Manufacturing Cell Design: An Integer Programming Model Employing Genetic Algorithms

Jeffrey A. Joines
Graduate Research Assistant

C. Thomas Culbreth
Associate Professor

Russell E. King
Associate Professor

February 14, 1996

Department of Industrial Engineering
North Carolina State University
Box 7906 Raleigh, NC 27695-7906

Revised November 1994

This research was sponsored in part by the NCSU Furniture Manufacturing and Management
Center and the National Science Foundation (Grant # DDM-9215432).

Abstract

The design of a cellular manufacturing system requires that a part population, at least minimally described by its use of process technology (part/machine incidence matrix), be partitioned into part families and that the associated plant equipment be partitioned into machine cells. At the highest level, the objective is to form a set of completely autonomous units such that inter-cell movement of parts is minimized. We present an integer program that is solved using a genetic algorithm (GA) to assist in the design of cellular manufacturing systems. The formulation uses a unique representation scheme for individuals (part/machine partitions) that reduces the size of the cell formation problem and increases the scale of problems that can be solved. This approach offers improved design flexibility by allowing a variety of evaluation functions to be employed and by incorporating design constraints during cell formation. The effectiveness of the GA approach is demonstrated on several problems from the literature.

Keywords: Genetic Algorithms, Integer Programming, Cellular Manufacturing

1 Introduction

Flexibility and efficiency in the manufacturing system are crucial for companies that produce a large number of products in small-to-medium lot sizes. One approach used to gain these benefits has been to develop a cellular manufacturing system using the principles of group technology (GT) [24]. The manufacturing system is decomposed into several manageable subsystems, or groups, by aggregating similar parts into part families and dissimilar machines into cells [37]. The ideal cell is "independent," i.e., it completely manufactures its part family(s), has balanced setups, and requires minimal part backtracking. Automation and/or control are simplified through the creation of independent cells. Setup times, work-in-progress inventory, throughput time, and material handling can be reduced, and opportunities for improved quality can be gained [10, 15, 24].

In this paper, we first discuss the various approaches to cell design found in the literature. Then, an integer programming formulation of the cell formation problem is presented that uses a unique representation scheme for individuals (parts/machines). This formulation reduces the size of the cell formation problem. A genetic algorithm (GA) is then used to solve the integer program. This cell design methodology offers several advantages over existing methods. It simultaneously groups parts and machines into families and cells and eliminates the need for visual analysis of the solution. Also, it offers improved design flexibility by allowing the cell designer to use various objective functions and incorporate design constraints during cell formation. These capabilities allow alternative cell configurations to be generated and reviewed easily.

The following section reviews existing cell design methods. Section 3 provides an introduction to genetic algorithm fundamentals, while Section 4 explains the GA approach used to find solutions to the cell formation problem. In Section 5, experimentation on several problems from the literature is reported, using grouping efficacy as the evaluation function.

Section 6 discusses the effects of constraining the number of permissible cells. We show that the approach can determine the naturally occurring clusters as well as find solutions when the number of cells is constrained. Section 7 summarizes the significance of the work presented.

2 Previous Research

Methodologies for cell design can be categorized as (1) those that group parts into families based upon similar design features and (2) those that manipulate process sequences or routings to form the corresponding machine cells. Extensive literature (see Joines, King, and Culbreth [27, 28] and Selim, Askin, and Vakharia [53]) has been devoted to various techniques that partition a population of parts, as described by the use of process technology (part/machine incidence matrix), into part families and associate them with machine cells. Cell formation techniques that operate on the part/machine incidence matrix are generally referred to as the second phase of Production Flow Analysis (PFA) [8] and involve combinatorial optimization. Cell formation methods based on process routings can be further categorized according to the type of algorithm employed to cluster the data, e.g., array-based clustering, hierarchical and non-hierarchical cluster analysis, mathematical programming methods, graph theoretic approaches, artificial intelligence techniques, and other heuristics.

(i) Array-based methods: The array-based clustering methods perform a series of column and row manipulations to form part families and machine cells simultaneously. The bond energy algorithm [42] and the rank order clustering algorithm [31] are two of the oldest and most commonly applied array-based methods. Subsequent modifications and improvements of these algorithms have been described by Chan and Milner [10], Chandrasekharan and Rajagopalan [11], Khator and Irani [30], King and Nakornchai [32], and Kusiak and Chow [38]. In Chu's [17] comprehensive comparison of three array-based clustering techniques, the bond energy algorithm outperformed the other array-based clustering techniques. In general, array-based techniques operate only on the part/machine incidence matrix and incorporate no other manufacturing information, such as limits on the size of each cell, machining times, etc. These methods require visual inspection of the solution to determine part families and machine cells and are affected by exceptional elements and bottleneck machines.

(ii) Hierarchical clustering methods: Hierarchical methods can be divided into two subcategories: divisive and agglomerative. Divisive algorithms start with all the machines (or parts) in a single group and then iteratively generate partitions until each machine is in a singleton cluster [56]. Agglomerative algorithms start with each machine as an initial cluster and then proceed to merge groups into successively larger partitions until a single cluster contains the whole set. The agglomerative approach has been applied by Carrie [9], McAuley [41], Mosier [47], and Seifoddini [51]. The similarity index used as the basis in the formation of groups by hierarchical clustering methods offers a distinct advantage over array-based methods. It allows information, such as annual part demands, to be included in the clustering logic in addition to the routing sequences [21]. These methods present the designer with a hierarchy of solutions from which the best alternative is chosen. However,

these methods do not form the cells and families simultaneously, so additional methods must be employed to complete the design of the system. A dendrogram is typically used to represent the solution hierarchy, making the choice of the correct solution difficult for large problems.

(iii) Non-hierarchical clustering methods: These clustering techniques include GRAPHICS (Srinivasan and Narendran [55]) and ZODIAC (Chandrasekharan and Rajagopalan [12]). ZODIAC is a modified version of MacQueen's clustering method [1]. It has been shown that these methods outperform the array-based methods and the hierarchical methods, according to a comprehensive comparison of nine clustering methods by Miltenburg [45]. The effect of bottleneck machines and exceptional elements is minor compared to the array-based clustering methods. However, the number of clusters (cells) must be specified *a priori*, and the input is limited to the part/machine incidence matrix.

(iv) Graph theoretic methods: Rajagopalan and Batra [50] used a graph partitioning algorithm that utilizes cliques to group the machines. Lee and Garcia-Diaz [40] transformed the cell formation problem into a network flow formulation and used a primal-dual algorithm developed by Bertsekas and Tseng [4] to solve for the machine cells. This algorithm is more efficient computationally than the p-median model by Kusiak [34]. Other graph approaches include the minimum spanning tree by Ng [48], a heuristic graph partitioning approach by Askin and Chiu [2], and a network flow approach using a modified version of the Gomory-Hu algorithm by Vohra *et al.* [58].

(v) Mathematical programming methods: Clustering can be treated as an optimization problem and, therefore, mathematical programming methods offer several distinct advantages. These methods have the capability of constraining the size of each cell, the number of cells, as well as the ability to incorporate the use of different process plans for each of the parts. The p-median model by Kusiak [34], one of the more popular models, minimizes the total sum of distances between each pair of parts (machines). Other integer programming models have been developed by Boctor [5], Choobineh [14], Gunasingh and Lashkari [20], and Purcheck [49]. Srinivasan *et al.* [55] developed an assignment model for the cell formation problem, while Steudal and Ballakur [57] applied a dynamic programming model. Most of these methods involve linear or non-linear integer programming models, making them difficult to solve for large numbers of parts and machines due to the computational complexity. Also, most of the models do not simultaneously form the machine cells and part families.

(vi) Artificial intelligence methods: Recent applications of artificial intelligence (AI) techniques have shown promise as tools for the design of cellular manufacturing systems. The unsupervised artificial neural network model, ART [29], offers the advantage of not requiring the storage of the part/machine incidence matrix in memory. The fuzzy c-means clustering algorithm by Chu and Hayya [16] presents the designer with a fuzzy solution. In fuzzy solutions, the parts have a different membership grade in several different cells. In crisp solutions, parts are either present or absent with respect to a cell. The designer can use membership grade information to chose a solution that increases machine utilization. Other AI approaches include expert systems [35], formal language theory [59], and a simulated annealing approach [5].

(vii) Other heuristics: Other heuristics based on branch and bound techniques have

also been developed to aid in the design of cellular manufacturing systems [36]. Minis et al. [46] developed a technique that groups machines into cells and parts into families by minimizing the inter-cell traffic subject to capacity constraints. This method has the capability of including unique, as well as multi-functional, identical machines in the grouping procedure. A simple two-part heuristic algorithm, which minimizes inter-cell movement, was developed by Harhalakis *et al.* [22]. This procedure can accommodate ordered operations and non-consecutive operations on the same machine while minimizing inter-cell movement. Seifoddini [52] developed a probabilistic model to overcome assumptions of deterministic demand for parts. The closest neighbor algorithm [6], the branch and bound algorithm, and the minimum spanning tree algorithm [48] are among the best in the literature for forming manufacturing cells using only the part/machine incidence matrix.

3 Genetic Algorithm Fundamentals

Commonly, the manufacturing cell designer's objective is to form a set of completely autonomous manufacturing units that eliminates any inter-cell movement of parts. Comprehensive comparisons of cell formation techniques show that most work well for small, well-defined problems [6, 17, 37, 45, 55]. Although some methods offer generally superior results, no single technique has been shown to provide the best solution for a broad range of applications. Most techniques do not offer cell designers the flexibility to change objective functions and selectively include constraints. In addition, several meaningful evaluation criteria have been proposed that cannot explicitly be used as an objective function in many clustering algorithms. Finally, most clustering algorithms cannot identify all naturally occurring clusters and find solutions with a constrained number of clusters.

