algorithm Based Heuristic

The genetic algorithm developed in this work is tailored in accordance with the nature of the cell formation problem. This includes the development of solution representation, fitness evaluation, genetic operators and heuristic techniques that are specific to the model presented in the previous section.

3.1. The Chromosomal Encoding of a Solution

The chromosomal encoding of a solution is the first task in applying a genetic algorithm. In this research we developed a chromosomal representation of a solution which can satisfy some of the constraints of the model. Figure 1 illustrates a chromosome structure for a particular problem with two planning period and 11 part types. The variable $\bar{\eta}_i(t)$ takes a value in [0, 1] denoting the proposition of the total demand of part i subcontracted during period t. The variable c_j takes a value in $\{1, 2, ...L\}$ representing the cell in which operation j is performed. The x_j's assume values in [0, 1] and are used to calculate the proportions of the production volume among alternative routings. Operations without alternative routings do not have x's associated with them. In the chromosome structure, parts which do not have demand in a given period are not included in the segment of the chromosome representing the product mix for that particular period. For example parts 2, 5 and 10 shown in figure 1 do not appear in the first half of the chromosome since there is no such demand in period 1. The detailed representation of part 6 is shown in this figure. This part is assumed to have nine operations and the 2^{nd}, 8^{th} and 9^{th} operations have two alternative routings each. The 4^{th} operation has three alternative routings and the remaining operations have only one route each.

Period 1								Period 2							
Part 1	Part 3	Part 4	Part 6	Part 7	Part 8	Part 9	Part 11	Part 2	Part 3	Part 4	Part 5	Part 7	Part 8	Part 9	Part 10

Details for Part 6

$\bar{\eta}_6(1)$	Opr.1	Opr.2		Opr.3	Opr.4			Opr.5	Opr.6	Opr.7	Opr.8		Opr.9	
	c_1	c_2	$x_{2,1}$	c_3	c_4	$x_{4,1}$	$x_{4,2}$	c_5	c_6	c_7	c_8	$x_{8,1}$	c_9	$x_{9,1}$

Figure 1. A chromosome structure for a two-planning period problem

3.2. Decoding a Chromosome

The decision variables $\bar{\eta}_i(t)$ and $\eta_{jikl}(t)$ are determined by decoding a chromosome under consideration. The value of $\bar{\eta}_i(t)$ is directly read from the chromosome. For an operation with n alternative routings along machines $k^1, k^2, \cdots, k^n$, the values of $\eta_{jik^1l}(t), \eta_{jik^2l}(t), \cdots \eta_{jik^nl}(t)$ are determined using the sets of equations given in figure 2. In this set of equations the values

of x_{j1}, x_{j2}, $\cdots, x_{j,n-1}$ are obtained from the chromosome and x_{jn} is set to 1. The subscript l takes the value c_j which is also obtained from the chromosome.

$$
\begin{aligned}
\eta_{jik^1l}(t) &= (1-\bar{\eta}_i(t)) \times x_{j1}, \\
\eta_{jik^2l}(t) &= (1-\bar{\eta}_i(t)) \times (1-x_{j1}) \times x_{j2}, \\
\eta_{jik^3l}(t) &= (1-\bar{\eta}_i(t)) \times (1-x_{j1}) \times (1-x_{j2}) \times x_{j3}, \\
&\vdots \quad\quad \vdots \\
\eta_{jik^nl}(t) &= (1-\bar{\eta}_i(t)) \times (1-x_{j1}) \times (1-x_{j2}) \times \cdots \times (1-x_{j,n-2}) \times x_{j,n-1} \times x_{jn}, \\
\& & \\
\eta_{jikl}(t) &= 0\ ; \quad k \notin \{k^1, k^2, \cdots k^{n-1}\},\ l \neq c_j
\end{aligned}
$$

Figure 2. Equations required for decoding a chromosome

As an example, for an operation with three alternative routings along machines k^1, k^2 and k^3, the values $\eta_{jik^1l}(t)$, $\eta_{jik^2l}(t)$ and $\eta_{jik^3l}(t)$ are determined using the first three equations for $x_{j3} = 1$ while x_{j1} and x_{j2} are obtained from the chromosome. Here, it can be seen that:

$$
\begin{aligned}
\sum_{l=1}^{L}\sum_{k=1}^{K} \eta_{jikl}(t) &= \eta_{jik^1,\ cj}(t) \ + \ \eta_{jik^2,\ cj}(t) \ + \ \eta_{jik^3,\ cj}(t) \\
&= (1-\bar{\eta}_i(t)) \times x_{j1} \ + \ (1-\bar{\eta}_i(t)) \times (1-x_{j1}) \times x_{j2} \\
&+ \ (1-\bar{\eta}_i(t)) \times (1-x_{j1}) \times (1-x_{j2}) \\
&= 1-\bar{\eta}_i(t)
\end{aligned}
$$

This ensures the constraint in Eq. (2). From figure 2, $\eta_{jikl}(t) = 0$ for all $k \notin \{k^1, k^2, k^3\}$. This ensures the constraint in Eq. (3) which restricts the assignment of an operation to those machines that can process the operation. Moreover, $\eta_{jikl}(t) = 0$ for $l \neq c_j$. This is in agreement with the constraints in Eqs. (4) and (5) which limit the assignment of an operation to only one cell. From the above discussion it can be seen that the chromosomal encoding enables a randomly generated solution satisfying the constraints in Eqs. (2)–(5) of the model presented in Section 2.

3.3. The Fitness Function

The purpose of the fitness function is to measure the fitness of candidate solutions in the population with respect to the objective and constraint functions of the model. It is given by

Eq. (17) as the sum of the objective function and the penalty terms of constraint violations. Workload balancing and cell size constraints are enforced by such penalty terms. The factors f_{wb} and f_{cs} are used for scaling the penalty terms corresponding to these constraints. The minimum allowable and the actual work performed in a cell in time period t are $W_{min}(t)$ and $W_l(t)$ and computed by Eqs. (18) and (19), respectively.

$$F = \text{Model Objective Function} \quad + \quad f_{wb} \cdot \sum_{t=1}^{T} \sum_{l=1}^{L} \max\{0, \quad W_{min}(t) - W_l(t)\}$$
$$+ \quad f_{cs} \cdot \sum_{t=1}^{T} \sum_{l=1}^{L} \max\left\{0, \quad \sum_{k=1}^{K} N_{kl}(t) - LB_l \ , \quad \sum_{k=1}^{K} N_{kl}(t) - UB_l\right\} \tag{17}$$

$$W_{min}(t) = \frac{q}{L} \sum_{l=1}^{L} \sum_{k=1}^{K} \sum_{i=1}^{I} \sum_{j=1}^{J_i} d_i(t) \times \eta_{ijkl}(t) \times h_{ijk} \tag{18}$$

$$W_l(t) = \sum_{k=1}^{K} \sum_{i=1}^{I} \sum_{j=1}^{J_i} d_i(t) \times \eta_{ijkl}(t) \times h_{ijk} \tag{19}$$

