FOUNDATIONS OF COMPUTING AND DECISION SCIENCES
Vol. 37 (2012) No. 2

DOI: 10.2478/v10209-011-0008-7

# A TABU SEARCH ALGORITHM FOR THE CELL FORMATION PROBLEM WITH PART MACHINE SEQUENCING

G. PAPAIOANNOU [*] [†], J.M. WILSON [*] [‡]

**Abstract.** This paper presents extensions of the IP model where part-machine assignment and cell formation are addressed simultaneously and part machine utilisation is considered. More specifically, an integration of inter-cell movements of parts and machine set-up costs within the objective function, and also a combination of machine set-up costs associated with parts revisiting a cell when the part machine operation sequence is taken into account are examined and an enhanced model is formulated. Based upon this model's requirements, an initial three stage approach is proposed and a tabu search iterative procedure is designed to produce a solution. The initial approach consists of the allocation of machines to cells, the allocation of parts to machines in cells and the evaluation of the objective function's value. Special care has been taken when allocating parts to machine cells as part machine operation sequence is preserved making the system more complex but more realistic. The proposed tabu search algorithm integrates short term memory and an overall iterative searching strategy where two move types, single and exchange, are considered. Computational experiments verified both the algorithm's robustness where promising solutions in reasonably short computational effort are produced and also the algorithm's effectiveness for large scale data sets.

**Keywords:** cell formation, mixed integer linear programming, tabu search, part allocation, part machine operation sequence

[*]School of Business & Economics, Loughborough University, Loughborough, LE11 3TU, UK
[†]Corresponding author. E-mail: g.papaioannou@lboro.ac.uk (G. Papaioannou), Tel no.:+44(0)1509-228281; Fax no.:+44(0)1509-223960
[‡]School of Management & Business, Aberystwyth University, Aberystwyth, SY23 3DD

## 1 Introduction

Cellular manufacturing has been a prosperous research area for the last three decades and received a lot of attention from academicians because of its strategic importance to 'modern' industrial and manufacturing areas. The design of cellular manufacturing systems has been called Cell Formation (CF). CF is the process of grouping parts with similar design features or processing requirements into parts families and machines into machine cells. Extensive reviews of CF problems can be found, for example, in [49, 50], [39], [38] and [34].

Mathematical Programming formulations can be used in a number of circumstances involving a wide range of manufacturing data. Several types of integer programming formulations for the cell formation problem have been proposed over the past years. In most of these formulations parts are assigned to individual machines and individual machines are allocated into cells simultaneously. A number of major results in the literature have addressed the CF problem having as a main criterion the minimisation of intercellular movements and have been discussed by: [35], [28], [29], [48], [36], [54], [38], [47], [14], [2]. In addition, Vakharia and Wemmerlov [44] proposed a cell formation method which integrated the issues of cell formation and within cell material flows. However, none of the studies attempted to handle the minimisation of intercellular movements when machine set-up costs, part/machine utilisation, multiple machines of the same type and part machine operation sequences are taken into account. In our view, and of authors such as Selim [38] this is important and a contribution of the current paper is to solve a more comprehensive CF model than has hitherto been attempted. The aim of this will be to produce a more practical CF system.

Applying mathematical programming models to solve the CF problem, which is known to be a NP-hard problem [15], becomes difficult especially when large scale data sets need to be taken into account. Extensive research has been devoted to the CF problem with many methods developed on the basis of heuristic clustering techniques to obtain near-optimal or good solutions ([5], [32], [9], [23]). Due to their reliable performance in solving other combinatorial optimisation problems, metaheuristic algorithms such as simulated annealing [26], genetic algorithms [20] and recently ant colony optimisation [13] provide another class of search methods that have been adopted to efficiently solve the CF problem with very promising results obtained. Some of these methodologies can be found employed in [52], [21], [46], [30], [33], [22] and [40]. In addition to these search methods Tabu Search (TS) [19], [16] has proved to be a very successful metaheuristic with a number of applications in the area of cellular manufacturing. More specifically a number of papers has been produced each proposing different strategies for CF. For example, in Vakharia and Chang [43] two heuristic methods for the CF problem were proposed based on simulated annealing and tabu search algorithms. The objective function of their model was the minimisation of the total cost of machines required as well as materials handling cost for loads transferred between cells. In Spiliopoulos and Sofianopoulou [41] a three stage approach is proposed for the initial creation of cells where grouping of parts is followed by an elimination of intercellular movements and the development of a TS algorithm later on to obtain a solution. In Wu, Low and Wu [51] two approaches for generating initial solutions were proposed, each followed by a TS algorithm when the main CF operation was the decomposition of the manufacturing shop into several manufacturing cells so that the total part flow within cells could be maximised. As with mathematical programming models, there is no proposed metaheuristic algorithm in the literature combining features such as part machine operation sequence, part/machine utilisation amounts and multiple machines of the same type. All of the latter elements would produce a very realistic and robust system for CF which could be tested for its robustness and effectiveness for large scale problem sizes via an effective higher search methodology, i.e. a metaheuristic.

The aim of the paper is fourfold: first, to produce a comprehensive integer programming (IP) model able to assign parts to machines and machines to cells simultaneously when part/machine utilisation is considered and to minimise jointly the cost of intercellular movements, the set-up cost of machines and the cost of parts revisiting a cell for a later machine operation; second, to propose a three stage heuristic approach for the creation of an initial solution following the IP model requirements; third, to develop a tabu search algorithm for obtaining good solutions; fourth to test the latter on a variety of data sets, especially medium to large problem sizes, and prove its effectiveness.

## 2 Mathematical model: problem statement

The proposed model is an extension of a model developed by Foulds, French and Wilson [14], the first to simultaneously optimise cell formation, machine-cell allocation and part machine cell allocation minimising intercellular movements when multiple machines of the same type were considered. This model was formulated as an integer program but solved heuristically achieving solutions to medium-sized problems within short time limits. In the current study additional elements, such as part/machine set up costs and also costs incurring when parts revisit a cell for a later machine operation are added into the system. Please note that in this study each part is constrained to strictly follow a *machine operation sequence* in order to be fully processed, reflecting real production conditions. Further, the objective function is enhanced to simultaneously minimise the number of distinct allocations of parts to cells, the set-up costs of machines and the number of times a part travels back to an already visited cell. Although the model is able to determine which machines should be allocated to which cells, because machines are only available in limited numbers it may turn out that some parts do not receive all their processing in one cell and have to travel from cell to cell in order to comply with the machine operation sequence. For notation of the model see the corresponding Appendix.

The complete formulation of the mathematical programming model is shown below:

$$Min\ \left(\sum_{j=1}^{NP}\sum_{q=1}^{NC}(M_{j,q}\times w_{j,q})+\sum_{i=1}^{NM}\sum_{j=1}^{NP}(SETUP_{i,j}\times\sum_{q=1}^{NC}s_{i,j,q})+\sum_{q=1}^{NC}\sum_{j=1}^{NP}\sum_{z=1}^{ZTYPES_j}(A_j\times extra_{q,j,L_{j,z}})\right) \quad (1)$$

subject to

$$\sum_{q=1}^{NC} y_{i,k,q} = 1 \quad \forall\ i,k \quad (2)$$

$$\sum_{q=1}^{NC} x_{i,j,q} = UTIL_{i,j} \quad \forall\ i,j \quad (3)$$

$$x_{i,j,q} \le s_{i,j,q} \quad \forall\ i,j,q \quad (4)$$

$$x_{i,j,q} \ge UTILMIN \times s_{i,j,q} \quad \forall\ i,j,q \quad (5)$$

$$\sum_{j=1}^{NP} x_{i,j,q} \le \sum_{k=1}^{KM_i} y_{i,k,q} \quad \forall\ i,q \quad (6)$$

$$\sum_{i=1}^{NM}\sum_{k=1}^{KM_i} y_{i,k,q} \le v_q \times E_{MAX} \quad \forall\ q \quad (7)$$

$$\sum_{i=1}^{NM}\sum_{k=1}^{KM_i} y_{i,k,q} \ge v_q \times E_{MIN} \quad \forall\ q \quad (8)$$

$$x_{i,j,q} \le UTIL_{i,j} \times w_{j,q} \quad \forall\ i,j,q \quad (9)$$

$$x_{i,j,q} \le UTILMAX \times xx_{i,j,q} \quad \forall\ i,j,q \quad (10)$$

$$x_{i,j,q} \ge UTILMIN \times xx_{i,j,q} \quad \forall\ i,j,q \quad (11)$$

$$xx_{L_{j,z},j,q} + xx_{L_{j,r},j,q} - \sum_{zz=z+1}^{r-1} xx_{L_{j,zz},j,q} \le extra_{q,j,L_{j,z}} + 1 \quad \forall\ q,j,z,r \quad (12)$$

$$v_{q+1} \le v_q \quad \forall\ q \quad (13)$$

$$\sum_{q=1}^{NC} q \times y_{i,k,q} \le \sum_{q=1}^{NC} q \times y_{i,k+1,q} \quad \forall\ i,k \quad (14)$$

$$y_{i,k,q},\ v_q,\ w_{j,q},\ extra_{q,j,i},\ xx_{i,j,q} = 0\ or\ 1; \quad 0 \le x_{i,j,q} \le 1; \quad s_{i,j,q}\ \ integer \quad (15)$$

Objective function, 1, combines a mixture of the following three requirements to:

- *Minimise number of distinct cells used by each part*
- *Minimise number of times a part revisits a cell for a later machine operation, thereby, with the above, minimising intercellular travel.*
- *Minimise set-up costs when allocating parts to machines*

It is assumed that $SETUP_{i,j}$ is fixed for a given machine of type $i$ and part $j$ and defined independently of the machine operation sequence. It would be possible to generalise and extend the model by introducing the subscript $z$ in $SETUP_{i,j}$. For simplicity this has not been done.

