3. Group Technology / Cellular Manufacturing[1]

3.1 Introduction

As early as in the 1920ies it was observed, that using product-oriented departments to manufacture standardized products in machine companies lead to reduced transportation. This can be considered the start of ***Group Technology*** (GT). Parts are classified and parts with similar features are manufactured together with standardized processes. As a consequence, small "focused factories" are being created as independent operating units within large facilities.

More generally, Group Technology can be considered a theory of management based on the principle that *"similar things should be done similarly"*. In our context, "things" include product design, process planning, fabrication, assembly, and production control. However, in a more general sense GT may be applied to all activities, including administrative functions.

The principle of group technology is to divide the manufacturing facility into small groups or *cells* of machines. The term ***cellular manufacturing*** is often used in this regard. Each of these cells is dedicated to a specified family or set of part types. Typically, a cell is a small group of machines (as a rule of thumb not more than five). An example would be a machining center with inspection and monitoring devices, tool and Part Storage, a robot for part handling, and the associated control hardware.

The idea of GT can also be used to build larger groups, such as for instance, a department, possibly composed of several automated cells or several manned machines of various types. As mentioned in Chapter 1 (see also Figure 1.5) pure item flow lines are possible, if volumes are very large. If volumes are very small, and parts are very different, a functional layout (job shop) is usually appropriate. In the intermediate case of *medium-variety, medium-volume* environments, group configuration is most appropriate.

GT can produce considerable improvements where it is appropriate and the basic idea can be utilized in all manufacturing environments:

- To the *manufacturing engineer* GT can be viewed as a role model to obtain the advantages of flow line systems in environments previously ruled by job shop layouts. The idea is to form groups and to aim at a product-type layout within each group (for a family of parts). Whenever possible, *new parts* are designed to be compatible with the processes and tooling of an existing part family. This way, production experience is quickly obtained, and standard process plans and tooling can be developed for this restricted part set.
- To the *design engineer* the idea of GT can mean to standardize products and process plans. If a new part should be designed, first retrieve the design for a similar, existing part. Maybe, the need for the new part is eliminated if an existing part will suffice. If a new part is actually needed, the new plan can be developed quickly by relying on decisions and documentation previously made for similar parts. Hence, the resulting plan will match current manufacturing procedures and document preparation time is reduced. The design engineer is freed to concentrate on optimal design.

[1] This chapter is based on Chapter 6 of Askin & Standridge (1993). It is recommended to read this chapter parallel to the course notes.

In this GT context a typical approach would be the use of composite Part families. Consider e.g. the parts family shown in Figure 3.1.

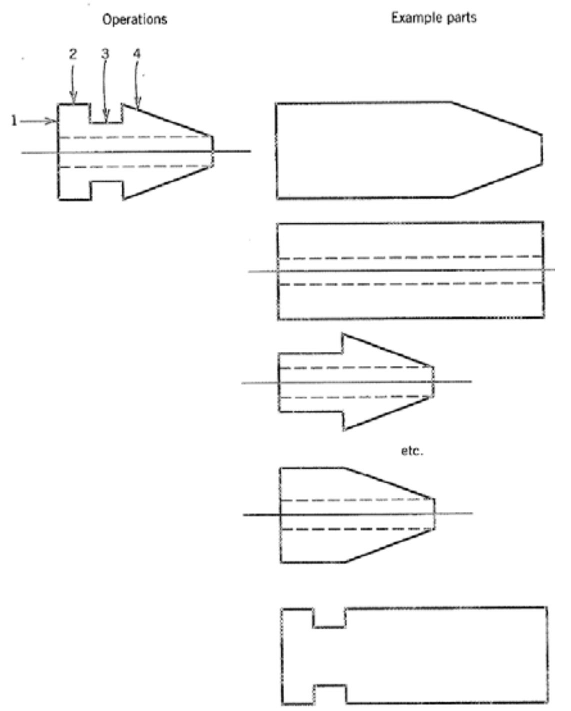


Figure 3.1. Composite Group Technology Part
(Askin & Standridge, 1993, p. 165).

The parameter values for the features of this single part family have the same allowable ranges. Each part in the family requires the same set of machines and tools; in our example: turning/lathing (Drehbank), internal drilling (Bohrmaschine), face milling (Planfräsen), etc.

Raw material should be reasonably consistent (e.g. plastic and metallic parts require different manufacturing operations and should not be in the same family).

Fixtures can be designed that are capable of supporting all the actual realizations of the composite parts within the family.

Standard machine setups are often possible with little or no changeover required between the different parts within the family (same material, same fixture method, similar size, same tools/machines required).

In the *functional process* (job shop) layout, all parts travel through the *entire shop*. Scheduling and material control are complicated. Job priorities are difficult to set, and large WIP inventories are used to assure reasonable capacity utilisation. In *GT*, each part type flows only through its specific group area. The reduced setup time allows faster adjustment to changing conditions.

Often, workers are cross-trained on all machines within the group and follow the job from Start to finish. This usually leads to higher job satisfaction/motivation and higher efficiency.

For smaller-volume part families it may be necessary to include several such part families in a machine group to justify machine utilization.

One can identify three *different types group layout*:

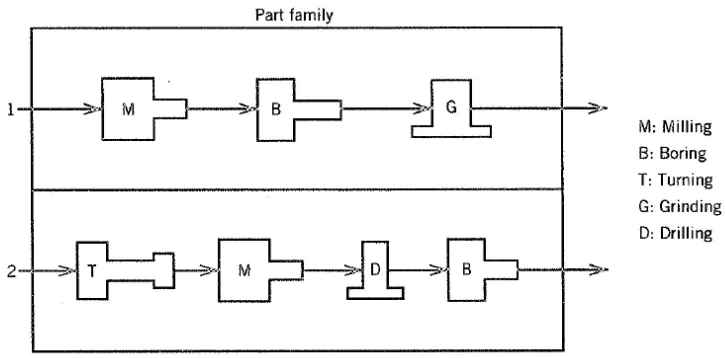


Figure 3.2a. GT flow line
(Askin & Standridge, 1993, p. 167).

In a ***GT flow line*** concept all parts assigned to a group follow the same machine sequence and require relatively proportional time requirements on each machine.

The GT flow line operates as a *mixed-product assembly line* system; see Figure 3.2a. Automated transfer mechanisms may be possible. See also Chapter 4 for mixed-product assembly lines.

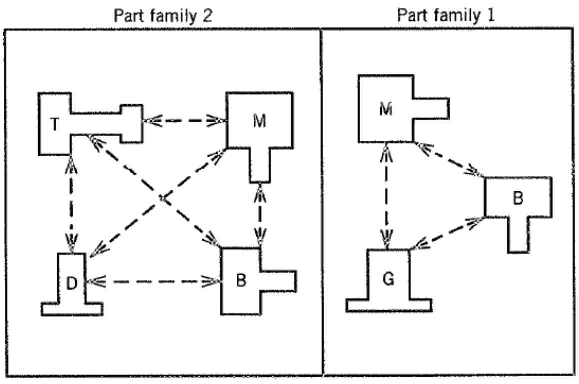


Figure 3.2b. GT cell
(Askin & Standridge, 1993, p. 167).

The *classical* ***GT cell*** allows parts to move from any machine to any other machine. Flow is not unidirectional. However, since machines are located in close proximity short and fast transfer is possible.

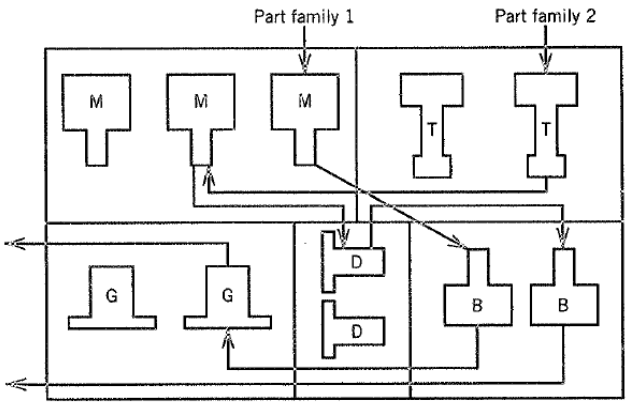


Figure 3.2c. GT center
(Askin & Standridge, 1993, p. 167).