For solutions of a given generation, the most promising ones are those with the minimum F. However, genetic algorithm works with maximization function. Hence, the raw fitness scores need to be transformed so that the minimum raw fitness will correspond to the maximum transformed fitness. This is achieved using Eq. (20) where F_{max} and F_{min} are the maximum and the minimum values of the raw fitness F in the current population.

$$\widetilde{F} \quad = \quad \begin{cases} 1 & ; \quad if \ F_{max} = F_{min} \\ \frac{F_{max} - F}{F_{max} - F_{min}} & ; \quad if \ \frac{F_{max} - F}{F_{max} - F_{min}} > \beta \\ \beta & ; \quad otherwise. \end{cases} \tag{20}$$

The value of β is to be chosen in [0, 1) and is used to control the discriminating power of the transformed fitness function among the less and the high ranking individuals in the population. For higher value of β, individuals with lower ranking also get higher value of the transformed fitness and higher probability of being selected for the next generation. This makes the transformed fitness function less discriminating and may help to maintain population diversity and intensify the exploration of the solution space. On the other hand, low values of β makes the transformed fitness function more discriminating against lower ranked individuals. Consequently, this may reduce the population diversity and intensify

the exploitation of the solution space around the high ranking solutions only. In solving the problems presented in this research, we have found that the values of β in the range from 0.1 to 0.3 can provide satisfactory results by making a balanced trade-off between exploration and exploitation.

3.4. Genetic Operators

Genetic operators make the population to evolve by creating promising candidate solutions to replace those less promising ones. These operators are generally classified as selection, crossover, and mutation operators.

Selection Operator: The selection operator selects, with replacement, individuals having the potential to replace the current population according to the transformed fitness function $\widetilde{F}$. This selection operator may be implemented by simulating a biased roulette wheel where each individual chromosome in the current population has a roulette wheel slot sized in proportion to its transformed fitness (Goldberg, 1989). Once the mating pool is formed, individuals are randomly paired and crossover operator will be applied to each pair with certain probability.

Crossover Operators: The crossover operators produce children by exchanging information contained in the parents. The genetic algorithm used to solve the proposed model employs the standard single-point crossover operator. This operator randomly generates a single crossover point along the length of the chromosome. The crossover point divides each of the parent chromosomes into two segments. Children are then created by exchanging the right-hand-side segments of the parents. The algorithm also incorporates three problem specific crossover operators: the period-swap, part-swamp and operations-swamp crossover operators. The period-swap crossover operator randomly selects a period in the planning horizon and exchanges the subcontracting and operations assignment information of all the parts for the selected period between the parents. The part-swap (operation-swamp) crossover operator randomly selects a part (an operation) in the length of the chromosome and exchanges all the information about this part (operation) between the parents.

Mutation Operators: Selection and crossover do not introduce new genetic material into the population pool. This task is performed by the mutation operators acting at the gene level to alter information contained in the gene. Mutation operators are critical to the success of genetic algorithms since they diversify search directions and avoid premature convergence to local optima. Hong *et. al.* (1990) suggested that each problem, even each stage of the genetic process in a single problem, may require appropriately defined multiple mutation operators for best results. In this research, we developed six different mutation operators. These are part-level cell mutator, operation-level cell mutator, subcontract mutator, alternative route mutator, subcontract degenerator, and alternative route degenerator.

The part-level cell mutator performs cell mutation. This operator randomly alters the c_j's of all the operations of a part to other identical values in $\{1, 2, \cdots L\}$. It is applied during the first phase of the genetic search where the quest is to find the best configuration with independent cells. Similarly, operation-level cell mutator alters the value of c_j's. However, this operator is applied for each operation independently and may result in different values of c_j's of the operations of a part. Hence, it may result in inter-cell movements. For this reason, this operator is applied in the second phase of the genetic search where the attempt is to optimize the cost of inter-cell movement along other cost terms of the model. Moreover, this operator is applied at a lower mutation rate than part-level cell mutator to avoid unnecessary perturbations. Subcontract mutator randomly steps up or down the variable $\bar{\eta}_i$ with a step-amount. This operator also takes care of the constraint in Eq. (13) of the model. Alternative route mutator randomly steps up or down the the variable x_j's for all the operations having alternative routings with a step-amount while keeping these values in [0, 1]. Since both $\bar{\eta}_i$ and x_j's are kept in [0, 1], the variable η_{jikl} will also be kept in the same interval (see figure 2) to ensure the constraint in Eq. (12) of the model. All the mutation operators discussed above are applied with small probabilities.

The subcontract and alternative route degenerators are non-probabilistic mutation operators. The subcontract degenerator sets the value of $\bar{\eta}_i = 0$ if its current value is less than a degeneration limit, d_1. It also sets $\bar{\eta}_i = 1$ if its current value is greater than $1 - d_1$. The alternative routing degenerator sets a value of $x_j = 0$ or 1 based on the magnitude of flow along the alternative routes of an operation. For an operation j of part i with two alternative routings, the operator sets $x_{j1} = 0$ if its current value time $(1 - \bar{\eta}_i)$ is less than

a degeneration limit, d_2. It also sets $x_{j1} = 1$ if its current value times $(1 - \bar{\eta}_i)$ is greater than $1 - d_2$. The purpose of developing and applying these two operators is to speed up convergence by quickly degenerating insignificantly small values of the continuous decision variables.

3.5. Repair Heuristic

The Repair Heuristic is run for each individual chromosome representing a solution which violates the machine separation constraint. This heuristic is required because the machine separation constraint becomes difficult to satisfy by the penalty method. The Repair Heuristic first arbitrarily forms a certain number of mutually exclusive sets of machines taken from S such that the machines in a given set can be placed in the same cell. Once these sets of machines are identified, a certain number of cells will be associated with each set. A cell will be associated with at most one set of machines while a set can be associated with more than one cell. Finally, for a chromosome under repair, operations that require a machine in a given set are arbitrarily assigned to one of the cells associated with that particular set. This guarantees the fulfillment of machine separation constraint. The heuristic has randomness behavior making it compatible with the genetic search.

3.6. Machine Assignment Heuristic

The continuous variables $\eta_{jikl}(t)$ and $\bar{\eta}_i(t)$ are determined by decoding a chromosome. The corresponding integer variables $N_{kl}(t)$, $y^+_{kl}(t)$ and $y^-_{kl}(t)$ are determined using the Machine Assignment Heuristic. The heuristic was applied for each individual chromosome so that the corresponding value of the fitness function can be evaluated. Once the values of the continuous variables $\eta_{jikl}(t)$ and $\bar{\eta}_i(t)$ are known, the minimum number of each type of machines $M_{kl}(t)$ to satisfy capacity requirement in each cell during each period is determined by the following equation:

$$M_{kl}(t) = \left\lceil \frac{\sum_{i=1}^{I} \sum_{j=1}^{J_i} \left(d_i(t) \cdot \eta_{jikl}(t) \cdot h_{jik} \right)}{C_k} \right\rceil \tag{21}$$

where $\lceil x \rceil$ returns the smallest integer greater or equal to x.

