

Modul

Layout and Design

Prof. Dr. Richard F. Hartl
Dr. Margaretha Gansterer

SS 2010



© Produktion und Logistik

Contents

0. Introduction	5
1. Methodological Basics	9
Complexity	9
Costs and distances	10
Basics on Graph Theory	11
2. Job shop production	13
2.1. The Linear Assignment Problem	14
2.1.1. Formulation as transportation problem	15
2.1.2. Assignment Method (Kuhn's Algorithm)	17
2.2. The Quadratic Assignment Problem (QAP)	22
2.2.1. QAP: Mathematical formulation.....	24
2.2.2. Starting heuristics	26
2.2.3. Improvement methods	27
2.2.4. „Umlaufmethode“	28
2.2.5. Different space requirements.....	30
2.2.5.1. CRAFT Algorithm	31
3. Group Technology / Cellular Manufacturing	38
3.1 Introduction	38
3.2 How to form groups	41
3.3 Coding schemes	42
3.4 Classification (group formation)	45
3.5 Production Flow Analysis (PFA)	45
3.5.1 Binary Ordering (Rank Order Clustering, King's Algorithm)	47
3.5.2 Single-Pass Heuristic Considering Capacities (Askin and Standridge).....	50
3.5.3 LP-Model for the model by Askin and Standridge	52
3.5.4. Clustering using Similarity Coefficients	54
3.5.5. Group Formation using Graph Partitioning	57
3.5.6 Group analysis without binary ordering: "key" machine	65
3.6 Metaheuristics	67
References	67
4. Exact Methods for Assembly Line Balancing	69

Jackson Algorithm (Dynamic Programming, Decision Tree).....	69
The B&B algorithm by Johnson von (FABLE)	75

0. Introduction¹²

Layout decisions are one of the key facts determining the long-run efficiency of operations. Layouts have numerous strategic implications because they establish an organization's competitive priorities in regard to capacity, processes, flexibility, and cost. They are associated with the tactical decision horizon and are dedicated to the concretion of strategic decisions like, e.g., facility location. Configured production systems are input for the operational level, where the goal is to run the given system as efficiently as possible.

An efficient layout facilitates and reduces costs of material flow, people, and information between areas. To achieve these objectives, a variety of configuration designs have been developed. The most relevant ones, in the context of this course, are:

1. **Fixed-position layout**: addresses the layout requirements of large, bulky projects
2. **Job shop production** (Process-oriented layout): deals with low-volume, high-variety production
3. **Cellular manufacturing** systems (work cell layout): arranges machinery and equipment to focus on production of a single product or group of related products
4. **Flow shop production** (Product-oriented layout): seeks the best personnel and machine utilization in repetitive or continuous production.

As a matter of fact layouts 1 and 2 are often described as centralized, and layouts 3 and 4 as decentralized manufacturing systems.

Example²: To illustrate the differences in fixed-position layout, job shop production, cellular manufacturing systems, and flow shop production consider a situation in which four parts (A, B, C, D) are to be produced and assembled into a single product. The processing sequence for part A is saw, turn, mill, and drill; for part B it is saw, mill, drill, and paint; for part C the processing sequence is grind, mill, drill, and paint; and for part D the sequence is weld, grind, turn, and drill. All parts go to a central assembly department. The following table contains the proportional capacity requirement of each part on each machine relative to the capacity availability of the machine in one time period.

Part	Equipment requirements						
	Weld	Grind	Saw	Turn	Mill	Drill	Paint
A	-	-	0.5	0.5	0.3	0.2	-
B	-	-	0.4	-	0.5	0.3	0.2
C	-	0.4	-	-	0.3	0.5	0.3
D	0.3	0.5	-	0.3	-	0.2	-

¹ Heizer, J., Render, B., Operations Management, Prentice Hall, 2006, Chapter 9

² Francis, R., McGinnis, L., White, J., Facility Layout and Location: An Analytical Approach, Prentice Hall, 1992

Based on the given capacity requirements we know that the minimum equipment needed is: 1 weld, 1 grind, 1 saw, 1 turning machine, 2 mills (0.3+0.5+0.3 > 1), 2 drills, and 1 painting machine.

According to the layout concepts listed above the following configurations for the example problem could be realized (this is not a complete list of all possible configurations but an illustrative selection of possible realizations).

1. In case of a fixed-position layout it may be sufficient to have the minimum machine equipment (see above). But depending on how production is scheduled it could also be necessary to install more machines in order to come up with the needed production output.

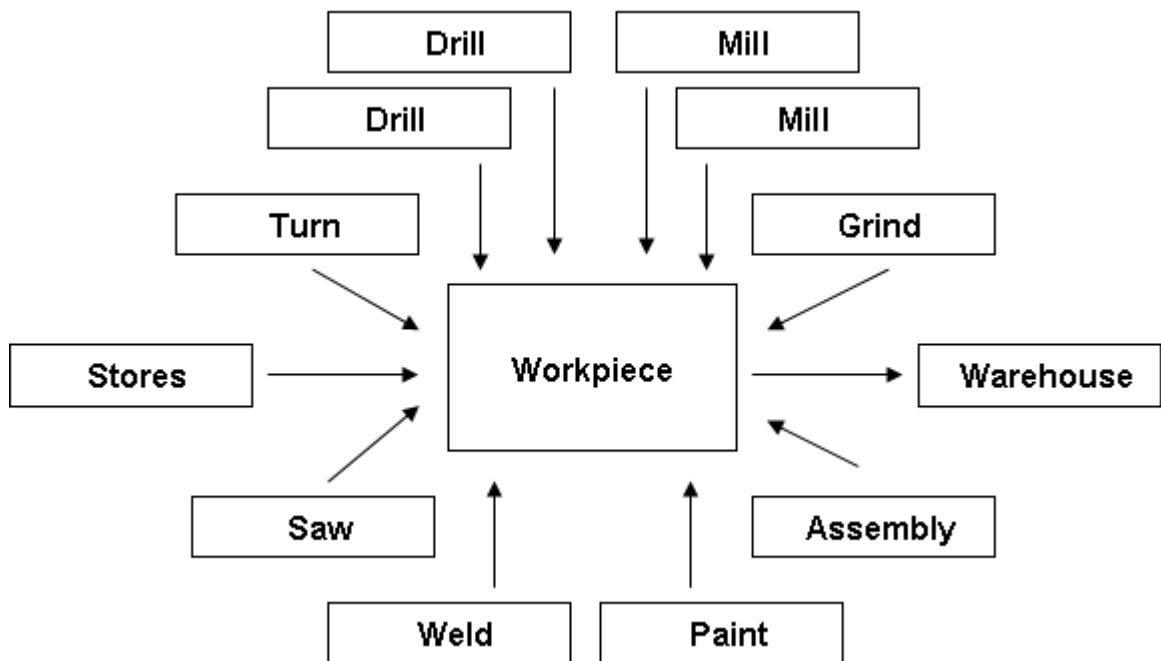


Figure 0-1: Fixed-position layout

2. By applying a job shop production system we are able to reach the minimum machine equipment. Clearly, depending on production scheduling it may become necessary to install more machines than the minimum equipment.

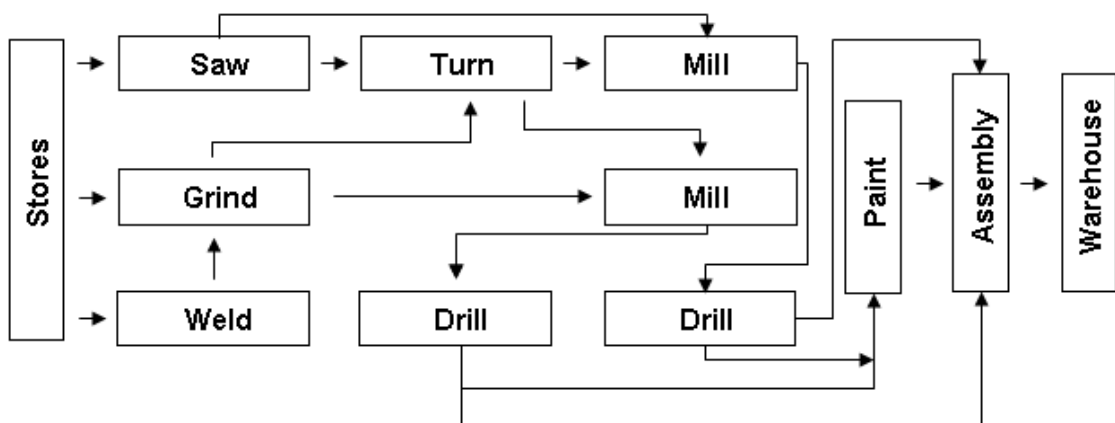


Figure 0-2: Job shop production

- Figure 0-3 illustrates a cellular manufacturing system for the example problem. For the chosen configuration (2 work cells) it is not possible to realize the minimum machine equipment. We need an additional turning machine and an additional painter.

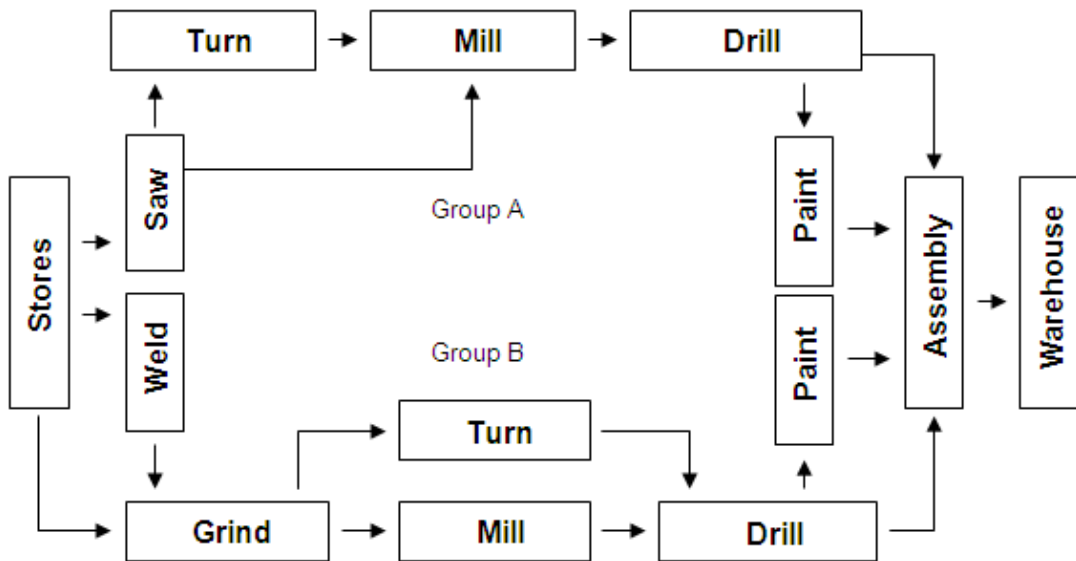


Figure 0-3: Cellular manufacturing system

- Figure 0-4 shows a flow shop production system for the example problem. In this case we need 5 machines additional to the minimum equipment (1 grind, 1 saw, 1 turning machine, 1 mill, and 1 paint):

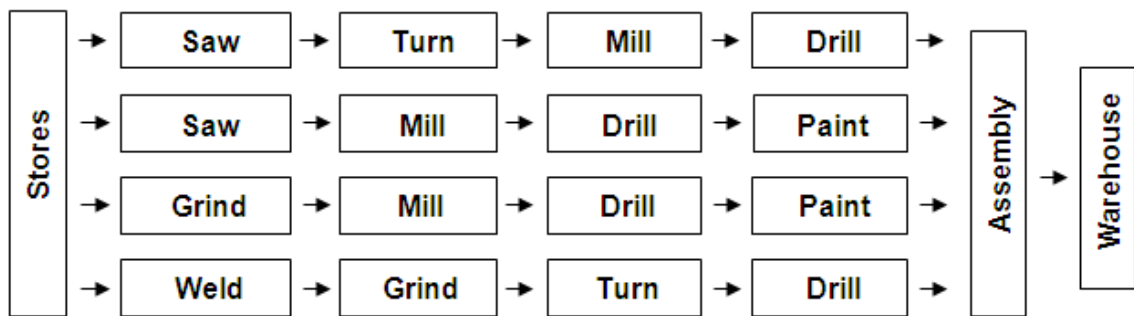


Figure 0-4: Flow shop production

The decision to use a fixed-position layout is generally dictated by a particular characteristic of the workpiece. It layout is used when the product is too large or cumbersome to be moved through the various processing steps. Consequently, the processes are brought to the product rather than taking the product to the processes (e.g. aircraft industry,...).

This concept is realized by locating workstations or production centres around the product in the appropriate processing sequence. Considerable logistics are involved in ensuring that the right processes are brought to the product at the right times and are located in the right places.

Advantages:

- Material movement is reduced.
- Promotes job enlargement by allowing individuals or teams to perform the “whole job”.
- Highly flexible; can accommodate changes in product design, product mix, and production volume.
- Independence of production centres allows scheduling to achieve minimum total production time.

Limitations:

- Increased movement of personnel and equipment.
- Equipment duplication may occur.
- Higher skill requirements for personnel.
- General supervision required.
- Cumbersome and costly positioning of material and machinery.
- Low equipment utilization.

However, the decision to use either a job shop, work cell, or flow shop layout generally depends on the volumes of production and variety of products being manufactured. Figure 0-5 illustrates a volume-variety chart³.

³ Francis, R., McGinnis, L., White, J., Facility Layout and Location: An Analytical Approach, Prentice Hall, 1992

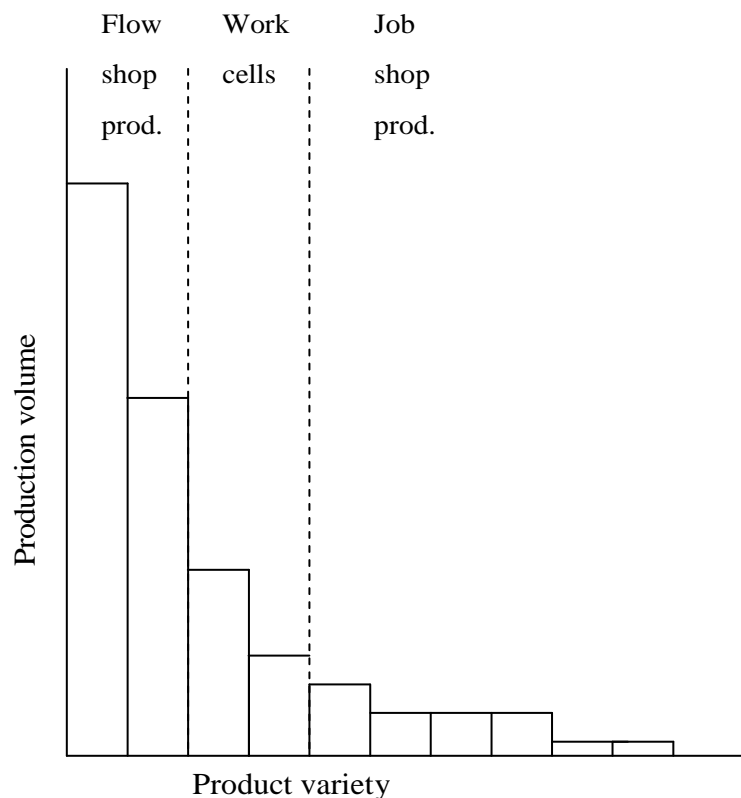


Figure 0-5: Volume-variety chart

Flow shop production is appropriate for high-volume, low variety conditions. Working cell manufacturing systems are usually used for “in between” conditions, and job shop production is applied for low-volume high-variety settings. In fact, many real world layouts tend to be a combination of all three of them (hybrid layout). The volume-variety mix among products can be such that a few products are manufactured using flow shop production, others using job shop production, and the remainder using working cell manufacturing. Similarly, it may be useful to appropriate to use either job shop production or working cells for the production of individual components and to use a flow shop system for the assembly of the components.

In the following we are going to discuss job shop production, cellular manufacturing systems and flow shop production in more detail. Occurring optimization problems and dedicated solution methods will be discussed as well.

1. Methodological Basics

Complexity

Almost all optimization problems occurring in production and logistics can be solved either exactly or by applying heuristic methods. The selection of a solution method may depend on:

- Software availability
- Cost-benefit
- Problem complexity

Even if we know adequate (time consuming) exact methods we are going to apply heuristic methods if we do not have adequate software available or costs (installation, personnel instruction, etc.) exceed the expected benefit.

On the other hand we know a number of combinatorial problems, which are classified to be „NP-*hard*“, which indicates the assumption that the computational effort for solving the problem will not increase polynomial with the problem dimension. In case of real-world applications with the according problem size we face unacceptable computational times, even for high performance IT-systems, regularly.

LP-Problems (average case) are to be solved with polynomial effort, since the number of simplex-iterations increases linearly with the number of constraints (and each iteration causes quadratic effort).