The ***GT center*** may be appropriate when

- large machines have already been located and cannot be moved, or
- product mix and part families are dynamic and would require frequent relayout.

Then, machines may be located as in a process layout by using functional departments (job shops), but each machine is dedicated to producing only certain Part families. This way, only the tooling and control advantages of GT can be achieved. Compared to a GT cell layout, increased material handling is necessary.

GT offers numerous benefits w.r.t. throughput time, WIP inventory, materials handling, job satisfaction, fixtures, setup time, space needs, quality, finished goods, and labor cost; **read also Chapter 6.1 of Askin & Standridge, 1993.**

In general, GT simplifies and standardizes. The approach to simplify, standardize, and internalize through repetition produces efficiency.

Since a workcenter will work only on a family of similar parts generic fixtures can be developed and used. Tooling can be stored locally since parts will always be processed through the same machines. Tool changes may be required due to tool wear only, not part changeovers (e.g. a press may have a generic fixture that can hold all the parts in a family without any change or simply by changing a part-specific insert secured by a single screw. Hence ***setup time*** is reduced, and tooling cost is reduced. Using queuing theory (M/M/1 model) it is possible to show that if setup time is reduced, also the throughput time for the system is reduced by the same percentage.

3.2 How to form groups

Askin & Standridge, 1993, Chapter 6.2 provides a list of seven characteristics of successful groups:

Characteristic	**Description**
Team	specified team of dedicated workers
Products	specified set of products and no others
Facilities	specified set of (mainly) dedicated machines equipment
Group layout	dedicated contiguous space for specified facilities
Target	common group goal, established at start of each period
Independence	buffers between groups; groups can reach goals independently
Size	Preferably 6-15 workers (small enough to act as a team with a common goal; large enough to contain all necessary resources)

Clearly, also the organization should be structured around groups. Each group performs functions that in many cases were previously attributed to different functional departments. For instance, in most situations employee bonuses should be based on group performance.

Worker empowerment is an important aspect of manned cells. Exchanging ideas and work load is necessary. Many groups are allocated the responsibility for individual work assignments. By cross-training of technical skills, at least two workers can perform each task and all workers can perform multiple tasks. Hence the there is some flexibility in work assignments.

The group should be an independent profit center in some sense. It should also retain the responsibility for its performance and authority to affect that performance. The group is a single entity and must act together to resolve problems.

There are three basic steps in group technology planning:

1. coding
2. classification
3. layout.

These will be discussed in separate subsections.

3.3 Coding schemes

The knowledge concerning the similarities between parts must be coded somehow. This will facilitate determination and retrieval of similar parts. Often this involves the assignment of a symbolic or numerical description to parts (part number) based on their design and manufacturing characteristics. However, it may also simply mean listing the machines used by each part.

There are four major issues in the construction of a coding system:

- part (component) population
- code detail
- code structure, and
- (digital) representation.

Numerous codes exist, including Brisch-Birn, MULTICLASS, and KK-3. One of the most widely used coding systems is OPITZ. Many firms customize existing coding systems to their specific needs. Important aspects are

- The code should be sufficiently flexible to handle future as well as current parts.
- The scope of part types to be included must be known (e.g. are the parts rotational, prismatic, sheet metal, etc.?)
- To be useful, the code must discriminate between parts with different values for key attributes (material, tolerances, required machines, etc.)

Code detail is crucial to the success of the coding project. Ideal is a short code that uniquely identifies each part and fully describes the part from design and manufacturing viewpoints,

- Too much detail results in cumbersome codes and the waste of resources in data collection.
- With too few details and the code becomes useless.

As a general rule, all information necessary for grouping the part for manufacturing should be included in the code whenever possible. Features like outside shape, end shape, internal shape, holes, and dimensions are typically included in the coding scheme.

W.r.t. ***code structure***, codes are generally classified as, hierarchical (also called monocode), chain (also called polycode), or hybrid. This is explained in Figure 3.3 (taken from Askin & Standridge, 1993).

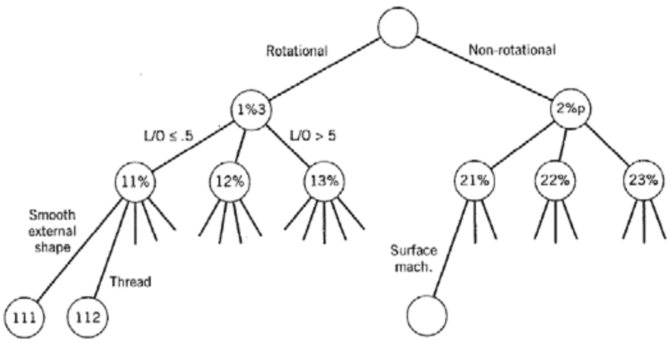


Figure 3.3a. Hierarchical structure.

Hierarchical code structure: the meaning of a digit in the code depends on the values of preceding digits. The value of 3 in the third place may indicate

- the existence of internal threads in a rotational part: "12**3**2"
- a smooth internal feature: "21**3**2"

Hierarchical codes are efficient; they only consider relevant information at each digit. But they are difficult to learn because of the large number of conditional inferences.

Code Digit Feature	1 Outside shape	2 Inside shape	3 Holes	4 Surface Machining	...
Value					
1	None	None	No	None	
2	Smooth	Smooth	Smooth axial	External groove	
3	Stepped ends	Stepped ends	Smooth radial	External spline	
4	Stepped and threads	Stepped and threads	Axial and radial	Internal curved	
⋮					

Figure 3.3b. Chain structure.

***Chain code*:** each value for each digit of the code has a consistent meaning. The value 3 in the third place has the same meaning for all parts.

They are easier to learn but less efficient. Certain digits may be almost meaningless for some parts.

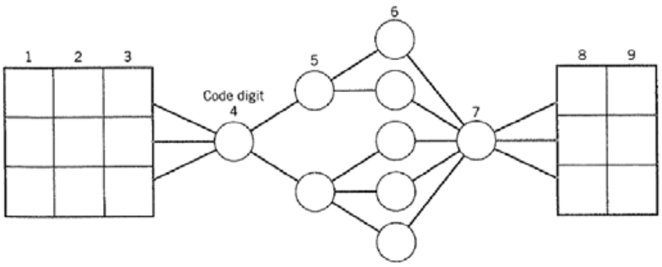


Figure 3.3c. Chain structure.

Since both hierarchical and chain codes have advantages, many commercial codes are ***hybrid***: combination of both:

Some section of the code is a chain code and then several hierarchical digits further detail the specified characteristics. Several such sections may exist. One example of a hybrid code is OPITZ.

The final decision is, ***code representation.*** The digits can be

- *numeric* or even *binary*; for direct use in computer (storage and retrieval efficiency)
- *alphabetic*; humans are more comfortable with a coding like "S" for smooth or "T" for thread (*Gewinde*) than with digits

The proper decision process involves the design engineer, manufacturing engineer, and Computer scientist working together as a team.

A well known coding system is OPITZ. It can have 3 sections:

- it starts with a five-digit "geometric form code"
- followed by a fourdigit "supplementary code."
- This may be followed by a company-specific four-digit "secondary code" intended for describing production operations and sequencing.