Constraint (2) ensures that the $k^{th}$ machine of type $i$ is assigned to exactly one cell. Constraint (3) handles the requirements for processing part $j$ on machine $i$: the number of machines or fraction thereof required to process part $j$ in cell $q$ is equal to the utilisation of machine $i$ required to process part $j$ in cell $q$. Constraint (4) ensures that the total number of machines (in terms of machine utilisation) required to process part $j$ in cell $q$ is less than or equal to the integer number of machines of type $i$ that will be used by part $j$ in cell $q$. Constraint (5) forces variable $s$ to get the value 0 whenever $x$ variable is 0, and also variable $x$ to assume values $>= UTILMIN$ as soon as $s$ is non-zero. Although not a strictly necessary constraint, it was found to aid branch and bound. Constraint (6) ensures that the total number of machines of type $i$ used in cell $q$ should be less than or equal to the number of machine instances of type $i$ assigned to cell

$q$. It will be assumed that no cell will contain more than $E_{MAX}$ machines. Constraints (7), (8) restricting the number of machines in each cell are required because space in a cell will be limited (see for instance [38]). Constraint (9) picks which cells are used by parts. Both constraints (10) and (11) ensure that whenever a part uses a machine or a fraction thereof ($x_{i,j,q} >= UTILMIN$) variable $xx_{i,j,q}$ is assigned the value 1, otherwise it is assigned to 0. The *key constraint* (12) picks out the number of times a part travels back to a cell for a later machine operation. A part $j$, whose $z^{th}$ machine operation is processed in cell $q$, could revisit the cell $q$ for a later machine operation i.e. $(z + r)^{th}$, only when the second machine operation $(z + 1)$ is not processed within the same cell. In this case the value of $extra_{q,j,L_{j,z}}$ is assigned to 1. Constraint (13) ensures that cells are formed in successive numerical order. Constraint (14) assigns duplicate machines when needed to lower numbered cells in successive numerical order. Constraints (13) and (14) are included to eliminate certain symmetries and assist branch and bound - all cells are treated as equivalent by an integer programming solver so (13) reduces the need to explore similar trial solutions for every cell. Similarly, (14) reduces the need for the solver to explore trial solutions involving permutation of identical machines.

It is assumed that the machine utilisation for processing a part $j$ is equal to the processing time of part $j$ in machine $i$. For this reason no time element is considered for the current model. Further, for each machine of type $i$ only a maximum of 1.0 units of its capacity can be spent on processing a part $j$. If, for example, the machine utilization of a certain part is more than 1.0 units then more than one machine instance will be used for processing the current part.

Figure 1 provides a visual representation of the CF model which was solved by running `XPRESS-MP` [53], a general purpose mathematical programming solver. The data used are presented in Table 1. Each item in the square boxes, i.e. $M_i^k$, denotes the instance $k$ of machine of type $i$ currently used within a cell. The elements in the arrows indicate machines usage by parts, e.g. 2(0.3) describes that part 2 is using 0.3 capacity units of the machine that the arrow is pointing at. As shown in this figure all parts follow a certain route and it is worth noting that the dotted line represents the route of part 2 which revisits a cell in order to be fully processed.

The IP model is appropriate for smaller problem instances, because they will be solvable, but not for larger instances for which a heuristic approach is needed. The IP model will be used to compare with the heuristic approach (see section 7).

## 3 Proposed initial heuristic approach

The proposed approach for the construction of an initial solution following the requirements of the mathematical model described in section 2 consists of the following phases:

1. Random initial allocation of machines to cells;
2. Allocation of parts to machine cells;
3. Objective function value evaluation.

Obtaining an initial solution is not straightforward, thus the steps are outlined in sections 3.1-3.3. Prior to this the number of the cells created is determined. According to Foulds, French and Wilson [14] the number of cells created could be any integer number between:

$$CELL_{MIN} = \lceil (1/E_{MAX} \times \sum_{i=1}^{NM} KM_i) \rceil$$

and

$$CELL_{MAX} = \lfloor (1/E_{MIN} \times \sum_{i=1}^{NM} KM_i) \rfloor$$

where $KM_i$, as shown in the Appendix as well, is given by the formulae below.

$$KM_i = \lceil \sum_{j=1}^{NP} UTIL_{i,j} \rceil$$

However, it was found in practice, that in the majority of cases the best solutions are produced when the number of cells is set to $CELL_{MIN}$. For simplicity the same scheme is adopted here as well.

## 3.1 Machine cell allocation

This first stage concerns the random grouping of machines into cells and the creation of a machine based cell formation configuration system. Before describing the process of allocating machines to cells the strategy employed for determining the capacity in terms of the number of machine instances that each cell should consist of is described. Each cell should accommodate between $E_{MIN}$ and $E_{MAX}$ number of machines. The capacity of the first chosen cell is randomly generated using a uniform distribution with parameters $[E_{MIN}, E_{MAX}]$ and rounded to the nearest integer. For the second chosen cell its capacity is not randomly generated but is produced from the equation below:

$$cell\ \ capacity = round(\frac{sum\ \ of\ \ all\ \ machine\ \ instances - total\ \ previous\ \ capacity\ \ used}{remaining\ \ cells\ \ to\ \ be\ \ filled\ (including\ \ current)})$$

The same applies for the remaining cells in the system in order for their capacity to be found. For better illustration of cell capacity determination an example follows next. Assume that the total number of machine instances is twenty, $E_{MIN}$ is equal to four, $E_{MAX}$ is equal to six and there are four cells waiting to be filled with machine instances. If for the first cell the machine capacity randomly generated is equal to four then for the second cell its capacity, according to the formulae above, will be equal to five. For the remaining two cells their capacities will be five and six respectively. As can be observed, the total capacity in magnitude is equal to the total number of machine instances in the system, thus all machine instances have been accommodated in the existing cells.

In pseudo terms, the allocation of machines to cells is provided within steps 1-3 below.

1. Find all pairs consisting of machine types and their corresponding instances in the system, list them in ascending order (i.e. machine type 1 and all its instances, machine type 2 and all its instances etc.) and store them in a matrix named $MACHCOORD$;
2. Initialise a 3-D matrix named $CELLMATRIX$ of size $(NM \times KMAX \times NC)$ to hold machine type, machine instance and cell type respectively. Allocate machines to cells:
   (a) Choose cell to fill (store number to avoid filling this cell again);
   (b) Find the capacity for the current cell (as shown earlier);
   (c) Choose randomly a machine instance pair from $MACHCOORD$ (delete pair in order to avoid choosing this pair again) and update $CELLMATRIX$ for corresponding cell;
   (d) Repeat the above step until the corresponding chosen cell is filled with a number of machine instances equal to the capacity determined in 2(b);
   (e) Repeat steps 2(a)-2(d) until all cells are filled;
3. Re-arrange the elements of the $CELLMATRIX$ from step 2 so that duplicate machines are allocated to lowered numbered cells in successive numerical order.

For example, if the data displayed in Table 1 is used, a possible random allocation of all machine instances of all types to the three cells could be as shown below. Please note that first dimension refers to machine type, the second to machine instances and the third to cells.

$$CELLMATRIX(:,:,1) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

$$CELLMATRIX(:,:,2) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

$$CELLMATRIX(:,:,3) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{16}$$

For example, a machine of type 5 has its instances allocated in the three cells as follows:

$$CELLMATRIX(5,:,:) = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \tag{17}$$

Machine instances 1, 2 and 3 are allocated in cell 1, machine instances 4 and 5 in cell 2, whereas 6 and 7 are in cell 3.

## 3.2 Part machine cell allocation

The allocation of parts to machine cells forms the *key* stage of the proposed heuristic algorithm as the *ordered part machine sequence* is taken into account imposing a great restriction on the solution strategy to be developed. Each part is tied to its ordered machine sequence, therefore for each part there is only one route to be followed in order for its operation to be complete. Moreover, for each machine in the operation sequence there is a certain utilisation amount to be used by current part. The above restrictions in conjunction with the elements of the objective function, equation (1), such as intercellular movements, set-up costs and later revisits of parts to already visited cells, plus all remaining model constraints, were taken into account for the design of allocating parts to machine cells.

The procedure of allocating parts to machine cells is outlined below.

