IIE Transactions

ISSN: 0740-817X (Print) 1545-8830 (Online) Journal homepage: https://www.tandfonline.com/loi/uiie20

An exact algorithm for maximizing grouping efficacy in part–machine clustering

Michael J. Brusco

To cite this article: Michael J. Brusco (2015) An exact algorithm for maximizing grouping efficacy in part–machine clustering, IIE Transactions, 47:6, 653-671, DOI: 10.1080/0740817X.2014.971202

To link to this article: https://doi.org/10.1080/0740817X.2014.971202

Published online: 12 Feb 2015.

IIE Transactions (2015) **47**, 653–671

ISSN: 0740-817X print / 1545-8830 online
DOI: 10.1080/0740817X.2014.971202

An exact algorithm for maximizing grouping efficacy in part–machine clustering

MICHAEL J. BRUSCO*

Florida State University, College of Business,, Tallahassee, FL 32306-1110, USA
E-mail: mbrusco@cob.fsu.edu

Received January 2014 and accepted September 2014

The Grouping Efficacy Index (GEI) is well-recognized as a measure of the quality of a solution to a part–machine clustering problem. During the past two decades, numerous approximation procedures (heuristics and metaheuristics) have been proposed for maximization of the GEI. Although the development of effective approximation procedures is essential for large part–machine incidence matrices, the design of computationally feasible exact algorithms for modestly sized matrices also affords an important contribution. This article presents an exact (branch-and-bound) algorithm for maximization of the GEI. Among the important features of the algorithm are (i) the use of a relocation heuristic to establish a good lower bound for the GEI; (ii) a careful reordering of the parts and machines; and (iii) the establishment of upper bounds using the minimum possible contributions to the number of exceptional elements and voids for yet unassigned parts and machines. The scalability of the algorithm is limited by the number of parts and machines, as well as the inherent structure of the part–machine incidence matrix. Nevertheless, the proposed method produced globally optimal solutions for 104 test problems spanning 31 matrices from the literature, many of which are of nontrivial size. The new algorithm also compares favorably to a mixed-integer linear programming approach to the problem using CPLEX.

Keywords: Part–machine clustering, group technology, cellular manufacturing, grouping efficacy index, exact algorithm

1. Introduction

Group Technology (GT) is a strategy for product and process design that facilitates manufacturing practice for a diverse product line. The basic concepts of GT and its components emerged from the pioneering work of Burbidge (1963) and Mitrofanov (1966). Among the important components of GT is cellular manufacturing, which requires the establishment of families of parts and machines to form manufacturing (or production) cells. A manufacturing cell should consist of a subset of parts and a subset of machines, such that the parts in the cell are similar in the sense that they tend to require processing on the same subset of machines. Thus, an important initial step in cellular manufacturing is the clustering of parts and machines to identify the manufacturing cells. This process of constructing cells, which is known as the cell formation problem, has been studied extensively during the past several decades (see, for example, McAuley (1972); King (1980); Heragu and Kakuturi (1997); Selim *et al*. (1998); Papaioannou and Wilson (2010); Ghosh *et al*. (2011); Askin (2013).

A key input to a cell formation problem is commonly a binary incidence matrix, $\mathbf{A} = [a_{ij}]$, whereby $a_{ij} = 1$ if part i requires production on machine j and $a_{ij} = 0$ otherwise. One approach to the formation of cells is to permute the rows and columns of $\mathbf{A}$ with the goal of inducing a block-diagonal form in the reordered matrix (McCormick *et al*., 1972; Askin *et al*., 1991; Shargal *et al*., 1995; Sarker and Khan, 2001; Brusco and Steinley, 2007). An inherent limitation of permutation methods is that they do not explicitly place parts and machines into cells and, accordingly, this must be accomplished either visually or via some type of post-processing method.

An alternative approach to cell formation is the direct clustering of the part–machine matrix. At its fundamental level, the process is one of two-mode partitioning (van Rosmalen *et al*., 2009), which is also known as co-clustering or biclustering (Madeira and Oliveira, 2004; Wilderjans *et al*., 2013) in some research domains. The parts and machines are the two modes of data, and a partition of the objects in each of these modes is required. Over the past few decades, numerous objective criteria and partitioning methods have been proposed for the formation of cells. Most objective criteria are functions of parts not requiring processing on machines in their cell (*voids*) and parts requiring processing on machines that are in different cells (*exceptional elements*). The *structure* of a part/machine clustering matrix reflects the extent to which it is possible to partition the parts and machines to produce few

*Corresponding author

voids and exceptional elements. Reviews of objective criteria (grouping efficiency/efficacy indices) are provided by Sarker and Mondal (1999) and Sarker (2001). Methods for manufacturing cell formation include mathematical programming models (Shtub, 1989; Srinivasan *et al*., 1990; Ventura *et al*., 1990; Crama and Oosten, 1996), traditional similarity-based clustering heuristics (Yin and Yasuda, 2006), simulated annealing (Adil *et al*., 1997; Wu *et al*., 2008), genetic/evolutionary algorithms (Gonçalves and Resende, 2004), tabu search (Lei and Wu, 2005), ant colony methods (Li *et al*., 2010), neural networks (Yang and Yang, 2008), and hybrid heuristics incorporating variable neighborhood search (Paydar and Saidi-Mehrabad, 2013). A review of implementations of different metaheuristics is provided by Ghosh *et al*. (2011).

One of the most popular objective criteria for the cell formation problem is the Grouping Efficacy Index (GEI) developed by Kumar and Chandrasekharan (1990). The desirable properties of the GEI, which have been observed in the literature (Gonçalves and Resende, 2004; Paydar and Saidi-Mehrabad, 2013) include (i) its ability to capture both intra-cell (voids) and inter-cell (exceptional elements) deviations from ideal block-diagonal form; (ii) its ability to differentiate between well-structured and ill-structured part–machine matrices; (iii) its avoidance of weighting parameters; and (iv) the fact that its value is not necessarily a monotonically increasing function of the number of cells. The GEI has also been criticized with respect to the fact that it places differential weights on voids and exceptional elements that vary as such deviations increase (Nair and Narendran, 1996; Sarkar and Mondal, 1999). Its strengths and limitations notwithstanding, the GEI has been the most popular index in recent comparisons of methods for cell formation problems (see, for example, Table 1 of Paydar and Saidi-Mehrabad, 2013). Therefore, throughout the remainder of this article, we focus on the GEI as the criterion of choice.

Crama and Oosten (1996) presented a nonlinear mathematical programming formulation for maximizing GEI. They observed that the special structure of the objective function can enable re-formulation as a sequence of zero-one integer programming problems using a fractional programming method developed by Dinkelbach (1967). Nevertheless, the computational demands of this approach are substantial, and tremendous care is needed for the design of valid inequalities to facilitate efficient implementation of the method (see also Elbenani and Ferland (2012); Bychkov *et al*. (2014). In light of the considerable challenges associated with the maximization of the GEI, numerous metaheuristics have been proposed in recent years, including, genetic algorithms (Tariq *et al*., 2009), simulated annealing (Wu *et al*., 2008), ant colony methods (Li *et al*., 2010), particle swarm optimization (Anvari *et al*., 2010), water flow algorithms (Wu *et al*., 2010), and evolutionary algorithms with embedded variable neighborhood search (Paydar and Saidi-Mehrabad, 2013).

Table 1. A complete search tree for a small example

		Cluster memberships for objects 1 to 5					
Stage	*q*	*1*	*2*	*3*	*4*	*5*	*Sequence of steps*
0	0						1-2
1	1	1					2-3-4-5-2
2	2	1	1				2-3-4-5-2
3	3	1	1	1			2-3-4-5-2
4	4	1	1	1	1		2-3-4-5-2
5	5	1	1	1	1	1	2-3-6-7
6	**5**	**1**	**1**	**1**	**1**	**2**	**7-3-4-5-6-8-6-7**
7	4	1	1	1	2		7-3-4-5-2
8	**5**	**1**	**1**	**1**	**2**	**1**	**2-3-4-5-6-7**
9	**5**	**1**	**1**	**1**	**2**	**2**	**7-3-4-5-6-8-6-8-6-7**
10	3	1	1	2			7-3-4-5-2
11	4	1	1	2	1		2-3-4-5-2
12	**5**	**1**	**1**	**2**	**1**	**1**	**2-3-4-5-6-7**
13	**5**	**1**	**1**	**2**	**1**	**2**	**7-3-4-5-6-8-6-7**
14	4	1	1	2	2		7-3-4-5-2
15	**5**	**1**	**1**	**2**	**2**	**1**	**2-3-4-5-6-7**
16	**5**	**1**	**1**	**2**	**2**	**2**	**7-3-4-5-6-8-6-8-6-8-6-7**
17	2	1	2				7-3-4-5-2
18	3	1	2	1			2-3-4-5-2
19	4	1	2	1	1		2-3-4-5-2
20	**5**	**1**	**2**	**1**	**1**	**1**	**2-3-4-5-6-7**
21	**5**	**1**	**2**	**1**	**1**	**2**	**7-3-4-5-6-8-6-7**
22	4	1	2	1	2		7-3-4-5-2
23	**5**	**1**	**2**	**1**	**2**	**1**	**2-3-4-5-6-7**
24	**5**	**1**	**2**	**1**	**2**	**2**	**7-3-4-5-6-8-6-8-6-7**
25	3	1	2	2			7-3-4-5-2
26	4	1	2	2	1		2-3-4-5-2
27	**5**	**1**	**2**	**2**	**1**	**1**	**2-3-4-5-6-7**
28	**5**	**1**	**2**	**2**	**1**	**2**	**7-3-4-5-6-8-6-7**
29	4	1	2	2	2		7-3-4-5-2
30	**5**	**1**	**2**	**2**	**2**	**1**	**2-3-4-5-6-7**
31	**5**	**1**	**2**	**2**	**2**	**2**	**7-3-4-5-6-8-6-8-6-8-6-8-6-8-6-8**
32	0						8-Terminate

No bounds or objective functions are assumed at Steps 4 and 5 for this illustration. The 15 partitions of $n + m = 5$ objects into $K = 2$ cells are displayed in bold.

Although the development of effective heuristic approaches for cell formation is vital for applications associated with large part–machine clustering matrices, progress in the design of efficient exact methods for small- to medium-sized instances is also imperative. Exact methods have inherent value because they can provide guaranteed globally optimal solutions for applications of modest size. In addition, they afford definitive benchmarks for the evaluation of heuristic methods. In this article, we develop a standalone branch-and-bound clustering procedure that can be used to solve the part–machine clustering problem associated with the maximization of the GEI. The algorithm uses an initial bound from a fast relocation heuristic, as well as a favorable reordering of the list of parts and machines.

Most important, the algorithm incorporates a process that constructs upper bounds on the best possible contributions to voids and exceptional elements that can be realized from parts and machines that have not yet been assigned to a cell. The algorithm has been successfully applied to 31 part–machine incidence matrices from the literature, which ranged in size from 5 × 7 to 40 × 24. Computation times varied substantially across the test problems, and were affected by the number of parts, the number of machines, the number of cells, and whether the incidence matrix was well-structured or ill-structured. The principal conclusion extracted from the computational findings is that the proposed branch-and-bound algorithm does not obviate the need for the development of good metaheuristics; however, it does provide a useful advancement in the design of exact methods for part–machine clustering matrices of modest size.

A formal development of the underlying optimization problem associated with maximizing the GEI is offered in Section 2. A Mixed-Integer Linear Programming (MILP) formulation of the problem is described in Section 3. The new special-purpose branch-and-bound algorithm is presented in Section 4. Computational results for 104 problems spanning 31 matrices from the cell formation literature are reported in Section 5. The article concludes with a brief summary in Section 6.