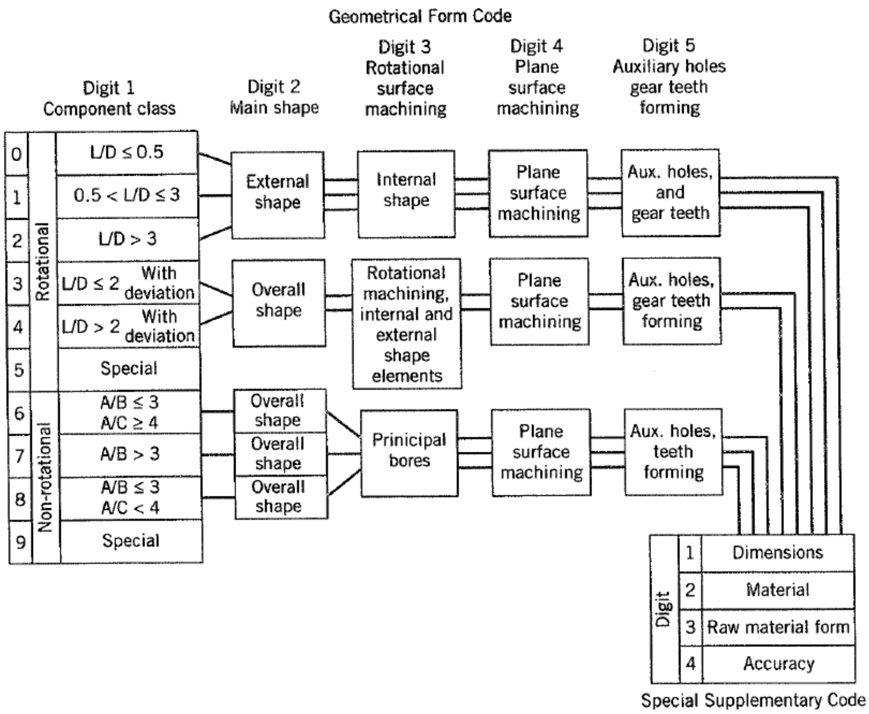


Figure 3.4. Overview of the Opitz code (Askin & Standridge, 1993, p. 167).

Digit 1: shows whether the part is rotational and also the basic dimension ratio (length/diameter if rotational, length/width if nonrotational).

Digit 2: main external shape; partly dependent on digit 1.

Digit 3: main internal shape.

Digit 4: machining requirements for plane surfaces.

Digit 5: auxiliary features like additional holes, etc.

For more details on the meaning of these digits see Figure 6.6 in Askin & Standridge, 1993.

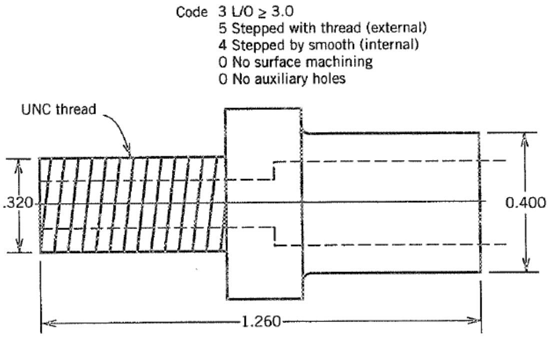


Figure 3.4. Opitz code for sample part (Askin & Standridge, 1993, p. 167).

An example for a coded Part is shown in Figure 3.5.

Correct code: 2 2 4 0 0

Part coding is helpful for design and group formation. But, the time and cost involved in collecting data, determining part families, and rearranging facilities can be seen as the major disadvantage of GT. For designing new facilities and product lines, this is not so problematic: Parts must be identified and designed, and facilities must be constructed anyway. The extra effort to plan under a GT framework is marginal, and the framework facilitates standardization and operation thereafter. Hence, GT is a logical approach to product and facility planning.

3.4 Classification (group formation)

Here, part codes and other information are used to assign parts to families. Part families are assigned to groups along with the machines required to produce the parts. A variety of models for forming part-machine groups are available in the literature, as can be seen from the following figure:

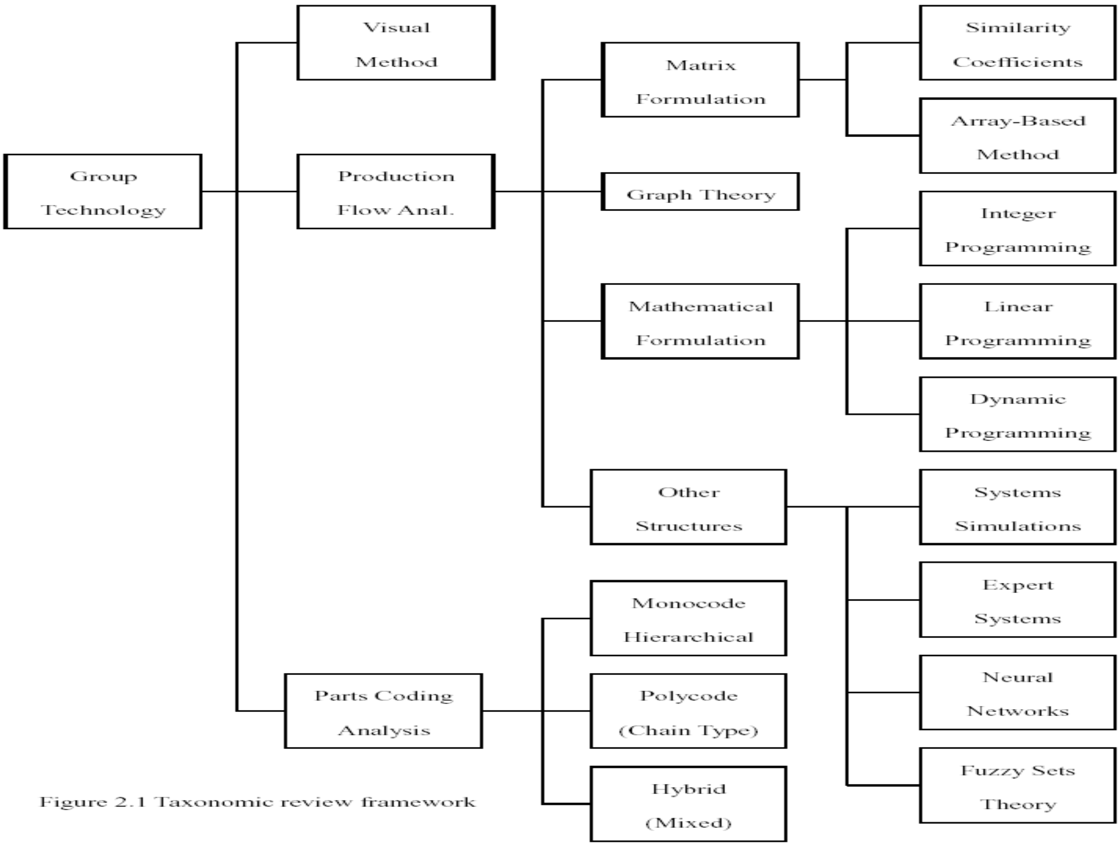


Figure 3.5. Methods of group formation (xxxx).

In addition to simple visual methods based on experience and the use of coding schemes, there is a class of mathematical methods called Production Flow Analysis (PFA).

3.5 Production Flow Analysis (PFA)

To group machines, part routings must be known. Section this presents a method for clustering part operations onto specific machines to provide this routing information.

The basic idea is:

- identify items that are made with the same processes / the same equipment
- These parts are assembled into a part family
- Can be grouped into a cell to minimize material handling requirements.

The clustering methods can be classified into:

- *Part family grouping*: Form part families and then group machines into cells
- *Machine grouping*: Form machine cells based upon similarities in part routing and then allocate parts to cells
- *Machine-part grouping*: Form part families and machine cells simultaneously.

The most typical methods are the *machine-part grouping* ones. Typically one starts with a matrix that shows which part types require which machine types. The aim is to sort the part types and machines such that some kind of block diagonal structure is obtained:

	Part														
Machine	13	2	8	6	11	5	1	10	7	4	3	15	9	12	14
B	■	■	■	■											
D	■		■	■	■										
A						■	■	■	■	■					
H						■	■		■	■	■				
I						■			■		■				
E							■		■	■	■				
C												■	■	■	
G												■	■		■
F												■		■	■

Figure 3.6. Matrix of machine usage (Askin and Standridge).