1. Initialise a 3-D matrix named $PARTMATRIX$ of size $(NM \times KMAX \times NP)$

to hold for a certain machine instance the allocated part/machine instance utilisation;

2. Sort the parts into ascending order of total processing requirements, i.e. for each part sum up all relevant part/machine utilisation amounts and sort them in ascending order;
3. Let $j$ be the next non-allocated part to be allocated;
4. Identify part $j$'s machine operation sequence;
5. Determine a sequence of cells by identifying a *maximum continuous* machine operation sequence within cells relative to the part machine sequence;
6. Let $d$ be the first machine in current part's machine sequence;
7. Let $q$ be the first cell in sequence;
8. Check if first cell $q$ in the cell sequence, determined in step 5, includes the machine $d$ of the corresponding part machine operation sequence. If it does, find all the instances of current machine in candidate cell otherwise, check the next cell in the cell sequence;
9. Let $k$ be the first machine instance found;
10. Check current part/machine utilisation and also remaining capacity of the relevant machine instance within current cell. Depending upon the values of both, a number of cases are examined, as illustrated below, and allocation commences (assume remaining capacity of a current machine instance is equal to $a$ and current part/machine utilisation is equal to $b$).
    (a) $a < 1 \;\; \& \;\; b >= 1$
    (b) $a < 1 \;\; \& \;\; b < 1 \;\; \& \;\; a <= b$
        (i) $a + b = 1$ | (ii) $a + b > 1$ | (iii) $a + b < 1$
    (c) $a < 1 \;\; \& \;\; a >= b$
        (i) $a + b = 1$ | (ii) $a + b > 1$ | (iii) $a + b < 1$

For the cases where either current part/machine utilisation is greater than or equal to one, i.e. case 10(a), or current part/machine utilisation plus the remaining capacity for current machine instance pair is greater than one, i.e. cases 10(b)(ii) and 10(c)(ii), a certain utilisation amount will be used for allocation. The value of the latter is determined to be equal to one minus the remaining capacity. This utilisation amount is used to update $PARTMATRIX$. The rest of the unused utilisation amount for the current part/machine is stored in a temporary utilisation matrix and the procedure continues with the allocation of current part to another instance of the same machine type. This process will end when the corresponding entry in the temporary utilisation matrix becomes zero. For the remaining cases involved in step 10, the allocation process is straightforward where $PARTMATRIX$ is updated with the part/machine utilisation initially given. Please note that for all cases, once current part machine allocation relative to its machine operation sequence finishes, the procedure continues with the next part in sequence, unless the length of its machine sequence has not been reached yet, i.e. there are still machines for current part to be allocated to, so the process will continue from there.

At each stage where $PARTMATRIX$ is updated two additional elements are also updated. The first named $part\_moves$ stores for each part all its related cell movements together with the corresponding machine instance pairs used relative to the part machine operation sequence. Moreover, in order to have a full reference of the part allocation where all allocated parts and the cells used together with the machine instances pairs employed can be presented, a 4D binary matrix, named $PCMATRIX$ of size $(NM \times KMAX \times NP \times NC)$ was developed. Both of the above elements will be mainly used for the evaluation of the objective function described next but prior to this an example is provided to illustrate the procedure for part allocation.

Assuming here the same data as shown in Table 1, and the machine cell allocation as presented in matrices 16 the first part to allocated will be part 1 since this has the least processing requirements, i.e. 0.4 units from machine of type 5. Part 1 is allocated to the first instance of machine of type 5 since this the first instance to be checked for its availability in cell 1. The actual allocation can be seen in $PARTMATRIX$ below.

$$PARTMATRIX(:,:,1) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{18}$$

$PCMATRIX$ below confirms the allocation of part 1 in cell 1:

$$PCMATRIX(:,:,1,1) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{19}$$

Part 10 is next for allocation. Its sequence is $M_2$, $M_6$ thus a check takes place to examine whether there is a cell that holds both machines in order to avoid any unnecessary intercellular moves. From matrices (16) it can be seen that $M_2$ instances are located in cell 1 and cell 3, whereas the three instances of $M_6$ in cell 2. Hence, an intercellular move is unavoidable. The $PARTMATRIX$ for part 10 looks as follows:

$$PARTMATRIX(:,:,1) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.6 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{20}$$

The $PCMATRIX$ shows that part 10 is allocated in cells 1 and 2 using the first instance of machine of type 2 and the first instance of machine of type 6 respectively:

$$PCMATRIX(:,:,10,1) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

$$PCMATRIX(:,:,10,2) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{21}$$

In a similar way part 3 will be allocated to the first instance of machine of type 3 in cell 1 as that was available. For part 9, which is next in sequence to be allocated, things are slightly different. Part 9 has a machine sequence, $M_3, M_5, M_2$ and it requires 0.1 units from $M_3$, 0.1 a unit from $M_5$ and 0.1 units from $M_2$. Cell 1 holds all machines of its corresponding sequence thus the process starts from cell 1. Part 9 is first allocated to the first instance of machine of type 3 in cell 1. For $M_5$ which is next, only 0.6 units are allocated to its first instance since 0.4 units are used already from part 1. The remaining 0.4 units needed are allocated to the second instance of machine of type 5 which happens to be available. Finally, 0.1 units of $M_2$ are allocated in the first instance of machine of type 2. This instance had available 0.4 units since only 0.6 units have been used from part 10 earlier. $PARTMATRIX$ for part 9 will now look as follows:

$$PARTMATRIX(:,:,9) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.6 & 0.4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{22}$$

It is also worth noting that *part_moves*, which was mentioned earlier, will now look as follows:

$$part\_moves = \begin{bmatrix} 1 & 1 & 5 & 1 \\ 10 & 1 & 2 & 1 \\ 10 & 2 & 6 & 1 \\ 3 & 1 & 3 & 1 \\ 9 & 1 & 3 & 1 \\ 9 & 1 & 5 & 1 \\ 9 & 1 & 5 & 2 \\ 9 & 1 & 2 & 1 \end{bmatrix} \tag{23}$$

For example part 2 (see rows two and three) has visited cells 1, 2 using the first instance of machine of type 2 and the first instance of machine of type 6 in sequential manner respectively.

The rest of the parts are allocated to cells in a similar way.

### 3.3 Objective value evaluation

For determining the value of the objective function for the solution obtained via the part machine cell allocation a number of routines were implemented. For finding the number of the distinct allocations of parts to cells, *part_moves* was used. For every part its corresponding segment (i.e. the cells occupied relative to the current part's machine operation sequence) in *part_moves*, was found and a search was conducted. For example, for a cell sequence of the form [3 1 1 3 3 1] which is relative to current part's machine operation sequence two distinct allocations were recorded since cells 3 and 1 were occupied by the current part.

For determining the total part/machine set-up cost, $PCMATRIX$ was used to find the number of machines instances of certain machine type used by current part in each cell. The total set-up cost was finally determined by multiplying the latter with the corresponding cost, i.e. $SETUP_{i,j}$.

In a similar way, a routine was built to determine the number of later revisits of parts to already visited cells. For each part the visited cells sequence was considered and revisits to cells were accumulated when encountered. For example, for a cell sequence of the form [3 1 1 3 3 1] two revisits of a certain part to cells are counted: i) when part returns to cell 3 for an intermediate machine operation and ii) when part revisits cell 1 for the final machine operation.

## 4 Main tabu search components

Tabu search is one of the most successful metaheuristics for application to combinatorial problems. The basic ideas of TS were introduced by Glover [16]. A description of the method and its concepts can be found in Glover [17, 18], and Glover and Laguna [19]. The basic idea of TS is the explicit use of search history, both to escape local minima and to implement an explorative strategy.

TS can either employ short term memory and be characterised as a 'simple' searching process, or long term memory and become more complex. For the purpose of this paper short term memory will be utilised and act as the main framework for the TS development. This framework will be employed within an iterative procedure where a number of techniques or more minor heuristic approaches useful for producing overall an effective algorithm will be proposed. Before proceeding with the actual presentation of the TS algorithm developed, a number of its key elements and tools will be presented.

## 4.1 Definition of a feasible solution

In the context of the proposed tabu search approach only feasible solutions are produced strictly ‘adopting’ all the requirements and specifications of the current MIMP model for CF. More specifically, a feasible solution consists of an assignment of machines to cells and an assignment of parts to machine cells when the part machine operation sequence is taken into account. While machines are allocated to cells certain elements are stored, holding information about the capacity used for each cell and to which cell each machine is allocated. These elements provide all the information needed for the next step which is the allocation of parts to machine cells. The latter as the most complex and key stage for the formation of the solution involves a significant number of look-up tables to store information while every part preserving its machine operation sequence is allocated to machines in cells. After parts are fully allocated, which is indicated by the fact that when a temporary part/machine utilisation matrix becomes empty, the calculation of the relevant costs within the objective function can be carried out. The assignment of parts to cells allows for the calculation of the number of distinct cells used by each part and also the later revisits of parts to already visited cells. Finally, depending upon the number of machines of a specific type used by a part in a particular cell the machine set-up costs can also be determined leading to the total cost calculation for CF.

## 4.2 Neighborhood generation - moves definition

In TS methods, each iteration of the search process focuses on finding a good neighborhood solution with better quality than the current solution. For the current study the neighborhood of a current solution is defined as the set of all feasible solutions that can be reached by a move. Two types of moves (a) single and (b) exchange will be considered for generating neighboring solutions for each configuration in the system. Please note that only feasible moves are examined due to the structure of the algorithm and the fact that there always exists a feasible schedule. The single move is an operation that moves a machine instance pair, $(i, k)$, from its current cell $q$ (source cell) to a new cell $q'$ (destination cell). The exchange move is an operation which consists of two independent single moves. If a machine instance pair $(i, k)$ is moved from its source cell $q$ to another cell $q'$ (first single move), then another machine of type $i'$[1] of instance $k'$ will be moved from the destination cell $q'$ of the first move to the source cell $q$ of the first move (second single move) in exchange. Thus the two moves generated are $((i, k),\ q')$ and $((i', k'),\ q)$.

