A Hybrid-Genetic Algorithm for Manufacturing Cell Design

Jeffrey A. Joines
Graduate Research Associate

Michael G. Kay
Assistant Professor

Russell E. King
Associate Professor

February 24, 1997

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

This research was sponsored in part, by the NCSU Furniture Manufacturing and Management Center, Office of Naval Research Grant N0014-90-J-1045, and the National Science Foundation (Grant # DDM-9215432).

Abstract

Global competition is demanding innovative ways of achieving manufacturing flexibility and reduced costs. One approach is through cellular manufacturing, an implementation of the concepts of group technology. The design of a cellular manufacturing system requires that a part population be at least minimally described by its use of process technology (part/machine incidence matrix), and 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. This paper presents a stochastic global optimization technique utilizing genetic algorithms (GAs) and local improvement procedures to solve the cell formation problem. The combination of local improvement procedures with GAs is shown to improve the performance of the GA in terms of quality of solution and computational efficiency. Several different incorporation methods are investigated. The concepts of these hybrid techniques can easily be extended to other variations of the cell formation problem as well as to other local improvement procedures.

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 is to develop a cellular manufacturing system using the principles of group technology (GT) [18]. The manufacturing system is decomposed into several manageable subsystems, or groups, by aggregating similar parts into part families and dissimilar machines into cells [27]. Through the creation of independent cells, automation and/or control can be simplified. Companies using manufacturing cells have reported reduced setup times, throughput time, work-in-process inventory, and material handling as well as an improvement in quality [3, 6, 18]. The benefits of cellular manufacturing are often reduced when intercell movements associated with exceptional elements occur. Exceptional elements are parts that require processing outside their assigned cells [22].

A common objective for the designers of cellular manufacturing systems is to create a set of autonomous manufacturing units that eliminates the intercell movement of parts. Techniques for partitioning the part/machine incidence matrix into machine cells and associated part families are varied and extensive.

Cell formation techniques that operate on the part/machine incidence matrix are generally referred to as the second phase of Production Flow Analysis (PFA) [2] and involve combinatorial optimization. Cell formation 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.

Mathematical programming approaches to the cell formation problem are nonlinear or linear integer programming problems. 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. Second, the number of machine cells must be specified a priori, affecting the grouping process and potentially obscuring 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 [28]. However, these approaches are able to incorporate other meaningful manufacturing information (i.e., volume demand, redundant machinery, alternative part operations or fixed routings, machine capacity constraints, etc.)

There is an extensive literature devoted to heuristic cell formation techniques, however most do not guarantee optimality and typically work exclusively with a part/machine incidence matrix [24]. While some of these clustering algorithms offer superior results for specific applications, no single technique has been shown to provide the best solution for a broad range of applications. These approaches typically involve single criteria objective functions and do not permit an interchange of evaluation functions or the selective use of constraints. Several meaningful evaluation criteria have also been proposed that cannot explicitly be used as objective functions in clustering algorithms. Most heuristics cannot identify all naturally occurring clusters as well as find solutions with a constrained number of clusters. The interested reader is referred to the comprehensive reviews by Selim et al. [38] and Joines et al. [24] for more details.

To overcome these limitations, an approach using a genetic algorithm (GA) to solve integer programming formulations of the cell design problem was developed by Joines et al. [21]. The GA approach offers several advantages over existing techniques. The general integer variables use in the GA reduce the size of each problem as compared to the use of binary part/machine incidence variables. The GA simultaneously groups the parts and machines into families and cells and assigns them to each other without visual analysis of the solution. The designer can incorporate or remove constraints on the number of permissible cells. Unconstrained solutions containing the naturally occurring clusters can be generated as well as constrained solutions. The integer-based GA approach allows the system designer freedom to substitute various types of evaluation functions, permitting alternative designs to be generated and reviewed quickly. Cell formation on the basis of multiple attributes (machine cost, processing times, etc.) can be performed, eliminating the restriction imposed by most existing methods that operate strictly on the part/machine incidence matrix. Additional

information that is not contained in the part/machine incidence matrix can be considered, such as redundant machines and alternative routings [23].

In this paper, the GA approach is extended to incorporate local improvement procedures (LIP)s. An example of one LIP is developed and incorporated into the GA to demonstrate the hybrid-GA capabilities. Through this incorporation, the GA's performance in terms of solution quality and computational efficiency is improved. The hybrid-GA approach is tested on several data sets and compared with the GA without the inclusion of the LIP.

Section 2 provides an introduction to the genetic algorithm approach used to find solutions to the cell formation problem. Incorporation of LIPs to form hybrid-GAs is discussed in Section 3. In Section 4, an LIP for the cell formation problem is discussed and incorporated into the GA. Section 5 describes experimentation using the GA (with and without the LIP) applied to problems from the literature and industry. Section 6 analyzes the effectiveness of the LIP. Finally, Section 7 summarizes the significance of the work presented.

2 Cell Formation Using an Integer-based GA

The genetic algorithm used for this research is based upon the one described in Joines et al. [21] and is described briefly here. A common integer programming formulation of the cell design problem with k cells , m machines and n parts requires $k(m+n)$ assignment variables and $m+n$ constraints:

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

The constraints ensure that each machine and part is assigned to only one cell or family, respectively. As the number of parts and machines increases, the models become too

large to be stored in memory or become computationally intractable because of the integer constraints [28]. To overcome these limitations, an integer programming formulation was developed using the following general integer variables as a set notation that incorporates the assignment constraints that each machine and part can only be assigned to one cell/family:

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

This formulation reduces the number of variables by a factor of k to $m+n$ and eliminates the $m+n$ constraints. Each part and machine variable is equal to the number of its assigned family or cell. The part families are assigned to the respective machine cell of the same number.

A conventional integer programming solution technique cannot be employed because of the objective function's inability to decode this variable representation. However, genetic algorithms (GAs) have been used successfully to solve difficult problems where the objective function does not possess "nice" properties such as continuity, differentiability, satisfaction of the Lipschitz Condition, etc. [8, 12, 15, 30]. GAs maintain and manipulate a family, or population, of solutions in their search for better solutions by implementing a "survival of the fittest" strategy. This provides an implicit as well as explicit parallelism that allows for the exploitation of several promising areas of the solution space at the same time. The genetic algorithm used to solve the cell formation problem is summarized in Algorithm 1. A more complete discussion of genetic algorithms, including extensions and related topics, can be found in Davis, [8] Goldberg, [12] Holland, [15] and Michalewicz [30].

Each individual in the population is described by a chromosome representation which is a sequence of variables. For the cell formation problem, 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 with a range of 1 to the maximum number of cells or families ($k_{\max}$):

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

Unlike most integer formulations, $k_{\max}$ represents an upper bound on the number of cells rather than the explicit number of cells to form. Therefore, the GA can be used to determine the naturally occurring clusters, as well as to constrain the number of cells due to space or management concerns [21].

The initial population is created either randomly or by seeding, and each individual is

Algorithm 1 A Simple Genetic Algorithm

1. Set generation counter $i \leftarrow 0$.
2. Create the initial population, Pop(i), by randomly generating N individuals.
3. Determine the fitness of each individual in the population by applying the objective function to the individual and recording the value found.
4. Increment to the next generation, $i \leftarrow i + 1$.
5. Create the new population, Pop(i), by selecting N individuals stochastically based on the fitness from the previous population, Pop($i - 1$).
   (a) Randomly select R parents from the new population to form the new children by application of the genetic operators.
   (b) Evaluate the fitness of the newly formed children by applying the objective function.
6. If $i <$ the maximum number of generations to be considered, go to Step 4.
7. Print out the best solution found.

evaluated using the objective function to determine its fitness or value. 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 totally ordered set.

Even though Ng [33] and Joines et al. [21] showed that it is not an ideal objective function in an algorithmic sense, "grouping efficacy" [26] was used in this study as the evaluation criterium for two reasons. First, maximizing grouping efficacy forces block diagonal cell formation to occur [26]. Second, it has been used in the past by many researchers and thus is useful for comparing solutions generated by the GA to those of other techniques found in the literature [26]. However, it should be noted that the approach is not limited to this objective.

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. An exceptional element is a part operation which is performed outside the parts designated cell. Formally, grouping efficacy (Γ) is defined as

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

where e is the number of operations in the data matrix, e_v is the number of voids in the diagonal blocks, and e_o is the number of exceptional elements.