In case of the example in Figure 3.6, it is easy to build groups:

- Group 1: parts {13, 2, 8, 6, 11 }, machines {B, D}
- Group 2: parts { 5, 1, 10, 7, 4, 3}, machines {A, H, I, E}
- Group 3: parts { 15, 9, 12, 14}, machines {C, G, F}

But the question is how this sorting can be done. Various heuristic and exact methods have been developed. The simplest one is binary ordering, also known asrank order clustering or King's algorithm

3.5.1 Binary Ordering (Rank Order Clustering, King's Algorithm)

This is is done in three steps

- Interpret rows and columns as binary numbers
- Sort rows w.r.t. decreasing binary numbers
- Sort columns w.r.t. decreasing binary numbers

This will be illustrated in a simple **example** (from Günther and Tempelmeier, 1995) with 6 parts and 5 machines:

	part					
machine	*1*	*2*	*3*	*4*	*5*	*6*
A	-	1	-	1	-	-
B	1	-	1	-	1	1
C	-	1	1	1	-	1
D	1	-	-	-	1	1
E	-	-	-	1	1	-

First, the *rows* are interpreted as binary numbers and sorted

	part						
machine	*1*	*2*	*3*	*4*	*5*	*6*	value
A	-	1	-	1	-	-	
B	1	-	1	-	1	1	
C	-	1	1	1	-	1	
D	1	-	-	-	1	1	
E	-	-	-	1	1	-	
2^x	*32*	*16*	*8*	*4*	*2*	*1*	

This gives a new ordering of the machines: B – D – C – A – E. Next, we sort *columns* w.r.t. decreasing binary numbers (note the new order of rows here):

	part						
machine	*1*	*2*	*3*	*4*	*5*	*6*	*2^x*
B	1	-	1	-	1	1	*16*
D	1	-	-	-	1	1	*6*
C	-	1	1	1	-	1	*4*
A	-	1	-	1	-	-	*2*
E	-	-	-	1	1	-	*1*
value							

This gives a new ordering of parts: 6-5-1-3-4-2.

The matrix with rows and columns in the new order is:

	part					
machine	*6*	*5*	*1*	*3*	*4*	*2*
B	1	1	1	1	-	-
D	1	1	1	-	-	-
C	1	-	-	1	1	1
A	-	-	-	-	1	1
E	-	1	-	-	1	-

Now 2 groups can be formed

- Group 1: parts {6, 5, 1}, machines {B, D}
- Group 2: parts { 3, 4, 2}, machines {C, A, E}

Parts 1, 4, and 2 can be produced in one cell. The remaining items 6, 5, and 3 are outside the bold rectangles (indicating the block diagonal structure) and cause problems. There are, in principle 3 possibilities:

1. these parts produced in both cells, i.e. part 6 is mainly produced in cell 1 but for operation on machine C it has to be transported to cell 2
2. machines B, C, and E have to be duplicated, so that all parts can be produced within one cell
3. some parts that do not fit at all could also be given to subcontractors

Binary Ordering is a simple heuristic ⇒ **no guarantee that „optimal“ ordering is obtained.**

Sometimes a better better block-diagonal structure is obtained by repeating the Binary Ordering until there is no change anymore. In the above example this yields the final form of the matrix

	part						
machine	*6*	*5*	*1*	*3*	*4*	*2*	value
B	1	1	1	1	-	-	60
D	1	1	1	-	-	-	56
C	1	-	-	1	1	1	39
E	-	1	-	-	1	-	3
A	-	-	-	-	1	1	18
value	28	26	24	20	7	5	

Hence, repeated Binary Ordering did not help in this example.

3.5.2 Single-Pass Heuristic Considering Capacities (Askin and Standridge)

In the previous section we assumed that all machines have sufficient capacity to produce all products that need to go on this machine, i.e. we ignored capacity. The following algorithm by Askin and Standridge extends the model by introducing capacity considerations:

We make the following assumptions:

- All parts *must* be processed in one cell (machines must be duplicated, if off-diagonal elements occur in the matrix)
- All machines have **capacities** (normalized to be 1)
- There are constraints on number of identical machines in a group
- There are constraints on total number of machines in a group

Example: We will demonstrate the methods in an example (from Günther and Tempelmeier, 1995) with 7 parts and 6 machines. At most 4 machines can be in a group and not mot than one copy of each machine is allowed in each group. The following matrix contains the processing times (incl. set up times) for typical lot size of parts on machines (i.e., the entries in matrix are not just 0/1 for used/not used). All times are normalized as percentage of total machine capacity:

	part								
machine	*1*	*2*	*3*	*4*	*5*	*6*	*7*	sum	min. # machines
A	0.3	-	-	-	0.6	-	-	0.9	1
B	-	0.3	-	0.3	-	-	0.1		
C	0.4	-	-	0.5	-	0.3	-		
D	0.2	-	0.4	-	0.3	-	0.5		
E	-	0.4	-	-	-	0.5	-		
F	-	0.2	0.3	0.4	-	-	0.2		

By summing up all entries in a row we obtain total machine utilization. If this value exceeds one, at least two machines are needed. More generally, this number must be rounded up to the next integer to give the *minimum number of machines* needed. It should be noted, that this minimum number of machines is a *lower bound*. It may be necessary to use more copies of some machines than this minimum number suggests.

Summing up the *minimum number of machines* for all machine types we obtain, that at least 9 machines are needed. Since not more than 4 machines are permitted in a group, we know that at least 9/4 = 2,25 groups are needed. Since only integer numbers of groups make sense, this must be rounded up to obtain the *lower bound on the number of groups*: at least 3 groups.

The **Single-Pass Heuristic by Askin and Standridge** consists of the two steps

1. obtain (nearly) block diagonal structure (e.g. using Binary Ordering)
2. form cells/groups one after another:
    - Assign parts to groups (in sorting order)
    - Also include necessary machines in group
    - Add parts to group until either
        - the capacity of some machine would be exceeded, or
        - the maximum number of machines would be exceeded

Example continued:

For binary sorting treat all entries as 1s. The result is the matrix

	part						
machine	1	5	7	3	4	6	2
D	0.2	0.3	0.5	0.4	-	-	-
C	0.4	-	-	-	0.5	0.3	-
A	0.3	0.6	-	-	-	-	-
F	-	-	0.2	0.3	0.4	-	0.2
B	-	-	0.1	-	0.3	-	0.3
E	-	-	-	-	-	0.5	0.4

Hence, the parts are considered in the following order: D – C – A – F – B – E.

Iteration	part chosen	group	assigned machines	remaining capacity
1	1			
2	5			
3	7			
4	3			
5	4			
6	6			
7	2			

The final solution consists of the three cells:

- Group 1: parts {1, 5}, machines {D, C, A}
- Group 2: parts {7, 3, 4}, machines {D, F, B, C}
- Group 3: parts {6, 2}, machines {C, E, F, B}

We can compare the machines used with the theoretical minimum numbers computed earlier:

	part									
machine	1	2	3	4	5	6	7	sum	min. #	**Single-Pass Heuristic**
A	0.3	-	-	-	0.6	-	-	0.9	**1**	**1**
B	-	0.3	-	0.3	-	-	0.1	0.7	***1***	***2***
C	0.4	-	-	0.5	-	0.3	-	1.2	***2***	***3***
D	0.2	-	0.4	-	0.3	-	0.5	1.4	**2**	**2**
E	-	0.4	-	-	-	0.5	-	0.9	**1**	**1**
F	-	0.2	0.3	0.4	-	-	0.2	1.1	**2**	**2**

Apparently, we need one more copy of machine B (2 instead of 1) and one more copy of machine C (3 instead of 2).

We should note, that the Single-pass heuristic of Askin und Standridge is a simple heuristic. Hence, it gives not necessarily an optimal solution (min possible number of machines).