To meet the need for a flexible, efficient, and effective clustering technique, we have developed a cell formation method that uses a genetic algorithm. GAs can find solutions to linear and nonlinear problems by simultaneously exploring multiple regions of the state space and exponentially exploiting promising areas through mutation, crossover, and selection operations [43]. Genetic algorithms have proven to be an effective and flexible optimization tool that can find optimal or near-optimal solutions. Unlike many other optimization techniques, GAs do not make strong assumptions about the form of the objective function [43].

Most optimization methods maintain a single solution and improve it until an optimal solution is found. GAs differ in that they maintain and manipulate a family, or population of solutions, in the search for an optimal solution. GAs mimic the evolutionary process by implementing a "survival of the fittest" strategy. In general, the fittest individuals of any population tend to reproduce and pass their genes to the next generation, thus improving successive generations. However, some of the worst individuals do, by chance, survive and reproduce. A more complete discussion of GAs, including extensions to the general algorithm and related topics, can be found in books by Davis [18], Goldberg [19], Holland [23], and Michalewicz [43].

Each solution or individual in the population is described by a vector of variables (chromosome representation). The first step in the GA procedure is to initialize the population

either randomly or by seeding. Once the initial population is generated, each individual, i, is evaluated using the objective function to determine its fitness or value, F_i . A subset of the population is selected to parent the next generation. An individual in the population can be selected to be a parent more than once. A probabilistic selection is performed such that the fittest individuals have an increased chance of being selected. These parents then undergo reproduction using genetic operators to produce a new population. To complete the new population, a subset (arbitrary or otherwise) of the old population is added to the new population. For example, the elitist model ensures that the best individual of one generation is included in the next. The GA moves from generation to generation until some specified stopping condition is met.

4 Integer Program Formulation and GA Solution Technique

Mathematical programming approaches for the clustering problem are nonlinear or linear integer programming problems [5, 34, 39]. These formulations suffer from three critical limitations. First, because of the resulting nonlinear form of the objective function, most approaches do not concurrently group machines into cells and parts into families [5]. Second, the number of machine cells must be specified *a priori*, which affects the grouping process and potentially obscures natural cell formations in the data. Third, since the variables are constrained to integer values, most of these models are computationally intractable for realistically sized problems [40].

To overcome these limitations, a new solution technique using a genetic algorithm to solve integer programming formulations was developed. The following variable declarations and constraints represent a common integer programming formulation of the cell design problem. The constraints ensure that each machine and part is assigned to only one cell or family, respectively.

$$
\begin{aligned}
x_{il} &= \begin{cases} 1, & \text{if machine } i \text{ is assigned to cell } l \\ 0, & \text{otherwise} \end{cases} \\
y_{jl} &= \begin{cases} 1, & \text{if part } j \text{ is assigned to part family } l \\ 0, & \text{otherwise} \end{cases} \\
\sum_{l=1}^{k} x_{il} &= 1 \quad \forall i = 1, \ldots, m \\
\sum_{l=1}^{k} y_{jl} &= 1 \quad \forall j = 1, \ldots, n \\
k &= \text{the number of cells(families) specified} \\
m &= \text{the number of machines} \\
n &= \text{the number of parts}
\end{aligned}
$$

For the m machine and n part problem with k cells, there are a total of $k(m+n)$ variables and $m+n$ constraints. As the number of parts and machines increases, the models become too large to be stored in memory or become computationally intractable [40]. To overcome these limitations, an integer programming model with the following variable declarations is proposed.

$$
\begin{aligned}
x_i &= l, \quad \text{machine } i \text{ is assigned to cell } l \\
y_j &= l, \quad \text{part } j \text{ is assigned to part family } l
\end{aligned}
$$

This model reduces the number of variables by a factor of k to $(m+n)$ and eliminates the constraints. Each part and machine variable is equal to the number of its assigned family or cell. For example, $y_2 = 1$ indicates that part 2 is assigned to part family 1, while $x_3 = 2$ indicates that machine 3 is assigned to machine cell 2. Part families are assigned to the machine cell with the same number. This unique representation effectively reduces the state space by incorporating the constraints into the variable formulation. A standard integer programming solution technique cannot be employed without significant modification because of the objective function's inability to decode this variable representation. However, for the genetic algorithm, the objective function is a computer procedure that can easily decode and evaluate a solution.

In building a genetic algorithm, six fundamental issues that affect the performance of the GA must be addressed: chromosome representation, initialization of the population, selection strategy, genetic operators, termination criteria, and evaluation measures. In the following subsections, these issues are introduced and described specifically for the proposed genetic algorithm.

4.1 Chromosome Representation

For any GA, a chromosome representation is needed to describe each individual in the population of interest. The representation scheme determines how the problem is structured in the GA, as well as the genetic operators that can be used. Each individual or chromosome is made up of a sequence of genes from a certain alphabet. The alphabet can be a set of binary numbers, real numbers, integers, symbols (i.e., A, B, C, D), or matrices, for example.

In a binary representation, individuals are represented by a collection of attributes or variables that are represented by a single binary string as follows:

$$\text{Individual} \longrightarrow (\underbrace{001\ldots111}_{x_1}\underbrace{000\ldots101}_{x_2}\ldots\underbrace{011\ldots111}_{x_n})$$

where x_i is a real variable in the problem. The use of a float, or real representation, alleviates some of the problems associated with a binary representation and such representation has been shown to be more efficient and to produce better solutions than a binary representation [43]. In a float representation, each individual is represented by a vector of floating point variables as shown below. (This is a more natural representation since no problem transformation is necessary.)

$$\text{Individual} \longrightarrow (x_1, x_2, \ldots, x_n)$$

Binary and float representations are used primarily for numerical optimization and typically do not work well for combinatorial optimization problems such as the traveling salesman problem or the cell formation problem [18, 43]. For these types of problems, three different representations are typically used: adjacency, ordinal, and order-based or path.

Determining an appropriate representation of the variables is a necessary first step in designing the GA. Because there are particular inefficiencies associated with using a float representation in integer programming, we developed a representation in which each gene or variable in an individual or chromosome comes from an alphabet consisting of the integers between the variable's upper and lower bounds.

$$\text{Individual} \longrightarrow (x_1, x_2, \ldots, x_{m+n})$$

Specifically, for the cell formation problem, an integer alphabet $1, ..., k_{max}$ is employed where k_{max} represents an upper bound on the number of part families/machine cells. The effect of selecting various values of k_{max} is discussed in Section 6 (the larger the value of k_{max} chosen, the larger the search space). For an individual, the first m variables represent the machines while the last n variables are associated with the parts. Therefore, each individual is a vector of $m + n$ integer variables.

$$\text{Individual} \longrightarrow (\underbrace{x_1, x_2, \ldots, x_m}_{\text{machines}}, \underbrace{y_1, y_2, \ldots, y_n}_{\text{parts}})$$

4.2 Initialization of the Population

For any GA, it is necessary to initialize the population. The most common method is to randomly generate solutions for the entire population. Since GAs iteratively improve existing solutions, the beginning population can be seeded by the cell designer with individuals from other cell formation algorithms or from an existing solution to the problem. The remainder of the population is then seeded with randomly generated solutions. All experiments discussed in this paper employ a completely random seeding of the beginning population.

4.3 Selection Strategy

The selection of parents to produce successive generations plays an extremely important role in the GA. The goal is to allow the "fittest" individuals to be selected more often to reproduce. However, all individuals in the population have a chance of being selected to reproduce the next generation. Each individual is assigned a probability of being selected, with better individuals having larger probabilities. There are several schemes for determining and assigning the selection probability, e.g., roulette wheel selection and its extensions, scaling techniques, and ranking methods [19, 43]. For other non-probabilistic selection procedures, including tournament selection and elitist models, refer to [19, 43].

Discussion of selection methods is limited to the normalized geometric ranking scheme used for the cell design problem described in this paper. Individuals in the population are ranked from best to worst, according to their fitness value. Then, each individual is assigned a probability of selection based upon some distribution, e.g., triangular or geometric. Michalewicz [43] and Joines and Houck [26] have shown that GAs incorporating ranking methods based upon the geometric distribution outperform those based upon the triangular distribution. Joines and Houck also showed that a pure geometric distribution is not appropriate since its range is defined on the interval one to infinity. Thus, for a finite range (population size), the geometric distribution shown below is a dishonest distribution.

$$\sum_{Population} P[\text{Selecting the } r\text{th individual}] = \sum_{i=1}^{P} q(1-q)^{r-1} < 1$$

where

P – number of individuals in the population
q – probability of selecting the best individual and
r – rank of the individual where 1 is the best.

To alleviate this problem, they developed a normalized geometric distribution:

$$\sum_{Population} P[\text{Selecting the } r\text{th individual}] = \sum_{i=1}^{P} q'(1-q)^{r-1}$$

where

$$q' = \frac{q}{1-(1-q)^P}.$$

4.4 Genetic Operators

Reproduction is carried out by the application of genetic operators on the selected parents. Mutation and simple crossover are the two basic types of genetic operators [44]. Mutation operators alter or mutate one parent by changing one or more variables (gene or alphabet) in some way, or by some random amount, to form one offspring. Crossover operators combine information from two parents such that the two children have a "resemblance" to each parent. The application of these types, and their derivatives, depends on the chromosome representation. Since the proposed method uses modified float operators, an example illustrating both a mutation and crossover operator is given in the context of a float representation. For operators used with other representations, refer to [18, 19, 23, 43].

Float mutation operators require a chromosome representation consisting of genes that are real variables with lower bound $[a_i]$ and upper bound $[b_i]$. As one example of mutation, the uniform float mutation operator randomly selects one of the variables, v_i, from a parent and sets it equal to a random number uniformly distributed between a_i and b_i. This mutation operator makes a uniform random change to one of the variables in the parent.