Grouping efficacy cannot be used as an objective function in classical integer program-

ming formulations because of its nonlinear form, demonstrating one aspect of the GA's flexibility in incorporating a variety of evaluation functions [21]. This paper demonstrates the usefulness of this property.

After the population (of size N) has been evaluated, a new population of size N individuals is selected from the previous generation. The selection of individuals to produce successive generations plays an extremely important role in a genetic algorithm. These individuals do not have to be distinct; that is, an individual in the population can be selected more than once. A probabilistic selection is performed where each individual is assigned a probability based upon its fitness (objective function value) such that the better individuals have an increased chance of being selected. However, all individuals in the population have a chance of being selected to help produce the next generation. The genetic algorithm used in these experiments uses a normalized geometric ranking scheme and employs the elitist model in which the best individual from the previous generation is always included in the current one [21].

After the new population is selected, R parents are randomly chosen from the population to produce children by applying genetic operators. Mutation and crossover are the two basic types of genetic operators [30]. Mutation operators tend to make small random changes in one parent to form one child in an attempt to explore all regions of the state space. Crossover operators combine information from two parents to form two offspring such that the children inherit a "likeness" (a set of building blocks) from each parent. The specific application of the basic genetic operators and their derivatives, depends on the chromosome representation used. Joines et al. [21] modified six float operators developed by Michalewicz [30] to work with an integer representation: uniform mutation, non-uniform mutation, multi-non-uniform mutation, boundary mutation, simple crossover, and arithmetic crossover. Based on the cell formation representation, two problem-specific genetic operators, cell-swap crossover and cell-two-point crossover were also developed and shown to enhance the GA's performance substantially.

The uniform mutation operator randomly selects one of the variables from a parent and sets it equal to a random integer number uniformly distributed between the variable's lower bound, 1, and upper bound, $k_{\max}$. The boundary mutation operator randomly selects one of the variables from a parent and randomly sets it equal to its integer lower or upper bound.

Michalewicz [30] developed two mutation operators to help with local searching: non-uniform mutation and multi-non-uniform mutation. In early generations, this operator is similar to the uniform mutation operator; but as the number of generations increases, the distribution narrows, increasing the exploitation of the local solution. GAs which incorporate these operators have been shown to outperform those which do not [30]. The non-uniform

mutation operator randomly selects one of the variables from a parent and sets it equal to a random integer number from a non-uniform distribution [30]. The multi-non-uniform mutation operator applies the non-uniform operator to all of the variables in the parent. The simple crossover operator randomly selects a cut point, dividing each parent into two segments. The first child is created by combining the first segment from the first parent and the second segment from the second parent. The second child is created from the two remaining segments of the parents. The arithmetic crossover operator produces a complementary pair of linear combinations produced from random proportions of the parents.

The first of the problem-specific operators, cell-swap crossover, performs a simple crossover in which the cut point is always generated between the machine and part variables. Effectively, the two parents exchange their part variables and the children are created by concatenating the two different segments. The second problem-specific operator, cell-two-point crossover, works in a similar manner to simple crossover. Rather than randomly selecting a single cut point, two cut points are generated. 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. For more detailed information on the problem-specific operators and the modifications made to the standard float operators, see Joines et al. [21].

The GA moves from generation to generation, repeating Steps 4–6 of Algorithm 1 until the termination criterion is met. Some termination strategies include the use of population convergence criteria and checking to see if there has been no improvement of the best individual over a specified number of generations. Several termination strategies can be used in conjunction as well. However, the stopping criterion used in these experiments is the specification of a maximum number of generations. This allows one to preset the maximum number of (not necessarily unique) solutions that are evaluated. In this study, we preset the maximum number of generations and then determine at which generation the GA finds the best solution.

3 Incorporation of a Local Improvement Procedure

Many researchers [7, 8, 16, 17, 30, 34, 35] have shown that GAs perform well for global searching because they are capable of quickly finding and exploiting promising regions of the search space, but they take a relatively long time to converge to a local optimum. For example, Figure 1 shows three replications of a typical GA run for the 100×40 problem of Chandrasekharan and Rajagopalan [4] (referred to as Chan). As can be seen, the GA quickly

finds a good solution but requires many more generations to reach the optimal solution (or the best known solution). Table 1 shows the number of generations for each replication along with the mean number of operations to reach a certain percent of optimal. On average, the GA reaches 90% of optimal after only 65% of the total computation time. It takes approximately a third of the time to go from 90% of optimal to optimal. Michalewicz [30] and Joines [20] developed two mutation operators to help with local searching: non-uniform mutation and multi-non-uniform mutation. As mentioned in Section 2, these operators are similar to the uniform mutation operator in early generations; but as the number of generations increases, the distribution narrows, increasing the exploitation of the local solution.

% of Optimal	Repl. 1 Gen.(% Gen.)	Repl. 2 Gen.(% Gen.)	Repl. 3 Gen.(% Gen.)	Mean Gen.(% Gen.)
10	2 (0.2)	5 (0.3)	3 (0.2)	3.3 (0.2)
25	108 (10.4)	95 (6.1)	89 (5.2)	97.3 (6.8)
50	294 (28.3)	265 (16.9)	256 (15.0)	271.7 (18.9)
75	440 (42.3)	441 (28.2)	884 (51.7)	588.3 (40.9)
90	623 (60.0)	1021 (65.2)	1173 (68.6)	939.0 (65.3)
95	657 (63.2)	1233 (78.7)	1275 (74.5)	1055.0 (73.3)
99	855 (82.3)	1566 (100.0)	1711 (100.0)	1377.3 (95.7)
100	1039 (100.0)	1566 (100.0)	1711 (100.0)	1438.7 (100.0)

Table 1: The Number of Generations Needed to Reach % Optimal for Chan Dataset

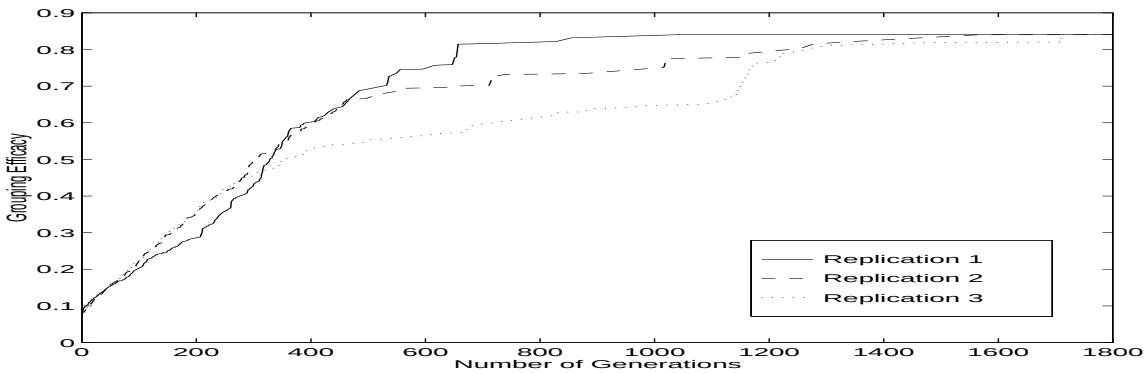

Figure 1: The Solution Quality of the GA versus Generations for Chan Dataset

Local improvement procedures, e.g., two-opt switching for combinatorial problems and gradient descent for unconstrained nonlinear problems, quickly find the local optimum of a small region of the search space, but are typically poor global searchers. Because these

procedures do not guarantee optimality, in practice, several random starting points are generated and used as input into the local search technique and the best solution is recorded. This global optimization technique (multi-start) has been used extensively but is a blind search technique since it does not take account past information [36, 17]. Genetic algorithms, unlike multi-start, utilize past information in the search process. Therefore, local improvement procedures (LIPs) have been incorporated into GAs in order to improve their performance through what could be termed "learning." Such hybrid-GAs have been used successfully to solve a wide variety of problems [7, 8, 17, 16, 30, 35]. Houck, Joines, and Kay [17] showed that for the continuous location-allocation problem, a GA that incorporated a LIP outperformed multi-start and a two-way switching procedure, where both methods utilized the same local improvement procedure as the hybrid-GA.

There are several ways local improvement procedures can be incorporated into a GA. The following methods are used in the experimentation described in Section 5 in order to gain insight into which implementation method is best.

