Modul

**Management Science**

**(for QEM)**

Prof. Dr. Richard F. Hartl

SS 2016

****

© Produktion und Logistik

# Flow Shop - Assembly Line Balancing

# Possible Layouts of Production Systems[[1]](#footnote-1),[[2]](#footnote-2)

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 a 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 - *similar machines are arranged in "work shops*"
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.

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 for coming up with the needed production output.

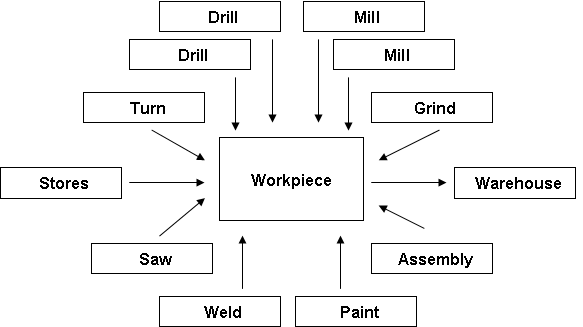


Figure 2‑1: Fixed-position layout

1. 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.

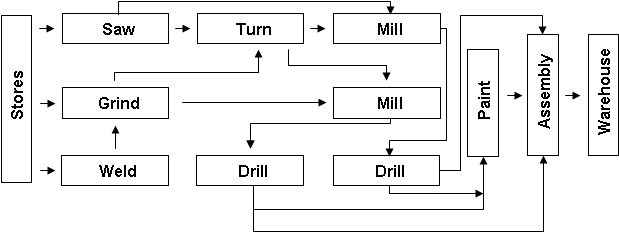


Figure 2‑2: Job shop production

1. Figure 2‑3 illustrates a cellular manufacturing system for the example problem:   
   2 cells for 2 product groups.   
   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.

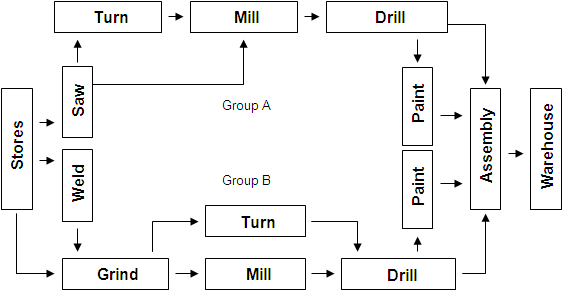


Figure 2‑3: Cellular manufacturing system

1. Figure 2‑4 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):

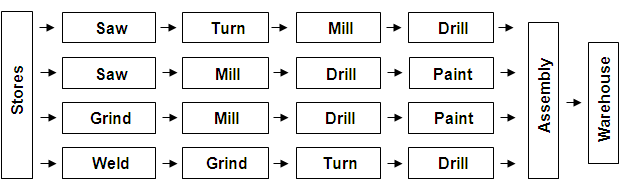


Figure 2‑4: Flow shop production

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 2‑5 illustrates a volume-variety chart[[3]](#footnote-3).



Figure 2‑5: 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.

# Complexity

Almost all optimization problems occuring 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 applyinga 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 can use “general purpose”-heuristics or metaheuristics (e.g. Simulated Annealing, Tabu Search, Variable Neighbourhood search, or Genetic Algorithms) in order to leave local optima during improvement steps.

# Flow Shop Production

The arrangement of working systems is based on the work plans of the goods to be used.For uniform flow of material the work systems are arranged according to their position in the work plans of the products to be produced, which is usually linear. Of course this is only useful if, in the range considered, a single basic product or a limited number of product variants is manufactured. One distinguishes:

#### Flow shop production without fixed time restriction (Series production)

Series production is when there is no time limit, for the implementation of the *work content* of a station, specified. This often means that buffer inventories must be set up to accommodate partially machined workpieces until the next station to be traversed is free again.



Here, the flow of materials for all products is almost identical. Individual workstations can indeed be skipped, setbacks are not possible. Because of the possibility of temporary storage in the buffer the individual products may differ in the processing times.