3.5.3 LP-Model for the model by Askin and Standridge

The assignment of machines and parts to groups can easily be formulated as a binary integer program BIP. Let us consider exactly the same problem as in the previous subsection and let the objective be the (weighted) number of machines used.

We will use the following notation:

$i \in I$	cells, groups
$j \in J$	parts
$k \in K$	machine types
a_{jk}	capacity of machine type k needed for part j
M	max number of Maschinen per group

Furthermore, per group only one copy of each machine type is permitted. The decision variables are:

x_{ij} =1, if part j is assigend to group i (and = 0, otherwise)

y_{ik} =1, if machine type k is assigend to group i (and = 0, otherwise)

The objective is the toral number of machines used:

$$\sum_{i \in I} \sum_{k \in K} y_{ik} \to \text{min!}$$

subject to the constraints:

$$\sum_{i \in I} x_{ij} = 1 \qquad j \in J \qquad \text{(each part } j \text{ in exactly one group)}$$

$$\sum_{j \in J} a_{jk} \cdot x_{ij} \le y_{ik} \qquad i \in I, k \in K \qquad \text{(capacity of machine } k \text{ in group } i\text{)}$$

$$\sum_{k \in K} y_{ik} \le M \qquad i \in I \qquad \text{(not more than } M \text{ machines in group } i\text{)}$$

$$x_{ij} \in \{0,1\} \qquad i \in I, j \in J \qquad \text{(binary variables)}$$

$$y_{ik} \in \{0,1\} \qquad i \in I, k \in K \qquad \text{(binary variables)}$$

The opti al solution can be computed using some standard LP solvers. In the simple **example above**, this can be dobe using the EXCEL solver – see XLS file on the course homepage. The optimal solution is:

group	parts	machines	Remaining capacity
1	2, 4, 6	B, C, E, F	B (0.4), C (0.2), E (0.1), F (0.4)
2	1, 5	A, C, D	A (0.1), C (0.6), D (0.5)
3	3, 7	B, D, F	B (0.9), D (0.1), F (0.5)

Hence, the simple single-pass heuristic did not find the optimal solution:

	part										
machine	1	2	3	4	5	6	7	sum	min. #	Single-Pass Heur.	opt
A	0.3	-	-	-	0.6	-	-	0.9	**1**	**1**	**1**
B	-	0.3	-	0.3	-	-	0.1	0.7	***1***	***2***	***2***
C	0.4	-	-	0.5	-	0.3	-	1.2	***2***	***3***	***2***
D	0.2	-	0.4	-	0.3	-	0.5	1.4	**2**	**2**	**2**
E	-	0.4	-	-	-	0.5	-	0.9	**1**	**1**	**1**
F	-	0.2	0.3	0.4	-	-	0.2	1.1	**2**	**2**	**2**
Sum									**9**	**11**	**10**

3.5.4. Clustering using Similarity Coefficients

Another method of clustering is based on similarity coefficients. The idea is to identify machines which are used more or less for the same parts and to put these in a group. We define:

n_i ... Number of parts visiting machine *i*

n_{ij} ... Number of parts visiting machines *i* and *j*

Then the *similarity coefficient* between machines *i* and *j* is defined as:

$$s_{ij} = \max\left\{\frac{n_{ij}}{n_i}, \frac{n_{ij}}{n_j}\right\} = \frac{n_{ij}}{\min\{n_i, n_j\}}$$

Example: (from Askin and Standridge) 6 machines and 8 parts. All these calculations can easily be performed using EXCEL; → see *the course homepage*.

	parts								n_i
machine	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	
A	1	1	1						3
B	1	1	1						3
C			1	1	1	1			4
D				1	1	1	1		4
E							1	1	2
F							1	1	2

The values n_{ij} can be computed:

n_{ij}	parts							
machine	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
A								
B								
C								
D								
E								
F								

This gives the similarity coefficients:

s_{ij}	parts					
machine	*A*	*B*	*C*	*D*	*E*	*F*
A	1	1	0,33	0	0	0
B	1	1	0,33	0	0	0
C	0,33	0,33	1	0,75	0	0
D	0	0	0,75	1	0,5	0,5
E	0	0	0	0,5	1	1
F	0	0	0	0,5	1	1

These have a similar function as the savings values known from transportation logistics. The following hierarchical clustering heuristic is very similar to the savings algorithm known from VRP.

Before proceeding, one can eliminate all entries with $s_{ij} \leq T$, where T is some parameter between 0 and 1. By omitting the "weak" links the structure becomes clearer. Here, we choose $T = 1$ and we do not eliminate any links at the moment.

Hierarchical clustering heuristic:

1. Form N initial clusters (one for each machine). Compute similarity coefficients s_{ij} for all machine pairs.
2. Merge clusters: Let *i* and *j* range over all clusters. Choose the pair if clusters (i^*, j^*) that has the highest similarity coefficient s_{ij}. Merge clusters i^* and j^* if possible.

   If more than one cluster remains, go to 3. otherwise stop.
3. Update coefficients: Remove rows and columns i^*, j^* from the similarity coefficient matrix. Replace them with a new row *k* and a new column *k*. For all remaining clusters *r*, the updated similarity coefficients of this new cluster *k* are computed as:

$$s_{rk} = \max \{s_{ri^*}, s_{rj^*}\}$$

In step 3, when clusters i^* and j^* are joined to become the new cluster *k* the new similarity coefficient to some other cluster *k* is computed as the maximum of the corresponding similarity coefficient of clusters i^* and j^*. This is one possible setting.

- *Other updating rules are possible, such as e.g. the average of the corresponding similarity coefficients.*

In the first iteration, groups i^* = A and j^* = B are joined to become new group k = AB. The updated similarity coefficients are

s_{ij}	parts				
machine	AB	C	D	E	F
AB	1	0,33	0	0	0
C	0,33	1	0,75	0	0
D	0	0,75	1	0,5	0,5
E	0	0	0,5	1	1
F	0	0	0,5	1	1

In the next iteration, clusters i^* = E and j^* = F are joined to become new group k = EF. The updated similarity coefficients are:

s_{ij}		parts		
machine	AB	C	D	EF
AB	1	0,33	0	0
C	0,33	1	0,75	0
D	0	0,75	1	0,5
EF	0	0	0,5	1

Next, clusters *i** = C and *j** = D are joined to become new group *k* = CD. The updated similarity coefficients are:

s_{ij}		parts	
machine	AB	CD	EF
AB	1	0,33	0
CD	0,33	1	0,5
EF	0	0,5	1

If groups should be joined further (because the constraints permit this), clusters *i** = CD and *j** = EF are joined to become new group *k* = CDEF.

The following figure shows at which thresholds (corresponding to *T* mentioned above) which groups can be formed.

For *T* = 1 only the groups AB and EF can be formed, while machines C an d form their own single machine roups.

For *T* = below 0.33 all machines are joined in one group.

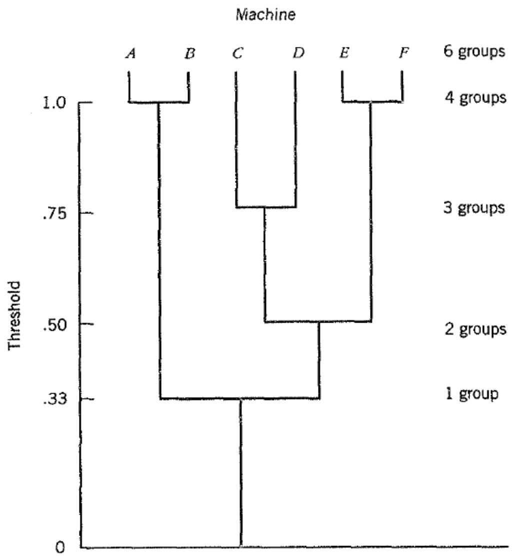


Figure 3.7. Dendogram for a hierarchical clustering (Askin and Standridge).

