Journal of the Operational Research Society (1999) 50, 509–516 ©1999 Operational Research Society Ltd. All rights reserved. 0160-5682/99 $12.00
http://www.stockton-press.co.uk/jor

A one-step tabu search algorithm for manufacturing cell design

S Lozano[1], B Adenso-Diaz[2], I Eguia[1] and L Onieva[1]
[1]*Universidad de Sevilla and* [2]*Universidad de Oviedo, Spain*

As part of the cellular manufacturing design process, machines must be grouped in cells and the corresponding part families must be assigned. Limits on both the number of machines per cell and the number of parts per family can be considered. A weighted sum of intracell voids and intercellular moves is used to evaluate the quality of the solutions. We present a tabu search algorithm that systematically explores feasible machine cells configurations determining the corresponding part families using a linear network flow model. The performance of this tabu search is benchmarked against two simulated annealing approaches, another tabu search approach and three heuristics: (ZODIAC, GRAFICS and MST).

Keywords: cellular manufacturing; cell design; metaheuristics; tabu search

Introduction

Cellular manufacturing consists in designing and operating a portion of a firm's manufacturing system around a small number of machine cells, each one dedicated to the manufacture of a family of similar parts. The machines in a cell are physically close to one another, therefore simplifying intracell material handling. Besides, the similarity among the parts in a family allows setup reductions that bring about smaller lot sizes. All this translates into shorter lead times, reduced in-process inventories, higher throughput, improved product quality and improved productivity, among other potential benefits.[1,2]

The first step in the cellular manufacturing design process is part-machine grouping, that is, determining the part families and associated machine cells. To do this, a machine-part incidence matrix (a_{ij}) whose components take a binary value to indicate whether or not machine i is required to process part j, may be used. More detailed data would include part demands, the sequence and duration of operations, alternative process plans, machine capacities, intercell transportation costs, machine acquisition costs, and part subcontracting costs, etc. The simplest criterium used to form the cells is the minimisation of intercellular moves. Other criteria may be the maximisation of a weighted grouping measure, cost minimisation, within-cell load balancing, and intercell load balancing, etc. As for constraints, limits to cell sizes or to the number of cells may be imposed, as well as machine capacity constraints, and budget constraints, etc.

Correspondence: Dr S Lozano, Escuela Superior de Ingenieros, Camino de los Descubrimientos, s/n 41092 Sevilla, Spain.
E-mail: slozano@cica.es

The cell formation problem has attracted a great deal of research and many different procedures have been proposed, including part/machine reordering,[3] similarity coefficient-based hierarchical clustering,[4] non-hierarchical clustering,[5] neural network clustering,[6] fuzzy clustering,[7] graph theory,[8] mathematical programming[9] and heuristics.[10] Some approaches simultaneously determine part families and machine cells, while others solve the problem in two steps: firstly grouping machines and then assigning parts or viceversa.

In this paper, we first formulate the problem and review existing solution approaches. We then present a new Tabu Search (TS) algorithm which explores feasible machine cells configurations. The algorithm includes solving a linear minimum cost network flow program for part assignment and objective function evaluation. Finally, computational experience with the proposed approach is reported and conclusions are drawn.

Problem formulation and solution approaches

We consider the joint problem of simultaneously determining machine cells and associated part families. The following notation are used.

Input data

$i =$ machine index
$j =$ part index
$k =$ cell/family index
$M =$ number of machines
$P =$ number of parts
$M_{\min} =$ minimum number of machines per cell

$M_{\max}$ = maximum number of machines per cell
$P_{\min}$ = minimum number of parts per family
$P_{\max}$ = maximum number of parts per family
$C_{\max} = \min(\lfloor M/M_{\min}\rfloor, \lfloor P/P_{\min}\rfloor)$ maximum number of cells
$A = [a_{ij}]$ = binary part-machine incidence matrix.

Decision variables

$$x_{ik} = \begin{cases} 1 & \text{if machine } i \in \text{cell } k \\ 0 & \text{otherwise} \end{cases}$$

$$y_{jk} = \begin{cases} 1 & \text{if part } j \in \text{cell } k \\ 0 & \text{otherwise} \end{cases}$$

$$z_k = \begin{cases} 1 & \text{if cell } k \text{ is formed} \\ 0 & \text{otherwise} \end{cases}$$

The part-machine grouping problem may be formulated as

$$\min \sum_{k=1}^{C_{\max}} \sum_{i=1}^{M} \sum_{j=1}^{P} [q(1-a_{ij})x_{ik}y_{jk} + (1-q)a_{ij}(1-x_{ik})y_{jk}] \quad (1)$$

subject to

$$\sum_{k=1}^{C_{\max}} x_{ik} = 1 \qquad \forall i \quad (2)$$

$$\sum_{k=1}^{C_{\max}} y_{jk} = 1 \qquad \forall j \quad (3)$$

$$M_{\min} z_k \le \sum_{i=1}^{M} x_{ik} \le M_{\max} z_k \qquad \forall k \quad (4)$$

$$P_{\min} z_k \le \sum_{j=1}^{P} y_{jk} \le P_{\max} z_k \qquad \forall k \quad (5)$$

$$x_{ik}, y_{jk}, z_k = \{0,1\} \qquad i = 1,2,\ldots,M$$
$$j = 1,2,\ldots,P \qquad k = 1,2,\ldots,C_{\max}$$