#### Flow shop production with fixed time restriction (Assembly line)

In temporal link between the operations, each station has a fixed predetermined maximum time (**cycle time**) for machining a workpiece (or a lot) available. This is called *flow production with time pressure* or *synchronized flow manufacturing*.

If the coupling is done by independent conveyors, the individual pieces can be moved independently (asynchronous flow of materials), it is called *flow production* (e.g. assembly of televisions).

A *concatenation* to a single automated system is called a *transfer line* (e.g. motor production) or an *assembly line*. In this case, the workpiece is fixed to the transport system and can only be moved simultaneously (synchronous material flow).



In the following we focus primarily on the case of the timed flow production. Here exactly one work piece leaves the conveyor after the expiration of the cycle time; the *production rate* corresponds to the reciprocal of the cycle time.

A *timed transport* can be achieved by allowing the conveyor to move forward with a continuously velocity. During the processing of a workpiece within a cycle, the persons working at the conveyore move parallel to the production line forward and at the end of the cycle they move back to the beginning of the station.

Another possibility for cyclic transport is to stop the conveyore during the processing and the workpieces move to the next station at the end of each cycle (*intermittent* transport).

Especially with time restriction, there is a standard model for configuration and performance tuning:

# Flow Shop Production

Due to technological conditions temporal order or precedence constraints may exist between operations. They can be displayed using a precedence graph:



The (multi-stage) production process for each product to be produced (order) can be decomposed into **n operations**. These are indivisible elementary activities or a series of work items that are for economic or technical reasons, to be run immediately consecutively. Each operation j can be assigned to its **processing time** *tj*.

|  |  |  |
| --- | --- | --- |
| Operation j | Predecessor | *tj* |
| 1 | - | 6 |
| 2 | - | 9 |
| 3 | 1 | 4 |
| 4 | 1 | 5 |
| *5* | 2 | 4 |
| 6 | 3 | 2 |
| 7 | 3, 4 | 3 |
| 8 | 6 | 7 |
| 9 | 7 | 3 |
| 10 | 5, 9 | 1 |
| 11 | 8,10 | 10 |
| 12 | 11 | 1 |

A **precedence graph** is a cycle-free directed graph *G* = (*V, E, t*) free of parallel arrows or loops. The node set *V* is set of all operations, the arrow set E represents all (direct) order relations, and the function *t : V→+* assigns each job *i* its processing time *ti*. G is cycle-free and thus topologically sorted, i.e., the nodes can be enumerated so that for all arrows (*i, j*) the relationship is *i<j*.

In flow shop production, the production units (labor and/or equipment) are arranged in the order of operations to be performed on a product. At each workstation one or more operations are run. Because every operation is indivisible it assigned to exactly one station. If *i* is to be processed before *j*, so (*i, j*) ∈ *E*, *i* and *j* can be assigned to either the same station, or *i* must be assigned to a previous station than *j*.

Since the precedence graph does not specify the order between all operations, there is a certain amount of discretion. This is to determine an assignment of operations to stations so that a time-or cost-based objective function is optimized subject to the precedence relationships and cycle time. At the same time the number of stations and the cycle time (and thus the production rate) are to be determined.

Even simple assembly line balancing problems belong to the class of NP-hard problems. Therefore, in general, no exact methods are given, determining the optimal solutions with polynomial computational complexity. Therefore, various heuristics have been developed.

### Single product problems (simple assembly line balancing problem)

#### A basic model with alternative objectives

It is based on the following assumptions

* production of **one** homogeneous product in n operations
* fixed predetermined processing times *ti* for the operations *j = 1,...,n*
* order relations in the form of a precedence graph
* all stations have the same cycle time
* fixed stimulus rate
* stations equipped equivalent (in terms of personnel and equipment)
* nor parallel stations
* closed stations
* immovable workpieces

Concerning the **objective** one can distinguish between three main alternative forms of the basic model.

##### **Alternative 1: Minimizing the number of stations at a given cycle time**

At a given cycle time *c* the number of stations *m* is to be minimized[[4]](#footnote-4). Here, a simple lower bound on the number of stations can be obtained from the processing times and the cycle time (ignoring the indivisibility of operations and precedence relations) [[5]](#footnote-5):



A total of Σ*jtj* units of time of job content have to be completed and per station a maximum of *c* time units can be completed (if there were no idle time).

