Disassembly planning

Reverse Logistics
for Industrial Engineers

Part 2

Modeling and Analysis of Reverse Logistics Systems

Brunel University, Uxbridge, Middlesex, UK (BU)
Eindhoven University of Technology, Eindhoven, NL (TUE)
Universitat Politècnica de Catalunya, Barcelona, SP (UPC)

Contents

Subject 2.3: Disassembly Planning ($^{©}$A.J.D. Lambert, TU/e)

2.3. Disassembly Planning

A.J.D. Lambert
Eindhoven University of Technology, Dept. of Technology Management
TM/AW IPO 2.32
a.j.d.lambert@tm.tue.nl

0. Introduction

Chapter 2.3 of this Course deals with modeling the disassembly process of discarded complex products and the context of this process. Therefore, the first two sections deal with the context, which includes *industrial ecology* and *chain analysis*, based on material and energy flows in the industrial system. It is focused on the role of recovery of discarded complex products in attaining the objective of closing the materials cycles, which is the crucial paradigm of industrial ecology. Disassembly, which is a principally non-destructive method of separating a complex product in its different components or modules, is focused on in section 3. Modeling disassembly processes is discussed in section 4. Disassembly modeling is crucial in disassembly sequencing, which involves the determination of the sequence in which the different disassembly operations have to be performed. This can be applied in answering questions on such as: how many possible disassembly sequences are available (section 5), how can we reduce the search space (section 6) and which of the sequences is the optimum one (section 7). Because the search space in this kind of problems typically increases exponentially with the number of components of the product, which is demonstrated in section 8, exact methods that are based on mathematical programming appear useful only up to a definite degree of complexity. Metaheuristic and heuristic methods have to be used if dealing with increasingly complex products. We will present some of these methods in section 9. Finally, a modified model for optimally meeting the demand on a typical set of components is discussed in section 10. The chapter concludes with a literature list aimed at obtaining further depth in this matter.

1. Industrial Ecology

1.1. Basic concepts

As reality is complex, one tries to gain insight in dedicated processes in reality by *modeling*, which involves the mapping of reality according to definite criteria that are selected by the modeler. In modeling, complete reality is called the *universe*, and part of the universe that is modeled is called the *system*, which is confined by some *system boundary*. Only a selection of the properties of the system is included in the model. Usually, a system is reduced to a set of *objects*, called *subsystems*, which are subjected to mutual *relationships* and also have relationships with the reality outside the systems boundary.

Confining ourselves to the biosphere of the planet Earth. This includes the soil, the waters (hydrosphere), the atmosphere, and the organisms living therein. Focusing on transformation of materials and energy flows only, we observe that this system, which is called natural or *ecosystem*, although it evolves, is strikingly stable on a time scale of about 10^9 years. This not only because of its principal energy source, solar radiation, is also stable on this time scale, but also because no waste is produced. Mass exchange between the biosphere and the universe is rather small, apart from some circulation processes in the Earth's crust and mantle, and the biosphere itself does not degrade by waste production. Two type of processes are considered. The inorganic or *geological cycles*, which include volcanism, erosion and sedimentation, evaporation and precipitation, etc., and the organic or *biological cycles*, which include biomass, such as living and dead bacteria, plants and animals, excrements, and exhalation products. These are recycled into new organisms, apart from some materials that leave the biosphere in order to enter the geological cycles, i.e. via formation of fossil fuels as coal, oil, and natural gas, limestone deposits, and so on.

The *ecosystem* thus is characterized by many different cycles, each on a different time scale. Many of these scales are short-cyclic, which involves time spans of 10 to 100 years. Well known examples are the water cycle, the carbon cycle, the sulfur cycle, etc.

Mankind evolved out of the ecosystem, but remained part of it. Technology was invented and gradually evolved to an industrial system that was powered successively by humans and animals, wind and water, wood, peat, fossil fuels and, ultimately, nuclear power. This industrial system is also called *technosystem*.

The technosystem was originally based on resources with limited *capacity*, such as water power, but this evolved to dependence on resources with finite *stock*, such as crude oil. Basic materials for manufactured products shifted from raw materials out of agriculture, such as wood, to raw materials obtained by mining, such as metals, and even synthetic materials such as plastics. In more recent times, an increasing amount of complex products is manufactured, consumed, and discarded. These products are slowly or not converted to naturally occurring substances. Hazardous substances can even leach out of these discarded products and cause environmental pollution. Therefore, the waste problem has become even more complex than it was already.

Although destruction of habitat and degradation of biodiversity, powered by both indifference and short-term thinking, remains the major environmental problem, the problems related with materials flows, specifically depletion of natural resources and accumulation of waste, should also be accounted for in an integral and systematic way. The science that is involved in this topic is called *industrial ecology*.

The term 'Industrial ecology' has been coined by Frosch and Gallopoulos (1989). It is the science that attempts to mimic ecological mechanisms in the technosystem, aimed at achieving sustainability. *Sustainability* is usually considered using natural resources in such a way that the possibilities of the future generations are not affected. In the well-known U.N. report 'Our Common Future' (Brundtland, 1997), this is spelled out as: *Meeting the needs of the present generation without compromising the ability of future generations to meet their needs*. Unfortunately, the meaning of this definition can be quite easily misinterpreted when subordinated to specific, short-term interests. In industrial ecology it involves two topics:

- Using sustainable, i.e. solar, energy;
- Considering waste as a 'residual' product that can be recycled to a resource. This apart from energy and materials conservation measures.

Quantitative industrial ecology studies the quantitative aspects of industrial ecology in an integral manner, focusing on the physical flows (materials and energy) in the technosystem. These are considered using systems theory, with the technosystem as the system considered, the ecosystem as its environment, energy and substance flows as relationships, and transformation processes as objects.
Transformation processes can be distinguished in those which cause change in time (storage) place (transportation), and quality of a material object. Transformation processes always require *ancillaries*, from which *energy* is the most essential one. *Production processes* are transformation processes intended for adding *value* to the product. Although 'value' is an economic and not a physical concept, it is usually connected to the physical properties of a product, which are a prerequisite for its (technical) *functionality*, which is considered apart from emotional issues.
A schematic production process is depicted in figure 1,

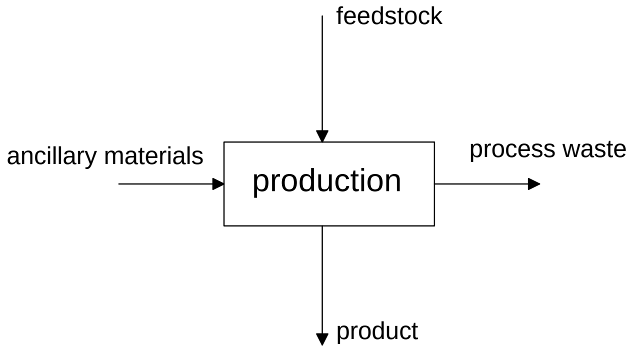

Figure 1. Materials flow in a production process.

Note that the mass is conserved here, as is the mass of the various individual chemical elements involved in the process. The process waste involves emissions to the atmosphere, to the surface or groundwater, to the soil, and solid waste. Also note that energy is not involved in this schematic.
In the industrial ecology concept, one is focused on reducing the feedstock required for a unit of product, as well as on usefully employing the waste, which is now called residual product. We discern intended products; co-products that are valuable as well; by-products which are unintended but have a positive, albeit low, value; and residual products that have a negative value and must be upgraded for being appropriate as a substitute for virgin materials in some production process that may be different from the original one.
Consumption processes are similar to production processes, but in the course of these processes the value of the product degrades, because the product is intended for delivering services to the consumer. The consumption process ultimately results, apart from process waste, in *discarded* or *end-of life products*.

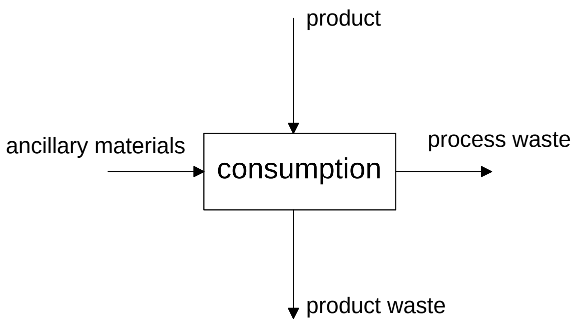

Figure 2. Materials flow in a consumption process.

1.2. Chain analysis

Production and consumption processes are no individual entities: these are combined which each other to form *product-process chains*, see figure 3.

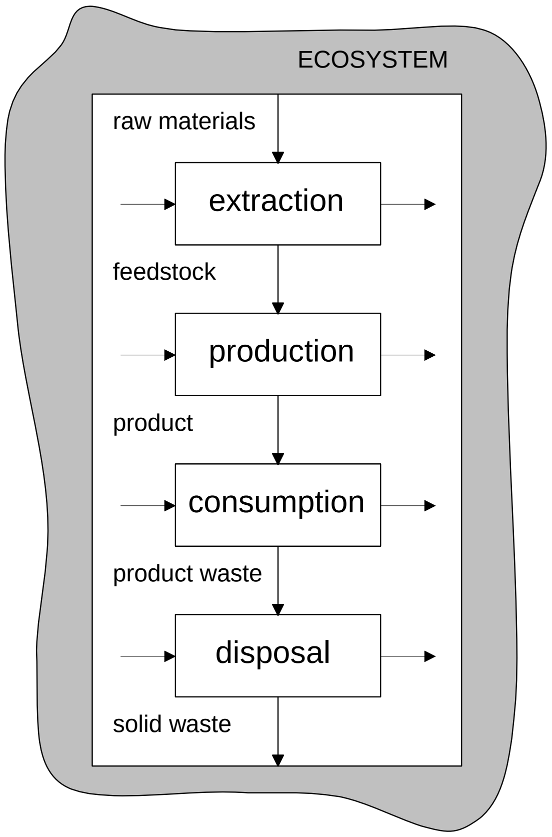

Figure 3. Aggregated linear product-process chain.

Obviously, these chains can be considered in different levels of aggregation. Highly aggregated chain models can be disaggregated in either a horizontal or a vertical manner. In the *horizontal* manner, different products are discerned or part of the technosystem, typically a particular product, is considered. In the *vertical* manner, a process is divided in sub-processes, such as is usual in *supply chain analysis* or in flow diagrams of production processes in a particular plant. We always implicitly apply horizontal disaggregation, as we usually consider one product or a class of products, such as discrete complex products, in stead of the complete technosystem. Definition of the systems boundary can become critical in this case.

Vertical disaggregation usually starts with decomposing the production process into a sequence of subprocesses. In case of discrete complex products, this sequence typically involves: (1) Extraction, which involves mining, minerals reclamation, and agriculture; (2) Materials production, which is the transformation of the intrinsic properties of materials. This takes place in the process industries; (3) Components production, which is the transformation of the extrinsic properties of materials, such as shape. This takes place in the manufacturing industries; (4) Assembly of the components to obtain discrete complex products.

Discarded products, on the other hand, can usually be upgraded such that these, entirely or partly, can either be used again as such, or in the shape of modules or components, or as secondary materials. Therefore the discarded products can be subjected to a 'reversed' production chain that might include: repair, remanufacturing, selective disassembly, bulk recycling, which includes freeing (shredding, grinding) and physical separation, components upgrading (rework) and materials upgrading, including chemical separation. Ideally, the functionality of the product should be maintained as much as possible. If not the product can be conserved as a whole, reuse of modules and components should be considered. If this is infeasible, recycling of the materials at an as high as possible level (homogeneity, purity) should be accounted for. If this can't be achieved, downcycling (cascading) to a lower-grade application can be applied. Ultimately, some residue will remain that has no application at all. This waste is discharged in a conventional way, via the hierarchy: energetic recycling, incineration without energy reclamation, controlled landfill.
An example of such a cyclic product-process chain is depicted in figure 4.

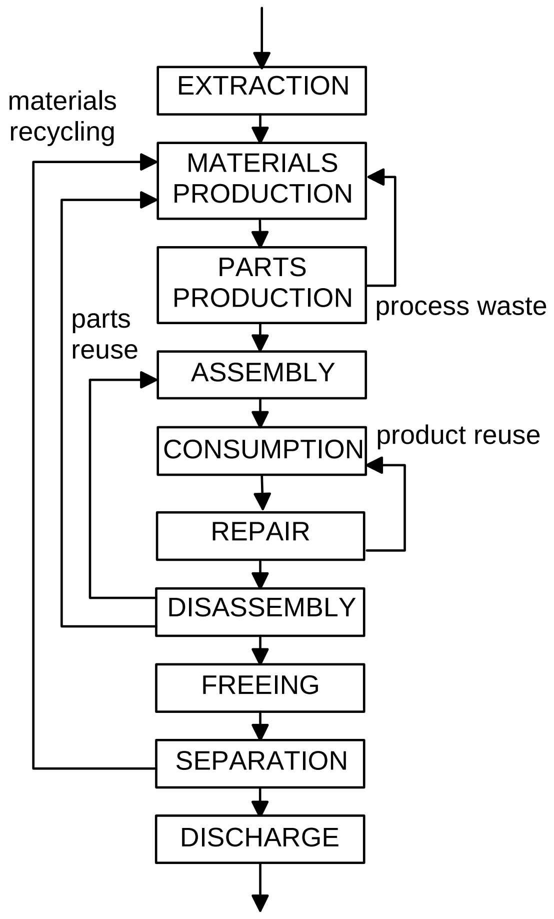

Figure 4. Product-process chain with reuse and recycling.

1.3. Numerical considerations
In *environmental life-cycle assessment* or LCA, the basic method is based on a *convergent* linear chain. This means that several partial product-process chains are merging together during the life-cycle. This refers to products that are made of different materials, such as glass and metal. Glass making and metal making are then considered in assessing the environmental impact of the product. Recycling the discarded product is usually *not* adequately supported by LCA software. At best, one can insert the percentages of the discarded product that are landfilled, incinerated, etc. So far the product is recycled, the environmental impact of the materials is considered the same, but with inverted sign, of that of the corresponding virgin materials, which are replaced by the secondary material, originating from recycling, see figure 5.

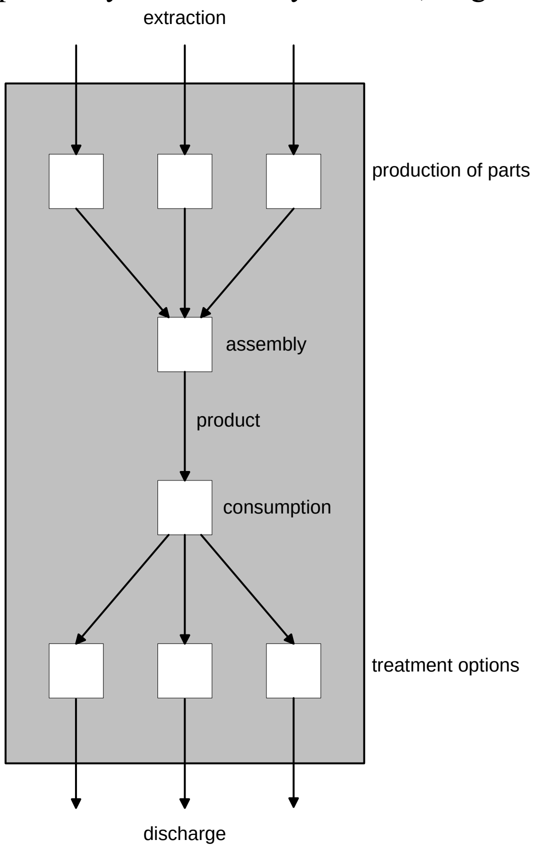

Figure 5. Product life-cycle chain in environmental life-cycle assessment.

Obviously, we might consider simple network analysis to assess the flows in the technosystem that are related to a specific product. In this case we start with a stationary approach in which the product stream is considered a continuous flow in the product-process diagram.

<u>*Exercise 1*</u>:
An example of such a diagram is in figure 6, which depicts a simplified life-cycle chain of a product. The flow of consumed products is normalized to 1, hence $x_4 = 1$. Recycling percentages are given as well, for instance the *reuse* ratio η_1 equals 40 mass% and the *recycling* ratio η_2 equals 80 mass%, or: $x_6/x_4 = \eta_1 = 0.4$, and $x_7/x_5 = \eta_2 = 0.8$.
Node equations are: $x_2 = x_1 + x_7$, and: $x_3 = x_2 + x_6$.
Obviously: $x_1 = x_8 = 1 - \eta_1 - \eta_1 + \eta_1 \cdot \eta_2 = 0.12$

This means that, even with such modest ratios, the same amount of products can be obtained at the expense of 12% of the original amount of raw materials only.

Costs can also be incorporated in this simple kind of model, as the costs for providing the consumer with a unit of products is given by:

$$C = \sum_i c_i \cdot x_i \quad \forall i \in P \tag{1.1}$$

Here *P* is the set of processes, c_i is the cost of processing one unit of input, and *C* refers to the total cost of the cycle. This kind of expression can be used for assessing the influence of , e.g., increasing landfill costs on decision making.

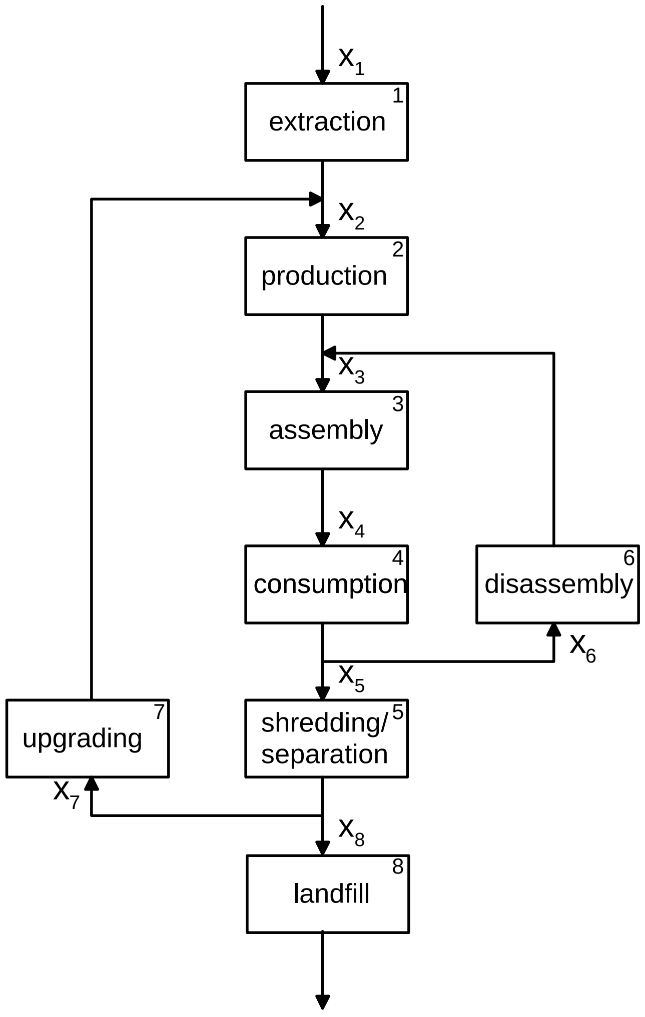

Figure 6. Simplified product life-cycle, see example 1

Note that practice is dynamic, including effects of stock at the different processes, caused by a considerable time of residence of the product at the consumer, e.g. 10 or more years for a refrigerator.

1.4. Waste flows

Waste is discerned into process waste and product waste. Particularly the complex products leave the consumption phase as discarded products, which still have a discrete character. Complex products leave their useful phase as discarded consumption goods and discarded capital goods. Although capital goods often have some different

characteristics, as these are more robust and are put available in larger and more homogeneous quantities, most of their characteristics are similar to those of discarded complex domestic products.
From a purely quantitative point of view, discarded complex products don't appear to play a major role, as their quantity is dwindling with respect to the total amount of waste produced, although its share tends to increase both in quantity and in complexity.
Although statistics appears to give good insight in the yearly amount of solid waste produced in a definite country, this figure must be considered taking notice of ambiguities in the definition of 'waste'. Dependent of the prevailing definition employed by the Statistics Office, the amount of waste is about several tons per capita per year ($t \cdot cap^{-1} \cdot yr^{-1}$). This might include dredging sludge, manure surplus and, in most countries, tailings and other mining refuse.
This bulk waste corresponds to several $t \cdot cap^{-1} \cdot yr^{-1}$. Building and demolition waste counts for about 500 $kg \cdot cap^{-1} \cdot yr^{-1}$. Coarse goods such as furniture count for about 50 $kg \cdot cap^{-1} \cdot yr^{-1}$. Discarded complex products count for about 125 $kg \cdot cap^{-1} \cdot yr^{-1}$. From this figure, about 50
$kg \cdot cap^{-1} \cdot yr^{-1}$ is due to discarded cars and other vehicles. 25 $kg \cdot cap^{-1} \cdot yr^{-1}$ is due to white, brown and gray goods. From this, the share of brown and gray goods is about 4 thru 10 $kg \cdot cap^{-1} \cdot yr^{-1}$.
2 $kg \cdot cap^{-1} \cdot yr^{-1}$ consists of Cathode Ray Tube (CRT) containing devices (TV and monitor) and 1 $kg \cdot cap^{-1} \cdot yr^{-1}$ consists of computer systems and peripheral equipment. 0.5 $kg \cdot cap^{-1} \cdot yr^{-1}$ is due to PWBs.
These figures are derived out of statistics from various industrialized countries. Unfortunately, the amount of discarded product is not easily measurable, as only part of those products is recollected. Production and sales figures, combined with an estimate of useful lifetime, are usually used for an estimate. Also the definition of the product categories is not unambiguous, with ever new types of product entering the market. Apart from this, double counting is introduced and overlap might take place. Apart from this, both the quantity and the diversity of these products is increasing, which straightly follows from production figures and the pace of the introduction of novel technology.
<u>Example</u>: Many expenditures have been done in recycling Cathode Ray Tubes, which are in TV sets, monitors, etc. Its glass contains Lead (Pb), Barium (Ba), and/or Strontium (Sr), which can be recycled to the same purpose. However, flat screens based on Liquid Crystal Display (LCD) have been introduced and recently massively penetrate in the domains of applications that were met with CRTs. This obstructs recycling, an only bulk application in enamel appears feasible yet, with a lot of CRT tubes still waiting for becoming discarded and recollected.
<u>Note</u>: *Large white goods* are refrigerators, washing machines, dishwashers, etc. *Small white goods* include small household appliances. *Brown goods* include TV and radio sets, CD and DVD players, and other entertainment equipment. *Grey goods* refer to ICT products such as PCs and peripherals, and telecommunication equipment.

2. Post-consumer processes

2.1. Framework

As we noticed before, the main objective of *reverse manufacturing* from an industrial ecology point of view is maintaining or restoring the original value of the product to a level as high as possible. The first step in the chain of operations aimed at doing so is recollection. Only part of the products in circulation will be recollected: the *recollection ratio* is smaller than 1. Apart from this, there occurs a serious restriction because of the dual character of the concept of 'value'. From a *functional* point of view, this value can be restored via repair, which might include the replacement of some worn-out components by new ones. Testing and control precedes the decision of doing so, as damage of the product can be of such an extent that repair is not feasible. From an *economic* point of view, the product's value can't easily be restored because the product can have become more or less obsolete. This results in a restricted or absent demand and/or a lower sales price. Technological progress surely plays a role here, but also fashion appears important. Sometimes, refurbishing is possible. Some definitions are given below:

- *Refurbishing* is the transformation of a discarded product to an as-new one via reconfiguration, via disassembly, rework if necessary, and subsequent reassembly with
- *Rework* is the processing of an out-of specification component in order to restore its functionality. The component can be out-of specification by errors in production or, in case of a discarded component, by wear or damage.
- *Disassembly* is the mainly non-destructive decomposition of a product in its constituents, which might be both modules and components.
- A *module* is a set of related components that has a functionality as such.
- A *component* is an object that makes part of a product and that cannot be decomposed in a non-destructive manner.
- *Functionality* is the ability to perform some function.

After disassembly, and sometimes in combination with this, dismantling operations might take place.

- *Dismantling* is the decomposition of a products in which some major components are irreversibly destroyed.

There are some reasons why disassembly and dismantling are only applied to a modest extent. The most important is cost, due to the labor-intensiveness of disassembly operations. Insufficient or absent demand on components can be a second argument. Technical arguments will be discussed in the next section. Robotic disassembly, which has been advocated by German and Japanese researchers in the nineties of past century, remains hardly attainable because of the restricted adaptive ness of present robot technology.

In many cases, *draining* is required, which consists of removing the working fluids, which are often hazardous and should not contaminate the recycled products. This is extremely important in car recycling, where petrol, lubricating oil, antifreeze, and some other fluids, are present.
Once being partly disassembled and dismantled, the products must be decomposed in chunks of a smaller size (size reduction) which is done in *shredders* or, in case of

smaller products, in mills or grinders. This is the usually a first step in a materials reclamation process.
Car shredders are huge machines that process car bodies to chunks by hammers and/or knives.
Dust is collected via a cyclone. The chunks are subjected to a physical separation process.

- *Physical separation* is the separation of a substance flow in subflows rich at definite materials via differences in physical properties, such as magnetism, electric conductivity, specific mass, optical properties, size, etc. Manual sorting and sieving are the most known of these methods.
- *Chemical separation* uses differences in chemical properties for the same purposes. Leaching and dissolving are typical chemical separation methods.