The simple float crossover operator randomly selects a cut point. The two parents are then split at this point, and children are created by concatenating the segments from both parents. For example, the cut point generated below is 3, creating a split within the parents ($P1$ and $P2$) after the third variable.

$$\begin{aligned} P1 &= (x_1, x_2, x_3 \mid x_4, x_5) \\ P2 &= (y_1, y_2, y_3 \mid y_4, y_5) \end{aligned}$$

The following two children, $C1$ and $C2$, are created from these two parents.

$$\begin{aligned} C1 &= (x_1, x_2, x_3 \mid y_4, y_5) \\ C2 &= (y_1, y_2, y_3 \mid x_4, x_5) \end{aligned}$$

Each offspring contains information from each parent. It should be noted that simple crossover works with any alphabet (binary, integer, or real) for unconstrained numerical optimization problems. Michalewicz [43] and others have developed six genetic operators to work with a float representation: uniform mutation, non-uniform mutation, multi-non-uniform mutation, boundary mutation, simple crossover, and arithmetic crossover.

Since the GA uses a modified float representation, these standard float genetic operators were adapted to produce integer results. A total of seven integer-based genetic operators are utilized in the algorithm. Mutation operators tend to make small random changes in an attempt to explore all regions of the state space, while crossover operators combine information from two parents [43]. In the following two subsections, we describe the adaptation and/or creation of the seven operators required to make them useful for the GA approach to cell formation.

4.4.1 Mutation Operators

The ***uniform mutation*** operator randomly selects one of the variables, v_i, from a parent and sets it equal to a random number uniformly distributed between the variable's lower (a_i) and upper (b_i) bounds. The standard uniform mutation operator was modified so that the randomly generated floating point number is truncated to an integer result.

The ***boundary mutation*** operator randomly selects one of the variables from a parent and randomly sets it equal to its lower (a_i) or upper bound (b_i). Since the bounds are typically specified as integers, the boundary mutation operator requires no modification in order to work with the integer-based GA.

The ***non-uniform mutation*** operator randomly selects one of the variables, v_i,, from a parent and adds to it a random displacement from the distribution described below. This operator requires the generation of two uniform (0,1) random numbers, r_1 and r_2. The first (r_1) is used to determine the direction of the displacement, while r_2 is used to generate the magnitude of the displacement.

$$v_i = \begin{cases} v_i + (b_i - v_i)f(G) & \text{if } r_1 < 0.5, \\ v_i - (v_i - a_i)f(G) & \text{otherwise,} \end{cases}$$

where

$$
\begin{aligned}
f(G) &= r_2(1 - \tfrac{G}{G_{max}})^2, \\
r_2 &= \text{a uniform random number between (0,1)}, \\
G &= \text{the current generation}, \\
G_{max} &= \text{the maximum number of generations.}
\end{aligned}
$$

In early generations, this operator is similar to the uniform mutation operator. However, as the number of generations increases, the distribution narrows, thus increasing the exploitation of the local solution. The ***multi-non-uniform mutation*** operator applies the non-uniform operator to all of the variables in the parent.

Both the non-uniform and multi-non-uniform operators require a slight modification to work in the integer-based GA. Each new value of a particular variable generated by this operator is equal to the current value plus or minus a random displacement. If the displacement is added to the current value, the new value is rounded up as shown below; otherwise, the displacement is subtracted from the current value and the new value is truncated, i.e.,

$$
v_i = \begin{cases} \lceil v_i + (b_i - v_i) f(G) \rceil & \text{if } r_1 \ < \ 0.5, \\ \lfloor v_i - (v_i - a_i) f(G) \rfloor & \text{otherwise,} \end{cases}
$$

where

$$
\begin{aligned}
\lceil x \rceil &= \text{smallest integer greater than or equal to } x, \\
\lfloor x \rfloor &= \text{largest integer less than or equl to } x.
\end{aligned}
$$

4.4.2 Crossover Operators

The ***arithmetic crossover*** operator shown below produces a linear combination of the parents and needs modification to work with the integer representation:

$$
\begin{aligned}
C1 &= aP1 + bP2 \\
C2 &= bP1 + aP2
\end{aligned}
$$

where $0 \leq a \leq 1$, and $b = 1 - a$.

$$
C1 \ = \ (\langle ax_1 + by_1 \rangle, \langle ax_2 + by_2 \rangle, \langle ax_3 + by_3 \rangle, \langle ax_4 + by_4 \rangle, \langle ax_5 + by_5 \rangle)
$$

To achieve the necessary integer representation of the variables, the following function is defined.

$$\langle ax_i + by_i \rangle = \begin{cases} \lceil ax_i + by_i \rceil & \text{if } x_i > y_i \\ \lfloor ax_i + by_i \rfloor & \text{otherwise.} \end{cases}$$

The four mutation operators and one crossover operator discussed are used in their standard or modified form to work with the cell formation problem. In general, the performance of any GA can be improved with the addition of problem specific operators. The integer programming formulation and its unique chromosome structure led to the development of two additional and highly problem-specific crossover operators to improve the speed and quality of the GA's solution. Both new operators are modifications of the simple crossover operator described earlier.

The first problem-specific operator is the ***integer swap crossover***. It differs from the simple float crossover in that, instead of randomly selecting a cut point, the cut point is generated between machines and parts. Effectively, the two parents exchange their part variables, and the children are created by concatenating the two different segments.

$$\begin{aligned} P1 &= (\overbrace{x_1, x_2, \ldots x_m}^{\text{machines}} \mid \overbrace{y_1, y_2, \ldots y_n}^{\text{parts}}) \\ P2 &= (w_1, w_2, \ldots w_m \mid z_1, z_2, \ldots z_n) \end{aligned}$$

The following two children, $C1$ and $C2$, are created from these two parents.

$$\begin{aligned} C1 &= (x_1, x_2, \ldots x_m \mid z_1, z_2, \ldots z_n) \\ C2 &= (w_1, w_2, \ldots w_m \mid y_1, y_2, \ldots y_n) \end{aligned}$$

The multi-point crossover operator also works in a manner similar to the simple float crossover. Rather than randomly selecting a single cut point, several cut points are generated. A ***two-point crossover*** was developed to take advantage of the special structure of the cellular manufacturing representation. One cut point is randomly selected over the range of the machine variables, and the other point is randomly selected over the range of part variables. Effectively, a single-point crossover is performed on the machine variables and another is performed on the part variables. The two parents are split at these points, and the children are created by reassembling the different quarters.

$$\begin{aligned} P1 &= ([x_1, x_2, \mid x_3, x_4, x_5][y_1, y_2, \mid y_3, y_4, y_5]) \\ P2 &= ([w_1, w_2, \mid w_3, w_4, w_5][z_1, z_2, \mid z_3, z_4, z_5]) \end{aligned}$$

Table 1: Average Number of Generations Using Special Operators Versus Simple Crossover

Data Set					Simple Crossover			Special Operators		
	Size	k_{max}	State Space	Best Γ from Lit.	Gen	No. Sol	Avg Γ	Gen	No. Sol	Avg Γ
Simple Chan and Milner (1982)[10]	10x15	3	$8.4729x10^{11}$	0.920	189	6728	0.920	97	3184	0.920
Complex Chan and Milner (1982)[10]	10x15	3	$8.4729x10^{11}$	0.800	286	9232	0.781	98	3216	0.800
Simple Joines (1993)[25]	20x25	5	$2.8422x10^{31}$	0.860	805	25840	0.860	429	13808	0.860
Complex Joines (1993)[25]	20x25	5	$2.8422x10^{31}$	0.775	1156	37072	0.757	671	21552	0.775
Burbidge (1969)[7]	20x35	4	$1.2981x10^{33}$	0.757	1272	40784	0.722	680	21840	0.757
King (1982)[32]	16x43	4	$3.3231x10^{35}$	0.493	20000	640080	0.439	1134	36368	0.493

k_{max} - The number of cells specified. No. Sol - The avg no. of unique solutions visited by the GA in worst case.
Gen - The average number of generations.

The following two children, $C1$ and $C2$, are created from the two parents, $P1$ and $P2$.

$$C1 = ([x_1, x_2, \mid w_3, w_4, w_5][y_1, y_2, y_3, \mid z_4, z_5])$$
$$C2 = ([w_1, w_2, \mid x_3, x_4, x_5][z_1, z_2, z_3, \mid y_4, y_5]$$

The effect of employing these two special genetic operators is shown in Table 1, which compares the average number of generations needed to solve six sample data sets over 10 replications with and without the special operators. For all six data sets in Table 1, a target efficacy value was established, i.e., the best known solution found in the literature. The GA was set up to run for a maximum of 20,000 generations or until the best known value was reached. The table displays the average (over the 10 replications) number of generations needed to reach the target solution. As can be seen, the average grouping efficacy is higher, and the average number of required generations is reduced in all cases where the special operators were employed. Table 1 also indicates the efficiency of the GA by comparing the average number of solutions evaluated (not necessarily unique solutions) versus the size of the state space.

4.5 Termination Criterion

The GA moves from generation to generation selecting and reproducing parents until a termination criterion is met. The most frequently used stopping criterion is a specified maximum number of generations. Another termination strategy involves population convergence criteria. In general, GAs will force much of the entire population to converge to a single solution. When the sum of the deviations among individuals becomes smaller than some specified threshold, the algorithm can be terminated. The algorithm can also be terminated due to a lack of improvement in the best solution over a specified number of generations. Alternatively, a target value for the evaluation measure can be established based on some arbitrarily "acceptable" threshold. Several strategies can be used in conjunction with each other. The experiments in this paper, other than those summarized in Table 1, employ a

maximum number of generations as the stopping criterion.