An *upper bound* on the number of stations results from the consideration that there is at least an optimal solution in which the first *m*-1 stations are *fully occupied*[[6]](#footnote-6):



Proof: Let *t*(*Sk*) be the occupancy times of the stations *Sk, k = 1, ..., m*. Then because of the integrality *t*max + *t*(*Sk*) > *c* also *t*(*Sk*) ≥ *c* + 1 - *t*max  for all *k* = 1,...,*m*-1.

By summing up the inequalities following results: 

The inequality  and the integrality of *m* result in the above upper bound.

### LP formulation for a given cycle time (Alternative 1)

We limit ourselves to the representation of an integer LP model. We use a (preferably good, i.e. low) upper bound on the number of stations *m*max, which for example was determined using a heuristic (otherwise n is of course a rather poor upper bound)

We define for all *j* = 1, ..., *n* and *k* = 1, ..., *m*max the binary variables *xjk*:



And note that is the number of the station, which the operation *j* is assigned to.

Assuming without loss of generality that the graph *G* has the node *n* as a single sink (that operation *n* must be the last), we obtain the following model formulation:

Minimize  ...number of the last station (with operation *n*)

Subject to the constraints

 for all *j = 1, ... , n* ...operation on exactly one station

 for all *k* = 1, ... , *m*max ... Compliance with the cycle time at station k

 for all  ... Precedence relations

 for all *j* and *k* ... binary variables

The model size can be reduced if one considers that some operations cannot be made in every station due to the cycle time and the order relations. E.g. *xnk* can be set to 0 for all k ≤ *m*min

Notes (possible extension of the IP):

When *assigning constraints* are in the form of operating material or position constraints, the corresponding variables from the model can be removed or fixed in advance to zero.  
If one wants to ensure for example that two operations *h* and *j* with (*h, j*) ∈ *Ε* cannot run together in the same station (operation constraint), then one requires in addition:  
  with (*h, j*) *∈ E.*

**Alternative 2: Minimizing the cycle time**

At a given number of stations *m* the cycle time *c* is minimized (that is, to maximize production speed). This is particularly important if an existing assembly line is to be returned. There are several lower bounds for the cycle time *c*:

* Let  *t*max =max {*tj* ⎜ *j = 1, ... , n*} be the duration of the longest operation, we obtain due to the indivisibility of operations immediately *c* ≥ *t*max.
* If a production or sales volume quantity *q*maxis specified in the planning period (e.g. in a shift) of lenght *T*, then
* With help of the given station number *m* of course: 

Overall, we obtain:



Similar considerations as for *m*max in alternative 1, one can determine upper bounds *c*max for the cycle time; see Chapter 4.3 in Domschke, School and Voß (1993). An upper bound is of course obtained from the minimum production quantity *q*minin period *T*:



##### **Alternative 3: Maximizing of efficiency**

This is the most complicated case. To determine is a positive cycle time *c* and a positive number of stations *m* so that with an feasible assignmet of the *n* operations to the *m* stations the *efficiency* or *“Bandwirkungsgrad”* BG (the utilization of the assembly line) is maximized

*BG = .*

Efficiency of 1 means utilization of 100%. This is only possible if there are no idle times.

The result is only a nontrivial optimization problem, if an *upper bound for the cycle time* *cmax* (for example over a minimum volume of production *q*min) is predetermined, because otherwise with the choice of *m* = 1 and *c* = Σ*jtj* an efficiency of 1 can always be achieved. From the maximum cycle time *cmax* a lower bound on the number of stations results:

.