Whenever a different machine instance pair becomes the candidate for moving it between cells another neighborhood with a certain set of solutions will be generated. From each neighborhood a solution will be chosen based on some criteria, discussed

[1]Note that the machine of type $i'$ in destination cell $q'$ could be of the same type as the machine of type $i$ in the source cell $q$. However, these machines will have a different instance number. Also note the way that this exchange might improve the current solution depending on the available capacity of each of the machines involved and the candidate part/machine requirements while the allocation process takes place.

within section 4.4, to become the current one from where the search will continue with the generation of a new neighborhood. This process continues until the number of iterations is reached which is the stopping condition for the current TS algorithm.

## 4.3 Aspiration criterion

The aspiration criterion (AC) employed here resembles the typical and customarily used global form of aspiration by objective, where the tabu status of a move is overridden when the move improves the best value found so far.

## 4.4 Tabu list and tabu tenure

The short term memory employed is the history of recent moves (recency memory) and aims primarily at preventing moving back to these moves, avoiding being trapped at local optima or causing cycling. The most common implementation of the short term memory is based upon the storage/update of the move attributes that were recently visited. Each time a move is implemented two entries will be recorded in the tabu list: the forward attribute of the move and the reverse of it. The forward attribute involves the corresponding part, machine instance pair, and the new cell assignment (destination cell), whereas the reverse involves the corresponding part, the machine instance pair and the old cell assignment (source cell). The forward attribute is stored in order to avoid considering this move for a substantial time interval, whereas the reverse is stored in order to avoid cycling. Please note that for each of these entries an additional entry, the tabu tenure, is added denoting their duration for remaining tabu active. A move which is implemented is added to the tabu list and one unit is subtracted from the tabu tenures of the remaining entries. Also the entries whose tabu tenure is zero are deleted from the list.

After a number of experiments were carried out towards the creation of an effective algorithm with a significant exploration of the search space, the tabu list size was decided to be fixed with values ranging from five to fifteen depending upon the size of the problem tested each time. This is in line with Glover [16]. More specifically, for small to medium problem instances the size of the tabu list was kept relatively small, i.e. five or similar, whereas for bigger problems, this value needed to be bigger, i.e. ten or greater in order to force the proposed strategy to move into unexplored areas. List sizes greater than fifteen were not found to be useful.

The question arising at this stage is which move should be implemented from each generated neighborhood as the iterative procedure goes along. Initially, all the solutions found within a neighborhood are sorted in ascending order of their corresponding objective values. If the solution with the smallest objective value happens to be less than the best value found so far then its corresponding move will become admissible for implementation no matter the tabu status of its forward attribute due to aspiration criterion. If this is not the case, then the procedure will continue locally, within the current neighborhood, in search of the move to implement. The solution with the best objective value will be examined first to check whether the tabu status of its forward attribute is still active. If this is happening then the search will continue with the solution that follows next in the sorted sequence and so on till a move that can become admissible is found.

## 5 Tabu search algorithm design

Before describing the complete TS iterative procedure a few comments will be made about the structure of the algorithm. The algorithm consists of two different stages. The first stage involves the neighborhood generation where for a certain part moves are considered. More specifically, a candidate machine instance pair relative to the current part's machine operation sequence is moved from the source cell under consideration, to the remaining cells (i.e. destination cells). The decision on which move type to be considered depends upon the capacity of machine instances that both source and destination cells have. Each move of the candidate machine instance pair entails the creation of a unique solution within the neighborhood. For each solution, part reallocation is performed to increase the possibilities of receiving a better solution before evaluating the value of the objective function. Also a temporary tabu list is updated storing all move attributes. The second stage is the main phase within the search process where the principles of the tabu search algorithm are adopted and decisions are made concerning the direction of the search. More specifically, a process commences in the search for an admissible move from where the process continues and a permanent tabu list is updated.

Please note that the algorithm commences by considering the number of parts and more specifically the part that causes the majority of the intercellular movements in the initial solution. The latter has been identified as an important element in the objective function (1) (see for instance [38]). The better the distinct allocation of parts to cells the better the value of the objective function for current CF problem.

### 5.1 Iterative procedure

The complete procedure of TS for CF is provided below.

1. Initialisation process;

2. Generate a feasible solution by the random approach of allocating machines to cells and parts to machines cells;

3. Order the parts in ascending order of the intercellular movements caused;

4. For the first part in sequence find its corresponding segment in *part_moves* and more specifically the occupied cells and machine instance pairs relative to part's machine operation sequence;

5. For the first machine instance pair identify source and destination cells in order for a move to commence;

6. Determine the capacity of both current source and destination cells; if capacity of destination cell is less than $E_{MAX}$ and capacity of source cell is less than or equal to $E_{MAX}$ and greater than or equal to $E_{MIN}$ a single move is considered; else an exchange;

7. After any of the move types is being considered and $CELLMATRIX$ is updated accordingly, reallocate parts to machine cells for the solution obtained and evaluate objective function value;

8. Update the temporary tabu list by storing for current solution both forward and reverse attributes of the corresponding move together with their maximum value of tabu tenure;

9. Continue doing the process above till all solutions for current machine instance pair configuration have been generated and the neighborhood has been formed;

10. From the current neighborhood sort the objective values of all solutions in ascending order;

11. If local best solution, is better than global best found so far, set global best to be equal to local best no matter the tabu status of the forward attribute for current move. This is happening due to the aspiration criterion; also update $part\_moves$, the distinct allocations of parts to cells and the permanent tabu list for the move just implemented;

12. Else continue searching locally for the move to accept as admissible whose tabu status is not tabu active, i.e. whose forward attribute is not in the temporary tabu list. Once this solution is found perform similar updates with step 11 and continue with step 5 by considering the next machine instance pair for the current part. When all machine instance pairs for current part have been considered and different neighborhoods have been generated move to step 4 and the next part in sequence;

13. If the iteration limits have been reached, stop the run; otherwise go to step 3.

Step 4 as described above implies the search in $part\_moves$ for the part currently under consideration. For example if we refer to matrix (23) and assume 9 as the first part to be considered within the iterative tabu search algorithm its segment will be:

$$part\_moves = \begin{bmatrix} 9 & 1 & 3 & 1 \\ 9 & 1 & 5 & 1 \\ 9 & 1 & 5 & 2 \\ 9 & 1 & 2 & 1 \end{bmatrix} \tag{24}$$

The first neighborhood will be generated based on the first row which describes that the first instance of machine of type 3 is in cell 1. Thus source cell will be cell

1 and destination cells the remaining three. Depending on the capacity of all cells, as described in step 6 above, a single or an interchange move will commence, e.g. first instance of machine of type 3 might be send to cell 2 without any exchange, but for moving it to cell 3 an exchange might be needed instead depending on the capacity. Everytime a move is considered the machine cell allocation is updated, part reallocation is also considered followed by an evaluation of the objective function. Note also that everytime a move is performed *part_moves* is also dynamically updated to record the latest allocation of machines to cells for each part in relation to its part machine operation sequence. The latter occurs since the part machine operation sequence has a key role in all operations increasing the complexity.

## 6 Tabu search algorithm assessment

For assessing the computational performance of the proposed tabu search algorithm a number of problem instances, twenty in total, was generated. The number of parts and machines types involved in each of the data sets was adapted from problems previously used in the literature where the majority of them are medium to large sized problems. No available problem instances matching the current mathematical model requirements, by providing values to all the necessary parameters, are available in the existing literature, so additional data for each problem involved had to be generated following the pattern used by other authors of modifying existing data sets. Table 2 shows the reference for each problem, the number of parts and the number of machines included for each problem instance involved[2].

For each problem instance, the part/machine utilisation ($UTIL_{i,j}$), the part/machine set-up costs ($SETUP_{i,j}$) and the parts sequence ($L_{j,z}$) involved were randomly generated. More specifically, the utilisation amount was generated using a uniform distribution with parameters [0.0, 1.2] and values rounded to units of 0.1. The value of the upper range was used in order to include a requirement for more than a unit where a split needs to be performed. Values of $UTIL_{i,j}$ are assumed to contain some in built redundancy. The set-up cost of each machine to be used by a certain part was also generated using a uniform distribution with parameters [0.00, 6.00], and values rounded to units of 0.01. Please note that the range includes reasonable values without the maximum being too extreme. For generating the machine operation sequence for each part a different method had to be employed. For identifying for each part the machine operations involved, the indices of the non-zero elements in $UTIL_{i,j}$ were found and stored. Before adding for each part the machine operations (corresponding to these indices) to matrix $L_{j,z}$, a perturbation was carried out in order to increase the possibilities of obtaining a randomly generated machine operation sequence for each part. Further, $M_{j,q}$ has a constant value of ten whereas, $A_j$ a value of one. These values were chosen on the basis of obtaining sensible results. Finally, $E_{MIN}$ was set to four and $E_{MAX}$ was set to take values between six and seventeen depending upon the size of the problem instance employed each time.

[2]Please note that data is available upon request.

## 7 Computational results

The tabu search algorithm presented in this study was coded in `MatLab`(TM),[3] whereas the optimum or best known integer programming solutions, which required much longer runs, were obtained from `XPRESS-MP`. Both `Matlab` and `XPRESS-MP` run on a Linux machine (Intel (P4 Xeon) 3GHz, 1.00 GB of RAM) accessed remotely.