LP-Problems with integer variables usually are solved by applying a Branch and Bound (B&B) method, where a common LP-model is solved in each iteration. Here the number of iterations increases exponentially with the number of integer variables. Thus, these problems cannot be solved with polynomial effort.

For some problem classes (e.g. transportation problems, (linear) assignment) due to their problem structure integer/binary property of the decision variables is guaranteed automatically leading to a low problem complexity.

Some problems with integer/binary variables can (by using special exact methods) be solved with polynomial effort, anyway.

Referring to heuristic methods we usually distinguish between:

- Starting heuristics (quick generation of a feasible solution)
- Improvement heuristics (start with a feasible solution and try to find a better one)
- Combinations of starting and improvement heuristics

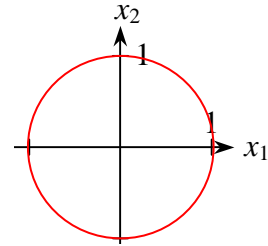
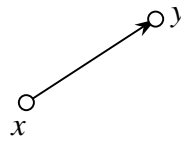
We use “general purpose”-heuristics or metaheuristics (e.g. Simulated Annealing, Tabu Search or Genetic Algorithms) in order to leave local optima during improvement steps.

Costs and distances

The majority of problems being dealt with during this course will be solved based on costs and/or distances c_{ij} . In most cases costs are determined based on given technical parameters (machine setup,..) or distances (e.g. distance between object i and j). In the following we are going to introduce three common distances:

Euclidean distance: $d(x,y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$

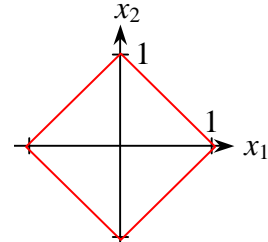
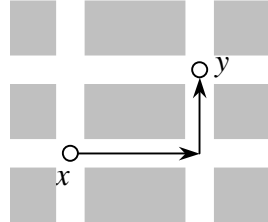
(Straight line distance between two points x and y)



Manhattan distance:

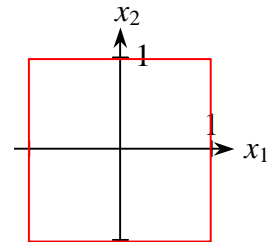
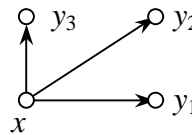
$$d(x,y) = \sum_{i=1}^n |x_i - y_i|$$

(The distance between two points measured along axes at right angle)



Maximum distance: $d(x,y) = \max_{i=1,\dots,n} |x_i - y_i|$

(Drilling plates, movement of cranes,...)

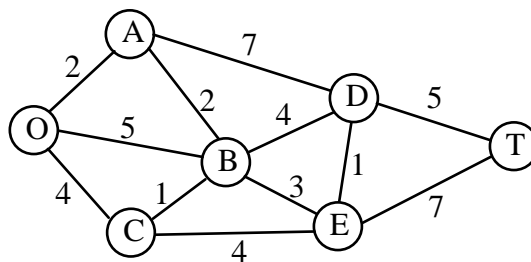


In most cases we know the distance between every couple of adjacent nodes (locations, customers,...). For determining the distance between any two nodes within the network, we have to solve a shortest path problem.

Basics on Graph Theory

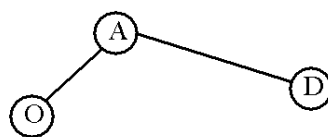
A **graph** consists of points known as **nodes** (*vertices*) which are connected with each other using **lines** (edges, arcs).

Graph:



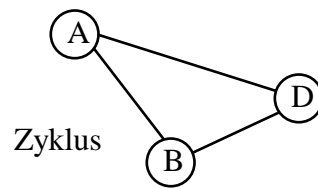
A **chain** between nodes I and j is a sequence of edges connecting these two nodes. A **path** is a chain where the direction is clear (oriented); oriented edges are usually called **arrows** (or arcs).

Chain from A to D:



A **cycle** is a chain that connects a node with itself, while no edge is traversed more than once.

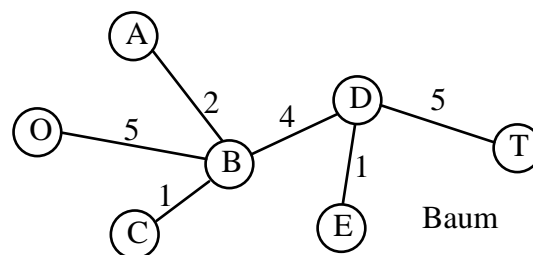
Cycle:



A graph is **connected** if for each pair of nodes there exists a path connecting these two.

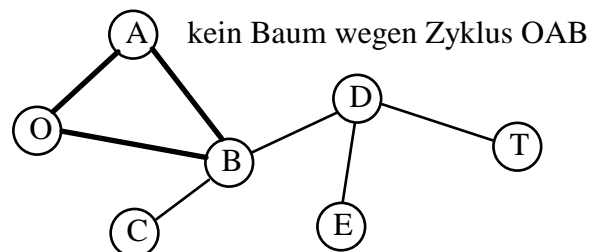
A **tree** is a connected graph without cycles.

Tree:

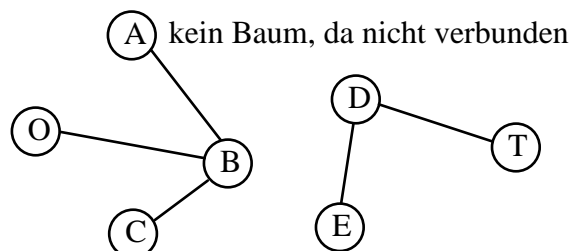


One of the theorems in graph theory indicates that a graph with n nodes is connected if it contains $(n-1)$ nodes, but no cycles.

No tree (due to cycle OAB):



No tree (because not connected):



The edge of graph is **directed** or is an **arrow** if an orientation is given (one way street). A **directed graph** contains only directed arcs. An **undirected graph** contains only undirected edges. A **mixed graph** contains both directed and undirected edges.

2. Job shop production⁴⁵

The process-oriented layout can simultaneously handle a wide variety of products. It is typically the low-volume, high-variety strategy. Each product or product group undergoes a different sequence of operations. It is produced moving it from one department to another in the required sequence. Different products have different material flows. Thus, it is not efficient to arrange machines due to a product-oriented layout (flow shop system) but according to a process-oriented layout.

A process-oriented layout consists of a collection of processing departments or cells. All machines involved in performing a particular process are grouped together in a machine shop (e.g. drill, weld,..). This concept is used when there are many low-volume, dissimilar products. It is also used in case of rapid changes in product mix or volume, as well as when conditions are such that neither product-oriented nor cellular manufacturing systems are useful. In comparison with cellular manufacturing systems this layout concept is characterized by high degrees of interdepartmental flow. A big advantage of this process-oriented layout is its flexibility in equipment and labor assignment. The breakdown of one machine need not halt an entire process; work can be transferred to other machines in the department.

Advantages:

- Better utilization of machines can result; consequently, fewer machines are required.
- A high degree of flexibility exists relative to equipment or manpower allocation for specific tasks.
- Comparatively low investment in machines is required.
- The diversity of tasks offers a more interesting and satisfying occupation for the operator.
- Specialized supervision is possible.

Limitations:

- Since longer flow lines are needed, material handling is more expensive.
- Production planning and control systems are more involved than for other layouts.
- Usually, total production time is longer than for other layouts.

⁴ Heizer, J., Render, B., Operations Management, Prentice Hall, 2006, Chapter 9

⁵ Francis, R., McGinnis, L., White, J., Facility Layout and Location: An Analytical Approach, Prentice Hall, 1992

- Due to the fact that jobs have to queue before being processed in a machine job comparatively large amounts of in-process inventory occur.
- Comparatively high degree of (machine) idle time because machines have to wait until the subsequent job is finished with its foregoing process.
- Space and capital are tied up by work in process.
- Because of the diversity of the jobs in specialized departments, higher grades of skill are required.

Negative effects like idle or waiting times should be reduced by using dedicated production planning methods (on the operational level) and optimized machine shop arrangement on the tactical level. These tactical optimization problems referring to the optimal arrangement of machines in job shop production systems are known as “Assignment problems”. Although the typical problem in this context refers to the “Quadratic assignment problem” we first want to introduce the basic model: the “Linear assignment problem”.

2.1. The Linear Assignment Problem

The *Linear Assignment Problem* (LAP) is one of the most famous problems in linear programming and in combinatorial optimization. Apart from its application to intra-company location planning it can be used for a number of other planning problems.

Given

- n machines (jobs, workers)
- n potential locations (periods, projects)
- c_{ij} ... cost of running machine i on position j .

Any machine can be assigned to any location, incurring some cost that may vary depending on the machine-location assignment. It is required to use all locations by assigning exactly one machine to each location in such a way that the total costs of the assignments are minimized.

The LP is formulated as follows.

$x_{ij} = 1$, if machine i is assigned to location j and 0 otherwise

$$C = \sum_{i=1}^n \sum_{j=1}^n c_{ij} x_{ij} \rightarrow \min \quad \text{s.t.} \quad \begin{aligned} \sum_{j=1}^n x_{ij} &= 1 \quad \text{for } i=1, \dots, n \quad \dots \text{assign all machines} \\ \sum_{i=1}^n x_{ij} &= 1 \quad \text{for } j=1, \dots, n \quad \dots \text{1 machine at each location} \\ x_{ij} &\geq 0 \quad \text{for } i=1, \dots, n \quad \text{and } j=1, \dots, n \end{aligned}$$

Example 1: 3 machines, 4 locations and the following costs c_{ij} (machine 2 may not be assigned to location 2 - > cost ∞):

		location $j =$			
		1	2	3	4
$i =$	maschine 1	13	10	12	11
	2	15	∞	13	20
	3	5	7	10	6

Convert the given problem into a symmetric one by adding dummy-machines (-rows) or dummy-locations (-columns) with cost 0:

		location $j =$			
		1	2	3	4
$i =$	maschine 1	13	10	12	11
	2	15	∞	13	20
	3	5	7	10	6
<i>dummy</i>	4	0	0	0	0

Locations being assigned to a dummy-machine remain empty. A machine being assigned to a dummy-location means that this machine is physically not allocated to any of the potential locations.

2.1.1. Formulation as transportation problem

Linear assignment problems can be interpreted as special case of a general *transportation problem* (TP). The latter is formulated as follows:

m supplier with supply $s_i, i = 1, \dots, m$

n consumer with demand $d_j, j = 1, \dots, n$

transportation costs c_{ij} per unit transported from i to j

decision variables x_{ij} indicate the amount of units transported from i to j

$$\text{Transportation cost } K = \sum_{i=1}^m \sum_{j=1}^n c_{ij} x_{ij} \rightarrow \min$$

$$\text{Supply } s_i = \sum_{j=1}^n x_{ij} \quad i = 1, \dots, m$$

$$\text{Demand } d_j = \sum_{i=1}^m x_{ij} \quad j = 1, \dots, n.$$

$$\text{Nonnegativity } x_{ij} \geq 0 \quad i = 1, \dots, m; j = 1, \dots, n.$$

In order to derive the LAP from the TP we interpret machines as suppliers with a capacity of 1 and each location as consumer with a demand of 1. Thus, each LAP can be solved as special case of transportation problems. It is known that due to the problem structure the optimal solution of a TP consists of $(m+n-1)$ integer basis variables (even if we do not explicitly constrain integer property!). Clearly, this problem characteristic is valid for the LAP (as it has been derived as special case of the TP) as well. Since for the LAP we assume “supply” (machines) = “demand” (locations) = 1 it is automatically guaranteed to obtain an optimal solution consisting of exactly n decision variables with value 1 while all other variables are 0 (although the formulation basically allows non-integer variables as well). Thus, we obtain a feasible solution for the LAP. Clearly, finding a feasible solution premises to have an equal number of machines and locations ($m=n$), which has already been mentioned above and is part of the mathematical LAP formulation as well.

Example 1: as TP:

$i \setminus j$	1	2	3	4	s_i
1	13	10	12	11	1
2	15	∞	13	20	1
3	5	7	10	6	1
4	0	0	0	0	1
d_j	1	1	1	1	

There is another problem characteristic which we are going to make use of in order to solve the LAP: it is always possible to reduce (or increase) all entries of any column or row by a certain value without changing the optimal solution (only the absolute costs change, the relation stays the same). We use this characteristic in order to generate the maximum amount of 0 entries in the cost matrix. By subtracting the smallest element of each column and row from all elements of this column/row we generate the maximum number of 0 entries while not having any influence on the optimal solution. Clearly, the absolute cost factors do change by this matrix reduction, but the relation of assignment costs for each machine/location definitely stays the same.

Example 2: Cost reduction

Cost matrix:

	I	II	III
A	1	8	15
B	6	2	10
C	7	9	3

In this case the column minimum method finds the optimal solution. Machines A, B, and C are assigned to locations I, II, and III, respectively. We end up with total assignment costs of 6 (=1+2+3).

Cost reduction:

	I	II	III
A	0	6	12
B	5	0	7
C	6	7	0
	-1	-2	-3

Again the column minimum method leads to the following assignment: A-I, B-II, C-III. This solution has total (reduced) assignment costs of 0 which implies that we found an optimal solution. By adding the sum of reduction values to the reduced assignment costs we again determine the total assignment costs: $0+1+2+3=6$.

Usually, especially when solving larger problems, it is necessary to apply some iterations of the Transport-Simplex-Method in order to find the optimal solution.

However, for the LAP we know specialized methods leading to the optimal solution more quickly (cf. **Fehler! Verweisquelle konnte nicht gefunden werden.** referring to problem complexity). The most famous one will be presented in the following.

2.1.2. Assignment Method (Kuhn's Algorithm)⁶

Kuhn's Algorithm involves adding/subtracting appropriate values to/from the given cost factors in order to find the lowest opportunity cost (foregone or not-obtained profits) for each assignment.

“There are 3 steps to be followed:

1. *Subtract the smallest number in each column from every number in that column and then, from the resulting matrix, subtract the smallest number in each row from every number in that row. This step has the effect of reducing the numbers in the table until a series of zeros (at least 1 per column and row), meaning zero opportunity costs, appear.”⁵*
2. Draw the minimum number of vertical and horizontal straight lines necessary to cover all zeros in the table. This minimum number of lines equals the maximum number of zero cost assignments. Thus, if the number of lines equals the number of rows/columns in the

⁶ Heizer, J., Render, B., Operations Management, Prentice Hall, 2006, Chapter 15

table, then we can make an optimal assignment. If the number of lines is less than the number of rows or columns, we proceed to step 3.

At this point we want to complete this step of the Assignment Method by specifying the procedure during step 2. In fact, finding the minimum number of vertical and horizontal lines necessary to cover all zeros in a matrix may be trivial in case of very small matrices, but should be solved systematically in case of larger ones. Thus, we want to introduce the following procedure in addition to step 2:

We proceed systematically by choosing a column or row with as few as possible zero entries (preferably exactly one 0) and framing (shading) a 0 in this column or row. This leads to an interim assignment.

Then we cross all remaining zeros in this column or row. Now in each column or row related to a framed 0 all other zeros are crossed which means that in this column or row no further assignments are possible.

Now the next column or row with as few as possible non-marked (not crossed and not framed) zeros is chosen and so on. We stop as soon as we do not have zeros left to be framed. Now we have an arrangement of marked columns and rows including all zeros.

If we are able to make an assignment with (reduced) costs of 0 for each machine we have found an optimal assignment otherwise we proceed as follows:

- 2.1. Mark (for example „X“) all rows with no framed 0
 - 2.2. Mark all columns having at least 1 crossed 0 in a marked row
 - 2.3. Mark all rows having a framed 0 in a marked column
 - 2.4. Repeat 2.2 and 2.3 until there is no column or row left to be marked
 - 2.5. Mark each non-marked row and each marked column (shaded) with a continuous line -> all framed zeros are crossed now and we have the minimum number of crossed -> lines and rows needed to cover all zeros, i.e. the maximum number of zero cost assignments. If this number equals the number of rows or columns an optimal assignment is already found (in this case it would not have been necessary to perform the given subprocedure (2.1.-2.6.) because we should already have succeeded in finding a zero cost assignment as described above).
3. *“Subtract the smallest number not covered by a line from every other uncovered number. Add the same number to any number(s) lying at the intersection of any two lines. Do not change the value of the numbers that are covered by only one line. Return to step 2 and continue until an optimal assignment is possible.*

Some assignment problems entail maximizing profit, effectiveness, or payoff of an assignment of people to tasks or of jobs to machines. It is easy to obtain an equivalent minimization problem by converting every number in the matrix to an opportunity loss. To convert a maximization problem to an equivalent minimization problem, we subtract every number in the original matrix from the largest single number in that matrix. We then proceed to step 1. It turns out that

minimizing the opportunity loss produces the same assignment solution as the original maximization problem.”⁷

Example 1:

$i \setminus j$	1	2	3	4	s_i	
1	13	10	12	11	1	-10
2	15	∞	13	20	1	-13
3	5	7	10	6	1	-5
4	0	0	0	0	1	
d_j	1	1	1	1		

Step 1: Reduced costs

$i \setminus j$	1	2	3	4	s_i
1	3	0	2	1	1
2	2	∞	0	7	1
3	0	2	5	1	1
4	0	0	0	0	1
d_j	1	1	1	1	

Step 2: Optimal solution?