Obviously for the efficiency, a nonlinear dependency from the variables *c* and *m* consists, which complicates the optimization. Therefore, it is useful to limit the range of values of the variables further if possible. Thus the lower bounds *t*max and  are valid for the cycle time as in the case of Alternative 2. With the minimum cycle time *cmin* one can use the upper bound *m*max from alternative 1, if one replaces *c* through *cmin*.



|  |  |  |
| --- | --- | --- |
| Operation j | Predecessor | tj |
| 1 | - | 6 |
| 2 | - | 9 |
| 3 | 1 | 4 |
| 4 | 1 | 5 |
| *5* | 2 | 4 |
| 6 | 3 | 2 |
| 7 | 3, 4 | 3 |
| 8 | 6 | 7 |
| 9 | 7 | 3 |
| 10 | 5, 9 | 1 |
| 11 | 8,10 | 10 |
| 12 | 11 | 1 |
| sum |  | 55 |

*The above example:*   
Shift duration of *T* = 7,5 hours  
Minimum production quantity *q*min = 600 Stück

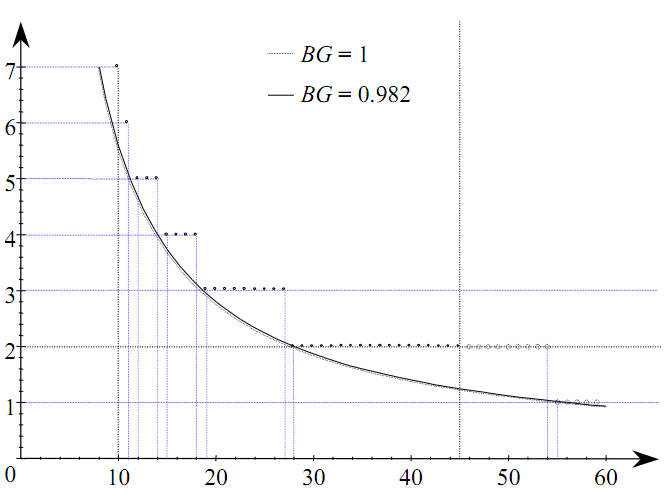
⇒  seconds/piece ... maximum cycle time



Because of Σ*tj =* 55 at least  stations are needed.

For no maximum sales volume ⇒ Cycle time at least *c*min = *t*max = 10 seconds/piece.

Number of stations m



Cycle time c

The above illustration (by Domschke, Scholl and Voß, 1993) shows the range of combinations of *m* and *c* for which a feasible solution to the problem (subject to the precedence relations) exists (Optimal solution at each given cycle time).

The theoretical maximum efficiency *BG* = 1 can only be achieved for forbidden values of *m* = 1 and *c* = 55. In the feasible range 10 ≤ *c* ≤45 and *m* ≥ 2 the optimal efficiency is *BG* = 0.982 and is achieved with the values *m* = 2 stations and *c* = 28 seconds/piece.

For *BG* = 0.982 (solid) and *BG* = 1 the (*c,m*)*-*isoquants are plotted. The further left down, the higher the efficiency. The table below shows the feasible cycle times *c* for the different number of stations *m*.

|  |  |  |  |
| --- | --- | --- | --- |
| Number of stations  *m* | Theoretical minimum cycle time | Minimum feasible cycle time *c* | efficiency  *55/c⋅m* |
| 1 | 55 | Not feasible as *c ≤ 45* | - |
| 2 | 28 | 28 | **0,982** |
| 3 | 19 | 19 | 0.965 |
| 4 | 14 | 15 | 0,917 |
| 5 | 11 | 12 | 0.917 |
| 6 | 10 | 10 | 0,917 |

With increasing cycle time efficiency is reduced (and the idle time share is increased) until a station can be saved. Therefore for any number of stations *m* the efficiency has a local maximum at the smallest cycle time *c*, for which a feasible solution with *m* stations exists.

Due to the complex influence of *m* and *c* on BG the literature usually assumes that on of the two values is given (alternative 1 or 2)

If *c* and *m* are to be minimized simultaneously, often the *weighted sum* of *cycle time* and *number of stations* is minimized.

##### Other objectives for the basic model

Maximizing the efficiency *BG* = Σ*tj*/*m⋅c* is due to the deterministic processing times equivalent to other time-oriented objectives:

Minimizing the *throughput time*: *D = m ⋅ c*

Minimizing the *sum of idle time*: 

Minimizing the *balance delay*: *LA = = 1 - BG*

Minimizing the *total waiting time*: *W = D -*

Since Σ*tj* is a constant, the given objectives are only determined by the cycle time *c* and the number of stations *m*.