1. Run the GA without the LIP and then apply it to the final solution obtained by the GA. This allows the precise local optimum around the final solution to be found.

2. Use the LIP as the GA's evaluation function. (Recall, the GA makes no assumptions on the form of the objective; only that it map the individuals in the population into a totally ordered set.) The LIP is used to evaluate individual in the population (i.e., determine the best objective value for this starting assignment).

3. Apply the LIP as a genetic operator in the same manner as any mutation operator. In this approach, the procedure is only applied to a small portion of the parents (i.e., the parents selected to undergo this particular type of mutation) rather than to all of the children created. Chu and Beasily [7] used a LIP as a genetic operator for the generalized quadratic assignment problem. This incorporation improved the quality of the solution as well as the computational efficiency of the GA.

Incorporating a LIP as an evaluation function gives rise to the concepts of the Baldwin Effect and Lamarckian evolution. Also, the concept of a one-to-one genotype to phenotype mapping is introduced, where genotype refers to the space the GA searches while the phenotype refers to the space of the actual problem.

3.1 Utilizing the Baldwin Effect and Lamarckian Evolution

Local improvement procedures have been incorporated into GAs in order to improve the algorithm's performance through learning. There are two basic models of evolution that have

been used to incorporate learning into a GA: the Baldwin Effect and Lamarckian evolution. The Baldwin Effect allows an individual's fitness (phenotype) to be determined based on learning, i.e., the application of local improvement. Like natural evolution, the result of the improvement does not change the genetic structure (genotype) of the individual, it just increases the individual's chances of survival. Lamarckian evolution, in addition to using learning to determine an individual's fitness, changes the genetic structure of an individual to reflect the result of the learning. Both "Baldwinian" and "Lamarckian" learning have been investigated in conjunction with hybrid GAs [35, 41].

3.1.1 Baldwin Effect

The Baldwin Effect, as utilized in genetic algorithms, was first investigated by Hinton and Nolan [14] using a flat landscape with a single well representing the optimal solution. Individuals were allowed to improve by random search, which in effect transformed the landscape to include a funnel around the well. Hinton and Nolan showed that, without learning, the genetic algorithm fails; however, with the random search, the GA is capable of finding the optimum.

Whitley et al. [41] demonstrated that "exploiting the Baldwin Effect need not require a needle in a haystack and improvements need not be probabilistic." They showed how using a LIP can, in effect, change the landscape of the fitness function into flat landscapes around the local basins. This transformation increases the likelihood of allocating more individuals to certain basins. In a comparison of Baldwinian and Lamarckian learning, Whitley et al. [41] showed that utilizing either form of learning is more effective than the standard GA approach without the LIP (a bitwise steepest ascent algorithm performed on a binary representation). Whitley et al. [41] argued that, while Lamarckian learning is faster, it may be suspectable to premature convergence to a local optimum as compared to Baldwinian learning. Three numerical optimization problems were used to test this conjecture; however, the results were inconclusive.

3.1.2 Lamarckian Evolution

Lamarckian learning forces the genotype to reflect the result of some form of local improvement. This results in the inheritance of acquired or learned characteristics that are well-adapted to the environment. The improved individual is placed back into the population and allowed to compete for reproductive opportunities. However, Lamarckian learning inhibits the schema processing capabilities of genetic algorithms [13, 40, 41]. Changing the genetic information in the chromosomes results in a loss of inherited schema, altering the

statistical information about hyper-plane partitions implicitly contained in the population.

While Lamarckian learning may disrupt the schema processing of a genetic algorithm, Baldwinian learning certainly aggravates the problem of multiple genotype to phenotype mappings. A genetic algorithm works on both genotypes and phenotypes. A genotype refers to the composition of the values in the chromosome or individual in the population, whereas a phenotype refers to the solution that is constructed from a chromosome. In a direct mapping, there is no distinction between genotypes and phenotypes. For example, to optimize the function $5\cos(x_1) - \sin(2x_2)$, a typical representation for the chromosome would be a vector of real numbers (x_1, x_2), which provides a direct mapping to the phenotype. However, for some problems, a direct mapping is not possible or desired [32]. The most common example of this is the traveling salesperson problem with an ordinal representation. Here, the genotype is represented by an ordered list of n cities to visit. However, the phenotype is a tour, and any rotation of the chromosome yields the same tour; thus, any rotation of a genotype results in the same phenotype. For example, the two tours (1,2,3,4) and (3,4,1,2) have different genotypes since their gene strings are different, but both strings represent the same tour and thus have the same phenotype.

It has been noted that having multiple genotypes map to the same phenotype may confound the GA [14, 32]. This problem also occurs when an LIP is used in conjunction with a GA. Consider the example of maximizing $\sin(x)$. Suppose a simple gradient-based LIP is used to determine the fitness of a chromosome. Then any genotype between $[-\pi/2, 3\pi/2]$ will have the same phenotype value of 1 at $\pi/2$.

3.1.3 Partial Lamarckian

Hybrid genetic algorithms need not be restricted to operating in either a pure Baldwinian or pure Lamarckian manner. Instead, a mix of both strategies, or what is termed "partial Lamarckianism"[16] could be employed. For example, a possible strategy is to update the genotype to reflect the resulting phenotype in 50% of the individuals. While this 50% partial Lamarckian strategy has no justification in natural evolution, for simulated evolution this mix is as valid as either pure Lamarckian or pure Baldwinian search.

Orvosh and Davis [9] advocated the use of a 5% rule for updating individuals when employing repair functions in genetic algorithms to solve constrained optimization problems. All infeasible solutions generated by the genetic algorithm are repaired to the feasible domain in order to determine their fitness. The 5% rule dictates that 5% of the infeasible individuals have their genetic representation updated to reflect the repaired feasible solution. This partial updating was shown on a set of combinatorial problems to be better than either

no updating or always updating. However, Michalewicz and Nazhiyath [31] determined that a higher percentage 20–25% of update did better when using repair functions to solve continuous constrained nonlinear programming problems.

Previous research has concentrated either on the comparison of pure Lamarckian and pure Baldwinian search, or the effectiveness of partial repair for constrained optimization. This work examines the use of partial Lamarckianism with regard to the use of LIPs on a set of bounded optimization problems.

4 Grouping Efficacy Local Improvement Procedure

To investigate the capabilities of the hybrid-GA approach for the cell formation problem, an LIP has been developed for the grouping efficacy measure (Γ) as s shown in Algorithm 2. The following notation is necessary and is used in the algorithm:

n = number of parts
m = number of machines
$k_{\max}$ = maximum number of cells/families
n_l = set of parts assigned to part family l
m_l = set of machines assigned to machine cell l
$a_{ij} = \begin{cases} 1, & \text{if part } j \text{ requires processing on machine } i \\ 0, & \text{otherwise} \end{cases}$
p_{il} = number of parts needing processing by machine i in cell l
q_{jl} = number of machines that operate on part j in family l
c = index of current cell/family assignment machine/part
c^* = index of the best cell/family to switch a machine/part to
α_{il}, α_{jl} = increase in the number of exceptional elements if a switch from c to l occurs for machine i or part j, respectively
β_{il}, β_{jl} = decrease in the number of voids if a switch from c to l occurs for machine i or part j, respectively

The improvement procedure is a single switching algorithm which utilizes switching rules defined by Ng [33]. Ng proved that switching machine i from its current cell c to cell l will only increase Γ if and only if Rule 1 is true.

$$\text{If } \beta_{il}\,\Gamma \;>\; \alpha_{il}, \text{ then switch machine } i \text{ from cell } c \text{ to cell } l. \qquad \text{(Rule 1)}$$

Likewise, switching part j from the current family c to family l will increase Γ if and only if

Algorithm 2 Grouping Efficacy Based Local Improvement Procedure

Step 1: For a given machine cell and part family assignment $(x_1, x_2, \ldots x_m, y_1, y_2, \ldots, y_n)$, compute $\Gamma = \frac{e-e_o}{e+e_v}$ which determines P_l and M_l for each cell $l = 1, \ldots, k_{\max}$.