3.5.5. Group Formation using Graph Partitioning

When machines have common parts, i.e., $n_{ij} > 0$ in the notation of Section 3.5.4, then ideally they should be in the same group. Otherwise, duplication of machines or transportation between groups is necessary. This could be graphically represented as a graph with the nodes being the machines, where edges between machines mean common parts:

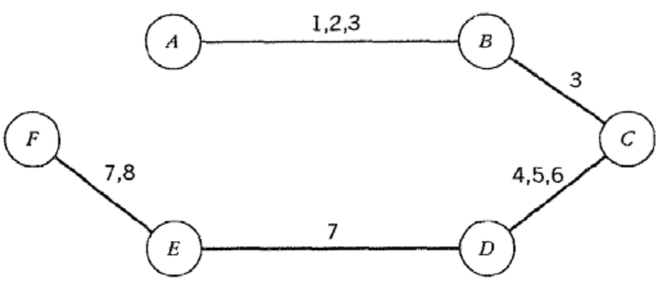


Figure 3.8. Graph representation of the example (Askin and Standridge); numbers at the edges are the common parts.

Then group formation can be seen as a special case of graph partitioning. This can be formulated as follows:

> *Given a graph with nodes and edges, find a partitioning of the node set into a (given) number of disjoint subsets of approximately equal size, such that the total cost of edges that connect nodes of different subsets is minimized.*

Graph partitioning is an np-hard combinatorial optimization problem. Various exact and heuristic methods have been developed over the past decades. We describe a simple and well known heuristic by Kernighan and Lin (1970) for clustering in two subsets.

3.5.5.1 Graph partitioning heuristic by Kernighan and Lin (KL)

Input: A weighted graph G = (V, E) with

- Vertex set V. (|V| = 2n)
- Edge Set E. (|E| = e)
- Cost c_{AB} for each edge (A, B) in E.

Output: 2 subsets X & Y such that

- $V = X \cup Y$ and $X \cap Y = \{ \}$ (i.e. partition)
- Each subset (group) has n vertices
- Total cost of edges "crossing" the partition is minimized.

Complete enumeration (brute force) is not possible (np-hard):

- Try <u>all</u> possible bisections. Choose the best one.
- If there are 2n vertices $\Rightarrow$ number of possibilities = (2n)! / (n!)2 = $n^{O(n)}$
- For 4 vertices (A,B,C,D), 3 possibilities
    1. X = {A, B} & Y = {C, D}
    2. X = {A, C} & Y = {B, D}
    3. X = {A, D} & Y = {B, C}
- For 100 vertices $\Rightarrow$ 5×10^{28} possibilities

KL-Algorithm:

The KL-Algorithm is an improvement algorithm, that starts with any initial partition X and Y (e.g. obtained using any constructive algorithm)

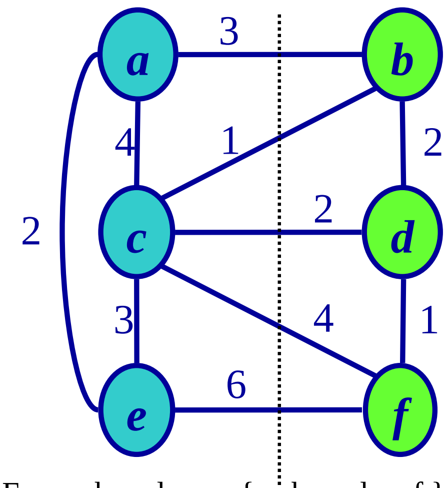
- A <u>*pass*</u> means exchanging each vertex $A \in X$ with each vertex $B \in Y$ exactly once:

    1. For *i* := 1 to *n* do
       From all unlocked (unexchanged) vertices,
       choose a pair (A, B) such that the gain(A, B) is largest.
       Exchange A and B. Lock A and B.
       Let g_i = gain(A, B). (can also be negative)
    2. Find the *k* s.t. $G = g_1 + ... + g_k$ is maximized.
    3. Switch the first k pairs.

- Repeat the pass until there is no more improvement ($G = 0$).

The complexity of this algorithm (in a naïve implementation) is as follows. For each pass, $O(n^2)$ time is needed to find the best pair to exchange; n pairs are exchanged $\Rightarrow$ the total time is O(*n*3) per pass. But there are better implementation that need $O(n^2 \lg n)$ time per pass. And the number of passes is usually small.

Example for KL-Algorithm:



Initial weighted graph G with 6 vertices (nodes),

V(G) = { a, b, c, d, e, f }.

Start with any partition of V(G) into X and Y, e.g.,

X = { a, c, e }
Y = { b, d, f }

The cut value is the sum of all edge costs between the 2 sets:

cut-size = 3 + 1 + 2 + 4 + 6 = 16

Try to improve this partitioning (i.e. reduce cut-size) using KL.

For each node x ∈ { a, b, c, d, e, f }.compute the gain values of moving node x to the others set:

$$G_x = E_x - I_x$$

where

E_x = cost of edges connecting node x with the other group (extra)
I_x = cost of edges connecting node x within its own group (intra)

This gives:

$G_a = E_a - I_a = 3 - 4 - 2 = -3$
$G_c = E_c - I_c = 1 + 2 + 4 - 4 - 3 = 0$
$G_e = E_e - I_e = 6 - 2 - 3 = +1$

$G_b = E_b - I_b = 3 + 1 - 2 = +2$
$G_d = E_d - I_d = 2 - 2 - 1 = -1$
$G_f = E_f - I_f = 4 + 6 - 1 = +9$

Cost saving when exchanging a and b is essentially $G_a + G_b$

However, the cost saving 3 of the direct edge (a, b) was counted twice. But this edge still connects the two different groups ⇒ must be added twice. Hence, the real "gain" (cost saving) of this exchange is

$$g_{ab} = G_a + G_b - 2c_{ab}$$

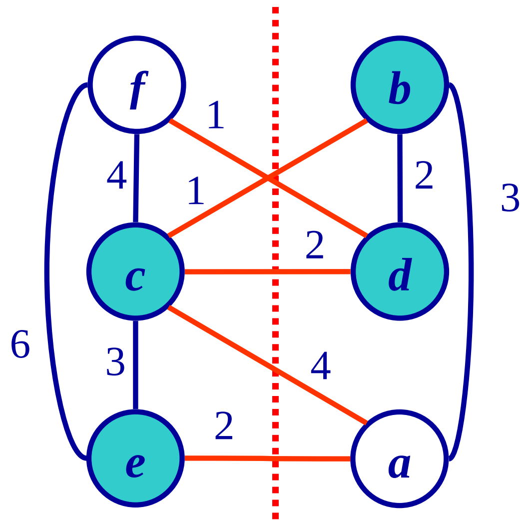
Must compute this for all possible combinations (pairs):

$g_{ab} = G_a + G_b - 2w_{ab} = -3 + 2 - 2 \cdot 3 = -7$
$g_{ad} = G_a + G_d - 2w_{ad} = -3 - 1 - 2 \cdot 0 = -4$
$\mathbf{g_{af} = G_a + G_f - 2w_{af} = -3 + 9 - 2 \cdot 0 = +6}$

$g_{cb} = G_c + G_b - 2w_{cb} = 0 + 2 - 2 \cdot 1 = 0$
$g_{cd} = G_c + G_d - 2w_{cd} = 0 - 1 - 2 \cdot 2 = -5$
$g_{cf} = G_c + G_f - 2w_{cf} = 0 + 9 - 2 \cdot 4 = +1$

$g_{eb} = G_e + G_b - 2w_{eb} = +1 + 2 - 2 \cdot 0 = +1$
$g_{ed} = G_e + G_d - 2w_{ed} = +1 - 1 - 2 \cdot 0 = 0$
$g_{ef} = G_e + G_f - 2w_{ef} = +1 + 9 - 2 \cdot 6 = -2$

The maximum gain is obtained by exchanging nodes a and f ⇒ **new cut-size = 16 – 6 = 10**.