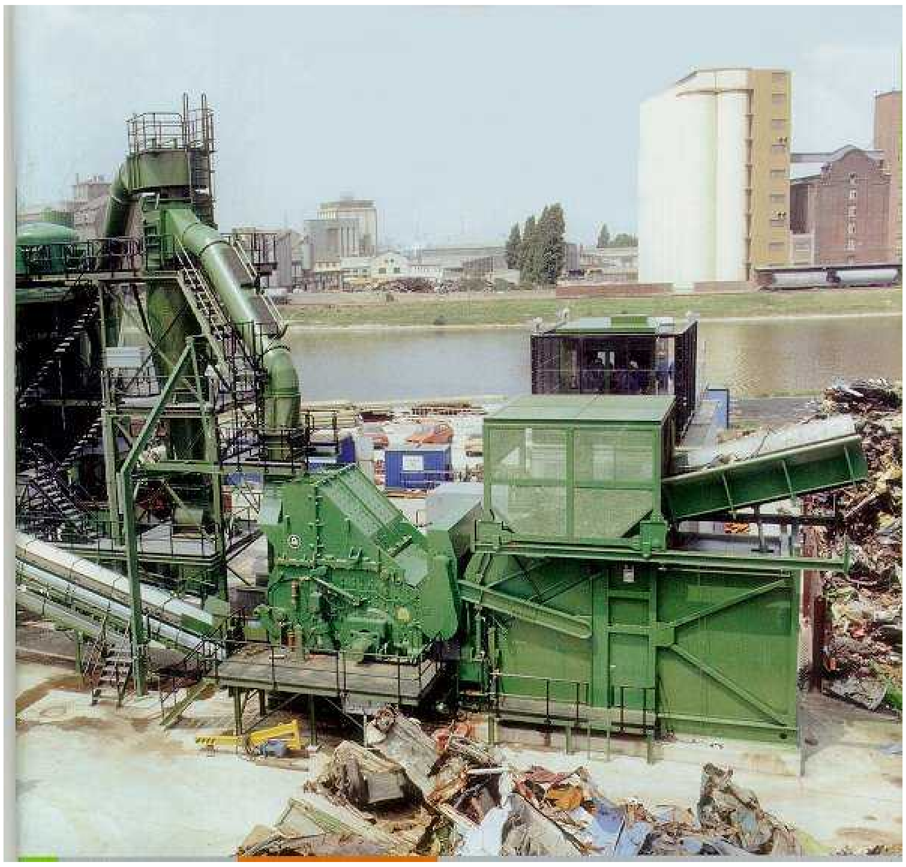
Figure 7. A car shredder.

Physical separation methods originate from mining, where chunks of ore are crushed (figure 8), after which separation in metal rich and metal poor chunks has to take place, which is called *ore concentration*.

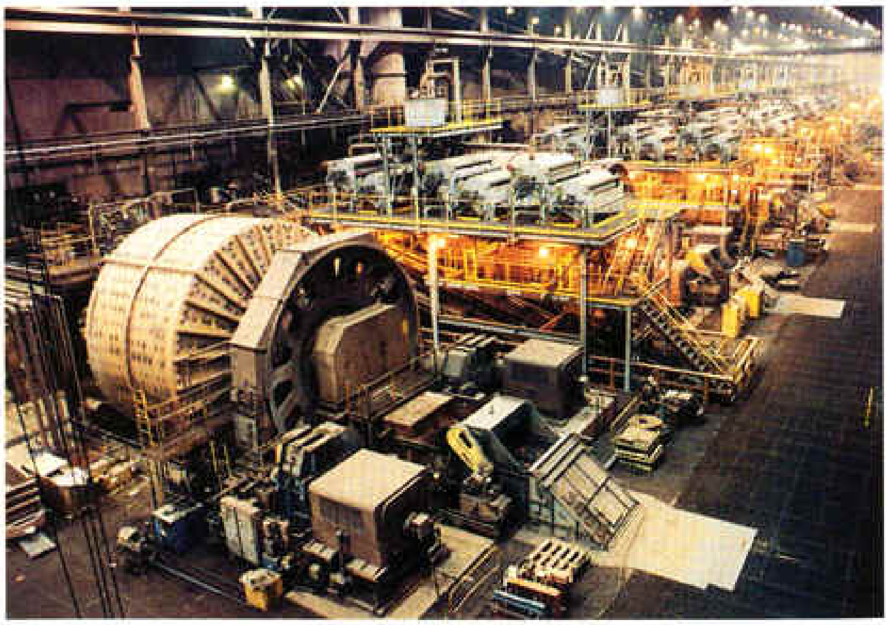
Figure 8. Ore crushing.

As discarded products can be considered a complex ore as well, methods derived from those in ore concentration can be used. Usually, a train of separation facilities is applied. The most important of these are magnetic separation and eddy current separation.

Magnetic separation uses an electromagnet, which attracts those chunks that are rich in ferrous materials.

Eddy current separation uses a rotating permanent magnet, which induces eddy currents in those chunks that have a good electric conductivity. These in turn cause a magnetic field that repels the conducting chunks from the conveyor, so these can be separated and collected in a bin, see figure 9 for an application in bin recycling.

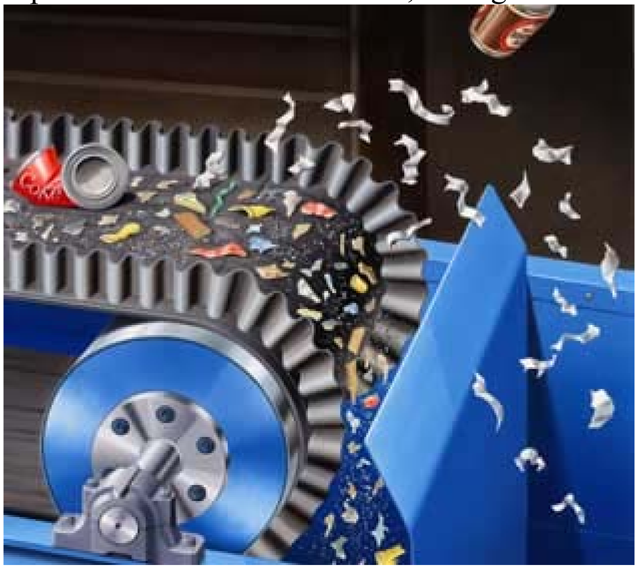
Figure 9. Eddy current separation

Because the eddy current separation is usually preceded by magnetic separation, it sorts out the chunks that are rich in *non-ferrous metals*, such as aluminum.

Air classifiers can be used for separating chunks of copper wire in electronic equipment recycling, by blowing a stream of air over the feed, thus carrying along the copper wire bits that have to be chopped beforehand, of course.

What remains, when the metals are separated, is the *shredder light fraction* or *shredder residue*. Here are dust particles and metal-poor chunks, usually consisting of light materials, such as various plastics, rubber, textile, glass, chips of paint, dirt, and related substances. Reclamation of some of these materials theoretically might be still possible, but the value is often low and the separation process is complex. In industrialized countries, landfill of shredder residue is not encouraged. Because of the presence of possible contamination, incineration has often to take place under conditioned circumstances, and the ultimately resulting slag must also be landfilled in a dedicated landfill.

A typical product-process diagram of the life-cycle chain of a complex product is then as follows:

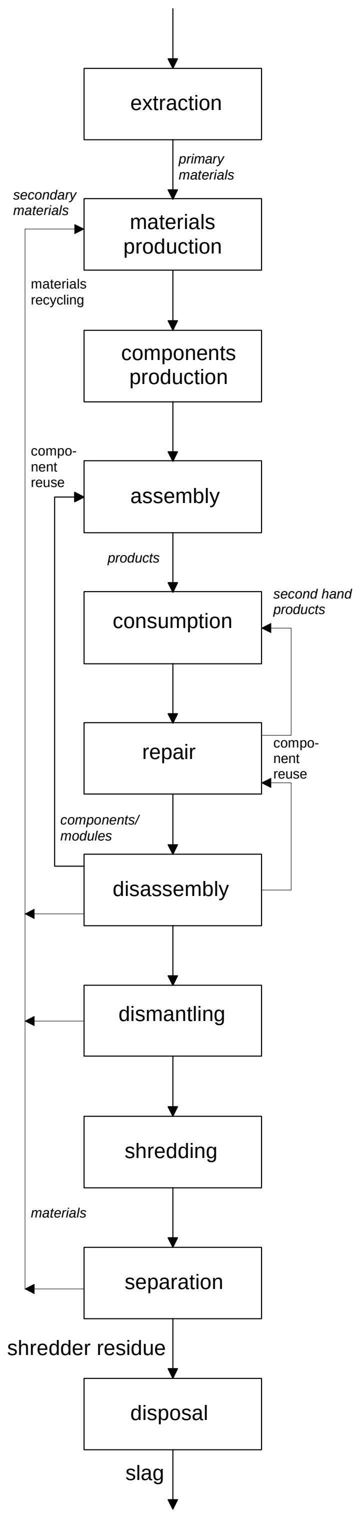

Figure 10. Product-process chain of a complex product, with reuse and recycling.

2.2. Bulk recycling

Because disassembly is labor intensive, disassembly is performed to a minor extent only in actual disassembly plants. In post-consumer car processing, disassembly of a number of components is enforced by regulation and can be made profitable as well, because the components to be removed are relatively massive compared by those that result from disassembling electronic equipment. Discarded cars and refrigerators must be drained. Cathode ray tubes are usually removed from discarded TV sets and moni-

tors. Some cables are removed prior to shredding for technical reasons. Some valuable components or modules are removed for reuse or sales, but the greater share of the product is shredded to chunks of different size, dependent of the kind of product. Self-evidently, cars are shredded to more massive chunks than mobilephones are, because of the size of the major components.
Separation is not completely selective, first of all because the chunks or grains are frequently inhomogeneous, and also because the efficiency of the separation method itself might be lower than 1. Selectivity increases if smaller chunks of grains are made, but this is at the extent of more energy consumption, more dust creation and quicker wear and tear of the shedder. A typical product-process diagram (flow sheet) of a German disassembly plant is in figure 11.

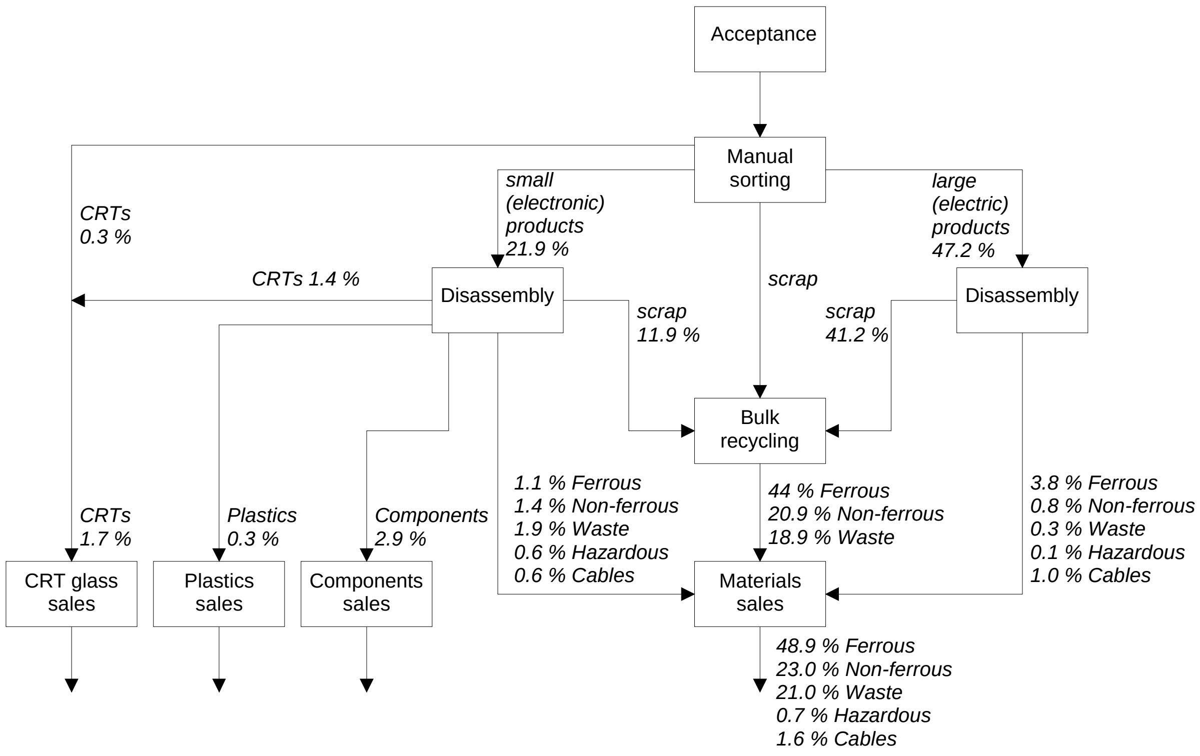

Figure 11. Flow sheet of a disassembly plant (Adapted from Spengler et al., 2003).

A mix of electric and electronic household appliances is disassembled here, and only 2.9 % by mass is sold for reuse as a component, mainly printed circuit boards and modules such as disk drives. As much as 83.8 % by mass consists of shredded and separated materials and the remaining 13.3 % is sold as a more or less homogeneous materials that is reclaimed via disassembly. This share also includes the hazardous materials that have to be separated to meet regulations, such as batteries. Disassembly prevent these from being mixed up with the materials to be sold.
As has been noticed before, the materials flows that are leaving the separation sequence are not homogeneous at all, but rather mixtures that are rich in some type of material, e.g., ferrous, or aluminum. Further upgrading is usually done in additional plants that are equipped for, e.g., metallurgical recycling. Unfortunately, much of the non-metals components are crushed and, after removal of the metals, are not recycled at all.
A typical bulk recycling sequence is depicted in figure 12.

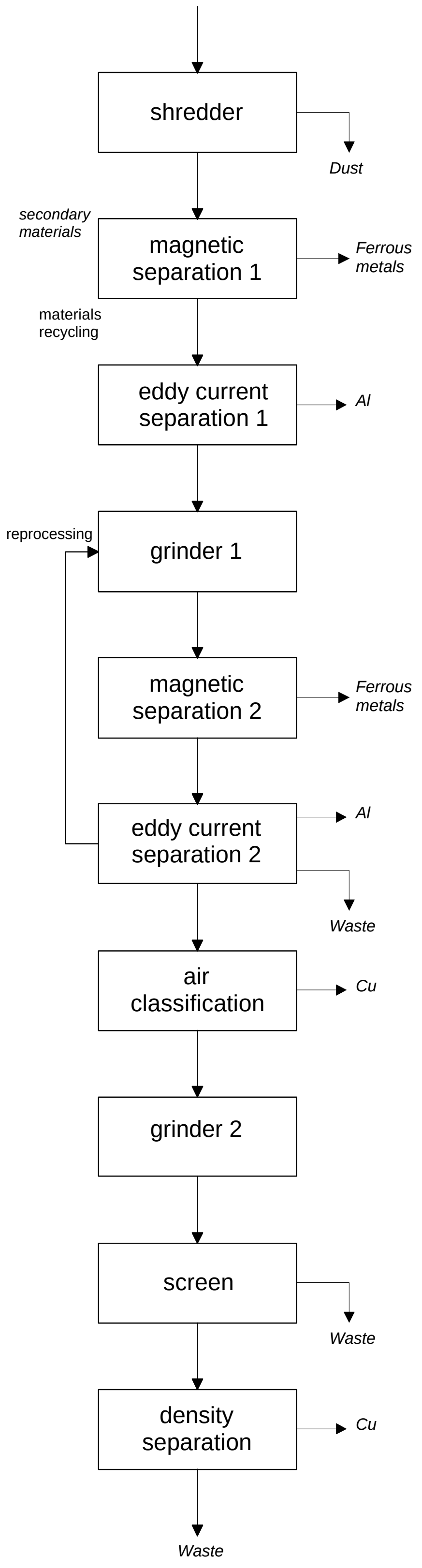

Figure 12. Bulk recycling sequence (adopted from Spengler et al., 2003).

Spengler *et al.* (2003) present a (simplified) short-term planning model in which the marginal income of the recycling plant is maximized. In their model, the authors discern 6 product types and 23 component or module types that arise from disassembly operations. The 10 bulk recycling unit operations from figure 12 are applied. Four

types of material (Fe, Cu, Al, and waste) are considered, apart from the category: 'hazardous', which is assessed via a 0,1 variable.
Selectivity of the separation steps is expressed by the separation coefficient, which is a dimensionless quantity (in kg/kg) between 0 and 1. For instance, magnet 1 separates 75% by mass of the ferrous materials present in the stream.
The materials breakdown of the product is also given, e.g.: 3% Cu, 0% Al, 13% Fe, and 85% waste for a typical TV set. The mass of one item of the product is also given. For a TV set this is 25 kg. Apart from this, it is given that a TV set contains hazardous materials, which are in the cathode ray tube, viz., the fluorescent powder.
The disassembly operations that are supported by the model are simple. There are only 10 of these in the model. For a TV set, 'removal of the cathode ray tube' is the only possible disassembly operation. The CRT accounts for 50% of the mass and consists completely of 'waste'. Besides this, it also contains all the hazardous materials that are present in the TV set.
A maximum is put to the disassembly labor time, to the capacities of the different shredding and separation steps, and to the stock capacity. Apart from this, a limit to the demand has to be accounted for. Extra constraints that originate from the compulsory removal of hazardous substances can be added.
The objective function consists of the marginal profit, which is composed of the difference between yield and costs. The yield comes from components and materials that will be sold, and from the acceptance fee of the discarded products. The costs are caused by the disassembly and bulk recycling operations, and by the disposal costs of the wastes that are produced.
Most of the variables are real variables. The numbers of every type of disassembly operation that must be executed are integer variables. Therefore, the optimum short term planning problem of the recycling plant can be modeled and optimized by a mixed integer linear programming problem.

Example:
One has a mix of TV sets and PCs. The mass of a TV set is 25 kg. The mass of a PC is 10 kg. The cathode ray tube has to be removed from the TV set; the battery has to be removed from the PC, because of the presence of hazardous substances. Both the tubes and batteries are externally processed. The recycler has to pay a fee for each of these items.
The mix of TVs without CRT and PCs without batteries, is shredded and, subsequently, is subjected to the sequence of separation processes in figure 12.
The (simplified) sequence of processes is as follows:

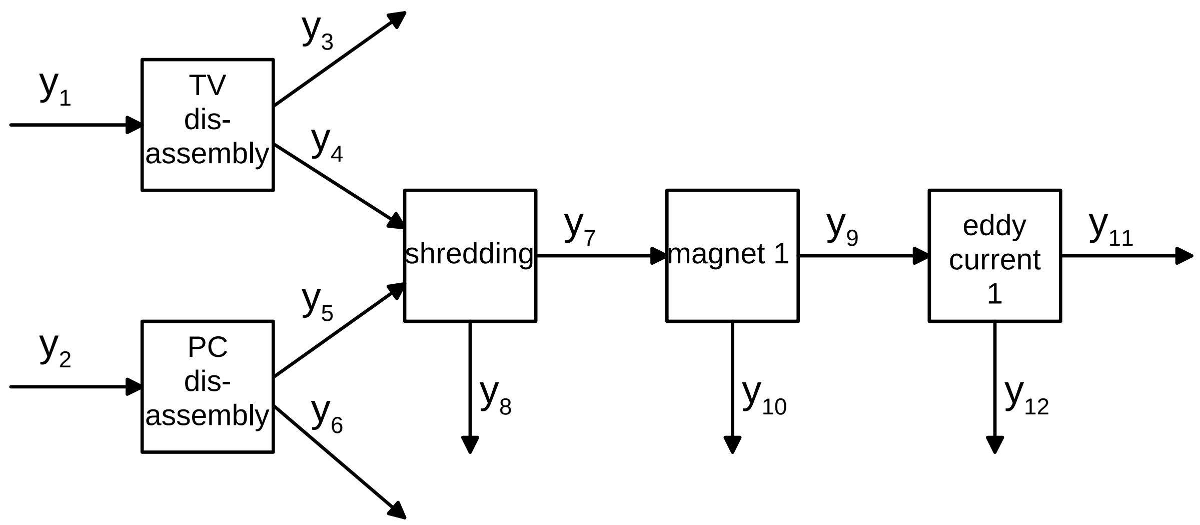

Figure 13. The first processes in a bulk recycling model.

For demonstrating the model, we have only two types of product, TVs and PCs. These are referred to by the indices $i = 1$ and 2, respectively. There are 4 relevant materials: Cu, Al, Fe, and waste. These are referred to by the indices $j = 1$ thru 4.
The model consists of three, connected, parts:

Part 1: The continuous, technical, part:
The share of each material in each product is represented by the matrix elements a_{ij}. Self-evidently, the elements of every row add to 1. The following matrix elements are given:

$a_{11} = 0.03$; $a_{12} = 0$; $a_{13} = 0.13$; $a_{14} = 0.85$
$a_{21} = 0.15$; $a_{22} = 0$; $a_{23} = 0.47$; $a_{24} = 0.38$

Because half the mass of the TV set is due to the CRT, and 15% of the mass of a PC unit is due to the batteries, we have:

$y_3/y_1 = 0.5$; $y_6/y_2 = 0.15$

Both the CRTs and the batteries are considered waste, hence:

$a_{31} = 0$; $a_{32} = 0$; $a_{33} = 0$; $a_{34} = 1$
$a_{41} = 0$; $a_{42} = 0$; $a_{43} = 0$; $a_{44} = 1$

Each of the separation steps is characterized by a separation coefficient, such as:

$y_8 = 0.3 \cdot (a_{44} \cdot y_4 + a_{45} \cdot y_5)$

and, consequently:

$y_{84} = 1$,

which represents the dust, produced by the shredder.
Similarly:

$y_{10} = 0.75 \cdot a_{73} \cdot y_7$

and, consequently:

$y_{73} = 1$

which represent the metal fraction, separated by the magnetic separator.
Obviously, the coefficients 0.3 and 0.75 refer to the selectivity of the process.
Capacity constraints put a limit on: y_1, y_2, (y_4+y_5), y_7, y_9, etc.
Node equations are: $y_1 = y_3 + y_4$; $y_2 = y_5 + y_6$; etc.
Partial node equations account for the conservation of each component, such as:

$a_{1j} \cdot y_1 = a_{3j} \cdot y_3 + a_{4j} \cdot y_4$, etc.

Part 2: The discrete part
If the number of products that have to be disassembled is large, we can use a linear programming approach. We then can rely on real numbers. E.g., if the solution reads that 2567.5 kg of TV sets have to be disassembled, and the mass of a TV set is 25 kg, we need 102.7 TV sets, which must be rounded to an integer. With the masses of the products given: m_1 = 25 kg, and m_2 = 10 kg, we define x_1 and x_2 as integer variables, which refer to the number of items of each product type to be disassembled. The connection between integer and real variables is given by:

$$y_1 = m_1 \cdot x_1; \quad y_2 = m_2 \cdot x_2$$

The time that is required for disassembly operation i is t_i, hence:

$$x_1 \cdot t_1 + x_2 \cdot t_2 \leq T$$

with T the maximum disassembly time available.
Disassembly is carried out by human labor, which is flexible.

Part 3: The economic part
Profit maximization is the objective here. The profit is composed of the following components:
1. Acceptance price e_i of the scrap products, in €/kg,
2. Purchase price e_{ij} of material j from flow i, in €/kg,
3. Labor cost in €/h for performing disassembly operations.
4. Process cost in €/kg for bulk recycling processes.

Obviously, both the acceptance and purchase prices can be positive as well as negative, in case a fee has to be paid. The model is easily expandable and can be used in multiple situations.
The model by Spengler et al. (2003) is more complex. There are more products to be disassembled, as well a multiple disassembly and assembly operations.

A continuous model that describes the recycling of reels that carry tapes with Surface Mounted Devices that have to be mounted by pick-and place machines, is discussed by Lambert et al. (2004).

3. Selective disassembly

3.1. Assembly and disassembly

Once we have discussed some topics of bulk recycling, we return to the disassembly processes. In the bulk recycling model, disassembly was added, albeit in an elementary way: the disassembly process has been considered fixed. A TV set is disassembled in one way, and alternative ways how, and to what extent, to disassemble it have not been considered yet.
We have already seen that disassembly is used for many purposes, both for component and module reuse, and for materials recycling. Disassembly is often applied for improving those processes that occur downstream of the chain, e.g., by removing cables that might obstruct the shredder, or by decreasing the amount of, hardly separable, shredder residue.
Scientific approach of disassembly processes emerged from assembly research (Bourjault, 1984). He considered disassembly as reverse assembly, but he also noticed the difference between assembly and disassembly. First of all there are *technical* differences. Disassembly is reverse assembly only if it is assumed that the components are *rigid bodies* (rigid body approach) and also that internal and external forces are absent. Forces in a product indeed result from deformation. Other forces, such as gravity, are assumed to be absent as well, so stability issues are not included in the initial consideration. Accessibility is another topic: if a piston is placed in a cylinder, it moves to the bottom via gravity, but it is difficult to remove the piston, without applying an operation that is different from the reverse 'assembly' operation, such as turning the cylinder upside down, or using a different tool.
In the rigid body approach, the components are characterized by their *dimensions* (*geometry*) and their *position* (*topology*) only.
In the sequel we will investigate how to deal with these technical issues. We have to notice that many assembly operations are not reversible at all.
Apart from technical issues, there are economic ones. Assembly is aimed at obtaining the complete product, which in general has a considerable value. Disassembly, apart from that, which is carried out for maintenance or repair purposes, is usually not carried out completely. Complete disassembly is not only technologically impossible in most of the cases, but it is also economically infeasible, even if we restrict ourselves to reversible operations such as loosening the bolts.
The main problem in assembly is determining what assembly operations should be carried out, and what sequence of operations has to be applied for obtaining the product. The reverted problem is complete disassembly.
In end-of life disassembly, the principal problem is extended with the *disassembly depth* as an additional decision variable. We speak of incomplete or *selective disassembly* if the disassembly process is not accomplished to its full extent.