Step 2: (Machine switching) For $i \leftarrow 1$ to m do

2a: For $l \leftarrow 1$ to $k_{\max}$ do

$$p_{il} \leftarrow \sum_{j \in \{J | y_j = l\}} a_{ij}, \qquad \text{where } J = \{1, 2, \ldots, n\}$$

2b: $c \leftarrow x_i$

2c: Determine the best cell switch for machine i:

$$c^* \leftarrow \arg\max_l \{\beta_{il}\,\Gamma - \alpha_{il} | c \neq l\},$$

where $\alpha_{il} \leftarrow p_{ic} - p_{il}$ and $\beta_{il} \leftarrow |P_c| - |P_l| - \alpha_{il}$.

2d: If $\beta_{ic^*}\,\Gamma - \alpha_{ic^*} > 0$ then

i. $M_{c^*} \leftarrow M_{c^*} + i$, ii. $M_c \leftarrow M_c - i$,
iii. $x_i \leftarrow c^*$, iv. $\Gamma \leftarrow \frac{e-e_o-\alpha_{ic^*}}{e+e_v-\beta_{ic^*}}$

Step 3: (Part Switching) For $j \leftarrow 1$ to n do

3a: For $l \leftarrow 1$ to $k_{\max}$ do

$$q_{ij} \leftarrow \sum_{i \in \{I | x_i = l\}} a_{ij}, \qquad \text{where } I = \{1, 2, \ldots, m\}$$

3b: $c \leftarrow y_j$

3c: Determine the best family switch for part j:

$$c^* \leftarrow \arg\max_l \{\beta_{jl}\,\Gamma - \alpha_{jl} | c \neq l\},$$

where $\alpha_{jl} \leftarrow q_{jc} - q_{jl}$ and $\beta_{jl} \leftarrow |M_c| - |M_l| - \alpha_{jl}$.

3d: If $\beta_{jc^*}\,\Gamma - \alpha_{jc^*} > 0$ (i.e., switching part j to family c^* increases Γ) then

i. $P_{c^*} \leftarrow P_{c^*} + j$, [Optional] ii. $P_c \leftarrow P_c - j$, [Optional]
iii. $y_j \leftarrow c^*$, iv. $\Gamma \leftarrow \frac{e-e_o-\alpha_{jc^*}}{e+e_v-\beta_{jc^*}}$

Step 4: Return Γ and $(x_1, x_2, \ldots, x_m, y_1, y_2, \ldots, y_n)$.

Rule 2 is true.

$$\text{If } \beta_{jl}\,\Gamma \;>\; \alpha_{jl}, \text{ then switch part } j \text{ from family } c \text{ to family } l. \qquad \text{(Rule 2)}$$

Algorithm 2 is a steepest descent algorithm since, in Steps 2c and 3c, the switch which yields the largest increase is used.

Given an initial machine cell and part family assignment, the first step of the algorithm is to determine the grouping efficacy measure as defined by Equation 1. From this calculation, the size and composition of each machine cell M_l and part family P_l is determined. Steps 2 and 3 perform the actual machine and part switching for all machines and parts, respectively. The order in which these steps are performed is not fixed. However, Step 3 (part switching) is dependent upon the current machine allocation that may be changed in Step 2 (likewise for machine switching if the order is reversed). However, it has been shown that forming machine cells first is typically better than forming families first [29, 33]. Because this is a local improvement procedure, an alternating approach of repeating Steps 2 and 3 could also be performed (at a computational cost) until no new improvement is found.

Step 2a determines p_{il}, the number of parts assigned to cell l that requires processing by machine i. This is used to determine the increase in the number of exceptional elements and the decrease in the number of voids in the diagonal blocks. The increase in the number of exceptional elements, α_{il}, is equal to the number of parts requiring processing in the current cell assignment c minus the number if machine i is switched to cell l. The decrease in the number of voids, β_{il}, is equal to the number of parts assigned to family c minus both the number of parts assigned to family l and α_{il}, the increase in the number of exceptional elements. Step 2c determines the best cell, c^*, to switch machine i from its current cell assignment, c. If the increase in Γ is positive, then in Step 2d machine i is switched from cell c to c^* and Γ and the number of machines in the two cells effected, M_c and M_{c^*}, are updated. Step 3 is identical to Step 2 except parts are switched instead of machines. In Step 3d i and ii, the number of parts in each family does not need to be updated in a single pass of the procedure.

5 Experimentation and Analysis

To investigate the trade-off of disrupted schema processing in Lamarckian learning and of multiple genotype mapping to the same phenotype in Baldwinian learning, a series of experiments using LIPs as evaluation functions and as genetic operators were performed on several different cell formation test problem instances. For each test problem instance, the

Table 2: Parameters Used in the Integer-Based Experiments

Parameter	Value
No. of Boundary Mutation Operators	4
No. of Uniform Mutation Operators	4
No. of Multi-Uniform Mutation Operators	4
No. of Non-Uniform Mutation Operators	4
No. of Multi-Non-Uniform Mutation Operators	8
No. of Local Heuristic Mutation Operators	0 or f
No. of Cell Swap Crossover Operators	6
No. of Cell Multi-point Crossover Operators	6
No. of Arithmetic Crossover Operators	6
$k_{\max}$, the maximum permissible no. of cells	†
q, the probability of selecting the best individual	0.08
Maximum no. of Function Evaluations	1 000 000
Population Size	80

† - Problem Specific

GA was run with varying levels of Lamarckianism from 0% (pure Baldwinian) to 100% (pure Lamarckian) to determine if there was a trend between the two extremes, or if combinations of Baldwinian learning and Lamarckian learning were beneficial. In these experiments, individuals were updated to match the resulting phenotype with a probability of 0, 5, 10, 20, 30, 40, 50, 60, 80, 90, 95, 100%. The GA was also run using the LIP as a mutation operator instead to determine if this method of hybridization was better in terms of computational efficiency and/or quality of solution. In these experiments, the number of LIP mutations performed was 3, 4, 5, or 6. A pure genetic approach, i.e., no local improvement was used for comparison purposes.

Each run of the GA was replicated 20 times, with common random seeds. The genetic algorithm parameters used in these experiments are defined in Table 2. The GA was terminated when either the optimal or best known solution was found or after one million function evaluations were performed. Using function evaluations allows a direct comparison between the hybrid GA methods and the pure GA.

Five different test problems were used during the experiments. The first three problems are common data sets found in the literature. The first data set, a 20 machine by 35 part problem (Burb), was taken from Burbidge [1]. The next data set, a 16 machine by 43 part problem (King) taken from King and Nakornchai [25], is one of the most commonly used data sets in the literature. The third data set, a 40 machine by 100 part problem (Chan) from Chandrasekharan and Rajagopalan [5], is one of the largest data sets in the literature.

The last two problems are real industry data sets used to test the ability of these methods to solve realistically sized problems. The first industry data set is a 20 machine by 148 part problem (Real1) from a rotary union factory. The last data set, a 115 machine by 2557 part problem (Real2), is taken from a high-end furniture case-goods plant. Also, for each of these problems, three different problem-specified values of $k_{\max}$ were used to determine if varying values influenced any of the methods in terms of convergence to the optimal. Several more problems were used as in the original experiments [21], but since their results are similar, they are not shown.

6 Results

For each problem instance (i.e., each test problem at a given $k_{\max}$ value), both the mean number of function evaluations and the final function value were examined using analysis of variance (ANOVA). Since the ANOVA shows a significant effect at an $\alpha > 0.0001$ for all fifteen problem instances in terms of function evaluations and final function value, the means of each of the 16 different search strategies considered (no local improvement, referred to as N; pure Baldwinian, referred to as 0; pure Lamarckian, 100; and the nine levels of partial Lamarckian, referred to as 5, 10, 20, 40, 50, 60, 80, 90, and 95, respectively), and the 4 levels of LIP mutation operator (referred to as –3, –4, –5, and –6) were compared using three statistical multiple means comparison tests. These methods are the Duncan approach to minimize the Bayesian loss function [10], Student-Newman-Keuls (SNK), which is a multiple range test using a multiple stage approach, and an approach developed by Ryan [37], Einot and Gabriel [11] and Welsch [39] (REGWF) which is also a multiple stage approach which controls the maximum experiment-wise error rate under any complete or partial hypothesis. Each of the tests were conducted using the general linear models routine in SAS v6.09 [19]. All three of these tests were used since there is no consensus test for multiple means comparison.