The rationale behind the objective function (1) is that not only is it important to minimise intercellular moves but also to keep part families homogeneous. A common method for trading off both objectives is to minimise a weighted sum of intercellular moves and intracell voids. Therefore, the first term in (1) represents the number of intracell voids while the second corresponds to the number of intercellular moves. Choosing the value of the weighting parameter q is the responsibility of the user. Many approaches[11,12] only consider intercellular moves ($q = 0$) while others[13] assign equal weighting ($q = 0.5$) or use q in the range $(0, 0.5)$.[14,15]

Constraints (2) and (3) guarantee that no machine or part is unassigned. Constraints (4) and (5) impose the limits on the number of machines per cell and parts per family respectively. The form of these constraints is such that no fixed number of cells need to be formed. In the case of a fixed number C of cells being desired, variables z_k would no longer be needed, the index k would run from 1 through C and constraints (4) and (5) should be replaced by

$$M_{\min} \le \sum_{i=1}^{M} x_{ik} \le M_{\max} \qquad k = 1,2,\ldots,C \quad (4')$$

$$P_{\min} \le \sum_{j=1}^{P} y_{jk} \le P_{\max} \qquad k = 1,2,\ldots,C \quad (5')$$

Simulated Annealing (SA) has been used[11] to solve a formulation similar to (1)–(5) but only considering an upper limit on the number of machines per cell. The maximum number of cells is used as an independent parameter and the problem can be solved with different values of $C_{\max}$. Since no lower limit on the number of machines per cell exists, some of the $C_{\max}$ cells may become empty. In that paper[11] it was also proven that the optimal solution remains binary even if the integrity of variables y_{jk} is dropped. More specifically, the problem of assigning parts to the C machine cells that have already been formed can be stated as

$$\min \sum_{k=1}^{C} \sum_{j=1}^{P} \left[q \sum_{i=1}^{M} (1-a_{ij})x_{ik} + (1-q) \sum_{i=1}^{M} a_{ij}(1-x_{ik}) \right] y_{jk}$$
$$= \sum_{k=1}^{C} \sum_{j=1}^{P} b_{jk} y_{jk} \quad (6)$$

subject to

$$\sum_{k=1}^{C} y_{jk} = 1 \qquad j = 1,2,\ldots,P \quad (7)$$

$$P_{\min} \le \sum_{j=1}^{P} y_{jk} \le P_{\max} \qquad k = 1,2,\ldots,C \quad (8)$$

$$y_{jk} \ge 0 \qquad j = 1,2,\ldots,P \qquad k = 1,2,\ldots,C$$

It has been shown[16] that this model is equivalent to a linear network flow problem and can therefore be efficiently solved.[17] In fact, when no limits on part family sizes exist, the problem can be solved by simple inspection[18] as

$$y_{jk} = \begin{cases} 1 & \text{if } k = \arg\min_r b_{jr} \\ 0 & \text{otherwise} \end{cases} \quad (9)$$

that is, each part j is assigned to the cell k in which its contribution to the objective function b_{jk} (namely, the number of operations of part j that would need to be performed out of cell k times $1-q$ plus the number of machines in cell k that would not be visited by part j times q) is minimum. However, assuming no lower limit on the number of parts per family exists, this can result in no parts becoming assigned to a certain cell which therefore would require merging it with other existing cells, thus modifying the assumed machine cells configuration.

A different implementation of SA has also been used[12] to solve (1)–(5), again considering only an upper limit on the number of machines per cell and not considering limits on part family sizes. The main differences lie in the neighborhood definition (which induces a different topology) as well as in the initial solution and the cooling schedule.

A TS algorithm for solving a quadratic assignment model with upper limits on the number of machines per cell has also been proposed.[19] The objective function used is the minimisation of intercellular material flow, which is analogous to (1) if $q = 0$. In addition to the basic TS algorithm, an enhanced variant is implemented using a look-ahead strategy that instead of selecting the single best, non-tabu move as the next move, several candidate moves are explored to see if their corresponding next move leads to a final improved value of the objective function.

The approach we propose in this paper is to solve the quadratic programming model (1)–(5) using a TS algorithm that is described in detail in the following section. The algorithm searches the machine cells configuration space, solving model (6)–(8) for each configuration to form the associated part families and to compute the value of the objective function. Therefore, whenever the machine cells configuration changes, parts are reassigned and the new value of the objective function is computed. To solve (6)–(8) we have used the RELAX IV code of the Relaxation algorithm.[20]

TS-ONE algorithm: implementation details

Tabu Search (TS) is a metaheuristic procedure that has proven itself to be a useful optimisation technique for solving a wide variety of combinatorial problems.[21] In recent years TS literature has grown rapidly in the production management field, with some promising results in layout[22,23] and group technology.[19,24] For a thorough description of TS, we refer the reader to the books[25,26] and tutorials[27] that cover this technique.