2. Maximizing the GEI

2.1. *Basic definitions and notation*

Our development of the cell formation problem associated with the maximization of the GEI uses the following notation:

$m :=$ the number of machines;
$n :=$ the number of parts;
$S :=$ the set of all objects (parts and machines) represented by indices, $S = \{1, 2, \ldots, n + m\}$, such that the order of the parts and machines in S will be conducive to effective design of the branch-and-bound process, as described in Section 4.1;
$\tau(q) :=$ 1 if object q in S is a part or 2 if object q is a machine, for $1 \le q \le n + m$;
$\psi(q) :=$ the part or machine index associated with object q in S, for $1 \le q \le n + m$;
$K :=$ the number of cells (or clusters);
$\mathbf{A} :=$ the part-machine incidence matrix with elements $a_{ij} = 1$ if part i requires processing on machine j and 0 otherwise, for $1 \le i \le n$ and $1 \le j \le m$;
$P :=$ a partition, $P = \{C_1, \ldots, C_K\}$, of the objects in S into K cells, where $C_k \subset S$ is the set of object indices assigned to cell k, for $1 \le k \le K$. Throughout the article, we will denote the cardinality (or number of objects) in cell k as $|C_k|$;
$\Pi :=$ the set of all partitions of the $n + m$ objects into K cells;
$e :=$ the total number of operations in the part–machine incidence matrix ($\mathbf{A}$),

$$e = \sum_{i=1}^{n} \sum_{j=1}^{m} a_{ij}; \tag{1}$$

$e_V(P) :=$ the total number of intra-cell voids in the cells associated with P,

$$e_V(P) = \sum_{k=1}^{K} \left(\sum_{i \in C_k : \tau(i)=1} \sum_{j \in C_k : \tau(j)=2} (1 - a_{\psi(i)\psi(j)}) \right); \tag{2}$$

$e_O(P) :=$ the total number of inter-cell exceptional elements in the cells associated with P,

$$e_O(P) = \sum_{k=1}^{K} \sum_{1 \le l \le K : l \ne k} \left(\sum_{i \in C_k : \tau(i)=1} \sum_{j \in C_l : \tau(j)=2} a_{\psi(i)\psi(j)} \right). \tag{3}$$

2.2. *The GEI*

The GEI for partition P is computed as a function of the number of voids, exceptional elements, and total number of operations as

$$\Gamma(P) = \frac{e - e_O(P)}{e + e_V(P)}. \tag{4}$$

With these definitions in place, the optimization problem associated with the maximization of the GEI can be concisely stated as

$$\max_{P \in \Pi} : \{\Gamma(P)\}. \tag{5}$$

Simply stated, the goal is to find the K-cell partition (P) from among the set of all possible K-cell partitions in Π that maximizes the grouping efficacy, $\Gamma(P)$. The challenge is that the number of ways to partition $n + m$ objects into K cells grows exponentially as a function of these parameters. The total number of partitions, $\theta(n, m, K)$, as a function of n, m, and K, is expressed using the following formula (see, for example, Brusco and Stahl (2005), p. 18):

$$\theta(n, m, K) = \frac{1}{K!} \sum_{k=0}^{K} (-1)^k \binom{K}{k} (K - k)^{(n+m)}. \tag{6}$$

For example, for a part–machine clustering problem with $n + m = 20$ objects (parts plus machines) and $K = 4$ cells, there are roughly 45.2 billion partitions in Π. For $n + m = 30$ objects (parts plus machines) and $K = 5$ cells, there are approximately 7.7×10^{18} (or 7.7 quintillion) partitions in Π. In light of the enormous solution space for problems of practical size, the emphasis on the design of metaheuristics for part–machine clustering problems is not surprising. In addition, it might be necessary to solve the optimization

problem in Equation (5) for different values of K to ensure that the proper number of cells is identified.

There are a couple of important assumptions regarding the part–machine clustering problem that require clarification before proceeding. These assumptions are critical because they explain some of the differences in the results for the same test problems that have been observed for various methods in the literature. The assumptions pertain to: (i) *singletons* and (ii) *residual cells*. A singleton is a cell that contains only one object. Some methods preclude this from occurring (see Paydar and Saidi-Mehrabad (2013), p. 987, for a list of heuristics that do not permit singletons). Residual cells occur when only parts (or only machines) are assigned to a cell. As noted by Li *et al*. (2010), the acceptance of residual cells can sometimes result in an improvement in efficacy because it reduces the number of voids. The viability of singletons and residual cells in part–machine clustering solutions is debatable. On the one hand, they lack the pragmatic aspect of a cell consisting of a family of parts *and* machines. However, allowing singletons and residual cells can be useful for facilitating the identification of parts or machines that are outliers in their respective categories.

In our formulation of the problem, both singletons and residual cells are permitted. Nevertheless, if desired, the inclusion of additional constraints to prevent these conditions from occurring is straightforward. For example, constraint sets (7) and (8) below will prevent residual cells from occurring because they require at least one part and one machine to be assigned to each cell:

$$|\{q \in C_k : \tau(q) = 1\}| \geq 1 \,\forall\, 1 \leq k \leq K, \tag{7}$$

$$|\{q \in C_r : \tau(q) = 2\}| \geq 1 \forall\, 1 \leq l \leq K. \tag{8}$$

If an analyst wanted to require two or more parts and two or more machines in each cell, then the right-hand sides of constraints (7) and (8) should be increased to two.

3. A MILP formulation

3.1. *A general two-mode clustering formulation*

In this section, we present an MILP formulation of a general two-mode clustering problem described by Brusco and Steinley (2009) within the context of block modeling of social networks. Two-mode clustering allows the number of clusters for row objects (e.g., parts) and column objects (e.g., machines) to be different. Although the number of clusters (cells) for parts and machines is typically the same for part–machine clustering, the presented formulation affords the ability to relax this restriction. It will subsequently be shown in Section 3.2 that part–machine clustering can be considered as a one-mode special case of the two-mode formulation.

We denote the number of row and column clusters as K_1 and K_2, respectively. In addition to the two-mode binary input matrix **A**, we also prespecify $K_1 \times K_2$ image and penalty matrices, **B** and **D**, respectively. An element of the image matrix, b_{kl} (for all $1 \leq k \leq K_1$ and $1 \leq l \leq K_2$), indicates whether the elements of the submatrix associated with row cluster k and column cluster l should be all zeros ($b_{kl} = 0$) or all ones ($b_{kl} = 1$). The non-negative elements of the penalty matrix, d_{kl} (for all $1 \leq k \leq K_1$ and $1 \leq l \leq K_2$), reflect the penalty that is imposed on a violation within a submatrix (i.e., the presence of a one in a submatrix that should be all zeros, or *vice versa*). Matrices **A**, **B**, and **D** are used to produce penalty parameters η_{ijkl}, which are computed as follows for all $1 \leq i \leq n$, $1 \leq j \leq m$, $1 \leq k \leq K_1$, and $1 \leq l \leq K_2$:

$$\eta_{ijkl} = d_{kl}(b_{kl} + a_{ij} - 2b_{kl}a_{ij}). \tag{9}$$

To describe the rationale behind the η_{ijkl} parameters, it is helpful to reiterate that **A** and **B** are binary matrices. If $b_{kl} = a_{ij}$, then the parenthetical term in Equation (9) is zero. This is appropriate because, if $b_{kl} = a_{ij} = 1$, then the elements of the submatrix associated with row cluster k and column cluster l should be all ones, and a value of $a_{ij} = 1$ in that submatrix is appropriate and should not be penalized. Likewise, if $b_{kl} = a_{ij} = 0$, then the elements of the submatrix associated with row cluster k and column cluster l should be all zeros, and a value of $a_{ij} = 0$ in that submatrix is appropriate and should not be penalized. However, if $b_{kl} \neq a_{ij}$, then the implication is that there is either an element of $a_{ij} = 1$ in a submatrix that should contain zeros ($b_{kl} = 0$) or an element of $a_{ij} = 0$ in a submatrix that should contain ones ($b_{kl} = 1$). Note that, if $b_{kl} \neq a_{ij}$, then the parenthetical term in Equation (9) is one and thus the penalty of d_{kl} is incurred.

The decision variables for the formulation are as follows:

x_{ik} := the binary assignment variables for the row objects, where $x_{ik} = 1$ if row object i is assigned to row cluster k, and $x_{ik} = 0$ otherwise, for $1 \leq i \leq n$ and $1 \leq k \leq K_1$;

y_{jl} := the binary assignment variables for the column objects, where $y_{jl} = 1$ if column object j is assigned to column cluster l, and $y_{jl} = 0$ otherwise, for $1 \leq j \leq m$ and $1 \leq l \leq K_2$;

z_{ijkl} := the linearization variables that assume values of $z_{ijkl} = 1$ if row object i is placed in row cluster k and column object j is placed in column cluster l and $z_{ijkl} = 0$ otherwise, for $1 \leq i \leq n$, $1 \leq j \leq m$, $1 \leq k \leq K_1$, and $1 \leq l \leq K_2$.

The MILP formulation is as follows:

$$\text{Minimize} \sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{K_1}\sum_{l=1}^{K_2} \eta_{ijkl} z_{ijkl}, \tag{10}$$

$$\text{subject to} \sum_{k=1}^{K_1} x_{ik} = 1 \;\text{ for} 1 \leq i \leq n; \tag{11}$$

$$\sum_{l=1}^{K_2} y_{jl} = 1 \text{ for } 1 \leq j \leq m; \tag{12}$$

$$\sum_{i=1}^{n} x_{ik} \geq 1 \text{ for } 1 \leq k \leq K_1; \tag{13}$$

$$\sum_{j=1}^{m} y_{jl} \geq 1 \text{ for } 1 \leq l \leq K_2; \tag{14}$$

$$x_{ik} + y_{jl} - z_{ijkl} \leq 1 \text{ for } 1 \leq i \leq n,\ 1 \leq j \leq m,\ 1 \leq k \leq K_1,\ 1 \leq l \leq K_2; \tag{15}$$

$$x_{ik} \in \{0, 1\} \text{ for } 1 \leq i \leq n,\ 1 \leq k \leq K_1, \tag{16}$$

$$y_{jl} \in \{0, 1\} \text{ for } 1 \leq j \leq m,\ 1 \leq l \leq K_2, \tag{17}$$

$$0 \leq z_{ijkl} \leq 1 \text{ for } 1 \leq i \leq n,\ 1 \leq j \leq m,\ 1 \leq k \leq K_1,\ 1 \leq l \leq K_2. \tag{18}$$

The objective function (10) minimizes the penalty-weighted inconsistencies with the ideal submatrix structure imposed by the image matrix. Constraint set (11) ensures that each row object is assigned to exactly one row cluster. Constraint set (12) provides a similar restriction for column objects and clusters. Constraint sets (13) and (14) prevent empty row and column clusters, respectively. Constraint set (15) requires $z_{ijkl} = 1$ if $x_{ik} = y_{jl} = 1$. Constraint sets (16) and (17) guarantee the binary nature of the row object and column object assignment variables, respectively. The z_{ijkl} variables are bounded in constraint set (18).

Effectively, z_{ijkl} should be the product of the x_{ik} and y_{jl} variables, and constraints (15) and (18) together provide a linearization of the product term. In addition, it is implied that $z_{ijkl} \leq x_{ik}$ and $z_{ijkl} \leq y_{jl}$, but explicit incorporation of these constraints is not needed because of the goal of minimizing a function of the z_{ijkl} variables in Equation (10). A similar argument precludes the need to explicitly bound $z_{ijkl} \leq 1$. Finally, it should also be noted that only a subset of the z_{ijkl} variables is needed for any given formulation. There are $nK_1 + mK_2$ binary assignment variables (i.e., the x_{ik} and y_{jl} variables) in the formulation. There are potentially mnK_1K_2 of the z_{ijkl} variables; however, there are typically far fewer. A z_{ijkl} variable is only required if $\eta_{ijkl} > 0$ and, as noted above, $\eta_{ijkl} = 0$ for all instances where $b_{kl} = a_{ij}$. There are $n + m + K_1 + K_2$ assignment constraints spanning constraints (11) to (14). There are potentially mnK_1K_2 linearization constraints in Equation (15); however, elimination of a z_{ijkl} variable also eliminates one of these constraints. It is the introduction of the linearization constraints that precludes a totally unimodular constraint set and, accordingly, necessitates the use of integer programming methods. It should be noted, however, that a cutting plane method could possibly be developed to append linearization constraints in an efficient manner rather than incorporating them all at once (see, for example, Crama and Oosten (1996)).

3.2. *The special case of part–machine clustering*

The part–machine clustering problem can be viewed as a somewhat simplified special case of the two-mode clustering problem described in Section 3.1. The simplifying assumptions are as follows.

1. The number of row clusters equals the number of column clusters: $K_1 = K_2 = K$.
2. The image matrix, **B**, is a $K \times K$ identity matrix.
3. The penalty matrix, **D**, is a $K \times K$ matrix of ones.
4. If residual cells are permitted, then constraints (13) and (14) can be collapsed as follows:

$$\sum_{i=1}^{n} x_{ik} + \sum_{j=1}^{m} y_{jk} \geq 1 \text{ for } 1 \leq k \leq K. \tag{19}$$

Assumption 1 reduces the problem to one of K cells. Assumption 2 creates the block-diagonal form, whereby zeros in the main diagonal submatrices are voids and ones in the off-diagonal submatrices are exceptional elements. Assumption 3, in conjunction with Assumption 2, produces an objective function designed to minimize the total sum of voids plus exceptional elements. Assumption 4 allows residual cells to permit direct comparison to the branch-and-bound algorithm described in Section 4.