4.6 Evaluation Measure

Evaluation functions of many forms can be used in a GA, subject to the minimal requirement that the function can map the population into a partially ordered set. The "grouping efficacy" measure was used as the evaluation criterion to test the integer-based genetic algorithm on several data sets from the literature. Grouping efficacy was chosen as the initial evaluation measure because it has been used frequently in the literature and results are available for comparison. It generates block diagonal cell formation and practically viable solutions [33, 48]. Grouping efficacy seeks to minimize the number of exceptional elements and the number of voids (zeros) in the diagonal blocks. Exceptional elements represent inter-cell movements of parts, which reduce the effectiveness of cellular manufacturing. More specifically, grouping efficacy attempts to minimize the number of voids plus the number of exceptional elements divided by the total operational zone. The operational zone is defined by the number of operations (exceptional elements plus operations along the diagonal) plus the number of voids. Grouping efficacy has a value of one when there are no exceptional elements and no voids and a value of zero if the number of exceptional elements equals the total number of operations. Formally, grouping efficacy (Γ) is defined as

$$\max \quad \Gamma = \frac{(1-\psi)}{(1+\phi)} = \frac{1-\frac{e_o}{e}}{1+\frac{e_v}{e}} = \frac{e-e_o}{e+e_v},$$

where:

$$\begin{aligned} e &= \text{the no. of operations in the data matrix,} \\ e_v &= \text{the no. of voids in the diagonal blocks,} \\ e_o &= \text{the no. of exceptional elements.} \end{aligned}$$

Grouping efficacy cannot be used as an objective function in classical integer programming formulations because of its nonlinearity. Using the classical 0-1 assignment variable declarations, the nonlinear form of grouping efficacy is given in Appendix 1. The formulation of grouping efficacy using the cell formation variable declarations is given in Appendix 2.

5 Cell Formation Experiments Using the Integer-Based GA

The integer-based GA was tested on 17 data sets from the literature to study its effectiveness as a clustering tool. In all experiments, the genetic algorithm employed a ranking scheme based on the normalized geometric distribution, an elitist model, and the seven integer-based genetic operators previously described. To maximize cell design flexibility, an implementa-

tion was adopted that allows the user to employ various representation schemes, sets of operators, and evaluation measures. This flexibility is gained through an object-oriented GA design implemented using C++ [25].

The values for the GA parameters in Table 2 were selected after extensive experimentation and were shown to perform well with all the data sets. The values associated with the various genetic operators indicate the number of parents modified by that operator within each generation. For data sets taken from the literature, the maximum number of permisssable cells, k_{max}, was initially set equal to the best known number of cells, k^L, as determined by other cell formation algorithms. The maximum number of generations needed and the population size are dependent on the size of the problem. Since GAs are stochastic search algorithms, the number of generations needed to solve a particular problem also depends on the composition of the initial population. At any generation, the best individual to that point can easily be displayed, allowing the cell designer to run the GA for a fixed number of generations and evaluate the solution. If the solution is not acceptable, the GA can be resumed and continued until final stopping criteria are satisfied.

Table 2: Parameters Used in the GA Experiments

Parameter	Value
no. of Boundary Mutation Operators	4
no. of Uniform Mutation Operators	4
no. of Non-Uniform Mutation Operators	4
no. of Multi-Non-Uniform Mutation Operators	8
no. of Swap Crossover Operators	6
no. of Multi-point Crossover Operators	6
no. of Arithmetical Crossover Operators	6
kmax - maximum permissible no. of cells	k^L
q - the probability of selecting the best individual	0.1
a - parameter used in the arithmetical crossover	0.25
G_{max} - the maximum number of generations	5000 - 20000
Population Size	50-80

The data sets used to evaluate this clustering technique are of varying size and complexity. In each case, correct grouping is defined as being equivalent to the best known configuration provided in the literature based on grouping efficacy. Since the algorithm explicitly assigns machines and parts to cells, no ambiguity exists in determining the cell to which bottleneck machines are assigned. Table 3 summarizes the experimentation results for the 17 sample data sets. In all cases, the integer-based GA was able to determine a configuration with the same number of cells and a grouping efficacy measure at least equal to any previously reported result. Notice that for several of the datasets, the genetic algorithm found better solutions than those given in the literature.

Because grouping efficacy was chosen as the criterion for comparing individuals, the solutions forced the clusters to form along the diagonal block, simultaneously grouping parts and machines. The cell formations for selected data sets are shown in Figures 1- 6. Figure 1

contains the 20 x 35 (part/machine) incidence matrix taken from Burbidge [7]. The solution to this data set contains exceptional elements and is displayed in Figure 2. The next data set, taken from King [32], is a 16 x 43 problem containing two bottleneck machines and is shown with its solution in Figures 3 and 4. Figures 5 and 6 pertain to a large data set selected to test the ability of the algorithm to solve larger scale problems. It is a 100 x 40 matrix taken from Chandrasekharan and Rajagopalan [12] and, as indicated in Table 3, the solution for this data set (no. 17) improves on any published result.

Table 3: Experimental Results

Data Set				Benchmark		GA Result				
no.	Reference	Matrix	e	Γ	k^L	Γ	k	e_o	e_v	Gen.
1	Simple Chan and Milner(1982)[10]	10x15	46	0.9200	3	0.9200	3	0	4	97
2	Complex Chan and Milner(1982)[10]	10x15	49	0.8000	3	0.8000	3	5	6	97.9
3	Simple Joines(1993)[25]	20x25	86	0.8600	5	0.8600	5	0	14	916.1
4	Complex Joines (1993)[25]	20x25	97	0.7748	5	0.7748	5	11	14	800.0
5	Burbidge (1969)[7]	20x35	136	0.7571 0.7609	4 6	0.7571 0.7791	4 5	2 9	37 27	680.4 1785.1
6	King (1982)[32]	16x43	126	N.A.	4	0.4926	4	26	77	1134.7
7	Srinivasan(1990), et al. [54]	10x20	40	0.8163	4	0.8163	4	0	9	262.1
8	Askin and Subramanian (1987)[3]	14x23	58	0.6436	N.A.	0.6951	4	2	28	560.1
9	Chandrasekharan and Rajagopalan(1989) [13]	24x40	105	0.8015	7	0.8015	7	0	26	1637.7
10	Carrie(1973)[9]	24x18	88	0.5410	7	0.5701	7	27	19	986.3
11	Chandrasekharan and Rajagopalan (1986)[11]	8x20	61	0.8525	3	0.8525	3	9	0	119.9
12	Seifoddini (1989)[51]	11x22	78	0.7312	3	0.7312	3	10	15	212.8
13	Stanfel (1985)[56]	14x24	61	0.6555	N.A.	0.7051	5	6	17	895.0
14	Srinivasan, et al. (1990) [54]	16x30	116	0.6783	4	0.6783	4	19	27	432.0
15	Chandrasekharan and Rajagopalan(1989) [13]	24x40	130	0.8510	7	0.8510	7	11	11	2192
16	Chandrasekharan and Rajagopalan(1989) [13]	24x40	131	0.7351	7	0.7351	7	20	20	1789
17	Chandrasekharan and Rajagopalan(1987)[12]	40x100	420	0.8392	11	0.8403	10	36	37	5682.3

Size = no. of Machines by no. of Parts
e = the no. of operations
Benchmark - The best solution reported in literature
GA Result - The best solution generated by GA
Γ = grouping efficacy
k = no. of cells in the GA solution
e_o=no. of exceptional elements
e_v=no. of voids in the diagonal blocks
Gen.=The average no. of generations
k^L = no. of cells contained in best solution from literature
N.A .- Not Available

6 Effects of Constraining the Number of Permissible Cells

The solutions generated by integer programming models are extremely dependent on the *a priori* specification of the desired number of cells [40]. Specifying a maximum number of cells for the GA simply places an upper bound on the number of cells. Recalling that the GA is an optimization tool, it searches for the maximum grouping efficacy value under this constraint. Let k_{max} be the maximum number of cells permitted and k^* be the optimal number of cells.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
0:	1		1																	1			1		1										
1:		1					1			1		1	1					1						1				1			1				
2:	1		1		1										1		1												1		1				
3:		1					1					1	1											1			1								
4:								1						1		1																		1	
5:								1						1		1			1			1				1								1	
6:	1		1		1										1		1			1			1												
7:	1		1		1										1		1			1			1		1				1						
8:								1						1					1			1				1									
9:								1						1		1			1			1				1									
10:				1		1			1		1										1							1		1		1			1
11:				1		1			1		1										1												1		
12:		1										1	1											1											
13:		1					1			1		1	1					1						1			1				1				
14:				1		1			1		1										1								1		1				
15:				1		1			1		1										1								1		1		1		
16:	1		1		1										1		1			1			1		1				1						
17:		1								1		1	1					1						1							1				
18:				1					1		1										1								1		1		1		
19:								1						1					1				1			1									

Figure 1: The original part/machine incidence matrix for the $20x35$ Burbidge problem

Table 4: Solution to the Simple Chan and Milner Problem when $k_{max} > k^*$ ($\Gamma = 0.9200$ and $k^* = 3$)

($k_{max} = 4$ and 349 Generations)

Cell/Family No.	Machines	Parts
Cell/Family 0	1, 4, 7	2, 4, 7, 12, 14
Cell/Family 1	2, 3, 5, 8	0, 3, 5, 8, 13
Cell/Family 2	None	None
Cell/Family 3	0, 6, 9	1, 6, 9, 10, 11

($k_{max} = 5$ and 985 Generations)

Cell/Family No.	Machines	Parts
Cell/Family 0	1, 4, 7	2, 4, 7, 12, 14
Cell/Family 1	0, 6, 9	1, 6, 9, 10, 11
Cell/Family 2	None	None
Cell/Family 3	2, 3, 5, 8	0, 3, 5, 8, 13
Cell/Family 4	None	None

If $k_{max} \geq k^*$, then in the limit and in practice, the GA will find k^* cells. In effect, the upper bound constraint k_{max} is not binding and has no effect on the number of cells formed or the grouping efficacy value. However, if $k_{max} < k^*$, the constraint is binding, and the genetic algorithm will produce a solution with, at most, k_{max} cells at a reduced grouping efficacy value. Since the optimal number of cells k^* is generally unknown, experimentation focused on comparing the GA solution to the best known number of cells from the literature, k^L, as measured by grouping efficacy.