After this step, the heuristic sets the number of machines, $N_{kl}(1)$, in each cell for period 1 equal to $M_{kl}(1)$. The number of machines of each type installed in the various cells for $t > 1$

are determined as follows. Let $M_{kl}(t)$ and $\widetilde{M}_k(t)$ represent the minimum number of type k machines required in cell l and in the system respectively and $\widetilde{N}_k(t)$ represents the actual number of type k machines installed in the system during period t. If $\widetilde{N}_k(t-1) \leq \widetilde{M}_k(t)$, then the heuristic sets $N_{kl}(t) = M_{kl}(t)$. If $\widetilde{N}_k(t-1) > \widetilde{M}_k(t)$, then heuristic assigns those type k machines to the various cells to satisfy the required minimum number of machines in period t and leaves the extra machines in the cells in which they were perviously installed. This process of determining the number of machines of each type installed in each cell for $t > 1$ is illustrated using the pseudocode given in Appendix I with an example. From that example, one can see that the number of machines $N_{5,l}(t)$ is greater than or equal to the required number of machines $M_{5,l}(t)$. Hence, the machine capacity constraint, Eq. (6), is satisfied. At the same time, we have $\widetilde{N}_5(t') \geq \widetilde{N}_5(t)$ for $t' > t$. This guarantees the fulfilment of the constraint in Eq. (7) of the model. Once the decision variables $N_{kl}(t)$ are determined using the machine assignment heuristics, the configuration decision variables $y^+_{kl}(t)$ and $y^-_{kl}(t)$ can be determined using Eqs. (22) and (23) respectively. This last step satisfies the constraint in Eq. (10) of the proposed model.

$$y^+_{kl}(t) = \begin{cases} N_{kl}(t), & \text{if } t = 1, \\ max\{0,\ N_{kl}(t) - N_{kl}(t-1)\}, & \text{if } t > 1. \end{cases} \tag{22}$$

$$y^-_{kl}(t) = \begin{cases} 0, & \text{if } t = 1, \\ max\{0,\ N_{kl}(t-1) - N_{kl}(t)\}, & \text{if } t > 1. \end{cases} \tag{23}$$

3.7. Constraints Handling

Michalewicz (1994) stated that the central problem in the application of genetic algorithms is constraint handling. He proposed different approaches to handle constraints such as rejection of infeasible solutions (death penalty), penalty methods and repair algorithms. In this study, the latter two and other approaches were used to handle constraint violations. Constraints in Eqs. (8) and (9) are handled using the penalty method. This method adds penalty quantities if corresponding constraints are not satisfied. A problem specific Repair Heuristic is developed to handle the constraints in Eqs. (11)–(13). Constraints in Eqs.

(2)–(5) are handled by the use of the chromosomal structure to generate solutions satisfying these constraints. The Machine Assignment Heuristic handles constraints in Eqs. (6), (7) and (10). Constraints in Eqs. (14) and (15) are handled by genetic operators which assign values in [0, 1] to $\bar{\eta}_i(t)$ and x_j's. The integrality constraint given by Eq. (16) is handled by the Machine Assignment Heuristic and the Repair Heuristic.

3.8. The Two-Phase Genetic Algorithm

Generating independent cells is computationally simpler than generating cells which optimize inter-cell movements and other cost terms of the objective function. With this consideration, the search process was divided into two phases. In the first phase, the algorithm attempts to quickly find the best configuration with independent cells. To perform this task, the algorithm applies genetic operators which do not produce solutions having inter-cell movement. These include all the operators other than the single-point crossover and the operation-level cell mutator. Inter-cell movement can occur during the first phase only as a result of the Repair Heuristic. In the second phase, the algorithm finds the configuration with the best inter-cell movements based on the solution found in the first phase. All the genetic operators other than the part-level cell mutator are applied during the second phase. In addition to running in two phases, the proposed genetic algorithm has a procedure called population rejuvenation. This procedure reduces and focusses the search domain around the best solution found if this value does not improve in a given number of successive generations. After a given number of successive generations without any improvement of the incumbent solution, we consider that the genetic algorithm has detected the promising area. A certain number of individuals of the current population will be replaced by the duplicates of the best individual so far found. Hence, the population diversity will be reduced and centered around the best solution. This process rejuvenates the population which might have been deteriorated by a large number of applications of the mutation operators.

The steps of the genetic algorithm based heuristic will be presented next. Additional symbols will be used in presenting the algorithm. These symbols and notations are explained below.

ρ - Generation counter, $\rho = 1, 2, ..., \rho_{max}$,

ρ_{max} - Maximum number of generations,

$PP(\rho)$ - Parents population of generation ρ,

$CP(\rho)$ - The current children population which makes up $PP(\rho + 1)$,

BI - Best feasible individual so far found,

$Phase$ - The current phase of the search which equals to 1 for the first phase or 2 for the second phase

ρ_{phase} - Generation at which the value of $Phase$ should be set equal to 2 if it were not previously set to this value by other conditions,

φ - Number of successive generations counted without any improvement of the best individual so far found,

φ_{max} - Maximum value of φ at which point population rejuvenation is to be performed,

ϖ - Number of successive population rejuvenations counted without any improvement of the best individual so far found,

ϖ_{max1} - Maximum value of ϖ at which point the second phase is to be entered if $Phase$ was equal to 1,

ϖ_{max2} - Maximum value of ϖ in the second phase at which point the search will be terminated.

The following notations are used in presenting step 3, the Repair Heuristic

S' - Set of machines (unpaired) included in S,

$p,\ q$ - counter variables,

S_{pq} - Set of machines from which q^{th} machine of the p^{th} set is to be arbitrarily, taken,

k_{pq} - q^{th} machine of the p^{th} set arbitrarily taken from S_{pq}

Δ - A function that takes a set of machines $S_{pq} \setminus \{k_{pq}\}$ and machine k_{pq} as the first and second arguments respectively and returns a set of machine $S_{p,\ q+1}$, taken from its first argument, that can be placed in the same cell with machine type k_{pq},

Υ_p - The p^{th} set of machines taken from S' that can be placed in the same cell.

Using the above notations, the steps of the proposed genetic algorithm based heuristic

are listed below. These steps were coded using C++ programming language and run on a Pentium-4 (2.4 GH, 768 MB Ram) personal computer.

Step 1. Initialize the counters ρ, φ, ϖ and $Phase$ such that $\rho = \varphi = \varpi = 0$ and $Phase = 1$. Initialize the Best-Individual BI so far found with a null value.

Step 2. Randomly generate initial parent-population $PP(0)$ such that each individual represents a solution having independent cells (i.e. the c_j's of all the operations of a given part are set equal),

Step 3. Apply the Repair Heuristic (Steps 3a to 3g) for each individual chromosome violating machine separation constraint.

Step 3a. Set $p = 1$ and $q = 1$ and $S_{1,1} = S'$.

Step 3b. Arbitrarily take machine k_{pq} from S_{pq}.