The one remaining complication is that the objective function considered herein is to maximize the GEI, not minimize the sum of voids plus exceptional elements. As noted in the Introduction, the resulting objective function is nonlinear in the decision variables (see Crama and Oosten (1996), p. 1705). However, this difficulty is rectified via conversion of the fractional GEI using a procedure originally devised by Dinkelbach (1967). To illustrate, let us assume that we have a lower bound (Γ_{LB}) for the GEI, which can be obtained by any available heuristic from literature. The solution to the MILP model must produce a GEI that equals or exceeds this lower bound:

$$\frac{e - e_O}{e + e_V} \geq \Gamma_{LB}. \tag{20}$$

This constraint can be re-written to isolate variables on the left and constants on the right as follows:

$$e_O + \Gamma_{LB} e_V \leq e(1 - \Gamma_{LB}). \tag{21}$$

Expressing Equation (21) using the decision variables from the MILP model yields

$$\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{K}\sum_{l\neq k} \eta_{ijkl} z_{ijkl} + \Gamma_{LB}\left[\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{K} \eta_{ijkk} z_{ijkk}\right] \leq e(1 - \Gamma_{LB}). \tag{22}$$

Alternatively, Equation (22) can be written as

$$\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{K}\sum_{l\neq k}\eta_{ijkl}z_{ijkl} + \Gamma_{LB}\left[\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{K}\eta_{ijkk}z_{ijkk}\right] - e(1-\Gamma_{LB}) \leq 0. \quad (23)$$

Crama and Oosten (1996) observed that the crux of the Dinkelbach (1967) method in this context is to use the left-hand side of Equation (23) as a linear objective function for an MILP formulation with Equations (11) and (12) and (15) to (19) as the constraints. A better solution (i.e., one with a larger GEI than Γ_{LB}) than the one obtained by the heuristic can only exist if the optimal objective function is negative. If a solution with a negative objective function is obtained, then Γ_{LB} is updated, the new objective function (23) is constructed, and the problem is re-solved. This process of solving a series of MILPs continues until the objective function value is zero.

We tested the Dinkelbach (1967) approach to the problem as described by Crama and Oosten (1996); however, we found that a mild variation of this procedure generally produced confirmed globally optimal solutions in less time. Our variation emphasizes constraint satisfaction and presupposes that the heuristic solution is of excellent quality and likely optimal. Given that we know in advance that the heuristic solution is feasible, we solve the MILP formulation using an arbitrary objective function, constraints (11) and (12), (15) to (19), and (22) using ($\Gamma_{LB} + \varepsilon$), where ε is a small positive constant. If the resulting problem is infeasible, then we know that the heuristic solution is ε-optimal (i.e., its objective function is within ε of the true optimal objective function value). However, if a feasible solution with a GEI of Γ_{UB} is obtained, then we set $\Gamma_{LB} = \Gamma_{UB}$ and re-solve the formulation using ($\Gamma_{LB} + \varepsilon$) in Equation (22). This process is repeated until infeasibility occurs.

The objective function in our proposed solution process is arbitrary because the problem is essentially one of constraint satisfaction. One natural selection for the objective function is to minimize the left-hand side of Equation (23); however, we obtained slightly better results using a null objective function (all zeros) and employing the "set mipopt emphasis mip" "integer feasibility" setting in CPLEX 12.1, which emphasizes constraint satisfaction to the exclusion of objective function improvement.

4. An exact algorithm

4.1. *An overview*

The exact algorithm for maximizing the GEI is a forward branch-and-bound method that uses a depth-first search process (see, for example, Klein and Aronson (1991); Brusco and Stahl (2005), Chapters 2-5). The depth (or level) of the search tree is measured by the object (i.e., part or machine) that is currently under consideration for assignment, whereas the breadth at any given level is controlled by the number of possible cells to which that object can be assigned. We will use the term *partial assignment* to refer to incomplete solutions where some (but not all) of the objects have been assigned. The term *complete assignment* is a feasible solution to the part–machine clustering problem such that all objects have been assigned. Ideally, the branch-and-bound process should prune many partial solutions early in the search tree by detecting that they cannot ultimately lead to a complete solution that is better than the current incumbent lower bound (Γ_{LB}). This pruning process prevents the need to explicitly generate the vast number of feasible partitions in Π that comprises the solution space.

It is also beneficial to ensure that the parts and machines are placed in S in an alternating sequence based on their maximum number of operations. For example, assume that the parts are ranked in descending order based on the number of machines that they require and, similarly, the machines are ranked in descending order based on the number of parts that they process. The first part in the ordered sequence of parts is placed as the first object in S and, therefore, $\tau(1) = 1$ and $\psi(1) = 1$. The first machine in the ordered sequence of machines is placed as the second object in S ($\tau(2) = 2$ and $\psi(2) = 1$). The second part in the ordered sequence of parts is the third object in S ($\tau(3) = 1$ and $\psi(3) = 2$) and the second machine in the ordered sequence of machines is the fourth object in S ($\tau(4) = 2$ and $\psi(4) = 2$), and so on. By alternating the positioning of parts and machines in S in this manner, good upper bounds for the GEI can be achieved more rapidly.

4.2. *Steps of the algorithm*

The steps of the branch-and-bound algorithm are as follows.

Step 0. Obtain lower bound. Select a desired number of cells, K, and apply a heuristic procedure to establish the initial lower bound (Γ_{LB}) for the GEI. Proceed to Step 1.

Step 1. Initialize. Place the elements in S in alternating sequence and define $\tau(q)$ and $\psi(q)$, as described in Section 4.1, $\forall\ 1 \leq q \leq n+m$. Set $q = 0$, $\sigma = K$ (the number of empty clusters) $e_V = 0$, $e_O = 0$, and $C_l = \varnothing$ ($\forall\ 1 \leq l \leq K$). Proceed to Step 2.

Step 2. Branch forward. Set $q = q + 1$, $k = 1$, and $C_k = C_k \cup \{q\}$. If $|C_k| = 1$, then set $\sigma = \sigma - 1$. Compute the contribution of assigning object q to cell k to the number of voids (e_V) and the number exceptional elements (e_O). Proceed to Step 3.

Step 3. Feasibility test. If $(n + m) - q < \sigma$, then go to Step 6; otherwise, proceed to Step 4.

Step 4. Bound test. Compute an upper bound, Γ_{UB}. If $\Gamma_{UB} \leq \Gamma_{LB}$, then go to Step 6; otherwise, proceed to Step 5.

Step 5. Update incumbent? If $q < n + m$, then go to Step 2; otherwise, set $P^* = \{C_1, \ldots, C_K\}$, $\Gamma_{LB} = \Gamma_{UB}$, and proceed to Step 6.

Step 6. Branch right or retract? If $(k = K) \vee (|C_k| = 1 \wedge |C_{k+1}| = 0)$, then go to Step 8; otherwise, proceed to Step 7.

Step 7. Branch right on cell assignment. Remove the contribution of assigning object q to cell k to the number of voids (e_V) and the number of exceptional elements (e_O). Set $C_k = C_k \setminus \{q\}$ and, if $|C_k| = 0$, then set $\sigma = \sigma + 1$. Set $k = k + 1$. Set $C_k = C_k \cup \{q\}$ and, if $|C_k| = 1$, then set $\sigma = \sigma - 1$. Compute the contribution of assigning object q to cell k to the number of voids (e_V) and the number exceptional elements (e_O). Return to Step 3.

Step 8. Depth retraction. Remove the contribution of assigning object q to cell k to the number of voids (e_V) and the number of exceptional elements (e_O). Set $C_k = C_k \setminus \{q\}$ and, if $|C_k| = 0$, then set $\sigma = \sigma + 1$. Set $q = q - 1$. If $q = 0$, then return P^* with Γ_{LB} as the globally optimal value of the GEI and STOP; otherwise, set $k : q \in C_k$ and proceed to Step 6.

The branch-and-bound algorithm is implemented for a user-selected value of K. Once K is selected, a lower bound for the GEI can be established in Step 0 using any of a variety of possible methods. At one extreme, the lower bound could be prespecified as $\Gamma_{LB} = 0$, whereas at the other extreme sophisticated metaheuristics could be applied to obtain Γ_{LB}. In this article, we use a relocation heuristic to obtain the lower bound. This heuristic is described in Section 4.3.

Parameters for the branch-and-bound algorithm are initialized in Step 1. Step 2 adds the next object in the ordered list (q) to the first cell ($k = 1$). This is referred to as a branch forward operation because the algorithm is moving deeper into the search tree. The values of e_V and e_O correspond, respectively, to the actual number of voids and exceptional elements associated with the assignment to cells of the first q parts and machines in S. These values are dynamically updated each time a cell assignment is changed or added, which is far more computationally economical than recomputing these values from scratch each time the GEI needs to be computed. The details of the updating process are described in Section 4.4. If $(n + m) - q < \sigma$ in the feasibility test at Step 3, then there are not enough remaining objects to fill up the remaining empty cells. The partial assignment is therefore pruned and control is passed to Step 6, where a decision is made whether to branch right or retract. The bound test at Step 4 is the most crucial component associated with the success of the algorithm. At Step 4, an upper bound is computed, which represents the best possible GEI index that could be established based on the completion of the current partial assignment. The details of the upper bound are presented in Section 4.5. If $\Gamma_{UB} \leq \Gamma_{LB}$ at Step 4, then control is passed is to Step 6; however, if $\Gamma_{UB} > \Gamma_{LB}$, the algorithm proceeds to Step 5. For partial assignments ($q < n + m$), Step 5 returns control to Step 2 for assignment of the next object, whereas complete assignments ($q = n + m$) are installed as the new incumbent solution.

Step 6 makes a determination as to whether to branch right or retract. A branch right operation occurs when the current object (q) is moved from cell k to the next cell on the right (i.e., $k + 1$). This is accomplished in Step 7 by removing the contribution of q based on its assignment to cell k, removing the assignment to cell k, incrementing k to $k + 1$ and reassigning q to that cell, and returning control to Step 3. If $k = K$ at Step 6, then the current object (q) cannot be assigned to the next cell on the right because there is not a cell $K + 1$. Moreover, if object q is the only object in cell k, then it is not advantageous to branch right because this will create a cell $(k + 1)$ with an object, when there is a cell with a lower index (i.e., k) that has no objects assigned. Avoiding the branch right operation based on this condition is vital because it prevents the evaluation of "symmetric" cell assignments that differ only with respect to the possible $K!$ labelings of cells (see Klein and Aronson (1991); Brusco and Stahl (2005), Chapters 2-3).

The depth retraction process in Step 8 begins with the removal of the contribution of q based on its assignment to cluster k in a manner similar to Step 7. The assignment of object q to cell k is removed and, subsequently, the object pointer q is reduced (i.e., $q = q - 1$). If $q = 0$, then all partitions have been either implicitly or explicitly evaluated and the algorithm terminates with P^* as an optimal partition. If $q > 0$, then control is passed to Step 6.

To illustrate the construction of a search tree, we consider an example with $n + m = 5$ objects and $K = 2$ cells. The objective function and its corresponding lower and upper bounds are not relevant to this demonstration, as the goal is to illustrate a complete search tree. The partial and complete solutions are displayed in Table 1. Table 1 provides, for each solution, the position pointer q, the cell assignments for objects 1 to q, and the sequence of processing steps for the solution. For clarity, the beginning of the sequence for each stage is re-displayed from the last step of the sequence from the previous stage.

Consider, for example, stage 1 in Table 1. At this stage, $q = 1$ and the first object has been assigned to cluster 1 but none of the other objects are assigned to clusters. The assignment of object 1 to cluster 1 occurs in Step 2, and processing then proceeds through to Steps 3, 4, and 5. At Step 5, because $q = 1 < n + m = 5$, processing returns to Step 2 for another branch forward operation (hence, the sequence 2-3-4-5-2). At stage 5, $q = n + m = 5$ and *all* objects are assigned to cell 1, which means that $\sigma = 1$ because one cell (cell 2) is empty. This is, therefore, not a partition of the objects into $K = 2$ cells, and the solution is pruned at Step 3 because $(n + m) - q = 0 < \sigma = 1$. Step 3 passes control to Step 6, which passes control to Step 7 for a branch right operation where object $q = 5$ will be moved from cluster $k = 1$ to cluster $k = 2$ (hence, the 2-3-6-7 sequence for stage 5). Accordingly, stage 6 is the first compete assignment of objects to cells, such that $C_1 = \{1, 2, 3, 4\}$ and $C_2 = \{5\}$.

The stages corresponding to the 15 possible partitions of $n + m = 5$ objects into $K = 2$ cells are highlighted in bold in Table 1. At stage 31, the final partition is evaluated. Proceeding through stages 7-3-4-5-6, a decision to retract is made at Stage 6 because $k = K = 2$ (i.e., object $q = 5$ is currently assigned to cell 2). Retraction at Step 8 reduces the pointer index to $q = 4$, but we cannot branch right on that object either because it is also assigned to cluster $k = 2$, so further retraction occurs. In fact, retraction occurs all the way back to $q = 0$ (this repeated retraction is the reason for the 6-8-6-8-6-8-6-8-6-8 at the end of the sequence at stage 31. It is important to recognize that the last retraction in this sequence, from $q = 1$ to $q = 0$ occurs, not because $k = K$ but because $|C_1| = 1$ and $|C_2| = 0$. This process is crucial because it avoids evaluating partitions that are symmetric, mirror images (i.e., differ only in cluster labeling) of those already evaluated. Upon retraction of the object index to $q = 0$, the algorithm terminates.