The computational results for all data sets are presented in Table 3. The elements determined for each instance are: number of cells, total number of machine instances, problem size, i.e. the product of the number of constraints and the number of decision variables involved in the mathematical model and determined by `XPRESS-MP`, the best known objective value obtained (including the optimum when reached) via `XPRESS-MP`, and the corresponding CPU time. Also the best value of the objective function found via TS and the required CPU time together with the value of the initial objective value are recorded. Please note that for the best value generated via TS especially when a large data set was employed a number of simulation runs was conducted for exploring different solutions and choosing the best among them. For the reader's interest, the number of runs for all instances is also added in Table 3. Moreover, a comparison is made between the value of the initial objective function and the best value found and the percentage of improvement/deviation from the best known value is recorded. The latter is a performance criterion for the tabu search and its efficient exploration of the search space. In addition, the percentage deviation of each solution obtained via TS from the optimum or best known value obtained via `XPRESS-MP` is indicated. Furthermore, the mean CPU time and the mean deviation values are found and presented at the bottom of the table. Finally, the numbers of parts and machine types involved in each problem are also listed for completeness.

As can be seen from Table 3, when `XPRESS-MP` is employed the optimum solution is only found in three problem instances (two, sixteen and twenty), whereas for the rest of the instances it was not possible to obtain optimal solutions within a very generous time limit, so the best known solutions[4] were recorded. In the current study this limit was set to be equal to thirty hours.

The results presented in Table 3 are encouraging compared to integer programming methods proving that the tabu search algorithm is effective. More specifically, the mean deviation value of the TS algorithm from the best known objective value is nine percent. The required CPU time for all but two of the problem instances does not exceed 1206 seconds overall (problem twelve is exceptionally hard to solve, as the largest data set of all and needs a substantial CPU time in order for the iterative procedure to end). For more than 50% of the problems the CPU time is less than ten minutes. The integer programming solutions will in almost all cases be an overestimate of the optimal solutions, hence the percentage deviations will also be an overestimate. However, it is unlikely that there will be significant overestimates given

[3]`MatLab` is a trademark of the MathWorks, Inc., 1994-2007.

[4]Please note that when the best known solutions were recorded the optimality gap indicated via `XPRESS-MP` was still large but this does not necessarily imply that solutions were still far from the optimum as many unexplored nodes could turn out to be of no interest. Thus it was decided not to tabulate the gap as it was large.

the length of time the integer programming software ran without any improvements to solutions. Thus the testing has indicated that the tabu search is able to provide:

- reasonably good quality solutions;
- solutions in reasonable amounts of CPU time;
- solutions that are a substantial improvement on heuristically generated initial solutions.

In order to illustrate the behaviour of the TS iterative procedure problem instance fourteen is employed where the deviation of the TS from the best known value is one percent. Figure 2(a) shows the fluctuation of the objective values with respect to the iterations involved when both moves admissible and non-admissible are considered. Further, Figure 2(b) presents for the same problem instance a portion on the trend of the objective values when only the admissible moves are recorded.

It can be seen from Figure 2(a) that the total number of iterations needed are approximately 2600. Moreover, the best objective value of magnitude 743.19 units, as specified on Table 3, is first obtained at iteration number 2326 and then appears again at later iterations. Although a rapid descent flow of the objective values is shown in Figure 2(a) a significant number of solutions is investigated in the search space increasing the possibilities of receiving a better TS deviation from the best known objective value. Moreover, since the search does not always continue from the best solution found, unless the AC is met, this leads to the investigation of areas that might not sound very promising but could lead later on to very useful results.

The deviation between the initial objective value and the best cost obtained is 30% which is very significant indicating that within the iterative procedure and the neighborhood generation for each candidate machine instance pair, exploration of many areas occurs implying the investigation of many non-visited solutions. For achieving the latter, the utilisation of short term memory and more specifically the use of a tabu list played an important role.

An important element in the operation of the tabu search iterative procedure is its robustness when an initial solution is fed into it. More specifically, once the initial solution is generated the iterative procedure commences and it finishes when the total number of iterations is reached producing a certain output. If another initial solution is fed into the iterative procedure, another solution will be produced whose deviation from the best known value will not differ much from the deviation obtained via the first attempt. Figure 2(c) illustrates the latter where a different initial solution is generated for problem instance fourteen and the iterative procedure is run only once. Note that both the total number of iterations and the value of tabu tenure remain the same. It can be seen from Figure 2(c) that the total number of iterations is approximately 2600, as before. The deviation of the TS from the best known solutions is 3%, when the value of the initial solution is 1134.1 and the best cost found is 752.37. These aspects of robustness were found to hold for all problem instances.

Overall, different runs with different initial solutions produce solutions whose deviation values do not differ much in magnitude from each other. In the case, of course, that the problem is too big such as problem instance twelve, multiple runs, i.e fifteen, were conducted for exploration of different initial solutions together with suitable adjustments to either the number of iterations or the value of the tabu tenure leading to the best possible value among all. Figure 3 illustrates the fluctuation of the objective values involved for problem instance twelve (the largest data set tested).

As can be seen from Table 3, in general computational times rise as problem size measured by $TMI$ increases with problem instance 20 the fastest to solve and problem instance 12 the slowest. It was found that quality of solution did not necessarily decrease in proportion to increase in $TMI$ or $NM$, but that problem instances where the ratio $TMI/NM$ was highest were usually lowest in solution quality.

Finally, a few comments on the part machine operation sequence. As already mentioned Foulds, French and Wilson [14] considered a less complex type of CF problem with no inclusion of the part machine operation sequence. The latter produced a simpler system to be solved using a heuristic algorithm. For the current study the part machine sequence was one of the most important features to be included and to be taken into account for almost all the heuristic approaches implemented within the initial search. Later in the design of the tabu search, the part machine operation sequence acted as a key element since with every move of a specific machine instance pair certain updates were taking place for the part machine cell allocation relative to the part machine sequence of all parts involved. Although, the part machine operation sequence made the initial solution more complex and the tabu search more difficult to converge to a value very close to the best known value after exploring a number of areas in the search space, it incorporated realism and produced a more practical CF system.

## 8 Summary

This paper presented a sophisticated mixed integer linear programming model for the cell formation problem where machines are grouped into cells and parts into machine cells simultaneously when the part machine operation sequence is taken into account to meet real conditions. Based on the mathematical model a three stage heuristic approach was proposed to produce a randomly generated initial solution ready to be fed into an iterative algorithm that searches the space in a higher level. This was achieved with the proposal of a tabu search algorithm for the cell formation problem having as a main aim the creation of a stable and robust system able to cope with the incorporation of the part machine sequence. Finally, a number of problem instances was generated to assess the algorithm's performance. The computational results proved to be promising thus increasing the possibilities that all the tools developed could become useful tools for production planners.

The solution times and quality based on the current algorithms can be experimented on further. Also the developed model and algorithms can be further extended with the combination of the tabu search algorithm with another heuristic or metaheuristic approach to form a hybrid search approach for the cell formation problem. Moreover, the part allocation phase within the initial solution can also be extended to allow for some flexibility on the choice of the machines to be used by certain parts for allocation instead of following a certain path. This implies the creation of a system based on relative feedback produced at each stage.

## Appendix

| Indices | Description |
|---|---|
| $i$ | machine type index: $i = 1, \ldots, NM$ |
| $j$ | part index: $j = 1, \ldots, NP$ |
| $q$ | cell index: $q = 1, \ldots, NC$ |
| $k$ | machine instance index: $k = 1, \ldots, KM_i$ |
| $z, r$ | machine operation indices: $z, r = 1, \ldots, ZTYPES_j$ |
| Input Parameters | Description |
| $E_{MIN}$ | minimum number of machines allowed in a cell |
| $E_{MAX}$ | maximum number of machines allowed in a cell |
| $NC$ | number of cells to be created in the system |
| $NP$ | number of parts involved |
| $NM$ | number of machine types in the system |
| $M_{j,q}$ | weighting factor if part $j$ is allocated to cell $q$ |
| $A_j$ | weighting factor for part $j$ traveling back to an already visited cell |
| $UTIL_{i,j}$ | utilisation of machine $i$ by part $j$ |
| $KM_i$ | number of machines instances for each machine type $i$, determined as: $KM_i = \lceil \sum_{j=1}^{NP} UTIL_{i,j} \rceil$ |
| $SETUP_{i,j}$ | set-up cost of machine $i$ needed to process part $j$ |
| $ZTYPES_j$ | number of different operations (machine types) required by part $j$ |
| $L_{j,z}$ | for part $j$ the machine used for the $z^{th}$ machine operation in sequence |
| $UTILMIN$ | minimum amount of utilisation in $UTIL_{i,j}$ matrix ($UTILMIN$ is non zero) |
| $UTILMAX$ | largest amount of machine utilisation used |
| $KMAX$ | the maximum number of machine instances recorded for all machine types in $KM_i$ |
| $TMI$ | total number of machine instances in the system |
| $CELL_{MIN}$ | minimum number of cells |
| $CELL_{MAX}$ | maximum number of cells |

| Decision Variables | Description |
|---|---|
| $x_{i,j,q}$ | amount of processing by machines of type i for part $j$ in cell $q$ |
| $y_{i,k,q}$ | =1 if $k^{th}$ machine instance of type $i$ is assigned to cell $q$, 0 otherwise |
| $w_{j,q}$ | =1 if part $j$ is processed in cell $q$, 0 otherwise |
| $v_q$ | =1 if cell $q$ is formed, 0 otherwise |
| $s_{i,j,q}$ | number of machines of type $i$ used by part $j$ in cell $q$ |
| $extra_{q,j,L_{j,z}}$ | =1 if after the $z^{th}$ operation of part $j$ in cell $q$ the part leaves cell $q$ but returns later, 0 otherwise |
| $xx_{L_{j,z},j,q}$ | =1 if part $j$ is processed in cell $q$ for $z^{th}$ machine operation, 0 otherwise |

## References

[1] Adil, G.K., Rajamani, D. and Strong, D. (1997). Assignment allocation and simulated annealing algorithms for cell formation, *IIE Transactions*, 29 (11), 53-67.