An as uniformly as possible utilization of the stations can be pursued, in comparison to maximizing efficiency, as a subordinate objective.

#### LP formulation for a given number of stations

In this case one replaces *m*max with the given number of stations *m,* sets the cycle time *c* as an additional variable and minimized the cycle time:

Minimize *Z*(*x, c*) *= c* ...Cycle time

Subject to the coinstraints

 for all *j = 1, ... , n* ...operation on exactly one station

 for all *k* = 1, ... , *m* ... Compliance with the cycle time at station k

 for all  ... Precedence relations

 for all *j* and *k* ... binary variables

*c ≥ 0 integer*

#### Mathematical formulation for maximization of efficiency (BG)

Is neither the cycle time *c* nor the number of stations *m* given, the LP formulation for given cycle time can be taken, where the cycle time *c* acts as additional variable with additional constraints *c*≤ *cmax* and *c*≥ *cmin*.The objective function

Minimize 

is then not linear. To obtain a LP again, one can fall back on the weighting of cycle time and number of stations with factors *w1* and *w2* and the result is the linear objective function:  
 Minimize *Z*(*x,c*) *= w1⋅*(*Σk⋅xnk*) *+ w2⋅c*.

Even with not too large problems, the resulting LP-models are very large, especially the number of binary variables. Accordingly, in addition to special exact methods in particular heuristics play a major role.

# Heuristic procedures for a given cycle time

For the basic model of assembly line balancing a number of heuristics were developed. These are mostly priority rule procedures, but also shortened exact and enumerative methods.

#### Priority Rule Methods

Priority rule methods assign a rank value *RWj* to each operation *j*, using a priorityrule, yielding a possible consideration of the sequence of the allowed operations (priority list). A not yet assigned operation *j* can be **assigned** to a station *k*, if all his predecessors in the precedence graph are assigned to a station *1,..., k* and the current idle time of station *k* is not smaller than the processing time from *j*.

**Priorityrule methods**

Prerequisite: Cycle time *c*; to be scheduled operations *j=1,...,n* with processing times *tj ≤ c;* precedence graph is given by the set of predecessors *V*(*j*)

*k* Number of the current station  
 Idle time of the current station  
*Lp* List of already assigned operations (according to scheduling sequence)  
*Ls* Sorted list of n operations according to priorityrule

An operation *j∉LP* can be scheduled, if *tj ≤ * and *h∉Lp* hold for all *h∈V*(*j*).

One proceeds station by station and and from the set of not yet assigned operations the one with the highest priority is assigned. Most of the procedures only open a new station when the current station is considered fully occupied (that is if no further operation can be assigned to that station).

**Start**: determine list *Ls* using a priorityrule; *k := 0; LP := <]; ...* nothing scheduled yet

**Iteration:**

**repeat**

k := k+1; := c;

**while** a schedulable operation in the list *Ls* exists for station *k* **do**

begin

chose and remove the first schedulable operation *j* from list *Ls*;

Lp:= < Lp,j]; :=- tj

end;

**until***Ls**= <];*

**Ergebnis**: *Lp* contains a feasible sequence of the operations with *m = k* stations.

Depending on whether such a procedure is run through one or more times (with different or mixed priority rules), one distinguishes in literature *single-pass* and *multi-pass heuristics*.

The following **priorityrules** can be found among others in the literature:

**Rule 1**: *Random selection* of operations

**Rule 2**: Select the operations according to monotonically decreasing (or increasing) *processing time* *tj*: *RWj: = tj*

**Rule 3:** Select the operations according to monotonically decreasing (or increasing) *number of immediate succesors*: *RWj : =* ⏐*Ν*(*j*)⏐

**Rule 4:** Select the operations according to monotonically increasing *depth of the operations* in G:  
*RWj* : = Number of arrows on the route with the most arrows from a source of the precedence graph after *j.*

**Rule 5:** Select the operations according to monotonically decreasing *weight of position (value of position)*:   
 *RWj : = tj +*

**Rule 6:** Select the operations according to monotonically increasing *upper bound* of *j* and its predecessors needed *number of stations*:  
 **

**Rule 7**: Select the operations according to monotonically increasing *upper bound* for the *latest possible station* of operation *j*:  
*RWj : = *Where  (≤ n) is the number of stations in the best-known feasible solution. The term in brackets indicates the minimum number of stations occupied by a single operation *j* and all its successors.