Perform this exchange

Verify: **new cut-size = 1 + 1 + 2 + 4 + 2 = 10**

Lock all exchanged nodes (*a* and *f*)

New sets of unlocked nodes:

$\boldsymbol{X' = \{ c, e \}}$
$\boldsymbol{Y' = \{ b, d \}}$

Update the G-values of unlocked nodes

$\boldsymbol{G'_c = G_c + 2c_{ca} - 2c_{cf} = 0 + 2(4 - 4) = 0}$
$\boldsymbol{G'_e = G_e + 2c_{ea} - 2c_{ef} = 1 + 2(2 - 6) = -7}$

$\boldsymbol{G'_b = G_b + 2c_{bf} - 2c_{ba} = 2 + 2(0 - 3) = -4}$
$\boldsymbol{G'_d = G_d + 2c_{df} - 2c_{da} = -1 + 2(1 - 0) = 1}$

Compute the gains:

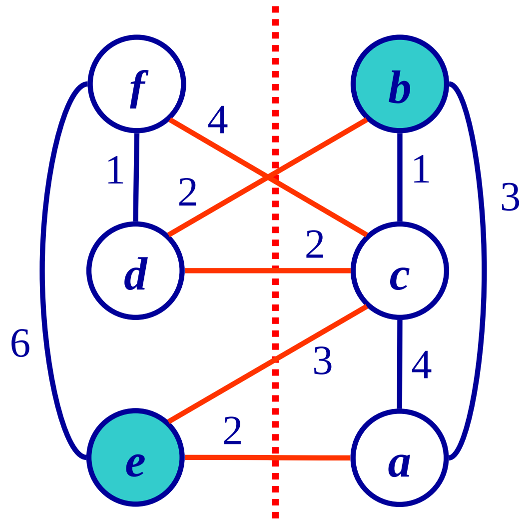
$g_{cb} = G_c + G_b - 2w_{cb} =$

$g_{cd} = G_c + G_d - 2w_{cd} =$

$g_{eb} = G_e + G_b - 2w_{eb} =$

$g_{ed} = G_e + G_d - 2w_{ed} =$

Pair with maximum gain (can also be neative) is (c, d).



Perform this exchange between *c* and *d*.

new cut-size = = 10 – (-3) = 13

Lock all exchanged nodes (*c* and *d*)

New sets of unlocked nodes:

$\boldsymbol{X' = \{ e \}}$
$\boldsymbol{Y' = \{ b \}}$

Update the G-values of unlocked nodes

$\boldsymbol{G'_e = G_e + 2c_{ed} - 2c_{ec} =}$
$\boldsymbol{G'_b = G_b + 2c_{bd} - 2c_{bc} =}$

Compute the gains:

$\boldsymbol{g_{eb} = G_e + G_b - 2c_{eb} = -1 - 2 - 2 \cdot 0 = -3}$

Summary of the Gains…

- $g_1 = +6$
- $g_1 + g_2 = +6 - 3 = +3$
- $g_1 + g_2 + g_3 = +6 - 3 - 3 = 0$

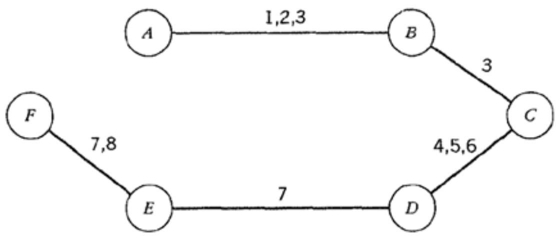
Maximum gain is $g_1 = +6 \Rightarrow$ Exchange only nodes *a* and *f*. End of 1 pass.

This pass must be repeated until no changes are observed any more.

3.5.5.1 Application of graph partitioning (KL) to group formation

We do this in the above example:

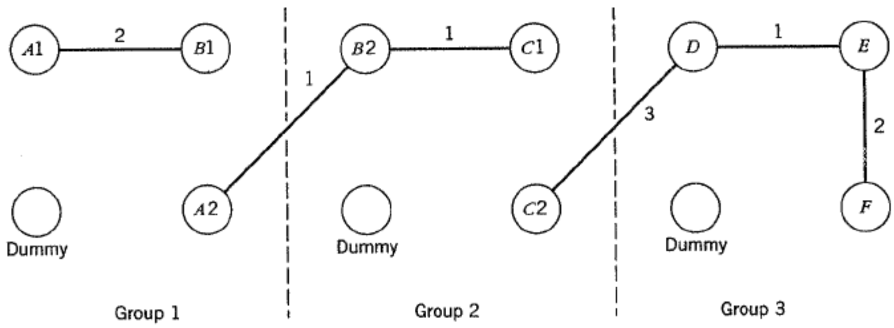
	parts							
machine	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
A	1	1	1					
B	1	1	1					
C			1	1	1	1		
D				1	1	1	1	
E							1	1
F							1	1



Assume that from capacity considerations (min number of machines) it is clear that at least 2 copies of machines A, B, and C are necessary. Hence we duplicate machines A, B, and C:

	parts							
machine	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
A1	1	1						
B1	1	1						
A2			1					
B2			1					
C1			1					
C2				1	1	1		
D				1	1	1	1	
E							1	1
F							1	1

This gives the graph, where the costs $c_{ij} = n_{ij}$ from Section 3.5.4 (i.e. the number of common parts).



Let us assume that we need 3 clusters with at least 2 and at most 4 machines each. We start with an initial clustering with 3 machines each. For this, we simply use the rows of the above matrix (apparently this is not the best clustering, but we want to demonstrate the improvement step).

Note that we have also added dummy machines /with zero cost connections) to represent empty spaces that could be occupied by real machines (note that up to 4 machines are permitted).

We start with optimizing the partition Group 1 = {A1, A2, B1, Dummy1} and Group 2 = {B2, C1, C2, Dummy2} while we keep Group 3 = {D, E, F, Dummy3} unchanged for the moment.

Next, we apply the KL heuristic to Group 1 and Group 2:

For all nodes in these groups, we compute E_x, I_x, and G_x.

Group	Node i	E_i	I_i	G_i
1	A1	*0*	*2*	*-2*
	B1	*0*	*2*	*-2*
	A2	*1*	*0*	*1*
	Dummy1	*0*	*0*	*0*
2	B2	*1*	*1*	*0*
	C1	*0*	*1*	*-1*
	C2	*0*	*0*	*0*
	Dummy2	*0*	*0*	*0*

Next we compute the G_{ij}

Node i	Node j	G_{ij}	G_{ij}'	G_{ij}''
A1	B2	*-2*	***-4***	
	C1	*-3*	*-3*	
	C2	*-2*	*-2*	
	Dummy2	*-2*		
B1	B2	*-2*	***-4***	
	C1	*-3*	*-3*	
	C2	*-2*	*-2*	
	Dummy2	*-2*		
A2	B2	*-1*		
	C1	*0*		
	C2	***1***		
	Dummy2	***1* ←**		
Dummy1	B2	*0*	*-2*	
	C1	*-1*	*-1*	
	C2	*0*	*0* ←	
	Dummy2	*0*		

We could choose the pairs (A2, C2), (A2, Dummy2), or (Dummy1, C1). We arbitrarily choose (A2, Dummy2) and fix these two machines (nodes). Then we update G_i:

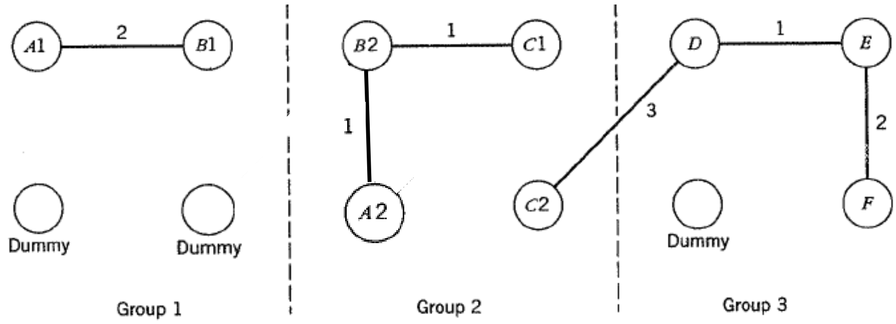
$G_i' = G_i + 2c_{iA2} - 2c_{iDummy2}$ in Group 1 and $G_j^{new} = G_j + 2c_{iDummy2} - 2c_{jA2}$ in Group 2.