In TS, the crucial implementation decisions include the definition of the neighborhood structure, and the way the actual solution is modified according to the history of the search. The representation we have used consists in a permutation of the M machines and a partition of this permutation into $C_{\max}$ subsets each with a cardinality c_k. Therefore the solution space is

$$\Omega = \{\langle \sigma, \langle c_1, \ldots, C_{C\max} \rangle\rangle;\ \sigma \text{ is a permutation of } M \text{ machines}, c_k \geq 0, \textstyle\sum c_k = M, c_k = 0 \Rightarrow c_t = 0\ \forall t = k+1, \ldots, C_{\max}\}$$

Note that with this definition, different elements in Ω may represent the same machine cells configuration. However, the inconvenience, which can be attenuated by means of an efficient implementation of the search, results in a simplification of the treatment of movements.

Given a solution $S = \langle \sigma, \{c_k\}\rangle \in \Omega$, different types of solutions in the neighborhood $N(S)$ of S (Figure 1) may be considered:

(a) *Exchange of machines*, $N_E(S)$. Two machines belonging to different cells are exchanged, this move will not modify any of the values c_k of S, only the permutation σ will be modified.
(b) *Insertion of a machine in a different cell*, $N_I(S)$. A machine is deleted from its cell and included in a different one. The cardinality of the source cell must be greater than the minimum number of machines per cell $M_{\min}$. In addition, the cardinality of the target cell must be less than the maximum number of machines per cell $M_{\max}$. The move will increment the cardinality of the target cell by one unit, reducing that of the source cell in the same quantity.
(c) *Union of two cells*, $N_U(S)$. All the machines from two different cells are put together. The cardinality of the joint cell is the sum of the cardinalities of the joining cells and cannot exceed $M_{\max}$.
(d) *Splitting of a cell*, $N_S(S)$. To allow an increase in the number of cells in the search process, we consider the possibility of splitting one of the cells into two. This is only possible if the cardinality of the cell to be split is large enough to produce two cells with more than $M_{\min}$ machines each. The effect over values c_k is exactly the opposite to the union of two cells.

We define neighborhood $N(S)$ as the union of all these types of neighborhoods, $N(S) = N_E(S) \cup N_I(S) \cup N_U(S) \cup N_S(S)$. Let C be the actual number of cells in S. Then, $\text{card}(N_I(S)) = M(C-1)$, $\text{card}(N_U(S)) = C(C-1)/2$, and $\text{card}(N_E(S)) = \sum_k c_k \sum_{t>k} c_t \leq (C-1)M^2/2C$. Note that these three cardinalities only depend on the number of cells and the number of machines which are not large in real problems. Therefore, it is not necessary to define partial explorations of these neighborhoods through candidate lists, a complete exploration being possible. In addition, not all

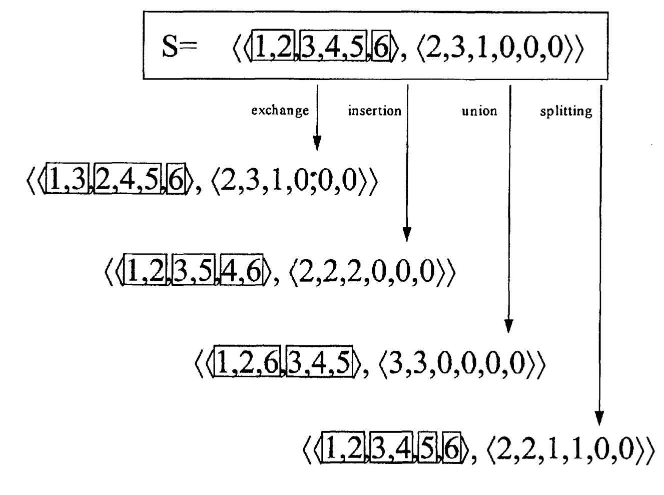

Figure 1 Example of solutions in $N(S)$ for a solution S with 3 cells and 6 machines.

the elements in $N(S)$ must be considered, therefore reducing the number of candidates moves to evaluate. For example, when a cell has just one machine, an insertion movement of that machine in another cell is equivalent to a union movement or if the machine is exchanged with a second one in another one-machine cell, the new solution would be equal to the current one. The savings that can be obtained in this way depend on the current solution and are usually within 10–30% of the total neighborhood size.