Step 3c. Determined $S_{p,\ q+1} = \Delta(S_{pq} \setminus \{k_{pq}\},\ k_{pq})$

Step 3d. If $S_{p,\ q+1} \neq \emptyset$, set $q = q + 1$ and go to step 2; otherwise go to step 5.

Step 3e. $\Upsilon_p = \{k_{p,1},\ k_{p,2},\ \ldots k_{pq}\}$ forms the p^{th} set of machines arbitrarily taken from S' that can be placed in the same cell. Set $S_{p+1,1} = S_{p,1} \setminus \Upsilon_p$. If $S_{p+1,1} \neq \emptyset$, set $p = p + 1$ and $q = 1$ and go to step 3b; otherwise go to step 3f.

Step 3f. At this step p equal to the number of sets of machines taken from S which can be placed in the same cell. Randomly and evenly associate the cells with the p sets identified such that a cell is associated with at most one set and each set is associated with minimum of L mod p cells.

Step 3g. For a chromosome under repair, assign operations that require a machine in a given set to one of the cells associated with that particular set.

Step 4. For each chromosome, using the Machine Assignment Heuristic (Step 4a to 4e), determine the number of machines of each type assigned to the various cells during each period.

Step 4a. Decode a chromosome under consideration and determine the values $\eta_{jikl}(t)$.

Step 4b. Calculate the minimum number of each type of machines $M_{kl}(t)$ required in each cell during each planning period using Eq. (21).

Step 4c. Set the number of machines of each type installed in each cell during period-1 equal to the minimum required, i.e. $N_{kl}(t) = M_{kl}(t)$ for $t = 1$.

Step 4d. By implementing the pseudocode given in figure A-I determine $N_{kl}(t)$ for $t > 1$.

Step 4e. Determine the decision variables $y_{kl}^{+}(t)$ and $y_{kl}^{-}(t)$ using the Eqs. (22) and (23) respectively.

Step 5. For each individual, determine $\widetilde{F}$. If an improvement has been obtained update the previous BI by the current BI, set $\varphi = \varpi = 0$; otherwise increase φ by one.

Step 6. If $\varphi = \varphi_{max}$, rejuvenate the population by replacing k% of the population by the duplicates of previous BI and reset $\varphi = 0$ and increase ϖ by one.

Step 7. If $\varpi = \varpi_{max1}$ and $Phase = 1$ set $Phase = 2$ and $\varphi = \varpi = 0$.

Step 8. Select individuals from the current parent-population to become parents of the next generation according to their fitness value.

Step 9. Randomly mate parent-population and create children-population $CP(\rho)$ by applying genetic operators valid to the current phase of the search.

Step 10. Replace the current parent-population $PP(\rho)$ by the children-population $CP(\rho)$ and form the new parent-population $PP(\rho+1)$. Increase the generation counter, $\rho = \rho + 1$. If $\rho = \rho_{phase}$ and $Phase = 1$, then set $Phase = 2$.

Step 11. If the termination criterion has not been met, go to step 3. Termination criterion is $\rho = \rho_{max}$, or $\varpi = \varpi_{max2}$ for $phase = 2$.

4. Numerical Examples

A number of numerical examples were developed to evaluate the computational efficiency of the solution procedure developed. These examples were solved using the heuristic developed. The heuristic solutions were compared with those generated by LINGO, a general

purpose optimization package. In this section, we present three numerical examples in detail. Example 1 demonstrates the computational efficiency of the proposed genetic algorithm and illustrates the effects of some of its features on its performance. A reasonably large problem is considered and its global optimum solution is not reached using LINGO even after long computational time. Hence, the quality of the solution generated by the proposed method is not compared with the global optimum. In Examples 2 and 3, we further evaluate the computational performance of the proposed method using two problem instances whose global optimum solutions can be found. In Example 2, an instance of a small problem solved to optimality using LINGO is considered. In Example 3, we present an instance of a large problem whose data has been systematically generated so that the problem can be decomposed into several subproblems. Each subproblem can be solved to optimality using LINGO and the aggregate solution represents the global optimum of the large problem.

4.1. Example 1

The problem presented in this example consists of 10 different types of machines and 25 part types. The machines are to be grouped into three relatively independent cells. For this numerical example, the model presented in Section 2 has a total of 6,438 variables having potentially non-zero values. 2,100 of these variables were integers. The corresponding number of functional constraints is 6,723. In this and in the following numerical examples, the variables which can be fixed to zero were not counted as we have eliminated them from the LINGO formulations using spares set definition. The data and the solution details of this and the next two examples can be obtained from the authors upon request.

4.1.1. Computational Performance

The performance of the heuristic developed was evaluated against the results from LINGO. To find solutions using LINGO, the absolute value term of the objective function in the integer programming model was linearized. An absolute value term of the form $|x - y|$ can be replaced by a non-linear term $p + n$ with the added constraints $x = y + p - n$, $p \leq M^{\infty} \cdot b$ and $n \leq M^{\infty} \cdot (1 - b)$, where p and n are non-negative real variables, b is a binary variable and M^{∞} is a large positive number. As can be seen from table 1, the lower bound and the best objective function value found by LINGO after 5 hours of computation were 1.69346

Table 1. Comparison of the proposed GA with LINGO

Time	Lower Bound	Objective value using	
hr:min:sec	determined by LINGO	LINGO	the proposed GA
00:00:15	-	×	1.933360
00:00:30	1.69274	×	1.785400
00:01:00	1.69274	×	1.781189
00:02:00	1.69274	×	1.779745
00:06:00	1.69274	1.86962	1.776739
00:08:00	1.69275	1.86962	1.762127
00:10:00	1.69277	1.86962	1.761973
00:12:00	1.69282	1.81068	1.761863
00:16:00	1.69282	1.81068	1.761793
00:20:00	1.69309	1.81068	1.761793
00:30:00	1.69311	1.77401	1.761793
01:00:00	1.69323	1.76934	∗
03:00:00	1.69344	1.76934	∗
05:00:00	1.69346	1.76865	∗
10:00:00	1.69353	1.76865	∗
15:00:00	1.69366	1.76865	∗
20:30:00	1.69377	1.76865	∗

× - Feasible solution was not found
∗ - A termination criterion was met.
Note: Values are in Millions.

and 1.76865 respectively. From this table and figure 3, it can be seen that the heuristic developed was able to determine a solution having an objective function value equal to 96% of the lower bound in less than 20 minutes. The figure also shows that the average of the objective function values of the feasible solutions varies from generation to generation and is far from the best objective function value found. This indicates that the genetic algorithm has maintained diversity of the population in performing a global search. In addition to maintaining the population diversity, the genetic algorithm has converged to an acceptable solution very quickly. This was achieved by using those problem specific procedures as systematically dividing search phases, population rejuvenation, constraint handling and other related genetic operators. As shown in figures 4 and 5, dividing the search into phases, populations rejuvenating and using the degenerator operators enable the genetic algorithm to find near optimal solutions very quickly. The values of the various parameters of the GA used for this computation are given in table 6 under Test 1. The experiment was repeated under other 12 parameter settings given in table 6 under the Tests 2 though 13. The convergence graphs of the GA for these tests are given figure 6. From this figure it can be seen that the GA was able to converge to an acceptable solution for several parameter settings. We were also able to find several other parameter settings for which the genetic algorithm performs