[2] Ahkioon, S., Bulgak, A.A. and Bektas, T. (2007). Cellular manufacturing systems design with routing flexibility, machine procurement production planning, and dynamic system reconfiguration, *International Journal of Production Research*, iFirst, 1-28.

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

[4] Askin, R.G., Selim, H.M. and Vakharia, A.J. (1997). A methodology for designing flexible cellular manufacturing systems, *IIE Transactions*, 29, 599-610.

[5] Beaulieu, A., Gharbi, A. and Ait-Kadi. (1997). An algorithm for the cell formation and machine selection problems in the design of a cellular manufacturing system, *International Journal of Production Research*, 35 (7), 1857-1875.

[6] Boe, W.J. and Cheng, G.H. (1991). A close neighbour algorithm for designing cellular manufacturing systems, *International Journal of Production Research*, 29 (10), 2097-2116.

[7] Burbidge, J.L. (1989). *Production flow analysis for planning group technology*, UK, Oxford Science Publications.

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

[9] Chan, W.M., Chan, C.Y. and Ip, W.H. (2003). A heuristic algorithm for machine assignment in cellular layout, *Computers and Industrial Engineering*, 44, 49-73.

[10] Chan, H.M. and Milner, D.A. (1982). Direct clustering algorithm for group formation in cellular manufacturing, *Journal of Manufacturing Systems*, 1 (1), 65-75.

[11] Congawave, T. and Ham, I. (1981). Cluster analysis applications for group technology manufacturing systems, *Proceedings, North American Manufacturing Research Conference (NAMRC), $9^{TH}$ (Dearborn)*, 65-75.

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

[13] Dorigo, M., Maniezzo, V. and Colorni, A. (1996). Ant system: Optimisation by a colony of cooperating agents, *IEEE Transactions on Systems, Man and Cybernetics - Part B Cybernetics*, 26, 29-41.

[14] Foulds, L.R., French, A.P. and Wilson, J.M. (2006). The sustainable cell formation problem: manufacturing cell creation with machine modification costs, *Computers and Operations Research*, 33, 1010-1032.

[15] Garey, M.R. and Johnson, D.S. (1979). *Computers and Intractability,* San Fransisco, CA: Freeman.

[16] Glover, F. (1986). Future paths for integer programming and links to artificial intelligence, *Computers and Operations Research*, 13, 533-549.

[17] Glover, F. (1989). Tabu Search - Part I, *ORSA Journal on Computing*, 1 (3), 190-206.

[18] Glover, F. (1990). Tabu Search - Part II, *ORSA Journal on Computing*, 2 (1), 4-32.

[19] Glover, F. and Laguna, M. (1997). *Tabu Search*, Norwell, MA: Kluwer Academic.

[20] Holland, J.H. (1975). *Adaptation in Natural and Artificial Systems*, Ann Arbor, University of Michigan, Michigan.

[21] Jayaswal, S. and Adil, G.K. (2004). Efficient algorithm for cell formation with sequence data, machine replications and alternative process routings, *International Journal of Production Research*, 42 (12), 2419-2433.

[22] Kim, C.O, Baek, J.G. and Baek, J.K. (2004). A hybrid grouping genetic algorithm for the cell formation problem, *Computers and Operations Research*, 34, 2059-2079.

[23] Kim, C.O, Baek, J.G. and Baek, J.K. (2004). A two-phase heuristic algorithm for cell formation problems considering alternative part routes and machine sequences, *International Journal of Production Research*, 18, 3911-3927.

[24] 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.

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

[26] Kirkpatrick, S., Gelatt Jr., C.D. and Vecchi, M.P. (1983). Optimisation by simulated annealing, *Science*, 220, 671-680.

[27] Kumar, K.R., Kusiak, A. and Vannelli, A. (1986). Grouping of parts and components in flexible manufacturing systems, *European Journal of Operational Research*, 24, 387-397.

[28] Kusiak, A. (1987). The generalized group technology concept, *International Journal of Production Research*, 25, 561-569.

[29] Kusiak, A. and Heragu. S.S. (1987). Group Technology, *Computers in Industry*, 9, 83-91.

[30] Mak, K.L. and Wong, Y.S. and Wang, X.X. (2000). An adaptive genetic algorithm for manufacturing cell formation, *International Journal of Advanced Manufacturing Technology*, 16, 491-497.

[31] McCormick, W.T., Schweitzer, P.J. and White, T.W. (1972). Problem Decomposition and data reorganisation by a clustering technique, *Operations Research*, 20, 993-1009.

[32] Mukattash, A.M., Adil, M.B. and Tahboub, K.K. (2002). Heuristic approaches for part assignment in cell formation, *Computers and Industrial Engineering*, 42, 329-341.

[33] Onwubolu, G.C. and Mutingi, M. (2001). A genetic algorithm approach to cellular manufacturing systems, *Computers & Industrial Engineering*, 69, 373-383.

[34] Papaioannou, G. & 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.

[35] Purcheck, G.F. (1975). Programming method for combinatorial group of an incomplete power set, *Journal of Cybernetics*, 5, 51-76.

[36] Sankaran, S. (1990). Multiple objective decision making approach to cell formation - A goal programming model, *International Journal of Production Research*, 13, 71-81.

[37] Sankaran, S. and Kasilingam, R.G. (1993). On cell size and machine requirements planning in group technology systems, *European Journal of Operational Research*, 69, 373-383.

[38] Selim, M.S., 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, 3-20.

[39] Singh, N. (1993). Design of Cellular Manufacturing Systems: An Invited Review, *European Journal of Operational Research*, 69, 284-291.

[40] Solimanpur, M. and Vrat, P. and Shankar, R. (2004). Ant colony optimisation algorithm to the inter-cell layout problem in cellular manufacturing, *European Journal of Operational Research*, 157, 592-606.

[41] Spiliopoulos, K. and Sofianopoulou, S. (2003). Designing manufacturing cells: a staged approach and a tabu search algorithm, *International Journal of Production Research*, 41 (11), 2531-2546.

[42] Stanfel, L.E. (1985). Machine clustering for economic production, *Engineering Costs and Production Economics*, 9, 73-81.

[43] Vakharia, A.J. and Chang, Y.-L. (1997). Cell formation in group technology: a combinatorial search approach, *International Journal of Production Research*, 35 (7), 2025-2043.

[44] Vakharia, A.J. and Wemmerlov, U. (1990). Designing a cellular manufacturing system: a materials flow approach based on operation sequences, *IIE Transactions*, 22 (1), 84-97.

[45] Vannelli, A. and Kumar, K.R. (1986). A method for finding minimal bottle-neck cells for grouping part-machine families, *International Journal of Production Research*, 24 (2), 387-400.

[46] Venugopal, V. and Narendran, T.T. (1992). A genetic algorithm approach to the machine-component grouping problem with multiple objectives, *Computers and Industrial Engineering*, 22 (4), 469-480.

[47] Wang, J. (1998). A linear assignment algorithm for formation of machine cells and part families in cellular manufacturing, *Computers and Industrial Engineering*,35, 81-84.

[48] Wei, J.C. and Gaither, N. (1992). An optimal model for cell formation decisions, *Decision Sciences*, 21, 416-433.

[49] Wemmerlov, U. and Hyer, N.L. (1986). Procedures for the Part/Machine Group Identification Problem in Cellular Manufacturing, *Journal of Operations Management*, 6, 125-147.

[50] Wemmerlov, U. and Hyer, N.L. (1986). Cell Manufacturing in the US Industry: A Survey of Users, *International Journal of Production Research*, 27, 1511-1530.

[51] Wu, T.-H., Low, C. and Wu, W.-T. (2004). A tabu search approch to the cell formation problem, *International Journal of Advanced Manufacturing Technology*, 23, 916-924.

[52] 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 (3), 434-446.

[53] Xpress-MP. Dash Optimisation, Blisworth, Northamptonshire, UK.

[54] Zhu, Z., Heady, R.B. and Reiners, S. (1995). An efficient zero-one formulation of the cell formation problem, *Computers and Industrial Engineering*, 28, 911-916.