The ranking of both the number of function evaluations and the ranking of the final fitness are presented in Tables 6–10 of Appendices Appendix B:– Appendix F: for each of the 15 problem instances considered in this paper: the five test problems (Burb, King, Chan, Real1, Real2), each at three different values of $k_{\max}$, where, e.g., Burb at $k_{\max} = 4$ is referred to as instance Burb-4. Each of the three multiple means comparisons were performed on each instance. In the table, if all three of the multiple means comparison methods do not agree, the results of each are presented; otherwise, the single result for all of the methods is presented.

Tables 3 and 4 concisely present a ranking of each search strategy based upon the statis-

Table 3: Summary Fitness Value (SNK)

Problem Instance	N	−3	−4	−5	−6	0	5	10	20	40	50	60	80	90	95	100
Burb-4	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Burb-5	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Burb-6	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
King-4	3	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1
King-5	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
King-6	1	1	2	1	1	1	1	1	1	1	1	1	1	1	1	1
Chan-10	1	2	1	1	1	2	1	1	1	1	1	1	1	1	1	1
Chan-11	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Chan-12	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Real1-4	3	3	2	3	3	4	1	1	1	1	1	1	1	1	1	1
Real1-5	2	2	2	2	2	3	1	1	1	1	1	1	1	1	1	1
Real1-6	2	2	2	2	2	3	1	1	1	1	1	1	1	1	1	1
Real2-6	4	6	5	4	4	7	3	2	1	1	1	1	1	1	1	1
Real2-7	2	2	2	3	1	4	4	1	5	1	1	1	1	1	1	1
Real2-8	5	2	1	3	2	4	2	1	1	1	1	1	1	1	1	1

tical tests shown in detail in Tables 6–10 in the appendices. This ranking was constructed for both the fitness of final result and number of function evaluations required. The rankings were determined as follows: find the group which yielded the best results for each test problem instance, as determined by the SNK means analysis, i.e., if the strategy was placed into a single group, that group's rank was used; however, if the method was placed into several overlapping groups, the method was placed into the group with the best rank. Therefore, these rankings represent how many groups of means are significantly better for each test problem.

Table 3 shows the results of these rankings for the final fitness of the solutions returned by each of the search strategies, where 1 represents the best rank and is shaded. All of the strategies employing at least a 40% Lamarckian learning were better in terms of final fitness value. Interestingly, pure Baldwinian learning (0) did find significantly worst solutions for several test problem instances. In terms of rank, the LIP mutation methods were in the best group for almost all the literature data sets but were significantly worse when applied to the real data sets; however, they did fair better than pure Baldwinian Learning.

Table 4 shows the results of the rankings for the number of function evaluations required to locate the final functional value returned by the search strategy. This table demonstrates that for the literature data sets any level of Lamarckian learning consistently yields the quickest convergence. This is due in part to the fact that the literature data sets were

Table 4: Summary of Convergence (SNK)

Problem Instance	N	−3	−4	−5	−6	0	5	10	20	40	50	60	80	90	95	100
Burb-4	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Burb-5	3	1	1	1	1	2	1	1	1	1	1	1	1	1	1	1
Burb-6	3	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1
King-4	4	3	2	2	2	2	1	1	1	1	1	1	1	1	1	1
King-5	3	2	1	1	1	2	1	1	1	1	1	1	1	1	1	1
King-6	2	3	3	2	2	2	1	1	1	1	1	1	1	1	1	1
Chan-10	2	3	2	1	2	4	1	1	1	1	1	1	1	1	1	1
Chan-11	3	1	1	1	1	2	1	1	1	1	1	1	1	1	1	1
Chan-12	4	1	2	1	1	3	1	1	1	1	1	1	1	1	1	1
Real1-4	3	3	2	3	3	3	1	1	1	1	1	1	1	1	1	1
Real1-5	4	4	4	4	3	4	2	2	2	2	2	1	1	2	1	1
Real1-6	4	4	4	4	3	4	2	1	1	1	1	1	1	1	1	1
Real2-6	3	3	3	3	3	3	3	2	2	2	1	1	1	1	1	1
Real2-7	3	3	3	3	3	3	3	2	3	2	2	1	1	1	1	1
Real2-8	4	4	3	4	3	4	3	3	3	2	1	1	1	1	1	1

extremely easy to solve when employing the LIP. However, for the real data sets, a level of greater than 60% Lamarckian is required, with exception of 90% for Real1-5 problem instance (note, in this case, 80% was actually better than 100% but it was not significantly better). There were several problem instances were pure Lamarckian was not ranked the best.

The average computation time for 50 replications needed to initialize the population and run 1000 generations of the pure GA and the Hybrid-GA using the LIP as an evaluation function is shown in Table 5. The table shows that there is no statistically significant difference (at 95% confidence) between the various values of $k_{\max}$ for a given problem. Therefore, increasing $k_{\max}$ does not greatly increase the computational effort of the LIP.

7 Conclusions

Techniques for partitioning the part/machine incidence matrix into machine cells and associated part families are varied and extensive. Optimization techniques are limited in practice to problems of small scale. Most heuristic methods lack the flexibility to form cells using a variety of evaluation measures or handle constraints adequately. The hybrid-GA approach developed in this paper overcomes these limitations.

It was demonstrated that GAs are very efficient at exploring the entire search space;

Problem	k_{max}	GA		Hybrid GA	
		Initial†	Generation‡	Initial†	Generation‡
Burb	4	0.0260	10.0980	0.0850	25.5170
	5	0.0260	10.0950	0.0900	26.6790
	6	0.0270	10.0900	0.0960	28.5180
King	4	0.0272	10.6848	0.0864	29.9856
	5	0.0274	10.6080	0.0918	27.1724
	6	0.0278	10.6554	0.0990	29.3078
Chan	10	0.0724	23.2222	0.3892	109.4566
	11	0.0744	23.1994	0.3962	111.0308
	12	0.0714	23.1604	0.4080	113.6542

†-time to initialize population ‡-time to run 1000 generations

Table 5: Computational Time of the GA versus the Hybrid GA in (CPU seconds)

however, they are relatively poor at finding the precise local optimal solution in the region at which the algorithm converges. Hybrid-GAs are the combination of improvement procedures, usually working as evaluation functions, and genetic algorithms. Using LIPs was shown to enhance the performance of the genetic algorithm. Even though Lamarckian learning disrupts the schema processing of the genetic algorithm, it reduces the problem of a one-to-one genotype to phenotype mapping. Whereas Baldwinian learning while not affecting the schema processing capabilities of the genetic algorithm, results in a large number of genotypes mapping to the same phenotype.

In the empirical investigation, a general trend was observed: increasing use of Lamarckian learning led to the quicker convergence of the genetic algorithm to the best known solution as compared to both Baldwinian learning and using the LIP as a mutation operator. By forcing the genotype to reflect the phenotype, the GA converges more quickly and to better solutions than by leaving the chromosome unchanged after evaluating it. This may seem counterintuitive since forcing the genotype to be equal to the phenotype might have forced the GA to converge prematurely to a local optimum. For the cell formation problem, a higher percentage of Lamarckian learning gives the best mix of computational efficiency and solution quality.

One possible reason for the poor performance of pure Baldwinian learning is that the multiple genotype to phenotype mapping causes the GA to waste a lot of precious evaluations in the same basin. For the larger problems (i.e., Chan, Real1 and Real2 problems), a large number of the functional evaluations are used to generate the starting population. Therefore, Baldwinian Learning does not have a lot of generations to overcome this confounding problem and it never reaches the optimal for many replications of these problems. Even though

Lamarckian is not natural (i.e., "Darwinian") evolution, it seems to be ideal for real-world problem solving using evolutionary computation.

When employing the LIP, the problems from the literature turned out to be very easy. The reason may be that the fitness landscape is transformed into very flat regions around the local minima, making it easier for the GA to search. For these instances, a multi-start procedure utilizing the same local improvement would have been just as effective. However, for the real problems, this would not be case since it takes the GA a number of generations to reach the best known solution. Another interesting observation is that as the problem size increased for the non-binding cases, (Burb-6, Chan-11, Chan-12), the hybrid-GA methods had quicker convergence to the optimal than the non-binding cases,(Burb-5 and Chan-10). The non-binding cases represent when the upper bound on the number of cells is greater than the number natural occurring cells. This may be due in part to the fact that the local improvement procedure has more room to maneuver in the non-binding cases.