3.2. End-of life disassembly

Usually we carry out the disassembly process aimed at profit maximization. Because disassembly is a divergent process, we have multiple outputs, which consist of components and modules. and possibly a remaining object, or *carcass*, that consists of multiple components but is not a module, as it has no functionality left. This carcass is usually shredded. In the example of the preceding section, the 'PC without batteries' is shredded.
The components and modules can either be reused as such, or they can be recycled as materials. The reason that these components nevertheless are disassembled might be

that these components are almost homogeneous and well-defined, such as bumpers of cars and the casings of PCs. Besides that, the materials will not appear in the shredder residue, from where it is difficult to isolate them.
Usually, some components and modules are valuable, but the demand is usually limited. This is the case in the recycling plant, where some PCBs are disassembled for resale. Also in used car processing, some components, but not all of them, are disassembled for use as repair part, so far demand exists.
We will give a review of the purposes of selective disassembly below:

- Reuse purposes
    - Valuable module or component: engine, PCB, lens.
    - Compulsory (hazardous): battery.
- Direct recycling purposes
    - Isolation of precious materials rich components: connectors.
    - Valuable, well-defined, nearly homogeneous materials: casing, bumper.
    - Compulsory (hazardous): working fluids, cathode ray tube.
- Indirect recycling purposes (reduction of shredder residue)
    - Car's interior, wooden TV set casing, concrete from washing machine.
- Technical purposes
    - Components that obstruct the removal of desired components: casing.
    - Components that damage shredder operation: cables, cast iron.

Typical precious materials are gold, silver, platinum, and palladium. These are applied in minor quantities to improve electric conductivity, mainly in connectors, and more in professional than in consumer products. Some prices of materials and components are listed in Table 3.1. These have been taken from various sources, which are indicated by the abbreviations.

Table 3.1. Prices for materials scrap from various sources, in \$/kg.

Material	Åk	Smi	Gup	Das	Chen	Stu	Lu	Dini	Sodhi	RecW	SpM	Gao
Ferrous	0.02	0.04	0.11	0.045	0.11	0.05	0.11	0.1	0.10	0.085		0.02
Stainless steel		0.33		0.51/0.22	0.78					0.6		0.23
Non-ferrous			0.88			0.62						
Al	0.60	1.89		1.08/0.42				0.61	1.06	1.57		0.49
Cu	1.25	0.18		2.20/0.45	0.79			0.31	2.16	2.25		
Brass					0.96[1]					1.15		
Ni					2.78				4.92			
Pb					2.26				0.46	0.42		
Zn					0.37				1.06	2.20		
Sb					1.34				5.18			
Precious					174.00							
Au									8,566.00		10,200	
Ag									75.80		143	
Pd									11,065.00		10,200	
Pt											19,000	
Plastics	0.05			0.045				0.02	0.22	0.02		0.11
ABS		0.33										
PC										0.57		
PE					0.30							
PS		0.04			0.36							
POM		0.04										
PVC					0.24							
Mix				-0.24		-0.05						
Foam				0.00	0.08							0.06
Glass								0.15				
CRT glass		-0.25				-0.19	-0.19			-1.00		
PWBs		0.08		2.07		1.67		0.00		2.20		0.50
Cables				0.40		0.11						0.18
Batteries				-0.50						-1.65		
Waste				-0.11								
PC										-0.02		

Obviously, prices are fluctuating. Also of interest is the *elemental breakdown analysis* of some typical products. Virtually every chemical element is present in an electronic product. The elemental breakdown is performed via an *analytical method* that is carried out in a laboratory and takes place after a product is grinded to fine powder, and subsequently a sample is taken, brought into solution, and analyzed by spectral analysis. This can be used, e.g., if the chemical composition of PCBs has to be determined. Table 3.2 presents some results from various authors.

Table 3.2. Elemental breakdown of PCBs.

	Sodhi	Brodersen	Angerer	Angerer
Material	Content (%)			
Silica	30.2	49	-	-
Plastics	30.2	19	-	-
Bromine	-	4	2.7	-
Iron	8.1	6	10.8	5-10
Copper	20.1	7	3.7	10-20
Aluminum	2	-	4.8	1
Tin	4	1	3.1	2
Nickel	2	3	0.32	1-3
Lead	2	-	-	1-5
Zinc	1	2	1.45	0.3
Silver	0.2	-	0.08	0.05-0.3
Gold	0.1	-	0.01	0.0003-0.001
Manganese	-	-	2.15	-
Antimony	-	-	0.45	-
Barium	-	-	0.36	-
Chlorine	-	-	0.19	-
Sodium	-	-	0.18	-
Chromium	-	-	0.16	-

[1] Alloys included

Cadmium	-	-	0.04	-
Tantalum	-	-	0.02	-
Palladium	0.005	-	-	0.004-0.003
Other metals	-	9	-	-

Obviously, silica and plastics, which are applied for the support and the housings of components, are not further analyzed, as these do not represent any valuable substance.

The materials breakdown analysis proceeds via a *technical method*, simply consisting of disassembling the product and weighing the different more or less homogeneous components. Obviously, one is left with alloys, various plastics, etc., from which the composition remains unclear to some extent. Apart from this, one discerns some types of complex component or module, such as PCB, switch, etc., that are not disassembled to their full extent. The composition of these classes of modules can be estimated from tables. A materials breakdown of some electronic products can be found in Table 3.3.

Table 3.3. Materials breakdown of some grey goods.

Material	M1	M2	Pr	C1	C2	S	K	Te	Mp
	Content (%)								
Ferrous metals	25	15.4	39	42	232	19	423	314	3
Non-ferrous metals									
Cu	4	8.5	13	1		1			15
Al		5.1		11		4			75
Light fraction									
Plastics	46	17.6	36	17	23	41	31	40	49
Glass	21	42.5		13					
PWB	(23)	10.6		16	10	25			
CRT unit					29				
Cables					5				
Other	4		10			10	27	29	256
Product mass (kg)	8	21.4	7			3.67	1.643	0.615	0.154

M1	Monitor
M2	Monitor 17″
Pr	Matrix printer
C1	PC configuration with system, monitor, and keyboard
C2	PC configuration with system, monitor, and keyboard
S	PC system
K	PC keyboard
Te	Telephone set
Mp	Mobilephone

A materials breakdown of some typical modules is in Table 3.4.

Table 3.4. Provisional data on material content of complex components.

Material	PCB	Electric Motor	Trans former	Wire Cable	Wire Cable
	Content (%)				
Ferrous metal	-	75	65	-	-
Copper	12	15	25	40	36
Aluminum	-	10	5	-	18
Plastics	70	-	-	60	45
Other	18	-	5	-	-

Obviously, the percentages of the different substances strongly vary, even if a similar component of a similar product is considered. Apart from this, there is a tendency It should be mentioned that, because of the both technological advance and regulation, the composition of the products and their parts always evolves according to some tendencies. There is, e.g., miniaturization. LCD monitors are different from cathode ray tube monitors; PCBs evolve to ever sophisticated, integrated and miniaturized modules; some substances, such as mercury in relays and asbestos in washing machines, are banned, etc.

We conclude by listing some of the hazardous substances that might be present in electronic scrap. We discern elements, such as Cadmium, and compounds, such as polychlorinated biphenyls, which are carcinogenic, see Table 3.5.

[2] With Al and Cu included
[3] All metals included
[4] All metals included
[5] Ni, Zn, and Ag included
[6] 9% epoxy and 16% ceramics

Table 3.5. Hazardous substances in electronic products.

Substance	Application
Heavy metals	
- Cd; Ni; Zn; Pb; Hg	Batteries; fluorescent tubes
- Sn; Pb; Cd	Solder
- Ba; Sr; Pb	CRT glass
- Cd; Y; Eu; Se; Zn	Fluorescent powder
- Hg	Relays
Semiconductors	
- B; Ga; In; As	Integrated circuits
- GaAs	LEDs; photovoltaic cells
- Se; Ge	Diodes
- Se	Photocopying drums
Organic compounds	
- PCBs (polychlorinated biphenyl)	Capacitors
- PBDEs	Flame retardants
- Mineral oil	Lubricant
Additives in plastics	
- Cl	PVC
- Cd; Pb; Ni; Ti; Sb; Diazo compounds	Pigment
- Pb; Ba; Cd; Sn	Stabilizer in plastics

3.3. The disassembly process

If we carry out a disassembly process in practice, we can do it in many different ways. We have to consider the structure of a product. A product is considered a *set of components*. The components are related via *connections*, i.e. physical interfaces. One can consider the concept of component in different ways: one can consider those objects that were discrete entities before assembly as a component, but in disassembly this does not make much sense, as it is not always possible to separate these components from each other via disassembly or even dismounting. For this reason, we often consider larger objects, sometimes even rather complex modules as a component. We have encountered this already, in defining printed circuit boards, and comparable items, as a component.

Connections keep the components in place, although some of them are able to move with respect to some degrees of freedom. The essential of a disassembly operation is the separation of a set of connected components in two or more subsets of connected components. Obviously this includes the disestablishment of one or more connections, and moving one of the subsets (which will frequently be a single component) away from the remaining subset, which is typically hold in place via a fixture or something like this.

There are different kinds of connection. The most simple is *mating*, which is completely reversible. Many connections are irreversible, e.g., by *soldering*. Reversible connections can be established by dedicated components, called fasteners. This can be bolts, screws, etc. In practice, we therefore distinguish *fasteners* and *structural components*. Obviously, this distinction can be somewhat arbitrary. Fasteners are often small components, with a minor mass, such as bolts, and these don't need to be treated as a full fledged component. Irreversibly breaking a fastener, such as a solder connection, can be included in disassembly, for it does not destroy a structural component. In this case a wire connection is not considered structural, although it is not dedicated for mechanical fastening, but rather a *duct*. Fasteners and ducts can be collected, but usually contribute to minor extent to the material's mass to be recovered.

Intuitively, the disassembly process usually proceeds according to a *disassembly tree* structure. One loosens some fasteners, after which a module can be separated from the product. Both the freed module and the remaining product can be disassembled further, thus creating new (sub-)modules and some individual components. Sometimes a module happens to be a 'microcosms', i.e. a product as such, but on a smaller scale. This refers, for instance, to car electronic equipment, or to a disk drive in a PC. An example of such a disassembly tree is in figure 14. It refers to a monitor.

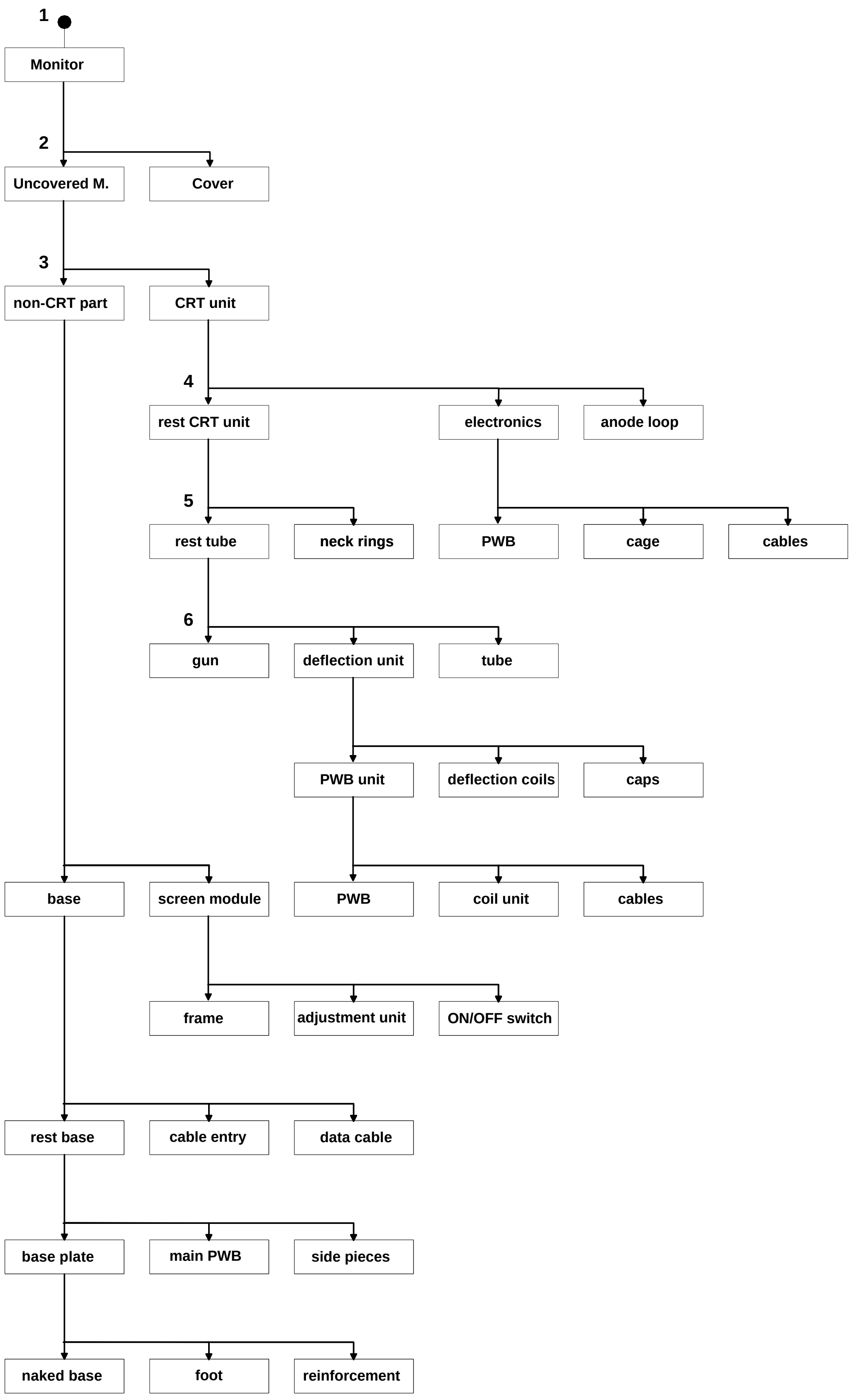

Figure 14. Disassembly tree of a PC monitor.

First some bolts have to be unscrewed, after which the cover of the monitor can be removed. This consists of several components made of engineering plastic (ABS), which are virtually homogeneous. For these components have a considerable mass with respect to the complete monitor, recycling is feasible. It should be noticed that, when the casing would be shredded with the complete product, recycling of the ABS would be hardly possible. After the cover is removed, the CRT unit is disassembled. Obviously, some wires have to be cut for this purpose. The CRT is disassembled separately. Some components of it, notably the deflection units, are glued to the neck of the tube. The neck has to be broken for release of the vacuum, which is considered a dismounting operation. The gun, made from an alloy rich in nickel, is recovered for recycling. The tube, which accounts for about half of the monitor's mass, is also recycled.
The disassembly tree of the type that is depicted in figure 14 is frequently encountered in industry. Planning problems that have similar structures as a basis, are discussed in section 10 of this course.

3.4. Mass distribution of components

If we disassemble an end-of product, we are left with a set of disconnected components, each with a different mass. Intuitively, we suppose that there are few components with a large mass, and many with a small mass. German researchers have investigated the mass distribution of the components used in car assembly, aimed at subdividing these components in a number of categories. They made this for selecting some representative component types in order to accomplish a Life Cycle Assessment study of a car. According to their findings, the mass distribution function approached a hyperbola.
End-of life car disassembly is essentially different from assembly, as it is focused on nonferrous parts, apart from the power train (engine, gear box, etc.) which is often disassembled as a complete module. A disassembly experiment on cars has been carried out in The Netherlands, where a batch of test cars was disassembled up to different disassembly depths. Costs, expressed in disassembly time, and the mass of the released components was monitored. Some of the experiments intended to disassemble and dismantle as deep as possible, leaving the body with some coating, the paint, and the dirt attached to it, and some remaining components welded to it, as a carcass. As expected, components that were heavy and/or easily removable were disassembled first. Every next disassembly operation resulted in slightly increased costs and/or a slightly lower yield, expressed in mass. This because disassembling a non-metallic component resulted in a decrease in the amount of shredder residue caused by shredding the remaining carcass. Actually, a plot of the cumulative disassembled mass against the cumulated disassembly time shows a saturation curve, see figure 15 for the car example.
We must keep in mind that, for car recycling, the following regulations are into force:

- Harmful substances and components must be isolated. According to this, the car must be drained, batteries must be removed, airbags must be made harmless.
- A list of specific components must be removed. This might include: tires, car glass, bumpers, etc.
- A minimum recycling rate must be attained. This becomes increasingly difficult as the share of nonmetallic substances in cars gradually increases.

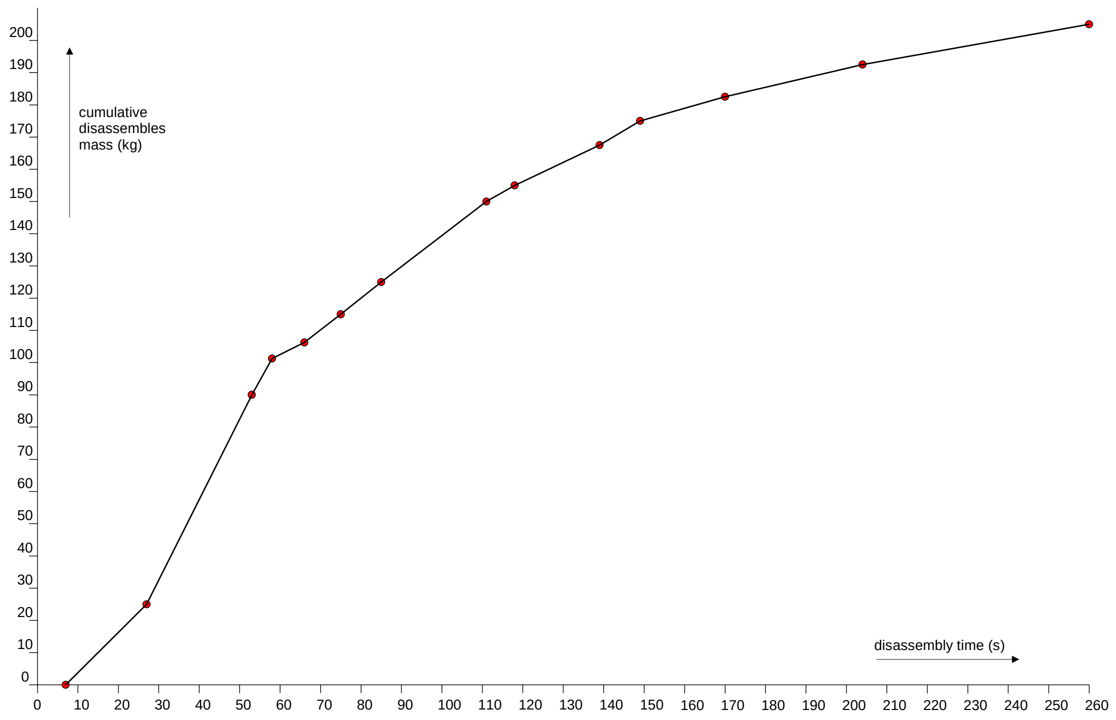

Figure 15. The amount of disassembled nonmetal components in kg, as a function of the cumulative disassembly time in sec.

Figure 16 depicts the mass distribution of non-metallic components in a car. Notice that some of the components are aggregated, e.g., the component with mass 26 kg represents the aggregate mass of tires, tubes, hubcaps, and wheels.

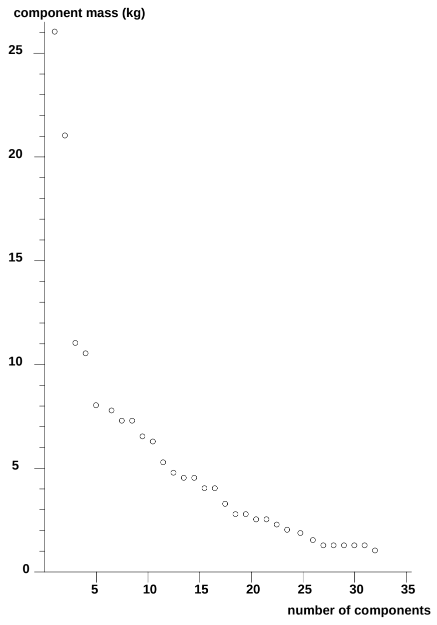

Figure 16. Mass distribution of non-metallic parts ≥ 1 kg, in a car with m = 1,070 kg.

Disassembly experiments resulted in the following mass distribution plots. The sensitivity of the weigh scale was 1 g here:

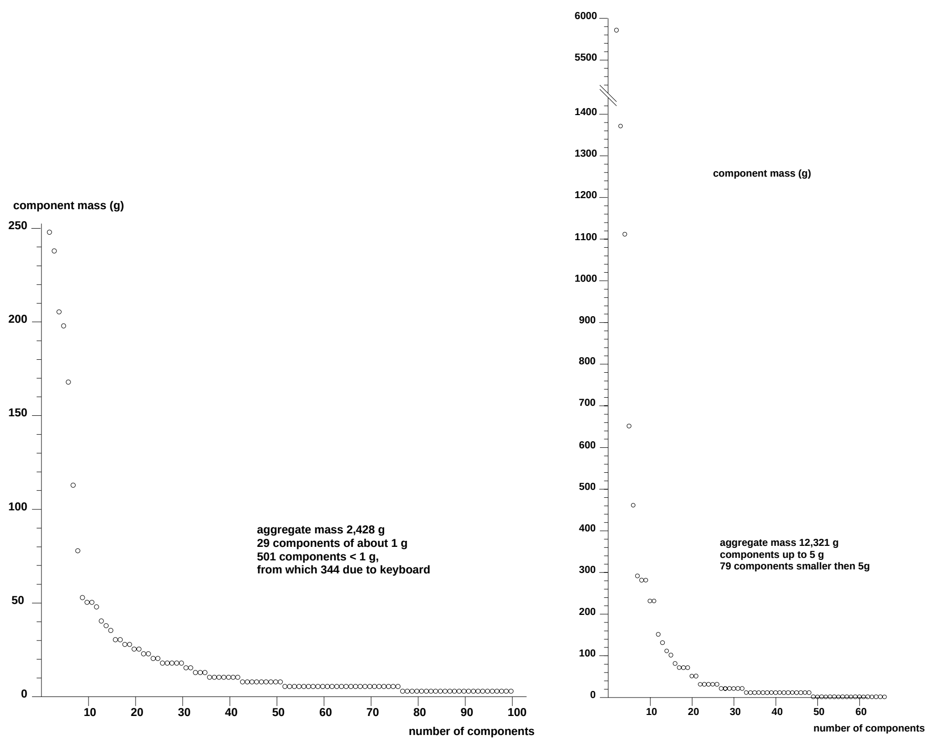

Figure 17. Mass distribution plots of a notebook PC and a monitor.

The tail in the plots is due to the fasteners that represent many components.
In the notebook there are 99 bolts. Apart from this, we encounter 84 springs, 85 washers, etc. The total mass accounts for 46 g, which is hardly 2% of the notebook's mass. The most massive components are the 8-battery pack (431 g), the bottom casing (223 g), the motherboard (190g), which is a non-disassemblable module, the HD drive (189 g), which is a disassemblable module, and two other parts of the casing (189 g and 179 g, respectively). The three casings parts, made of PC+ABS engineering plastics, make out 25% of the mass.
In the monitor, there are 37 bolts and 4 major screws, 4 spacers, 9 clips, and 5 nylon cable bundlers, accounting for 76 g or less than 1% of the monitor's mass. The most massive components are the naked, which is a non-disassemblable module and account for 42% of the total mass. The three major casings parts (649 g, 462 g, and 282 g) account for 11% of the mass. These components are made of ABS and recyclable. The deflection coils add to 4% of the mass. These form a non-disassemblable module, made of copper, plastics, and ceramics, with 31% copper contents, which is a rich 'ore' indeed.
Comparison of the three types of product discussed so far reveals that products come in different scales. Many components of a car are much more massive than a small electronic product as such. This puts a serious limit on the disassembly depth, as the yield of individual components tends to decrease at smaller mass, while the disassembly costs hardly decrease. For appropriate disassembly, some modular structure that is easily to take apart, is advisable in the product design.
The hyperbolic function can be presented in an alternate way, in which the number of components is plotted on a logarithmic scale. In this case, the components with an intermediate mass are plotted on a line that appears approximately straight. This applies for a broad range of products and is robust, even if some of the components are aggregated. Figure 18 depicts some plots for a tumbler dryer, a monitor, and a laser jet printer. We observe major deviations of the straight line on the low-mass end of the

plot, where an excess of low-mass components are present, and on the high-mass end, because there are only a few of them, so that noise plays a dominant role here. Notice that the plot is an empirical one, and does not result from a strict 'law of nature'.