Table 1 displays the worst-case scenario for the algorithm, which involves complete enumeration of the search tree in the absence of bound information, for a problem with $n + m = 5$ objects and $K = 2$ cells. The key to the success of the branch-and-bound algorithm is to circumvent complete enumeration via the bound test in Step 4. For example, suppose that bound information were available, and that the bound test at Step 4 in stage 17 of Table 1 yielded $\Gamma_{UB} \leq \Gamma_{LB}$, resulting in a pruning of the search tree at this stage. Control would be passed from Step 4 to Step 6 (instead of Step 5) and, subsequently, to Step 8 for retraction because $k = 2$. A second retraction would then occur based on $|C_1| = 1$ and $|C_2| = 0$ and the algorithm would terminate, thus avoiding the explicit enumeration of the partial and complete assignments in stages 18 and beyond.

4.3. *Relocation heuristic to obtain a lower bound (Γ_{LB})*

Prior to the initiation of the branch-and-bound process, it is necessary to establish a good lower bound (Γ_{LB}) for the GEI. This is accomplished using multiple restarts of a relocation heuristic, which is similar in design to those used in cluster analysis (Jancey, 1966; Banfield and Bassill, 1977). More precisely, the relocation heuristic used herein is an adaptation of a generic transfer algorithm developed by Banfield and Bassill (1977) within the context of nonhierarchical classification. This type of relocation heuristic has also been employed gainfully within the context of blockmodeling of social networks (Doreian *et al*., 2005). The relocation heuristic begins with the random assignment of parts and machines to cells. Each part and machine is then considered in turn for transfer to different cells, and such transfers are accepted if they improve the GEI. The heuristic terminates when no relocation of a part or machine can further increase the GEI. The pseudo-code for the relocation heuristic is provided in Fig. 1.

Although analytical worst- and average-case performance information for the relocation heuristic is not available, there is computational evidence that multiple restart (or multistart) implementations of the method perform exceptionally well for partitioning binary data (Doreian *et al*., 2005; Brusco *et al*., 2013). For example, when clustering a 141 × 272 binary matrix containing United Nations General Assembly voting data, Brusco *et al*. (2013) found that a multiple restart relocation heuristic produced results that were highly competitive with tabu search and variable neighborhood search. In this article, the relocation heuristic is implemented using 100 000 restarts (each using a randomly generated initial solution) and the best-found GEI across all restarts is used to establish the lower bound. The impact of reducing the number of restarts to 100 is investigated in Section 5.4.

4.4. *Dynamic updating of (e_V) and (e_O)*

It would be computationally demanding to recompute e_V and e_O from scratch for each partial solution. Therefore, these values are dynamically updated each time an object, q, is added or removed from a cell. Additions of an object (q) to a cell (k) can occur in either Step 2 or Step 7. If object q is a part (i.e., $\tau(q) = 1$), then the number of voids and exceptional elements are updated using Equations (24) and (25), respectively, as follows:

$$e_V = e_V + \sum_{r \in C_k : \tau(r) = 2} (1 - a_{\psi(q)\psi(r)}); \tag{24}$$

$$e_O = e_O + \sum_{r \notin C_k : \tau(r) = 2} a_{\psi(q)\psi(r)}. \tag{25}$$

Similarly, if object q is a machine (i.e., $\tau(q) = 2$), then the number of voids and exceptional elements are updated using Equations (26) and (27), respectively, as follows:

$$e_V = e_V + \sum_{r \in C_k : \tau(r) = 1} (1 - a_{\psi(r)\psi(q)}); \tag{26}$$

$$e_O = e_O + \sum_{r \notin C_k : \tau(r) = 1} a_{\psi(r)\psi(q)}. \tag{27}$$

Removal of an object from a cell occurs in the branch right (Step 7) and depth retraction (Step 8) steps. If object q is a part (i.e., $\tau(q) = 1$), then the number of voids and exceptional elements are reduced using Equations (28) and (29), respectively, as follows:

$$e_V = e_V - \sum_{r \in C_k : \tau(r) = 2} (1 - a_{\psi(q)\psi(r)}); \tag{28}$$

$$e_O = e_O - \sum_{r \notin C_k : \tau(r) = 2} a_{\psi(q)\psi(r)}. \tag{29}$$

Similarly, if object q is a machine (i.e., $\tau(q) = 2$), then the number of voids and exceptional elements are reduced using Equations (30) and (31), respectively, as follows:

$$e_V = e_V - \sum_{r \in C_k : \tau(r) = 1} (1 - a_{\psi(r)\psi(q)}); \tag{30}$$

```
% Randomly Construct an Initial Partition
        Randomly select K parts (without replacement) to serve as cell exemplars based on a uniform
        distribution. Assign each of the remaining n – K parts to the cell corresponding to the exemplar
        to which they are nearest. Repeat this process for machines and denote the resulting initial
        partition of parts and machines as P.
        Set ξ = 1.
% Implement the Relocation Heuristic Process
While ξ = 1
        ξ = 0
        % Check All Relocations of Parts to Different Cells
        For q ∈ S : τ(q) = 1
                h = h : q ∈ C_h
                if | C_h | > 1
                        for k = 1 to K
                                if k ≠ h then
                                        P′ = {P \ {C_k, C_h}} ∪ {C_k ∪ {q}} ∪ {C_h \ {q}}
                                        if Γ(P′) > Γ(P)
                                                P = P′
                                                ξ = 1
                                        end if
                                end if
                        next k
                end if
        next q
        % Check All Relocations of Machines to Different Cells
        For q ∈ S : τ(q) = 2
                h = h : q ∈ C_h
                if | C_h | > 1
                        for k = 1 to K
                                if k ≠ h then
                                        P′ = {P \ {C_k, C_h}} ∪ {C_k ∪ {q}} ∪ {C_h \ {q}}
                                        if Γ(P′) > Γ(P)
                                                P = P′
                                                ξ = 1
                                        end if
                                end if
                        next k
                end if
        next q
End While
Return P
```

Fig. 1. Pseudo-code for the relocation heuristic.

$$e_O = e_O - \sum_{r \notin C_k : \tau(r)=1} a_{\psi(r)\psi(q)}. \tag{31}$$

4.5. *Computation of the upper bound,* Γ_{UB}

A description of the computation of the upper bound, Γ_{UB}, for a partial assignment is facilitated by a decomposition of the part–machine clustering matrix, **A**. The complete part–machine clustering matrix has nm (i.e., n times m) elements. Now, consider a partial assignment of the first $n_1 < n$ parts and $m_1 < m$ machines, this enables a decomposition of nm as follows:

$$\begin{aligned} nm = {} & n_1 m_1 + n_1(m - m_1) + (n - n_1)m_1 \\ & + (n - n_1)(m - m_1). \end{aligned} \tag{32}$$

The contribution to voids and exceptional elements stemming from the first term ($n_1 m_1$) on the right-hand side of Equation (32) is readily available because the $q = n_1 + m_1$ objects have been assigned to cells. These assigned items produce e_V and e_O, which correspond to the number of voids and exceptional elements, respectively, that are directly contributed by the assignment of the first q objects in S. The remaining components on the right-hand side of Equation (32) cannot be directly measured because they involve objects that have not yet been assigned to cells. Nevertheless, we have devised a process establishing lower bounds on the number of voids and exceptional elements, respectively, that can stem from the $n_1(m - m_1) + (n - n_1)m_1$ terms, which correspond to relationships between the q assigned objects and the $(n + m) - q$ objects that are yet unassigned. To illustrate, consider the case of bounds for the number of voids that can arise between the assigned and unassigned objects. Each unassigned object u ($\forall\ q + 1 \leq u \leq n + m$) is considered in turn. Each object u is (temporarily) placed in each of the K cells and its contribution to the number of voids in each of the cells is measured. The minimum contribution across all K cells, which is defined as δ_u, represents a best-case scenario for placement of u with respect to the currently assigned objects. More formally, the lower bound on the contribution to voids that can stem from the assignment of the $(n + m) - q$ unassigned objects is computed as follows:

$$\delta_u = \begin{cases} \min\limits_{1\leq l\leq K}\left\{\sum\limits_{r\in C_l:\tau(r)=2}(1-a_{\psi(u)\psi(r)})\right\} & \text{if } \tau(u)=1 \\ \min\limits_{1\leq l\leq K}\left\{\sum\limits_{r\in C_l:\tau(r)=1}(1-a_{\psi(r)\psi(u)})\right\} & \text{if } \tau(u)=2 \end{cases} \quad \forall q+1 \leq u \leq n+m. \tag{33}$$

Likewise, the lower bound on the contribution to exceptional elements that can stem from the assignment of the $(n + m) - q$ unassigned objects is computed as follows:

$$\varphi_u = \begin{cases} \min\limits_{1\leq l\leq K}\left\{\sum\limits_{r\notin C_l:\tau(r)=2}a_{\psi(u)\psi(r)}\right\} & \text{if } \tau(u)=1 \\ \min\limits_{1\leq l\leq K}\left\{\sum\limits_{r\notin C_l:\tau(r)=1}a_{\psi(r)\psi(u)}\right\} & \text{if } \tau(u)=2 \end{cases} \quad \forall\, q+1 \leq u \leq n+m. \tag{34}$$

To summarize, e_V and e_O correspond to the actual number of voids and exceptional elements, respectively, that are realized from the assignment of the first q objects. For each unassigned object, $q + 1 \leq u \leq n + m$, the δ_u and φ_u values represents the minimum possible (i.e., best-case scenario) additions to the number of voids and exceptional elements, respectively, that can stem from that object's ties to the previously assigned q objects. Therefore, adding the sum of the δ_u values to e_V provides a lower bound on the number of possible voids in a complete assignment, and adding the sum of the φ_u values to e_O provides a lower bound on the number of possible exceptional elements in a complete assignment. These lower bounds for voids and exceptional elements translate to an upper bound for the GEI as follows:

$$\Gamma_{UB} = \frac{\left(e - \left(e_O + \sum_{u=q+1}^{n+m}\varphi_u\right)\right)}{\left(e + \left(e_V + \sum_{u=q+1}^{n+m}\delta_u\right)\right)}. \tag{35}$$

The quality of Γ_{UB} is contingent on a couple of factors. First, when q is small (i.e., $q << n+m$), the largest component of Equation (32) is $(n - n_1)(m - m_1)$. Because Γ_{UB} does not capture contributions to e_V and e_O from this component, the upper bounds are typically weak at smaller values of q. However, as q increases, the other three components on the right-hand side of Equation (32) become dominant and the quality of Γ_{UB} improves. Second, as discussed in Section 4.1, the placement of parts and machines with the most non-zero elements in **A** earlier in S is important because it enables the $m_1 n_1$ component on the right-hand side of Equation (32) to improve more quickly. Nevertheless, it must be acknowledged that, even with a sharp lower bound (Γ_{LB}), the quality of Γ_{UB} for ill-structured part–machine matrices (those with no well-defined cells) can be quite weak, in which case the algorithm devolves toward complete enumeration.

5. Computational study

5.1. *Test problems*

The evaluation of the branch-and-bound algorithm described in Section 4 was completed using 31 part–machine incidence matrices from the literature. Twenty-one of the matrices were selected from the set of 35 matrices recently used in a study conducted by Paydar and Saidi-Mehrabad (2013), and the remaining 10 were taken from studies by McAuley (1972), De Witte (1980), Chan and Milner (1982, Fig. 3a), Seifoddini (1988), Lesowsky *et al*. (1990), Vakharia and Wemmerlöv (1990), Shargal *et al*. (1995), and Yang and Yang (2008). Fourteen matrices from the Paydar and Saidi-Mehrabad (2013) study were excluded because the exact algorithm would require an enormous amount of computation time to obtain confirmed globally optimal solutions to solve those problems.

The computational requirement of the algorithm is affected by the total number of objects ($n + m$), the number of clusters (K), and the inherent structure of the part–machine clustering matrix. The sizes of the test matrices that we considered ranged from 5 × 7 (12 objects) to 40 × 24 (64 objects). Although the Paydar and Saidi-Mehrabad (2013) study contained some larger part–machine matrices from the literature (e.g., McCormick *et al*. (1972); King and Nakornchai (1982); Stanfel (1985)), these matrices with $n + m \geq 65$ were not attempted because the algorithm would likely require multiple weeks, months, or even years to obtain solutions to those problems. Nevertheless, it should be realized that the largest

part–machine clustering problem that we solved ($n + m = 64$ objects and $K = 7$ cells) has an enormous solution space of approximately 2.419×10^{50} partitions, as computed using Equation (6).