$i \setminus j$	1	2	3	4	s_i
1	3	0	2	1	1
2	2	∞	0	7	1
3	0	2	5	1	1
4	0	0	0	0	1
d_j	1	1	1	1	

In this case we see at first glance that an optimal solution is obtained with the following assignment: 1-2 2-3 3-1 4-4 (we have a zero cost assignment for each machine; the minimum number of lines needed to cover all zero elements would be equal to the number of rows/columns).

Example 2:

⁷ Heizer, J., Render, B., Operations Management, Prentice Hall, 2006, Chapter 15

Step 1: Cost reduction

17,5	15	9	5,5	12
16	16,5	10,5	5	10,5
12	15,5	14,5	11	5,5
4,5	8	14	17,5	13
13	9,5	8,5	12	17,5

-4,5 -8 -8,5 -5 -5,5

⇒

-0,5

⇒

Step 2: Optimal solution?

12,5	6,5	0	0	6
11,5	8,5	2	0	5
7,5	7,5	6	6	0
0	0	5,5	12,5	7,5
8,5	1,5	0	7	12

We are not able to make an assignment with (reduced) costs of 0 for each machine. Thus, we proceed with finding the minimum arrangement of marked columns and rows including all 0 elements.

12,5	6,5	0	0	6
11,5	8,5	2	0	5
7,5	7,5	6	6	0
0	0	5,5	12,5	7,5
8,5	1,5	0	7	12

12,5	6,5	0	0	6	X (1c)
11,5	8,5	2	0	5	X (2c)
7,5	7,5	6	6	0	
0	0	5,5	12,5	7,5	
8,5	1,5	0	7	12	X (1a)
		X (1b)	X (2b)		

Step 3: Generation of additional zeros.

From all covered elements we choose the smallest. This element a is going to be subtracted from all not covered elements and is going to be added to all elements being covered twice.

11	5	0	0	4,5	
10	7	2	0	3,5	
7,5	7,5	7,5	7,5	0	
0	0	7	14	7,5	
7	0	0	7	10,5	1 additional zero (assignment 5 \rightarrow 2) increases the chance the find an assignment with total (reduced) costs of 0.

Step 2: Optimal solution?

Again we have to find out if we already are able to determine the optimal assignment.

Iteration 2:

11	5	0	0	4,5
10	7	2	0	3,5
7,5	7,5	7,5	7,5	0
0	0	7	14	7,5
7	0	0	7	10,5

We have found the optimal solution with reduced assignment costs of 0.

The total costs are calculated by summing up all reduction values (clearly, element "a" determined in step 3 is a reduction value as well):

$$K = (4,5 + 8 + 8,5 + 5 + 5,5 + 0,5) + (1,5) = 33,5$$

2.2. The Quadratic Assignment Problem (QAP)

The more common mathematical formulation for intra-company location problems (especially in case of job shop production) is the *Quadratic Assignment Problem* (QAP). For the QAP the cost of an assignment is determined by the distances and the material flows between all given entities. While, in case of LAP the costs for assigning a machine to a location do not depend on the location chosen for any other machine we now want to take distances of locations and material flow between entities into account as well. In fact, we are now going to minimize the total *transportation* costs occurring due to the chosen assignment whereas for the LAP we minimize isolated location-oriented costs.

So called “*Activity Relationship Charts*” are useful graphical means of representing the desirability of locating pairs of machines/operations near to each other. The following letter codes have been suggested in literature for determining a “closeness” rating:⁸

- “A *Absolutely necessary. Because two machines/operations use the same equipment or facilities, they must be located near each other.*
- E *Especially important. The facilities may for example require the same personnel or records.*
- I *Important. The activities may be located in sequence in the normal work flow.*
- O *Ordinary importance. It would be convenient to have the facilities near each other, but it is not essential.*
- U *Unimportant. It does not matter whether the facilities are located near each other or not.*
- X *Undesirable. Locating a wedding department near one that uses flammable liquids would be an example of this category.”⁷*

Example⁸: “*Met Me, Inc., is a franchised chain of fast-food hamburger restaurants. A new restaurant is being located in a growing suburban community near Reston, Virginia. Each restaurant has the following departments:*

1. *Cooking burgers*
2. *Cooking fries*
3. *Packing and storing burgers*
4. *Drink dispensers*
5. *Counter servers*
6. *Drive-up server*

⁸ Nahmias, S.: Production and Operations Analysis, 4th ed., McGraw-Hill, 2000, Chapter 10

The burgers are cooked on a large grill, and the fries are deep fried in hot oil. For safety reasons the company requires that these cooking areas not be located near each other. All hamburgers are individually wrapped after cooking and stored near the counter. The service counter can accommodate six servers, and the site has an area reserved for a drive-up window.

An activity relationship chart for this facility appears in the following. In the chart, each pair of activities is given one of the letter designations A, E, I, O, U, or X. Once a final layout is determined, the proximity of the various departments can be compared to the closeness ratings in the chart. Figure 2-1 illustrates the activity relationship chart for Met me Inc .”⁸

In the original conception of the QAP a number giving the reason for each closeness rating is needed as well. In case of closeness rating “X” a negative value would be used to indicate the undesirability of closeness for the according machines/operations.

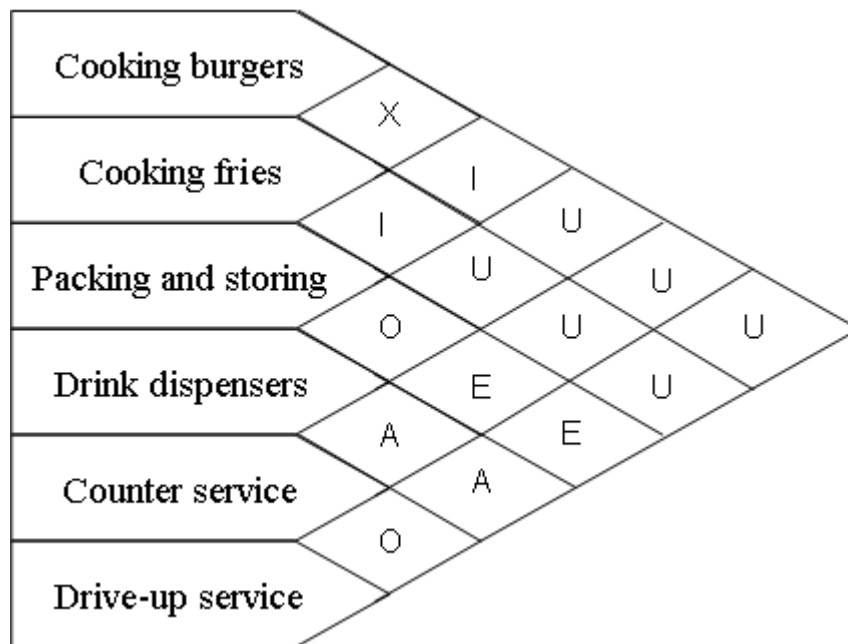


Figure 2-1: Activity relationship chart

2.2.1. QAP: Mathematical formulation

For the model formulation we need both distances between the locations and material flow between *organizational entities* (OE):

- n organizational entities (OE): all of them are of same size and can therefore be interchanged with each other.
- n locations: each can be provided with exactly one OE.
- t_{hi} ... intensity of material flow from OE h to OE i
- d_{jk} ... distance between location j and location k (e.g. shortest distance of central points); distances are not necessarily symmetric. Transportation costs are proportional to amount transported *and* to distance.

If OE h is allocated to location j and OE i to location k the transportation costs per unit from OE h to OE i are defined by d_{jk} . Similar to the LAP we define

$$\text{binary decision variable } x_{hj} = \begin{cases} 1 & \text{if OE } h \text{ is assigned to location } j \\ 0 & \text{otherwise} \end{cases}$$

Transportation costs per unit from OE h to OE i are $\sum_{j=1}^n \sum_{k=1}^n d_{jk} x_{hj} x_{ik}$.

The objective function minimizes total transportation costs between all OE:

$$\sum_{h=1}^n \sum_{i=1}^n \sum_{j=1}^n \sum_{k=1}^n t_{hi} d_{jk} x_{hj} x_{ik} \rightarrow \min$$

where we refer to the following constraints (similar to the LAP):

$$\sum_{j=1}^n x_{hj} = 1 \quad \text{for } h = 1, \dots, n \quad \dots \text{ each OE } h \text{ on exactly 1 location } j$$

$$\sum_{h=1}^n x_{hj} = 1 \quad \text{for } j = 1, \dots, n \quad \dots \text{ each location } j \text{ gets exactly 1 OE } h$$

$$x_{hj} = 0 \text{ or } 1 \quad \dots \text{ binary decision variable}$$

While all constraints are still linear we now face a non-linear objective function. Due to the combination of integer property and non-linearity finding optimal solutions for larger problems is almost impossible (cf. **Fehler! Verweisquelle konnte nicht gefunden werden.**). Thus, heuristic methods are applied in most cases. As usual we distinguish between starting heuristics and improvement methods or a combination of both of them.

Example: Calculation of costs of 3 OE (1, 2, 3) and 3 locations (A, B, C)

	Distances between locations d_{jk}	Intensity of material flow t_{hi}																																
	<table style="border-collapse: collapse; margin: 0 auto;"> <tr> <td style="border: none;"></td> <td style="border: none; text-align: center;">A</td> <td style="border: none; text-align: center;">B</td> <td style="border: none; text-align: center;">C</td> </tr> <tr> <td style="border: none; text-align: center;">A</td> <td style="border: 1px solid black; padding: 5px;">0</td> <td style="border: 1px solid black; padding: 5px;">1</td> <td style="border: 1px solid black; padding: 5px;">2</td> </tr> <tr> <td style="border: none; text-align: center;">B</td> <td style="border: 1px solid black; padding: 5px;">1</td> <td style="border: 1px solid black; padding: 5px;">0</td> <td style="border: 1px solid black; padding: 5px;">1</td> </tr> <tr> <td style="border: none; text-align: center;">C</td> <td style="border: 1px solid black; padding: 5px;">3</td> <td style="border: 1px solid black; padding: 5px;">1</td> <td style="border: 1px solid black; padding: 5px;">0</td> </tr> </table>		A	B	C	A	0	1	2	B	1	0	1	C	3	1	0	<table style="border-collapse: collapse; margin: 0 auto;"> <tr> <td style="border: none;"></td> <td style="border: none; text-align: center;">1</td> <td style="border: none; text-align: center;">2</td> <td style="border: none; text-align: center;">3</td> </tr> <tr> <td style="border: none; text-align: center;">1</td> <td style="border: 1px solid black; padding: 5px;">0</td> <td style="border: 1px solid black; padding: 5px;">1</td> <td style="border: 1px solid black; padding: 5px;">1</td> </tr> <tr> <td style="border: none; text-align: center;">2</td> <td style="border: 1px solid black; padding: 5px;">2</td> <td style="border: 1px solid black; padding: 5px;">0</td> <td style="border: 1px solid black; padding: 5px;">2</td> </tr> <tr> <td style="border: none; text-align: center;">3</td> <td style="border: 1px solid black; padding: 5px;">3</td> <td style="border: 1px solid black; padding: 5px;">1</td> <td style="border: 1px solid black; padding: 5px;">0</td> </tr> </table>		1	2	3	1	0	1	1	2	2	0	2	3	3	1	0
	A	B	C																															
A	0	1	2																															
B	1	0	1																															
C	3	1	0																															
	1	2	3																															
1	0	1	1																															
2	2	0	2																															
3	3	1	0																															

One possible solution would be $1 \rightarrow A, 2 \rightarrow B$ and $3 \rightarrow C$, i.e. $x_{1A} = 1, x_{2B} = 1, x_{3C} = 1$ and all other $t_{hj} = 0$. All constraints are fulfilled.

Total transportation cost: $0 \cdot 0 + 1 \cdot 1 + 2 \cdot 1 + 1 \cdot 2 + 0 \cdot 0 + 1 \cdot 2 + 3 \cdot 3 + 1 \cdot 1 + 0 \cdot 0 = 17$

Obviously, this solution is not optimal since OE 1 and 3 (which have the highest intensity of material flow) are assigned to the locations with the highest distance between them (A and C).

A better solution would for example be $1 \rightarrow C, 2 \rightarrow A$ and $3 \rightarrow B$, i.e. $x_{1C} = 1, x_{2A} = 1, x_{3B} = 1$.

Total transportation cost: $0 \cdot 0 + 3 \cdot 1 + 1 \cdot 1 + 2 \cdot 2 + 0 \cdot 0 + 2 \cdot 1 + 1 \cdot 3 + 1 \cdot 1 + 0 \cdot 0 = 14$

	Distances	Intensity of material flow																																
	<table style="border-collapse: collapse; margin: 0 auto;"> <tr> <td style="border: none;"></td> <td style="border: none; text-align: center;">C</td> <td style="border: none; text-align: center;">A</td> <td style="border: none; text-align: center;">B</td> </tr> <tr> <td style="border: none; text-align: center;">2→A</td> <td style="border: 1px solid black; padding: 5px;">0</td> <td style="border: 1px solid black; padding: 5px;">3</td> <td style="border: 1px solid black; padding: 5px;">1</td> </tr> <tr> <td style="border: none; text-align: center;">3→B</td> <td style="border: 1px solid black; padding: 5px;">2</td> <td style="border: 1px solid black; padding: 5px;">0</td> <td style="border: 1px solid black; padding: 5px;">1</td> </tr> <tr> <td style="border: none; text-align: center;">1→C</td> <td style="border: 1px solid black; padding: 5px;">1</td> <td style="border: 1px solid black; padding: 5px;">1</td> <td style="border: 1px solid black; padding: 5px;">0</td> </tr> </table>		C	A	B	2→A	0	3	1	3→B	2	0	1	1→C	1	1	0	<table style="border-collapse: collapse; margin: 0 auto;"> <tr> <td style="border: none;"></td> <td style="border: none; text-align: center;">1</td> <td style="border: none; text-align: center;">2</td> <td style="border: none; text-align: center;">3</td> </tr> <tr> <td style="border: none; text-align: center;">1</td> <td style="border: 1px solid black; padding: 5px;">0</td> <td style="border: 1px solid black; padding: 5px;">1</td> <td style="border: 1px solid black; padding: 5px;">1</td> </tr> <tr> <td style="border: none; text-align: center;">2</td> <td style="border: 1px solid black; padding: 5px;">2</td> <td style="border: 1px solid black; padding: 5px;">0</td> <td style="border: 1px solid black; padding: 5px;">2</td> </tr> <tr> <td style="border: none; text-align: center;">3</td> <td style="border: 1px solid black; padding: 5px;">3</td> <td style="border: 1px solid black; padding: 5px;">1</td> <td style="border: 1px solid black; padding: 5px;">0</td> </tr> </table>		1	2	3	1	0	1	1	2	2	0	2	3	3	1	0
	C	A	B																															
2→A	0	3	1																															
3→B	2	0	1																															
1→C	1	1	0																															
	1	2	3																															
1	0	1	1																															
2	2	0	2																															
3	3	1	0																															

2.2.2. Starting heuristics

Some starting heuristics refer to the combination of one of the following possibilities to select an OE and a location. The *core* is defined by the already chosen OE. After each iteration another OE is added to the core due to one of the following *priorities*:

- **Selection of (non-assigned) OE**

- A1. those having the maximum sum of material flow to all (other) OE
- A2. a) those having the maximum material flow to the last-assigned OE
b) those having the maximum material flow to an assigned OE
- A3. those having the maximum material flow to all assigned OE (core)
- A4. random choice

- **Selection of (non-assigned) locations**

- B1. those having the minimum total distance to all other locations
- B2. those being neighbouring to the last-chosen location
- B3. a) those leading to the minimum sum of transportation cost to the core
b) like a) but furthermore we try to exchange the location with neighbored OE
c) a location (empty or allocated) such that the sum of transportation costs within the new core is minimized (in case an allocated location is selected, the displaced OE is assigned to an empty location)
- B4. random choice

Example: By combining the simple rules A1 and B1 we have to arrange all OE referring to decreasing sum of material flow and all locations referring to increasing sum of distances to all other OE/locations (i.e. the last columns in the following tables):

OE	1	2	3	4	5	6	7	8	9	Σ	St.	A	B	C	D	E	F	G	H	I	Σ
1	-	-	-	-	3	-	-	-	-		A	-	1	2	1	2	3	2	3	4	
2		-	3	1	2	-	4	-	-		B		-	1	2	1	2	3	2	3	
3			-	3	5	2	-	3	4		C			-	3	2	1	4	3	2	
4				-	-	-	1	-	-		D				-	1	2	1	2	3	
5					-	2	2	1	-		E					-	1	2	1	2	
6						-	-	-	-		F						-	3	2	1	
7							-	-	-		G							-	1	2	
8								-	-		H								-	1	
9									-		I									-	

This leads to the following sequence of OE and locations: 3, 5, 2, 7, 4, 6, 8, 9, 1 ; E, B, D, F, H, A, C, G, I.