Several experiments using the Chan and Milner problems were conducted to demonstrate the effects of k_{max} as a constraint. Tables 4 and 5 show the results of experiments with the integer-based GA when k_{max}¿k^L, i.e., the maximum permissible number of cells exceeds the number of cells in the best known solution from the literature. For the two Chan and Milner problems (data sets no.1 & no.2), the maximum number of cells specified was $k_{max} = 4$ and $k_{max} = 5$ ($k^L = 3$). The results were revealing in that in all four experiments, the GA determined that three was the appropriate number of cells. As expected, no members were assigned to the additional $k_{max} - k^L$ cell(s).

As the value of k_{max} increases, the state space the GA must explore also grows, thus

Family/Cell #	Parts	Machines
Cell/Family 0	0, 2, 6, 7, 16	0, 2, 4, 14, 16, 19, 22, 24, 28
Cell/Family 1	10, 11, 14, 15, 18	3, 5, 8, 10, 20, 27, 29, 31, 32, 34
Cell/Family 2	1, 3, 12, 13, 17	1, 6, 9, 11, 12, 17, 23, 26, 30
Cell/Family 3	4, 5 ,8, 9, 19	7, 13, 15, 18, 21, 25, 33

	0	2	4	14	16	19	22	24	28	3	5	8	10	20	27	29	31	32	34	1	6	9	11	12	17	23	26	30	7	13	15	18	21	25	33
0:	1	1				1	1	1																											
2:	1	1	1	1	1				1																			1							
6:	1	1	1	1	1	1	1																												
7:	1	1	1	1	1	1	1	1	1																										
16:	1	1	1	1	1	1	1	1	1																										
10:										1	1	1	1	1	1	1	1		1																
11:										1	1	1	1	1				1																	
14:										1	1	1	1	1	1	1																			
15:										1	1	1	1	1	1	1	1																		
18:										1		1	1	1	1	1	1																		
1:																				1	1	1	1	1	1	1	1	1							
3:																				1	1		1	1		1	1								
12:																				1			1	1		1									
13:																				1	1	1	1	1	1	1	1	1							
17:																				1		1	1	1	1	1		1							
4:																													1	1	1				1
5:																													1	1	1	1	1	1	1
8:																													1	1		1	1	1	
9:																													1	1	1	1	1	1	
19:							1																						1	1		1		1	

Figure 2: Integer GA solution for the $20x35$ problem taken from Burbidge(1969) with $\Gamma = 0.7571$

Table 5: Solution to the Simple Chan and Milner Problem when $k_{max} > k^*$ ($\Gamma = 0.9200$ and $k^* = 3$)

($k_{max} = 4$ and 355 Generations)

Cell/Family No.	Machines	Parts
Cell/Family 0	2, 3, 7	0, 2, 3, 5, 13
Cell/Family 1	0, 5, 6, 8	1, 6, 10, 11
Cell/Family 2	1, 4, 9	4, 7, 8, 9, 12, 14
Cell/Family 3	None	None

($k_{max} = 5$ and 886 Generations)

Cell/Family No.	Machines	Parts
Cell/Family 0	2, 3, 7	0, 2, 3, 5, 13
Cell/Family 1	0, 5, 6, 8	1, 6, 10, 11
Cell/Family 2	1, 4, 9	4, 7, 8, 9, 12, 14
Cell/Family 3	None	None
Cell/Family 4	None	None

increasing, on average, the number of generations required. In practice, the number of generations required increases at a much slower rate than the growth of the state space. Figure 7 shows this phenomena for the simple and complex Chan and Milner problems over the range of k_{max} from 3 to 10. For these two relatively small data sets, k_{max} was set equal to the number of unique machines to allow identification of any naturally occurring clusters. A cell designer might wish to begin with k_{max} set to the maximum number of cells believed practical for the facility. In subsequent analysis, the value of k_{max} could be progressively restricted to reflect limits arising from material handling constraints, efforts to achieve effective work team sizes, or excessive machine redundancy requirements, for example.

Experiments involving a range of values for k_{max} were conducted for all 17 data sets in order to explore the behavior of the GA, both when $k_{max} \geq k^L$ and $k_{max} < k^L$. Table 7 summarizes the results. To further analyze constrained aggregation, the complex Chan and

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
0:																																					1					1	
1:		1								1																		1				1					1	1		1		1	
2:							1										1																	1	1	1							
3:					1				1					1					1		1		1						1														
4:					1			1	1					1	1	1			1		1		1						1				1									1		1
5:	1	1				1	1	1				1	1	1			1		1				1									1	1	1			1		1	1		1	1
6:	1												1												1																		
7:	1	1	1					1	1		1	1			1				1	1	1		1	1			1	1			1						1	1			1		1
8:		1		1						1								1										1				1					1	1		1		1	
9:	1											1	1												1	1					1								1				
10:			1						1											1				1			1			1													
11:											1											1		1			1			1													
12:			1																					1																			
13:		1				1											1																		1								
14:					1									1					1		1												1									1		1
15:		1					1			1								1										1									1	1				1	

Figure 3: The original part/machine incidence matrix for the $16x43$ King problem

Family/Cell #	Parts	Machines
Cell/Family 0	3, 4, 7, 14	4, 7, 8, 13, 14, 15, 18, 20, 22, 28, 30, 32,40, 42
Cell/Family 1	0, 1, 8, 15	1, 3, 9, 17, 27, 31, 36, 37, 39 ,41
Cell/Family 2	10, 11, 12	2, 10, 19, 21, 23, 26, 29
Cell/Family 3	2, 5, 6, 9, 13	0, 5, 6, 11, 12, 16, 24, 25, 33, 34, 35, 38

	0	2	4	14	16	19	22	24	28	3	5	8	10	20	27	29	31	32	34	1	6	9	11	12	17	23	26	30	7	13	15	18	21	25	33
0:	1	1				1	1	1																											
2:	1	1	1	1	1				1																			1							
6:	1	1	1	1	1	1	1																												
7:	1	1	1	1	1	1	1	1	1																										
16:	1	1	1	1	1	1	1	1	1																										
10:										1	1	1	1	1	1	1	1		1																
11:										1	1	1	1	1				1																	
14:										1	1	1	1	1	1	1																			
15:										1	1	1	1	1	1	1	1																		
18:										1		1	1	1	1	1	1																		
1:																				1	1	1	1	1	1	1	1	1							
3:																				1	1		1	1		1	1								
12:																				1			1	1		1									
13:																				1	1	1	1	1	1	1	1	1							
17:																				1		1	1	1	1	1		1							
4:																													1	1	1				1
5:																													1	1	1	1	1	1	1
8:																													1	1		1	1	1	
9:																													1	1	1	1	1	1	
19:							1																						1	1		1		1	

Figure 4: Integer GA solution for the $16x43$ problem taken from King (1980) with $\Gamma = 0.4926$

Milner problem, for which $k^L = 3$, is reexamined. Given $k_{max} = 2$, the GA could produce four possible configurations; any one of three different two-cell configurations or a single cell. Table 6 displays the grouping efficacy for each of the four possibilities for the complex Chan and Milner problem. The two cells containing the bottleneck machine were combined to form a larger cell producing the highest grouping efficacy. In this way, the GA determined the "optimal" configuration under the binding constraint $k_{max} = 2$.

The choice of the objective (evaluation) function plays a major role in the generation of solutions. Ng [48] demonstrated that grouping efficacy can be an inappropriate objective function from an algorithmic point of view. For example, when $k_{max} = 4$ was specified for the Burbidge [7] data set (no. 5), the genetic algorithm determined a solution with four cells and $\Gamma = 0.7571$. However, when $k_{max} = 6$ was specified, the algorithm generated a solution with a five-cell configuration and $\Gamma = 0.7791$. Although the grouping efficacy

Table 6: Comparison of Alterntive Configurations with $k_{max} = 2$

Combined	Γ
0 and 1	0.60494
0 and 2	0.46809
1 and 2	0.49887
All 3	0.32667

Table 7: Experiments with binding and nonbinding k_{max}

No.	k^L	$k_{max}(k)$	Γ	Gen.	$k_{max}(k)$	Γ	Gen.	$k_{max}(k)$	Γ	Gen.	$k_{max}(k)$	Γ	Gen.
1	3	5(3)	0.9200	985	4(3)	0.9200	349	3(3)	0.9200	97	2(2)	0.5750	59
2	3	5 (3)	0.8000	886	4 (3)	0.8000	355.3	3 (3)	0.8000	97.9	2 (2)	0.6049	81
3	5	6 (5)	0.8600	1189	5 (5)	0.8600	916	4 (4)	0.6143	429	3 (3)	0.4778	283
4	5	6 (5)	0.7748	1059	5 (5)	0.7748	800	4 (4)	0.5906	671	3 (3)	0.4813	364
5	4 6	6 (5)	0.7791	1798	5 (5)	0.7791	1125	4 (4)	0.7571	680	3 (3)	0.4273	158
6	4	6 (6)	0.5663	1689	5 (5)	0.5538	1432	4 (4)	0.4926	1134	3 (3)	0.4072	489
7	4	6 (4)	0.8163	683	5 (4)	0.8163	389	4 (4)	0.8163	262	3 (3)	0.5758	83
8	N.A.	5 (4)	0.6951	936	4 (4)	0.6951	560	3 (3)	0.5152	356	2 (2)	0.3492	189
9	7	8 (7)	0.8051	1956	7 (7)	0.8015	1637	6 (6)	0.6562	1232	5 (5)	0.5440	768
10	7	7 (7)	0.5701	986	6 (6)	0.5496	753	5 (5)	0.5214	624	4 (4)	0.4855	432
11	3	5 (3)	0.8525	389	4 (3)	0.8525	254	3 (3)	0.8525	119	2 (2)	0.6292	69
12	3	7 (3)	0.7312	595	5 (3)	0.7312	386	3 (3)	0.7312	212	2 (2)	0.5669	129
13	N.A.	6 (5)	0.7051	1596	5 (5)	0.7051	895	4 (4)	0.6705	225	3 (3)	0.5278	183
14	4	12 (6)	0.6899	1896	6 (6)	0.6899	1253	5 (4)	0.6783	896	4 (4)	0.6783	432
15	7	8 (7)	0.8510	2354	7 (7)	0.8510	2192	6 (6)	0.7073	1657	5 (5)	0.5862	1468
16	7	8 (7)	0.7351	1986	7 (7)	0.7351	1789	6 (6)	0.6286	1432	5 (5)	0.5437	1287
17	11	11 (10)	0.8403	6855	10 (10)	0.8403	5682	9 (9)	0.7838	2829	6 (6)	0.5646	2389

value is larger, visual inspection of the solution in Figure 8 reveals a single machine cell (Cell/Family 2) where two of its four assigned parts require no operations on machine four. This occurs because grouping efficacy places more emphasis on minimizing voids in the cells than on preventing exceptional elements. Thus, the four-cell arrangement actually offers a higher quality solution (despite its lower value) than the five-cell solution. To overcome these problems, Ng [48] developed a weighted grouping efficacy, γ (see Appendix 3), which is a more discriminating objective function. Incorporating this new evaluation function with $k_{max} = 6$, the GA produces the best known solution as shown in Figure 9. The GA also showed its ability to find quality solutions using different objective functions (Γ&γ) in cases where the maximum number of permissible cells was specified or omitted.