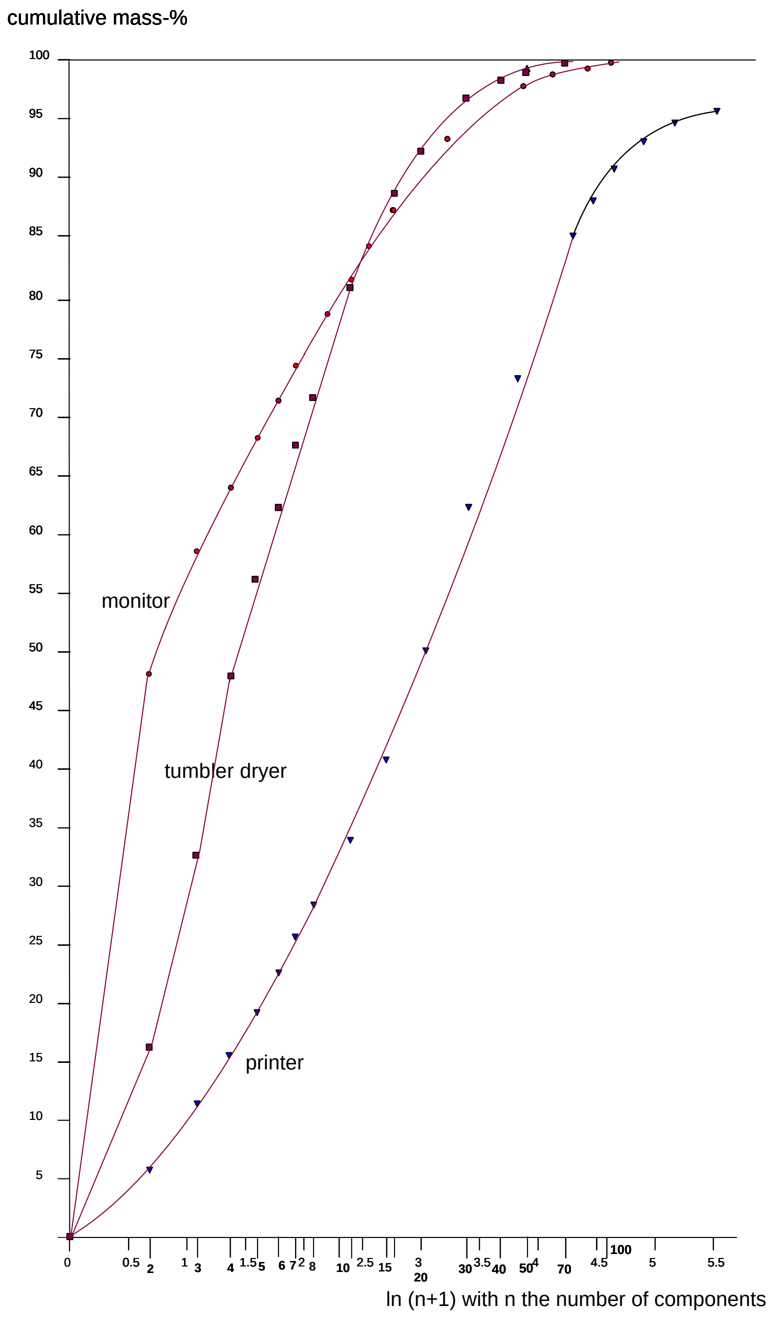

Figure 18. Cumulative mass of three selected products, plotted vs. number of coefficients.

One of the most interesting conclusions of this plot is the fact that the slope of the plot tends to increase with decreasing product complexity. It is even so that if the same product is considered, but with some additional aggregation, the straight line remains approximately intact, but at a steeper slope.

4. Graphical Representation of Disassembly Processes

4.1. Introduction

We observed that disassembly processes can often be described as a hierarchical tree. Hierarchical tree representation is particularly useful when a product is mainly considered a mine of components. In many products, the hierarchical tree does not include all the possible disassembly sequence, but only part of it.
Particularly in mechanical products, one must account for the geometrical structure of the product into more detail than is usual in the hierarchical tree. This is also of interest in assembly issues that have been at the basis of systematic disassembly theory.
As has already been mentioned, systematic disassembly theory starts with including the *rigid body assumption*. This implies that every component of the product is considered rigid and non-deformable, even when this is not valid in practice. Both internal and external forces, such as gravity, elasticity, and friction are considered absent as well. Although this is obviously in contrast with reality, the rigid body assumption is often very useful for a first analysis of a disassembly problem. Force-related issues can be added to the problem in a later phase of the analysis.
We thus restrict ourselves to both the topology and the geometry of the product. The *topology* reflects whether or not the components of the product are connected, without going into detail on the exact nature of the connection. *Geometry* is related to the obstruction of disassembly operations by the presence of some components. Early disassembly theory deals with an assembly problem. Usually, there are multiple ways of assembling a finished product from its components. However, if one starts with an arbitrary sequence of adding components, many of these attempts result in a *deadlock*. This occurs if the already assembled components inhibit the addition of a subsequent component. Every motorcycle amateur has been confronted with a 'complete', but not functioning, bike, accompanied by some components left over. The most elementary example of a deadlock is demonstrated with the 'devil in the box', consisting of the box (A), the lid (B), and the devil-with-spring (C) (the spring is being considered contracted, which can be assumed under de rigid body approach). Mounting the lid on the box leaves us with the devil and a product that does not cause any fear to the audience. This ordering is represented by the infeasible assembly sequence A-B-C.
If one considers disassembly, the reverse sequence C-B-A is excluded a priori, because removing the devil first is obstructed by the lid. Therefore, deadlocks are a priori avoided here, and feasible sequences only are considered. The action to be taken next is reverting the sequences found by this method. After this, one must consider the non-realistic implications of the rigid body approach, thus further excluding some sequences. For example, the assembly sequence C-B-A will be not be a wise choice from a technical point of view.

4.2. Topological constraints

Usually, a product is represented by assembly drawings, virtual models, bills of material, product specifications etc. Formally, a product can be considered a *set of components*. These components are related to each other. The relationship that is apparently the most essential in disassembly theory, is the physical connection. Topology is involved in the structure of the connections.
A *subassembly* is defined as a connected subset of the product. This includes the product itself as well as any individual component. A *disassembly operation* is defined as a separation of a subassembly into <u>two</u> (and no more than two) complementary new subassemblies. If we consider the case in which one of these subassemblies

is an individual component, we speak of a *sequential* or one-at a time *disassembly operation*. In case, both of the subassemblies consist of multiple components, we are confronted with a *parallel disassembly operation*.
The topology of a product can be summarized in a *connection diagram*. This is a non-directed graph in which the components are the nodes and the connections are the arcs. The exact nature of a connection is arbitrary to some extent, particularly if it refers to a mating connection only. Fasteners, such as bolts, can be considered connections, not separate components, but this depends on the interest one has on the substance of the fastener, e.g., the mass, the value, or the presence of hazardous properties.
What is a component can also be subjected to the opinion of the user of the model. It might be quite possible to define a cluster of components as a separate component (supercomponent or *module*). An illustrative example is the engine of a car, which is usually disassembled as a whole.
With these considerations in mind, we will illustrate the connection diagram at the hand of Bourjault's ballpoint: an extremely simple, but classical example that has been at the basis of disassembly theory, see figure 19.

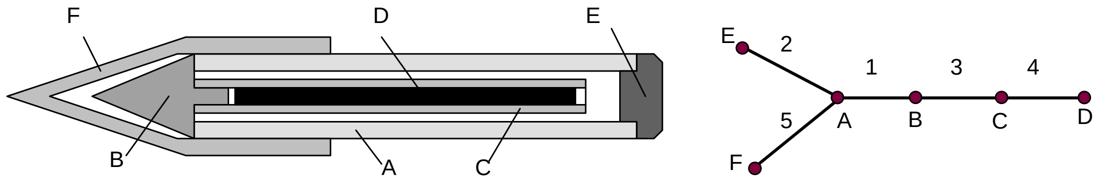

Figure 19. Simple axially symmetric example: Bourjault's ballpoint. (a) assembly; (b) connection diagram. (Source: Bourjault, 1984).

This product consists of 6 components, indicated as capitals A thru F. These are: body (A); head (B); cartridge (C); ink (D); button (E); and cap (F). The corresponding connection diagram is also depicted in the figure. In this case, the ink is provisionally considered a rigid body. Notice that B and F are not considered connected here, which is arbitrary to some extent, as the components are slightly touching each other.
The graph can be stored in computer memory via an alternate representation, the connectivity matrix, see figure 20.

$$\begin{array}{c|cccccc}
 & A & B & C & D & E & F \\
\hline
A & - & 1 & 0 & 0 & 1 & 1 \\
B & 1 & - & 1 & 0 & 0 & 0 \\
C & 0 & 1 & - & 1 & 0 & 0 \\
D & 0 & 0 & 1 & - & 0 & 0 \\
E & 1 & 0 & 0 & 0 & - & 0 \\
F & 1 & 0 & 0 & 0 & 0 & -
\end{array}$$

Figure 20. Connectivity matrix of Bourjault's ballpoint.

The rows and columns of this matrix represent the components. Apart from the diagonal elements, whose value is indifferent, the elements corresponding to a pair of connected components equal 1, the others 0. Notice that the matrix is symmetric, which

implies that the *extremely lower left triangular matrix* contains all the necessary information.
If one considers the subassemblies, one concludes from the data on connections that there are less subassemblies than there are subsets. This is because of the requirement that subassemblies must be connected subsets. Therefore, AC, AD, etc. are no feasible subsets because of *topological constraints*.
Notice that one does not need the assembly drawing for determining the topologically feasible subsets, as this can be derived from either the connection diagram or the connectivity matrix. This can completely be done automatically. This demonstrates the advantage of modeling, which is indeed the construction of a simplified map of reality, aimed at obtaining a better understanding of some aspect of reality.

Exercise: List all the topologically feasible subsets of Bourjault's ballpoint from figure 19.
Exercise: List from the axially symmetric product of figure 21, a gearbox, all the topologically feasible subsets with three components. Neglect the fasteners M thru Q. Hint: start with drawing the connection diagram.

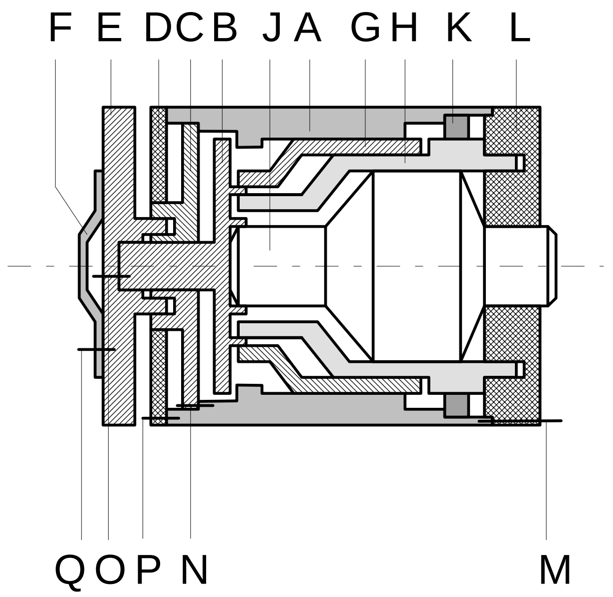

Figure 21. Automatic transmission (Source: De Fazio and Whitney, 1987).

4.3. Geometric constraints

Although we were able to automatically list all the topologically feasible subassemblies, some remarks must be made. First, the number of these subassemblies increases dramatically with the complexity of the product considered. Secondly, the list might include many subassemblies that can not be derived from the original product by disassembly operations only. An example is the subassembly AEF, which is connected indeed, but not realizable, unless by adding some reassembly operations to the sequence. We call this type of subassembly: *geometrically infeasible*, because some components are present that inhibit the assembly of some other component. Obviously, when assembling a product, this type of assembly appears in a deadlock. Listing of the geometrically infeasible subassemblies is an instrument that detects potential deadlocks in assembly sequences!
Unfortunately, a completely automated test of a subassembly on geometric infeasibility is not possible in the general case, although it can be done in many cases, as we

will demonstrate in the sequel. For the general case, we can minimize the number of manual interventions that have to be performed during calculation.
We will describe here the procedure that helps us with a minimum number of such interventions. An intervention is a query, put to the computer operator, which should be answered manually. There should be a minimum number of queries, and each query should be as easy as possible. It should be answered simply with YES or NO, without any complex calculation or consideration.
The method used here is the cut-set method. A *cut-set* is a set of connections in a subassembly that, when disconnected, leaves us with two subassemblies. The simultaneous disestablishment of a cut-set thus corresponds with a disassembly operation.
Within this framework, we usually speak of the separation of a *parent subassembly* in two *child subassemblies*.
Two rules will be discussed that considerably reduce the human intervention: the subset rule and the superset rule.
The *subset rule* states that, if a parent subassembly can be separated into two child subassemblies, a subset of this parent subassembly can also be separated, provided the subsets are connected.
The *superset rule* states that, if a parent subassembly cannot be separated into two child subassemblies, a superset of the parent subassembly can also not be separated in the resulting child subassemblies that are, of course, supersets of the original child subassemblies.
We will demonstrate this for the case of Bourjault's ballpoint.

The procedure is as follows:
Start with the original product, ABCDEF.
Query: ABCDEF → A + BCDEF, is this a feasible cut-set?
Answer: NO.
Because the answer is NO, select one of the simplest cut-sets that are also infeasible.
In this case, two simple, infeasible cut-sets appear, namely:

ABE → A + BE,
AFE → A + FE.

Note that we are not restricted to *connected* subsets here.
As there might be many infeasible cut-sets, we select only one of these, to avoid redundancy. Let us select ABE → A + BE. Obviously, it follows from the infeasibility of this cut-set that, e.g., ABEF → AF + BE, and ABEF → A + BEF, are infeasible too, according to the superset rule.

<u>Exercise</u>: List all the infeasible cut-sets that are inhibited by the superset rule, provided ABE → A + BE is infeasible.

Clearly, the superset rule is a very effective tool to reduce the number of queries.
Apart from this, the rule can be applied automatically.
An infeasible cut-set can be transformed into a *selection rul*e. This simply rules out those subsets that include two specific components and that don't include a third specific component. In case ABE → A + BE is infeasible, the corresponding selection rule reads: BE not A. This has to be interpreted as follows: any subassembly that includes the components B and E, but that doesn't include the component A, is an infeasible subassembly, apart from topological considerations.

Table 4.1 Selection procedure with geometric constraints.

Query		**Selection rule**	**Feasible cut-set**
ABCDEF → A + BCDEF	N	BE not A (1)	
ABCDEF → B + ACDEF	N	AF not B (2)	
ABCDEF → C + ABDEF	N	BE not C (3)	
ABCDEF → D + ABCEF	N	BE not D (4)	
ABCDEF → E + ABCDF	Y		ABCDEF → E + ABCDF (1)
ABCDEF → F + ABCDE	Y		ABCDEF → F + ABCDE (2)
ABCDE → A + BCDE	N	(1)	
ABCDE → B + ACDE	Y		ABCDE → B + ACDE (3)
ABCDE → C + ABDE	N	(3)	
ABCDE → D + ABCE	N	(4)	
ABCDE → E + ABCD	Y		(1)
ABCDF → A + BCDF	Y		ABCDF → A + BCDF (4)
ABCDF → B + ACDF	N	(2)	
ABCDF → C + ABDF	Y		ABCDF → C + ABDF (5)
ABCDF → D + ABCF	Y		ABCDF → D + ABCF (6)
ABCDF → F + ABCD	Y		(2)
ABCD → A + BCD	Y		(4)
ABCD → B + ACD	Y		(3)
ABCD → C + ABD	Y		(5)
ABCD → D + ABC	Y		(6)
ABCF → A + BCF	Y		(4)
ABCF → B + ACF	N	(2)	
ABCF → C + ABF	Y		(5)
ABCF → F + ABC	Y		(2)
ABDF → A + BDF	Y		(4)
ABDF → B + ADF	N	(2)	
ABDF → D + ABF	Y		(6)
ABDF → F + ABD	Y		(2)
BCDF → B + CDF	N	(2)	
BCDF → C + BDF	Y		(5)
BCDF → D + BCF	Y		(6)
BCDF → F + BCD	Y		(2)
ACDE → A + CDE	Y		ACDE → A + CDE (7)
ACDE → C + ADE	Y		ACDE → C + ADE (8)
ACDE → D + ACE	Y		ACDE → D + ACE (9)
ACDE → E + ACD	Y		(1)
ACD → A + CD	Y		(4)
ACD → C + AD	Y		(5)
ACD → D + AC	Y		(6)
ACE → A + CE	Y		(7)
ACE → C + AE	Y		(8)
ACE → E + AC	Y		(1)
ADE → A + DE	Y		(7)
ADE → D + AE	Y		(9)
ADE → E + AD	Y		(1)
CDE → C + DE	Y		(8)
CDE → D + CE	Y		(9)
CDE → E + CD	Y		(1)
BCD → B + CD	Y		(3)
BCD → C + BD	Y		(5)
BCD → D + BC	Y		(6)
BCF → B + CF	N	CF not B (5)	
BCF → C + BF	Y		(5)
BCF → F + BC	Y		(2)
BDF → B + DF	N	DF not B (6)	
BDF → D + BF	Y		(6)
BDF → F + BD	Y		(2)
ABD → A + BD	Y		(4)
ABD → B + AD	Y		(3)
ABD → D + AB	Y		(6)
ABF → A + BF	Y		(4)
ABF → B + AF	N	(2)	
ABF → F + AB	Y		(2)
ABC → A + BC	Y		(4)
ABC → B + AC	Y		(5)
ABC → C + AB	Y		(6)

Let us investigate how to proceed when some query is answered by YES, e.g.:
Query: ABCDEF → E + ABCDF, is this a feasible cut-set?
Answer: YES.

Because this cut-set is feasible, cut-sets such as ABDEF → E + ABDF are feasible too, according to the subset rule. Therefore, we list the appropriate feasible cut-set and conclude from this the feasibility of multiple additional cut-sets.
Once this in mind, we will proceed by solving the problem mathematically. This is illustrated in table 4.1:

<u>Exercise</u>: List all the possible subassemblies of Bourjault's ballpoint, without regarding topological constraints, and skip those that violates the selection rule: BE not A. Notice that the number of subassemblies equals: $2^N - 1$, with N equaling the number of components. For Bourjault's ballpoint, $N = 6$.

By this calculation we did not account for topological constraints. Geometric constraints are condensed in 6 selection rules. Apart from this, a list of 9 feasible cut-sets is compiled. There are 15 queries that should be answered by human intervention, regarding for the product geometry. The other queries are answered automatically, via superset (column 3) or subset (column 4) rules. The figures between brackets refer to the relevant selection rule that is violated, or the feasible cut-set that is applied. Although this list is fairly long, we can reduce it via eliminating the cut-sets that result in disconnected subassemblies. These are shaded in the table. By this, the extent of the calculation is considerably reduced, see Table 4.2.

Table 4.2 Selection procedure with topological and geometric constraints.

Query		Selection rule	Feasible cut-set
ABCDEF → D + ABCEF	N	BE not D (1)	
ABCDEF → E + ABCDF	Y		ABCDEF → E + ABCDF (1)
ABCDEF → F + ABCDE	Y		ABCDEF → F + ABCDE (2)
ABCDE → D + ABCE	N	(1)	
ABCDE → E + ABCD	Y		(1)
ABCDF → D + ABCF	Y		ABCDF → D + ABCF (3)
ABCDF → F + ABCD	Y		(2)
ABCD → A + BCD	Y		ABCD → A + BCD (4)
ABCD → D + ABC	Y		(3)
ABCF → C + ABF	Y		ABCF → C + ABF (5)
ABCF → F + ABC	Y		(2)
BCD → B + CD	Y		BCD → B + CD (6)
BCD → D + BC	Y		(3)
ABF → B + AF	N	AF not B (2)	
ABF → F + AB	Y		(2)
ABC → A + BC	Y		(4)
ABC → C + AB	Y		(5)

Apart from the strongly reduced amount of automated calculation, the number of queries that is put to the operator is reduced from 15 to 8.

From this table, it is evident that the following subassemblies can be obtained via sequential disassembly:
ABCDEF, ABCDE, ABCDF, ABCD, ABCF, ABC, ABF, BCD, AB, AF, BC, CD, as well as the individual components.

Although this analysis often reveals most of the subassemblies, it is not necessarily complete. Parallel disassembly should also be considered. This is reflected in Table 4.3 for Bourjault's ballpoint, regarding for both topological and geometric constraints.

Table 4.3 Selection procedure with topological and geometric constraints, parallel disassembly.

Query		**Selection rule**	**Feasible cut-set**
ABCDEF → CD + ABEF	N	(1)	
ABCDE → AE + BCD	Y		ABCDE → AE + BCD (7)
ABCDE → CD + ABE	N	(1)	
ABCDF → AF + BCD	N	(2)	
ABCDF → CD + ABF	Y		ABCDF → CD + ABF (8)
ABCD → AB + CD	Y		ABCD → AB + CD (9)
ABCF → AF + BC	N	(2)	

This table reveals that three extra operations are possible, each including parallel disassembly. Three additional queries are put to the operator and no new subassemblies appear by this extra calculation.
In more complex products, such as the automatic transmission of Figure 21, relaxation because of the topological constraints is less pronounced, because there are more connections per component on the average. For a *weakly connected* product, there are $N-1$ connections. Bourjault's ballpoint is an example of a weakly connected product.

4.4. Graphical representations

There are three methods of graphically representing disassembly processes:

- State diagram
- AND/OR graph
- Disassembly precedence graph (DPG)

All of these are directed graphs. The arcs in these graphs represent disassembly operations in the state diagram and the AND/OR graph, and precedence relationships in the DPG.
The nodes in these graphs represent partitions or states in the state diagram, subassemblies in the AND/OR graph, and disassembly operations in the DPG.
Many authors also use *disassembly Petri nets,* which are related to the AND/OR graphs.
We will briefly discuss the different types of representation at hand of Bourjault's ballpoint example:

4.4.1. State diagram.

Let a product be a set of connected components, such as ABCDEF. The disassembly state of the product is a *k*-partition of this set, with $1 \le k \le N$. An example is the 3-partition: AC,BDE,F. Obviously, a *k*-partition is obtained after $k - 1$ disassembly operations. A disassembly operation is a transition from a *k*-partition to a $k + 1$-partition. The state diagram is organized such that all the feasible *k*-partitions are at the same row.
For Bourjault's ballpoint, the state diagram is depicted in figure 22.
It is evident that every complete disassembly sequence is depicted in this diagram by a trajectory from state ABCDEF to state A,B,C,D,E,F. There are multiple ways to arrive at that state. Disassembly sequencing is the art of searching the optimum path. Incomplete disassembly proceeds along similar lines, but it ends in another state than the completely disassembled state. If both complete and *incomplete disassembly* is allowed, the number of possible trajectories and, consequently, the number of possible sequences increases still more. In the subsequent subsection, we will introduce some methods for evaluating the number of sequences.
Because of the typical appearance of the state diagram, it is sometimes nicknamed: *diamond diagram*.

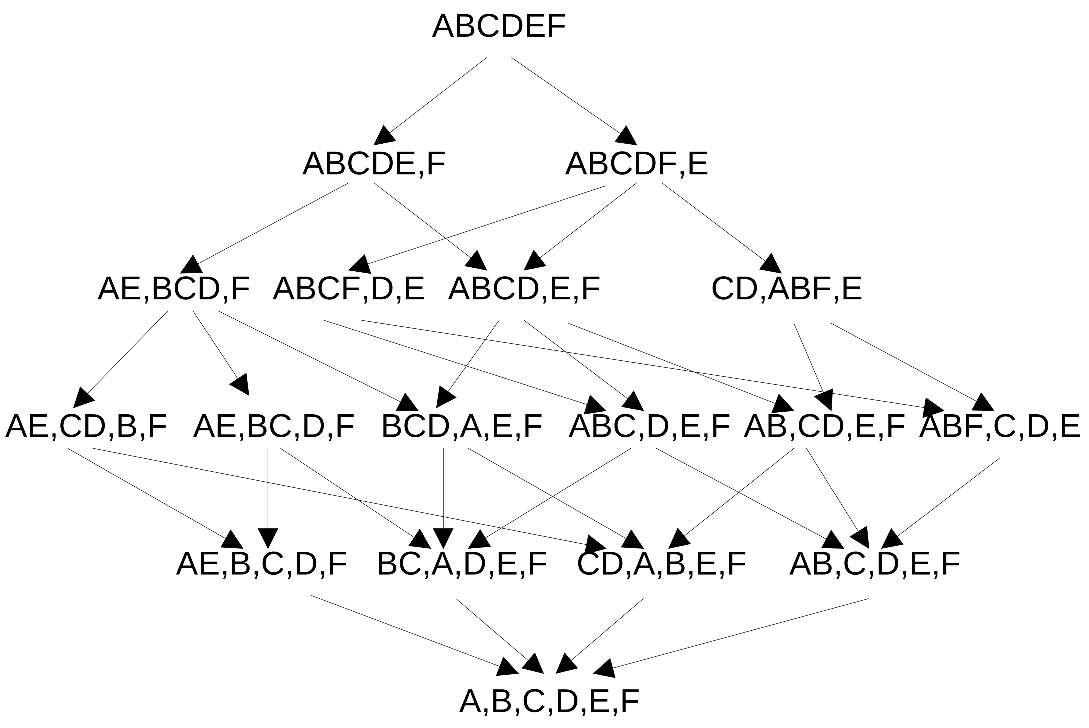

Figure 22. State diagram of Bourjault's ballpoint.

4.4.2. AND/OR graph

Although the state diagram presents a clear picture of the disassembly process, the number of states typically increases steeply with the complexity of the product. A more condensed way of notation is by the AND/OR graph. In this notation, the nodes represent subassemblies rather than states. When a parent subassembly is subjected to a disassembly operation, two child subassemblies are created. Therefore, a disassembly operation is represented by a *hyper arc*, which consists of two connected directed arcs, pointing from the parent subassembly to both child subassemblies. As the child subassemblies are complementary with respect to the parent subassembly, we might confine ourselves to depicting only one edge of the hyper arc, which reduces the graph's complexity, see figure 23.

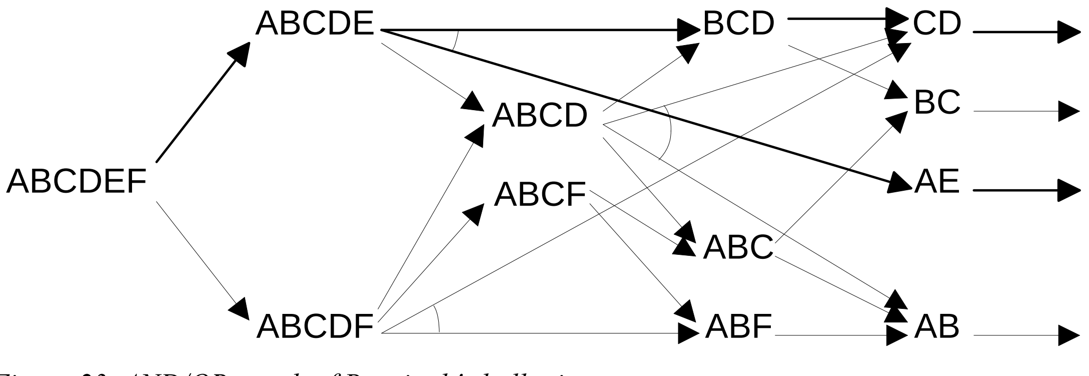

Figure 23. AND/OR graph of Bourjault's ballpoint.