The ability to solve the largest problems in a matter of seconds will allow the cell designer to solve more complex versions of the cell formation problem. Even though grouping efficacy was used as the evaluation function, the concepts developed in this paper can be easily extended to other LIPs. The grouping efficacy LIP has been modified to work on extended models which includes alternative operations, machine redundancy, alternative routings and part demand [23].

References

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

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

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

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

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

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

[7] P.C. Chu and J.E. Beasley. A genetic algorithm for the generalised assignment problem. Technical report, The Management School Imperial College London, 1995.

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

[9] L Davis and D. Orvosh. Using a genetic algorithm to optimze problems with feasibility constraints. In *1994 IEEE International Symposium Evolutionary Computation*, pages 548–553, Orlando, Fl, 1994.

[10] D Duncan. t-tests and intervals for comparisons suggested by the data. *Biometrics*, 31:339–359, 1975.

[11] I. Einot and K. Gabriel. A study of the powers of several methods of multiple comparisons. *Journal of the American statistical association*, 70:351, 1975.

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

[13] F. Gruau and D. Whitley. Adding learning to the cellular development of neural networks. *Evolutionary Computation*, 1:213–233, 1993.

[14] G.E. Hinton and S.J. Nolan. How learning can guide evolution. *Complex Systems*, 1:495–502, 1987.

[15] J.H. Holland. *Adaptation in natural and artificial systems.* The University of Michigan Press, Ann Arbor, 2nd edition, 1992.

[16] C. R. Houck, J. A. Joines, and M. G. Kay. Utilizing lamarckian evolution and the baldwin effect in hybrid genetic algorithms. Technical Report NCSU-IE Technical Report 96-01, North Carolina State University, 1996.

[17] C.R. Houck, J.A. Joines, and M.G. Kay. Comparison of genetic algorithms, random restart, and two-opt switching for solving large location-allocation problems. *Computers & Operations Research*, 23(6):587–596, 1996.

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

[19] SAS Institute Inc. *SAS/STAT User's Guide.* Cary, NC, 4th edition, 1990.

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

[21] J.A. Joines, C.T. Culbreth, and R.E. King. Manufacturing cell design: An integer programming model employing genetic. *IIE Transactions*, 28(1):69–85, 1996.

[22] 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.

[23] J.A. Joines, R.E. King, and C.T. Culbreth. A genetic algorithm based integer program for manufacturing cell design. In *International Conference on Flexible Automation and Integrated Manufacturing*, Montreal, Canada, 1995.

[24] J.A. Joines, R.E. King, and C.T. Culbreth. A comprehensive review of production-oriented manufacturing cell formation techniques. *International Journal of Flexible Automation and Integrated Manufacturing*, 3(3&4):161–200, 1996.

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

[26] 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.

[27] A. Kusiak. *Intelligent Design and Manufacturing.* John Wiley and Sons, New York, 1992.

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

[29] C. Liu and J. Wu. Machine cell formation: using the simulated annealing algorithm. *International Journal of Computer Integrated Manufacturing*, 6(6):335–349, 1993.

[30] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* AI Series. Springer-Verlag, New York, 3rd edition, 1996.

[31] Z. Michalewicz and G. Nazhiyath. Genocop iii: A co-evolutionary algorithm for numerical optimization problems with non-linear constraints. In *The Second IEEE Conference on Evolutionary Computation*, pages 647–651, Perth, Australia, December 1995.

[32] R. Nakano. Conventional genetic algorithm for job shop problems. In *Proc. of the 4th ICGA*, pages 474–479, San Mateo, CA, 1991.

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

[34] J-M. Renders and H. Bersini. Hybridizing genetic algorithms with hill-climbing methods for global optimization: Two possible ways. In *The First IEEE Conference on Evolutionary Computation*, 1994.

[35] J.-M. Renders and S. Flasse. Hybrid methods using genetic algorithms for global optimization. *IEEE Transactions on Systems, Man, and Cybernetics Part B:*, 26(2):243–258, April 1996.

[36] A. Rinnooy Kay and G. Timmer. Stochastic global optimization methods part i. *Mathematical Programming*, 39(1):27–56, 1987.

[37] T. Ryan. Significance tests for multiple comparison of proportions, variances, and other statistics. *Psychological Bulletin*, 57:318–328, 1960.

[38] 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.

[39] R. Welsch. Stepwise multiple comparison procedures. *ournal of the American statistical association*, 72:359, 1977.

[40] D. Whitley. A genetic algorithm tutorial. *Statistics and Computing*, 4:65–85, 1994.

[41] D. Whitley, S. Gordon, and K. Mathias. Larmarckian evolution, the Baldwin effect and function opimization. In Y. Davidor, H.P. Schwefel, and R. Manner, editors, *Parallel Problem Solving from Nature-PPSN III*, pages 6–15. Springer-Verlag, 1994.

Appendix

The ranking of both the number of function evaluations and the ranking of the final fitness are presented in Tables 6–10 for each of the 15 problem instances considered in this paper: the five test problems (Burb, King, Chan, Real1, Real2), each at three different values of $k_{\max}$, where, e.g., Burb at $k_{\max} = 4$ is referred to as instance Burb-4. Each of the three multiple means comparisons were performed on each instance.

The multiple means tests provide information on sets of means whose differences are statistically significant. For example, all three multiple means tests agree that the no local improvement (N) method yields significantly worse results for the Burb-4 instance. According to all the tests, any form of hybridization (i.e., learning methods or local improvement mutation operators), yield effectively the same final fitness value. Each of the problems are discussed below; a summary of the results for both speed of convergence and final fitness value is provided in Tables 3 and 4 of Section 6. In the tables, the rank indicates the ranking of the average number of function evaluations or average fitness value, where the ranking is from left (worst) to right (best). For example, for the Burb-4 instance, even though the local mutation hybrid methods (–6, –5, –3, –4) were not significantly better than learning methods, the rank indicates they had the lowest average number of function values.

Appendix B: Burbidge Problem

As seen in Table 6, the local improvement procedure greatly improves the efficiency and quality of solution for this data set. The optimal solution was readily found by all hybridization methods, whereas no local improvement was terminated after one million function evaluations. When the maximum number of cells ($k_{\max}$) equals 4 there was no statistical significant difference in the number of function evaluations required to obtain the best known solution for each of the hybrid methods. However, when $k_{\max}$ increases to 5 and 6, the LIP mutation hybrid methods require a significant greater number of function evaluations to obtain the best known solution. This is not as clear in the Burb-5 instance because some overlap in the groups exists; for this instance, while Baldwinian learning (0) strategy found the optimal, it took significantly longer to converge.

Table 6: Multiple Means Comparison for Burbidge Problem

Problem Instance	Criteria																	
Burb-4	#Evals	Rank	N	0	5	10	80	90	95	60	20	40	50	100	-6	-5	-3	-4
		Duncan	A															
				B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
		SNK	A															
				B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
		REGW	A															
				B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
	Fitness	Rank	N	0	5	10	80	90	95	60	20	40	50	100	-6	-5	-3	-4
		Duncan	A															
				B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
		SNK	A															
				B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
		REGW	A															
				B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
Burb-5	#Evals	Rank	N	0	-3	-4	-5	5	-6	10	20	40	50	60	80	90	95	100
		Duncan	A															
				B	B													
					C	C	C	C	C	C	C	C	C	C	C	C	C	C
		SNK	A															
				B	B	B												
					C	C	C	C	C	C	C	C	C	C	C	C	C	C
		REGW	A															
				B	B	B	B											
					C	C	C	C	C	C	C	C	C	C	C	C	C	C
	Fitness	Rank	N	0	-3	-4	-5	5	-6	10	20	40	50	60	80	90	95	100
		Duncan	A															
				B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
		SNK	A															
				B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
		REGW	A															
				B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
Burb-6	#Evals	Rank	N	-3	-4	-6	-5	0	5	10	20	40	90	50	80	60	95	100
		Duncan	A															
				B	B	B	B											
								C	C	C	C	C	C	C	C	C	C	C
		SNK	A															
				B	B	B	B											
								C	C	C	C	C	C	C	C	C	C	C
		REGW	A															
				B	B	B	B											
								C	C	C	C	C	C	C	C	C	C	C
	Fitness	Rank	N	-3	-4	-6	-5	0	5	10	20	40	90	50	80	60	95	100
		Duncan	A															
				B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
		SNK	A															
				B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
		REGW	A															
				B	B	B	B	B	B	B	B	B	B	B	B	B	B	B

Appendix C: King Problem

This data set contains two large bottleneck machines assuming equal demands for the parts [25]. It is harder than the Burbidge problem. In terms of fitness, the hybrid methods produced significantly better solutions for the three $k_{\max}$ values than was reported by the no local improvement procedure (N), while, for the King-4 instance the learning methods were significantly better than the local mutation methods. In terms of convergence, the results are a little more interesting since any percentage of Lamarckian learning was significantly better than Baldwinian Learning. Also, for the King-5 instance, several of the local mutation

methods were significantly better than pure Baldwinian Learning. See Table 7 for more detail.