Group	Node i	G_i'	G_i''
1	A1	*-2+0-0 = -2*	
	B1	*-2+0-0 = -2*	
	Dummy2		
	Dummy1	*0*	
2	B2	*0+0-2 =* ***-2***	
	C1	*-1+0-0 = -1*	
	C2	*0+0-0 = 0*	
	A2		

Then we update G_{ij}. We can do this in the above table in a new column. No improvements possible, but the switch (Dummy1, C2) is the best one (no change in cost). This change is performed and the machines Dummy1, C2 are fixed. New group 1 = {A1, B1, ~~Dummy2~~, ~~C2~~} and group 2 = {B2, C1, ~~Dummy1~~, ~~A2~~} where fixed values are cancelled. Next step with G_i'' and G_{ij}''.

We see that only the first step brought an improvement and get the new partition:
group 1 = {A1, B1, Dummy2, Dummy1}, and group 2 = {B2, C1, C2, A2}.

We could repeat this pass of the KL heuristic, but since the cut-value of this partition is zero, we know that this is already the optimal partition of these 8 machines (including 2 dummies).



In a similar way, the KL heuristic can be applied to groups 2 and 3 to exchange C2 and Dummy3. Then the optimal partition with cut-value zero is obtained in this example.

In general, this procedure is a heuristic and it is *not* guaranteed that an optimal partition is found.

3.5.6 Group analysis without binary ordering: “key" machine

In the previous section we have briefly discussed graph theoretic methods based on KL. This was an improvement heuristic (to improve a given partition), or it could also be used as a constructive method. Using the idea of recursive bisection, first two groups (af approvimately equal size) are formed. Each of these is then split into two subgroups and so on. After *k* such steps one has 2^k groups.

Askin and Standridge (1993, § 6.4.1) also present another simple algorithm, that does not need binary ordering and where the opposite approach is used, i.e., where “atomic” subgroups are formed that can subsequently be combined to larger groups:

1. The machine with the fewest part types is called the **"key" machine**. A subgroup is formed from all the parts that visit this key machine along with all machines required by these part types.
2. Check if (except for the key machine) the machines in the subgroup fall into two or more disjoint sets with respect to the parts they service. If **disjoint subsets** of the subgroup exist, the subgroup is again subdivided into multiple subgroups.
   If any machine is included in the subgroup due to just one part type, then this machine is termed **exceptional** and removed.
   Steps 1 and 2 are repeated until all parts and machines are assigned to subgroups.
3. The final step involves combining subgroups into groups of the desired size. Subgroups with the greatest number of common machine types are combined.

Example:

(no duplication of machines)

	parts							
machine	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
A	1	1	1					
B	1	1	1					
C			1	1	1	1		
D				1	1	1	1	
E							1	1
F							1	1

The data has been ordered using binary ordering so that similarities are more easily seen. However, this is not necessary in this method.

Solution:

Iteration 1:

Step 1. Identify a key machine.
Machines E and F receive the fewest components ⇒ Arbitrarily choose **E** as key machine.
Parts 7 and 8, visit E. These parts require machines D, E, and F, thus forming a subgroup.

Step 2. Check for subgroup division:
Ignoring machine E, all parts visit machine F ⇒ subgroup cannot be further subdivided.
Machine D is used only for part 7 ⇒ D is exceptional for this subgroup and is removed.

Iteration 2:

Step1 . Identify new key machine. Six parts remain.
All machines receive at least three parts ⇒ Arbitrarily choose A.
Parts 1, 2, and 3 form the subgroup along with machines A, B, and C.

Step2 . Subgroup division:
Removing machine A does not create disjoint subgroups for parts 1,2, and 3.
Machine C is used for part 3 only ⇒ exceptional ⇒ remove.

Iteration 3:

Step1 . Identify a new key machine. Only parts 4, 5, and 6 remain.
C is the key machine. The subgroup becomes parts 4, 5, and 6 along with machines C and D.

Step2 . No further subdivision is possible. No exceptional machine.

Result of Steps 1 and 2:

parts

machine	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
A	1	1	1					
B	1	1	1					
C			1	1	1	1		
D				1	1	1	1	
E							1	1
F							1	1

Step3 . Aggregation: The decision maker can now attempt to recombine the three subgroups into a set of workable groups of desired size.

3.6 Metaheuristics

We have briefly discussed some of the classical constructive heuristics and improvement heuristics from the literature.

Since we are dealing with a tactical problem (that is not solved every day) where long computation times are acceptable, it makes sense to invest more time. This can be done by applying metaheuristics, exact methods (up to a certain problem size) and combined methods (matheuristics).

There is a large literature on applying metaheuristics and grouping or clustering problems (mainly genetic algorithms or tabu search). Nevertheless, various possibilities exist to come up with new metaheuristic approaches.

Examples:

- Since the similarity coefficients are rather similar to the savings values of transportation logistics (VRP), the idea of a savings based ant system for VRP could be transferred to grouping problems.
- The KL algorithm could be considered a local search (maybe in a simplified faster version), and could be combined with some larger shaking steps to a VNS. Other fast local searches (exchange and move) could be considered.
- A matheuristic could easily be constructed by applying e.g. the principle of destroy and reconstruct: for a large problem, a subset of groups could be "destroyed" and all their machines and parts could be freed. Then this smaller problem (considering only these parts and machines) could be solved using some exact algorithm (e.g. applying CPLEX to a MIP formulation).

When designing metaheuristics or matheuristics for grouping problems, there are also 2 possibilities:

- Work directly on the model formulation (e.g. the above examples)
- Use a more aggregated representation and then apply some constructive algorithm to compute the solution out of it. For example, the metaheuristic could just work on the ordering of parts and machines (to give a better block diagonal structure than binary ordering) and then the single pass heuristic by Askin and Standridge could be used to construct a solution.

It should also be noted that there ate various classes of grouping problems that differ w.r.t. objective and constraints. This concerns e.g. duplication of machines and/or inter-group transport, etc.

References

Askin, R.G., Standridge, C.R.: Modeling & Analysis Of Manufacturing Systems, John Wiley & Sons, 1993.

B. Kernighan and S. Lin (1970): An Efficient Heuristic Procedure for Partitioning of Electrical Circuits, Bell System Technical Journal, 291-307.

Internet sources on Opiz, KK3 und some methods, e.g.:

- http://www.ielm.ust.hk/dfaculty/ajay/courses/ieem513/GT/GT.html

Examples for metaheuristics:

- T.L. James, E.C. Brown, K.B. Keeling (2007): A hybrid grouping genetic algorithm for the cell formation problem, Computers and Operations Research, Volume 34 (7, July) 2059-2079 .
- Mahdavi, M.M. Paydar, M. Solimanpur, A. Heidarzade (2009): Genetic algorithm approach for solving a cell formation problem in cellular manufacturing, Expert Systems with Applications: An International Journal, Volume 36 (3, April) 6598-6604.
- T. Tunnukij, C. Hicks (2009): An Enhanced Grouping Genetic Algorithm for solving the cell formation problem, International Journal of Production Research, Volume 47 (7, Jan.) 1989 – 2007.
- D. Cao and M. Chen (2004): Using penalty function and Tabu search to solve cell formation problems with fixed cell cost, Computers & Operations Research, Volume 31 (1, Jan.) 21-37.
- J. Schaller (2005): Tabu search procedures for the cell formation problem with intra-cell transfer costs as a function of cell size, Computers and Industrial Engineering, Volume 49 (3, Nov.), 449 – 462.