We see here the same product as in figure 22, and we also observe that a disassembly sequence is depicted as a divergent subgraph, see the highlighted arrows. The AND/OR graph's name is due to the occurrence of both AND relationships, which are represented by the 2 connected arcs of each hyper arc, and OR relationships, which are represented by multiple incoming arcs, such as can be seen, e.g., at subassembly

ABCD. This is actually an exclusive OR or XOR relationship, as only one of the operations can be performed, excluding the other ones.
Here we observe an essential difference with the state diagram, because one subgraph in the AND/OR graph might represent multiple trajectories in the state diagram, see the enhanced arrows in figure 24, which expresses the same process as is depicted in figure 23. This is due to the fact that the AND/OR graph doesn't discern between the ordering of operations that can be executed in parallel.

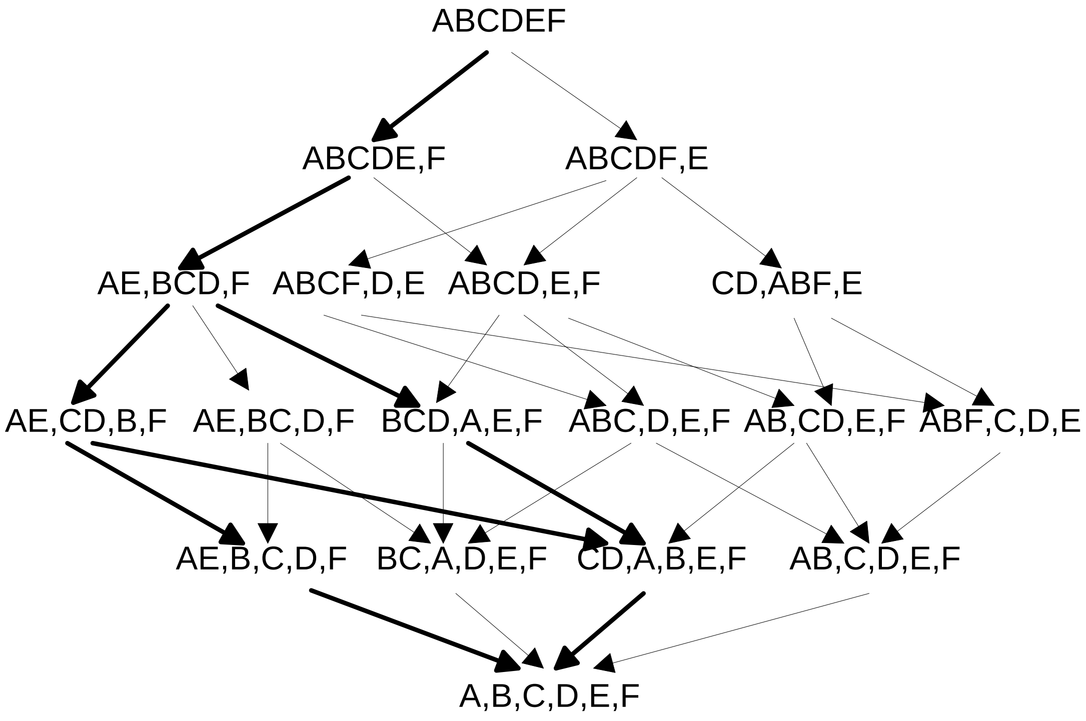

Figure 24. Sequences represented by the divergent subgraph of figure 23.

From this, it becomes clear that the state diagram depicts more sequences than the AND/OR graph, because it discerns between the ordering of parallel disassembly operations. In this particular case, three different sequences in the state diagram count for only one subgraph in the AND/OR graph.

4.4.3. Disassembly precedence graph

The disassembly precedence graph (DPG) is based on an approach that is different from both previously explained graphical representation methods, as it is derived from the task precedence graph, which is commonly used in industry, project planning, and related topics. The well known PERT (project evaluation and review technique) uses an application of mathematical programming on this kind of model. As a task precedence graph can be derived for virtually every kind of process, it can also be used for disassembly purposes. In the conventional notation, the nodes represent operations and the arcs represent *precedence relationships*. The latter are essential in disassembly as it occurs frequently that some component must be removed prior to disassembling some other component. If component A must be removed prior to the removal of component B, the precedence relationship is represented by A → B. In the DPG it is represented by an arc pointing from A to B.
Precedence relationships are more complex in practice. A selection rule, such as AF not B, also refers to a precedence relationship. It states that either A or F must be removed prior to the disassembly of B. This is obviously an OR relationship.

Examples of disassembly precedence graph, referring to Bourjault's ballpoint, are in figure 25. It can be observed that two *directions of preference* are present. Figure 25a depicts the DPG if motion in the +*x* direction (to the right) only is allowed, and figure 25b depicts the DPG if motion in the –*x* direction only is permitted. Notice that the three incoming arcs in B and E respectively, refer to AND relationships. The notation is such that the operations are indicated by the component that is removed by the operation. The start is at the *'initial state'*, which is represented by a virtual operation. The diagram is organized such that in the first column is the initial state, in the second column those components that can be removed without previous physical operation, in the second column those that can be removed with only one previous operation, etc. A combined DPG is presented in figure 25c.

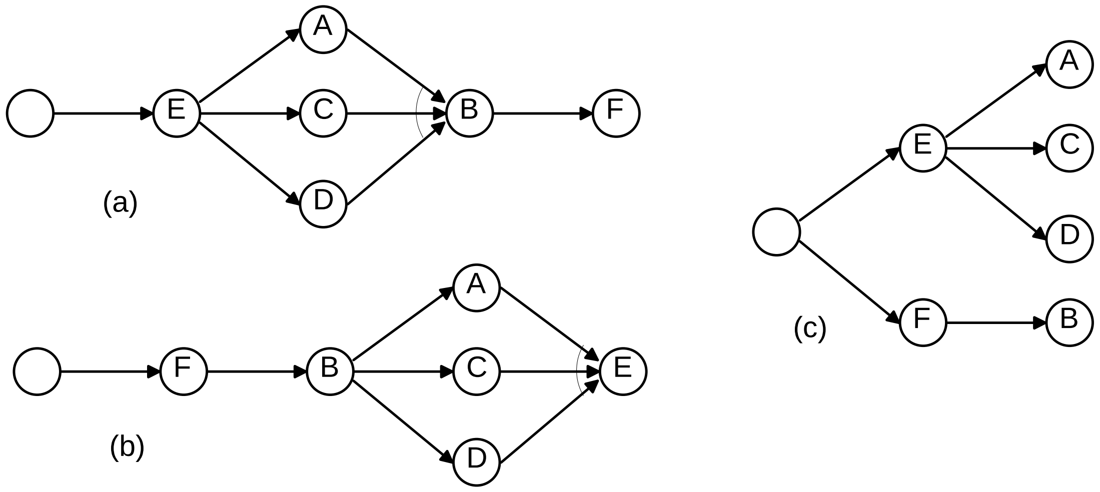

Figure 25. DPGs for Bourjault's ballpoint. (a) +x direction only; (b) –x direction only; (c) combined graph.

Parallel disassembly operations are not included in these graphs. There have been some attempts in the literature to include such operations, but these methods were rather artificial and appropriate only for quite simple products.
But we also see that none of the graphs depicts all the possible sequential sequences, e.g., the sequence EACDBF is not included in figure 25c. The sequence EACDFB is not included in either graph 25a or in graph 25b. Notice that the diagrams in figure 25 account for geometrical constraints only!
In advance, we mention that, in case of Bourjault's ballpoint, the state diagram (figure 22) represents 24 complete sequences, the AND/OR graph (figure 23) represents 14 complete sequences, and the disassembly precedence diagram figure 25c represents 90 sequences, although a lot of these are infeasible because of topological constraints. If topological constraints were not considered in the state diagram and the AND/OR graph, these would become very complex, already in case of this simple product. The DPG thus turns out to be an extremely compact notation. Unfortunately, it typically does not represent the complete set of disassembly sequences. A discussion on determination of the size of the search space is in chapter 5.

4.5. Technical constraints

It is evident that many of the disassembly processes, such as represented by figure 23, are not feasible in practical cases. For instance, the ink, i.e. component D, is not a rigid body at all. In case of assembly, e.g., the ink can only be applied if the head B is mounted to the cartridge C. In case of disassembly, however, the ink can only be removed when the head is removed from the cartridge. This kind of technical or soft constraints can be translated in addition to the already found selection rules. For assembly, we add CD not B, and for disassembly, we add BC not D. This implies that, in case of assembly, the subassembly CD is not longer feasible. In case of disassembly, the subassemblies BC, ABC, and ABCF are not longer feasible. The operations that act on these subassemblies are also removed from the diagram.

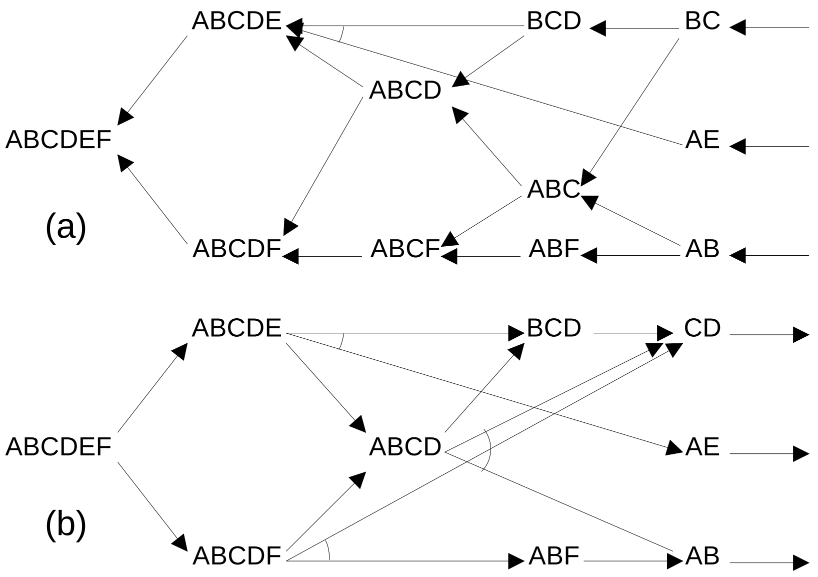

Figure 26. Assembly (a) and disassembly (b) AND/OR graph with technical constraints included.

Figure 26 demonstrates that in case technical constraints play a role, disassembly and assembly graphs are not longer each others reverse. An example of a sequential assembly sequence is, e.g.,

B→BC→ABC→ABCF→ABCDF→ABCDEF,

see figure 26a. An example of a sequential disassembly sequence is:

ABCDEF→ABCDE→ABCD→BCD→CD→C,

see figure 26b.
Technical constraints are also called soft constraints because they are, in contrast to topological and geometric constraints, not completely predetermined, as they might depend on tools available, preferences of the management, production schedule, etc. and can often be modified if desired, without modifying the product.

4.6. Disassembly precedence matrices

4.6.1. General
Many products, although not designed for disassembly, are structured such that they have only a restricted set of disassembly directions. In many products, these directions are a set of orthogonal coordinates, which results in 6 translational directions: $\pm x$, $\pm y$, $\pm z$. The examples in figure 19 and 21 even have only 2 translational directions of disassembly, as these products are axially symmetric. The modest number of preferential disassembly directions arises from design for assembly, which is usually practiced from a costing point of view. In this case, we can characterize the geometrical properties of a product by a single matrix, or in case of a complete Cartesian system, by *three* matrices. Once this matrix known, the disassembly graph can be determined

completely automatically, and queries are no longer required! We will demonstrate this at hand of Bourjault's ballpoint.
Let us consider a disassembly precedence matrix (DPM) for the +*x* direction. The rows refer to the components, and the components refer to those components that obstruct disassembly of the 'row'-component via the +*x* direction. Disassembly means here: motion from the product to 'infinity', i.e. to a place away from the product. It can be observed, e.g. that component A is obstructed by E only in the *x*-direction. The +*x* matrix is in figure 27a.

$$
\begin{array}{c|cccccc}
 & A & B & C & D & E & F \\
\hline
A & 0 & 0 & 0 & 0 & 1 & 0 \\
B & 1 & 0 & 1 & 1 & 1 & 0 \\
C & 0 & 0 & 0 & 0 & 1 & 0 \\
D & 0 & 0 & 0 & 0 & 1 & 0 \\
E & 0 & 0 & 0 & 0 & 0 & 0 \\
F & 1 & 1 & 1 & 1 & 1 & 0
\end{array}
\qquad
\begin{array}{c|cccccc}
 & A & B & C & D & E & F \\
\hline
A & 0 & 1 & 0 & 0 & 0 & 1 \\
B & 0 & 0 & 0 & 0 & 0 & 1 \\
C & 0 & 1 & 0 & 0 & 0 & 1 \\
D & 0 & 1 & 0 & 0 & 0 & 1 \\
E & 1 & 1 & 1 & 1 & 0 & 1 \\
F & 0 & 0 & 0 & 0 & 0 & 0
\end{array}
$$

(a) (b)

Figure 27. Disassembly precedence matrices for Bourjault's ballpoint. (a) +x direction; (b) –x direction.

It can be seen, e.g., that disassembly of component F is obstructed in the +*x* direction by all the other components, and that disassembly of component C is obstructed by component E only. If a component is obstructed by another component in that particular direction, the value 1 is assigned to the corresponding matrix element. All the other elements are put zero.
We can also establish the matrix for the –*x* direction, see figure 27a.

<u>Exercise</u>: Check this with the help of figure 19.

It can be noticed that matrix 27b follows from matrix 27a via exchanging rows and columns, or:

$$D_{-x} = D_x^T$$

In other words, it can be stated that the *transpose* of the disassembly precedence matrix for a particular direction equals the disassembly precedence matrix that corresponds to the opposite direction. This implies that only a single matrix for every relevant Cartesian axis must be fed to the computer prior to fully automatic calculation.

4.6.2. Sequential disassembly
It is interesting to know that the connectivity matrix and the relevant DPMs contain all the topological and geometric information that is required for determining the complete set of disassembly sequences. This means that the assembly drawing is not longer needed for the calculation, as it contains redundant information for this phase

of determining the optimum sequence. We can restrict ourselves to matrix 27a in case of Bourjault's ballpoint. If we start with disassembling, we search for those components with all their row elements equal to zero (or, equivalently, with all their column elements equal to zero, if the opposite direction is considered).
We see that component E is the component that is searched for. Alternatively, F can be disassembled in the opposite direction, for its column elements are zero. We proceed with the 5×5 matrix that corresponds to either the subassembly ABCDF or ABCDE. In case E is disassembled, we remove both the row and the column that correspond to component E. Then there appear three components, A, C, and D that have all their row elements zero. There are no components other than F with all column elements equaling zero. With regard for the topological constraints, it is component D that has to be removed subsequently. We are left with C and A that have all the row elements equal zero. Obviously, C must be disassembled if the $+x$ direction is selected.
In figure 28 is depicted how the matrices are modified when first E is removed in the $+x$ direction, and next F in the $-x$ direction.

$$\begin{array}{c|ccccc}
 & A & B & C & D & F \\
\hline
A & 0 & 0 & 0 & 0 & 0 \\
B & 1 & 0 & 1 & 1 & 0 \\
C & 0 & 0 & 0 & 0 & 0 \\
D & 0 & 0 & 0 & 0 & 0 \\
F & 1 & 1 & 1 & 1 & 0
\end{array}
\qquad
\begin{array}{c|cccc}
 & A & B & C & D \\
\hline
A & 0 & 0 & 0 & 0 \\
B & 1 & 0 & 1 & 1 \\
C & 0 & 0 & 0 & 0 \\
D & 0 & 0 & 0 & 0
\end{array}$$

(a) (b)

Figure 28. Modified DPM when (a) component E is removed in the +x direction and (b) next component F is removed in the $-x$ direction.

4.6.3. Parallel disassembly
Possible parallel disassembly operations can also be detected with the DPM. One simply has to combine the columns of both the rows and the columns that refer to the components of some child subassembly, or supercomponent, which one considers a candidate for disassembly. In figure 29, the subassembly CD is considered.

$$\begin{array}{c|ccccc}
 & A & B & \mathbf{C} & \mathbf{D} & F \\
\hline
A & 0 & 0 & 0 & 0 & 0 \\
B & 1 & 0 & 1 & 1 & 0 \\
\mathbf{C} & 0 & 0 & 0 & 0 & 0 \\
\mathbf{D} & 0 & 0 & 0 & 0 & 0 \\
F & 1 & 1 & 1 & 1 & 0
\end{array}
\qquad
\begin{array}{c|cccc}
 & A & B & \mathbf{C'} & F \\
\hline
A & 0 & 0 & 0 & 0 \\
B & 1 & 0 & 1 & 1 \\
\mathbf{C'} & 0 & 0 & 0 & 0 \\
F & 1 & 1 & 1 & 0
\end{array}$$

(a) (b)

Figure 29. Contraction of a DPM. Components C and D are merged here to C'.

In case the components of the subassembly are no neighbors in the matrix, rows and columns can be interchanged appropriately. The elements of the sub matrix that refers to the components selected subassembly are replaced by zeroes. The sub matrix is contracted to a single element, by redefining a combined column and a combined row. The relevant rows and columns are combined such that if at least one 1 appears in a column or row that refers to a single component, the combined element becomes 1. The zero value is assigned to all other components. This procedure, which is called *contraction*, seems rather artificial, but it effectively removes the obstruction that components of a subassembly can experience from each other, and it combines the obstruction of the components not belonging to such a subassembly.

<u>Exercise</u>: Investigate with the DPM all the operations in figure 23.

4.7. Summary

This chapter discussed, at the hand of a simple example, the basic features of constructing a graphical representation of disassembly processes. The DPG is the oldest method, initially applied to assembly processes and considering such processes as any other process. Bourjault realized that assembly processes can be approached considering it as reverted disassembly, thus developing disassembly theory. Based on this theory, Homem de Mello and Sanderson (1990, 1991) developed the disassembly AND/OR graph that, in contrast with DPGs, represented all the possible disassembly operations. These authors were interested in robotized construction and repair activities in harsh environments, such as space stations and nuclear power plants. De Fazio and Whitney (1987), and Baldwin *et al.* (1991) focused on assembly processes such as in the automotive industry. They developed the cut-set method, including the subset and superset rule. They used the state diagram, which automatically results from Bourjault's theory. As these authors made use of the 'connection oriented approach', which focuses on disestablishing connections rather than the 'component oriented approach', which is advocated here, their method appeared to be rather complicated. These authors used the automatic transmission as a case. Investigation of this rather simple product was already at the limit of their possibilities.
The first application of DPMs is less unequivocal. Multiple authors apply various matrix notation in a different way. Among the first authors that applied this tool were Laperriere and ElMaraghy, 1992.
We have reduced the problem of detecting all the feasible subassemblies and operations (Lambert, 2005) to semi-automatically deriving a set of feasible subassemblies and operations, accounting for both topological and geometric constraints. Once such a set exists, appropriate technical constraints can be added, thus reducing the set of subassemblies and operations.
With the DPM method, all the subassemblies and operations can be determined automatically, once the connectivity matrix and the DPMs are established. This method is restricted to products with a restricted set of disassembly directions.
The resulting set of subassemblies and operations includes the complete set of disassembly sequences, and provides us with the framework of the mathematical model that is used for selecting the optimum disassembly sequence.
Before doing this, we will discuss some methods for investigating the size of the search space we are confronted with.

5. Size of search space

5.1 Introduction

The size of the search space of a graph can be detected via *combinatorial analysis* or via graphical methods. Combinatorial analysis is of interest for the general case, in which neither topological nor geometric constraints are included. The figures that are encountered by this, mathematical, methods give the upper boundary for the search space of a product with *N* components. One might calculate the maximum number of subassemblies, operations, cut-sets, complete and incomplete sequences, etc., in state diagrams, AND/OR graphs, and disassembly precedence graphs. The first systematic study on this topic is due to Wolter (1992). A review of such calculations has been presented by Lambert and Gupta (2005), see chapter 4. A set of tables is included in this work. We will not go deep into this kind of mathematics, although it is not extremely difficult. For a general product with neither topological nor geometric constraints, the state diagram would include 115,975 nodes and 1,146,931 operations, resulting in 2,571,912,000 complete disassembly sequences. If the same product is represented by an AND/OR graph, there still would be 1,023 nodes, 28,501 operations, and 34,459,425 complete disassembly sequences. These numbers are huge indeed, and steeply increasing with the number of components. Although, in practice, these numbers are considerably reduced by both topological and geometric constraints, they still tend to exponentially increase with the product's complexity. The automatic transmission in figure 21 has 11 components. The corresponding AND/OR graph counts 68 subassemblies, and 216 operations. The number of sequences might still be about one hundred thousand. An enumerative search for the optimum sequence of still more complex products thus will be unattainable, even for the fastest computers.

5.2. State diagram

Fortunately, the size of the search space can be easily determined graphically if either the state diagram or the AND/OR graph is known. This starts with the individual components and works up upstream till the complete product node is attained.
The procedure for the state diagram is depicted in figure 30.

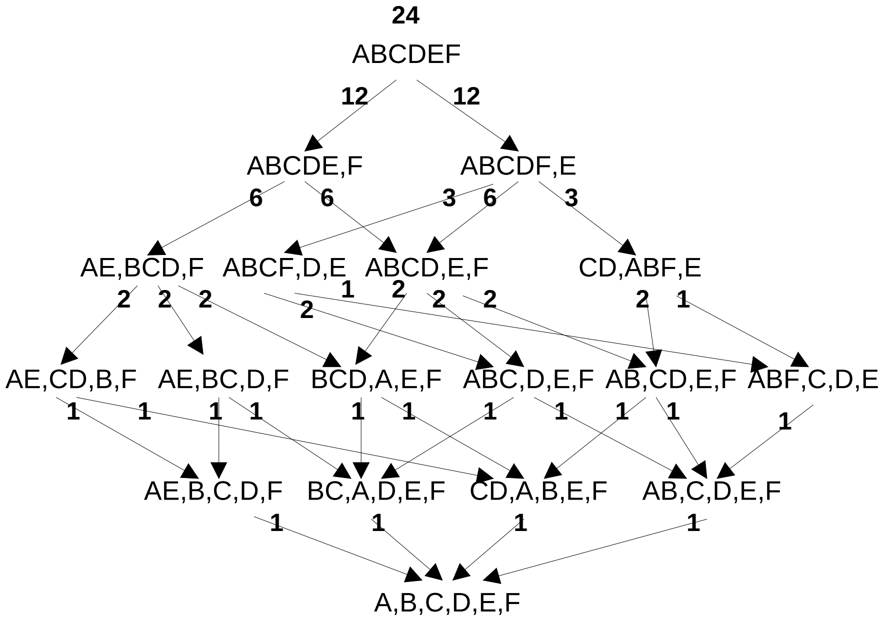

Figure 30. Graphical method for determining the number of disassembly sequences in the state diagram.

Starting with the disassembled state, we observe that there is only one way to transform the state AE,B,C,D,F into the disassembled state. The number '1' is assigned to the corresponding operation. The number that is assigned to each of the incoming arcs of any state equals the sum of the numbers that are assigned to the outgoing arcs. By consequently applying this rule, we end up with 24 complete disassembly sequences.

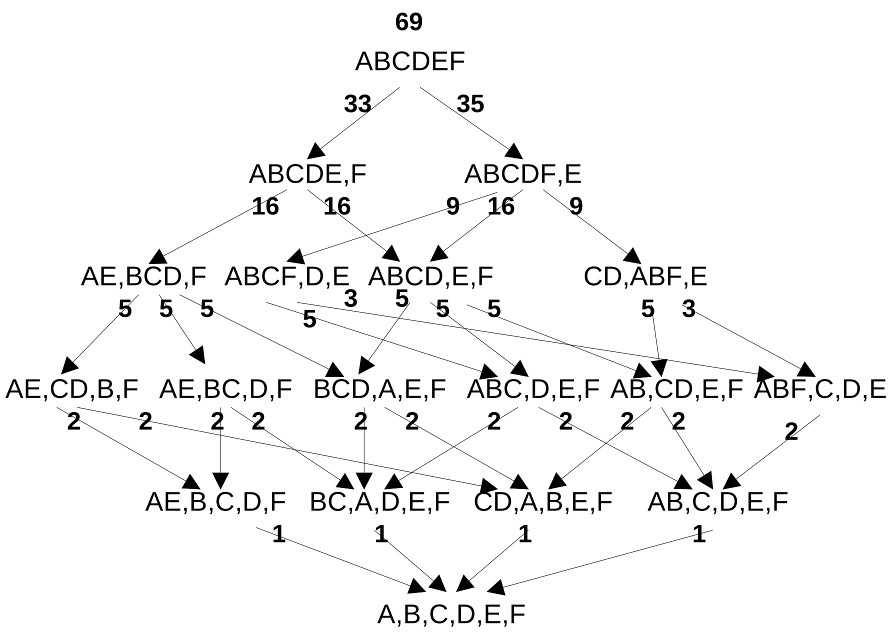

Figure 31. Number of disassembly sequences, when incomplete disassembly is permitted.