We obtain the following assignment of OE to locations:

OE	1	2	3	4	5	6	7	8	9
St.	I	D	E	H	B	A	F	C	G

OE	1	2	3	4	5	6	7	8	9
1	-	-	-	-	3×3	-	-	-	-
2		-	3×1	1×2	2×2	-	4×2	-	-
3			-	3×1	5×1	2×2	-	3×2	4×2
4				-	-	-	1×2	-	-
5					-	2×1	2×2	1×1	-
6						-	-	-	-
7							-	-	-
8								-	-
9									-

Calculation of costs:

1 and 5 are assigned to I and B with material flow of 3 and

3 (distance 1-5) × 3 (flow I-B)

and so on

Total cost = 61

In case of random choice (rules A4 and/or B4) one has the possibility to generate a set of different solutions and to choose the best one out of it.

2.2.3. Improvement methods

We basically try to improve solutions (i.e. reduce costs) by exchanging OE-pairs (see also the introductory example above). In case of acceptable computational times one can also try to exchange OE-triples. Even in case of pair wise exchanges we have different possibilities:

- **Selection of potential pairwise exchanges:**

C1. all $n(n - 1)/2$ pairs

C2. a subset of pairs

C3. random choice

- **Selection of pairwise exchanges that are finally performed:**

D1. that pair whose exchange of locations leads to the highest cost reduction. (best pair)

D2. the first pair whose exchange of locations leads to a cost reduction (first pair)

A combination of C1 and D1 increases solution quality but also computational time. A common method is to start with C2 and skip to C1 as soon as the solution is reasonably good. (A combination of C1 and D2 is equivalent to the 2-opt method which we use to solve TSP).

A well-known (heuristic) method is *CRAFT* (computerized relative allocation of facilities techniques) which equals (in case of OE with similar place requirements) a combination of C1 and D1 (this method will be introduced later in this chapter in the context of OE with unequal place requirements).

In case of random choice (C3 and D2) we quite often find good results. Especially the fact that sometimes the best exchange of all exchanges which have been checked leads to an increase of costs is no disadvantage, because it reduces the risk to be trapped in local optima.

The basic idea and several adaptations/combinations of A, B, C, and D are discussed in literature.

2.2.4. „Umlaufmethode“

„Umlaufmethode“ is one of the numerous heuristics which combine the idea of starting heuristics and improvement methods. This method consists of the following components:

Initialization ($i = 1$):

Those OE having the maximum sum of material flow [A1] is assigned to the centre of locations (i.e. the location having the minimum sum of distances to all other locations [B1]).

Iteration i ($i = 2, \dots, n$): assign OE i

Part 1: (*Selection of OE and of free location*):

- select those OE with the maximum sum of material flow to all OE assigned to the core [A3]
- assign the selected OE to a free location so that the sum of transportation costs to the core (within the core) is minimized [B3a]

Part 2: (*Improvement step in iteration $i = 4$*):

- check pair wise exchanges of the last-assigned OE with all other OE in the core [C2]
- if an improvement is found, the exchange is conducted and we start again with Part 2 [D2].

The method ends with the finalization of iteration $i = n$ having assigned all OE.

Example, at first without improvement step (only Part 1):

Initialization ($i = 1$):

E is the centre

We assign OE 3 to the centre (E)

A	B	C
D	E 3	F
G	H	I

Sequence of assignments:

$i =$	1	2	3	4	5	6	7	8
OE	3							
1	0							
2	3							
3	↑							
4	3							
5	5							
6	2							
7	0							
8	3							
9	4							

$i = 3$: 2 has the max. mat.fl. to the core (3, 5)

$i = 1$: at first we assign 3

$i = 5$

$i = 2$: 5 has the max. mat.fl. to 3

$i = 6$

$i = 4$

$i = 7$

$i = 8$

Iteration $i = 2$ (Part 1): the maximum material flow to the core (3) is from OE 5.

Distances $d_{BE} = d_{DE} = d_{FE} = d_{HE} = 1$ is equally minimal \Rightarrow we select D

\Rightarrow In step $i = 2$ we assign D-5.

Iteration $i = 3$ (Part 1): the maximum material flow to the core (3, 5) is from OE 2.

Find a location X so that $d_{XE} \cdot t_{23} + d_{XD} \cdot t_{25} = d_{XE} \cdot 3 + d_{XD} \cdot 2$ is minimal (only A, B, G or H)

$$X = A \quad d_{AE} \cdot 3 \quad + \quad d_{AD} \cdot 2 \quad = \quad 2 \cdot 3 \quad + \quad 1 \cdot 2 \quad = \quad 8$$

$$X = B \quad d_{BE} \cdot 3 \quad + \quad d_{BD} \cdot 2 \quad = \quad 1 \cdot 3 \quad + \quad 2 \cdot 2 \quad = \quad 7$$

$$X = F \quad d_{FE} \cdot 3 \quad + \quad d_{FD} \cdot 2 \quad = \quad 1 \cdot 3 \quad + \quad 2 \cdot 2 \quad = \quad 7$$

$$X = G \quad d_{GE} \cdot 3 \quad + \quad d_{GD} \cdot 2 \quad = \quad 2 \cdot 3 \quad + \quad 1 \cdot 2 \quad = \quad 8$$

$$X = H \quad d_{HE} \cdot 3 + d_{HD} \cdot 2 = 1 \cdot 3 + 2 \cdot 2 = 7 \quad \text{B, F or H} \Rightarrow \text{B is chosen}$$

\Rightarrow In Step $i = 3$ we assign B-2.

Iteration $i = 4$ (Part 1) the maximum material flow to the core (2, 3, 5) is from OE 7.

Find a location X so that $d_{XE} \cdot t_{73} + d_{XD} \cdot t_{75} + d_{XB} \cdot t_{72} = d_{XE} \cdot 0 + d_{XD} \cdot 2 + d_{XB} \cdot 4$ is minimal in the map we see that A is the best choice

\Rightarrow in iteration $i = 4$ we tentatively assign A-7.

(Part 2) try to exchange A with E, B or D and calculate the costs of these assignments:

From Part 1: E-3, D-5, B-2, A-7	Cost	=	$1 \cdot 5 + 1 \cdot 3 + 2 \cdot 0 + 2 \cdot 2 + 1 \cdot 2 + 1 \cdot 4$	=	18
E-3, D-5, A-2 , B-7	Cost	=	$1 \cdot 5 + 2 \cdot 3 + 1 \cdot 0 + 1 \cdot 2 + 2 \cdot 2 + 1 \cdot 4$	=	21
E-3, A-5 , B-2, D-7	Cost	=	$2 \cdot 5 + 1 \cdot 3 + 1 \cdot 0 + 1 \cdot 2 + 1 \cdot 2 + 2 \cdot 4$	=	25
A-3 , D-5, B-2, E-7	Cost	=	$1 \cdot 5 + 1 \cdot 3 + 2 \cdot 0 + 2 \cdot 2 + 1 \cdot 2 + 1 \cdot 4$	=	18

An exchange of A and E is possible but does not lead to a cost reduction. Thus, we do not conduct this exchange but take the solution determined in Part 1.

After 8 iterations (without part 2) we end up with the solution from above with total costs = 54.

Inclusion of part 2 leads to an exchange of the last-assigned OE 9 with OE 4 in iteration 8. By this exchange we increase total costs to 51.

While a manual calculation of larger problems is obviously quite time consuming an implementation and therefore computerized calculation is relatively simple.

2.2.5. Different space requirements

The solution methods discussed above are also feasible for problems considering OE with different space requirements (OE are assumed to be either rectangularly shaped or composed of rectangular pieces). But here an exchange of OE may have an influence on the shapes and locations of other (even not-exchanged) OE. Furthermore, one has to define the way of measuring distances between locations, since the distance between two OE may depend on their shapes. The most common distances in this context probably are:

- Orthogonal distance between OE-boundaries: the (shortest) distance between 2 OE is determined by the orthogonal distance between the closest points of them. Thus, OE having at least 1 vertex in common have a distance of 0.
- Rectilinear distance of centre points: the distance between 2 OE is assumed to be the rectilinear distance between centroid locations. This implies the assumption that OE are located at their centroids. The centroid is another term for the coordinates of the centre of gravity or balance point. The accuracy of assuming that an OE is located at its centroid depends upon the shape of the OE. The assumption is most accurate when the shape of the OE is square or rectangular, but is less accurate for oddly shaped OE.

Rectilinear distances between centre points are, e.g., used for the well known CRAFT algorithm, which we are going to discuss in the following.

2.2.5.1. CRAFT Algorithm⁹¹⁰

CRAFT (computerized relative allocation of facilities techniques) was one of the first computer-aided layout routines developed. It is an improvement method which means that it requires an initial layout to be used as a starting solution.

Again we try to improve the given solution by moving around OE. The additional challenge now is that the shapes of OE are not fixed. Thus, the problem simply has too many degrees of freedom for us to devise a good method for modifying the starting solution. All the common improvement methods are based on limiting the kinds of changes that are permitted. This has already been addressed in the context of problems with similar space requirements.

We know that a pair of OE that can be exchanged without a direct influence on the shapes or locations of all remaining OE has to satisfy one of the following conditions:

1. the OE have the same space requirement,
2. the OE share a common boundary (having a common boundary means that the OE share at least 1 side boundary of their rectangles). E.g. OE 2 and OE 5 in Figure 2-2 share 1 side boundary.

In Figure 2-2 you see that an exchange of OE 2 and OE 5 would be possible without affecting the shape or the location of the remaining OE. In this case it does not matter that OE 2 and OE 5 have different space requirements because the total area used for them stays the same. Clearly, an exchange of same sized OE (no matter if they are neighbored or not) is always possible. On the other hand exchanging, e.g., OE 1 and OE 3, is not possible without changing the shape or the location of for example OE 2.

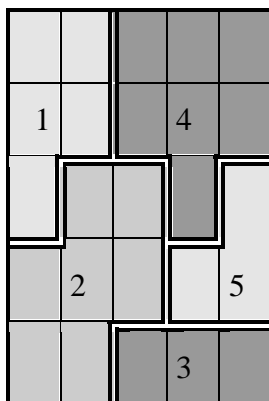


Figure 2-2: Different sized OE

⁹ Francis, R., McGinnis, L., White, J., Facility Layout and Location: An Analytical Approach, Prentice Hall, 1992

¹⁰ Nahmias, S.: Production and Operations Analysis, 4th ed., McGraw-Hill, 2000, Chapter 10

The CRAFT algorithm, which is probably the earliest widely known improvement algorithm, uses an estimate of the transportation cost that is based on the rectilinear distance between the centroid locations. If, e.g., OE 2 and 5 are considered for an exchange, the new costs is estimated by assuming that the new centroid of OE 2 is the old centroid of OE 5 and vice versa. This method of estimating transportation costs for the new layout is exact if OE have the same space requirement, but can be in error if the requirements are different. In this case we revise the estimated transportation costs by developing a distance chart for the new layout and calculating the “real” total transportation costs. This is done whenever an exchange has been identified to be the most useful (based on the estimation of costs) in an iteration. The algorithm continues until no further reductions in the predicted transportation costs can be achieved.

So we summarize the steps to be followed according to CRAFT:

1. Estimate total transportation costs considering all pairwise exchanges of OE that share at least 1 border or that are of same size (i.e. equal number of rectangles).
2. Perform that exchange that leads to the minimum estimated total transportation costs (based on an estimation of distances as described above). If all possible exchanges lead to an increase of predicted total costs, stop here.
3. Revise the estimated distance chart and calculate the new total costs. Go back to step 1.

You see that by applying this procedure the “best” exchange could be passed over, due to estimation errors. This generally will be the case for any improvement algorithm that does not actually evaluate every exchange possible.

Example¹¹: A local manufacturing firm has recently completed construction of a new plant to house 4 departments: A, B, C, and D. The plant is 100m^2 by 50m^2 . The plant manager has chosen an initial layout of the 4 departments. This layout is given in Figure 2-3. From the figure we see that department A requires 1800 m^2 , department B 1200m^2 , department C 800m^2 , and department D 1200m^2 .

¹¹ Nahmias, S.: Production and Operations Analysis, 4th ed., McGraw-Hill, 2000, Chapter 10

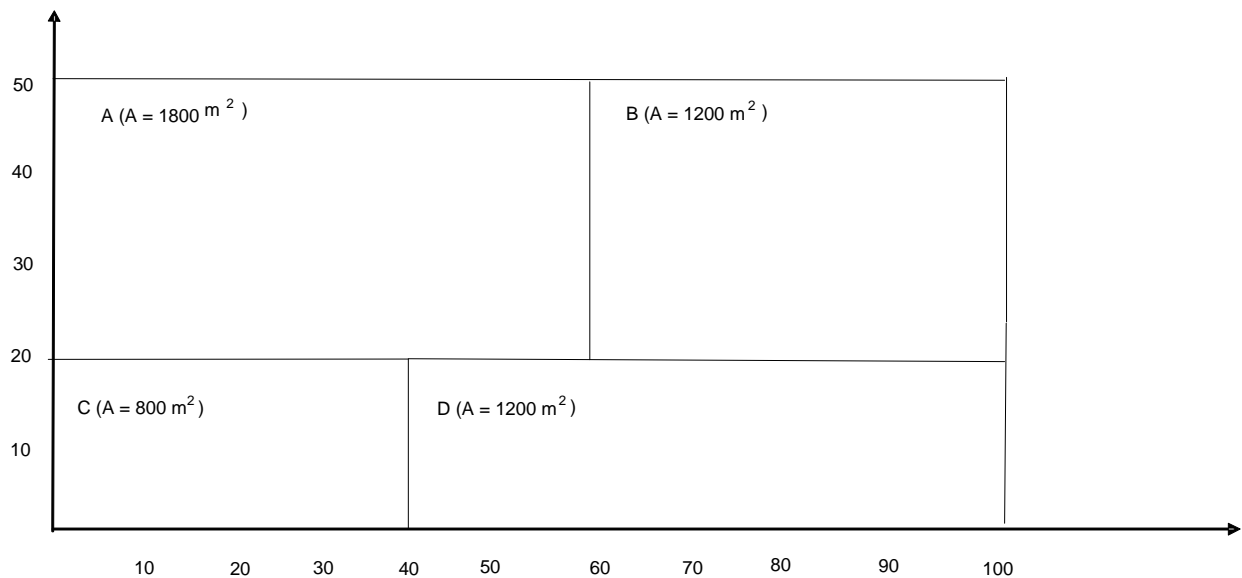


Figure 2-3: Initial plant layout for example problem

The following table contains the material flows between all departments.

Material Flow	A	B	C	D
A	0	2	7	4
B	3	0	5	5
C	6	7	0	3
D	8	2	3	0

The distance between 2 departments is to be assumed as the rectilinear distance between the centroid locations of the corresponding departments. Try to improve the initial layout by applying the CRAFT algorithm (pairwise exchanges).

1. Determination of distances referring to the initial layout: The centroid of a rectangular R is defined by 2 points \bar{x} , \bar{y} .

$$\bar{x} = \frac{(x_1 + x_2)}{2} \quad \bar{y} = \frac{(y_1 + y_2)}{2}$$

The rectilinear distance between 2 centroid locations is

$$d_{12} = |\bar{x}_1 - \bar{x}_2| + |\bar{y}_1 - \bar{y}_2|$$

For the initial layout we obtain the following centroid locations

	Centroid	
	x	y
A	30	35
B	80	35
C	20	10
D	70	10

and the following distances

Distance	A	B	C	D
A	0	50	35	65
B	50	0	85	35
C	35	85	0	50
D	65	35	50	0

- Determination of current total costs:

We multiply the distances with the material flows -> **Total transportation cost: 3050**

- Now we list all possible pairwise exchanges:

A-B -> possible

A-C -> possible

A-D -> possible

B-D -> possible

C-D -> possible

All other exchanges are not possible, because the departments are neither neighboured nor of same size.