In relation to the neighborhood $N_S(S)$, its cardinality is somewhat larger $\text{card}(N_S(S)) = (\sum_k 2^{c_k}) - 2M \leq 2^{M-C+1} - 2(M - C + 1)$ and good results were obtained when it was considered only after a number δ of iterations with no improvement. For diversification purposes, that is in order to escape from an already explored region, it is interesting to search solutions with a larger number of cells. Therefore, after η iterations without improvement, a new solution is generated as current having a number of cells half way between the current number and $C_{\max}$, just changing the values c_k, and leaving the permutation σ as it is.

Any feasible solution may be used as the initial solution. In the case of a feasible solution provided by a heuristic method being available, it is recommendable to use it as the starting solution because starting the search from a good enough solution usually leads to a better performance (both in terms of solution quality and computing time) than starting from a random solution. In our case, we have used the solution provided by the fast Maximum Spanning Tree (MST) heuristic which is described in the appendix.

Once the best valid solution $S' \in N(S)$ has been selected, the move that transforms S into S' is considered so as to update the corresponding tabu list. We have defined three different tabu lists, one for each type of neighborhood $N_E(S), N_I(S), N_U(S)$. Since $N_S(S)$ is used very sporadically it does not require this short-term control tool. When validating a movement, the corresponding list is checked according to the type of movement. In the case of exchange, we try to avoid the reverse exchange of machines; in the case of insertion, we try to avoid the opposite insertion; finally, in the case of the union of two cells in a unique cell, we try to avoid joining the same machines again. Each of these lists has its own tabu tenure. In our implementation we use the simple aspiration criterion that overrules the tabu restriction if the candidate solution is better than the current best encountered one.

We present a pseudocode for the algorithm in Figure 2. Note that the search continues until a predefined number of iterations, γ, with no improvements occurs. In our implementation, for simplicity, we opted for making both η and δ dependent on γ. The latter must be selected trading-off computing time and solution quality, both of which increase with γ. In general, a fractional factorial design methodology may be required to tune the parameters used by the algorithm in its calculation process.

```
S^act:=Define_initial_solution;
S^best:=S^act;
tabu_list(i):=∅, ∀i=1..3;
REPEAT
  iteration:=iteration+1;
  Generate {S1∈N_E: S1 obtained through exchange movement allowed};
  Generate {S2∈N_I: S2 obtained through insertion movement allowed};
  Generate {S3∈N_U: S3 obtained through union movement allowed};
  IF number_iterations_with_no_improvement>δ THEN
    Generate {S4∈N_S: S4 obtained through splitting movement allowed};
  Select best solution S' from all those generated, S'=min{S1,..,S4};
  IF number_iterations_with_no_improvement>η THEN S'=Diversifies(S);
  Update corresponding tabu_list(i), i≤3;
  S^act:=S';
  IF value(S^act)<value(S^best) THEN S^best:=S^act;
UNTIL number_iterations_with_no_improvement>γ;
WRITE(S^best);
```

Figure 2 Pseudocode for TS-ONE algorithm.

Computational experiences

The first set of experiments that will be reported here involved ten problems with 16 machines and 30 part types.[11] Each problem was solved with $q = 0$ and with a maximum number of machines per cell $M_{\max}$ varying from 6 through 12 in steps of 1, giving a total of 70 instances. Optimal objective function values are available,[11] although in that reference there must be an error for problem 3 and $M_{\max}$ 6 since the published optimal solution is 9 and we were able to find a feasible solution with an objective function value of 8.

These problems were resolved using two SA approaches,[11,12] a TS approach[19] and the proposed approach TS-ONE. The values used for TS-ONE parameters were $\gamma = 60, \eta = 0.5\gamma, \delta = 0.2\gamma$. These values were chosen after performing a restricted experimental design using a subset of the instance data set. To guarantee the selection of the correct parameter values from a theoretical point of view, we have considered the Taguchi methodology[28] using orthogonal arrays of type $L_{9,3^4}$. As for the length of the different tabu lists, they were also selected experimentally. Since it was observed that the results were more sensitive to the tabu tenure of exchange moves (too low a value not being effective in preventing cycling) than to those of the other types, the former was made variable with the number of machines (exactly, one third of the number of machines) while for the rest a constant value (exactly, 20) was used.

Of the four solution methods tested, our approach is the only one that does not require the maximum number of cells $C_{\max}$ as input. For the other three methods the maximum number of cells was fixed at 3 for $M_{\max}$ 6, 7, 8 and 9 and was fixed at 2 for $M_{\max}$ 10, 11 and 12. Since the two SA approaches are stochastic, ten runs were performed for each of the 70 instances. Table 1 shows the best, average and maximum percentage error over the optimal solution for these approaches as well as the average and maximum percentage errors for the two TS approaches and for the MST heuristic. Note that each row corresponds to seven instances corresponding to the different values of $M_{\max}$. The