5.2. *Implementation*

The relocation heuristic, branch-and-bound algorithm, and matrix input generator for the MILP formulation were written in Fortran 90. The programs were implemented on a 2.2 GHz Pentium 4 PC with 1 GB of RAM. This hardware platform was selected based on the availability of the CPLEX 12.1 software for solving the MILP formulations. For each of the first 22 data matrices (matrices 1 through 22), the relocation heuristic, standalone branch-and-bound algorithm, and MILP approach were applied for one or more values of K. Obtaining results for a range of values for K facilitates an examination of the sensitivity of the methods to changes in K. In practice, an analyst might not necessarily need to apply the branch-and-bound or MILP methods for multiple values of K. For example, one strategy might be to run the relocation heuristic for all values of K on some pre-defined range, $K_a \leq K \leq K_b$. Subsequently, the branch-and-bound algorithm could be implemented for only the value of K for which the relocation heuristic yielded its maximum GEI value. In the current study, the relocation heuristic was applied using 100 000 restarts for each value of K considered for each data matrix. The best-found GEI index (Γ_{LB}) across the 100 000 restarts then served as an input to branch-and-bound and MILP methods.

With respect to the detail of the MILP implementation, CPLEX 12.1 was applied to the formulation using $(\Gamma_{LB} + \varepsilon)$ in constraint (22) and an arbitrary (null) objective function where the objective function coefficients for all variables are zero. Because the resulting problem is purely one of constraint satisfaction, we selected "integer feasibility" in the "set mipopt emphasis mip" option in CPLEX 12.1. This option, which directs the focus of the algorithm toward finding a feasible solution (as opposed to the default setting that balances feasibility and optimality) provided up to a 50% reduction in computation time for some of the larger problems.

Results for matrices 1 through 22 were collected with respect to the following performance measures: (i) the best-found GEI value (Γ_{LB}) for the relocation heuristic; (ii) the attraction rate for the relocation heuristic, which is the percentage of restarts for which the best-found GEI value was obtained; (iii) the computation time for the relocation heuristic; (iv) the optimal GEI index (Γ^*) obtained by the branch-and-bound algorithm; (v) the computation time for the branch-and-bound algorithm (including the time to resequence the parts and machines); and (vi) the CPLEX 12.1 computation time for solving the MILP formulation (matrix input generation time, the time to read the matrix input, and the pre-solve time are negligible and not included in the reported times for the MILP approach).

The remaining nine data matrices (matrices 23 through 31), which were larger and/or more difficult than the first 22, were not computationally feasible for the MILP implementation. However, we were able to successfully obtain globally optimal solutions for these problems using the branch-and-bound algorithm when implemented on a 3.4 GHz Intel Core i7-2600 processor with 8.0 GB of RAM.

5.3. *Computational results*

The results for the smallest ($n + m \leq 25$) part–machine matrices in the suite of test problems are reported in Table 2. The multistart implementation of the relocation heuristic identified the globally optimal partition for all instances in Table 2. This is particularly beneficial for the MILP method because it means that global optimality of the heuristic will be immediately identified by the infeasibility of the formulation using $(\Gamma_{LB} + \varepsilon)$ in constraint (22). Moreover, the attraction rate for the relocation heuristic was at least 20% for the vast majority of problems in Table 2; although there were a few instances where the attraction rate was in the 2–3% range. The maximum computation time for 100 000 restarts was 6.97 seconds for the test problems in Table 2.

The most important aspect of the findings in Table 2 is that the branch-and-bound algorithm was extremely effective for these test problems. Across all values of K examined for each test problem, the branch-and-bound algorithm never required more than 0.46 seconds of computation time to obtain the globally optimal partition. Most of the observed branch-and-bound times in Table 2 are reported as "0.00," which means that the computation time was less than one-hundredth of a second. The MILP computation time exceeded the branch-and-bound computation time for all 46 problems in Table 2. The MILP computation times also reflected greater variability, ranging from 0.02 seconds to 52.47 seconds.

The results for the part–machine matrices ranging in size from $28 \leq n + m \leq 38$ are reported in Table 3. As was the case for the smaller problems in Table 2, the multistart relocation heuristic identified the globally optimal partition for all values of K for each test problem in Table 3. The structural quality of the part–machine matrix affected the performance of the relocation heuristic with respect to the attraction rates. Some of the attraction rates for ill-structured matrices were as low as 0.02%. Nevertheless, this still implies that the global optimum was found at least 20 times across the 100 000 restarts.

The branch-and-bound computation times for the problems in Table 3 are larger and exhibit greater variability relative to the results in Table 2. The minimum and maximum computation times for the branch-and-bound algorithm when applied to the problems in Table 3 are 0.00 and 1148.61 seconds, respectively. Table 3 reveals that the

Table 2. Results for selected cell formation matrices from the literature with $n + m \leq 25$

Matrix number	*Source matrix*	*Size*	*Cells*	*Relocation heuristic*			*Branch-and-bound*		*CPLEX time*
				Γ_{LB}	*Time*	*Attraction*	Γ^*	*Time*	
1	King and Nakornchai (1982)	5 × 7	2	0.823 53	0.53	100.00	0.823 53	0.00	0.02
1	King and Nakornchai (1982)	5 × 7	3	0.812 50	0.98	27.15	0.812 50	0.00	0.06
1	King and Nakornchai (1982)	5 × 7	4	0.785 71	1.36	25.77	0.785 71	0.00	0.13
2	Waghodekar and Sahu (1984)	5 × 7	2	0.695 65	0.67	60.12	0.695 65	0.00	0.02
2	Waghodekar and Sahu (1984)	5 × 7	3	0.652 17	1.30	25.14	0.652 17	0.00	0.09
2	Waghodekar and Sahu (1984)	5 × 7	4	0.608 70	1.65	7.28	0.608 70	0.00	0.38
3	Kusiak and Cho (1992)	6 × 8	2	0.769 23	0.74	96.35	0.769 23	0.00	0.03
3	Kusiak and Cho (1992)	6 × 8	3	0.791 67	1.42	72.29	0.791 67	0.00	0.06
3	Kusiak and Cho (1992)	6 × 8	4	0.739 13	1.95	43.83	0.739 13	0.00	0.33
4	Shargal et al. (1995)	7 × 9	2	0.615 38	1.00	55.35	0.615 38	0.00	0.06
4	Shargal et al. (1995)	7 × 9	3	0.666 67	2.09	10.44	0.666 67	0.00	0.17
4	Shargal et al. (1995)	7 × 9	4	0.685 71	2.95	2.65	0.685 71	0.00	0.47
4	Shargal et al. (1995)	7 × 9	5	0.666 67	3.36	28.63	0.666 67	0.00	1.53
5	Kusiak and Chow (1987)	7 × 11	2	0.487 80	1.26	57.57	0.487 80	0.00	0.09
5	Kusiak and Chow (1987)	7 × 11	3	0.586 21	2.30	31.99	0.586 21	0.01	0.25
5	Kusiak and Chow (1987)	7 × 11	4	0.600 00	3.17	9.57	0.600 00	0.05	1.03
5	Kusiak and Chow (1987)	7 × 11	5	0.608 70	3.76	14.03	0.608 70	0.08	1.20
5	Kusiak and Chow (1987)	7 × 11	6	0.565 22	4.28	35.88	0.565 22	0.46	13.22
6	Boctor (1991, Fig. 1)	7 × 11	2	0.463 41	1.11	52.96	0.463 41	0.00	0.09
6	Boctor (1991, Fig. 1)	7 × 11	3	0.703 70	2.43	89.20	0.703 70	0.00	0.16
6	Boctor (1991, Fig. 1)	7 × 11	4	0.708 33	3.02	24.66	0.708 33	0.01	0.56
6	Boctor (1991, Fig. 1)	7 × 11	5	0.695 65	3.53	19.08	0.695 65	0.03	1.88
7	Seifoddini and Wolfe (1986)	8 × 12	2	0.553 57	1.43	36.29	0.553 57	0.02	0.13
7	Seifoddini and Wolfe (1986)	8 × 12	3	0.682 93	3.01	66.64	0.682 93	0.02	0.42
7	Seifoddini and Wolfe (1986)	8 × 12	4	0.694 44	4.14	28.31	0.694 44	0.02	1.16
7	Seifoddini and Wolfe (1986)	8 × 12	5	0.666 67	4.96	4.82	0.666 67	0.02	4.75
8	Mosier and Taube (1985)	10 × 10	2	0.461 54	1.33	58.45	0.461 54	0.00	0.17
8	Mosier and Taube (1985)	10 × 10	3	0.705 88	3.08	88.12	0.705 88	0.00	0.42
8	Mosier and Taube (1985)	10 × 10	4	0.733 33	3.82	21.05	0.733 33	0.01	1.27
8	Mosier and Taube (1985)	10 × 10	5	0.750 00	4.55	5.76	0.750 00	0.02	7.11
8	Mosier and Taube (1985)	10 × 10	6	0.730 77	4.99	6.91	0.730 77	0.03	24.92
9	McAuley (1972)	12 × 10	2	0.529 41	1.52	51.72	0.529 41	0.01	0.23
9	McAuley (1972)	12 × 10	3	0.733 33	3.46	64.41	0.733 33	0.00	0.31
9	McAuley (1972)	12 × 10	4	0.738 10	4.47	33.43	0.738 10	0.01	0.88
9	McAuley (1972)	12 × 10	5	0.725 00	4.99	16.05	0.725 00	0.03	4.88
10	Seifoddini (1989)	5 × 18	2	0.795 92	1.50	100.00	0.795 92	0.00	0.05
10	Seifoddini (1989)	5 × 18	3	0.808 51	3.06	89.93	0.808 51	0.00	0.19
10	Seifoddini (1989)	5 × 18	4	0.782 61	3.87	2.44	0.782 61	0.02	0.58
11	Chan and Milner (1982, Fig. 2a)	10 × 15	2	0.575 00	1.61	24.74	0.575 00	0.00	0.20
11	Chan and Milner (1982, Fig. 2a)	10 × 15	3	0.920 00	3.30	74.27	0.920 00	0.00	0.28
11	Chan and Milner (1982, Fig. 2a)	10 × 15	4	0.897 96	4.37	11.63	0.897 96	0.00	1.23
12	Chan and Milner (1982, Fig. 3a)	10 × 15	2	0.604 94	1.92	57.30	0.604 94	0.00	0.19
12	Chan and Milner (1982, Fig. 3a)	10 × 15	3	0.800 00	3.73	73.15	0.800 00	0.00	0.44
12	Chan and Milner (1982, Fig. 3a)	10 × 15	4	0.811 32	5.30	85.45	0.811 32	0.01	2.22
12	Chan and Milner (1982, Fig. 3a)	10 × 15	5	0.788 46	6.27	20.13	0.788 46	0.01	10.34
12	Chan and Milner (1982, Fig. 3a)	10 × 15	6	0.764 71	6.97	2.59	0.764 71	0.07	52.47

branch-and-bound computation times generally tend to increase as a function of both the number of objects ($n + m$) and the number of cells (K). However, it is evident that, in addition to ($n + m$) and K, the performance of the branch-and-bound algorithm is sensitive to the structural quality of the incidence matrices. For test problems of comparable size, those with larger optimal GEI values (i.e., well-structured matrices) were typically solved in much less time than those with smaller optimal GEI values (i.e., ill-structured matrices). As an example, consider the results for matrix 20 (Seifoddini, 1988) in Table 3. The optimal values of the GEI for matrix 20 ranged from 0.67 to 0.70 over the

Table 3. Results for selected cell formation matrices from the literature with $26 \leq n + m \leq 38$