As we have discussed already, *incomplete disassembly* is often also permitted in end-of life disassembly. The number of possible disassembly sequences then increases further because an additional degree of freedom is introduced. The calculation proceeds as the previous one,. except that the number of possible sequences increases additionally with 1 when a node is encountered, accounting for the possibility of finishing at this state. The number of possible disassembly sequences thus equals 69 in this case, which is a considerable increase indeed.

5.3 AND/OR graph

We now discuss the determination of the number of complete disassembly sequences for the AND/OR graph. This proceeds similar to the state diagram, provided that the numbers that are assigned to both branches of the hyper arcs are *multiplied* in stead of added. In case of a sequential operation, this is done implicitly, as the number 1 is assigned to the branch that points to the single component.
The calculation of the number of complete sequences is demonstrated in figure 32 and the calculation of the number of both complete and incomplete sequences is in figure 33. These numbers are 16 and 58, respectively. That these are less than in case of the state diagram has already been discussed previously.

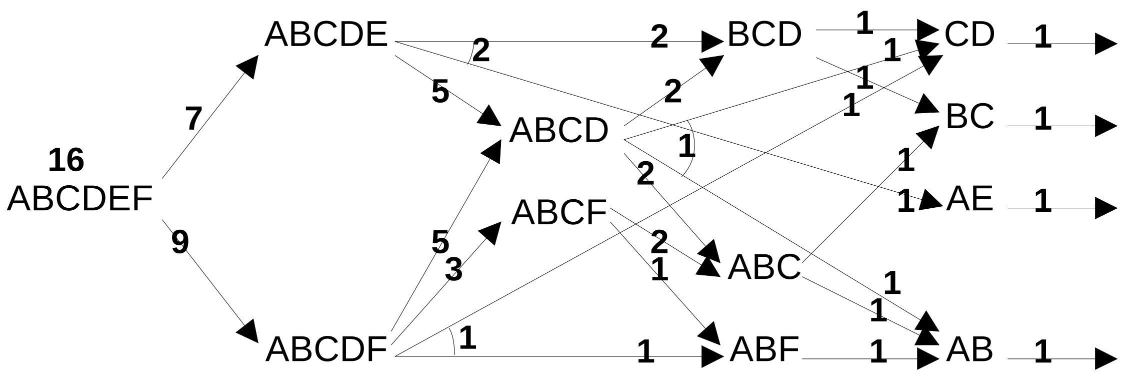

Figure 32. Number of complete disassembly sequences in an AND/OR graph.

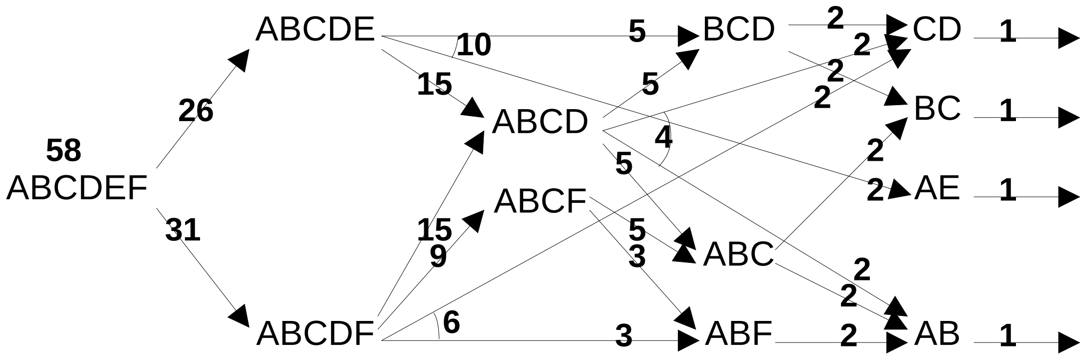

Figure 33. Number of complete and incomplete disassembly sequences in an AND/OR graph.

For the sake of simplicity in notation, numbers are assigned to both the operations and the subassemblies, see figure 34.

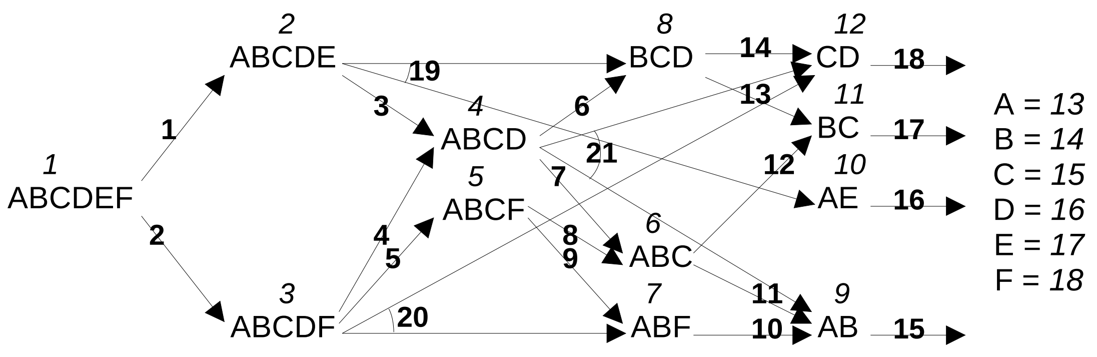

Figure 34. Abbreviated notation for subassemblies and operations.

With this, the set of complete sequences reads:

```
1-3-6-13-17    1-3-6-14-18
1-3-7-11-15    1-3-7-12-17
1-3-21-15-18                    (1-3-21-18-15)
1-19-13-16-17                   (1-19-13-17-16       1-19-16-13-17)
1-19-14-16-18                   (1-19-14-18-16       1-19-16-14-18)
2-4-6-13-17    2-4-6-14-18
2-4-7-11-15    2-4-7-12-17
2-4-21-15-18                    (2-4-21-18-15)
2-5-8-11-15    2-5-8-12-17
```

2-5-9-10-15
2-20-10-15-18 (2-20-10-18-15 2-20-18-10-15)
The sequences between brackets refer to the different order when parallel disassembly is applied. If these sequences are added, we arrive at the number of sequences that has been encountered in figure 30, where the state diagram is considered.

5.4 Disassembly precedence graph

For a DPG with no constraints, i.e. no precedence relations, the number of sequences is $N!$, with N the number of non-virtual nodes. This is the case only if all the components are accessible and if their removal does not result in disintegration of the remaining subassembly. This means that not any precedence relation is accounted for.
In the general case, with precedence relations present, a rigorous method for determining the number of possible sequences that is represented by is not available. This number surely can be determined by some algorithm, but the required processor time (CPU time) increases exponentially with the problem's complexity.
A rule for determining the number of sequences in a DPG is present indeed, but it is valid only for a *completely divergent DPG*, i.e. a DPG that does not include AND relationships. This means that any non-virtual node has only one incoming arc. The procedure is due to Uchiyama *et al.* (1994). It proceeds as follows, see figure 35.

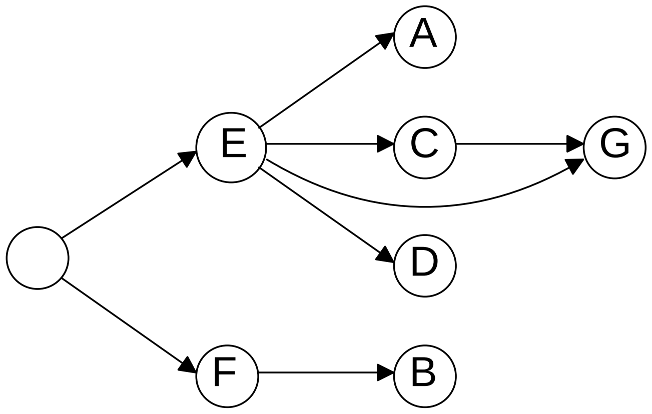

Figure 35. Determination of the number of sequences in a divergent DPG.

First, the graph is made *transient*, which means that every node is connected to all the nodes upstream of the node, i.e. those nodes that can be visited only when the node under consideration has been visited formerly. For node E, this refers to the node A, C, D, and G. As there are N nonvirtual nodes, the number of possible complete sequences equals:

$$\frac{N!}{\prod_{i=1}^{N}(r_i+1)} \qquad (5.1)$$

In this expression, r_i refers to the number of outgoing arcs in each node. In figure 35, we have: $r_E = 4$; $r_F = r_C = 1$; $r_A = r_G = r_B = 0$.
Consequently, the number of possible sequences equals 252.
The DPG in figure 25c has 90 possible sequences. Many of these sequences are, however, infeasible because of topological reasons.
From expression (5.1) it follows that the DPG is a compact representation, as a simple graph might result in a large number of sequences indeed.

6. Disassembly optimization with sequence independent costs

6.1 Introduction

As a product can be disassembled in many ways, we cannot enumeratively investigate all the possible solutions and select the best one according to some criterion. Usually, we have to translate the graph in a *mathematical model*, and solving this according to some set of rules. Actually, we will define an *objective function* or *profit function* that represents the criterion that has to be maximized. This criterion might be the financial profit indeed, which is the subtraction of yields, e.g. due to the value of the components, and the costs, e.g. caused by carrying out the disassembly operations. In stead of financial quantities, one can also include environmental benefits etc. The costs and yields are than considered generalized costs and yields. Notice that the quantity that has to optimized is a subjective one, which depends on the needs of the investigator. Costs and yields are fed to the model as a set of parameters. The values of these parameters can be determined empirically, but are typically guessed by the user. The mathematical model is used to find the optimum solution, which has the maximum value of the profit function. In practice, however, the solution must often meet multiple criteria. The optimum solution with respect to a single criterion can than be very unfavorable with respect to a second criterion, which might be independent of the criterion under consideration. In this case, the generation of a set of near optimum solutions might be of interest, as the solutions in the set can than enumeratively be checked with respect to the other criteria.
There are three types of methods for solving the optimization problem:

1. Exact methods.
These are based on mathematical programming, particularly Linear programming (LP), Binary integer linear programming (BILP), Mixed integer linear programming (MIP), Non-linear programming (NLP). Exact methods return the exact optimum solution, apart from NLP, which might return a local optimum. This is exact in the mathematical sense only, of course, as it is assumed that the parameters represent exact values. We noticed already that these are estimates. Apart from linear programming, which is based on an efficient algorithm, binary and full integer programming problems are suffering from other restrictions, i.e., the required CPU time strongly increases with the complexity of the problem, as it is related to the number of integer variables that is required.

2. Heuristic methods.
These are based on a set of rules-of thumb that are specific to the problem considered. The required CPU time is short, even for complex problems. It is not guaranteed that the exact solution is obtained. In this framework, one speaks about 'good enough' solutions. How good these solutions are, i.e. how far these are removed from the optimum, is unclear. Some heuristic methods return a set of suboptimal solutions.

3. Metaheuristic methods.
These are based on a set of rules that are generally applicable to a wide variety of problems. The required CPU time is reasonable and, although there is no guarantee of returning an optimum solution, efficient solutions are often obtained. If the complexity of a problem increases, the needed CPU time might become restrictive. The set of rules is often based on biological or physical processes, such as genetic algorithms,

ant colonies, simulated annealing. Actually, the metaheuristic offers a framework from which problem-specific heuristics can be derived without major modification.

In this course we will deal with focus on exact methods, but we will also deal with some heuristics, in which the exact methods are used for checking the validity of the heuristics at low and moderate complexity.

6.2. Mathematical model

Once the AND/OR graph is established, we proceed with the construction of the mathematical model. It has been supposed for long that the problem of finding the optimum sequence is NP-complete, because of the search space that exponentially increases with *N*. This appears not true, provided sequence independency of the costs of a disassembly operation is assumed.
Mathematical modeling aimed at disassembly sequencing starts with establishing the AND/OR graph. As an example, we refer to figure 36, with both the subassemblies and the operations enumerated.

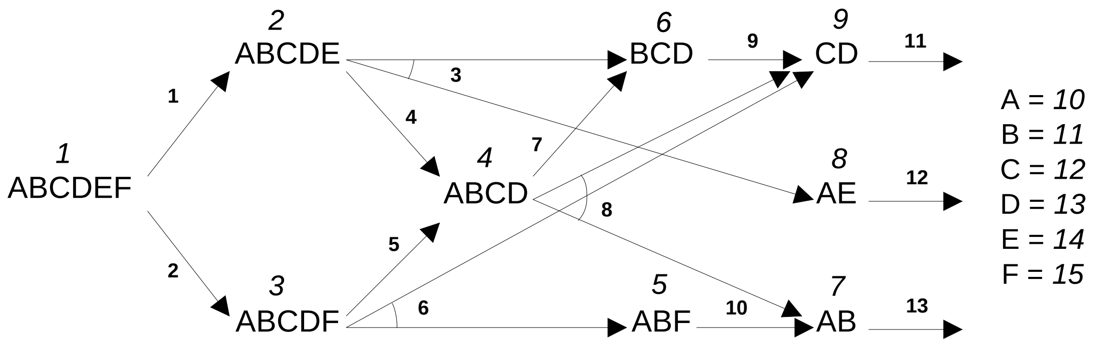

Figure 36. AND/OR graph of Bourjault's ballpoint. Subassemblies and operations are enumerated.

The subassemblies and the operations are labeled by the indices *i* and *j*, respectively. Let there be *I* subassemblies and *J* operations. The virtual initial operation, $j = 0$, is added for convenience. Labeling proceeds usually in descending order of components, next in alphabetical order. The structure of the AND/OR graph is condensed in the *transition matrix*, which is an $I \times J$ matrix. The element T_{ij} reflects whether or not the subassembly *i* is destroyed, or created, by the operation *j*. It equals +1 when subassembly *i* is created by operation *j*, it equals –1 when subassembly *i* is destroyed by operation *j*. In other cases, the element equals 0. The product $i = 1$ is created by the virtual operation $j = 0$.
Apart from the structure of the problem, the parameter values have to assigned to the problem. These are condensed in two vectors. The *cost vector C* with components C_j is defined, corresponding with the cost of operation *j*. The *yield vector Y* with components Y_i is defined, corresponding with the revenues of subassembly *i*.
The variables are the so called *flow variables* x_j. Variable x_j returns the value 1 if operation *j* is actually performed. In other cases, the value 0 is returned. Although the variables appear to be binary integers, they can actually be defined as real variables, as they are confined automatically to the values zero or 1, when the problem is adequately formulated.
An example of a transition matrix, based upon figure 36, is presented below:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
ABCDEF	1	-1	-1	0	0	0	0	0	0	0	0	0	0	0
ABCDE	0	1	0	-1	-1	0	0	0	0	0	0	0	0	0
ABCDF	0	0	1	0	0	-1	-1	0	0	0	0	0	0	0
ABCD	0	0	0	0	1	1	0	-1	-1	0	0	0	0	0
ABF	0	0	0	0	0	0	1	0	0	0	-1	0	0	0
BCD	0	0	0	1	0	0	0	1	0	-1	0	0	0	0
AB	0	0	0	0	0	0	0	0	1	0	1	0	0	-1
AE	0	0	0	1	0	0	0	0	0	0	0	0	-1	0
CD	0	0	0	0	0	0	1	0	1	1	0	-1	0	0
A	0	0	0	0	0	0	0	1	0	0	0	0	1	1
B	0	0	0	0	0	0	0	0	0	1	0	0	0	1
C	0	0	0	0	0	0	0	0	0	0	0	1	0	0
D	0	0	0	0	0	0	0	0	0	0	0	1	0	0
E	0	0	1	0	1	0	0	0	0	0	0	0	1	0
F	0	1	0	0	0	1	0	0	0	0	1	0	0	0

Figure 37. Transition matrix of Bourjault's ballpoint.

For every node, a node equilibrium equation must be established. It has a ≤ sign if selective disassembly is permitted, and it has a = sign if disassembly should be carried out completely. For the disassembly case of Bourjault's ballpoint, we have:

Node 0: $x_0 = 1$ (initialization)
Node 1: $x_1 + x_2 \leq x_0$
Node 2: $x_3 + x_4 \leq x_1$
Node 3: $x_5 + x_6 \leq x_2$
Node 4: $x_7 + x_8 \leq x_4 + x_5$
Node 5: $x_{10} \leq x_6$
Node 6: $x_9 \leq x_3 + x_7$
Node 7: $x_{13} \leq x_8 + x_{10}$
Node 8: $x_{12} \leq x_3$
Node 9: $x_{11} \leq x_6 + x_8 + x_9$

The node equilibrium equations for non-virtual nodes can be condensed via the notation:

$$\sum_j T_{i,j} \cdot x_j \geq 0 \qquad (6.1)$$

The second row of the example, $i = 2$, reveals: $x_1 - x_3 - x_4 \leq 0$, which coincides with the node equilibrium equation for the second node.

The objective function or profit function is the difference between yields and costs. In case no operation is performed, the process stops after the virtual operation x_0, which makes the product available. The yield is than the yield of the product, when sold. This yield can be, and will often be, negative. The aim of the disassembly operation is increasing the profit. One also has to comply with regulation, such as isolation of hazardous substances and/or components. This is somewhere embedded in the values of the subassemblies. The product, e.g., must be disposed of somehow, which might be more expensive when hazardous components are still present in it.

The objective function thus must be equal the sum of the yields of the subassemblies available when disassembly stops, with the sum of the operation costs subtracted from it. The sum of the operation costs simply equals the sum of the products of the flow variables and the costs that are assigned to it via the cost vector. When the flow vector is zero, these costs are not added.
In the general case, we thus reveal:

$$\Pi = Y - C = \sum_j \sum_i T_{i,j} \cdot x_j \cdot Y_i - \sum_j x_j \cdot C_j \qquad (6.2)$$

Obviously, Π must be maximized.

Example: Let the sequence b: 0-1-3, which means: $x_0 = x_1 = x_3 = 1$, and the other flow variables equal zero. The yield reads:

$$Y = Y_1 \cdot x_0 + (-Y_1 + Y_2 + Y_{15}) \cdot x_1 + (-Y_2 + Y_6 + Y_8) \cdot x_3 = Y_6 + Y_8 + Y_{15}$$

This is the yield indeed of the subassemblies we are left with, viz.: BCD, AB, and F. The costs obviously equal: $C_0 + C_1 + C_3$.

Although this problem is a binary integer linear programming problem at the first sight, it relaxes to a linear programming problem, because the value of the flow variables, which can only be either 0 or 1, spontaneously will attain one of these values, as linear programming variables tend to arrive at an extreme value, when not constrained by additional constraints. We distinguish 'hard' and 'soft' constraints. The soft ones are accounted for by, e.g., extremely high costs and/or extremely low yields. The hard constraints refer to, e.g., limited capacity to perform some specified operation, obligatory release of some component, a limited demand, etc.

7. Extended problems

7.1. Weakly and strongly connected products

We have presented the theory on end-of life disassembly via referring to a quite simple problem, viz., Bourjault's ballpoint. In this section, we will briefly discuss some features that are related to some more extended problems. We already introduced the automatic transmission as an example of a problem with enhanced complexity. Although still axially symmetric, with directions of disassembly in the $\pm x$ direction only, it has 11 components, A thru L, apart from fasteners M, N, O, P, Q. These are, e.g., bolts and are not considered distinct components. Obviously, there are more than one bolts of each type, but we assume that these will be simultaneously released via a single operation.

Both the assembly drawing and the related connection diagram are in figure 38ab.

In contrast with the connection diagram of Bourjault's ballpoint, we now discern loops in this diagram, which means that the ratio of the number of connections and of components is increased. In a weakly connected product, the connection diagram counts $N-1$ connections, in a strongly connected product, it counts $\frac{1}{2}N(N-1)$ connections, which is the maximum number of connections possible. In case of L loops, there are $N+L-1$ connections. The connection diagram in figure 38b has 8 loops, and there are 18 connections indeed. Notice that the loops that are related to fasteners have not been included here.

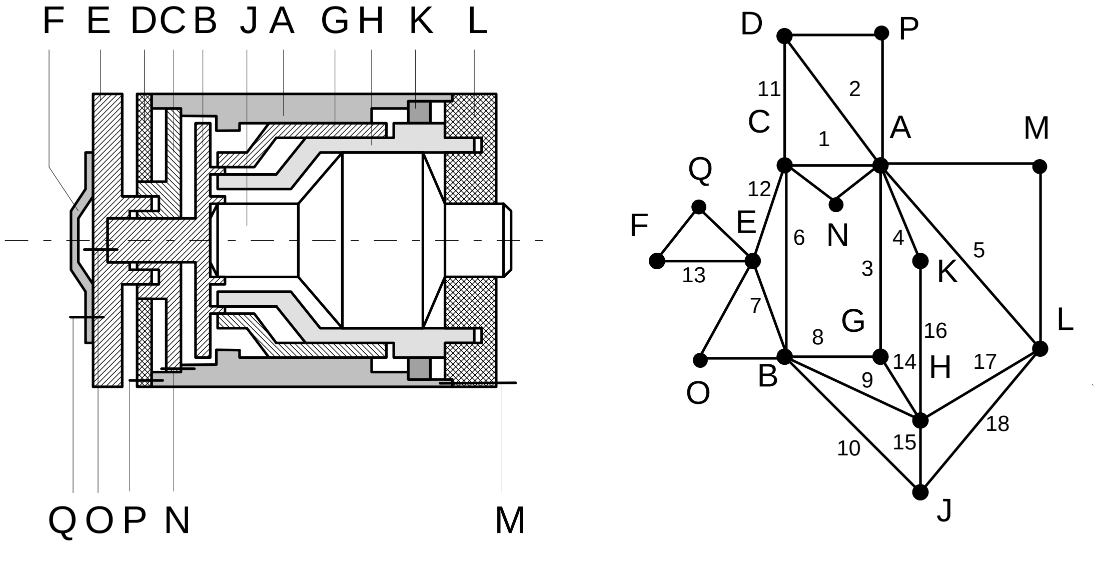

Figure 38. Automatic transmission line with fasteners. (a) Assembly drawing; (b) Connection diagram.

Within this framework, the definition of then *index of complexity*, α, should be mentioned. It is given by the expression:

$$\alpha = \frac{2K}{N} \tag{7.1}$$

The symbol K refers to the number of connections. It equals $2-\frac{1}{N}$ for weakly connected products, and $N-1$ for strongly connected products. For the automatic transmission, it equals 3.27 when fasteners are not included, and 3.07 when fasteners are included. Experience from practice reveals that, for a typical product, $2 \leq \alpha \leq 4$.

7.2. AND/OR graph construction by sequential disassembly

Let us start with not including the fasteners. We first want to systematically derive a list of feasible subassemblies, accounting for both topological and geometric constraints. Assuming the rigid body approach and starting with sequential disassembly, such as in table 4.2, we have, see table 7.1, which present part of the procedure. When completed, we end up with 13 selection rules and 15 feasible cut-sets.
The additional selection rules read:

BK not A (10)
AH not G (11)
BD not C (12)
DF not E (13)

The set of sequential operations is depicted by the AND/OR graph, see figure 39. This calculation can thus be done straightforwardly, and the number of queries is restricted to a modest amount indeed. Because there is a restricted number of preferential disassembly directions, the procedure even can proceed fully automatically.

7.3. AND/OR graph construction with parallel disassembly

A problem now remains how to deal with parallel disassembly operations. The reason is, that the set of subassemblies, such as depicted in figure 39, might not be complete. It should be noticed that figure 39 refers to the situation with fasteners present, which is restrictive to some extent. This results in a reduced number of subassemblies, which will be explained in the subsection to follow. This might indeed be possible. A closer look to figure 38a reveals that subassembly EF, e.g., can be obtained by disassembly operations only, so it is feasible, but it can not be obtained by sequential disassembly alone, even when no fasteners are present. The reason is that we have to disassemble in the +x direction, and are finally left with CDEF. Component D cannot be disassembled, because of a geometric constraint, and C cannot be disassembled because of a topological constraint. Consequently, CD must be disassembled as a whole, leaving us with EF, which is parallel disassembly.
Searching for parallel disassembly might be cumbersome indeed, as the number of combinations that must be investigated will be large, although not many additional subassemblies might be detected. Fortunately, we can reduce the number of combinations to be investigated by subjecting the possible subassemblies to the known selection rules and the topological constraints.
In case of the automatic transmission, the following 2-subassemblies are possible with regard to the topological constraints, see figure 38b:
AC, AD, AG, AK, AL, BC,BE, BG, BH, BJ, CD, CE, EF, GH, HJ, HK, HL, JL

Table 7.1 Selection procedure with topological and geometric constraints.