Table 1 Percentage error over optimal solution

	SA-BOCTOR			*SA-CHEN*			*TS-SUN*		*MST*		*TS-ONE*	
Problem	*Best*	*Aver.*	*Max.*	*Best*	*Aver.*	*Max.*	*Aver.*	*Max.*	*Aver.*	*Max.*	*Aver.*	*Max.*
1	0.8	19.6	145.5	0.8	1.5	11.1	2.1	9.1	252.1	400.0	0.8	5.6
2	2.4	30.6	333.3	0.0	0.2	16.7	0.0	0.0	38.9	66.7	0.0	0.0
3	6.5	76.9	300.0	1.8	7.8	100.0	6.5	33.3	4.8	33.3	0.0	0.0
4	0.0	14.5	76.9	0.0	0.1	3.7	5.5	23.1	102.1	171.4	0.0	0.0
5	0.0	24.3	160.0	0.0	5.5	50.0	8.9	50.0	75.2	150.0	2.9	20.0
6	4.8	35.0	150.0	0.0	7.5	50.0	21.4	66.7	47.6	100.0	0.0	0.0
7	5.7	57.6	400.0	0.0	5.1	50.0	0.0	0.0	67.5	100.0	0.0	0.0
8	6.3	31.8	181.8	5.0	6.0	27.3	5.0	10.0	169.7	340.0	5.0	10.0
9	9.5	57.5	125.0	0.0	2.3	62.5	0.0	0.0	105.8	160.0	8.6	60.0
10	0.0	49.5	220.0	0.0	1.4	40.0	10.7	75.0	67.1	100.0	0.0	0.0
All	3.6	39.7	400.0	0.8	3.7	100.0	6.0	75.0	93.1	400.0	1.7	60.0

column labelled *Aver.* represents the average percentage error over all instances in a row. Therefore, for the single-run approaches, it is the average of 7 values, while for the SA approaches, it is the average of 70 values. Analogously, the column labelled *Max.* represents the maximum percentage error over all instances in a row. Therefore, for the single-run approaches, it is the maximum of 7 values, while for the SA approaches, it is the maximum of 70 values. Finally, for the SA approaches, the column labelled *Best* represents the average percentage error over all instances in a row given by the best run.

Note that TS-ONE outperforms TS-SUN as well as both the best run and the average run of SA-BOCTOR. Although SA-CHEN is also outperformed on average, its best run has a smaller error than TS-ONE. As for maximum errors, TS-ONE is best and SA-BOCTOR worst. The solution provided by MST is generally poor but good enough to initialise the TS algorithm. The difference in quality between MST and TS-ONE must be credited to the TS algorithm.

Figure 3 shows CPU times on an Intel Pentium 133 MHz PC *vs* percentage error. It can be seen that, although fast, MST can have large errors. Although the analysis of MST is not the object of this paper, it can theoretically be argued and experiments confirm that, due to its greedy nature, MST generally gives worse solutions when $q = 0$ than when $q > 0$. The results show that SA-BOCTOR is also unreliable. SA-CHEN is much better, although slower and TS-SUN is somewhat in between. Finally, TS-ONE is the best of all methods and it is much faster than second-placed SA-CHEN.

We claim that one reason for TS-ONE outperforming the other metaheuristics is that it uses a more complex neighborhood than the others. SA-CHEN uses just the insert move while SA-BOCTOR and TS-SUN only use variants of the insert move. Although exploring a larger neighborhood should take longer, the existence of more paths in the solution space allows the algorithm to search more effectively so that with the same or less computing effort it can obtain better solutions than the other methods.

The aim of the second set of experiments was to demonstrate that TS-ONE is able to solve a more general problem than the other aforementioned SA and TS methods. Table 2 shows six problems from the literature, with their sizes and references. These problems were selected because their natural clusters involved either single-machine cells or single-part families or large cells or large families, so that imposing limits on machine cell size and on part family size was deemed convenient. The problems were first solved for a value of $q = 0.1$, without size constraints, using TS-ONE and three heuristics: MST, ZODIAC[5] and GRAFICS.[32] MST (see Appendix) is a greedy heuristic which can handle cell size constraints while the other two heuristics are two commonly-used,[33,34] non-hierarchical clustering approaches for manufacturing cell formation.

The values for all TS-ONE parameters were the same as for the first set of problems. The results are shown in Table 3. Note that although faster than MST, both ZODIAC and GRAFICS usually give worse solutions. TS-ONE gives the best solutions although it requires more computing time. Note that almost always TS-ONE improves the initial MST solution and also that, as in the first set of experiments, the TS-ONE computing times shown include the time to obtain the MST solution from which it starts.

The same problems were solved again using $q = 0.1$ but forbidding single-machine cells and machine cells with more than 10 machines as well as single-part families and part-families composed of more than 30 parts. This time only MST and TS-ONE could be used since the other methods cannot impose those cell size constraints. The results are shown in Table 4. Note how imposing cell size constraints leads to higher value of the objective function (1). Again, TS-ONE improves the initial MST solution requiring relatively low computing times. The reason why, for the same problem, the computing times in Table 4 are lower than in Table 3 is that the existence of the constraints reduces the feasible region, therefore easying the search.