Matrix number	*Source matrix*	*Size*	*Cells*	*Relocation heuristic*			*Branch-and-bound*		*CPLEX time*
				Γ_{LB}	*Time*	*Attraction*	Γ^*	*Time*	
13	Chandrasekharan and Rajagopalan (1986a)	20 × 8	2	0.629 21	2.08	31.25	0.629 21	0.00	0.19
13	Chandrasekharan and Rajagopalan (1986a)	20 × 8	3	0.852 46	4.06	78.11	0.852 46	0.00	0.36
13	Chandrasekharan and Rajagopalan (1986a)	20 × 8	4	0.819 67	5.27	67.63	0.819 67	0.00	1.14
14	Chandrasekharan and Rajagopalan (1986b)	20 × 8	2	0.587 16	2.32	9.77	0.587 16	0.08	0.42
14	Chandrasekharan and Rajagopalan (1986b)	20 × 8	3	0.575 47	5.34	17.46	0.575 47	5.00	16.27
14	Chandrasekharan and Rajagopalan (1986b)	20 × 8	4	0.567 31	7.49	6.02	0.567 31	30.78	25.75
15	Yang and Yang (2008, Fig. 6b))	15 × 15	2	0.495 65	3.43	16.94	0.495 65	0.03	1.75
15	Yang and Yang (2008, Fig. 6b))	15 × 15	3	0.67089	6.61	41.25	0.670 89	0.22	8.61
15	Yang and Yang (2008, Fig. 6b))	15 × 15	4	0.870 97	9.19	78.05	0.870 97	0.00	1.63
15	Yang and Yang (2008, Fig. 6b))	15 × 15	5	0.838 71	10.02	33.84	0.838 71	0.02	9.05
16	Yang and Yang (2008, Fig. 6c)	15 × 15	2	0.418 18	2.89	25.18	0.418 18	0.05	0.80
16	Yang and Yang (2008, Fig. 6c)	15 × 15	3	0.592 11	6.42	34.81	0.592 11	0.50	3.78
16	Yang and Yang (2008, Fig. 6c)	15 × 15	4	0.81818	9.24	75.66	0.818 18	0.03	2.44
16	Yang and Yang (2008, Fig. 6c)	15 × 15	5	0.814 81	10.11	20.98	0.814 81	0.06	10.20
17	Yang and Yang (2008, Fig. 6d)	15 × 15	2	0.417 39	3.36	12.39	0.417 39	0.10	3.28
17	Yang and Yang (2008, Fig. 6d)	15 × 15	3	0.576 92	7.15	34.64	0.576 92	1.44	35.83
17	Yang and Yang (2008, Fig. 6d)	15 × 15	4	0.725 81	10.41	77.07	0.725 81	0.68	61.11
17	Yang and Yang (2008, Fig. 6d)	15 × 15	5	0.721 31	11.23	18.70	0.721 31	2.44	409.97
18	Vakharia and Wemmerlöv (1990)	19 × 12	2	0.495 80	3.18	42.62	0.495 80	0.06	1.36
18	Vakharia and Wemmerlöv (1990)	19 × 12	3	0.548 08	6.54	7.87	0.548 08	4.59	43.66
18	Vakharia and Wemmerlöv (1990)	19 × 12	4	0.576 47	10.42	14.97	0.576 47	28.14	84.17
18	Vakharia and Wemmerlöv (1990)	19 × 12	5	0.578 31	13.19	25.34	0.578 31	78.49	823.67
18	Vakharia and Wemmerlöv (1990)	19 × 12	6	0.573 17	14.76	2.23	0.573 17	150.83	2058.53
18	Vakharia and Wemmerlöv (1990)	19 × 12	7	0.571 43	15.83	0.07	0.571 43	212.43	11 915.53
18	Vakharia and Wemmerlöv (1990)	19 × 12	8	0.566 27	16.90	0.02	0.566 27	339.57	15 962.61
19	De Witte (1980)	19 × 12	2	0.537 82	3.45	96.84	0.537 82	0.03	1.23
19	De Witte (1980)	19 × 12	3	0.565 66	6.93	33.91	0.565 66	3.11	20.64
19	De Witte (1980)	19 × 12	4	0.577 32	10.46	26.23	0.577 32	31.95	93.14
19	De Witte (1980)	19 × 12	5	0.576 09	12.26	2.45	0.576 09	102.13	1608.95

(Continued on next page)

Table 3. Results for selected cell formation matrices from the literature with $26 \leq n + m \leq 38$ *(Continued)*

Matrix number	*Source matrix*	*Size*	*Cells*	*Relocation heuristic*			*Branch-and-bound*		*CPLEX time*
				Γ_{LB}	*Time*	*Attraction*	Γ^*	*Time*	
19	De Witte (1980)	19 × 12	6	0.574 71	14.24	0.70	0.574 71	156.92	1525.63
19	De Witte (1980)	19 × 12	7	0.571 43	16.16	0.34	0.571 43	236.64	8206.16
19	De Witte (1980)	19 × 12	8	0.564 71	18.00	0.13	0.564 71	413.84	77 435.77
20	Seifoddini (1988)	11 × 22	2	0.539 68	3.38	61.60	0.539 68	0.04	0.91
20	Seifoddini (1988)	11 × 22	3	0.673 68	7.47	63.00	0.673 68	0.48	2.42
20	Seifoddini (1988)	11 × 22	4	0.695 12	9.89	8.33	0.695 12	3.72	10.92
20	Seifoddini (1988)	11 × 22	5	0.700 00	11.34	13.20	0.700 00	8.69	110.56
20	Seifoddini (1988)	11 × 22	6	0.692 31	12.58	1.93	0.692 31	16.90	150.20
20	Seifoddini (1988)	11 × 22	7	0.688 31	13.94	0.31	0.688 31	26.20	857.23
20	Seifoddini (1988)	11 × 22	8	0.671 05	15.46	0.02	0.671 05	73.50	2268.22
21	Askin and Subramanian (1987)	23 × 14	6	0.728 57	17.86	11.11	0.728 57	178.71	393.33
21	Askin and Subramanian (1987)	23 × 14	7	0.742 42	20.71	5.15	0.742 42	186.38	689.84
21	Askin and Subramanian (1987)	23 × 14	8	0.742 42	22.28	0.52	0.742 42	268.82	5371.77
22	Stanfel (1985)	24 × 14	6	0.716 22	17.28	6.70	0.716 22	673.95	1859.75
22	Stanfel (1985)	24 × 14	7	0.728 57	19.46	1.92	0.728 57	809.21	2807.55
22	Stanfel (1985)	24 × 14	8	0.728 57	21.71	0.30	0.728 57	1148.61	14 537.02

interval ($3 \leq K \leq 8$), and the branch-and-bound computation times ranged from 0.48 to 73.50 seconds (monotonically increasing) over the same range for K. Now, compare the results for matrix 20 to those obtained for matrix 19 (De Witte, 1980), which is slightly smaller. The optimal values of the GEI for matrix 19 ranged from 0.56 to 0.58 over the interval ($3 \leq K \leq 8$), and the branch-and-bound computation times ranged from 3.11 to 413.84 seconds (monotonically increasing) over the same range for K. Clearly, matrix 19 was more difficult than the larger matrix 20 because of its ill-structuredness, as evidenced by its lower GEI.

The MILP computation times in Table 3 exhibit tremendous variation, ranging from 0.19 seconds to approximately 21.5 hours. The MILP computation times exceed the branch-and-bound algorithm computation times for 44 of the 45 problems (matrix 14 at $K = 4$ is the exception). Moreover, it is not uncommon for the MILP computation times to be one to two orders of magnitude greater than the branch-and-bound computation times. As an illustration of the size of an MILP formulation, we consider matrix 20, which is 11 × 22 (i.e., 33 objects), at $K = 5$ cells. The problem has $33(5) = 165$ integer variables, potentially $11(22)(5)(5) = 6050$ z_{ijkl} variables, $5 + 11 + 22 = 38$ cell assignment constraints, one constraint on the GEI (constraint (21)), and potentially $11(22)(5)(5) = 6050$ linearization constrains in Equation (14). Given that 3700 of the η_{ijkl} parameters are zero, 3700 z_{ijkl} variables and 3700 linearization constraints are unnecessary, resulting in an MILP consisting of 2515 variables (165 binary and 2350 continuous) and 2389 (38 + 1 + 2350) constraints. Our MILP approach confirmed the optimality of the relocation heuristic solution in 110.56 seconds. By comparison, we also confirmed optimality using the Dinkelbach (1967) formulation described by Crama and Oosten (1996), which required 696.73 seconds.

The results for the largest part–machine matrices in our study, which ranged in size from $38 \leq (n + m) \leq 64$, are reported in Table 4. Table 4 conveys that the multistart relocation heuristic again identified the globally optimal partition for all considered values of K for each matrix. The ability of the rather straightforward relocation heuristic to obtain the globally optimal partition for every single problem in the test suite is, in itself, a useful finding. It suggests that, for many test problems in the literature, sophisticated metaheuristics might not be required because a more traditional local-search heuristic based on transfers or relocations of objects across cells can also obtain a global optimum provided that the number of restarts is sufficiently large.

The computation times for the branch-and-bound algorithm displayed in Table 4 exhibit dramatic variability. Moreover, the computational limitations of the algorithm become apparent, as some of the reported solution times in Table 4 are enormous. For some of the well-structured (e.g., $\Gamma^* \approx 0.7$ or greater) part–machine matrices (e.g., matrices 27, 28, 29, 30) in Table 3, the computation times are quite reasonable. Nevertheless, the results for part–machine incidence matrices 23 through 26 reveal that the branch-and-bound algorithm can require significant computation time for matrices with roughly $(n + m) = 40$ objects if they are ill-structured. Although a definitive threshold between well- and ill-structured is difficult to pinpoint, most of the

Table 4. Results for selected cell formation matrices from the literature with $n + m \geq 38$

Problem number	*Source matrix*	*Size*	*Cells*	*Relocation heuristic*			*Branch-and-bound*	
				Γ_{LB}	*Time*	*Attraction*	Γ^*	*Time*
23	Lesowsky et al. (1987)	13 × 25	7	0.539 47	6.94	2.54	0.539 47	68 889.78
24	McCormick et al. (1972, Fig. 8)	24 × 16	8	0.538 46	12.48	13.99	0.538 46	24 716.34
25	Carrie (1973, Fig. 4)	24 × 18	9	0.577 32	13.54	2.04	0.577 32	20 840.55
26	Kumar et al. (1986)	23 × 20	7	0.508 06	12.65	3.01	0.508 06	1375 608.66
27	Srinivasan et al. (1990)	16 × 30	5	0.693 43	7.67	71.83	0.693 43	298.88
27	Srinivasan et al. (1990)	16 × 30	6	0.699 25	8.57	31.52	0.699 25	429.91
27	Srinivasan et al. (1990)	16 × 30	7	0.699 25	9.23	6.59	0.699 25	551.14
28	Carrie (1973, Fig. 1)	35 × 20	5	0.783 13	10.73	79.53	0.783 13	361.83
28	Carrie (1973, Fig. 1)	35 × 20	6	0.788 82	12.26	25.71	0.788 82	1354.69
28	Carrie (1973, Fig. 1)	35 × 20	7	0.788 82	14.05	3.04	0.788 82	3113.48
29	Chandrasekharan and Rajagopalan (1989, dataset 1)	40 × 24	7	1.000 00	15.15	82.53	1.000 00	0.00
30	Chandrasekharan and Rajagopalan (1989, dataset 2)	40 × 24	7	0.851 06	22.23	98.70	0.851 06	42.29
31	Chandrasekharan and Rajagopalan (1989, dataset 3)	40 × 24	7	0.735 10	20.20	85.84	0.735 10	208 158.02

easier problems in Table 4 have Γ^* values of roughly 0.7 or above. By contrast, most of the problems requiring enormous computation times have Γ^* values of 0.6 or below. A good illustration is afforded by comparing the $K = 7$ results for matrix 22 in Table 3 to those for matrix 23 in Table 4, which both have $(n + m) = 38$ objects. The optimal solution for matrix 23 ($\Gamma^* = 0.539\ 47$) required roughly 19 hours of computation time, whereas only about 3 minutes was needed for matrix 22 ($\Gamma^* = 0.742\ 42$). The enormous computation time of 1375 608.66 seconds for matrix 26 ($n + m = 43$, $\Gamma^* = 0.508\ 06$) further exemplifies the dire effects of poor structure on computation time. Moreover, the 208 158.02 seconds required for matrix 31 ($\Gamma^* = 0.735\ 10$) reveals that an even greater degree of structure is needed for efficient solution of larger problems.