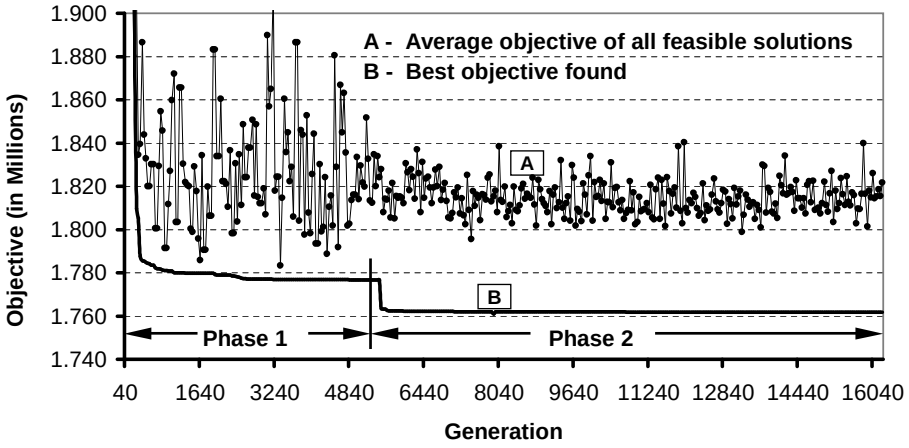

Figure 3. A 15-minutes convergence history of the proposed GA.

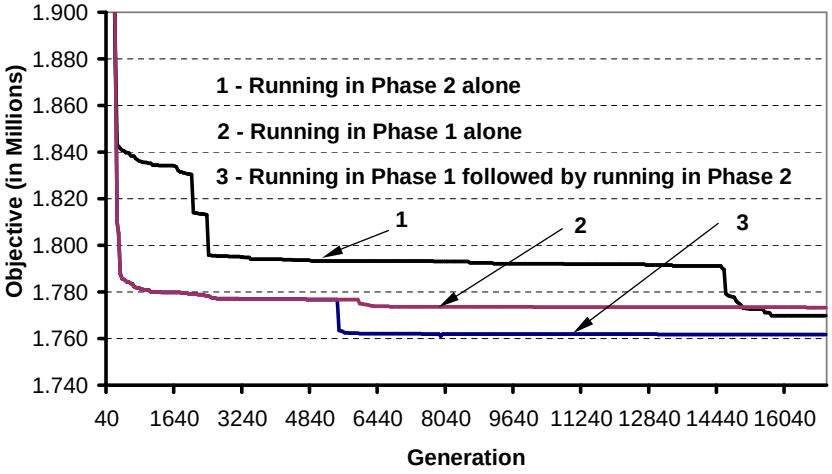

Figure 4. The effect of dividing the search into phases

very well. This can be an indication for the relative robustness of the proposed method.

4.2. Example 2

In this example we present a smaller problem instance consisting of generating 3 manufacturing cells out of 5 different types of machines for processing 12 part types in 2 planning periods. For this problem, the mathematical model has a total of 1,473 potentially non-zero variables; 486 of these variables are integers. The number of functional constraints is 1,494. The problem was solved to optimality using LINGO with 9 hours and 20 minutes of computation on a Pentium-4 personal computer. The genetic algorithm developed was able to find solutions that are 99.5% of the global optimum or better in less than five minutes

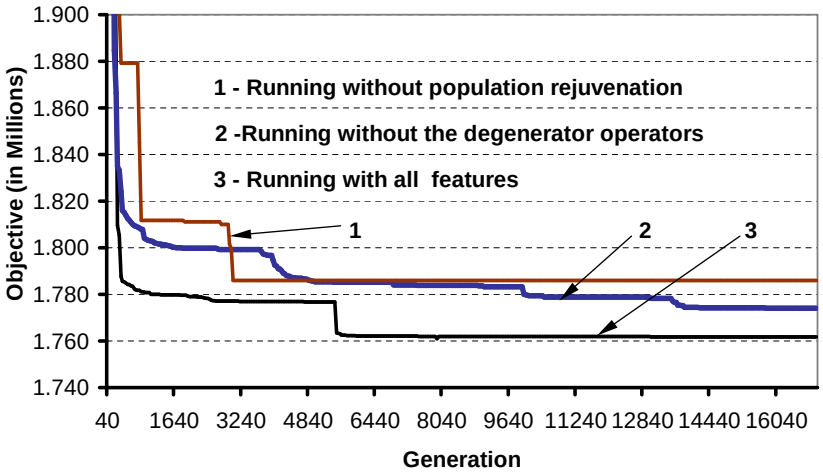

Figure 5. The effects of population rejuvenation and degenerator operators

for a number of combinations of parameter settings. At a particular parameter setting, a solution close to 99.94% of the global optimum was determined in less than two minutes. The optimal and sub-optimal cell formations found using LINGO and the genetic algorithm respectively are given in table 2. The corresponding objective function values along the time taken to reach these values are also shown in this table. As can be seen from this table, the cell compositions of the two solutions are similar. The assignment of the parts to the various routes and alternative routings of the two solutions are also found to be very close to each other. Several other small size problems were considered and solved very close to global optimum using the proposed method within very short times of computation. From this and other small examples, it can be seen that the proposed genetic algorithm has the ability to find solutions very close to the global optimum within very short computational times for small size problems.

4.3. Example 3

This section demonstrates the computational performance of the proposed method using a large problem instance whose global optimum solution has been systematically determined. The problem consists of 3 manufacturing cells, 30 different machine types, 90 part types and 5 planning periods. For this problem, the mathematical model has a total of 33,403 potentially non-zero variables; 11,100 of these variables are integers. The number of functional constraints is 31,484. Our computational experience showed that for problems of such size,