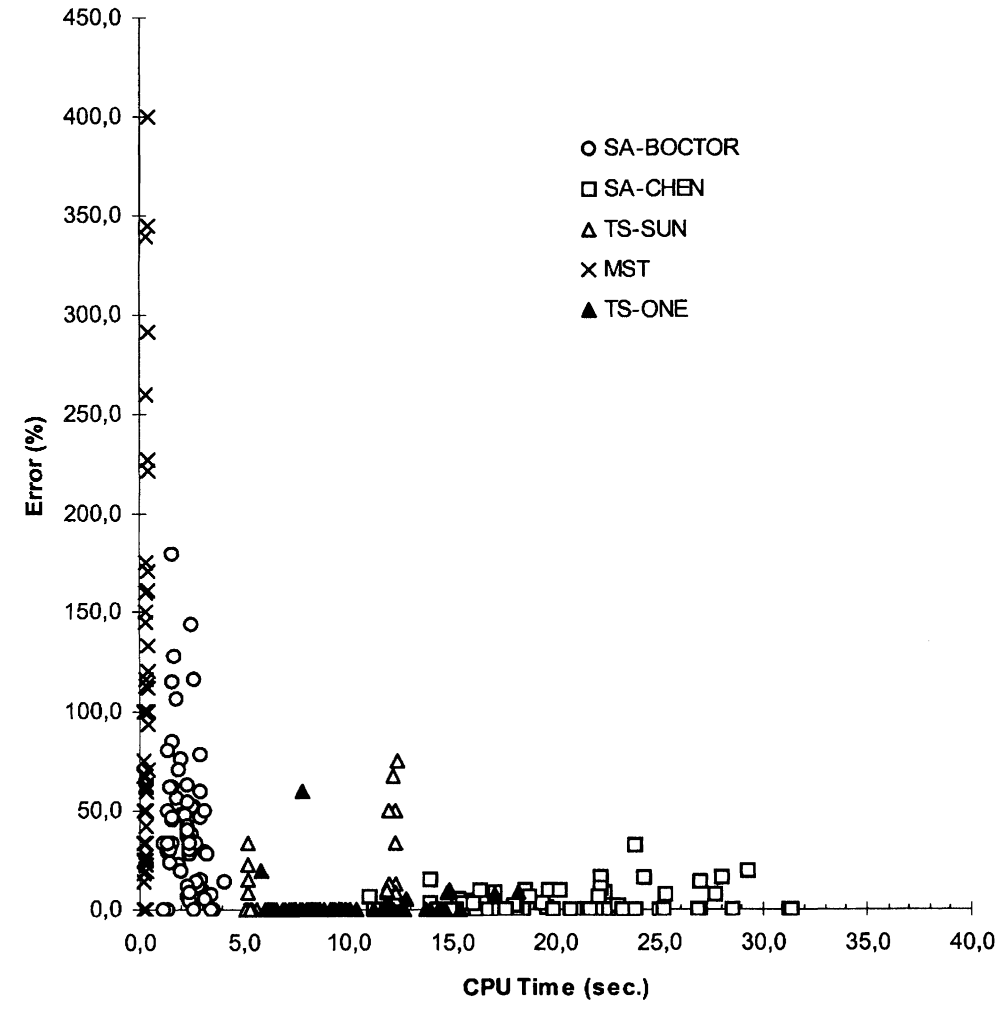

Figure 3 CPU time *vs* percentage error.

Conclusions

This paper presents a one-step approach to part-machine grouping with limits to both machine cells and part families sizes. A weighted sum of intracell voids and intercellular moves is used to evaluate the quality of the solution obtained. Since the resulting quadratic integer programming is difficult to solve, we implemented a TS algorithm which was benchmarked against different SA algorithms, heuristics and another TS algorithm. Computational experiences show TS-ONE to be superior to the other methods since it can solve a more general problem, finds better solutions and does not require longer computing times. One of the reasons for this performance is the neighborhood used which includes not only the common machine insertion but also machines exchange and cells union and splitting. Possible extensions of this work are: using the sequence of operations of the part types, including multiple process plans and considering more than one objective function (for example, workload balancing).

Table 2 Some problems from the literature

Problem	*No. machine*	*No. parts*	*Reference*
ADI97	26	37	15
KIN82B	30	90	3
KUM86	23	20	8
MCC72A	27	27	29
VEN90B	20	23	30
WEI89	30	41	31

Appendix

This appendix describes the Maximum Spanning Tree (MST) heuristic used in this paper. The use of MST-based methods for clustering has long been recognized.[17] For cell formation, several MST-based methods[34–37] have been proposed, each one using a particular measure of similarity/dissimilarity and a particular objective function. Our version of MST is aimed at the objective function (1),

Table 3 Results for $q=0.1$, $M_{\min}=1$, $M_{\max}=M$, $P_{\min}=1$, $P_{\max}=P$