Except for the creation of a priority list rule 7 can serve as a stopping criterion for a process as an improvement of a solution (compared to the best-known feasible solution) is no longer possible if operation *j* has not been assigned at least to the station RWj.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Example: We consider the above problem and apply rule 5. We receive for the operations in the following table weights of position (position values). | | | | | | |  | | | | | | | |
| *j* | 1 | 2 | 3 | 4 | 5 | 6 | | 7 | 8 | 9 | 10 | 11 | 12 |
| *tj* | 6 | 9 | 4 | 5 | 4 | 2 | | 3 | 7 | 3 | 1 | 10 | 1 |
| *RWj*(*5*) | 42 | 25 | 31 | 23 | 16 | 20 | | 18 | 18 | 15 | 12 | 11 | 1 |

Selecting the cycle time *c = 28* we receive the following heuristic solution with *m = 3* stations and an efficiency *BG = ∑tj/*(*3⋅28*) *= 0.655*:

*S1* = {1,3,2,4,6}, *S2* = {7,8,5,9,10,11}, *S3*= {12}

The following table contains the rank values for the rules 7 (für *= 3*), 6 and 2:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *j* | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| *RWj*(*7*) | 1 | 2 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| *RWj*(*6*) | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 2 |
| *RWj*(*2*) | 6 | 9 | 4 | 5 | 4 | 2 | 3 | 7 | 3 | 1 | 10 | 1 |

Applying primary rule 7 (latest possible station), for equality rule 6 (for *j* and all predecessor required number of stations) and for once again equality rule 2 (in order of decreasing *tj*) so that we obtain for *c* = 28 the following solution with *m* = *2* and *BG* = *0.982*:

*S1 =* {1,3,2,4,5}, *S2 =* {7,9,6,8,10,11,12}

#### More heuristic methods

*Stochastic variants* of the above deterministic priority rules 2-7 are obtained if for each scheduling the respective operation of the schedulable operations is chosen randomly. The selection probabilities can be determined proportionally or inversely proportionally to rank values. Another possibility is to determine a priority rule for each scheduling step randomly, where previous experience may be used.

Enumerative heuristics generate for example first all feasible assignments for the first station. The one station scheduling with the lowest idle time is picked. Based on the already scheduled operations, the stations 2.3, ... are formed similarly. (Greedy)

Because of the relationship with cutting and packing problems, their heuristics (with the additional consideration of precedence relations) can be adapted, e.g. generalization of the First-Fit-Decreasing heuristic for the bin packing problem.

There are also formulations as a shortest path problem with exponentially many nodes (in dependency of the data)

Permutation procedures (exchanging operations between stations) to improve (minimize the number of stations) the subordinate objectives of a uniformly utilization of stations are also possible: see literature in Domschke, Scholl and Voß (1993).

#### Worst-case analysis of heuristics

The following solution properties are guaranteed, for integrality of *c* und *tj*(*j* = 1,...,*n*), by the procedures of most heuristics for alternative 2:



The properties indicate that the sum of the occupation times, in each of two adjacent stations, must exceed the cycle time by at least 1 time unit, as they might otherwise be combined into a single station. From that, considering the integrality of *c, m* and *tj*, one can derive the following worst-case bounds on the deviation of a solution with *m* stations from the optimal solution with *m\** stations:

*m/m*\* ≤ 2 - 2/*m*\* for even *m* and *m/m*\* ≤ 2 - 1/*m*\* for odd *m*

*m* < *c⋅m*\*/(*c - t*max + 1) + 1

Here we are not interested in the (relatively simple) proofs and mention only that one can construct examples in which growing parameters *n* and *c* the deviation m / m \* converges to 2, i.e. Error bound a) can be assumed asymptotically. Also applying in the general case (i.e. without special assumptions on problem data) for the worst-case bound of each *polynomial* heuristics for the assembly line balancing problem: *m/m*\* ≥ 3/2.