*Received December, 2011*

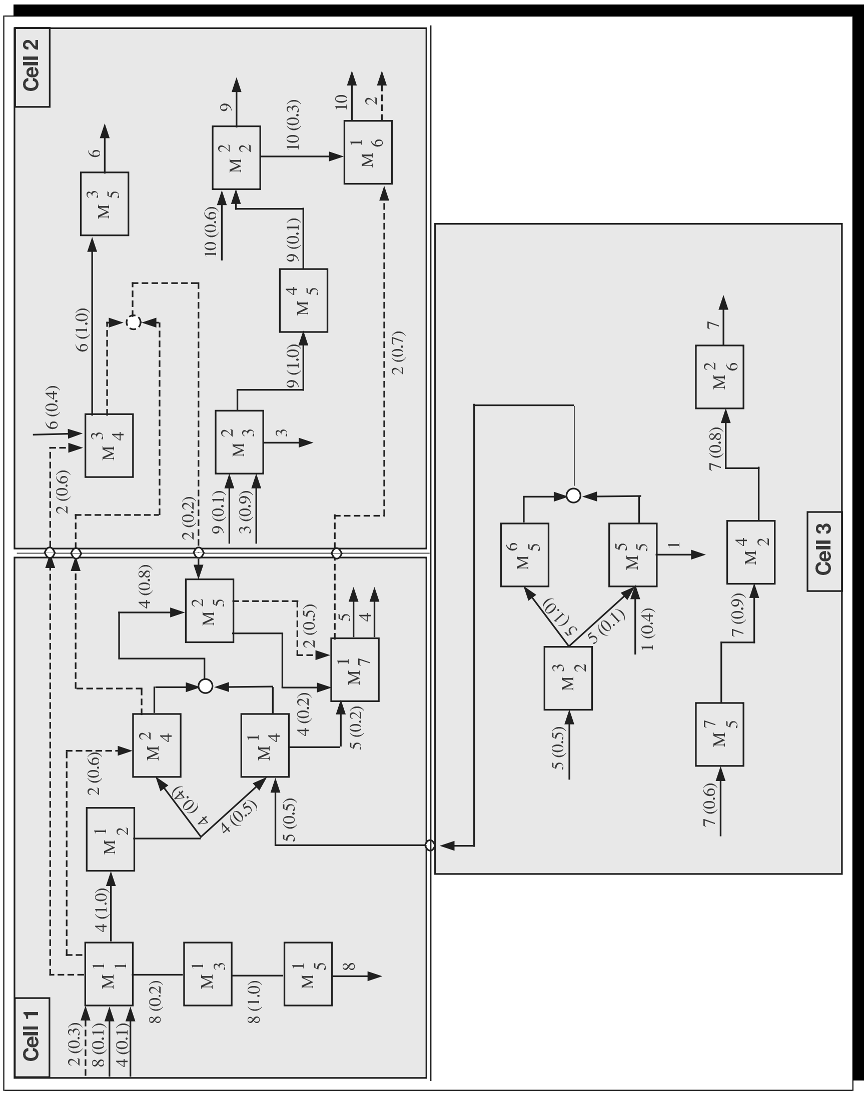
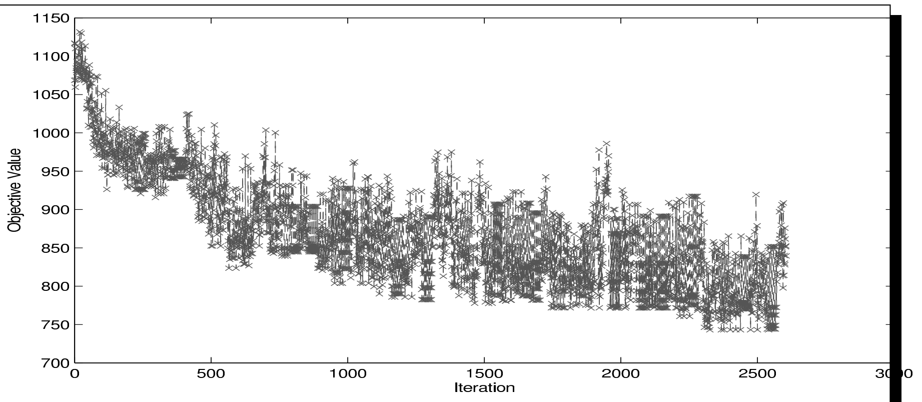
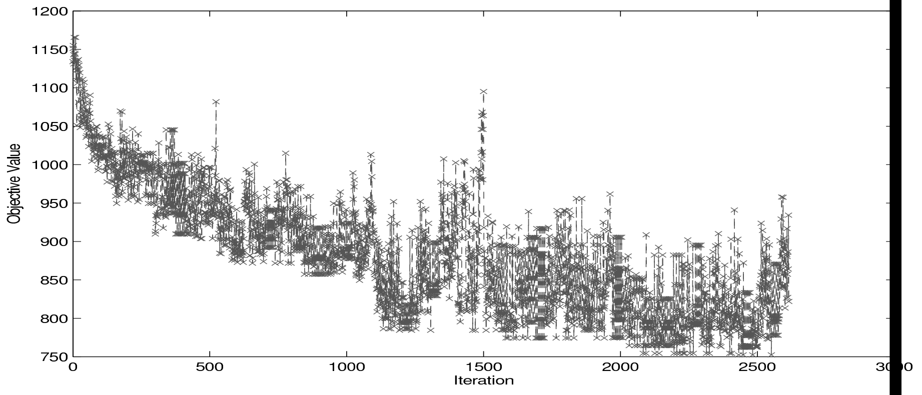


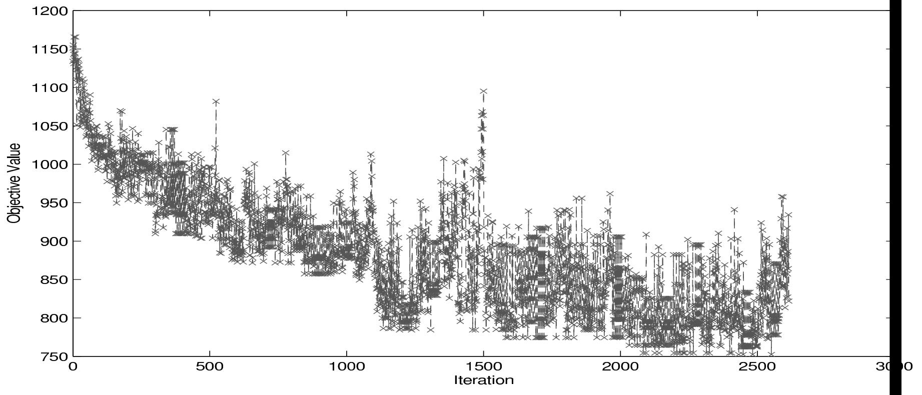
**Figure 1**: The problem of CF: An example