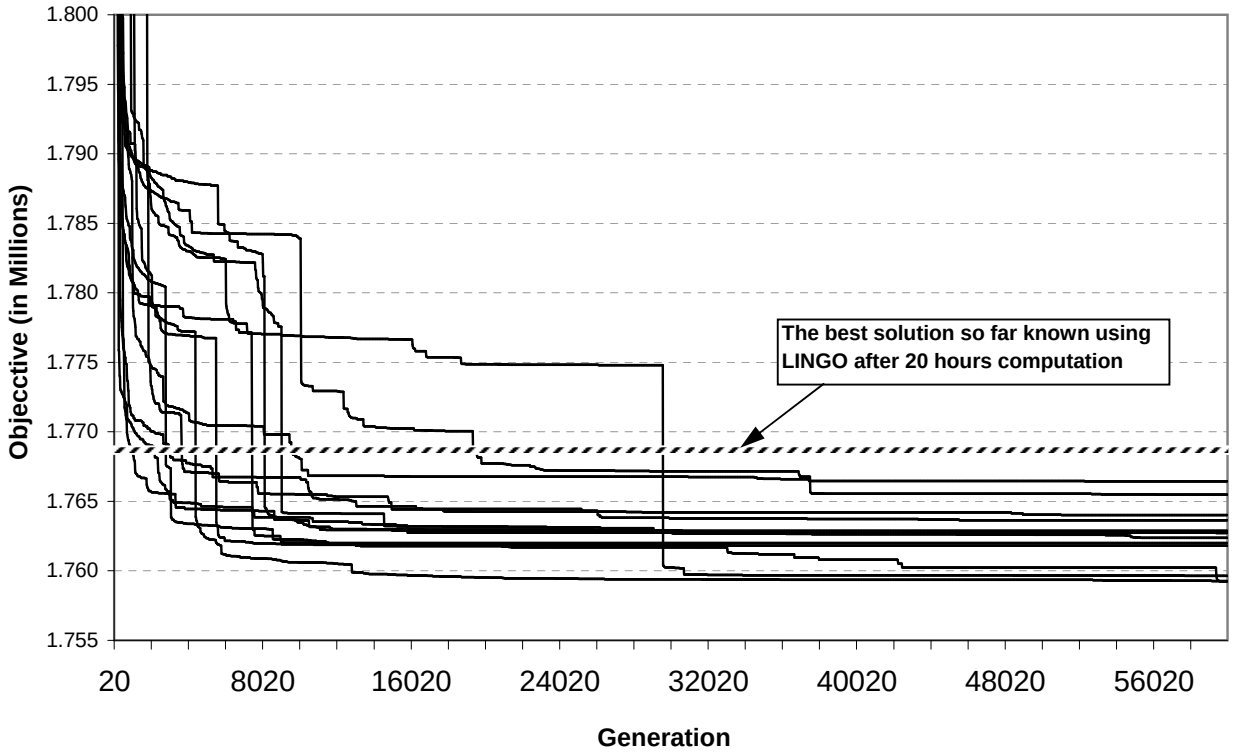

Figure 6. Less than an our convergence history of the proposed GA using 13 different parameter settings given in table 6

the mixed integer programming model developed for the dynamic cell formation problem cannot be optimally solved using LINGO on a personal computer even after several days of computation. For this reason, the data of this large problem were systematically generated so that the problem can be decomposed into subproblems corresponding to each planning period. Each subproblem represents a single period cell formation problem of the period to which can be solved to optimality using LINGO. The aggregate solution represents the global optimum of the large problem. The requirements on the data set to enable such decomposition are:

- Parts that have demand in a given time period do not have demand in the other periods,
- Machines required to process parts in a given time period are not required for processing parts in the other periods,
- There is no separation requirement between machines used in different periods and

Table 2. Cell formation for the small problem using LINGO and the genetic algorithm

	LINGO		GA	
	Machine compositions		Machine compositions	
Descriptions	period-1	period-2	period-1	period-2
Cell-1	1(1), 3(1)	1(1), 3(1), 5(1)	1(1), 2(1), 3(1)	1(1), 2(1), 5(1)
Cell-2	3(1), 4(1), 5(1)	3(1), 4(1), 5(1)	3(1), 4(1), 5(1)	3(1), 4(1), 5(1)
Cell-3	2(2), 3(1)	2(2), 3(1)	2(1), 3(1)	2(2), 3(1)
Objective value	371659.00		372017.00	
Solution status	Optimal		Suboptimal	
Computational time	09:20:00		00:04:00	

Note: The numbers in closed bracket indicate the number of machines of each type.

- The lower and the upper bounds of the cell sizes are low and high enough, respectively, so that the number of machines installed in a given period will not affect the placement of machines in the subsequent periods.

The last requirement is equivalent to relaxing the cell size constraint. In some mathematical models, relaxing the cell size constraint leads to the generation of a single cell containing all the machines. However, in the mathematical model presented in Section 2, the workload balancing constraint tends to prevent the generation of either very dense or very spares cells. Hence, the decomposed subproblems can represent a cell formation problem where a specified number of cells are to be generated. The solutions of each subproblem and the aggregate solution will consist balanced machine cells of comparable sizes.

For this large problem, part types 1-18 have demand only in period 1. These parts require tools that are available in machine types 1 to 6. These machines are not required by parts having demand in the other periods. Hence, the first subproblem consists of part types 1-18 and machine types 1 to 6. The other subproblems can be identified in similar ways and are indicated in table 3. These subproblems are optimally solved using LINGO and the corresponding machine holding cost, objective function values and the elapsed computational times are also given in table 3. The value of objective function in the aggregate solution is determined by summing up the objective function values of each subproblem and the machine holding costs of each subproblem for the subsequent periods as follows:

$$\text{Agregated objective function value} = \sum_{t=1}^{5} O_t + \sum_{t=1}^{5} H_t \cdot (5-t)$$

Table 3. Subproblems and globally optimum solutions information

Sub-Problem	Parts	Machines	No of Variables		Machine Holding Cost	Global Optimum	Time to Global
			Real	Integer			
1	1 to 18	1 to 6	960	474	17800.00	742641.00	10:42:00
2	19 to 36	7 to 12	926	450	10000.00	652886.00	08:54:00
3	37 to 54	13 to 18	857	426	14150.00	698569.00	11:26:00
4	55 to 72	19 to 24	881	432	14100.00	728689.00	14:27:00
5	72 to 90	25 to 30	887	438	13450.00	747132.00	12:31:00
Total	NA	NA	NA	NA	143600.00[†]	3713517.00[‡]	NA

[†]**143600.00** = $17800.00 \times 4 + 10000.00 \times 3 + 14500.00 \times 2 + 14100.00$

[‡]**3713517.00** = 742641.00 + 652886.00 + 698569.00 + 728689.00 + 747132.00 + 143600.00

where O_t and H_t are the objective function value and the machine holding cost of subproblem t, respectively. The addition of the machine holding cost is valid since the model does not remove the machine procured in a given period, even if it happens to become an excess machine in the following time periods. This calculation procedure is indicated in table 3 and the calculated value is equal to 3713517.00. This value represents the objective function value of the global optimum solution of this problem. Solutions closer to the global optimum were generated in less than 20 minutes for various combinations of parameter settings using the proposed heuristic method without decomposing the problem. Some typical values of the objective function of the heuristic solutions are 3722220.00, 3734490.00, 3737080.00, 3737570.00, 3740310.00 and 3751450.00. These values are as close as 99.77, 99.44, 99.37, 99.35, 99.28 and 98.98%, respectively, to the global optimum.

It is obvious that the systematically generated data of this numerical example may not represent an actual dynamic cellular environment. However, the size of the problem and the computational difficulty of the proposed mathematical model do represent those realistic problems in practice. The computational results of the example problem clearly demonstrate the effectiveness and efficiency of the proposed method.