Query		Selection rule	Feasible cut-set
ABCDEFGHJKL → A + BCDEFGHJKL	N	BG not A (1)	
ABCDEFGHJKL → B + ACDEFGHJKL	N	AC not B (2)	
ABCDEFGHJKL → C + ABDEFGHJKL	N	AD not C (3)	
ABCDEFGHJKL → D + ABCEFGHJKL	N	CE not D (4)	
ABCDEFGHJKL → E + ABCDFGHJKL	N	BF not E (5)	
ABCDEFGHJKL → F + ABCDEGHJKL	Y		ABCDEFGHJKL → F + ABCDEGHJKL (1)
ABCDEFGHJKL → G + ABCDEFHJKL	N	BH not G (6)	
ABCDEFGHJKL → H + ABCDEFGJKL	N	GJ not H (7)	
ABCDEFGHJKL → J + ABCDEFGHKL	N	HL not J (8)	
ABCDEFGHJKL → K + ABCDEFGHJL	N	AL not K (9)	
ABCDEFGHJKL → L + ABCDEFGHJK	Y		ABCDEFGHJKL → L + ABCDEFGHJK (2)
ABCDEGHJKL → A + BCDEGHJKL	N	(1)	
ABCDEGHJKL → B + ACDEGHJKL	N	(2)	
ABCDEGHJKL → C + ABDEGHJKL	N	(3)	
ABCDEGHJKL → D + ABCEGHJKL	N	(4)	
ABCDEGHJKL → E + ABCDGHJKL	Y		ABCDEGHJKL → E + ABCDGHJKL (3)
ABCDEGHJKL → G + ABCDEHJKL	N	(6)	
ABCDEGHJKL → H + ABCDEGJKL	N	(7)	
ABCDEGHJKL → J + ABCDEGHKL	N	(8)	
ABCDEGHJKL → K + ABCDEGHJL	N	(9)	
ABCDEGHJKL → L + ABCDEGHJK	Y		(2)
ABCDEFGHJK → A + BCDEFGHJK	N	(1)	
ABCDEFGHJK → B + ACDEFGHJK	N	(2)	
ABCDEFGHJK → C + ABDEFGHJK	N	(3)	
ABCDEFGHJK → D + ABCEFGHJK	N	(4)	
ABCDEFGHJK → E + ABCDFGHJK	N	(5)	
ABCDEFGHJK → F + ABCDEGHJK	Y		(1)
ABCDEFGHJK → G + ABCDEFHJK	N	(6)	
ABCDEFGHJK → H + ABCDEFGJK	N	(7)	
ABCDEFGHJK → J + ABCDEFGHK	Y		ABCDEFGHJK → J + ABCDEFGHK (4)
ABCDEFGHJK → K + ABCDEFGHJ	Y		ABCDEFGHJK → K + ABCDEFGHJ (5)

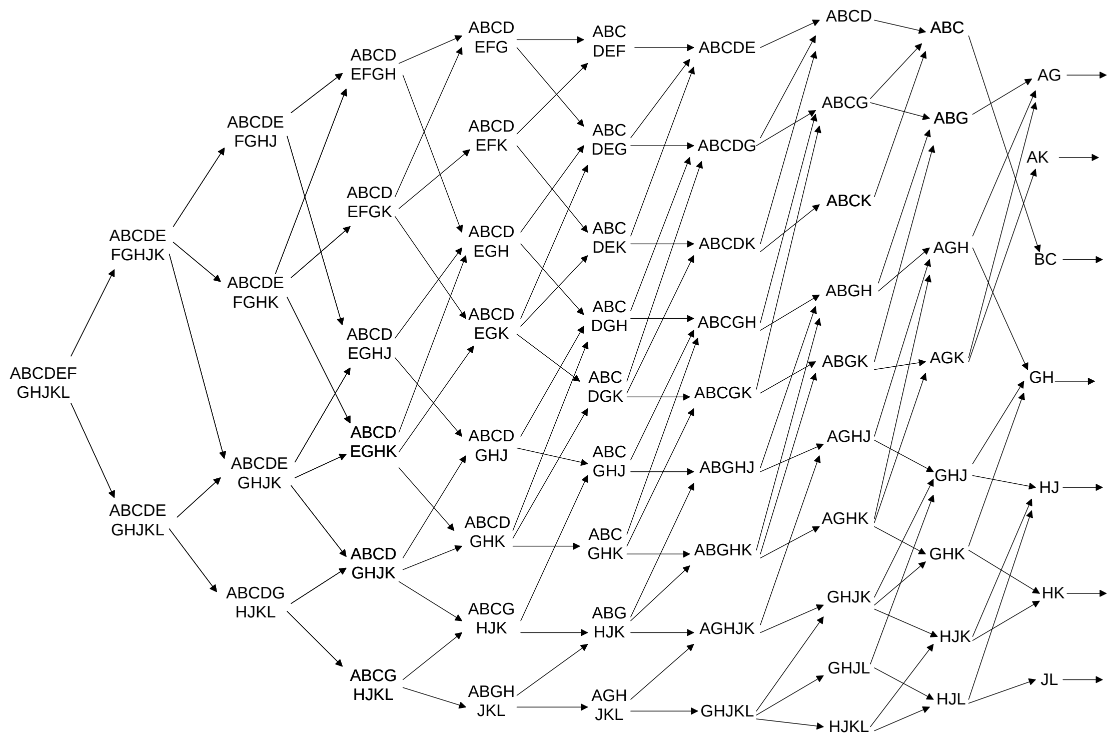

Figure 39. AND/OR graph for sequential disassembly of the automatic transmission.

Because of the selection rules, see table 7.1, many of these subassemblies can be rejected, and we are left with:

AG, AK, BC, BE, BJ, CD, EF, GH, HJ, HK, JL

Only four of them have not been detected yet:

BE, BJ, CD, EF

From these, it can be seen from the assembly drawing that both BE and BJ are not feasible because of geometrical constraints. Additional selection rules can be added, viz.: BE not C, and BJ not H.

Hence:

BE not C (14)

BJ not H (15)

We are thus left with:

AG, AK, BC, CD, EF, H, HJ, HK, JL

We then investigate 2-operations with the same procedure as in table 4.3:

Table 7.2 Investigating 2-disassembly operations

Query		**Selection rule**	**Feasible cut-set**
ABCDEFGHJKL → AG + BCDEFHJKL	N	(6)	
ABCDEFGHJKL → AK + BCDEFGHJL	N	(1)	
ABCDEFGHJKL → BC + ADEFGHJKL	N	(3)	
ABCDEFGHJKL → BE + ACDFGHJKL	N	(2)	
ABCDEFGHJKL → CD + ABEFGHJKL	N	(14)	
ABCDEFGHJKL → EF + ABCDGHJKL	Y		ABCDEFGHJKL → EF + ABCDGHJKL (16)
ABCDEFGHJKL → GH + ABCDEFJKL	N	(15)	
ABCDEFGHJKL → HJ + ABCDEFGKL	N	BL not J (16)	
ABCDEFGHJKL → HK + ABCDEFGJL	N	(7)	
ABCDEFGHJKL → JL + ABCDEFGHK	Y		ABCDEFGHJKL → JL + ABCDEFGHK (17)

By doing so, we have one additional selection rule, two additional feasible cut-sets, and additional 9-subassembly, as both ABCDEFGHK and ABCDGHJKL can be obtained by selective disassembly. EF is an additional 2-subassembly, where JF has also been obtained via sequential disassembly.

If an, even provisional, list of subassemblies is available, we can search for parallel disassembly operations via detection of *complementary triplet*s. The triplet:

ABCDEGHJKL, ABCDEGK, HJL

is an example of a complementary triplet. The search can be done automatically. The operation that separates the largest subset into the two smallest ones, is a candidate disassembly operation.

7.4. Fasteners

Let us investigate the role of the fasteners. As these are bolts or something like this, their yield either as a material or as a component is modest. Their environmental impact is also negligible, although bolts might contain, e.g., Cadmium. Clearly, there are many kinds of fasteners and these can even be an immaterial object, which simply has to be disestablished prior to disassembling some component. A component other than a typical fastener, which fulfills one or more specific functionalities within a product, can also have a fastening function, e.g., a spacer.

But let us assume that the bolts can be considered virtually immaterial, so these only enforce an additional constraint on the disassembly process. We thus do not include the fasteners M, N, O, P, or Q in the subassemblies, which considerably reduces the problem. Because M connects A and L, this fastener should be released before either A or L can be disassembled, although the bolt does not prevent the subassembly AL from being disassembled, which is prevented, however, by component K.

The precedence relations for the fasteners are as follows:

M → A or L

N → A or C

O → B or E

P → A or D
Q → E or F

Some of these bolts, such as M, can be released without prior detachment of any component. The bolts N, O, and P, however, need some previous removal of a component:

D → N
F → O
E → P

Combination of these two sets of precedence relationships yields:

D → A or C
F → B or E
E → A or D

This results in a set of six selection rules, viz.:

AD not C	CD not A
BF not E	EF not B
AE not D	DE not A

The consequence of the presence of the fasteners thus is that the subassemblies CD and EF are infeasible, i.e., these cannot be revealed from the product via disassembly alone.

7.5. Static and dynamic components

Till present, it has been assumed that a disassembly operation is symmetric, which means that there is no difference between both child disassemblies. In practice, however, the product is frequently immobilized by a fixture. This is often a body, housing, etc., for instance component A in both Bourjault's ballpoint and the automatic transmission. We call A the *static* component and the other components are called the *dynamic* ones. In this case, we must end up with AG or AK in figure 39. Considering the graph upstream and removing the operations and subassemblies that result in a 2-subassembly that does not include A, leaves us with the reduced graph of figure 40:

<u>Exercise</u>: Determine the number of sequences (complete only as well as the combined number of complete and incomplete sequences) in figure 40, using the graphical method. Also determine the number of complete sequences in figure 39.

Selection of the static component is according to two objective criteria: It should be reachable, which implies, it should be located at the external surface of the product and a fixture should be easily attached to it, without obstructing the disassembly process. Apart from this, it should possess as much as possible connections to other components. In the connection diagram by figure 38b, both the components A, B, and H have the maximum number of five connections. Both components B and H are completely internal. We are left with component A, which is indeed an appropriate candidate.

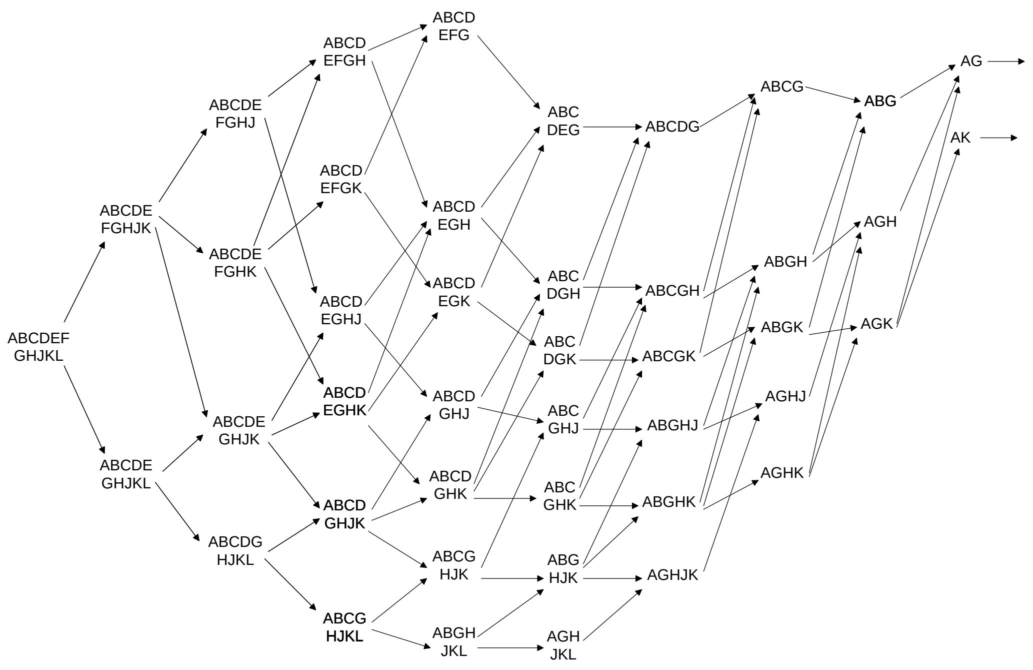

Figure 40. Sequential disassembly of the automatic transmission with A the static component.

7.6. Modules

Because, even confined to linear programming models, the number of both subassemblies and operations increases and thus the size of the model, there has been a challenge to automatically detect modules. Modules are then defined as subassemblies with a strong internal coherence and a weak external coherence.
In practice, e.g. in personal computers, selection of modules is often evident to some extent, as these are products as it is, and often easily separable from the rest of the product. Typical examples of this are DVD drives and plug-in boards.

Table 7.3. Selection of modules

Subassembly	Internal connections	External connections
A	0	5
B	0	5
C	0	4
D	0	2
E	0	3
F	0	1
G	0	3
H	0	5
J	0	3
K	0	2
L	0	3
AG	1	6
AK	1	5
BC	1	7
GH	1	6
HJ	1	6
HK	1	5
JL	1	4
ABC	2	10
ABG	2	9
AGH	2	9
AGK	2	6
GHJ	2	7
GHK	2	6
HJK	2	6
HJL	3	5
ABCD	4	8
ABCG	4	9
ABCK	3	10

ABGH	4	10
ABGK	3	9
AGHJ	3	10
AGHK	4	7
GHJK	3	7
GHJL	4	6
HJKL	4	5
ABCDE	6	7
ABCDG	6	7
ABCDK	5	8
ABCGH	5	12
ABCGK	5	9
ABGHJ	6	9
ABGHK	6	8
AGHJK	5	8
GHJKL	5	6
ABCDEF	7	6
ABCDEG	8	6
ABCDEK	7	7
ABCDGH	8	8
ABCDGK	7	7
ABCGHJ	8	9
ABCGHK	8	9
ABGHJK	8	7
AGHJKL	8	5

etc.

In mechanical devices, such as the automatic transmission of figure 38, selection of modules is less trivial. It is possible, with the assumption above, to select modules. We will make an attempt in the following table, selecting those subassemblies with *N* = 1, 2, etc., see table 7.3.
From this table, we observe that the component F is the best candidate for the 1-subassemblies. It has the minimum amount of external connections, for it is a *leave*. The 2-subassemblies always have 1 internal connection. JL is the best candidate here with the minimum amount of 4 external connections. If EF were feasible, it was still a better candidate with only 2 external connections.
Of the 3-subassemblies, HJL is the best candidate with both the maximum number of 3 internal connections, and the minimum number of 5 external connections.
Of the 4-subassemblies, HJKL is the best candidate, with 4 internal connections and only 5 external connections.
Of the 5-subassemblies, we detect ABCDE, ABCDG, and GHJKL as candidates. The first two have 6 internal connections and 7 external connections; the last one has 5 internal and 6 external connections.
Of the 6-subassemblies, we detect AGHJKL as a candidate module. Observation of the assembly drawing shows that it is a coherent subassembly indeed. Its complement, BCDEF, cannot be detached as a whole, because of the fasteners. If these were not present, with 5 internal and 5 external connections, it could be a candidate module as well.

8. Exact solutions for models subjected to sequence-dependent costs

8.1. Introduction

The exact method of search for the optimum disassembly sequence, which is based on linear programming methods and that was discussed in section 6, makes use of an amazingly simple model indeed. It suffers from one particular assumption, viz., that the costs of an operation are sequence independent. In practice, this is often not true, as the costs of a specific operation depend on the situation that has been left by the preceding operation. This is die to features such as *fixturing*, *product orientation*, and tool selection. In the automatic transmission, e.g., it might be useful to remove all the bolt at once, as this can proceed with the same tool. From a robot technology point of view, it is efficient to execute some operations that can be done in the same direction, so no product reorientation is required. As we have seen that the transmission has two directions of preference, it might be beneficial to carry out some operations in the $+x$ direction, prior to move the tool to the other side of the product aimed at operating in the $-x$ direction.
These technical considerations have resulted in attempts of solving the sequencing problem in case sequence dependent costs are present. In case exact methods were considered, one has to rely on integer programming methods. Unfortunately, CPU time tends to dramatically increase with product complexity, which is the reason why exact methods cannot help us when investigating other than simple products. Such problems are called *NP-hard*. For more complex problems, one thus has to rely on metaheuristic and heuristic methods. We will discuss a useful heuristic method in section 9.

8.2. Binary integer linear programming methods

8.2.1. Introduction

We can apply exact methods on state diagrams, AND/OR graphs, and disassembly precedence graphs. The first study, due to Johnson and Wang (1998), was on disassembly graphs. He used a *two-commodity network flow model* that is based on integer variables that account for the ordering of the operations in the sequence, which have to meet some set of precedence relationships. However, this approach is not the most efficient way to solve such problems, and we will derive modeling starting from an unconstrained network, which is actually a *traveling salesman problem*. Such a problem is a notoriously NP-hard one.

8.2.2. Unconstrained DPG

An example of an unconstrained problem is in figure 41b. Because $N = 6$, there are $N!$ = 720 possible complete sequences. The virtual node is considered the *source* here. We also introduce a sink, the node where the sequence stops. We confine ourselves to complete sequences. For the sake of simplicity, we will illustrate the model at the hand of a network with only four non-virtual nodes.
Because there are no precedence relationships, all the nonvirtual nodes are connected with each other via bidirectional arcs. Because the process can start at any non-virtual node, the source (with index 0) is connected to every non-virtual node. The process is finished when the sink (with index s) is attained. For the sake of symmetry in the continuity equations, there is an unidirectional arc from the sink to the source. As we only consider complete sequences, the sequence 0-s, which means that the product is not processed at all, is excluded.

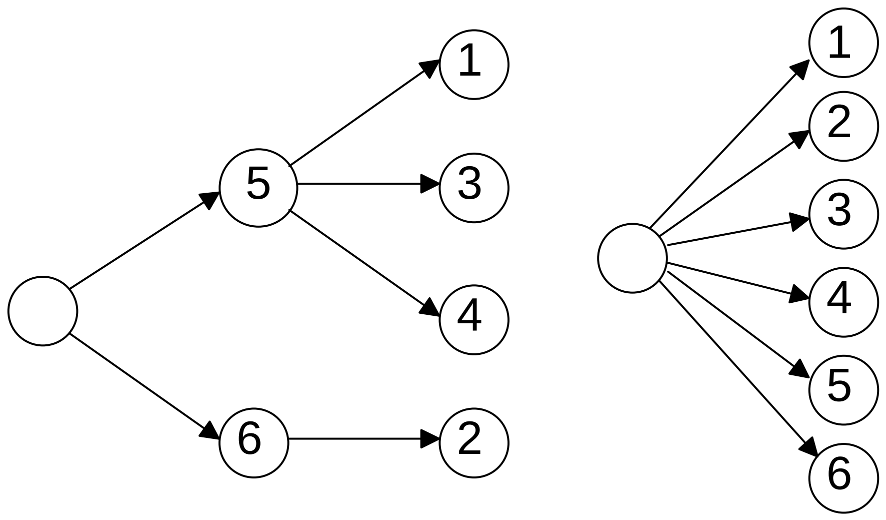

Figure 41. A constrained and an unconstrained problem.

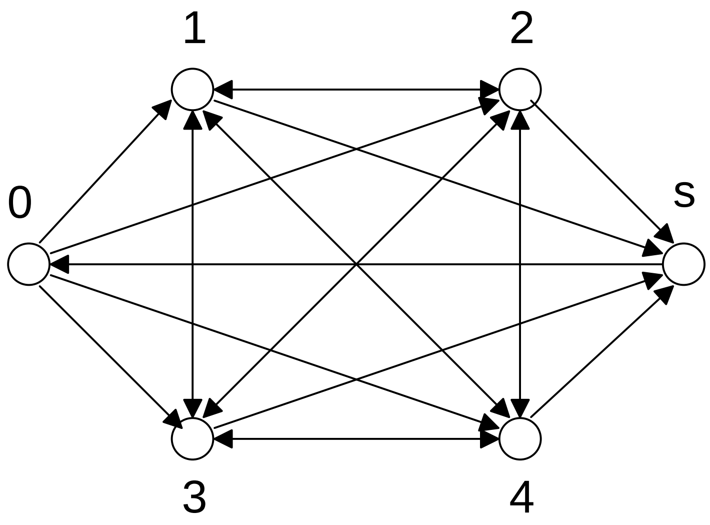

Figure 42. Unconstrained network with four non-virtual nodes.

The model is formulated as follows:
Because we consider a complete sequence here, the yields of the components are not included, as these are not affected by the selection of the sequence. The costs depend on the sequence, however. Therefore, a *cost matrix* in stead of a cost vector must be defined. For instance, the matrix element C_{jk} reflects the costs of performing operation k subsequent to operation j. As an operation can be done only once, the diagonal elements C_{jj} are put equal to zero.
The partial flow variables w_{jk} are defined as binary integer variables. These are put to 1 when operation k is performed subsequent to operation j, else the partial flow variable is zero.
The objective function thus reads:

$$C = \sum_{j,k} C_{jk} w_{jk} \tag{8.1a}$$

This function must be minimized.

Diagonal flows should be zero, hence:

$$w_{ll} = 0, \quad \forall l \tag{8.1b}$$

A continuity equation holds for any node:

$$\sum_j w_{jk} = \sum_j w_{kj} \tag{8.1c}$$

The initialization of the problem, guaranteeing that not a trivial 'all zero' solution is returned, is performed by the assignment:

$$w_{s0} = 1 \tag{8.1d}$$

The flow variables x_j are defined as follows:

$$x_j = \sum_k w_{kj} \tag{8.1e}$$

The flow variables must meet:

$$x_j = 1, \quad \forall j \tag{8.1f}$$

Because some arcs are unidirectional, we include:

$$\sum_{j \neq s} w_{j0} = 0 \tag{8.1g}$$

With this, it is apparently possible to solve the problem. Unfortunately, this formulation does not exclude *short tours*. Here we define the *long tour* as the sequence that includes the source and the sink, and a short tour, or cycle, is a tour that only visits nonvirtual nodes. For instance: the combined tours 0-1-s-0 and 2-4-3-2 can be returned as a solution of the problem. For this reason, traveling salesman problems are notoriously difficult.
It should be noticed that the problem has been presented here as a binary integer linear programming problem.

Example:
We ran the problem with 4 nonvirtual nodes and the cost matrix of figure 43.:

	1	2	3	4
0	40	84	25	69
1	0	46	9	63
2	74	0	16	13
3	74	29	0	26
4	72	21	80	0

Figure 43. Cost matrix in the example.

The initial solution reveals the long tour 0-1-3-s (or briefly written: 1-3), and the short tour 2-4-2, which is a 2-cycle. The objective is 83.

Inhibiting short tours might proceed rigorously, with is analogous to (8.2), which can be considered inhibition of 1-cycles. Inhibiting 2-cycles would result in the constraint:

$$w_{jk} + w_{kj} \leq 1, \quad \forall j,k \neq 0,s \tag{8.2a}$$

Inhibiting 3-cycles would result in:

$$w_{jk} + w_{kl} + w_{lj} \leq 1, \quad \forall j,k,l \neq 0,s\ , \tag{8.2b}$$

etc. Obviously, this will result in a huge amount of additional constraints, which makes the problem hardly executable for increasing problem size.

Inhibiting the short tours might also proceed specifically. In this case, we run the problem. If a short tour is encountered, we simply prohibit this specific tour.

Example:
In the instance of the small problem that already has been discussed, we encountered the short tour: 2-4-2. By adding the specific constraint:

$$w_{2,4} + w_{4,2} \le 1,$$

the specific short tour is eliminated. Running again the problem now reveals the solution:

1-3-2-4

which is a valid solution indeed.

The following simplification of the model can be done by letting source and sink coincide, see figure 44.

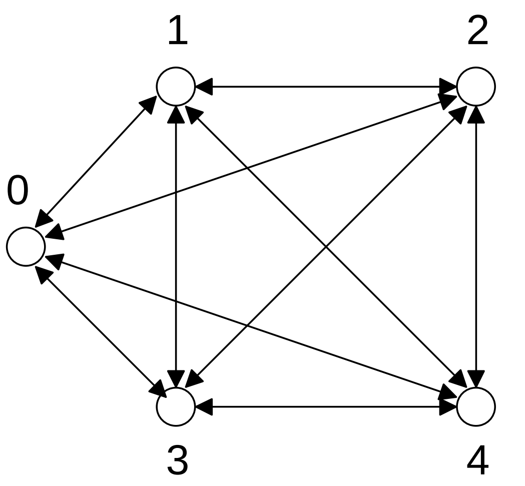

Figure 44. Simplified model with coincident source and sink.

By this, the model is completely symmetric. All the arcs are bidirectional, and the model simplifies to:
Minimize:

$$C = \sum_{j,k} C_{jk} w_{jk} \qquad (8.3a)$$

Subject to:

$$w_{ll} = 0, \quad \forall l \qquad (8.3b)$$

$$\sum_{j} w_{jk} = \sum_{j} w_{kj} \qquad (8.3c)$$

$$x_j = \sum_{k} w_{kj} \qquad (8.3d)$$

$$x_j = 1, \quad \forall j \qquad (8.3e)$$

$$w_{jk} \in \{0,1\}, \quad \forall j,k \qquad (8.3f)$$

Example:
We run this problem for $N = 6$, with the following instance of the cost matrix:

	1	2	3	4	5	6
0	40	84	25	69	73	80
1	0	46	9	63	42	86
2	74	0	16	13	36	51
3	74	29	0	26	73	16
4	72	21	80	0	1	42
5	16	42	60	26	0	60
6	26	73	42	36	73	0

Figure 45. Instance of the cost matrix for a problem with N = 6.

With this instance, we have the following solutions:

1. Long tour: 3-6; Short tour: 1-2-4-5-1; Objective 117.
   Inhibit the short tour.
2. Long tour: 1-3-6; Short tour: 2-4-5-2; Objective 121.
   Inhibit short tour.
3. Long tour: 3-6-1-2-4-5; Objective 127.

Consequently, the solution is revealed within two iterations, by adding two additional constraints, and without prohibiting all the possible cycles. Even in this case, there would be many of these cycles.

8.2.3. Constrained DPG
The constrained DPG of figure 41a is equivalent to that of figure 25c, except that the letters are replaced by figures here, for the sake of convenience. We can add some constraints in the mathematical model that reflect the precedence relations, such as:

$$w_{01} + w_{02} + w_{03} + w_{04} + w_{15} + w_{26} + w_{35} + w_{45} = 0$$

With the example that has already been discussed, we find the provisional solution:

1. Long tour 5-1-3-6; Short tour 2-4-2; Objective 148.
   Inhibit the short tour.
2. Long tour: 5-1-3-6-4-2; Objective 171.

Unfortunately, the set of constraints does not completely exclude those solutions that violate the precedence relationships. A solution such as 5-1-3-2-4-6 might be possible, although the subsequence 2-4-6 violates the precedence relationship. Rigorously inhibiting all the possible erroneous subsequences is a task comparable with that of inhibiting all the possible short tours. Therefore, we only exclude the shortest possible erroneous subsequence that is met, e.g.,

$$w_{24} + w_{46} \leq 1$$

This is equivalent with the extra constraints for inhibiting specific short tours.