7 Concluding Remarks

An integer programming formulation has been developed that uses a genetic algorithm to solve the cell formation problem. The method generates quality solutions and provides improved flexibility for analyzing the cell formation problem. The approach allows the system designer to substitute various types of evaluation functions, which permits alternative designs to be generated and reviewed quickly. Cell formation on the basis of multiple attributes (machine cost, processing times, etc.) can be performed, representing a significant improvement

over existing methods that operate strictly on the part/machine incidence matrix.

The integer-based approach allows the designer to incorporate or selectively remove constraints on the number of permissible cells. Unconstrained solutions containing the naturally occurring clusters can be generated as well as constrained solutions. By allowing alternative cell configuration evaluation the power of the technique as a tool of analysis is extended.

A new chromosome representation for integer variables was developed, thus reducing the size of each integer programming problem and providing a viable solution technique for problems of meaningful scale. New genetic operators were developed to improve the performance of the GA, including a swap-crossover operator and a multi-point crossover operator that take advantage of the special structure of the cellular manufacturing design problem. The efficiency of the algorithm was demonstrated by comparing the maximum number of states visited by the GA to the entire state space for sample data sets.

The performance of the integer-based GA model was compared with several conventional clustering techniques. Grouping efficacy was used as an objective function to produce tight clusters along the diagonal of the part/machine incidence matrix. This effective performance measure cannot be represented as a valid objective function in a traditional integer programming formulation. The integer-based GA model found the "best known" or better configurations in all 17 data sets used during experimentation, proving it to be an effective, as well as flexible, clustering technique. The CPU time required for the algorithm ranged from under a second for the smallest problems to under nine and half minutes for the largest problem on a DECstation 5000/25.

A practical advantage of the GA over existing cell formation techniques is its ability to group the parts and machines into families and cells simultaneously. Families and cells are identified and matched by the algorithm and do not require visual analysis of the solution.

Research in this area is still developing and a number of potentially valuable enhancements to this work should include the application of novel, multi-criteria evaluation functions and the addition of constraints that place upper bounds on cell size. The development of improved genetic operators and experimentation with parameter tuning during execution of the GA could lead to better solutions, improved performance, and a definitive stopping criterion.

ACKNOWLEDGMENTS

The authors wish to thank Ron Askin and Asoo Vakharia for their assistance in providing data sets used in the experiments and their thoughtful suggestions. Also, we would like to thank the referees for their insightful comments, which resulted in a much improved paper.

Appendix 1. Nonlinear closed form of Γ using the classical assignment variables

Recall that the following variables and constraints have been defined:

$$
\begin{aligned}
x_{il} &= \begin{cases} 1, & \text{if machine } i \text{ is assigned to cell } l, \\ 0, & \text{otherwise;} \end{cases} \\
y_{jl} &= \begin{cases} 1, & \text{if part } j \text{ is assigned to part family } l, \\ 0, & \text{otherwise;} \end{cases} \\
\sum_{l=1}^{k} x_{il} &= 1 \;\; \forall i = 1, \ldots, m; \\
\sum_{l=1}^{k} y_{jl} &= 1 \;\; \forall j = 1, \ldots, n; \\
k &= \text{the number of cells(families) specified;} \\
m &= \text{the number of machines;} \\
n &= \text{the number of parts;}
\end{aligned}
$$

$$
a_{ij} = \begin{cases} 1, & \text{if part } j \text{ needs processing on machine } i, \\ 0, & \text{otherwise;} \end{cases}
$$

$$
\Gamma = \frac{(1 - \frac{e_o}{e})}{(1 + \frac{e_v}{e})} = \frac{e - e_o}{e + e_v} = \frac{e - (e - e_d)}{e + (D - e_d)} = \frac{e_d}{e + D - e_d};
$$

$$
\begin{aligned}
e &= \text{the no. of operations in the data matrix;} \\
e_v &= \text{the no. of voids in the diagonal blocks;} \\
e_d &= \text{the no. of operations in the diagonal blocks;} \\
e_o &= \text{the no. of exceptional elements;} \\
D &= \text{the area covered by the diagonal blocks.}
\end{aligned}
$$

Given the definition of grouping efficacy (Γ) and the assignment variables, Γ can be defined as follows:

$$
Gamma = \frac{\sum_{l=1}^{k}\sum_{j=1}^{n}\sum_{i=1}^{m} y_{jl}\, x_{il}\, a_{ij}}{\sum_{j=1}^{n}\sum_{i=1}^{m} a_{ij} + \sum_{l=1}^{k}[(\sum_{j=1}^{n} y_{jl})(\sum_{i=1}^{m} x_{il})] - \sum_{l=1}^{k}\sum_{j=1}^{n}\sum_{i=1}^{m} y_{jl}\, x_{il}\, a_{ij}}
$$

Appendix 2,Nonlinear Form of Γ Using the Cell Formation Variable Declarations

Let X be the universal set and $\{A_i | i \in I\}$ be a family of crisp sets, where:

i = set identifier and I = identification set(universe of all sets),

$|A_i|$ = cardinality of set A_i, which equals the number of elements in set A_i (assuming finite countable).

The process which by individuals (x) are determined to be members or non-members of a given set can be defined by a characteristic function. For a given set A_i, this function assigns a value $\mu_{A_i}(x)$ for every $x \in X$ such that

$$\mu_{A_i}(x) = \begin{cases} 1, & \text{if only if } x \in A_i \text{ (i.e., } x \text{ is a member of set } Ai), \\ 0, & \text{otherwise} \end{cases}$$

Recall that the following variables have been defined for the cell formation problem:

$$\begin{aligned} x_i &= l, \quad \text{machine } i \text{ is assigned to cell } l; \\ y_j &= l, \quad \text{part } j \text{ is assigned to family } l. \end{aligned}$$

Let M_l = the set of all machines assigned to machine cell l and

$$\mu_{M_l}(x_i) = \begin{cases} 1, & \text{if only if } x_i \in M_l \text{ (i.e., machine } i \text{ is assigned to cell } l) \\ 0, & \text{otherwise} \end{cases}$$

Let P_l = the set of all parts assigned to part family l and

$$\mu_{P_l}(x_i) = \begin{cases} 1, & \text{if only if } y_j \in P_l \text{ (i.e., part } j \text{ is assigned to family } l) \\ 0, & \text{otherwise} \end{cases}$$

Using this notation, grouping efficacy(Γ) can be defined as follows.

$$\Gamma = \frac{\sum_{l=1}^{k_{ub}} \sum_{j=1}^{n} \sum_{i=1}^{m} \mu_{P_l}(y_j)\, \mu_{M_l}(x_i)\, a_{ij}}{\sum_{j=1}^{n} \sum_{i=1}^{m} a_{ij} + \sum_{l=1}^{k_{ub}} |P_l||M_l| - \sum_{l=1}^{k_{ub}} \sum_{j=1}^{n} \sum_{i=1}^{m} \mu_{P_l}(y_j)\, \mu_{M_l}(x_i)\, a_{ij}}$$

Appendix 3, Weighted grouping efficacy

$$\text{Max } \gamma \;=\; \frac{r(1-\psi)}{\psi + r(1+\phi-\psi)} \;=\; \frac{q(e-e_o)}{q(e+e_v-e_o)+(1-q)e_o} \;\; 1 - \frac{qe_v+(1-q)e_o}{qD+(1-q)e_o}$$

where:

e = the no. of operations in the data matrix;
e_v = the no. of voids in the diagonal blocks;
e_d = the no. of operations in the diagonal blocks;
e_o = the no. of exceptional elements;
D = the area covered by the diagonal blocks.
$r = \dfrac{q}{1-q}, \qquad 0 \le q \le 1$
q = the weight associated with the voids in the diagonal blocks
$(1-q)$ = the weight associated with the exceptional elements

References

[1] M. R. Anderberg. *Cluster Analysis for Applications.* Academic Press, New York, 1973.

[2] R. Askin and K. Chiu. A graph partitioning procedure for machine assignment and cell formation. *IJPR*, 28(8):1555–1572, 1990.

[3] R. Askin and S. Subramanian. A cost-based heuristic for group technology configuration. *IJPR*, 25(1):101–113, 1987.

[4] D. Bertsekas and P. Tseng. Relaxation methods for minimum cost ordinary and generalized network flow problems. *Operations Research*, 36(1):93–114, 1988.

[5] F. Boctor. A linear formulation of the machine-part cell formation problem. *IJPR*, 29(2):343–356, 1991.

[6] W. Boe and C.H. Cheng. A close neighbor algorithm for designing cellular manufacturing systems. *IJPR*, 29(10):2097–2116, 1991.