5. Discussions and Conclusions

In this research, we proposed a comprehensive mathematical programming model and an efficient genetic algorithm based heuristic method to generate manufacturing cells over multiple time periods. The model attempts to minimize machine investment cost, inter-cell material handling cost, operating cost, subcontracting cost, tool consumption cost, setup cost

and system reconfiguration cost in an integrated manner. The model also addresses many pragmatic issues such as alternative routings, sequence of operations, identical machines, workload balancing and machine separation requirement. For small size problems, off-shelf optimization software may be used to solve this formulation. For solving large size problems, branch and bound based general search algorithm employed by such software cannot give optimal or near optimal solutions within acceptable computational times on widely available platforms such as a personal computer. To this end, we propose an efficient heuristic based on genetic algorithm. The algorithm incorporates several problem specific genetic operators and constraint handling techniques. The proposed heuristic was evaluated by comparing its computational results and those available from optimization software. The results obtained by using the heuristic method were very encouraging. Our future work includes extending the model and solution method for the selection of material handling equipments, equipment layout and formation of maintenance and inspection policies. We also plan to develop procedures for production planning and control in cellular manufacturing systems. Since the solution method is based on a genetic search procedure, the impact of the parameter settings on the performance of the algorithm will be further investigated.

Acknowledgement: This research is supported by Discovery Grant from NSERC of Canada and by Faculty Research Support Fund from the Faculty of Engineering and Computer Science, Concordia University, Montreal, Quebec, Canada. The authors sincerely thank the two anonymous referees for their thorough reviews and valuable comments on the early version of this paper.

References

Caux, C., Bruniaux, R., and Pierreval, H. 2000, Cell formation with alternative process plans and machine capacity constraints: A new combined approach. *International Journal of Economics*, **64**, 179–284.

Chen, M., 1998, A mathematical programming model for system reconfiguration in a dynamic cellular manufacturing environment. *Annals of Operations Research*, **74**, 109–128.

Dimopoulos, C., and Mort, N. 2001, A hierarchical clustering methodology based on genetic programming for the solution of simple cell-formation problems. *International Journal of Production Research*, **39**, 1–19.

Goldberg, D.E., 1989, *Genetic Algorithms in Search, Optimization, and Machine Learning*. (Reading: Addison-Wesley).

Gonçalves, J.F., and Resendeb, M.J.C. 2004, An evolutionary algorithm for manufacturing cell formation. *Computers & Industrial Engineering*, **47**, 247–273.

Holland, J. H., 1975, *Adaptation in natural and artificial systems*. (The University of Michigan Press).

Hong Z., Wang H., and Chen W. 2000, Simultaneously applying multiple mutation operators in genetic algorithms. *Journal of Heuristics*, **6**, 439-455.

King, J.R., 1980, Machine-component grouping in PFA: an approach using a rank-order clustering algorithm. *International Journal of Production Research*, **18**, 213-232.

King, J.R. and Nakornchai, V., 1982, Machine-component group formation in group technology: review and extensions. *International Journal of Production Research*, **20**, 117-133.

Kusiak, A., 1987, The generalized Group Technology concept. *International Journal of Production Research*, **25**, 561–569.

McAuley, J., 1972, Machine grouping for efficient production. *Production Economics*, **51**, 53–57.

Michalewicz, Z., 1994, *Genetic Algorithms + Data Structures = Evolution Programs*. (New York: Springer).

Mungwattana, A., 2000, Design of cellular manufacturing systems for dynamic and uncertain production requirements with presence of routing flexibility. Ph.D. Thesis, Virginia Polytechnic Institute and State University, Blackburg, VA.

Rogers, D.F., and Kulkarni, S.S., 2005, Optimal bivariate clustering and a genetic algorithm with an application in cellular manufacturing. *European Journal of Operational Research*, **160**, 423–444.

Selim, H.M., Askin, R.G. and Vakharia, A.J., 1998, Cell formation in groupt technology: Review, evaluation and direction for future research. *Computers & Industrial Engineering*, **34**, 2–30.

Singh, N., 1993, Design of cellular manufacturing systems: An invited review. *European Journal of Operational Research*, **69**, 284–291.

Singh, N., 1996, *Systems Approach to Computer-Integrated Design and Manufacturing.* (New York: Wiley).

Solimanpury, M., Vratz, P. and Shankar, R., 2004, A multi-objective genetic algorithm approach to the design of cellular manufacturing systems. *International Journal of Production Research*, **42**, 1419–1441.

Sule, D.R., 1994, *Manufacturing Facilities: Location, Planning, and Design.* (Boston, MA: PWS Publ. Co.).

Suresh, N. C., Slomp, J. and Kaparthi, S.,1999, Seauence-dependent clusturing of parts and machines: a Fuzzy ART neural network approach. *International Journal of Production Research*, **37**, 2793–2816.

Vakharia A. J. and Slim H.M.,1994, Group Technology. In R.C. Dorf and A. Kusiak (eds), *The Handbook of Design, Manufacturing, and Automation* (Wiley, New York), pp. 435–460.

Wemmerlöv, U. and Hyer, N. L. 1986, Procedures for part-family/machine group identification problem in cellular manufacturing. *Journal of Operations Management*, **6**, 125–145.

Wemmerlöv, U. and Johnson, D.J. 1997, Cellular manufacturing at 46 user plants: implementation experiences and performance improvements. *International Journal of Production Research*, **35**, 29–49.

Wicks, E. and Reasor, R. 1999, Designing cellular manufacturing systems with dynamic part populations. *IIE Transactions*, **31**, 11–20.

A. Appendix

A-I. Psudocode and Illustrative Example for the Machine Assignment Heuristic

```
Declare new integer variable called Buffer and Add
FOR[1]  k = 1 to K
    FOR[2]  t = 1 to T − 1
        Ñ_k(t) = 0
        M̃_k(t + 1) = 0
        FOR[3]  l = 1 to L
            Ñ_k(t) = Ñ_k(t) + N_kl(t)
            M̃_k(t + 1) = M̃_k(t + 1) + M_kl(t + 1)
        END FOR[3]
        IF[1]  M̃_k(t + 1) < Ñ_k(t)
            Buffer = Ñ_k(t) − M̃_k(t + 1)
            FOR[4]  l = 1 to L
                IF[2]  M_kl(t + 1) < N_kl(t)
                    Add = min{Buffer,  N_kl(t) − M_kl(t + 1)}
                    N_kl(t + 1) = M_kl(t + 1) + Add
                    Buffer = Buffer − Add
                END IF[2]
                ELSE[2]
                    N_kl(t + 1) = M_kl(t + 1)
                END ELSE[2]
            END FOR[4]
        END IF[1]
        ELSE[1]
            FOR[5]  l = 1 to L
                N_kl(t + 1) = M_kl(t + 1)
            END FOR[5]  l = 1 to L
        END ELSE[1]
    END FOR[2]
END FOR[1]
```

To further illustrate, we present a simple example to place type 5 machines in a 4-cell system for 4 time periods in Appendix A-I. To illustrate the machine assignment heuristic presented in Section 3.6, here we present a simple example to place machines of type 5 in a 4-cell system for 4 time periods. Assume that a chromosome is decoded and the minimum number of machines of type 5 required in each cell for each period are determined using Eq. 21 as shown in table 4. For period 1, we have $N_{5,l}(1) = M_{5,l}(1)$. Hence, $N_{5,1}(1) = 2$, $N_{5,2}(1) = 1$, $N_{5,3}(1) = 3$, $N_{5,1}(1) = 2$ and $N_{5,4}(1) = 0$. The following step by step application of the pseudocode shown in figure A-I gives $N_{5,l}(t)$ for $t > 1$.