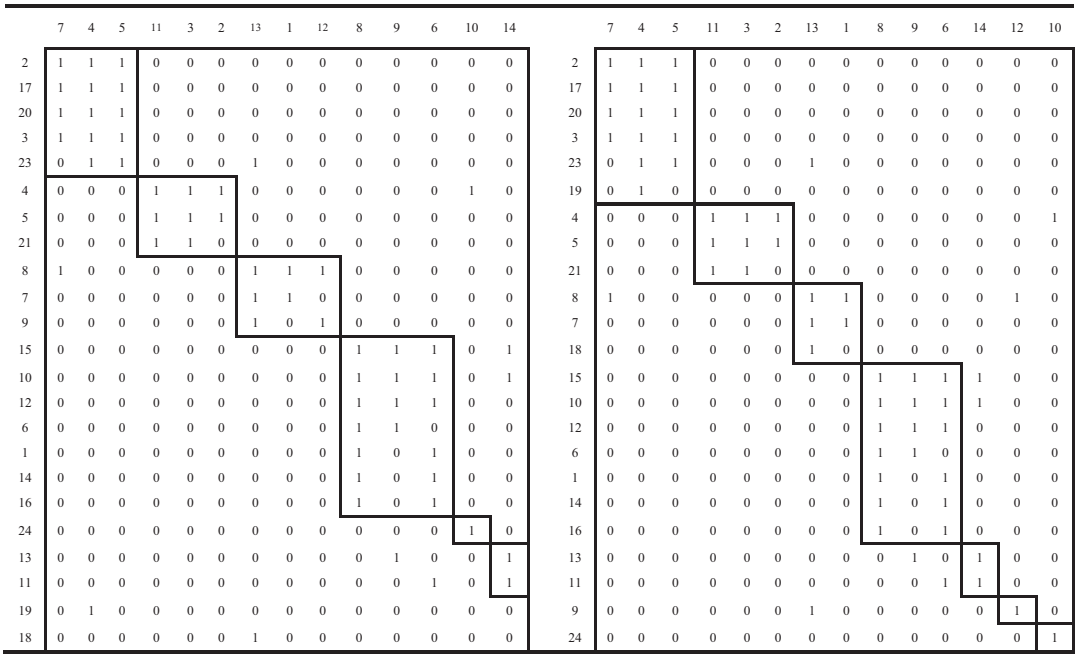
5.4. *Follow-up experiments to assess the quality of the bounds*

Two follow-up experiments were conducted to assess issues pertaining to the quality of the initial lower bound. In the first follow-up experiment, the branch-and-bound algorithm was reapplied to matrices 1 through 22 using initial lower bounds obtained from only 100 (instead of 100 000) restarts of the relocation heuristic. When using only 100 restarts, the maximum computation time for the relocation heuristic across the 91 problems in Tables 2 and 3 was only 0.03 seconds.

The effect of the reduction of the number of restarts on the computation time of the branch-and-bound algorithm was found to be negligible. Despite the fact that only 100 restarts were permitted, the relocation heuristic obtained the globally optimal value of the GEI for 84 of the 91 test problems spanning Tables 2 and 3, with a maximum deviation of 4.17% (for matrix 19 at $K = 8$ cells). For the seven test problems where the initial lower bound failed to match the globally optimal objective function value, the impact on branch-and-bound computation time was subtle. The ratio of the branch-and-bound computation time when using 100 restarts to obtain Γ_{LB} to the branch-and-bound computation time when using 100 000 restarts to obtain Γ_{LB} ranged from only 1.06:1 to 1.38:1 for these seven problems.

In light of the fact that using only 100 restarts of the relocation heuristic did not provide serious damage to the quality of the initial lower bounds, we completed a second follow-up experiment whereby a more serious and systematic degradation in the quality of the initial lower bound was enforced. In this second follow-up study, the lower bound obtained by the relocation heuristic (100 000 restarts) was reduced by 10% prior to implementation of the branch-and-bound algorithm. For example, if the relocation heuristic obtained a GEI of 0.6, the branch-and-bound algorithm was only supplied with an initial lower bound of $\Gamma_{\mathrm{LB}} = 0.54$.

The effects of the 10% degradation in the quality of the initial lower bound (Γ_{LB}) on the performance of the branch-and-bound algorithm were more noteworthy. For the 47 test problems in Tables 2 and 3 with branch-and-bound times of 0.03 seconds or greater, the ratio of branch-and-bound computation time using Γ_{LB} equal to a 10% reduction of the relocation heuristic objective function to branch-and-bound computation time using Γ_{LB} as the raw relocation heuristic objective function (which was the global optimum in all cases) ranged from 1:1 to 15:1. However, in only eight instances did the ratio exceed 4:1. The largest computation time ratio of 15:1 occurred for matrix 21 at $K = 7$ cells, where the branch-and-bound computation time using the raw lower bound (equal to the global



	7	4	5	11	3	2	13	1	12	8	9	6	10	14
2	1	1	1	0	0	0	0	0	0	0	0	0	0	0
17	1	1	1	0	0	0	0	0	0	0	0	0	0	0
20	1	1	1	0	0	0	0	0	0	0	0	0	0	0
3	1	1	1	0	0	0	0	0	0	0	0	0	0	0
23	0	1	1	0	0	0	1	0	0	0	0	0	0	0
4	0	0	0	1	1	1	0	0	0	0	0	0	1	0
5	0	0	0	1	1	1	0	0	0	0	0	0	0	0
21	0	0	0	1	1	0	0	0	0	0	0	0	0	0
8	1	0	0	0	0	0	1	1	1	0	0	0	0	0
7	0	0	0	0	0	0	1	1	0	0	0	0	0	0
9	0	0	0	0	0	0	1	0	1	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	1	1	1	0	1
10	0	0	0	0	0	0	0	0	0	1	1	1	0	1
12	0	0	0	0	0	0	0	0	0	1	1	1	0	0
6	0	0	0	0	0	0	0	0	0	1	1	0	0	0
1	0	0	0	0	0	0	0	0	0	1	0	1	0	0
14	0	0	0	0	0	0	0	0	0	1	0	1	0	0
16	0	0	0	0	0	0	0	0	0	1	0	1	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	1	0
13	0	0	0	0	0	0	0	0	0	0	1	0	0	1
11	0	0	0	0	0	0	0	0	0	0	0	1	0	1
19	0	1	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	1	0	0	0	0	0	0	0

	7	4	5	11	3	2	13	1	8	9	6	14	12	10
2	1	1	1	0	0	0	0	0	0	0	0	0	0	0
17	1	1	1	0	0	0	0	0	0	0	0	0	0	0
20	1	1	1	0	0	0	0	0	0	0	0	0	0	0
3	1	1	1	0	0	0	0	0	0	0	0	0	0	0
23	0	1	1	0	0	0	1	0	0	0	0	0	0	0
19	0	1	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	1	1	1	0	0	0	0	0	0	0	1
5	0	0	0	1	1	1	0	0	0	0	0	0	0	0
21	0	0	0	1	1	0	0	0	0	0	0	0	0	0
8	1	0	0	0	0	0	1	1	0	0	0	0	1	0
7	0	0	0	0	0	0	1	1	0	0	0	0	0	0
18	0	0	0	0	0	0	1	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	1	1	1	1	0	0
10	0	0	0	0	0	0	0	0	1	1	1	1	0	0
12	0	0	0	0	0	0	0	0	1	1	1	1	0	0
6	0	0	0	0	0	0	0	0	1	1	1	1	0	0
1	0	0	0	0	0	0	0	0	1	1	0	0	0	0
14	0	0	0	0	0	0	0	0	1	0	1	0	0	0
16	0	0	0	0	0	0	0	0	1	0	1	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	1	0	0
11	0	0	0	0	0	0	0	0	0	0	0	1	1	0
9	0	0	0	0	0	0	0	0	1	0	0	0	0	1
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fig. 2. Two (seven-cell) partitions of the part–machine matrix from Askin and Subramanian (1987, p. 111). The seven-cell partition on the left is optimal if residual cells are permitted. The seventh cell of this partition is residual because it contains two parts (19 and 18) but no machines. By contrast, the partition on the right is optimal if residual cells are forbidden. Note that the partition on the right does not contain any cells that have parts but no machines (or *vice versa*) but does have two cells with only one part and one machine.

optimum) was 186.38 seconds, but the computation time when this bound was reduced by 10% was 2793.34 seconds.

The findings obtained from the two follow-up experiments can be succinctly summarized as follows: (i) the reduction of the number of restarts of the relocation heuristic from 100 000 to 100 did not seriously degrade its performance in most instances; (ii) the modest reduction in the quality of the initial lower bound that occurred in some instances where fewer restarts were permitted did not markedly increase the computational burden for the branch-and-bound algorithm; and (iii) the more severe degradation (10% reduction) of the lower bound enforced in the second follow-up study did, in some instances, substantially increase the computation time associated with the branch-and-bound algorithm. Together, these findings suggest that there is merit to establishing a reasonable lower bound; however, it does not need to be perfect (i.e., the global optimum) to realize the benefits of the branch-and-bound algorithm.

5.5. *The impact of residual cells*

As a final consideration, we investigate the impact of residual cells on the solution to part–machine clustering problems. As noted previously, the permission of residual cells can often afford an improvement (i.e., increase) in the GEI by reducing the number of voids. To illustrate this circumstance, we expound on matrix 21 (Askin and Subramanian, 1987). The left-hand-side of Fig. 2 displays the globally optimal seven-cluster partition for this matrix when residual cells are permitted. The residual cell contains parts 18 and 19 but no machines. This partition has eight voids and nine exceptional elements and yields an optimal GEI of $\Gamma^* = 0.742\,42$. By contrast, the right-hand side of Fig. 2 displays the globally optimal seven-cluster partition for this matrix when residual cells are forbidden. This partition contains two cells, each having exactly one part and one machine; however, there are no cells with only parts (or only machines). Thus, by imposing a constraint to forbid residual cells, the number of voids is now nine, as is the number of exceptional elements, and the optimal GEI is reduced to $\Gamma^* = 0.731\,34$.

6. Conclusions

6.1. *Summary*

Numerous metaheuristics have been designed for the part–machine clustering problem; however, the development of efficient exact methods has received considerably less attention. We have presented a standalone branch-and-bound algorithm for part–machine clustering based on the maximization of the GEI (Kumar and Chandrasekharan, 1990). The algorithm was applied to 104 test problems

spanning 31 part–machine incidence matrices from the manufacturing cell formation literature. The results can be summarized as follows.

1. The branch-and-bound algorithm efficiently obtained globally optimal partitions for test problems where $(n + m) \leq 33$, even for ill-structured incidence matrices where the optimal GEI, Γ^*, is 0.6 or less.
2. For incidence matrices where $38 \leq (n + m) \leq 46$, the branch-and-bound algorithm was reasonably efficient for test problems where $\Gamma^* \approx 0.7$ or greater; however, drastic increases in computation were observed for incidence matrices of this size when $\Gamma^* \leq 0.6$.
3. Exact solutions were also efficiently obtained by the branch-and-bound algorithm for incidence matrices of size $55 \leq (n+m) \leq 64$ when $\Gamma^* \geq 0.75$. The algorithm is not computationally feasible for ill-structured matrices of this size.
4. The MILP approach efficiently obtained the globally optimal solution for incidence matrices of size $(n + m) \leq 30$, but generally required more computation time than the standalone branch-and-bound algorithm. The computation time advantages of the branch-and-bound algorithm relative to the MILP approach were often substantial for $(n + m) > 30$, particularly for larger values of K.
5. The relocation heuristic identified the globally optimal partition for all test problems in our experimental analyses. Nevertheless, it is important to recognize that this claim could not be made unless the exact approaches were able to verify the optimality of the heuristic solutions.
6. A good practical strategy would be to first run the efficient relocation heuristic for a selected range for the number of cells (K). Subsequently, the branch-and-bound algorithm could be used to obtain verifiably optimal solutions for promising values of K.
7. The branch-and-bound algorithm can also be used to establish the superiority of a selection of K cells to other options for K. For example, suppose that the optimal partition for $K = 7$ yields $\Gamma^* = 0.75$. The algorithm can then be implemented for some other value of K (e.g., $K = 6$) using $\Gamma_{LB} = 0.75$ as the initial lower bound. If the resulting problem is infeasible, then it is verified that $K = 6$ cannot produce a better GEI than $K = 7$.

6.2. *Limitations and extensions*

Like most exact solution methods, the principal limitation of the proposed branch-and-bound algorithm is scalability. As noted in Section 6.1, the algorithm is generally robust for problems with $(n + m) \leq 35$. However, larger matrices, particularly those that are ill-structured, can present a formidable challenge for the branch-and-bound algorithm. The implications of this finding are twofold. First, it is evident that the development of improved bounding procedures would provide an important extension to the proposed method. Second, the development of exact solution procedures clearly does not obviate the need for good heuristic methods that can be successfully applied to large or ill-structured matrices.

In addition to scalability, another area of concern associated with the proposed branch-and-bound method is its flexibility to accommodate objective functions other than the GEI. The algorithm could be adapted easily for direct minimization of the total number of voids and exceptional elements (or a weighted function of these quantities). It is also extensible to the special case of grouping efficiency (Kumar and Chandrasekharan, 1990) studied by Crama and Oosten (1996). At the present time, the proposed branch-and-bound method is not adaptable for objective functions that incorporate information concerning the relative sizes of the blocks. Capturing block size information in the process of constructing the lower bound for a partial assignment is nontrivial. Extension of the proposed method for these objective criteria would also afford a valuable contribution.

Finally, the adaptation of the branch-and-bound procedures for extensions of the cell formation problem also presents an opportunity for future research. For example, one useful adaptation is the re-design of the proposed algorithm for the case of identical machines (Xambre and Vilharino, 2003). More sophisticated extensions might include the development of branch-and-bound procedures for cell formation problems that also incorporate decisions pertaining to lot-sizing (Defersha and Chen, 2008) or routing (Lee and Chiang, 2002; Nsakanda *et al*., 2006).