### Methods for determining the cycle time

If the cycle time is not given, but is to be minimized due to a given number of stations (Alternative 3), or is to be optimized together with the number of stations to achieve a maximum efficiency (Alternative 1), so one can modify many of the procedures developed for alternative 2, in particular the exact procedures.

A simple procedure for alternative 3 is the following iterative procedure:

#### Iterative procedures to determine the minimum cycle time:

determine the theoretical minimum cycle time    
(or. *c*min = *t*max if it is larger) and set *c* = *c*min

find for the cycle time *c* an optimal solution with minimum number of stations *m*(*c*) using procedure for alternative 2 (vgl. § 2.3.2 und 2.3.3).

If *m*(*c*) is larger than the given number of stations, enlarge *c* by Δ (integer) and reapeat step 2.

Feasible solutions with cycle time ≤ *c* and number of stations ≤ *m* found.

If Δ > 1, one can stille make a *nest of intervalls*:   
if for the cycle time *c* a solution with number of stations ≤ *m* has been found and not for the cycle time *c*-Δ, on can still try *c*-Δ/2, etc.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Example: We consider the above problem and assume that exactly m = 5 stations can be filled. One is looking for the maximum possible production rate.  One is looking for the minimum cycle time. We apply again rule 5 and receive the following *position weights* | | | | | | |  | | | | | | | | |
| *j* | 1 | 2 | 3 | 4 | 5 | 6 | | | 7 | 8 | 9 | 10 | 11 | 12 |
| *tj* | 6 | 9 | 4 | 5 | 4 | 2 | | | 3 | 7 | 3 | 1 | 10 | 1 |
| *RWj*(*5*) | 42 | 25 | 31 | 23 | 16 | 20 | | | 18 | 18 | 15 | 12 | 11 | 1 |
| At least the cycle time  *c*min = Σ*tj*/*m* = 55/5 = 11 has to be chosen  (es ist 11 > *t*max = 10):  *We try c = 11:* The corresponding solution {1,3}, {2,6}, {4,7,9}, {8,5}, {10,11}, {12} needs 6 > *m* = 5 stations.  One immediately sees that for *c* = 12 the 5 stations are sufficient, as one can assign operation 12 to station 5: *S*5 = {10,11,12}. | | | | | | | |  | | | | | | | | |

In large problems often the *c*, for which a station assignment with a given number of stations exists, is significantly greater than *c*min, so that the gradual increase of *c* by 1 would take too long. Therefore, increaseses of Δ> 1

A B&B-procedure for alternative 3 can be found in § 4.3.4 of Domschke, Scholl and Voß (1993).

# 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, … , *n*} … set of all operations with

*tj* ... durations *tj* ≤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.. Zk ⊆ A

*L*1 ... list of all states in stage k-1

*L*2 ... list of states in stage k

*E*k ... set of possible alternative assignments to station *k*

*S*k ... current assignment to station *k* in stage *k*

**Start:** *L*1:= < {} ]; (*empty set - nothing assigned yet*)

**Iteration k = 1, 2, ... :**

*L*2:= <]; ... (*start with an empty station*)

**while** *L*1 ≠ < ] **do** (*as long as not all states of stage k-1 have been considered*)

**begin**

choose and remove the first element Zk-1 of L1:

construct the set *E*k of all possible allocations of station *k*:



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

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

;

**while** *Ek* ≠ {} **do** (*add the new stations k to the states in list* *L*2)

**begin**

select and remove an element Sk of the set *E*k;

*Zk*:= *Zk*-1∪*Sk*; (*add Sk to the previous state Zk-1*)

add Zk to list L2;

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

**end;**

**end;**

L1: = L2;

**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 sates and stations cen be ignored that are dominated by some other station with the same starting state *Zk*-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 Sk with starting state *Zk*-1 is dominated by station assignment S'k with the same starting state, if Sk ⊂ S'k.