- Estimate total costs for each valid exchange (assuming that the centroid locations stay the same):

A-B: estimated transportation cost = 2950

A-C: estimated transportation cost = 2715

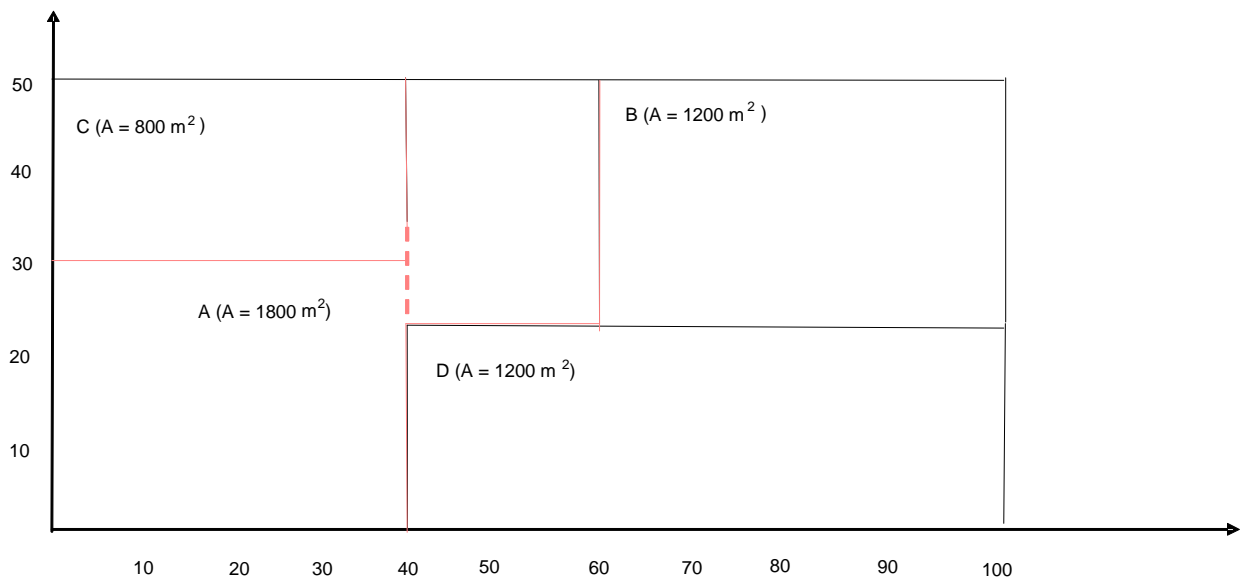
A-D: estimated transportation cost = 3185

B-D: estimated transportation cost = 2735

C-D: estimated transportation cost = 2830

-> Exchange A-C is supposed to be the best one

- Perform the selected exchange and calculate the new total costs.



If a plane R consists of a collection of rectangles R_1, R_2, \dots, R_k the respective boundaries are $[(x_{1i}, x_{2i}, (y_{1i}, y_{2i}))]$ for $1 \leq i \leq k$.

In order to find \bar{x}, \bar{y} (which describe the centroid location), we first have to obtain the moments of R :

$$M_x = \sum_{i=1}^k \frac{x_{2i}^2 - x_{1i}^2}{2} (y_{2i} - y_{1i})$$

$$M_y = \sum_{i=1}^k \frac{y_{2i}^2 - y_{1i}^2}{2} (x_{2i} - x_{1i})$$

Let $A(R)$ be the area of R . Then the centroid of R is given by

$$\bar{x} = \frac{M_x}{A(R)}$$

$$\bar{y} = \frac{M_y}{A(R)}$$

The new centroids are

	Centroid	
	x	y
A	30	21,667
B	80	35
C	20	40
D	70	10

leading to the following new distances

Distance	A	B	C	D
A	0	63,333	28,333	51,667
B	63,333	0	65	35
C	28,333	65	0	80
D	51,667	35	80	0

and total costs of 2809 (we see that we do not achieve the estimated costs for this scenario (2715), but compared to the initial layout's total cost we have a cost reduction anyway).

6. Now we list all possible pairwise exchanges:

A-B -> possible

A-D -> possible

B-D -> possible

All other exchanges are not possible, because the departments are neither neighboured nor of same size.

7. Estimate total costs for each valid exchange (assuming that the centroid locations stay the same):

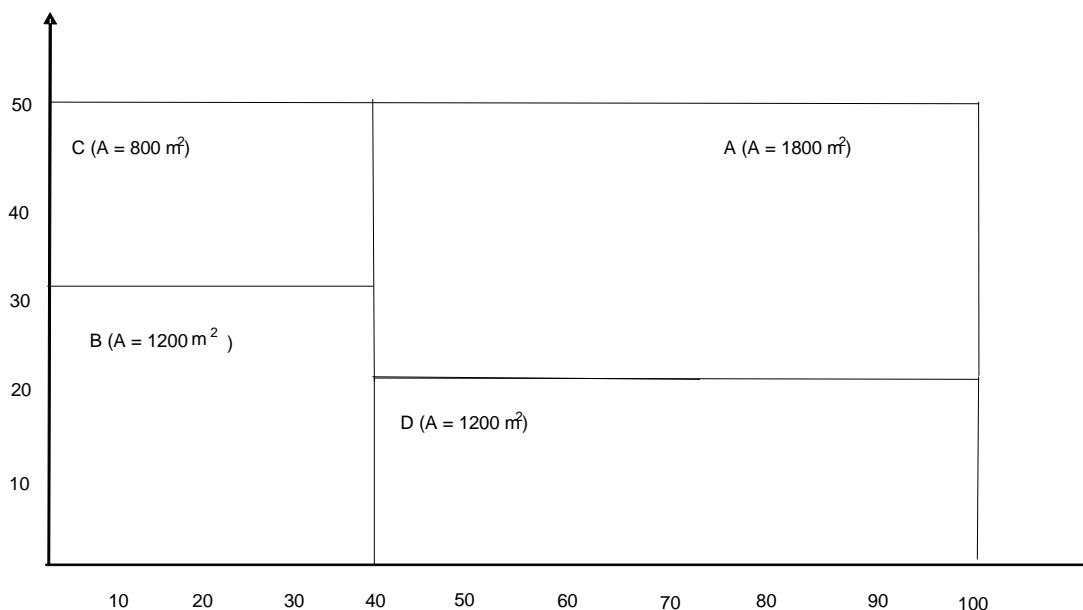
A-B: estimated transportation cost = 2763

A-D: estimated transportation cost = 3228

B-D: estimated transportation cost = 2982

-> Exchange A-C is supposed to be the best one

8. Perform the selected exchange and calculate the new total costs.



The new centroids are

	Centroid	
	x	y
A	70	35
B	20	15
C	20	40
D	70	10

leading to the following distances

Distance	A	B	C	D	
A	0	70	55	25	
B	70	0	25	55	
C	55	25	0	80	
D	25	55	80	0	

and total costs of **2530** (this time we had underestimated the cost reduction).

9. Now we list all possible pairwise exchanges:

A-C -> possible

A-D -> possible

B-C -> possible

B-D -> possible

All other exchanges are not possible, because the departments are neither neighbored nor of same size.

10. Estimate total costs for each valid exchange (assuming that the centroid locations stay the same):

A-C: estimated transportation cost = 3175

A-D: estimated transportation cost = 2735

B-C: estimated transportation cost = 2675

B-D: estimated transportation cost = 3325

-> no further cost reduction is expected! We stop with total costs of **2530!**

3. Group Technology / Cellular Manufacturing¹²

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

¹² This chapter is based on Chapter 6 of Askin & Standridge (1993). It is recommended to read this chapter parallel to the course notes.

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.

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.

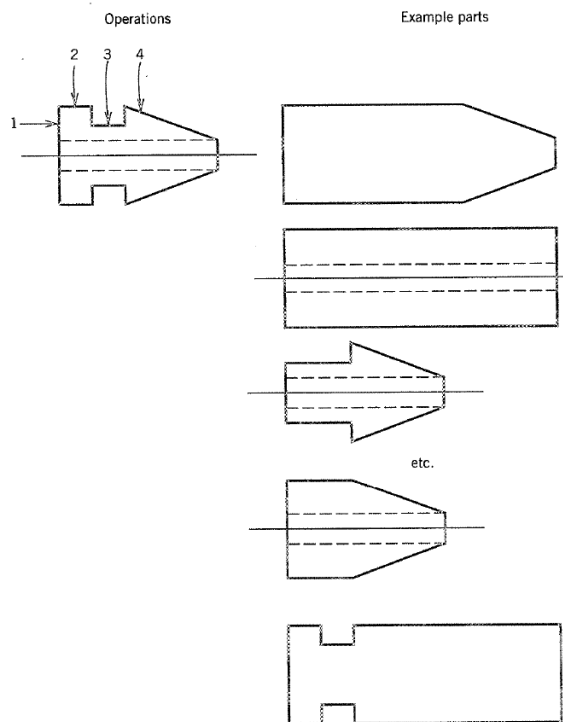


Figure 3.1. Composite Group Technology Part (Askin & Standridge, 1993, p. 165).

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*:

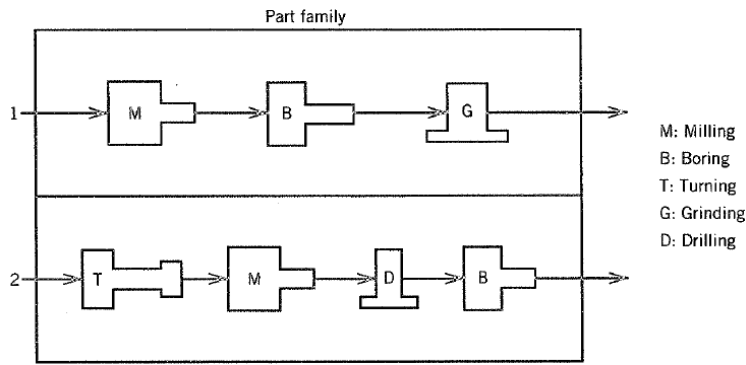


Figure 3.2a. GT flow line
(Askin & Standridge, 1993, p. 167).

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.

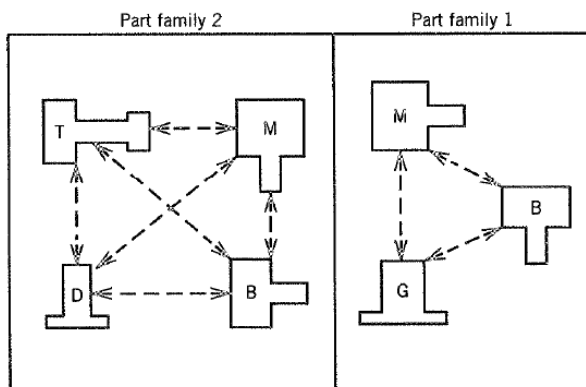


Figure 3.2b. GT cell
(Askin & Standridge, 1993, p. 167).

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.

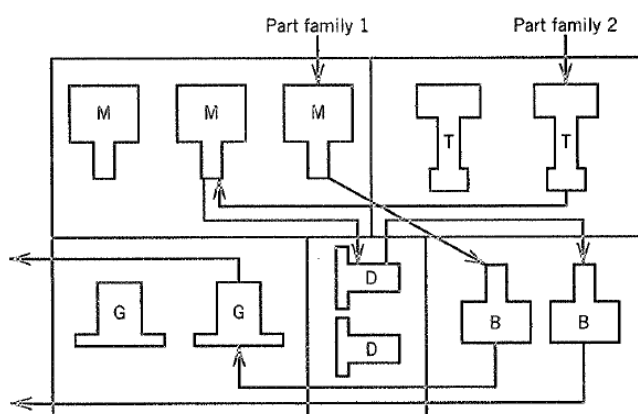


Figure 3.2c. GT center
(Askin & Standridge, 1993, p. 167).

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 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).

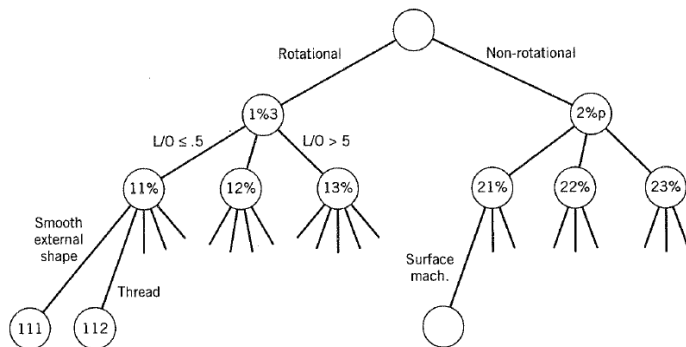


Figure 3.3a. Hierarchical structure.

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: "1232"
- a smooth internal feature: "2132"

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	None	None	No	None	
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	
...					

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 3.3b. Chain structure.

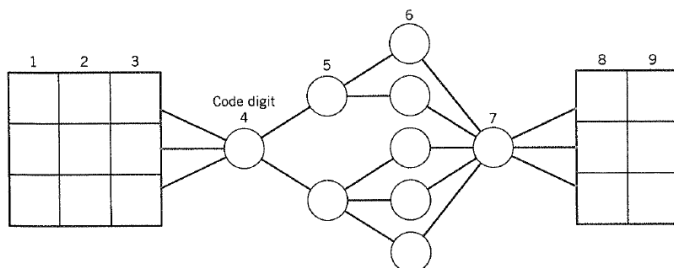


Figure 3.3c. Chain structure.

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.

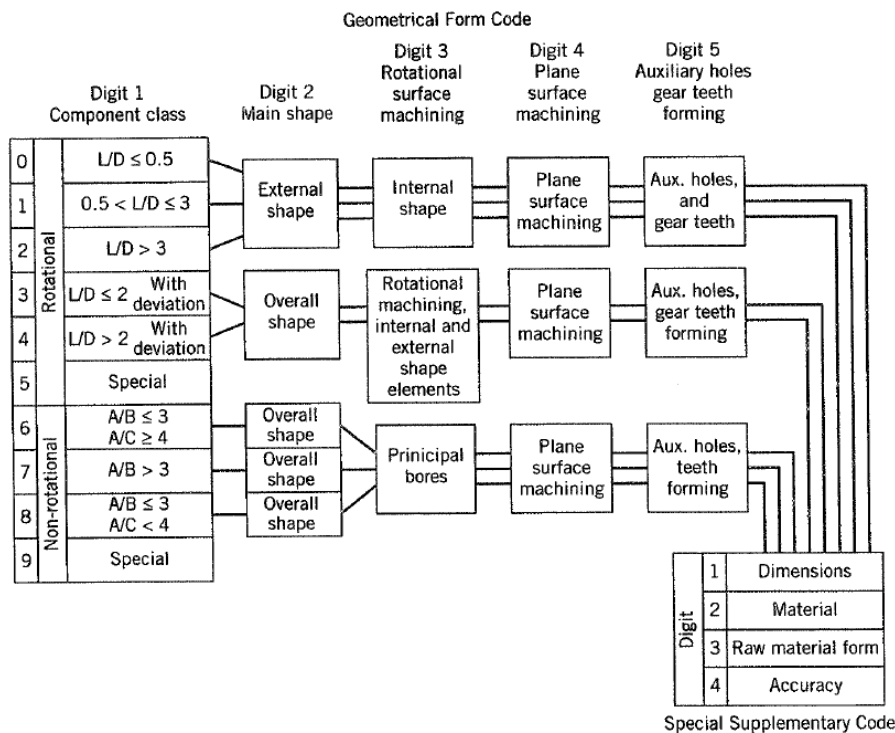
The final decision is, **code representation**. The digits can be

- *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.



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.

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

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

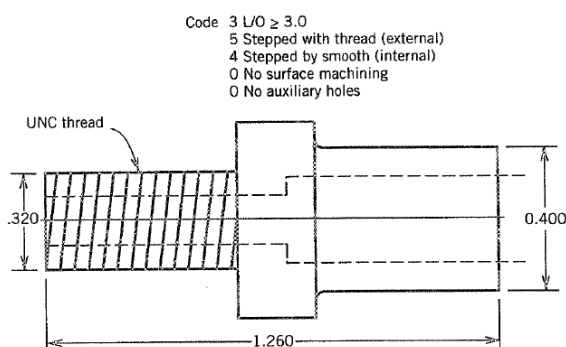


Figure 3.4. Opitz code for sample part (Askin & Standridge, 1993, p. 167).