Table 7: Multiple Means Comparison for King Problem

Problem Instance	Criteria																	
King-4	#Evals	Rank	N	-3	-5	-6	-4	0	5	95	100	10	20	40	50	60	80	90
		Duncan	A															
				B	B	B	B											
					C	C	C	C										
									D	D	D	D	D	D	D	D	D	D
		SNK	A															
				B	B	B	B											
					C	C	C	C										
									D	D	D	D	D	D	D	D	D	D
		REGW	A	A														
				B	B	B	B	B										
									C	C	C	C	C	C	C	C	C	C
	Fitness	Rank	N	-3	-5	-6	-4	0	5	95	100	10	20	40	50	60	80	90
		Duncan	A															
				B	B	B	B											
								C	C	C	C	C	C	C	C	C	C	C
		SNK	A															
				B	B	B	B											
								C	C	C	C	C	C	C	C	C	C	C
		REGW	A															
				B	B	B	B											
								C	C	C	C	C	C	C	C	C	C	C
King-5	#Evals	Rank	N	0	-3	-5	-4	-6	5	10	20	40	50	60	95	90	100	80
		Duncan	A															
				B	B													
						C	C	C										
							D	D	D	D	D	D	D	D	D	D	D	D
		SNK	A															
				B	B													
						C	C	C	C	C	C	C	C	C	C	C	C	C
		REGW	A															
				B	B													
					C	C												
						D	D	D	D	D	D	D	D	D	D	D	D	D
	Fitness	Rank	N	0	-3	-5	-4	-6	5	10	20	40	50	60	95	90	100	80
		Duncan	A															
				B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
		SNK	A															
				B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
		REGW	A															
				B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
King-6	#Evals	Rank	-4	-3	N	-5	-6	0	5	10	20	40	50	60	95	90	100	80
		Duncan	A	A	A													
				B	B	B	B											
						C	C	C										
									D	D	D	D	D	D	D	D	D	D
		SNK	A	A	A	A	A											
					B	B	B	B										
									C	C	C	C	C	C	C	C	C	C
		REGW	A	A	A	A	A											
					B	B	B	B										
									C	C	C	C	C	C	C	C	C	C
	Fitness	Rank	-4	-3	N	-5	-6	0	5	10	20	40	50	60	95	90	100	80
		Duncan	A															
				B	B	B	B											
					C	C	C	C	C	C	C	C	C	C	C	C	C	C
		SNK	A															
				B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
		REGW	A															
				B	B	B	B	B	B	B	B	B	B	B	B	B	B	B

Appendix D: Chandrasekharan Problem

This problem is the largest data set found in the literature [4]. In terms of final fitness values, the groups are not as well defined as in the previous problems, indicating that several of the replications terminated after one million function evaluations (see Table 8). For all three instances, pure Baldwinian learning (0) was significantly the worst method in terms of convergence: for the Chan-10 instance, it was also in the worst group in terms of fitness; for the Chan-11 instance, it was in the second worst group; and in all of the replications where it did find the optimal, it still took a significantly greater number of function evaluations to converge than the other learning methods. For all three statistical tests, any Lamarckian learning greater than 5% was always in the best groups in terms of final fitness value and convergence. As stated as the problem size increased, the problem seemed to get easier (i.e., more methods found the optimal more quickly). This may be due in part to the fact that there are ten natural occurring clusters and the local improvement procedure has more room to maneuver in the non-binding cases. The same phenomena occurred for the Burb-6 instance. For this instance, the Lamarckian learning methods took only a few generations to solve the problem, indicating that a multi-start procedure using the same local improvement procedure would probably work as well.

Table 8: Multiple Means Comparison for Chandrasekharan Problem

Problem Instance	Criteria																	
Chan-10		Rank	0	-3	-4	-6	N	-5	5	10	20	40	50	80	60	90	100	95
	#Evals	Duncan	A															
				B	B	B	B											
					C	C	C	C										
									D	D	D	D	D	D	D	D	D	D
		SNK	A															
				B	B	B	B											
					C	C	C	C										
								D	D	D	D	D	D	D	D	D	D	D
		REGW	A															
				B	B	B	B	B										
								C	C	C	C	C	C	C	C	C	C	C
	Fitness	Rank	0	-3	-4	-6	N	-5	5	10	20	40	50	80	60	90	100	95
		Duncan	A	A	A	A	A	A										
								B	B	B	B	B	B	B	B	B	B	B
		SNK	A	A	A	A	A	A										
					B	B	B	B	B	B	B	B	B	B	B	B	B	B
		REGW	A	A	A	A	A	A										
					B	B	B	B	B									
							C	C	C	C	C	C	C	C	C	C	C	C
Chan-11		Rank	N	0	-4	-5	-3	-6	5	10	20	40	50	60	80	90	95	100
	#Evals	Duncan	A															
				B	B	B	B											
					C	C	C	C										
								D	D	D	D	D	D	D	D	D	D	D
		SNK	A															
				B	B	B	B											
					C	C	C	C	C	C	C	C	C	C	C	C	C	C
		REGW	A															
				B	B	B	B	B										
							C	C	C	C	C	C	C	C	C	C	C	C
	Fitness	Rank	N	0	-4	-5	-3	-6	5	10	20	40	50	60	80	90	95	100
		Duncan	A	A	A													
				B	B	B	B											
						C	C	C	C	C	C	C	C	C	C	C	C	C
		SNK	A	A	A	A												
				B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
		REGW	A	A	A	A	A											
					B	B	B	B	B	B	B	B	B	B	B	B	B	B
Chan-12		Rank	N	0	-4	-6	-3	-5	5	10	20	40	50	60	80	90	95	100
	#Evals	Duncan	A															
				B														
					C													
						D	D	D	D	D	D	D	D	D	D	D	D	D
		SNK	A															
				B														
					C													
						D	D	D	D	D	D	D	D	D	D	D	D	D
		REGW	A															
				B														
					C	C	C	C										
						D	D	D	D	D	D	D	D	D	D	D	D	D
	Fitness	Rank	N	0	-4	-6	-3	-5	5	10	20	40	50	60	80	90	95	100
		Duncan	A															
				B	B	B	B	B										
					C	C	C	C	C	C	C	C	C	C	C	C	C	C
		SNK	A															
				B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
		REGW	A															
				B	B	B	B	B	B	B	B	B	B	B	B	B	B	B

Appendix E: Real Data Set 1

For this problem, pure Baldwinian learning was significantly worst in terms of final fitness value as compared to all the other methods including no local improvement. In terms of the mean fitness values shown in Table 9, any form of Lamarckian learning was consistently better than no local improvement and the local mutation hybrids. However, in terms of convergence, it is not nearly as clear since there is considerable overlap for all three instances for all three tests. This is due in part to the fact that none of the hybrid methods found

the best solution in all 20 replications (e.g., 80% Lamarckian learning found the optimal 15 out 20 times). For the Real1-4 instance, Baldwinian learning, no local improvement, and the local mutation hybrid methods were significantly worse than any Lamarckian learning method. For the Real1-5 instance, 80% Lamarckian learning was the best method in terms of mean value of function evaluations; however, 60, 95, and 100% overlapped with this level. Also, for this instance, there does not seem to be any general trend indicating that increasing the level of Lamarckian learning increases the convergence rate.