[7] J.L. Burbidge. An introduction of group technology. In *Seminar on Group Technology*, Turin, 1969.

[8] J.L. Burbidge. The simplification of material flow systems. *IJPR*, 20:339, 1982.

[9] A.S. Carrie. Numerical taxonomy applied to group technology and plant layout. *IJPR*, 11(4):399–416, 1973.

[10] H.M. Chan and D.A. Milner. Direct clustering algorithm for group formation in cellular manufacture. *JMS*, 1(1):65–74, 1982.

[11] M.P. Chandrasekharan and R. Rajagopalan. Modroc: an extension of rank order clustering for group technology. *IJPR*, 24(5):1221–1233, 1986.

[12] M.P. Chandrasekharan and R. Rajagopalan. Zodiac-an algorithm for concurrent formation of part families and machine cells. *IJPR*, 25(6):835–850, 1987.

[13] M.P. Chandrasekharan and R. Rajagopalan. Groupability: Analysis of the properties of binary data matrices for group technology. *IJPR*, 27(6):1035–1052, 1989.

[14] F. Choobineh. A framework for the design of cellular manufacturing systems. *IJPR*, 26(7):1161–1172, 1988.

[15] C.-H. Chu. Cluster analysis in manufacturing cellular formation. *International Journal of Management Science*, 17(3):289–295, 1989.

[16] C.-H. Chu and J.C. Hayya. A fuzzy clustering approach to manufacturing cell formation. *IJPR*, 29(7):1475–1487, 1991.

[17] C.-H. Chu and M. Tsai. A comparison of three array-based clustering techniques for manufacturing cell formation. *IJPR*, 28(8):1417–1433, 1990.

[18] L. Davis. *The Handbook of Genetic Algorithms.* Van Nostrand Reingold, New York, 1991.

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

[20] K. Gunasingh and R. Lashkari. Simultaneous grouping of parts and machines in cellular manufacturing systems - an integer programming approach. *Computers Industrial Engineering*, 20(1):111–117, 1990.

[21] T. Gupta and H. Seifoddini. Production data based similarity coefficient for machine-component grouping decisions in the design of a cellular manufacturing system. *IJPR*, 28(7):1247–1269, 1990.

[22] G. Harhalakis, R. Nagi, and J.M. Proth. An efficient heuristic in manufacturing cell formation for group technology applications. *IJPR*, 28(1):185–198, 1990.

[23] J.H. Holland. *Adaptation in natural and artificial systems.* The University of Michigan Press, Ann Arbor, 1975.

[24] N.L. Hyer and U. Wemmerlov. Group technology in the us manufacturing industry: A survey of current practices. *IJPR*, 27(8):1287–1304, 1989.

[25] J.A. Joines. Manufacturing cell design using genetic algorithms. Ms thesis, North Carolina State University, Raleigh, NC, 1993.

[26] J.A. Joines and C.R. Houck. On the use of non-stationary penalty functions to solve constrained optimization problems with genetic algorithms. In *1994 IEEE International Symposium Evolutionary Computation*, pages 579–584, Orlando, Fl, 1994.

[27] J.A. Joines, R.E. King, and C.T. Culbreth. A comprehensive review of manufacturing cell design. Technical Report NCSU-IE Technical Report 94-20, North Carolina State University, 1994.

[28] J.A. Joines, R.E. King, and C.T. Culbreth. A comprehensive review of manufacturing cell design. *IJFAIM*, forthcoming, 1995.

[29] S. Kaparthi and N.C. Suresh. Machine-component cell formation in group technology: A neural network approach. *IJPR*, 25(6):1353–1367, 1992.

[30] S.K. Khator and S.K. Irani. Cell formation in group technology: A new approach. *Computers Industrial Engineering*, 12(2):131–142, 1987.

[31] J.R. King. Machine-component grouping in production flow analysis: An approach using a rank order clustering algorithm. *IJPR*, 18(2):213–232, 1980.

[32] J.R. King and V. Nakornchai. Machine-component group formation in group technology: Review and extension. *IJPR*, 20(2):117–133, 1982.

[33] K.R. Kumar and M.P. Chandrasekharan. Grouping efficacy: a quantitative criterion for goodness of block diagonal forms of binary matrices in group technology. *IJPR*, 28(2):233–243, 1990.

[34] A. Kusiak. The generalized group technology concept. *IJPR*, 25(4):561–569, 1987.

[35] A. Kusiak. Exgt-s: A knowledge based system for group technology. *IJPR*, 26:887–905, 1988.

[36] A. Kusiak. Branching algorithms for solving the group technology problem. *JMS*, 10(4):332–343, 1991.

[37] A Kusiak. Group technology: Models and solution approaches. In *First Industrial Engineering Research Conference*, pages 349–352, 1992.

[38] A. Kusiak and W. Chow. Efficient solving of the group technology problem. *JMS*, 6(2):117–124, 1987.

[39] A. Kusiak and W. Chow. Decomposition of manufacturing systems. *RA*, 4(5):457–471, 1988.

[40] H. Lee and A. Garcia-Diaz. A network flow approach to solve clustering problems in group technology. *IJPR*, 31(3):603–612, 1993.

[41] J. McAuley. Machine grouping for efficient production. *Production Engineer*, 51(2):53–57, 1972.

[42] W. T. McCormick, Jr., P.J. Schweitzer, and T.W. White. Problem decomposition and data reorganization by a cluster technique. *Operations Research*, 20(5):993–1009, 1972.

[43] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. AI Series. Springer-Verlag, New York, 1994.

[44] Z. Michalewicz and N. Attia. Genetic algorithm + simulated annealing = genocop ii: A tool for nonlinear programming. Technical Report A working paper, University of North Carolina, Charollote, 1993.

[45] J. Miltenburg and W. Zhang. A comparative evaluation of nine will-known algorithms for solving the cell formation in group technology. *Journal of Operations Management*, 10(1):44–72, 1991.

[46] I. Minis, G. Harhalkis, and S. Jajodia. Manufacturing cell formation with multiple, functionally identical machines. *Manufacturing Review*, 3(4):252–261, 1990.

[47] C.T. Mosier and L. Taube. Weighted similarity measure heuristics for the group technology machine clustering problem. *International Journal of Management Science*, 13(6):577–583, 1985.

[48] S. Ng. Worst-case analysis of an algorithm for cellular manufacturing. *EJOR*, 69(3):384–398, 1993.

[49] G.F.K. Purcheck. A linear programming method for the combinatorial grouping of an incomplete power set. *Journal of Cybernetics*, 5:51–76, 1975.

[50] R. Rajagopalan and J.L. Batra. Design of cellular production systems: A graph-theoretic approach. *IJPR*, 13(6):567–579, 1975.

[51] H. Seifoddini. Single linkage versus average linkage clustering in machine cell formation applications. *Computers Industrial Engineering*, 16:419–426, 1989.

[52] H. Seifoddini. A probabilistic model for machine cell formation. *JMS*, 9(1):69–75, 1990.

[53] H.M. Selim, R.G. Askin, and A.J. Vakharia. Cell formation in group technology: Review, evaluation and directions of future research. Technical Report Working Paper, University of Arizona, Tucson, 1994.

[54] G. Srinivasan, T. Narendran, and B. Mahadevan. An assignment model for the part-families problem in group technology. *IJPR*, 28(1):145–152, 1990.

[55] G. Srinivasan and T.T. Narendran. Grafics-a non hierarchical clustering algorithm for group technology. *IJPR*, 29(3):463–478, 1991.

[56] L.E. Stanfel. Machine clustering for economic production. *Engineering Costs and Production Economics*, 9:73–81, 1985.

[57] H. Steudal and A. Ballakur. A dynamic programming based heuristic for machine grouping in manufacturing cell formation. *Computers Industrial Engineering*, 12(4):215–222, 1987.

[58] T. Vohra, D. Chen, J. Chang, and H. Chen. A network approach to cell formation in cellular manufacturing. *IJPR*, 28(11):2075–2084, 1990.

[59] H. Wu, M. Venugopal, and M. Barash. Design of a cellular manufacturing system: A syntactic pattern recognition approach. *JMS*, 5(2):81–88, 1986.