Acknowledgement

I am grateful to two anonymous reviewers and the Associate and Department Editors for their helpful comments that led to significant improvements in this article.

References

Adil, G.K., Rajamani, D., and Strong, D. (1997) Assignment allocation and simulated annealing algorithms for cell formation. *IIE Transactions*, **29**(1), 53–67.

Anvari, M., Saidi-Mehrabad, M., and Barzinpour, F. (2010) Machine-part cell formation using a hybrid particle swarm optimization. *International Journal of Advanced Manufacturing Technology*, **47**(5–8), 745–754.

Askin, R.G. (2013) Contributions to the design and analysis of cellular manufacturing systems. *International Journal of Production Research*, **51**(23–24), 6778–6787.

Askin, R.G., Cresswell, S.H., Goldberg, J.B., and Vakharia, A.J. (1991) A Hamiltonian path approach to reordering the part–machine matrix for cellular manufacturing. *International Journal of Production Research*, **29**(6), 1081–1100.

Askin, R.G. and Subramanian, S.P. (1987) A cost-based heuristic for group technology configuration. *International Journal of Production Research*, **25**(1), 101–113.

Banfield, C.F. and Bassill, L.C. (1977) Algorithm AS 113: a transfer algorithm for non-hierarchical classification. *Applied Statistics*, **26**(2), 206–210.

Boctor, F.F. (1991) A linear formulation of the machine-part cell formation problem. *International Journal of Production Research*, **29**(2), 343–356.

Brusco, M., Doreian, P., Lloyd, P., and Steinley, D. (2013) A variable neighborhood search method for a two-mode block modeling problem in social network analysis. *Network Science*, **1**(2), 191–212.

Brusco, M.J. and Stahl, S. (2005) *Branch-and-Bound Applications in Combinatorial Data Analysis*, Springer, New York, NY.

Brusco, M.J. and Steinley, D. (2007) Exact and approximate algorithms for part-machine clustering based on a relationship between interval graphs and Robinson matrices. *IIE Transactions*, **39**(10), 925–935.

Brusco, M.J. and Steinley, D. (2009) Integer programs for one- and two-mode blockmodeling based on prespecified image matrices for structural and regular equivalence. *Journal of Mathematical Psychology*, **53**(6), 577–585.

Burbidge, J.L. (1963) Production flow analysis. *Production Engineer*, **42**(12), 742–752.

Bychkov, I., Batsyn, M., and Pardalos, P.M. (2014) Exact model for the cell formation problem. *Optimization Letters*, **8**(8), 2203–2210.

Carrie, A.S. (1973) Numerical taxonomy applied to group technology and plant layout. *International Journal of Production Research*, **11**(4), 399–416.

Chan, H.M. and Milner, D.A. (1982) Direct clustering algorithm for group formation in cellular manufacturing. *Journal of Manufacturing Systems*, **1**(1), 65–74.

Chandrasekharan, M.P. and Rajagopalan, R. (1986a) An ideal seed non-hierarchical clustering algorithm for cellular manufacturing. *International Journal of Production Research*, **24**(2), 451–464.

Chandrasekharan, M.P. and Rajagopalan, R. (1986b) MODROC: an extension of rank order clustering for group technology. *International Journal of Production Research*, **24**(5), 1221–1233.

Chandrasekharan, M.P. and Rajagopalan, R. (1989) Groupability: an analysis of the properties of binary data matrices for group technology. *International Journal of Production Research*, **27**(6), 1035–1052.

Crama, Y. and Oosten, M. (1996) Models for machine-part grouping in cellular manufacturing. *International Journal of Production Research*, **34**(6), 1693–1713.

Defersha, F.M. and Chen, M. (2008) A linear programming embedded genetic algorithm for an integrated cell formation and lot sizing considering product quality. *European Journal of Operational Research*, **187**(1), 46–69.

De Witte, J. (1980) The use of similarity coefficients in production flow analysis. *International Journal of Production Research*, **18**(4), 503–514.

Dinkelbach, W. (1967) On non-linear fractional programming. *Management Science*, **13**(7), 492–498.

Doreian, P., Batagelj, V, and Ferligoj, A. (2005) *Generalized Block Modeling*, Cambridge University Press, Cambridge, UK.

Elbenani, B. and Ferland, J.A. (2012) Cell formation problem solved exactly with the Dinkelbach algorithm. Report, CIRRELT-2012-07.

Ghosh, T., Sengupta, S., Chattopadhyay, M. and Dan, P.D. (2011) Metaheuristics in cellular manufacturing: a state-of-the-art review. *International Journal of Industrial Engineering Computations*, **2**(1), 87–122.

Gonçalves, J.F., and Resende, M..C. (2004) An evolutionary algorithm for manufacturing cell formation. *Computers and Industrial Engineering*, **47**(2-3), 247–273.

Heragu, S.S. and Kakuturi, S.R. (1997) Grouping and placement of machine cells. *IIE Transactions*, **29**(7), 561–571.

Jancey, R.C. (1966) Multidimensional group analysis. *Australian Journal of Botany*, **14**, 127–130.

King, J.R. (1980) Machine-component grouping in production flow analysis: an approach using rank-order clustering algorithm. *International Journal of Production Research*, **18**(2), 213–232.

King, J.R. and Nakornchai, V. (1982) Machine-component group formation technology: review and extension. *International Journal of Production Research*, **20**(2), 117–133.

Klein, G. and Aronson, J.E. (1991) Optimal clustering: a model and method. *Naval Research Logistics*, **38**(5), 447–461.

Kumar, C.S. and Chandrasekharan, M.P. (1990) Grouping efficacy: a quantitative criterion for goodness of block diagonal forms of binary matrices in group technology. *International Journal of Production Research*, **28**(2), 233–243.

Kumar, K.R., Kusiak, A. and Vannelli, A. (1986) Grouping of parts and components in flexible manufacturing systems. *European Journal of Operational Research*, **24**, 387–297.

Kusiak, A. and Cho, M. (1992) Similarity coefficient algorithm for solving the group technology problem. *International Journal of Production Research*, **30**(11), 2633–2646.

Kusiak, A. and Chow, W.S. (1987) Efficient solving of the group technology problem. *Journal of Manufacturing Systems*, **6**(2), 117–124.

Lee, S.-D. and Chiang, C.-P. (2002) Cell formations in the uni-directional loop material handling environment. *European Journal of Operational Research*, **137**(2), 401–420.

Lei, D. and Wu, Z. (2005) Tabu search approach based on a similarity coefficient for cell formation in generalized group technology. *International Journal of Production Researc*h, **43**(19), 4035–4047.

Lesowsky, Z., Logan, L. and Vanelli, A. (1987) Group technology decision aids in an expert system for plant layout, in *Modern Production Management Systems: Proceedings of the IFIP TX 5/WG 5-7 Working Conference on Advances in Production Management Systems*, Kusiak, A. (ed), Elsevier, Amsterdam, The Netherlands pp. 561–571.

Li, X., Baki, M.F. and Aneja, Y.P. (2010) An ant colony optimization metaheuristic for machine-part cell formation problems. *Computers and Operations Research*, **37**(12), 2071–2081.

Madeira, S.C. and Oliveira, A.L. (2004) Biclustering algorithms for biological data analysis: a survey. *IEEE Transactions in Computational Biology and Bioinformatics*, **1**(1), 24–45.

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

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

Mitrofanov, S.P. (1966) *The Scientific Principles of Group Technology*. Boston Spa, UK (National Lending Library Translation).

Mosier, C.T. and Taube, L. (1985) The facets of group technology and their impact on implementation. *Omega*, **13**(5), 381–391.

Nair, G.J.K. and Narendran, T.T. (1996) Grouping index: a new quantitative criterion for goodness of block diagonal forms in group technology. *International Journal of Production Research*, **34**, 2767–2782.

Nsakanda, A.L., Diaby, M. and Price, W.L. (2006) Hybrid genetic approach for solving large-scale capacitated cell formation problems with multiple routings. *European Journal of Operational Research*, **171**(3), 1051–1070.

Papaioannou, G. and Wilson, J.M. (2010) The evolution of cell formation problem methodologies based on recent studies (1997–2008): review and directions for future research. *European Journal of Operational Research*, **206**(3), 509–521.

Paydar, M.M. and Saidi-Mehrabad, M. (2013) A hybrid-genetic-variable neighborhood search algorithm for the cell formation problem based on grouping efficacy. *Computers and Operations Research*, **40**(4), 980–990.

Sarker, B.R. (2001) Measures of grouping efficiency in cellular manufacturing systems. *European Journal of Operational Research*, **130**(3), 588–611.

Sarker, B.R. and Khan, M. (2001) A comparison of existing grouping efficiency measures and a new weighted grouping efficiency measure. *IIE Transactions*, **33**(1), 11–27.

Sarker, B.R. and Mondal, S. (1999) Grouping efficiency measures in cellular manufacturing: a survey and critical review. *International Journal of Production Research*, **37**(2), 285–314.

Seifoddini, H. (1988) Comparison between single linkage and average linkage clustering techniques in forming machine cells. *Computers and Industrial Engineering*, **15**(1-4), 210–216.

Seifoddini, H. (1989) A note on the similarity coefficient method and the problem of improper machine assignment in group technology applications. *International Journal of Production Research*, **27**(7), 1161–1165.

Seifoddini, H. and Wolfe, P.M. (1986) Application of the similarity coefficient method in group technology. *IIE Transactions*, **18**(3), 271–277.

Selim, H.M., Askin, R.G. and Vakharia, A.J. (1998) Cell formation in group technology: review, evaluation and directions for future research. *Computers and Industrial Engineering*, **34**(1), 3–20.

Shargal, M., Shekhar, S. and Irani, S.A. (1995) Evaluation of search algorithms and clustering efficiency measures for machine-part matrix clustering. *IIE Transactions*, **27**(1), 43–59.

Shtub, A. (1989) Modeling group technology cell formation as a generalized assignment problem. *International Journal of Production Research*, **27**(5), 775–782.

Srinivasan, G., Narendran, T.T. and Mahadevan, B. (1990) An assignment model for the part-families problem in group technology. *International Journal of Production Research*, **28**(1), 145–152.

Stanfel, L.E. (1985) Machine clustering for economic production. *Engineering Costs and Production Economics*, **9**(1–3), 73–81.

Tariq, A., Hussain, I. and Ghafoor, A. (2009) A hybrid genetic algorithm for machine part grouping. *Computers and Industrial Engineering*, **56**(1), 347–356.

Vakharia, A.J. and Wemmerlöv, U. (1990) Designing a cellular manufacturing system: a materials flow approach based on operations sequences. *IIE Transactions*, **22**(1), 84–97.

Van Rosmalen, J., Groenen, P.J.F., Trejos, J. and Castillo, W. (2009) Optimization strategies for two-mode partitioning. *Journal of Classification*, **26**(2), 155–181.

Ventura, J.A., Chen, F.F. and Wu, C.H. (1990) Grouping parts and tools in flexible manufacturing systems production planning. *International Journal of Production Research*, **28**(6), 1039–1056.

Waghodekar, P.H. and Sahu, S. (1984) Machine-component cell formation in group technology MACE. *International Journal of Production Research*, **22**(6), 937–948.

Wilderjans, T.F., Depril, D. and van Mechelen, I. (2013) Additive biclustering: a comparison of one new and two existing ALS algorithms. *Journal of Classification*, **30**(1), 56–74.

Wu, T.H., Chang, C.C. and Chung, S.H. (2008) A simulated annealing algorithm for manufacturing cell formation problems. *Expert Systems with Applications*, **34**(3), 1609–1617.

Wu, T.H., Chung, S.H. and Chang, C.C. (2010) A water flow-like algorithm for manufacturing cell formation problems. *European Journal of Operational Research*, **205**(2), 346–360.

Xambre, A.R. and Vilarinho, P.M. (2003) A simulated annealing approach for manufacturing cell formation with multiple identical machines. *European Journal of Operational Research*, **151**(2), 434–446.

Yang, M.-S. and Yang, J.-H. (2008) Machine-part cell formation in group technology using a modified ART1 method. *European Journal of Operational Research*, **188**(1), 140–152.

Yin, Y. and Yasuda, K. (2006) Similarity coefficient methods applied to the cell formation problem: a taxonomy and review. *International Journal of Production Economics*, **101**(2), 329–352.

Biography

Michael J. Brusco is the Synovus Professor of Business Administration at Florida State University. His research focuses on the development of exact and heuristic solution procedures for discrete optimization problems related to scheduling, layout, social network analysis, and variable selection in multivariate statistics. His work on these and other topics has been published in *IIE Transactions*, *Management Science*, *Naval Research Logistics*, *Operations Research*, *Psychometrika*, *Technometrics*, and other journals.