	ZODIAC		*GRAFICS*		*MST*		*TS-ONE*	
Problem	*Obj. func.*	*CPU time*	*Obj. func.*	*CPU time*	*Obj. func.*	*CPU time*	*Obj. func.*	*CPU time*
ADI97	42.8	0.33	43.4	0.44	43.9	2.31	42.4	67.89
KIN82B	143.4	2.86	129.0	0.99	110.8	3.74	97.0	436.54
KUM86	37.5	0.11	37.1	0.44	34.9	1.10	30.7	21.37
MCC72A	53.1	0.16	57.5	0.49	42.1	1.32	41.5	28.89
VEN90B	27.5	0.05	34.7	0.38	27.5	0.44	27.5	10.38
WEI89	26.8	0.38	33.9	0.60	22.3	2.75	21.7	140.12

which led us to define weighted similarity coefficients $s_{ij}(q)$ between every pair of machines i and j

$$s_{ij}(q)=\frac{(1-q)n_{ij}-q(\tilde{n}_{ij}+\tilde{n}_{ji})}{n_{ij}+\tilde{n}_{ij}+\tilde{n}_{ji}}=\frac{n_{ij}}{n_{ij}+\tilde{n}_{ij}+\tilde{n}_{ji}}-q$$
$$=s_{ij}(0)-q \qquad (10)$$

where n_{ij} is the number of parts requiring processing on both machines i and j, while $\tilde{n}_{ij}$ is the number of parts requiring processing on machine i but not on machine j. Note that $-q \le s_{ij}(q) \le 1-q$ and that they are a generalisation of Jaccard similarity coefficients[4] which correspond to the special case $q=0$. Recall that parameter $0 \le q \le 1$ represents the weight assigned to intracell voids in the objective function (1); correspondingly, intercellular moves are assigned a weight $1-q$. Since n_{ij} is the number of parts that are processed by both machines, if they do not fall into the same cell, this number of intercellular moves may be due. Therefore, n_{ij} is given weight $1-q$. On the contrary, if machines i and j fall into the same cell, $\tilde{n}_{ij}+\tilde{n}_{ji}$ represents the number of voids due. Since $s_{ij}(q)$ is used to decide whether i and j should fall into the same cell, the first term in the numerator is positive while the second is subtracting.

MST attempts to maximise the total sum of intracell pairwise similarity coefficients. To do this, it considers a fully connected, nondirected graph with as many nodes as machines and $s_{ij}(q)$ as arc weights. Then MST finds a Maximum Spanning Tree, that is, cycle-free set of $M-1$ arcs that connects all nodes and whose sum of weights is maximum. From this one-cell solution, the heuristic finds the arc whose deletion (resulting in the formation of one more cell) brings about the maximum increase in the objective function and does not violate cell size constraints. This process of deleting one arc at a time ends when no arc can be deleted without decreasing the objective function or violating cell size constraints.

This greedy heuristic is similar to a proposed heuristic[35] that uses Jaccard similarity coefficients and grouping efficiency[5] as objective function. It is also similar to another approach[36] that uses cosine similarity coefficients and minimisation of set-up times as objective function and to another approach[34] which uses Hamming distance and feeds the MST solution to a GRAFICS[32]-like process of alternate seed generation and clustering. Finally, another approach[37] uses a cost minimisation objective function and defines workload-based dissimilarity coefficients.

Table 4 Results for $q=0.1$, $M_{\min}=2$, $M_{\max}=10$, $P_{\min}=2$, $P_{\max}=30$

	MST		*TS-ONE*	
Problem	*Obj. func.*	*CPU time*	*Obj. func.*	*CPU time*
ADI97	45.6	0.99	42.6	50.64
KIN82B	117.7	1.76	102.7	289.35
KUM86	51.2	0.60	35.6	14.94
MCC72A	86.9	1.04	65.8	21.31
VEN90B	40.9	0.33	33.7	11.98
WEI89	23.2	1.43	21.8	105.13

Acknowledgements—The authors would like to thank Dr Frangioni from the University of Pisa for providing them with his C++ implementation of the RelaxIV algorithm for Min Cost Flow problems. Also, they would like to thank the referees for their helpful suggestions and comments.

References

1 Wemmerlöv U and Hyer NL (1989). Cellular manufacturing in the US industry: a survey of users. *Int J Prod Res* **27**: 1511–1530.
2 Singh N (1993). Design of cellular manufacturing systems: An invited review. *Eur J Opl Res* **69**: 284–291.
3 King JR and Nakornchai V (1982). Machine-component group formation in group technology: review and extension. *Int J Prod Res* **20**: 117–133.
4 McAuley J (1972). Machine grouping for efficient production. *Prod Engng* **51**: 53–57.
5 Chandrasekharan MP and Rajagopalan R (1987). ZODIAC—an algorithm for concurrent formation of part-families and machine-cells. *Int J Prod Res* **25**: 835–850.
6 Lozano S, Onieva L, Larrañeta J and Teba J (1993). A neural network approach to part-machine grouping in GT manufacturing. In: Takamori T and Tsuchiya K (eds). *Robotics, Mechatronics and Manufacturing Systems*. Elsevier: Amsterdam, pp 619–624.
7 Xu H and Wang HP (1989). Part family formation for GT applications based on fuzzy mathematics. *Int J Proc Res* **27**: 1637–1651.
8 Kumar KR, Kusiak A and Vannelli A (1986). Grouping of parts and components in flexible manufacturing systems. *Eur J Opl Res* **24**: 387–397.