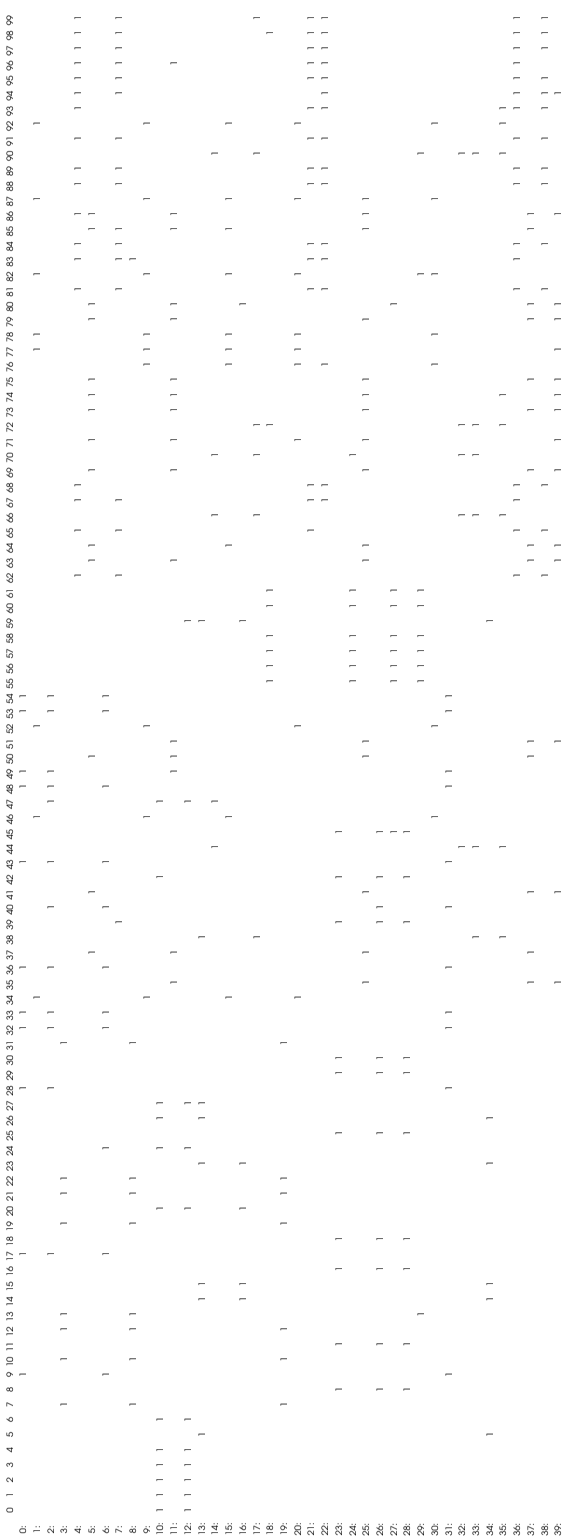

Figure 5: Chandrasekharan and Rajagopalan 40×100 part/machine incidence matrix

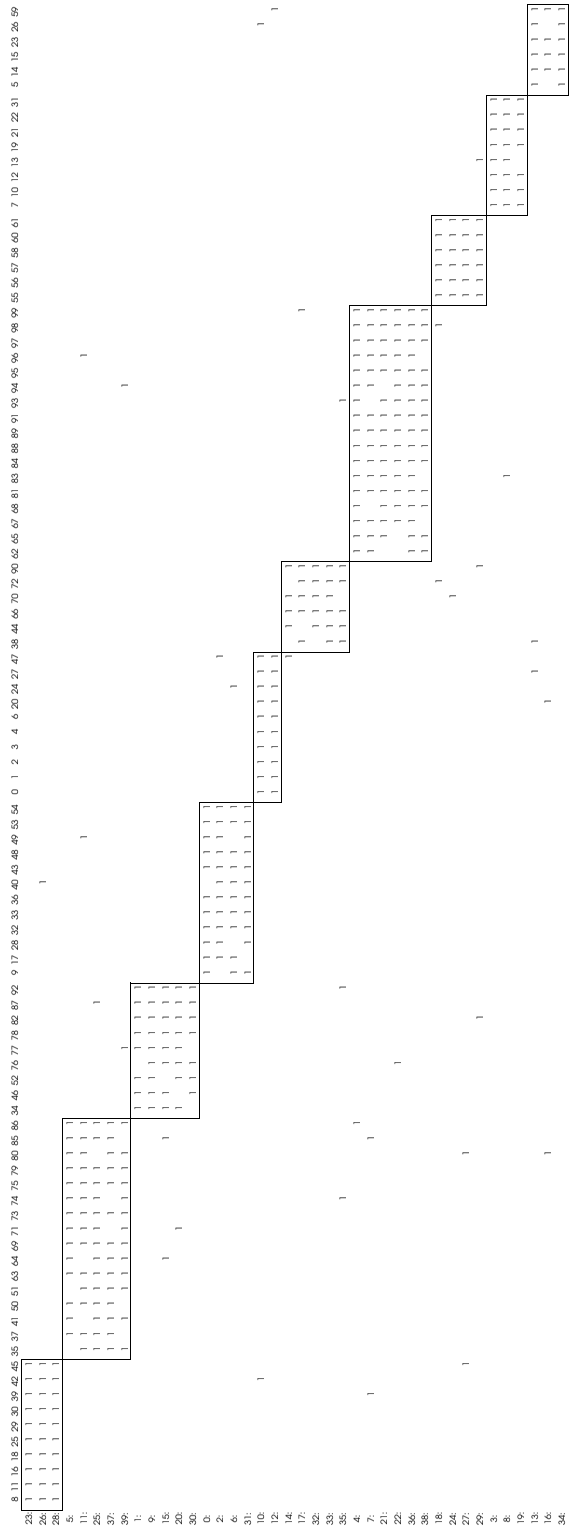

Figure 6: Integer GA solution for the 40×100 Chandrasekharan and Rajagopalan problem with $\Gamma = 0.8403$

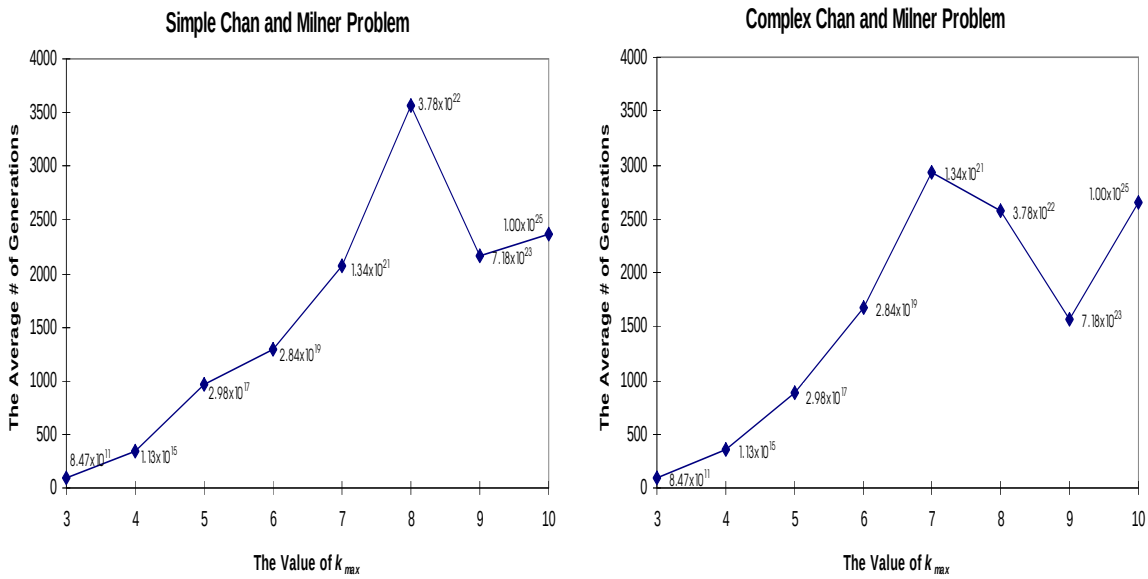

Figure 7: Average number of generations versus selcted values of k_{max}; Values in the graphs represent the number of unique solutions in the state space for each value of k_max

Family/Cell #	Parts	Machines
Cell/Family 0	1, 3, 12, 13, 17	1, 6, 9, 11, 12, 17, 23, 26, 30
Cell/Family 1	10, 11, 14, 15, 18	3, 5, 8, 10, 20, 27, 29, 31
Cell/Family 2	4	15, 32, 33, 34
Cell/Family 3	None	None
Cell/Family 4	0, 2, 6, 7, 16	0, 2, 4, 14, 16, 19, 22, 24, 28
Cell/Family 5	5, 8, 9, 19	7, 13, 18, 21, 25

	1	6	9	11	12	17	23	26	30	3	5	8	10	20	27	29	31	15	32	33	34	0	2	4	14	16	19	22	24	28	7	13	18	21	25
1:	1	1	1	1	1	1	1	1	1																										
3:	1	1		1	1		1	1																											
12:	1			1	1		1																												
13:	1	1	1	1	1	1	1	1	1																										
17:	1		1	1	1	1	1		1																										
10:										1	1	1	1	1	1	1	1				1														
11:										1	1	1	1	1					1																
14:										1	1	1	1	1	1	1																			
15:										1	1	1	1	1	1	1	1																		
18:										1		1	1	1	1	1	1																		
4:																		1		1											1	1			
0:																						1	1				1	1	1						
2:									1													1	1	1	1	1				1					
6:																						1	1	1	1	1	1	1							
7:																						1	1	1	1	1	1	1	1	1					
16:																						1	1	1	1	1	1	1	1	1					
5:																		1		1											1	1	1	1	1
8:																															1	1	1	1	1
9:																		1													1	1	1	1	1
19:																												1			1	1	1		1

Figure 8: Integer GA solution for the $20x35$ problem taken from Burbidge(1969) with $\Gamma = 0.7571$

Family/Cell #	Parts	Machines
Cell/Family 0	4, 5 ,8, 9, 19	7, 13, 15, 18, 21, 25, 33
Cell/Family 1	None	None
Cell/Family 2	0, 2, 6, 7, 16	0, 2, 4, 14, 16, 19, 22, 24, 28
Cell/Family 3	10, 11, 14, 15, 18	3, 5, 8, 10, 20, 27, 29, 31, 32, 34
Cell/Family 4	1, 3, 12, 13, 17	1, 6, 9, 11, 12, 17, 23, 26, 30
Cell/Family 5	None	None

	7	13	15	18	21	25	33	0	2	4	14	16	19	22	24	28	3	5	8	10	20	27	29	31	32	34	1	6	9	11	12	17	23	26	30
4:	1	1	1				1																												
5:	1	1	1	1	1	1	1																												
8:	1	1		1	1	1																													
9:	1	1	1	1	1	1																													
19:	1	1		1		1								1																					
0:								1	1				1	1	1																				
2:								1	1	1	1	1				1																			1
6:								1	1	1	1	1	1	1																					
7:								1	1	1	1	1	1	1	1	1																			
16:								1	1	1	1	1	1	1	1	1																			
10:																	1	1	1	1	1	1	1	1		1									
11:																	1	1	1	1	1				1										
14:																	1	1	1	1	1	1	1												
15:																	1	1	1	1	1	1	1	1											
18:																	1		1	1	1	1	1	1											
1:																											1	1	1	1	1	1	1	1	1
3:																											1	1		1	1		1	1	
12:																											1			1	1		1		
13:																											1	1	1	1	1	1	1	1	1
17:																											1		1	1	1	1	1		1

Figure 9: Integer GA solution for the $20x35$ problem taken from Burbidge(1969) with $\Gamma = 0.7571$