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:

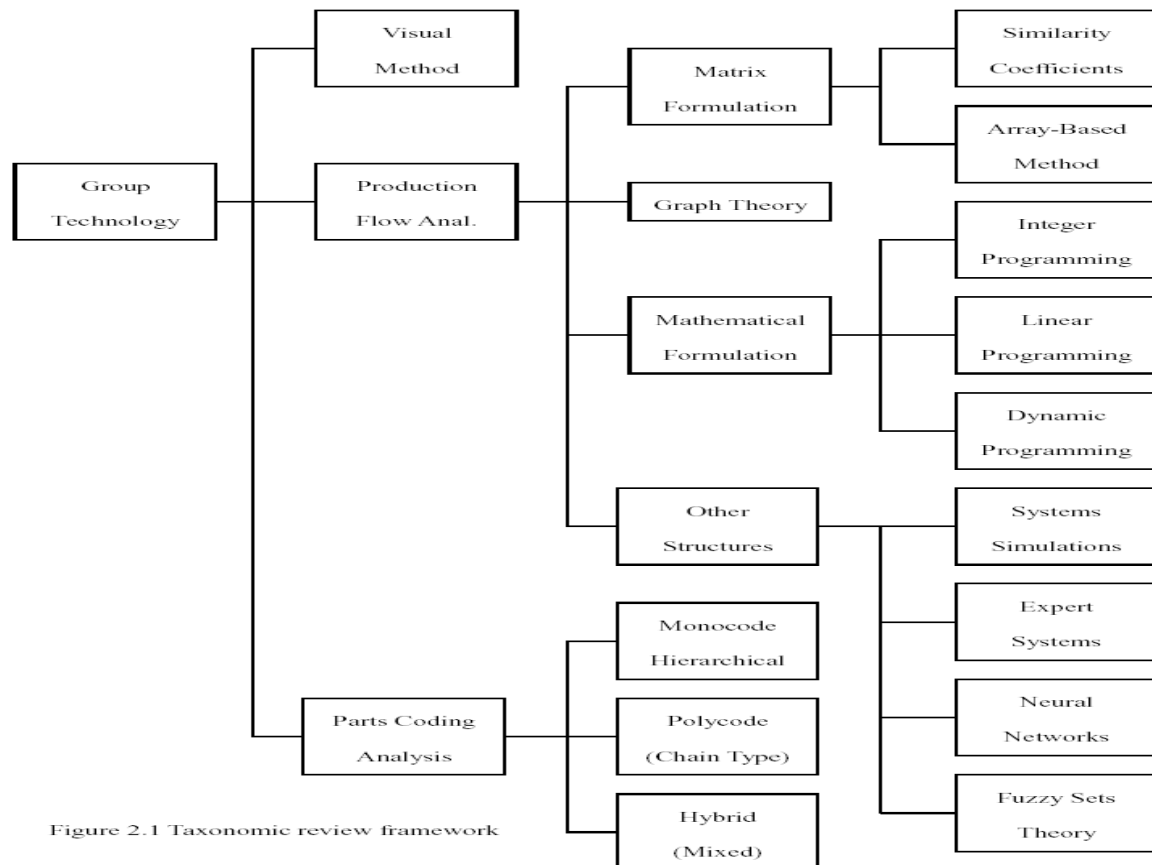


Figure 2.1 Taxonomic review framework

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:

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

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 as rank order clustering or King's algorithm

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

This 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	
D	1	-	-	-	1	1	
C	-	1	1	1	-	1	
A	-	1	-	1	-	-	
E	-	-	-	1	1	-	

<i>B</i>	1	-	1	-	1	1	<i>16</i>
<i>D</i>	1	-	-	-	1	1	<i>6</i>
<i>C</i>	-	1	1	1	-	1	<i>4</i>
<i>A</i>	-	1	-	1	-	-	<i>2</i>
<i>E</i>	-	-	-	1	1	-	<i>1</i>
<i>value</i>							

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
<i>B</i>	1	1	1	1	-	-
<i>D</i>	1	1	1	-	-	-
<i>C</i>	1	-	-	1	1	1
<i>A</i>	-	-	-	-	1	1
<i>E</i>	-	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 \Rightarrow **no guarantee that „optimal“ ordering is obtained.**

Sometimes a 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
<i>B</i>	1	1	1	1	-	-	60
<i>D</i>	1	1	1	-	-	-	56
<i>C</i>	1	-	-	1	1	1	39
<i>E</i>	-	1	-	-	1	-	3
<i>A</i>	-	-	-	-	1	1	18

<i>value</i>	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 more 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
<i>A</i>	0.3	-	-	-	0.6	-	-	0.9	1
<i>B</i>	-	0.3	-	0.3	-	-	0.1		
<i>C</i>	0.4	-	-	0.5	-	0.3	-		
<i>D</i>	0.2	-	0.4	-	0.3	-	0.5		
<i>E</i>	-	0.4	-	-	-	0.5	-		
<i>F</i>	-	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:

$$x_{ij} = 1, \text{ if part } j \text{ is assigned to group } i \text{ (and } = 0, \text{ otherwise)}$$

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

The objective is the total number of machines used:

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

subject to the constraints:

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

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

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

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

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

The optimal solution can be computed using some standard LP solvers. In the simple **example above**, this can be done 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:

machine	part							sum	min. #	Single-Pass Heur.	opt
	1	2	3	4	5	6	7				
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*.

machine	parts								n_i	
	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:

machine	n_{ij}							
	1	2	3	4	5	6	7	8
A								

<i>B</i>								
<i>C</i>								
<i>D</i>								
<i>E</i>								
<i>F</i>								

This gives the similarity coefficients:

	s_{ij}						parts
machine	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>	<i>F</i>	
<i>A</i>	1	1	0,33	0	0	0	
<i>B</i>	1	1	0,33	0	0	0	
<i>C</i>	0,33	0,33	1	0,75	0	0	
<i>D</i>	0	0	0,75	1	0,5	0,5	
<i>E</i>	0	0	0	0,5	1	1	
<i>F</i>	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 of 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 and D form their own single machine groups.

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

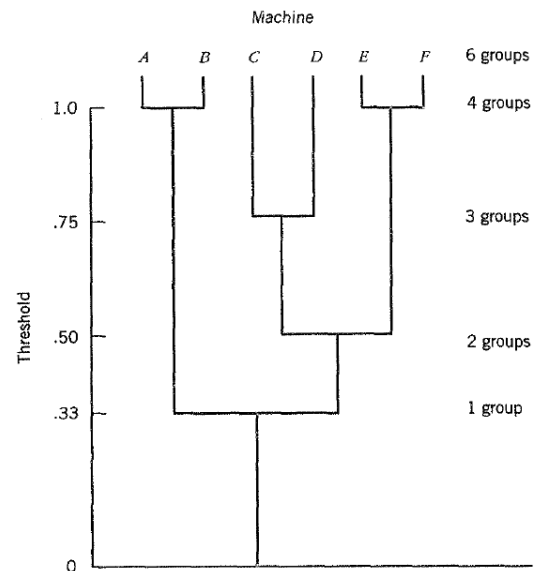
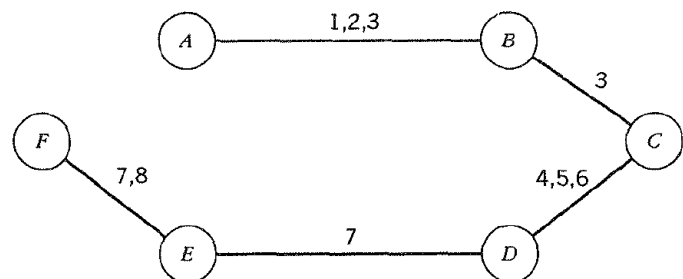


Figure 3.7. Dendrogram for a hierarchical clustering (Askin and Standridge).

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 all 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 $\Rightarrow 5 \times 10^{28}$ 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 pass means exchanging each vertex $A \in X$ with each vertex $B \in Y$ exactly once:
 1. For $i := 1$ to n do

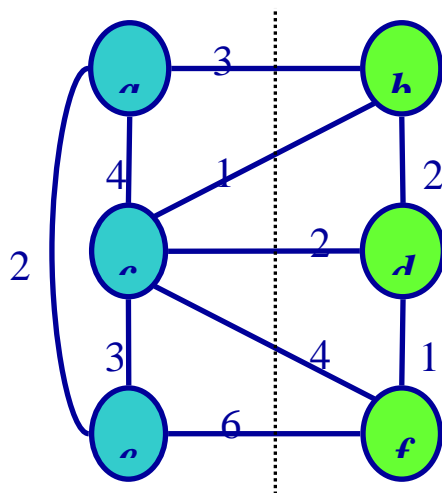
From all unlocked (unexchanged) vertices,

choose a pair (A, B) such that the $\text{gain}(A, B)$ is largest.

Exchange A and B . Lock A and B .

Let $g_i = \text{gain}(A, B)$. (can also be negative)
 2. Find the k s.t. $G = g_1 + \dots + g_k$ 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:

$$\text{cut-size} = 3 + 1 + 2 + 4 + 6 = 16$$

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

For each node $x \in \{ a, b, c, d, e, f \}$. 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 \Rightarrow 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):

$$g_{ab} = G_a + G_b - 2w_{ab} = -3 + 2 - 2 \cdot 3 = -7$$

$$g_{ad} = G_a + G_d - 2w_{ad} = -3 - 1 - 2 \cdot 0 = -4$$

$$g_{af} = G_a + G_f - 2w_{af} = -3 + 9 - 2 \cdot 0 = +6$$

$$g_{cb} = G_c + G_b - 2w_{cb} = 0 + 2 - 2 \cdot 1 = 0$$

$$g_{cd} = G_c + G_d - 2w_{cd} = 0 - 1 - 2 \cdot 2 = -5$$

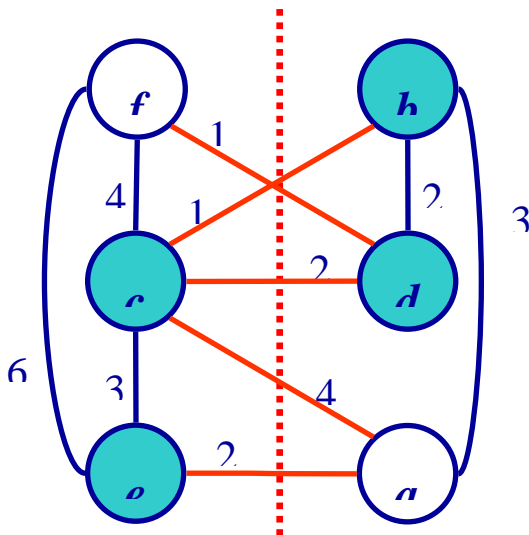
$$g_{cf} = G_c + G_f - 2w_{cf} = 0 + 9 - 2 \cdot 4 = +1$$

$$g_{eb} = G_e + G_b - 2w_{eb} = +1 + 2 - 2 \cdot 0 = +1$$

$$g_{ed} = G_e + G_d - 2w_{ed} = +1 - 1 - 2 \cdot 0 = 0$$

$$g_{ef} = G_e + G_f - 2w_{ef} = +1 + 9 - 2 \cdot 6 = -2$$

The maximum gain is obtained by exchanging nodes a and $f \Rightarrow$ **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:

$$X' = \{ c, e \}$$

$$Y' = \{ b, d \}$$

Update the G-values of unlocked nodes

$$G'_c = G_c + 2c_{ca} - 2c_{cf} = 0 + 2(4 - 4) = 0$$

$$G'_e = G_e + 2c_{ea} - 2c_{ef} = 1 + 2(2 - 6) = -7$$

$$G'_b = G_b + 2c_{bf} - 2c_{ba} = 2 + 2(0 - 3) = -4$$

$$G'_d = G_d + 2c_{df} - 2c_{da} = -1 + 2(1 - 0) = 1$$

Compute the gains:

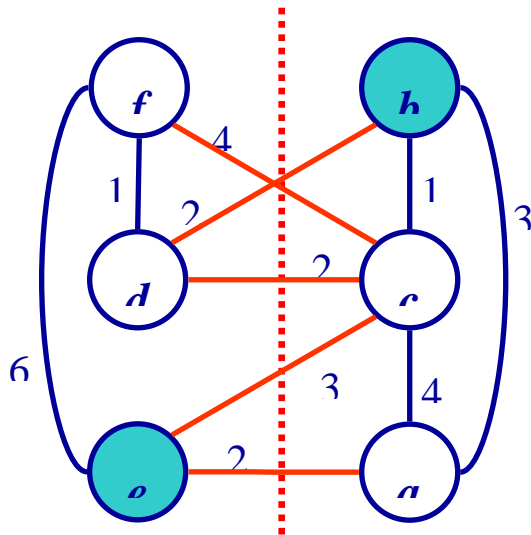
$$g_{cb} = G_c + G_b - 2w_{cb} =$$

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

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

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

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:

$$X' = \{ e \}$$

$$Y' = \{ b \}$$

Update the G-values of unlocked nodes

$$G'_e = G_e + 2c_{ed} - 2c_{ec} =$$

$$G'_b = G_b + 2c_{bd} - 2c_{bc} =$$

Compute the gains:

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

Summary of the Gains...

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

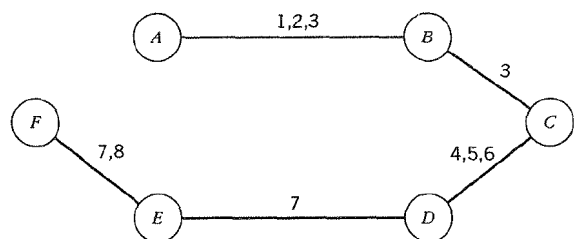
Maximum gain is $g_1 = +6 \Rightarrow$ Exchange only nodes a and f . End of 1 pass.

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:

machine	parts							
	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

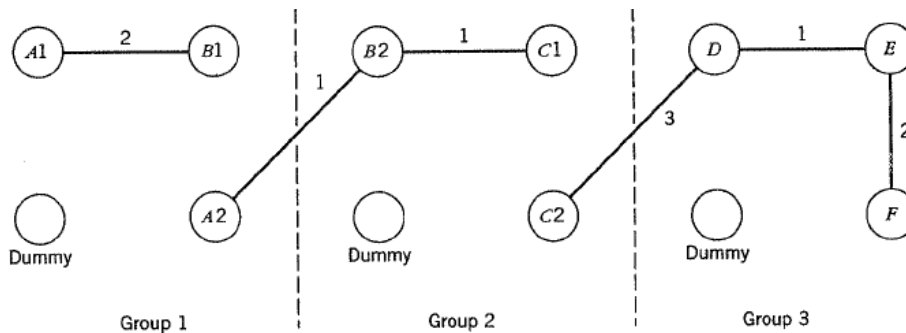


<i>F</i>	1	1
----------	---	---

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
<i>A1</i>	1	1						
<i>B1</i>	1	1						
<i>A2</i>			1					
<i>B2</i>			1					
<i>C1</i>			1					
<i>C2</i>				1	1	1		
<i>D</i>				1	1	1	1	
<i>E</i>							1	1
<i>F</i>							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).





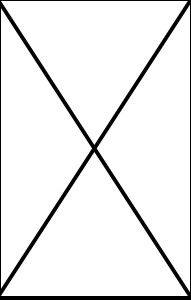
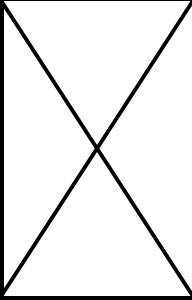

We start with optimizing the partition Group 1 = {A1, A2, B1, Dummy1} and Group 2 = {B2, C1, C2, Dummy2} while we keep Group 3 = {D, E, F, Dummy3} unchanged for the moment.

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	X	

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 :

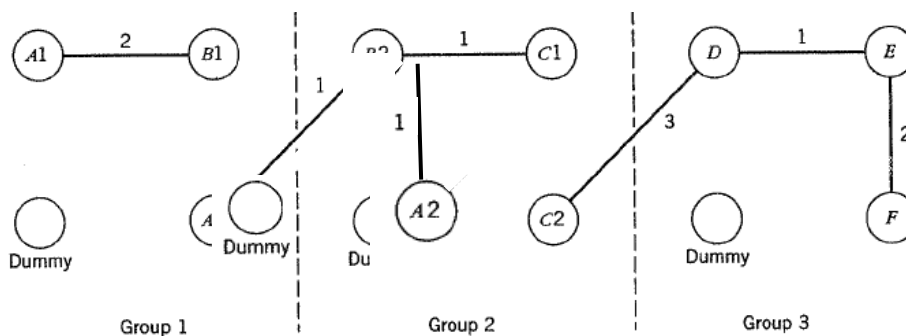
$$G_i' = G_i + 2c_{iA2} - 2c_{iDummy2} \text{ in Group 1 and } G_j^{new} = G_j + 2c_{iDummy2} - 2c_{jA2} \text{ in Group 2.}$$

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

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 (of approximately 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:

		parts								
		1	2	3	4	5	6	7	8	
(no duplication of machines)	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 \Rightarrow 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 \Rightarrow subgroup cannot be further subdivided.

Machine D is used only for part 7 \Rightarrow 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 \Rightarrow 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 \Rightarrow exceptional \Rightarrow 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:

machine	parts							
	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 are 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.

4. Exact Methods for Assembly Line Balancing

We have seen in the beginning of this chapter, that an Assembly Line Balancing (ALB) problem can be represented as a binary LP. Smaller instances can be simply solved by using a general purpose LP-solver. For very large instances of this np-hard problem, heuristics need to be used - see the previous sections.

Since ALB problems are tactical problems that are solved only now and then, the results need not be available very soon and computation time can in principle be quite long.

Hence, a number of tailored exact methods have been developed for ALB problems. The most well known ones are based on *Dynamic Programming* (DP) and *Branch & Bound* (B&B). In the next subsections we present two such algorithms for Alternative 1, i.e. where the cycle time is given and the number of stations has to be minimized.