Table 4. Minimum number of type 5 machine to meet capacity requirement

l	$M_{5,l}(1)$	$M_{5,l}(2)$	$M_{5,l}(3)$	$M_{5,l}(4)$
1	2	0	1	1
2	1	2	4	3
3	3	1	2	0
4	0	1	2	3
$\widetilde{M}_5(t)$	7	4	9	7

FOR^1 : $k = 5$

FOR^2 : $t = 1$

Initialize $\widetilde{N}_5(1) = 0$ and $\widetilde{M}_5(2) = 0$

FOR^3 : $l = 1$ to 4 (performing successive addition)

$\widetilde{N}_5(1) = 7$ and $\widetilde{M}_5(2) = 4$

IF^1 : The logical test $\widetilde{M}_5(2) = 4 < \widetilde{N}_5(1) = 7$ is TRUE

$Buffer = \widetilde{N}_5(1) - \widetilde{M}_5(2) = 7 - 4 = 3$

FOR^4 : $l = 1$

IF^2 : The logical test $M_{5,1}(2) = 0 < N_{5,1}(1) = 2$ is TRUE

$Add = min\{Buffer, N_{5,1}(1) - M_{5,1}(2))\} = min\{3, 2 - 0\} = 2.$

$N_{5,1}(2) = M_{5,1}(2) + Add = 0 + 2 = 2$

$Buffer = Buffer - Add = 3 - 2 = 1$

FOR^4 : $l = 2$

IF^2 : The logical test $M_{5,2}(2) = 2 < N_{5,2}(1) = 1$ is FALSE

ELSE^2 : $N_{5,2}(2) = M_{5,2}(2) = 2$

FOR^4 : $l = 3$

IF^2 : The logical test $M_{5,2}(2) = 1 < N_{5,1}(1) = 3$ is TRUE

$Add = min\{Buffer, N_{5,2}(1) - M_{5,2}(2))\} = min\{1, 3 - 1\} = 1.$

$N_{5,3}(2) = M_{5,3}(2) + Add = 1 + 1 = 2$

$Buffer = Buffer - Add = 1 - 1 = 0$

FOR^4 : $l = 4$

IF^2 : The logical test $M_{5,4}(2) = 1 < N_{5,4}(1) = 0$ is FALSE

ELSE^2 : $N_{5,4}(2) = M_{5,4}(2) = 1$

From the above steps

$N_{5,1}(2) = 2$, $N_{5,2}(2) = 2$, $N_{5,3}(2) = 2$ and $N_{5,4}(2) = 1$

FOR^2 : $t = 2$

Initialize $\widetilde{N}_5(1) = 0$ and $\widetilde{M}_5(2) = 0$

FOR^3 : $l = 1$ to 4 (performing successive addition)

$\widetilde{N}_5(2) = 7$ and $\widetilde{M}_5(3) = 9$

IF^1 : The logical test $\widetilde{M}_5(3) = 9 < \widetilde{N}_5(2) = 7$ is FALSE

ELSE^1 :

FOR^5 : $l = 1$ to 4 (performing successive assignment operation)

$N_{5,1}(3) = M_{5,1}(1) = 1$, $N_{5,2}(3) = M_{5,1}(2) = 4$

$N_{5,3}(3) = M_{5,1}(3) = 2$, $N_{5,4}(3) = M_{5,1}(4) = 2$

Continuing the above computation for $t = 3$, we can determine $N_{5,l}(t)$ for $t = 4$. The results of the above computations are summarized in table 5.

Table 5. Actual number of type 5 machines installed

l	$N_{5,l}(1)$	$N_{5,l}(2)$	$N_{5,l}(3)$	$N_{5,l}(4)$
1	2	2	1	1
2	1	2	4	4
3	3	2	2	1
4	0	1	2	3
$\widetilde{N}_5(t)$	7	7	9	9

A-II. Parameter Settings for the 13 Test Runs of Example 1

Table 6. Values of the parameters of GA used for 13 test runs of Example 1

	Test Run												
Parameter Name	1	2	3	4	5	6	7	8	9	10	11	12	13
Population Size	360	360	540	390	420	426	570	285	430	340	380	556	340
Crossover probability for													
Period-swamp crossover	0.6	0.6	0.9	0.85	0.55	0.45	0.9	0.74	0.76	0.65	0.77	0.72	0.65
Part-swamp crossover	0.6	0.6	0.8	0.75	0.85	0.82	0.9	0.82	0.9	0.55	0.66	0.76	0.75
Single point crossover	0.4	0.4	0.5	0.25	0.35	0.4	0.35	0.52	0.56	0.4	0.35	0.35	0.4
Operation-swamp crossover	0.1	0.1	0.3	0.15	0.2	0.3	0.4	0.32	0.26	0.1	0.18	0.18	0.1
Mutation Probability for													
Alternative route mutator	0.05	0.04	0.03	0.02	0.003	0.006	0.015	0.012	0.016	0.035	0.035	0.035	0.035
Subcontract mutator	0.004	0.006	0.02	0.012	0.03	0.02	0.015	0.014	0.018	0.01	0.01	0.02	0.01
Part-level cell mutator	0.004	0.006	0.008	0.01	0.04	0.007	0.012	0.015	0.007	0.006	0.006	0.005	0.005
Operation-level cell mutator	0.0004	0.0006	0.002	0.001	0.002	0.0006	0.003	0.003	0.0005	0.0006	0.0006	0.0005	0.0006
Degeneration limit													
d_1	0.06	0.06	0.06	0.04	0.05	0.05	0.05	0.07	0.07	0.05	0.05	0.05	0.05
d_2	0.06	0.06	0.06	0.04	0.04	0.04	0.04	0.07	0.07	0.05	0.05	0.05	0.05
Step amount for													
Subcontract mutator	0.15	0.2	0.05	0.07	0.07	0.06	0.05	0.065	0.072	0.2	0.06	0.06	0.15
Alternative route mutator	0.1	0.06	0.05	0.1	0.1	0.08	0.05	0.065	0.082	0.06	0.06	0.06	0.06
φ_{max}	15	15	10	10	8	12	12	8	10	12	14	12	12
ϖ_{max1}	100	100	120	120	200	220	220	195	212	100	100	100	100
	M	M	M	M	M	M	M	M					
Penalty cost factor for													
Workload balancing , f_{wb}	0.001	0.001	0.0005	0.006	0.002	0.005	0.005	0.015	0.008	0.001	0.001	0.001	0.001
Cell size , f_{cs}	1000	1000	800	850	900	1000	1000	1150	1000	1000	1020	1020	1000