Table 9: Multiple Means Comparison for Real Data Set 1

Problem Instance	Criteria																	
Real1-4	#Evals	Rank	0	-5	-3	N	-6	-4	20	5	40	80	50	10	90	60	95	100
		Duncan	A	A	A	A	A	A										
									B	B	B	B	B	B	B	B	B	
										C	C	C	C	C	C	C	C	C
		SNK	A	A	A	A	A	A										
								B	B	B	B	B	B					
									C	C	C	C	C	C	C	C	C	C
		REGW	A	A	A	A	A	A										
								B	B	B	B	B	B	B	B			
									C	C	C	C	C	C	C	C	C	C
	Fitness	Rank	0	-5	-3	N	-6	-4	20	5	40	80	50	10	90	60	95	100
		Duncan	A															
				B	B	B	B	B										
									C	C	C	C	C	C	C	C	C	C
		SNK	A															
				B	B	B	B	B										
								C	C	C	C	C	C	C	C	C	C	
									D	D	D	D	D	D	D	D	D	D
		REGW	A															
				B	B	B	B	B										
							C	C	C									
								D	D	D	D	D	D	D	D	D	D	D
Real1-5	#Evals	Rank	0	-5	-3	N	-4	-6	20	90	40	5	50	10	60	95	100	80
		Duncan	A	A	A	A	A	A	A	A	A	A						
						B	B	B	B	B	B	B	B	B				
										C	C	C	C	C	C	C	C	
															D	D	D	D
		SNK	A	A	A	A	A	A	A	A	A	A	A	A				
								B	B	B	B	B	B	B	B			
									C	C	C	C	C	C	C	C	C	
															D	D	D	D
		REGW	A	A	A	A	A	A	A	A	A	A	A					
				B	B	B	B	B	B	B	B	B	B	B				
							C	C	C	C	C	C	C	C	C			
									D	D	D	D	D	D	D	D		
										E	E	E	E	E	E	E	E	
														F	F	F	F	F
	Fitness	Rank	0	-5	-3	N	-4	-6	20	90	40	5	50	10	60	95	100	80
		Duncan	A															
				B	B	B	B	B										
									C	C	C	C	C	C	C	C	C	C
		SNK	A															
				B	B	B	B	B										
									C	C	C	C	C	C	C	C	C	C
		REGW	A															
				B	B	B	B	B										
									C	C	C	C	C	C	C	C	C	C
Real1-6	#Evals	Rank	0	-5	-4	N	-3	-6	5	10	20	60	40	50	95	90	80	100
		Duncan	A	A	A	A	A	A	A	A								
								B	B	B	B	B						
									C	C	C	C	C	C				
										D	D	D	D	D	D			
												E	E	E	E	E	E	E
		SNK	A	A	A	A	A	A	A	A	A	A	A					
								B	B	B	B	B	B	B				
									C	C	C	C	C	C	C			
										D	D	D	D	D	D	D	D	D
		REGW	A	A	A	A	A	A	A	A	A							
							B	B	B	B	B	B	B					
								C	C	C	C	C	C	C				
									D	D	D	D	D	D	D			
										E	E	E	E	E	E	E	E	
											F	F	F	F	F	F	F	F
	Fitness	Rank	0	-5	-4	N	-3	-6	5	10	20	60	40	50	95	90	80	100
		Duncan	A															
				B	B	B	B	B										
									C	C	C	C	C	C	C	C	C	C
		SNK	A															
				B	B	B	B	B										
									C	C	C	C	C	C	C	C	C	C
		REGW	A															
				B	B	B	B	B										
							C	C	C									
								D	D	D	D							
									E	E	E	E	E	E	E	E	E	E

Appendix F: Real Data Set 2

This is the largest of the problems considered. In the first pass, using one million function evaluations, none of the methods were able to find the best known solution for any of the replications. This is due in part to the fact that when employing local improvements, almost one million function evaluations were used to initialize the starting population. This illustrates a case where a multi-start procedure would not be as effective on this problem. Therefore, the maximum number of evaluations was increased to 10 million. Table 10 presents these results. Even in this case, none of the methods were able to find the best known solution in all 20 replications (e.g., 100% Lamarckian found the the best known solution in 10 of the 20 replications for the Real2-6 instance). What is interesting in these results is that pure Baldwinian as well as 5% Lamarckian did not find the best known solutions. For the Real2-7 instance, 20% Lamarckian learning produced significantly the worst solution, with Baldwinian and 5% Lamarckianism forming the second worst group. As indicated in the table, up to a level of approximately 40, 50, and 40% for the Real2-6, -7, and -8 instances, respectively, the method terminated after 10 million function evaluations. Even pure Lamarckian (100) was not clearly the best in terms of speed convergence. This is not statistically significant because of the overlap of the groups.

Table 10: Multiple Means Comparison for Burbidge Problem

Problem Instance	Criteria																	
Real2-6		Rank	0	-3	-4	-5	-6	N	5	10	20	40	80	50	60	90	100	95
	#Evals	Duncan	A	A	A	A	A	A	A	A	A	A						
										B	B	B	B	B	B			
													C	C	C	C	C	
															D	D	D	D
		SNK	A	A	A	A	A	A	A	A	A	A	A	A				
										B	B	B	B	B	B	B	B	
													C	C	C	C	C	C
		REGW	A	A	A	A	A	A	A	A	A	A						
						B	B	B	B	B	B	B	B					
								C	C	C	C	C	C	C				
									D	D	D	D	D	D	D			
										E	E	E	E	E	E	E		
											F	F	F	F	F	F	F	
													G	G	G	G	G	G
		Rank	0	-3	-4	-5	-6	N	5	10	20	40	80	50	60	90	100	95
	Fitness	Duncan	A															
				B														
					C	C												
						D	D											
							E	E										
									F	F	F							
											G	G	G	G	G	G	G	
												H	H	H	H	H	H	H
		SNK	A															
				B														
					C	C	C											
						D	D	D										
									E	E	E							
										F	F	F	F	F	F	F	F	
											G	G	G	G	G	G	G	G
		REGW	A															
				B														
					C	C	C											
						D	D	D										
									E	E	E	E						
										F	F	F	F	F	F	F	F	
											G	G	G	G	G	G	G	G

continued on the next page

continued from the previous page

Problem Instance	Criteria																	
Real2-7	#Evals	Rank	20	5	0	-5	N	-4	-3	-6	10	40	50	60	95	100	80	90
		Duncan	A	A	A	A	A	A	A	A	A	A	A					
											B	B	B	B	B			
												C	C	C	C	C	C	
																D	D	D
		SNK	A	A	A	A	A	A	A	A	A	A	A	A	A			
											B	B	B	B	B	B	B	
														C	C	C	C	C
		REGW	A	A	A	A	A	A	A	A	A	A	A					
							B	B	B	B	B	B	B	B				
									C	C	C	C	C	C	C			
										D	D	D	D	D	D	D		
											E	E	E	E	E	E	E	
													F	F	F	F	F	F
	Fitness	Rank	20	5	0	-5	N	-4	-3	-6	10	40	50	60	95	100	80	90
		Duncan	A															
				B	B													
						C	C	C										
							D	D	D									
										E	E	E	E	E	E			
											F	F	F	F	F	F	F	F
		SNK	A															
				B	B													
						C	C	C										
							D	D	D									
										E	E	E	E	E	E	E	E	E
		REGW	A															
				B	B													
						C	C	C	C									
										D	D	D	D	D	D	D	D	D
Real2-8	#Evals	Rank	N	0	-5	-3	5	-6	-4	10	20	40	50	60	80	90	95	100
		Duncan	A	A	A	A	A	A	A	A	A							
							B	B	B	B	B	B						
										C	C	C	C					
													D	D	D	D	D	
														E	E	E	E	E
		SNK	A	A	A	A	A	A	A	A	A	A						
							B	B	B	B	B	B	B					
												C	C	C				
													D	D	D	D	D	D
		REGW	A	A	A	A	A	A	A	A	A	A						
							B	B	B	B	B	B	B					
											C	C	C	C				
												D	D	D	D			
													E	E	E	E	E	E
	Fitness	Rank	N	0	-5	-3	5	-6	-4	10	20	40	50	60	80	90	95	100
		Duncan	A															
				B														
					C	C												
						D	D	D										
									E	E	E							
										F	F	F	F	F	F	F	F	F
		SNK	A															
				B														
					C	C												
						D	D	D										
									E	E	E	E	E	E	E	E	E	E
		REGW	A															
				B														
					C	C	C											
						D	D	D										
									E	E	E	E	E	E	E	E		
										F	F	F	F	F	F	F	F	F