Jackson Algorithm (Dynamic Programming, Decision Tree)

This was the first and simplest exact method that was specially designed for ALB problems. Later improved algorithms have been suggested but the dominance rules are still of general relevance.

Construction of a Decision Tree

The individual stations of the assembly-line are considered one by one.

In the *first stage* one generates all possibilities for the allocation of the first station, where one considers only *maximal stations* (i.e. no additional operations can be added). Hence, one obtains a number of different states, which are described by the operations already assigned to station 1.

Step from stage k-1 to stage k:

The state in stage *k-1* represents all operations already assigned to stations 1 to *k-1* (not only *k*).

In stage *k*, for each such state in stage *k-1*, one forms all maximal stations *k* and obtains the corresponding states in stage *k*.

As soon as a state is reached where all operations have been assigned, the optimal solution is reached and *k* is the minimal number of stations.

As usual in DP, the allocations of the individual stations can be determined by backtracking.

The problem can also be considered as a shortest path problem with nodes being the states and the edges representing the allocations of the stations. The starting node is the empty set and the terminal node represents the situation where all operations are assigned.

Jackson Algorithm

Given:

c ... cycle time

A = $\{1, \dots, n\}$... set of all operations with

t_j ... durations $t_j \leq c$;

Precedence graph (i.e. set of all immediate predecessors $V(j)$ or successors $N(j)$)

Notation used:

k ... Stage (station number)
 Z_k ... state in stage k ; set of all operations that have already been assigned in stages/stations 1 to $k-1$, i.e.. $Z_k \subseteq A$
 L_1 ... list of all states in stage $k-1$
 L_2 ... list of states in stage k
 \mathcal{E}_k ... set of possible alternative assignments to station k
 S_k ... current assignment to station k in stage k

Start: $L_1 := \langle \{ \} \rangle$; *(empty set - nothing assigned yet)*

Iteration $k = 1, 2, \dots$:

$L_2 := \langle \rangle$; ... *(start with an empty station)*

while $L_1 \neq \langle \rangle$ **do** *(as long as not all states of stage $k-1$ have been considered)*

begin

choose and remove the first element Z_{k-1} of L_1 :

construct the set \mathcal{E}_k of all possible allocations of station k :

$$\mathcal{E}_k := \left\{ S_k \mid \left(S_k \subseteq A - Z_{k-1} \right) \wedge \left(\forall j \in S_k \text{ gilt } v(j) \subseteq (Z_{k-1} \cup S_k) \right) \wedge \left(\sum_{j \in S_k} t_j \leq c \right) \right\};$$

(i.e. all subsets of the set of not yet assigned operations $A - Z_k$, such that all predecessors are already assigned and total workload does not exceed cycle time)

eliminate non maximal assignments: *(dominance rule 1)*

$$\mathcal{E}_k := \mathcal{E}_k \setminus \left\{ S_k \mid \left(\exists S'_k \in \mathcal{E}_k \text{ mit } S_k \subset S'_k \right) \right\};$$

while $\mathcal{E}_k \neq \{ \}$ **do** *(add the new stations k to the states in list L_2)*

begin

select and remove an element S_k of the set \mathcal{E}_k ;

$Z_k := Z_{k-1} \cup S_k$; *(add S_k to the previous state Z_{k-1})*

add Z_k to list L_2 ;

if $Z_k = A$ **then begin** $m := k$; **stop end**; *(all operations assigned)*

end;

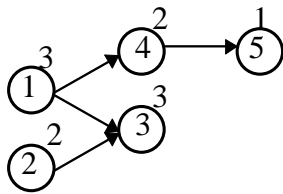
end;

$L_1 := L_2$;

Result: optimal assignment with m stations found.

Example: $c = 4$

precedence graph

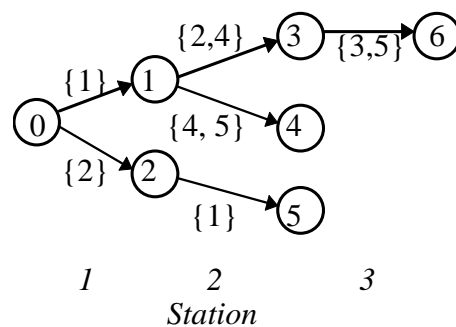


A possible decision tree is indicated below.

The columns represent the stages, the nodes correspond to the possible states, the arrows correspond to the possible station allocations,

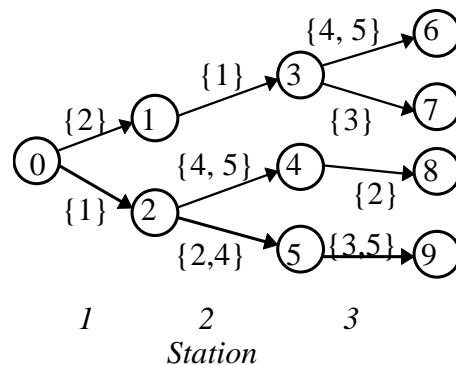
The numbers in the nodes indicate a possible sequence in which these states are generated (sequence is arbitrary within a stage).

If the operations are considered in sequence 1, 2, 3, 4, and 5 the following optimal solution is obtained:



If the operations are considered in the opposite sequence (5, 4, 3, 2, 1), one obtains the following decision tree with the first optimal solution on node 9,

i.e. it depends on the sequence when the optimal solution is found in the last stage. The states in the previous stages are however not affected by the sequence.



Dominance rules

Clearly, the decision tree can become very large in case of many operations.

Hence, one tries to reduce the size of the tree by deleting some of the branches as soon as possible.

Since (usually) just one optimal solution is required, all states and stations can be ignored that are dominated by some other station with the same starting state Z_{k-1} .

A state or station is dominated by another one, if the former cannot lead to a better solution than the latter.

The first dominance rule we have already considered in the algorithm:

Dominance rule 1: station assignment S_k with starting state Z_{k-1} is dominated by station assignment S'_k with the same starting state, if $S_k \subset S'_k$.

Example: In the above example in stage 2 the station assignments $S_2 = \{2\}$ and $S_2 = \{4\}$ are dominated by $S'_2 = \{2, 4\}$.

For the next dominance rules we need the following definition:

Für weitere Dominanzregeln definieren wir Nachfolgermengen von Knotenmengen J wie folgt:

$$N(J) = \bigcup_{j \in J} N(j) - J \quad \dots \text{ set of all immediate successors of all operations in set } J.$$

With this, we can formulate:

Dominance rule 2: station assignment S_k with starting state Z_{k-1} is dominated by station assignment S'_k with the same starting state, if the following holds:

$$\sum_{j \in J_1} t_j \leq \sum_{j \in J_2} t_j \quad \text{and} \quad N(J_1) \subseteq N(J_2)$$

where

$$J_1 = S_k - S'_k \quad \text{and} \quad J_2 = S'_k - S_k$$

Because of the first condition, station S'_k has more workload assigned (less idle time).

The second condition guarantees that all operations that depend on J_1 also depend on J_2 . This means, that all successors of J_1 are only available, if all operations in J_1 and J_2 have been assigned.

Choosing station assignment S'_k instead of S_k leads to a station that has not more idle time and represents not more restrictions for the planning in the subsequent stages.

The application of this rule can be time consuming. Hence, it is sometimes only applied in case of

$$|J_1| = |J_2| = 1.$$

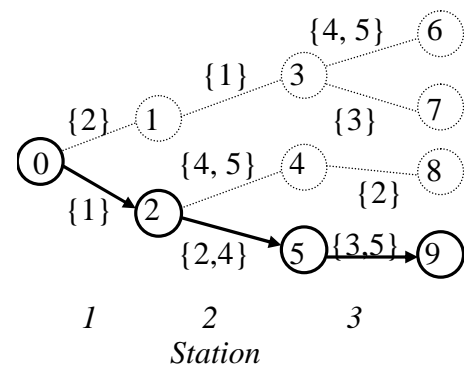
It is possible that two station assignments dominate each other. In this case one of them can be dropped while the other must be kept.

Example above: Because of dominance rule 2 station $S_1 = \{2\}$ is dominated by $S'_1 = \{1\}$ in stage 1, since

- S'_1 has more workload assigned (less idle time) than S_1 , $t_2 < t_1$ and
- $N(S_1 - S'_1) = N(\{2\}) = \{3\}$
 $N(S'_1 - S_1) = N(\{1\}) = \{3, 4\}$,
 i.e. $N(S_1 - S'_1) \subseteq N(S'_1 - S_1)$

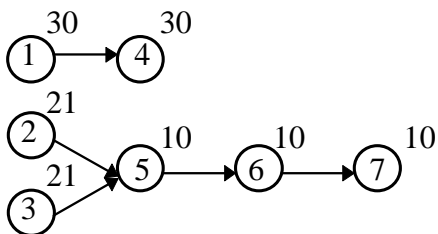
Hence the partial tree starting in node 1 can be eliminated.

In the same way, in stage 2 and $Z_1 = \{2\}$ the possible station assignment $S_2 = \{4, 5\}$ is dominated by $S'_2 = \{2, 4\}$.



Remark: The following example shows, that condition $N(J_1) \subseteq N(J_2)$ is actually needed and that a better workload alone does *not* guarantee dominance:

Example: $c = 40$



Although $t_1 \geq t_2$ and $t_1 \geq t_3$, the stations $S_1 = \{2\}$ and $S_1 = \{3\}$ are *not* dominated by $S_1 = \{1\}$.

This is because $J_1 = \{2\}$ and $J_2 = \{1\}$ so that $N(J_1) = \{5\}$ is *not* contained in $N(J_2) = \{4\}$.

The optimal solution is $S_1 = \{2\}, S_2 = \{3, 5\}, S_3 = \{1, 6\}, S_4 = \{4, 7\}$

It is only reached if $S_1 = \{2\}$ is chosen in the first stage. All other states in stage 1 yield a solution with 5 stations.

The next dominance rule extends dominance rule 1 from *stage k* (operations assigned in stage *k*) to *state k* (set of all operations assigned in stages 1 to *k*):

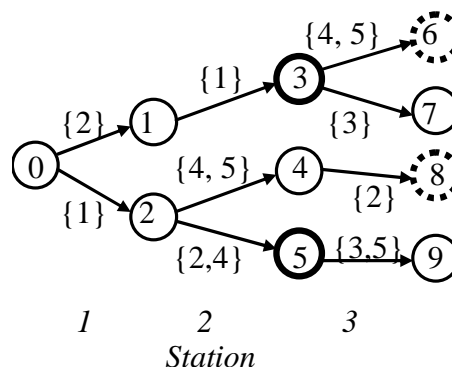
Dominance rule 3: A state Z_k is dominated by state Z'_k in the same stage *k*, if $Z_k \subseteq Z'_k$.

Example: In the above example state 3 represents the (assigned) operations $\{1,2\}$ while state 5 represents operations $\{1, 2, 4\}$.

Because of $\{1, 2\} \subset \{1, 2, 4\}$ state 3 is dominated by state 5.

If with 2 stations already operations 1, 2, and 4 can be assigned, then it makes no sense to keep a state where with 2 stations only operations 1 and 2 are assigned.

States 6 und 8 are identical, because they both represent the operations $\{1, 2, 4, 5\}$. One of them could be deleted.

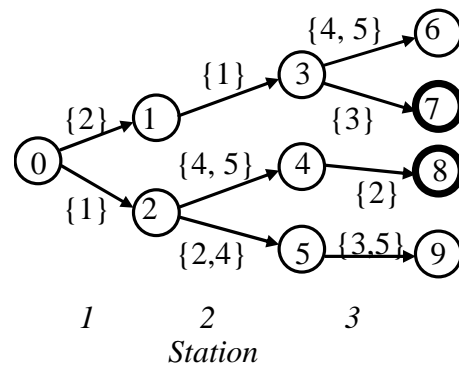


The next dominance rule extends dominance rule 3 from *stage k* (operations assigned in stage *k*) to *state k* (set of all operations assigned in stages 1 to *k*):

Dominance rule 4: A state Z_k is dominated by state Z'_k , if for $J_1 = Z_k - Z'_k$ and $J_2 = Z'_k - Z_k$ holds:

$$\sum_{j \in J_1} t_j \leq \sum_{j \in J_2} t_j \text{ and } N(J_1) \subseteq N(J_2)$$

Example: In the above example states 7 and 8 dominate each other and one of them could be deleted.



Rules 2 and 4 can be quite time consuming and it is not always clear whether they lead to a reduction in computation time.

Pinto Heuristic

As already mentioned, the ALB problem can be considered as a shortest path problem. We have seen that the complete graph need not be developed since one can stop as soon as in one node all operations have been assigned, and also because of pruning the tree by dominance rules.

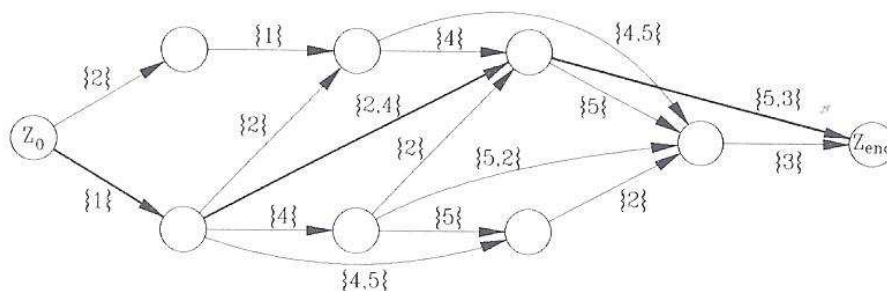
However, the graph/tree will still be very large. Therefore a heuristic has been developed that is based on this shortest path problem but only considers a subgraph (at the cost of losing the guarantee of optimality).

Heuristic by Pinto

1. Find some good (and feasible w.r.t. precedence) orderings of the operations using e.g. different priority rules
2. For each of these orderings (permutations) (j_1, \dots, j_n) of operations, define nodes (states) $Z_0 = \{\}, \{j_1\}, \{j_1, j_2\}, \dots, Z_{end} = \{j_1, \dots, j_n\}$.
3. Draw an arrow from node Z to Z' if $Z' - Z$ represents a feasible assignment of a station in the sense that cycle time is not exceeded: $\sum_{j \in Z'-Z} t_j \leq c$
4. In the resulting graph find the shortest path from $Z_0 = \{\}$ to $Z_{end} = \{j_1, \dots, j_n\}$.

Often this heuristic finds improved solutions compared to the application of simple priority rules. However there is *no* guarantee that the optimal solution is found.

Example: Reconsider the above example and choose the two orderings (2, 1, 4, 5, 3) and (1, 4, 5, 2, 3). With $c = 4$ one obtains the following graph:



The shortest path (minimum number of arrows) is shown in bold. By coincidence the optimal solution is reached.

The B&B algorithm by Johnson von (FABLE)

The above DP algorithm can be considered a "breadth" search in the sense that all nodes in a certain stage are considered, before proceeding to the next stage. This way, the first feasible (complete) solution is already the optimal one. If the algorithm is stopped because of time restrictions no feasible solution is available.

The B&B algorithm by Johnson tries to search the corresponding tree in the sense of "depth" search by trying to reach leaves of the search tree (complete solution) soon. It is also known as *FABLE* (*Fast Algorithm for Balancing Lines Effectively*).

Like in all B&B algorithms it is important to keep the tree small by appropriate pruning. In addition to the above dominance rules, also bounds are used.

First we describe the branching process, and then we will discuss the different ways of pruning the tree.

Branching Process

In the starting node 0 no operations have been assigned yet.

In each iteration an additional operation is assigned (or in a backtracking step an operation is removed).

A new station is opened whenever no further operation can be assigned to the previous one (because of cycle time and precedence). Hence, we again only consider maximal stations (compare dominance rule 1)

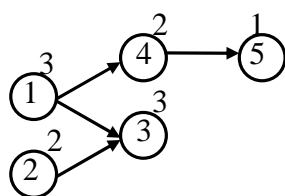
In B&B, there are always 2 possibilities last bound (the last node is extended) or best bound (the most promising node is extended). Here the last bound approach is chosen, i.e. in a kind of LIFO-strategy always the last generated subset is investigated further.

In order to obtain a good first solution, the operations are ordered according to some priority rules. In FABLE, the sorting is done (considering precedence relations) by the rules:

1. sort according to decreasing operation times t_j (i.e. allocate long operations first)
2. in case of a tie, use decreasing number of immediate successors
3. in case there is still a tie choose randomly

Example:

Cycle time $c = 4$



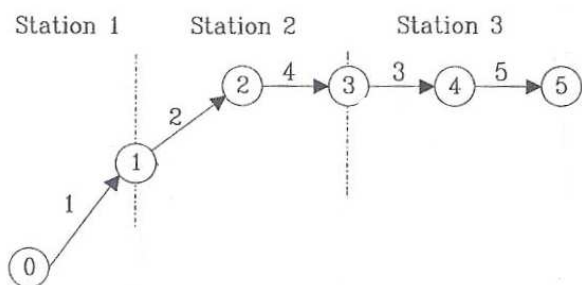
The graph on the left is the precedence graph. The nodes are the operations.