9 Kusiak A (1985). The part families problem in flexible manufacturing systems. *Annals Opns Res* **3**: 279–300.

10 Del Valle AG, Balarezo S and Tejero J (1994). A heuristic workload-based model to form cells by minimizing intercellular movements. *Int J Prod Res* **32**: 2275–2285.

11 Boctor FF (1991). A linear formulation of the machine-part cell formation problem. *Int J Prod Res* **29**: 343–356.

12 Chen CL, Cotruvo NA and Baek W (1995). A simulated annealing solution to the cell formation problem. *Int J Prod Res* **33**: 2601–2614.

13 Srinivasan G, Narendran TT and Mahadevan B (1990). An assignment model for the part-families problem in group technology. *Int J Prod Res* **28**: 145–152.

14 Boe WJ and Cheng CH (1991). A close neighbor algorithm for designing cellular manufacturing systems. *Int J Prod Res* **29**: 2097–2116.

15 Adil GK, Rajamani D and Strong D (1997). Assignment allocation and simulated annealing algorithms for cell formation. *IIE Trans* **29**: 53–67.

16 Lee H and Garcia-Diaz A (1996). Network flow procedures for the analysis of cellular manufacturing systems. *IIE Trans* **28**: 333–345.

17 Ahuja RK, Magnanti TL and Orlin JB (1993). *Network Flows. Theory, Algorithms and Applications*. Prentice Hall: Englewood Cliffs.

18 Chen WH and Srivastava B (1994). Simulated annealing procedures for forming machine cells in group technology. *Eur J Opl Res* **75**: 100–111.

19 Sun D, Lin L and Batta R (1995). Cell formation using tabu search. *Comp Ind Engng* **28**: 485–494.

20 Bertsekas D and Tseng P (1988). Relaxation methods for minimum cost for ordinary and generalized network flow problems. *Opns Res* **36**: 93–114.

21 Glover F (1986). Future paths for integer programming and links to artificial intelligence. *Comp Opns Res* **13**: 533–549.

22 Chiang W-C and Kouvelis P (1996). An improved tabu search heuristic for solving facility layout design. *Int J Prod Res* **34**: 2565–2585.

23 Heragu SS (1992). Recent models and techniques for solving the layout problem. *Eur J Opl Res* **57**: 136–144.

24 Aljaber N, Wonjang B and Chuen-Lung Ch (1997). A Tabu Search approach to the cell formation problem. *Comp & Ind Engng* **32**: 169–185.

25 Glover F and Laguna M (1995). Tabu Search. In: Reeves CR (ed). *Modern Heuristic Techniques for Combinatorial Problems*. McGraw-Hill: London, pp 70–150.

26 Glover F and Laguna M (1997). *Tabu Search*. Kluwer Academic Press: Boston.

27 Glover F (1990). Tabu Search: A tutorial. *Interfaces* **20**: 74–94.

28 Roy RK (1990). *A Primer on the Taguchi Method*. Van Nostrand Reinhold: New York.

29 McCormick WT, Schweitzer PJ and White TW (1972). Problem decomposition and data reorganization by a clustering technique. *Opns Res* **20**: 993–1009.

30 Ventura JA, Chen FF and Wu CH (1990). Grouping parts and tools in flexible manufacturing systems production planning. *Int J Prod Res* **28**: 1039–1056.

31 Wei JC and Kern GM (1989). Commonality analysis: A linear cell clustering algorithm for group technology. *Int J Prod Res* **27**: 2053–2062.

32 Srinivasan G and Narendran TT (1991). GRAFICS—a nonhierarchical clustering algorithm for group technology. *Int J Prod Res* **29**: 463–478.

33 Kandiller L (1994). A comparative study of cell formation in cellular manufacturing systems. *Int J Prod Res* **32**: 2395–2429.

34 Srinivasan G (1994). A clustering algorithm for machine cell formation in group technology using minimum spanning trees. *Int J Prod Res* **32**: 2149–2158.

35 Ng SM (1996). On the characterization of machine cells in group technology. *Opns Res* **44**: 735–744.

36 Bhaskar G and Narendran TT (1996). Grouping PCBs for set-up reduction: a maximum spanning tree approach. *Int J Prod Res* **34**: 621–632.

37 Lin TL, Dessouky MM, Kumar KR and Ng SM (1996). A heuristic-based procedure for the weighted production-cell formation problem. *IIE Trans* **28**: 579–589.

Received March 1998;
accepted November 1998 after two revisions