Example: In the above example in stage 2 the station assignments S2 = {2} and S2 = {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:

** ... set of all immediate successors of *all* operations in set *J.*

With this, we can formulate:

**Dominance rule 2:** station assignment Sk with starting state *Zk*-1 is dominated by station assignment S'k with the same starting state, if the following holds:  
 and    
where  
J1 = Sk - S'k and J2 = S'k - Sk

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 J1 also depend on J2. This means, that all successors of J1 are only available, if all operations in J1 and J2 have been assigned.

Choosing station assignment S'k instead of Sk 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   
| J1| = | J2| = 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  S1 = {2} is dominated by S'1 = {1} in stage 1, since   * S'1 has more workload assigned (less idle time) than S1, t2 < t1 and * N(S1 - S'1) = N({2}) = {3}  N(S'1 - S1) = N({1}) = {3, 4},  i.e. N(S1 - S'1) ⊂ N(S'1 - S1)   Hence the partial tree starting in node 1 can be eliminated.  In the same way, in stage 2 and Z1 = {2} the possible station assignment S2 = {4, 5} is dominated by S'2 = {2, 4}. |  |

**Remark:** The following example shows, that condition *N*(J1*)* ⊆ *N*(J2) is actually needed and that a better workload alone does *not* guarantee dominance:

|  |  |
| --- | --- |
| Example: c = 40 | Although t1 ≥ t2 and t1 ≥ t3, the stations S1 = {2} and S1 = {3} are *not* dominated by S'1 = {1}.  This is because J1 = {2} and J2 = {1} so that  *N*(J1) = {5} is *not* contained in *N*(J2) = {4}.  The optimal solution is  S1 = {2}, S2 = {3, 5}, S3 = {1, 6}, S4 = {4, 7}  It is only reached if S1 = {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 *Zk* is dominated by state *Z'k* in the same stage *k*, if *Zk* ⊆ *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} ⊂ {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 *Zk* is dominated by state *Z'k*, if for *J*1 = *Zk*- *Z'k* and *J*2 = *Z'k* *- Zk* holds:  and 

|  |  |
| --- | --- |
| **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.

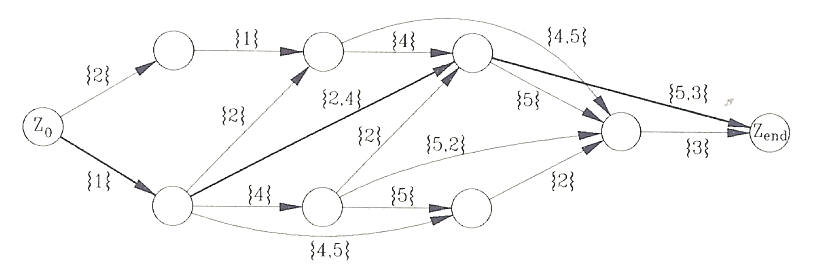
However, the graph/tree will still be very large. Therefore a heuristic has been developed that is based on this shortest path problem but only considers a subgraph (at the cost of loosing the guarantee of optimality).

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*l,. ... , *jn*) of operations, define nodes (states)   
   *Z*0 = {}, {*j*l}, {*j*l, *j*2}, ... , *Z*end = {*j*l, ... , *jn*}.
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: 
4. In the resulting graph find the shortest path from *Z*0 = {} to *Z*end = {*j*l, ... , *jn*}.

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 exampleand 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.

1. Heizer, J., Render, B., Operations Management, Prentice Hall, 2006, Chapter 9 [↑](#footnote-ref-1)
2. Francis, R., McGinnis, L., White, J., Facility Layout and Location: An Analytical Approach, Prentice Hall, 1992 [↑](#footnote-ref-2)
3. Francis, R., McGinnis, L., White, J., Facility Layout and Location: An Analytical Approach, Prentice Hall, 1992 [↑](#footnote-ref-3)
4. That c and tj have to be integer values can be required for practical problems without limitation; the input data is to scaled properly. [↑](#footnote-ref-4)
5. Thereby denotes  the next smaller and  the next largest whole number. [↑](#footnote-ref-5)
6. A station is fully assigned when no additional operation can be absorbed into the station without breaking the cycle time restriction or sequence relations. [↑](#footnote-ref-6)