This gives the ordering (1, 2, 3, 4, 5):

The first two candidates (ready to be assigned) are 1 and 2. Because of rule 1 we select 1.

Then we could assign 2 or 4. Rules 1 and 2 do not help, so we select 2 by rule 3, etc.

The algorithm starts like a normal priority rule method, i.e. the operations are assigned in the selected ordering. If some operation cannot be assigned anymore to a station, because the cycle time is exceeded, one tries to insert the next operations in the list.



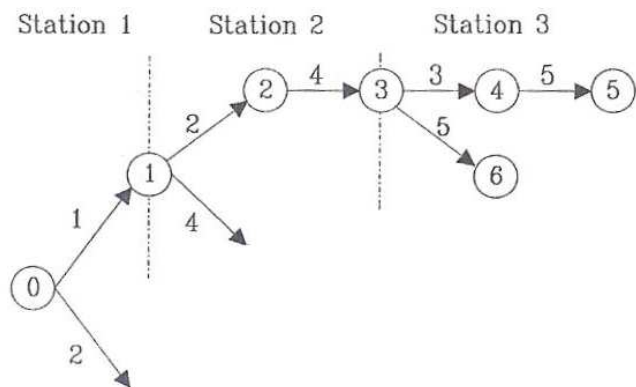
First, station 1 is built as {1}. No other operation can be assigned anymore.

In the next station we first insert operation 2. Then only 4 is ready to be assigned (3 would not fit!). So station 2 is completed.

Finally 3 and 5 enter station 3.

We have found a "leaf" of the B&B tree:

Feasible solution (3 stations): {1}, {2, 4}, {3, 5}



If we would use just a priority rule, we were happy now and would stop (or add an improvement heuristic).

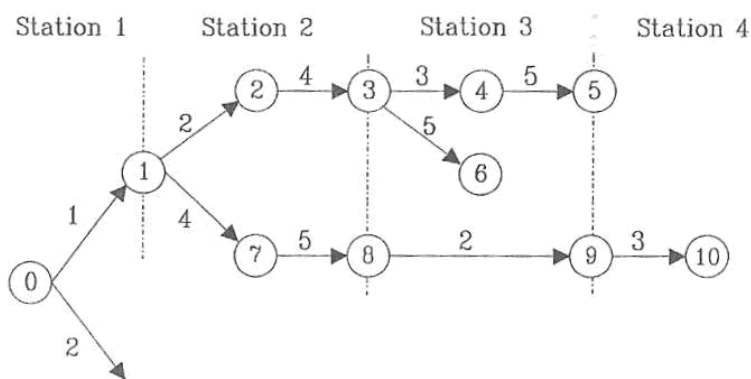
In an exact method we must check all alternative solutions in principle. Hence, one has to perform backtracking. We have to go back in the tree to find the last "crossroad" where not all directions /alternatives have been investigated yet.

On the left we see all these "branching" opportunities.

We go back to node 3, where instead of choosing operation 3 (by the heuristic rule) we could have also chosen operation 5.

By starting station 3 with node 5, the next selection would be node 3 and we would obtain the same station {3, 5}. In FABLE, this duplicate effort is avoided by permitting only increasing operation numbers within a station. I.e. after 5, operation 3 cannot be assigned anymore to the same station. Hence we do not consider node 6 any further, but proceed by further backtracking.

The next branching opportunity is in node 1 where instead of operation 2 one could also have chosen operation 4.

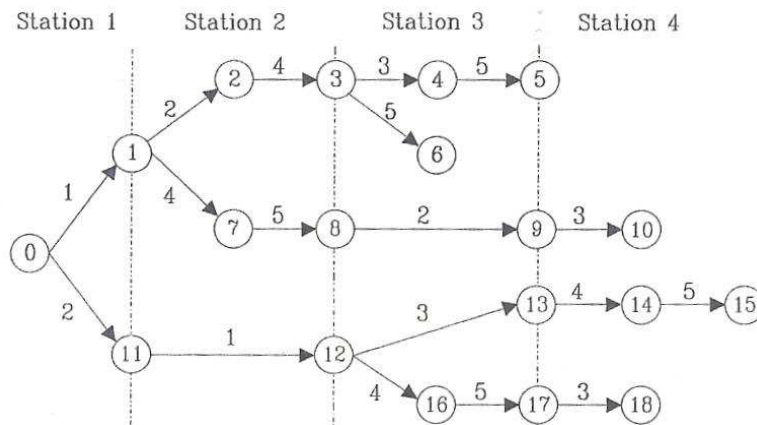


Again, in order to avoid double consideration of the station configuration {2, 4}, operation 2 after operation 4 in station 2 is not considered - within a station operations are only added with increasing numbers. Hence, the only possibility for station 2 is {4, 5}.

Then this station is maximal and one proceeds to station 3. The only possibility here is {3}.

Station 4 then contains only operation 3. However, already before (at node 9) one could have stopped, since it was clear that this branch leads to more than 3 stations and already one solution with 3 stations is known. Hence this branch cannot lead to an optimal solution and backtracking could be started already in node 9 without constructing node 10.

In fact, already in node 8 one could start backtracking, since it is clear that one needs at least 3 stations in this branch and - as mentioned - already one solution with 3 stations is known. This branch cannot lead to an improvement.



Finally, backtracking to the first "crossroads" one reaches branching node 0 and chooses operation 2 as the first one. This complete the maximal station 1 as {2}.

In the next step the only maximal station is {1}.

In node 12 one has again 2 possible operations to be chosen, 3 and 4. We start with the lower index, i.e. operation 3. Station {3} is maximal.

In node 13 or even in node 12 one could have already stopped, since it is clear that no solution with less than 3 stations will be found. All branches have been investigated and the algorithm stops.

The solution {1}, {2, 4}, {3, 5} with 3 stations is optimal.

For didactical reasons, in the above graph also the branching after node 12 is shown, even if it is not necessary.

Algorithmic Description of the Branching Process

Given: cycle time c ; n operations with durations $t_j \leq c$ ($j = 1, \dots, n$); precedence graph; ordering of operations according to priority rule (an arrow can only lead from a node to another node with a higher index number).

Notation used:

k current station number

p number of already assigned operations

$A[1..p]$ operations that are ready to be assigned (in stage p of the B&B tree)

$\bar{c}[1..k]$ idle time of the current station $1, \dots, k$

$|a|$ last operation considered: positive a means that a has been added; negative a means that $|a|$ was removed in the last backtracking step

K Set of operations, that are candidates to be assigned in the current station (observing precedence in this station)

An operation j is *ready to be assigned*, if j has not yet been assigned, all predecessors have been assigned, and its duration does not exceed the remaining cycle time in the current station.

Algorithm

Start: $k := 1$; $\bar{c}[1] := c$; $p := 0$; $K := \{1\}$

(Arbeitsgang 1 wird als erster eingeplant)

Iteration:

repeat

if $K \neq \emptyset$ **then** (candidates for assignment in station k ?)
begin $a := \min \{ j \in K \}$; *assign operation a* **end** (step forward)
else **if** $a < 0$ **or** an operation can be assigned **then**
eliminate operation A[p] (backtracking step)
else
begin $k := k + 1$; $\bar{c}[k] := c$; $a := 0$ **end**; (open new station)
 $K := \{ j \mid j \text{ can be assigned} \wedge j > |a| \}$ (set of candidates for station k)
until $p = 0$ **and** $K = \emptyset$; (no more branching possible?)

Result: The best found solution is optimal.

In this algorithm we use:

Procedure: *eliminate operation A[p];*

begin

if $\bar{c}[k] = c$ **then** $k := k - 1$; (delete empty station)
 $a := -A[p]$; (from the current station delete
 $\bar{c}[k] := \bar{c}[k] + t_{A[p]}$; $p := p - 1$ that operation that was added last)

end;

Procedure: *assign operation a;*

begin

$p := p+1; A[p] := a; \bar{c}[k] := \bar{c}[k] - t_{A[p]};$

if $p = n$ **then**

begin new feasible solution found; save it if it is the currently best one;

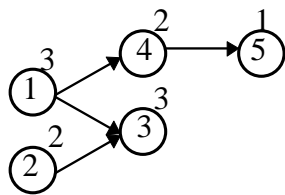
eliminate operation A[p]

end;

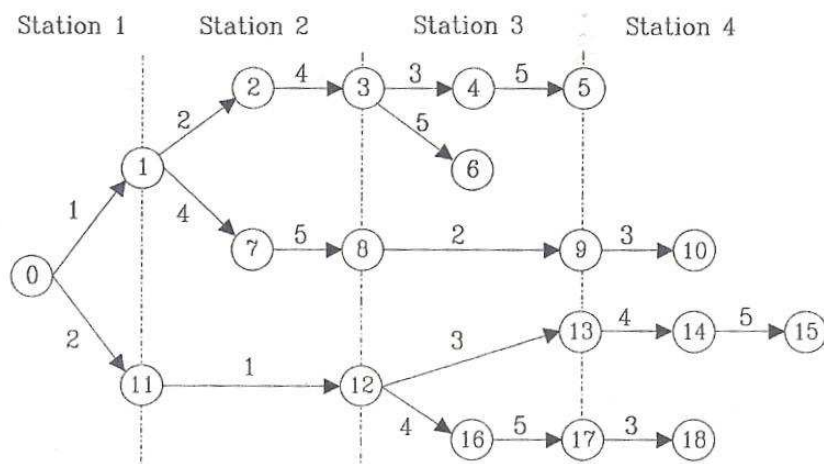
end

Above
example:

Cycle time $c =$
4



We illustrate some steps :



Start: $k := 1; \bar{c}[1] := 4; K := \{1\}$

It. 1: $a := 1$; assign operation 1 (node 1); $K := \emptyset$

It. 2: no operation can be assigned; *open station* $k = 2$; $K := \{2,4\}$

It. 3: $a := 2$; assign operation 2 (node 2); $K := \{4\}$

It. 4: $a := 4$; assign operation 4 (node 3); $K := \emptyset$

It. 5: no operation can be assigned; *open station* $k = 3$; $K := \{3,5\}$

It. 6: $a := 3$; assign operation 3 (node 4); $K := \{5\}$

It. 7: $a := 5$; assign operation 5 (node 5); **feasible solution with 3 stations:**

$S_1 = \{1\}, S_2 = \{2,4\}; S_3 = \{3,5\}$; save; eliminate operation 5;

$a := -5$ (back to node 4); $K := \emptyset$

- It. 8:** $a < 0$; eliminate operation 3; $a := 3$ (back to node 3); $K := \{5\}$
- It. 9:** $a := 5$; assign operation 5 (node 6); $K := \emptyset$
- It. 10:** operation 3 can be assigned; eliminate operation 5; $a := -5$ (back to node 3); $K := \emptyset$ (although 3 could be assigned, we do backtracking, since combination $\{3,5\}$ in station 3 was already considered)
- It. 11:** operations 3 and 5 could be assigned; eliminate op. 4; $a := -4$ (back to node 2); $K := \emptyset$
- It. 12:** operation 4 can be assigned; eliminate operation 2; $a := -2$ (back to node 1); $K := \{4\}$
- It. 13:** $a := 4$; assign operation 4 (node 7); $K := \{5\}$
- It. 14:** $a := 5$; assign operation 5 (node 8); $K := \emptyset$
- It. 15:** no operation can be assigned; *open station* $k = 3$; $K := \{2\}$
- It. 16:** $a := 2$; assign operation 2 (node 9); $K := \emptyset$
- It. 17:** no operation can be assigned; *open station* 4; $K := \{3\}$
- It. 18:** $a := 3$; assign operation 3 (node 10); **feasible solution with 4 stations: $S_1 = \{1\}$; $S_2 = \{4,5\}$; $S_3 = \{2\}$; $S_4 = \{3\}$** ; eliminate operation 3; $a := -3$ (back to node 9); $K := \emptyset$
- etc.

Fathoming of subproblems/branches in FABLE:

It is important to find ways to remove (delete, fathom) as many branches of the B&B tree in order to speed up the process. For that, one can use dominance rules (just like in DP) as well as bounds on the objective.

Here, we use *Dominance Rules 1 and 2*. Dominance rule 1 (maximal stations) is implicitly used in the algorithm; rule 2 must be slightly reformulated compared to FABLE.

Dominance Rule 2': A node of the tree can be deleted, if for the current maximal station S_k holds:

There exists an operation $h \in S_k$ and an operation j that has not yet been assigned such that $N(h) \subseteq N(j)$ and $t_h \leq t_j$ and $\bar{c}[k] - t_j + t_h \geq 0$.

If $t_h = t_j$ and $N(h) = N(j)$ gilt, also $j > h$ is required.

In the example node 8 is dominated by node 3, since there station 2 has more workload assigned and $N(5) \subseteq N(2)$ gilt.

Also node 11 is dominated by node 1.

Dominance Rule 2' can be applied in 2 steps. In a pre-processing step (before branching) all pairs of operations (h, j) are stored, that satisfy the first two conditions (potential dominance).

Just before opening a new station one checks for each operation h whether it is dominated by another operation $j \in K$ that is ready to be assigned. In this case we delete the node and start backtracking..

FABLE uses two more dominance rules:

First Station Dominance Rule: If the current maximal station is a subset of (or identical to) an alternative for station 1 that has already been constructed and stored, then this node can be deleted.

Example: node 12 can be deleted because of node 1.

FABLE also uses some **Labelling Dominance Rule**, which we do not present here.

The use of Bounds:

Nodes can also be deleted, if lower bounds for the remaining number of stations indicate that the currently best solution cannot be improved in this branch.

With J denoting the set of operations that have not yet been assigned (in Fable: $J = A$):

$$LB_1(J) = \sum_{j \in J} t_j / c$$

$$LB_2(J) := \left\lceil \left\{ j \in J \mid t_j > \frac{c}{2} \right\} \right\rceil + \frac{1}{2} \cdot \left\lceil \left\{ j \in J \mid t_j = \frac{c}{2} \right\} \right\rceil ;$$

$$LB_3(J) := \left\lceil \left\{ j \in J \mid t_j > \frac{2}{3}c \right\} \right\rceil + \frac{2}{3} \cdot \left\lceil \left\{ j \in J \mid t_j = \frac{2}{3}c \right\} \right\rceil + \frac{1}{2} \cdot \left\lceil \left\{ j \in J \mid \frac{1}{3}c < t_j < \frac{2}{3}c \right\} \right\rceil + \frac{1}{3} \cdot \left\lceil \left\{ j \in J \mid t_j = \frac{1}{3}c \right\} \right\rceil ;$$

Clearly, only an integer number of station makes sense. Hence the above bounds can be rounded up to the next integer.

- $LB_1(J)$ corresponds to the minimum number of stations m_{\min} from the beginning of Chapter 4.
- $LB_2(J)$ counts operations with durations exceeding half of the cycle time. Out of these only one can be in a station since two would not fit. Operations with duration $c/2$ need exactly half a station, so that two such operations can be in a station. All shorter operations are ignored.
- $LB_3(J)$ is a generalization of $LB_2(J)$ to thirds of the cycle time.

Example from the very beginning of this chapter:

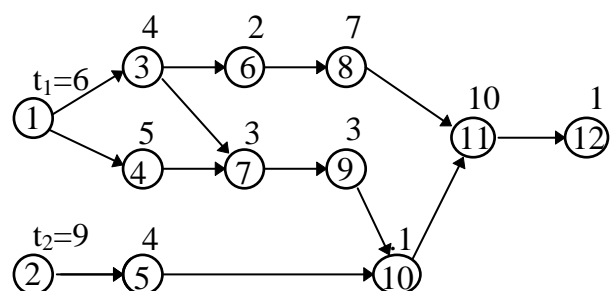
Cycle time now $c = 10$.

We compute bounds $LB_1(A)$ to $LB_3(A)$ for the artificial node P_0 (before 1 and 2):

$$LB_1(A) = 55/10 = 5.5 \quad \text{since } \sum t_j = 55.$$

$$LB_2(A) = \left\lceil \{1, 2, 8, 11\} \right\rceil + \frac{1}{2} \cdot \left\lceil \{4\} \right\rceil = 4.5$$

$$LB_3(A) = \left\lceil \{2, 8, 11\} \right\rceil + \frac{1}{2} \cdot \left\lceil \{1, 3, 4, 5\} \right\rceil = 5$$



Rounding up, we get $LB_1(A) = 6$ and $LB_2(A) = LB_3(A) = 5$. Hence we need at least 6 stations.

Extensions for mixed model assembly

There exists versions of the DP and of the B&B algorithm for mixed model assembly.

The idea is to compute ALL optimal solutions (with minimal number of stations). Out of these the ones where the workload is distributed best (cf. Thomopoulos) is chosen.

In order to find all optimal solutions (not just one) all nodes that can also lead to a solution with the same number of stations must be kept. Above we only kept those where an improvement was possible. Hence the tree becomes larger and the computation times will increase.