(a) Objective Function Value Evolution: All Moves



(b) Implemented Moves: Portion of Figure 2(a) between Iterations 2000-2500



(c) Objective Function Value Evolution when Initial Solution Differs

**Figure 2**: Problem instance 14

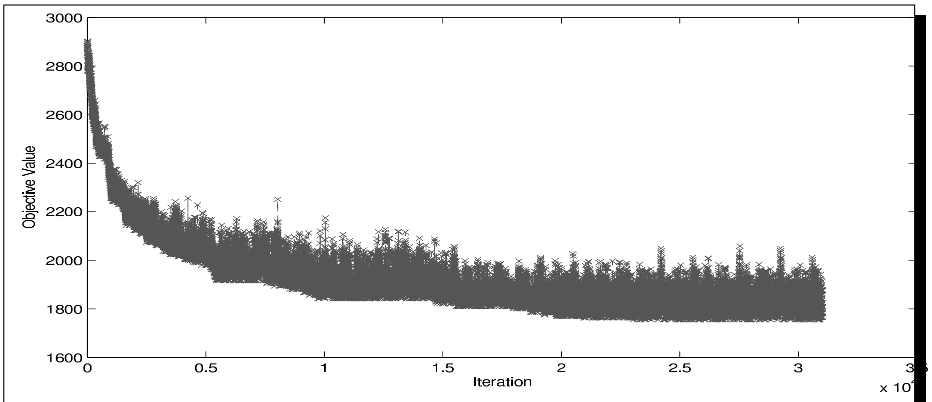


**Figure 3**: Problem instance 12

**Table 1**: Numerical values for IP CF model

| **Part/Machines utilisation & number of machine instances** | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $P_j/M_i$ | $P_1$ | $P_2$ | $P_3$ | $P_4$ | $P_5$ | $P_6$ | $P_7$ | $P_8$ | $P_9$ | $P_{10}$ | $\sum_{j=1}^{10} UTIL_{i,j}$ | $KM_i$ |
| $M_1$ | 0.0 | 0.3 | 0.0 | 0.1 | 0.0 | 0.0 | 0.0 | 0.1 | 0.0 | 0.0 | 0.5 | 1 |
| $M_2$ | 0.0 | 0.0 | 0.0 | 1.0 | 0.5 | 0.0 | 0.9 | 0.0 | 0.1 | 0.6 | 3.1 | 4 |
| $M_3$ | 0.0 | 0.0 | 0.9 | 0.0 | 0.0 | 0.0 | 0.0 | 0.2 | 0.1 | 0.0 | 1.2 | 2 |
| $M_4$ | 0.0 | 1.2 | 0.0 | 0.9 | 0.5 | 0.4 | 0.0 | 0.0 | 0.0 | 0.0 | 3.0 | 3 |
| $M_5$ | 0.4 | 0.2 | 0.0 | 0.8 | 1.1 | 1.0 | 0.6 | 1.0 | 1.0 | 0.0 | 6.1 | 7 |
| $M_6$ | 0.0 | 0.7 | 0.0 | 0.0 | 0.0 | 0.0 | 0.8 | 0.0 | 0.0 | 0.3 | 1.8 | 2 |
| $M_7$ | 0.0 | 0.5 | 0.0 | 0.2 | 0.2 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.9 | 1 |
| **Set-up costs** | | | | | | | | | | | | |
| $P_j/M_i$ | $P_1$ | $P_2$ | $P_3$ | $P_4$ | $P_5$ | $P_6$ | $P_7$ | $P_8$ | $P_9$ | $P_{10}$ | | |
| $M_1$ | 0.00 | 2.95 | 0.00 | 1.96 | 0.00 | 0.00 | 0.00 | 1.92 | 0.00 | 0.00 | | |
| $M_2$ | 0.00 | 0.00 | 0.00 | 5.12 | 2.42 | 0.00 | 5.05 | 0.00 | 1.54 | 2.16 | | |
| $M_3$ | 0.00 | 0.00 | 2.93 | 0.00 | 0.00 | 0.00 | 0.00 | 1.94 | 1.45 | 0.00 | | |
| $M_4$ | 0.00 | 5.54 | 0.00 | 2.91 | 2.42 | 2.38 | 0.00 | 0.00 | 0.00 | 0.00 | | |
| $M_5$ | 2.91 | 2.59 | 0.00 | 2.88 | 5.42 | 4.91 | 2.63 | 4.93 | 4.81 | 0.00 | | |
| $M_6$ | 0.00 | 2.82 | 0.00 | 0.00 | 0.00 | 0.00 | 2.73 | 0.00 | 0.00 | 2.49 | | |
| $M_7$ | 0.00 | 2.78 | 0.00 | 2.12 | 2.12 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | | |
| **Part-Machine operation sequences & number of operations for each part** | | | | | | | | | | | | |
| Sequence/$P_j$ | 1 | 2 | 3 | 4 | 5 | | | | | | $ZTYPES_j$ | |
| $P_1$ | $M_5$ | | | | | | | | | | 1 | |
| $P_2$ | $M_1$ | $M_4$ | $M_5$ | $M_7$ | $M_6$ | | | | | | 5 | |
| $P_3$ | $M_3$ | | | | | | | | | | 1 | |
| $P_4$ | $M_1$ | $M_2$ | $M_4$ | $M_5$ | $M_7$ | | | | | | 5 | |
| $P_5$ | $M_2$ | $M_5$ | $M_4$ | $M_7$ | | | | | | | 4 | |
| $P_6$ | $M_4$ | $M_5$ | | | | | | | | | 2 | |
| $P_7$ | $M_5$ | $M_2$ | $M_6$ | | | | | | | | 3 | |
| $P_8$ | $M_1$ | $M_3$ | $M_5$ | | | | | | | | 3 | |
| $P_9$ | $M_3$ | $M_5$ | $M_2$ | | | | | | | | 3 | |
| $P_{10}$ | $M_2$ | $M_6$ | | | | | | | | | 2 | |
| $E_{MIN} = 6$, $E_{MAX} = 8$ & $NC = 3$ | | | | | | | | | | | | |

**Table 2**: Number of parts and machines for problem instances involved

| Problem | Size Source | No. of Machine Types (NM) | No. of Parts (NP) |
|---|---|---|---|
| 1. | Askin, Selim and Vakharia [4] | 10 | 19 |
| 2. | Foulds, French and Wilson [14] | 7 | 10 |
| 3. | Stanfel [42] | 14 | 24 |
| 4. | Stanfel [42] | 30 | 50 |
| 5. | Venugopal and Narendran [46] | 15 | 30 |
| 6. | Vannelli and Kumar [45] | 30 | 41 |
| 7. | Kumar, Kusiak and Vannelli [27] | 23 | 20 |
| 8. | Chandrasekharan and Ragagopalan [8] | 8 | 20 |
| 9. | Chan and Milner [10] | 10 | 15 |
| 10. | Congawave and Ham [11] | 9 | 9 |
| 11. | Congawave and Ham [11] | 9 | 9 |
| 12. | King and Nakornchai [25] | 36 | 90 |
| 13. | King [24] | 14 | 24 |
| 14. | Boe and Cheng [6] | 20 | 35 |
| 15. | Askin, Golberg, Cresswell, Strong [3] | 16 | 43 |
| 16. | De Whitte [12] | 12 | 19 |
| 17. | Burbidge [7] | 16 | 43 |
| 18. | McCormick, Schweitzer and White [31] | 27 | 27 |
| 19. | Adil, Rajamani and Strong [1] | 26 | 37 |
| 20. | Sankaran and Kasilingam [37] | 6 | 8 |

**Table 3**: Problem data and computational results

| | | | | | | XPRESS-MP | | | Tabu Search (TS) | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Prob. | NM | NP | NC | TMI | $E_{MAX}$ | Prob. Size | Obj. | CPU time | Runs | Init. Obj. | Best Obj. | Dev.††† | CPU time (secs) | Dev.†† |
| 1. | 10 | 19 | 5 | 40 | 9 | (6366×4100) | 444.13† | > 30 hrs | 9 | 623.73 | 475.6 | 24% | 102.52 | 7% |
| 2. | 7 | 10 | 4 | 20 | 6 | (2385×1555) | 219.92* | 141 secs | 5 | 339.96 | 242.83 | 29% | 18.953 | 9% |
| 3. | 14 | 24 | 5 | 37 | 8 | (10531×7030) | 383.78† | > 30 hrs | 13 | 562.36 | 428.4 | 24% | 133.97 | 12% |
| 4. | 30 | 50 | 8 | 99 | 13 | (73862×49200) | 1100.02† | > 30 hrs | 15 | 1642.3 | 1178.8 | 28% | 1945.5 | 7% |
| 5. | 15 | 30 | 7 | 47 | 7 | (19651×13141) | 582.71† | > 30 hrs | 14 | 928.48 | 641.92 | 31% | 333.3 | 10% |
| 6. | 30 | 41 | 7 | 94 | 14 | (53389×35392) | 893.28† | > 30 hrs | 15 | 1643.6 | 968.3 | 41% | 1206.2 | 8% |
| 7. | 23 | 20 | 6 | 42 | 8 | (17203×11418) | 390.51† | > 30 hrs | 10 | 667.14 | 410.51 | 38% | 275.11 | 5% |
| 8. | 8 | 20 | 5 | 31 | 7 | (6207×4152) | 333.82† | > 30 hrs | 8 | 541.48 | 385.99 | 29% | 75.938 | 16% |
| 9. | 10 | 15 | 5 | 30 | 7 | (4892×3230) | 302.25† | > 30 hrs | 7 | 501.87 | 332.3 | 34% | 58.563 | 10% |
| 10. | 9 | 9 | 5 | 29 | 6 | (2873×1815) | 242.13† | > 30 hrs | 5 | 387.07 | 271.05 | 30% | 30.422 | 12% |
| 11. | 9 | 9 | 5 | 28 | 6 | (2885×1810) | 269.11† | > 30 hrs | 5 | 407.35 | 298.65 | 27% | 33.969 | 11% |
| 12. | 36 | 90 | 10 | 147 | 17 | (197544×131980) | 1441.62† | > 30 hrs | 15 | 2872.9 | 1755.2 | 39% | 21405 | 22% |
| 13. | 14 | 24 | 5 | 45 | 10 | (10694×7070) | 508.82† | > 30 hrs | 12 | 872.71 | 557.22 | 36% | 168.83 | 10% |
| 14. | 20 | 35 | 7 | 64 | 11 | (30458×20293) | 732.37† | > 30 hrs | 15 | 1068.6 | 743.19 | 30% | 507.52 | 1% |
| 15. | 16 | 43 | 9 | 65 | 11 | (29989×20043) | 799.08† | > 30 hrs | 15 | 1283.6 | 860.51 | 33% | 825.67 | 8% |
| 16. | 12 | 19 | 5 | 29 | 7 | (7198×4800) | 293.85* | 196 secs | 9 | 406.6 | 318.01 | 22% | 55.422 | 8% |
| 17. | 16 | 43 | 8 | 61 | 10 | (29961×19999) | 713.97† | > 30 hrs | 15 | 1051.3 | 810.97 | 23% | 742.47 | 14% |
| 18. | 27 | 27 | 8 | 76 | 10 | (36663×24160) | 763.51† | > 30 hrs | 14 | 1223.5 | 816.07 | 33% | 720.03 | 7% |
| 19. | 26 | 37 | 8 | 64 | 10 | (41407×27650) | 644.41† | > 30 hrs | 15 | 1131.3 | 671.61 | 41% | 670.11 | 4% |
| 20. | 6 | 8 | 4 | 19 | 6 | (1772× 1115) | 143.04* | 71 secs | 5 | 235.87 | 156.35 | 34% | 7.875 | 4% |
| | | | | | | | | | | | | Mean Rounded Values | | |
| | | | | | | | | | | | | Dev. | CPU | Dev. |
| | | | | | | | | | | | | 31% | 1466 secs | 9% |

* Optimum solution.

† Best known solution.

†† Percentage deviation of each solution from the optimum or best known solution, obtained via XPRESS-MP, and produced as follows:

$$Dev. = round((\frac{TS\ Best\ Obj. - XPRESS{-}MP\ Best\ Obj.}{XPRESS{-}MP\ Best\ Obj.}) \times\ 100)$$

††† TS Percentage of improvement: Init. Obj. vs. Best Obj. as follows:

$$Dev. = round((\frac{Init.\ Obj. - Best\ Obj.}{Init.\ Obj.}) \times\ 100)$$