8.2.4. Incomplete disassembly
Up to now, we considered complete disassembly only. If incomplete disassembly is permitted too, the relation (8.3e) is relaxed to:

$$x_j \leq 1, \quad \forall j \qquad (8.3e')$$

The objective function, (8.3a), changes into:

Maximize:

$$C = \sum_l Y_l \, x_l - \sum_{j,k} C_{jk} w_{jk} \tag{8.3a'}$$

Self-evidently, the yield vector should be defined. This includes the price of each component.

<u>Example</u>:
Let us consider the same instance of the product as before. We modify the model according to the two expressions above, and add the following instance for the yield vector:

$Y_1 = 99$; $Y_2 = 40$; $Y_3 = 93$; $Y_4 = 4$; $Y_5 = 60$; $Y_6 = 70$.

The following solutions are revealed:

1. Short tour 1-3-6-1; Short tour 2-5-2; Objective 233.
   Inhibit the short tour: 2-5-2.
2. Short tour: 1-3-6-1; Short tour: 2-5-4-2; Objective 232.
   Inhibit the erroneous subsequence: 4-2-5.
3. Short tour: 1-3-6-1; Short tour: 2-4-2; Objective 221.
   Inhibit the short tour: 2-4-2.
4. Short tour: 1-3-6-2-5-1; Objective 212.
   Inhibit the erroneous subsequence: 3-6-2-5.
5. Short tour: 1-3-6-1; Short tour: 2-5-4-2; Objective 232.
   Inhibit the erroneous subsequence: 4-2-5.
6. Long tour: 5-1-3-6; Objective 208.

Notice that the more valuable components are disassembled, and the less valuable ones are left unreleased.

8.2.5. Convergency considerations

The above mentioned *iterative method* is able to solve problems up to a definite degree of complexity. Unfortunately, we are confronted with decreasing convergence of the objective to its optimum value, in case the complexity of the problem increases. The number of iterations increases, and the CPU time tend to increase when the model is expanded with an increasing number of constraints. In spite of this, we were able to apply the method to products up to 25 components, in case only complete disassembly was considered. It must be noticed that the iterative method provides us with *superoptimal* quasi-solutions, because some structural constraints are violated. This means that these solutions, even when the iteration process is stopped before the optimum is attained, impose an upper limit on the profit. Heuristic methods, on the other hand, generate *suboptimum* solutions, that have lower profit than the maximum attainable. This is because the complete structure of the problem is incorporated in the heuristic. There exist rigorous methods that return the optimum. These will be discussed in the subsequent subsection. Unfortunately, these methods rely on full integer programming, which make them consume much CPU time. Iteration, that has been made explicit in the current subsections, is also present in the rigorous method, but embedded in solver via some branch-and-bound algorithm.

8.3. Integer linear programming method

8.3.1. Introduction

The integer linear programming method is used in traveling salesman problems. They rely on integer variables that act as a counter, which assigns an integer to each node in the network that is visited. In the type of problem we discuss, it is convenient to apply a decreasing counter, which starts with some value, which decreases by 1 for each subsequent node that is visited. The counter rigorously inhibits the short cycles. The counter is also applied for explicit introduction of the precedence relationships in the model.

8.3.2. Unconstrained model

We will apply the integer linear programming method to the problem that has already been discussed in subsection 8.2.4., at first the unconstrained version of it.

Example:
In case of the unconstrained situation with yields included, we find with the binary integer approach:

1. Short tour 1-3-6-1; Short tour 2-4-5-2; Objective 259.
Inhibit the short tour: 1-3-6-1.
2. Short tour: 1-2-4-5-1; Long tour: 3-6; Objective 249.
Inhibit the short tour: 1-2-4-5-1.
3. Short tour: 1-3-6-4-5-1; Objective 248.
Inhibit the short tour: 1-3-6-4-5-1.
4. Short tour: 2-4-5-2; Long tour: 1-3-6; Objective 245.
Inhibit short tour: 2-4-5-2.
6. Long tour: 3-6-1-2-4-5; Objective 239.

If counters are introduced, we have the rigorous model. Partial counters p_{jk} are introduces as integer variables. Aggregate counters a_l are introduced as well. Parameter N, equaling the number of nonvirtual nodes, is introduced. The model (8.3) is extended with four additional sets of constraint. It reads as follows:

Maximize:

$$C = \sum_l Y_l\ x_l - \sum_{j,k} C_{jk} w_{jk} \tag{8.4a}$$

Subject to:

$$w_{ll} = 0, \quad \forall l \tag{8.4b}$$

$$\sum_j w_{jk} = \sum_j w_{kj} \tag{8.4c}$$

$$x_j = \sum_k w_{kj} \tag{8.4d}$$

$$x_j \le 1, \quad \forall j \tag{8.4e}$$

$$w_{jk} \in \{0,1\}, \quad \forall j,k \tag{8.4f}$$

Initialization of partial counters:

$$\sum_j p_{0j} = N \tag{8.4g}$$

Aggregation of the counters:

$$a_l = \sum_j p_{jl}\,, \quad \forall l \tag{8.4h}$$

Counter decrement:

$$\sum_j p_{lj} = a_l - x_l \,, \ \forall l \qquad (8.4i)$$

Upper bound to partial counters:

$$p_{jk} \le (N+1) \cdot w_{jk} \,, \qquad \forall j,k \qquad (8.4j)$$

<u>Example</u>:
The model straightly returns the long tour 3-6-1-2-4-5. The following nontrivial values for the partial counters are returned:

$p_{0,3} = 6$; $p_{3,6} = 5$; $p_{6,1} = 4$; $p_{1,2} = 3$; $p_{2,4} = 2$; $p_{4,5} = 1$

The other p_{jk} equal zero.
For the aggregate counters, we have:

$a_3 = 6$; $a_6 = 5$; $a_1 = 4$; $a_2 = 3$; $a_4 = 2$; $a_5 = 1$

We indeed observe that these counters decrease by 1 at each subsequent node on the tour. This inhibits all kinds of short tour.

8.3.3. Constrained model
Rigorous introduction of the precedence relationships is possible in this model. According to figure 41a, we add the following constraints:

$a_5 \ge a_1 \qquad a_5 \ge a_3 \qquad a_5 \ge a_4 \qquad a_6 \ge a_2$

By this, the precedence relationships are completely introduced in the model, and the long tour 5-1-3-6 with objective 208 is returned once again.
If an operation is subjected to multiple precedence relationships, which corresponds to an OR relationship, the counter value that is assigned to any of the preceding nodes must be larger or equal than the succeeding one.
We notice that a decreasing counter is selected in the model in order to deal with zero values in case of incomplete disassembly.

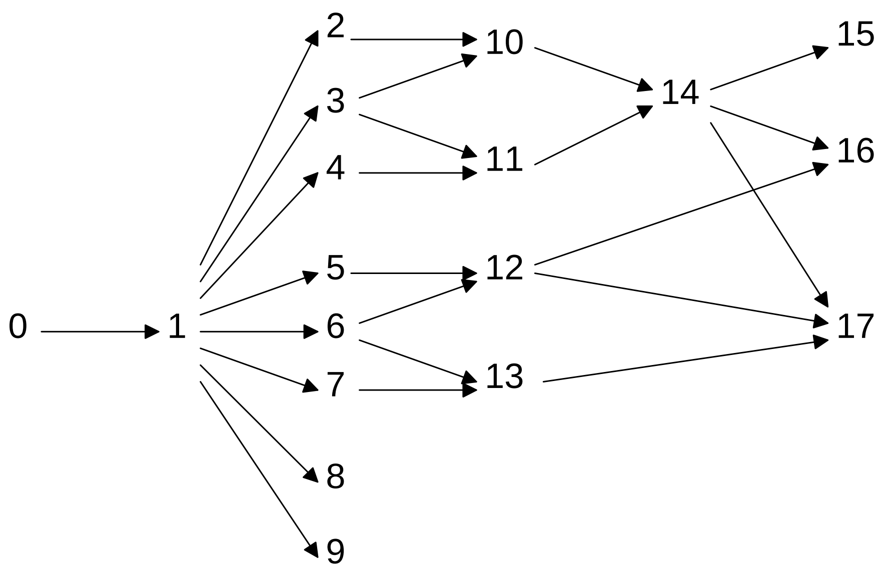

Figure 46. Larger problem (a PC) (Source: Gungor and Gupta, 1997).

Gradually extending the model with the subsequent nodes, we observe an exponentially increasing need for CPU time, particularly when increasingly more precedence relationships must be dealt with.

We found, for instance, the results that are listed in Table 8.1:

Table 8.1 CPU time required for rigorous calculation of an instance of the problem of figure 46, with increasing number of nodes.

N	CPU time (sec)	Sequence	Profit
6	0.55	1-3-6-4-5	224
7	0.28	1-3-6-4-5-7-2	268
8	0.6	1-3-6-4-5-7-8-2	313
9	0.98	1-3-6-9-8-2-4-5-7	411
10	1.15	1-3-6-7-2-4-5-10-9-8	461
11	7.53	1-3-7-2-4-5-10-9-8-6-11	507
12	14.99	1-3-6-9-8-7-2-4-5-10-12-11	617
13	64.92	1-7-2-3-6-13-4-5-10-9-8-12-11	698
14	296.22	1-7-2-3-6-13-4-5-10-12-11-8-14-9	730
15	n.a.	n.a.	n.a.

This table elucidates that increasingly complex solutions can hardly be solved by the rigorous method, for the required CPU time grows too lengthy for being of practical use. More powerful computers cannot help us definitely, as the problem persists in its need for an exponentially increasing amount of CPU time with increasing number of nodes, and precedence relationships. Specialized software, which is more or less tailored to the problem, could possibly help us further.
Although we can proceed indeed, along these lines, to some further extent by the explicitly iterative method, which relaxes the problem to a, simpler, binary integer problem, we are confronted with a similar problem, albeit at a higher degree of complexity, typically about 25 components. This puts a tight restriction on the application of exact methods. We have to rely on heuristic, or metaheuristic, methods, to deal with larger problems. In the following chapter, we will discuss a promising heuristic method for solving a similar type of problem that has been discussed here.

In appendix 1 we included an example of the AIMMS 2.20 code for Bourjault's ballpoint, solved via the iterative approach, which results in a binary integer linear programming model.

In appendix 2, an example of a AIMMS 2.20 code for the rigorous approach for the same problem is given, which results in an integer linear programming model.

Those parts of the code that represent data on the specific structure of the problem and the parameter values are marked by yellow.

Obviously, these codes can easily be transformed into alternative instances and alternative model structures, by modifying these parts. They can also be applied to other modeling environments.

9. Heuristic methods

9.1. General

Heuristic methods make use of a set of 'rules of thumb' that can be used for solving arbitrary complex problems. The reason is to avoid the exponentially increasing CPU times at the cost of some exactness, i.e., we cannot expect to attain the optimum solution, but rather some suboptimum solution that is often considered a 'good enough' solution, although it is not trivially known how far this solution is removed from the optimum, and how many solutions are possible between the solution that is generated by the heuristic, and the optimum solution.
Of course, as has been mentioned already, the exactness of the 'exact' solution is disputable as, in practice, the parameter values are rather estimates or stochastic averages. It is, on the other hand, beneficial to know the exact solution, in order to evaluate the results that have been obtained by heuristic methods and, by this, the heuristic method itself.

9.2. Greedy method

The greedy method works along the lines many politicians and other decision makers do: they restrict their scope to the minimum possible thus being not aware of advantages that might appear in a more remote future. When translated this to sequencing issues, the planning horizon is to the next operation only.

Table 9.1. Instance of the product in figure 46, with the first 14 components.

```
yield := TABLE
         1   2   3   4   5   6   7   8   9  10  11  12  13  14
!       --  --  --  --  --  --  --  --  --  --  --  --  --  --
1       99  40  93   4  60  70  52  79  80  63  87  76  95  63
,

cost := TABLE
         1   2   3   4   5   6   7   8   9  10  11  12  13  14
!       --  --  --  --  --  --  --  --  --  --  --  --  --  --
0       40  84  25  69  73  80  75  16  60  63  86  73  33  65
1        0  46   9  63  42  86  45  45  51  44  52  96  97   8
2       74   0  16  13  36  51  92  41  86  20  68  97  39  55
3       74  29   0  26  73  16  18  83  80  75  66  79  57  82
4       72  21  80   0   1  42  66  16   1  79  41  26  13  94
5       16  42  60  26   0  60  32  69  73  11  69  70  99  88
6       26  73  42  36  73   0  56  56  36  16  30  53   5  97
7       69  16  83  41  52  75   0  32  42  92  53  33  65  79
8       66  18  92  45  75  65  41   0  73  53  96  19  16  16
9       26  60  42  16  51  86  80   1   0  18  66  10  60  93
10      97  39  63  44  10  74  79  11  16   0  69   1  50  20
11      36  86  52  68  66  41  69  30  52  96   0  76  54  63
12      75  96  97  79  26  70  53  33  19  10   1   0  95  65
13      87  39  57  13  99   5  65  16  60  50  87  16   0   8
14      55  82  94  88  97  79  16  92  20   8  16  61  67   0
```

Considering the instance of the problem in Table 9.1, we must have operation #1 as the first in the sequence, because of the precedence relationships, see figure 46. It has profit 99 – 40 = 59.
For the second operation, we have to select from 2 thru 9. The partial profits that result from these operations are, respectively: – 6, 84, – 59, 18, – 16, 7, 34, and 29. Consequently, operation #3 will appear the best.

After this, the operations 2, 4, 5, 6, 7, 8, or 9 are possible. Operation #6 with partial profit 54 is selected.
Proceeding, operations 2, 4, 5, 7, 8, or 9 are possible. Operation #9 with partial profit 44 is selected.
Next, operations 2, 4, 5, 7, or 8 are possible. Operation #8 with partial profit 78 is selected.
Next, operations 2, 4, 5, or 7 are possible. Operation #2 with partial profit 22 is selected.
Next, operations 4, 5, 7, **or 10** are possible, see figure 46. Operation #10 with partial profit 43 is selected.
Subsequently, operations 4, 5, or 7 are possible. Operation #5 with partial profit 50 is selected.
Next, operations 4, 7, **or 12** are possible. Operation #7 with partial profit 20 is selected.
Next, operations 4, 12, **or 13** are possible. Operation #12 with partial profit 43 is selected.
Next, operations 4 or 13 are possible. Operation #4 results in partial profit – 79, and operation #13 results in partial profit 0.
If incomplete disassembly is permitted, the sequence stops here. It reads:

0-1-3-6-9-8-2-10-5-7-12

and it results in profit: 479.
If complete disassembly only is permitted, operation #13 with partial profit 0 is selected.
Next, operation 4 only is possible. Operation #4 with profit – 9 is selected.
Next, operation #11 is possible and selected. It results in profit 46.
Next, operation #14 is possible and selected. It results in profit 0.
As a result, the sequence:

0-1-3-6-9-8-2-10-5-7-12-13-4-11-14

with profit 516 is selected.
It can be noticed that the profit has increased while extending the sequence, although it would have been cut off in an earlier phase when no complete disassembly were required.
From the rigorous solution method, we detected a quite other sequence, viz.

0-1-7-2-3-6-13-4-5-10-12-11-8-14-9

with profit 730. This means that the greedy solution results in a suboptimum solution with a profit 29% lower than optimum.

Analysis of the procedure reveals that the yield of component 4 is extremely low, which inhibits the execution of operation 4. If not performed, operations 11 and 14 cannot be performed as well, although these can potentially generate profit. In the result of the rigorous calculation it is seen that operation 4 in the sequence is in a somewhat intermediate position, thus enabling performing subsequent operations.

9.3. Alternative heuristics

For enabling the performance of as many as possible operations as possible, we can consider an alternative criterion, which involves the disclosure of as many as possible new operations. From this point of view, we define the number of operations that are positioned downstream of any operation. This

For the product with the DPG of figure 46, we have the following amounts of operations that are upstream, see table 9.2:

Table 9.2. Number of downstream operations for the PC of figure 46.

Operation	Operations downstream	Id, $N = 14$
0	17	14
1	16	13
2	5	2
3	6	3
4	5	2
5	3	1
6	4	2
7	2	1
8	0	0
9	0	0
10	4	1
11	4	1
12	2	0
13	2	0
14	3	0
15	0	-
16	0	-
17	0	-

0-1-3-2-4-11-6-10-5-14-7-12-13-.....

If N =14, we arrive at:

0-1-3-6-2-4-5-10-11-7-8-13-12-9-14

with profit 637 in case complete disassembly is required, or:

0-1-3-6-2-4-5-10-11-7-8-13-12-9

with profit 667 in case incomplete disassembly is permitted. This is about 9% removed from the optimum solution, which is still better than the greedy result.

9.4. Extended greediness

In stead of the greedy procedure, we can apply *extended greediness*, which implies that we take a set of promising solutions of the size Λ, which is an integer. This parameter is called *greediness parameter*. Starting with the source, i.e. node 0, we place it in a list of a specified size, e.g., 40 positions. Next, we expand this node. The only possible expansion in the example of figure 46 is 0-1. The profit of this subsequence is calculated according to table 9.1. This reveals the value 59. This is also placed in the list, which is always in descending order of profit, which implies that some potential subsequences with relatively low profit are removed from the list and not further investigated. Next, the subsequence 0-1 is expanded. This potentially reveals the subsequences: 0-1-2, 0-1-3, 0-1-4, 0-1-5, 0-1-6, 0-1-7, 0-1-8, and 0-1-9. With Λ = 2, e.g., only those two with maximum profit are selected. These are also placed on the sorted list. Finally, when the expansion is complete, we end up with a list of suboptimum solutions that possibly includes the optimum sequence.

We have written some software in VisualBasic 6.0 that is flexible, as the value of Λ, the size of the model, the structure of the DPG, the instance of the model and the size of the list can be adapted to a variety of problems. It appears that results are obtained at the expense of a very short CPU time, which is less than 1 sec so far models of moderate complexity are considered.

We elaborated a problem on the basis of a small product, viz., that of figure 46, but restricted to the nodes 1 thru 6. The instance of the product is in table 9.1. There are 120 complete sequences possible, and 327 general sequences, which makes enumeratively solving the problem possible.

Table 9.3. Greedy solutions of the PC problem, N = 6.

$\Lambda = 1$	
Sequence	**Profit**
1-3-6	197
1-3-6-5	184
1-3-6-5-2	182
1-3-6-5-2-4	173
1-3	143
1	59
0	0

Exercise:
Calculate manually the suboptimum solution with the greedy method, both for complete sequences only and in case incomplete sequences are also permitted.

From table 8.1, we observed that the optimum solution has profit 224, which implies that the suboptimum solution has a profit which is 12% lower than optimum. A question that can not be answered, even when rigorous solution appears possible, refers to the number of suboptimum solutions that exceed the presently detected solution. For this is a problem with restricted size, it can be expected that this number will not be quite large. We have listed the results of the calculation for increasing values of Λ in table 9.4.
The sequences that newly appear at increasing Λ are highlighted. The 9 best sequences are listed here, and we also listed the complete set of solutions with profits exceeding the value 200. We found 10 of these solutions.
Notice that one of the solutions with profit 181 temporarily disappears at $\Lambda \geq 5$, while reappearing at $\Lambda \geq 9$. But then it is not longer listed, as we have a sufficient amount of solutions that are closer to optimum then.
The optimum solution appears at greediness parameter values $\Lambda \geq 6$.
Although there are restrictions in the software to extend the value for Λ to arbitrary high values, dependent on the amount of memory, the size of this small model enables Λ-values up to 987. It should be realized that in case of increasing Λ, the case of full enumerative solution is approached. For models with increasing N, the search space grows too large to perform this. With $N = 25$, which corresponds to a medium sized product, the size of the search space will already be in the order of magnitude of Avogadro's number, i.e., the number of atoms in a macroscopic quantity of matter. Nevertheless, it appears possible to detect solutions that are extremely close to the optimum value, if not the optimum value itself. This is comparable with finding a selected atom in a macroscopic quantity of matter indeed.

Table 9.4. Suboptimum solutions for the PC problem, N = 6, increasing Λ.

$\Lambda = 1$		$\Lambda = 2$		$\Lambda = 3$		$\Lambda = 4$	
Sequence	**Profit**	**Sequence**	**Profit**	**Sequence**	**Profit**	**Sequence**	**Profit**
1-3-6	197	1-3-6	197	1-3-6	197	1-3-6	197
1-3-6-5	184	1-3-2-5-6	188	1-3-2-5-6	188	1-3-2-5-6	188
1-3-6-5-2	182	1-3-6-5	184	1-3-6-5	184	1-3-6-5	184
1-3-6-5-2-4	173	1-3-6-5-2	182	1-3-6-5-2	182	1-2-3-6	184
1-3	143	1-3-2-5	178	1-3-6-5-4-2	181	1-3-6-5-2	182
1	59	1-3-6-5-2-4	173	1-3-2-5	178	1-3-6-5-4-2	181
0	0	1-3-2-5-6	156	1-3-2-6	173	1-3-2-5	178
		1-3-2	154	1-3-6-5-2-4	173	1-3-2-6	173
		1-3	143	1-3-6-5-4	162	1-3-6-5-2-4	173

$\Lambda = 5$		$\Lambda = 6$		$\Lambda = 7$		$8 \le \Lambda \le 10$	
Sequence	**Profit**	**Sequence**	**Profit**	**Sequence**	**Profit**	**Sequence**	**Profit**
1-3-6	197	1-3-6-4-5	224	1-3-6-4-5	224	1-3-6-4-5	224
1-3-4-5-2-6	197	1-3-6-4-5-2	222	1-3-6-4-5-2	222	1-3-6-4-5-2	222
1-3-4-5-6	190	1-3-6-4-2-5	208	1-3-6-4-2-5	208	1-3-6-4-2-5	208
1-3-2-5-6	188	1-3-6	197	1-3-6	197	1-5-2-3-6	206
1-3-6-5	184	1-3-4-5-2-6	197	1-3-4-5-2-6	197	1-3-6	197
1-2-3-6	184	1-3-4-5-6	190	1-3-4-5-6	190	1-3-4-5-2-6	197
1-3-6-5-2	182	1-3-2-5-6	188	1-3-2-5-6	188	1-3-4-5-6	190
1-3-4-5	180	1-3-6-5	184	1-3-6-2-5	188	1-3-2-5-6	188
1-3-2-5	178	1-2-3-6	184	1-3-6-5	184	1-3-6-2-5	188

$11 \le \Lambda \le 16$		$\Lambda = 17$		$18 \le \Lambda \le 22$		$23 \le \Lambda \le 32$	
Sequence	**Profit**	**Sequence**	**Profit**	**Sequence**	**Profit**	**Sequence**	**Profit**
1-3-6-4-5	224	1-3-6-4-5	224	1-3-6-4-5	224	1-3-6-4-5	224
1-3-6-4-5-2	222	1-3-6-4-5-2	222	1-3-6-4-5-2	222	1-3-6-4-5-2	222
1-3-2-4-5-6	214	1-3-2-4-5-6	214	1-3-2-4-5-6	214	1-3-2-4-5-6	214
1-3-6-4-2-5	208	1-3-6-2-4-5	214	1-3-6-2-4-5	214	1-3-6-2-4-5	214
1-5-2-3-6	206	1-3-6-4-2-5	208	1-2-3-6-4-5	211	1-2-3-6-4-5	211
1-3-2-4-5	204	1-5-2-3-6	206	1-3-6-4-2-5	208	1-3-6-4-2-5	208
1-3-6	197	1-3-2-4-5	204	1-5-2-3-6	206	1-5-2-3-6	206
1-3-4-5-2-6	197	1-3-6	197	1-3-2-4-5	204	1-3-2-4-5	204
1-3-4-5-6	190	1-3-4-5-2-6	197	1-3-6	197	1-3-2-6-4-5	200

$33 \le \Lambda \le \infty$	
Sequence	**Profit**
1-3-6-4-5	224
1-3-6-4-5-2	222
1-3-2-4-5-6	214
1-3-6-2-4-5	214
1-2-3-6-4-5	211
1-3-6-4-2-5	208
1-5-4-2-3-6	205
1-5-2-3-6	206
1-3-2-4-5	204
1-3-2-6-4-5	200

The optimum sequence is obtained here for $\Lambda \ge 6$. The table reveals here the complete ordered list of suboptimum solutions. This cannot be guaranteed at increasing N, for Λ cannot be grow infinitely high. It appears possible, however, to reveal a list of suboptimum solutions. Each of these can be evaluated with respect to alternative criteria. For instance, if component 2 must be detached anyway, e.g. because of regulation, one should prefer the second best option, with profit 222. It is also possible to check the complete list of 10 sequences with profit ≥ 200 on other criteria and thus select the sequence that is good enough on the cost criterion on characteristics that are independent of costs, e.g., environmental performance.

9.5. Partial branch and bound method

Although, for the relatively simple model discussed in the previous subsection, all the potential solutions were easily detectable, this is not longer possible for more complex models.

If we, e.g., proceed with the model for N = 10, the optimum solution is revealed when $\Lambda \ge 510$, at the expense of a CPU time of about 1.5 sec on a 2.6 GHz computer.

For N = 11, the optimum solution is obtained for $\Lambda \ge 130$, at the expense of negligible CPU time.

For N = 12, the optimum solution is obtained for $\Lambda \ge 79$, at the expense of negligible CPU time.

For $N = 13$, the optimum solution is obtained for $\Lambda \geq 211$, at the expense of CPU time about 0.5 sec.
For $N = 14$, the optimum solution is obtained again for $\Lambda \geq 211$, at the expense of CPU time about 0.5 sec.
Although obtaining the optimum value is obviously simple with this heuristic, there is a tendency to ever increasing values for Λ with increasing complexity, although the behavior of this dependency is more or less irregular.
The more or less whimsical nature of the profit vs. Λ curve is demonstrated in figure 47 for the current instance of the model of the PC with $N = 14$.

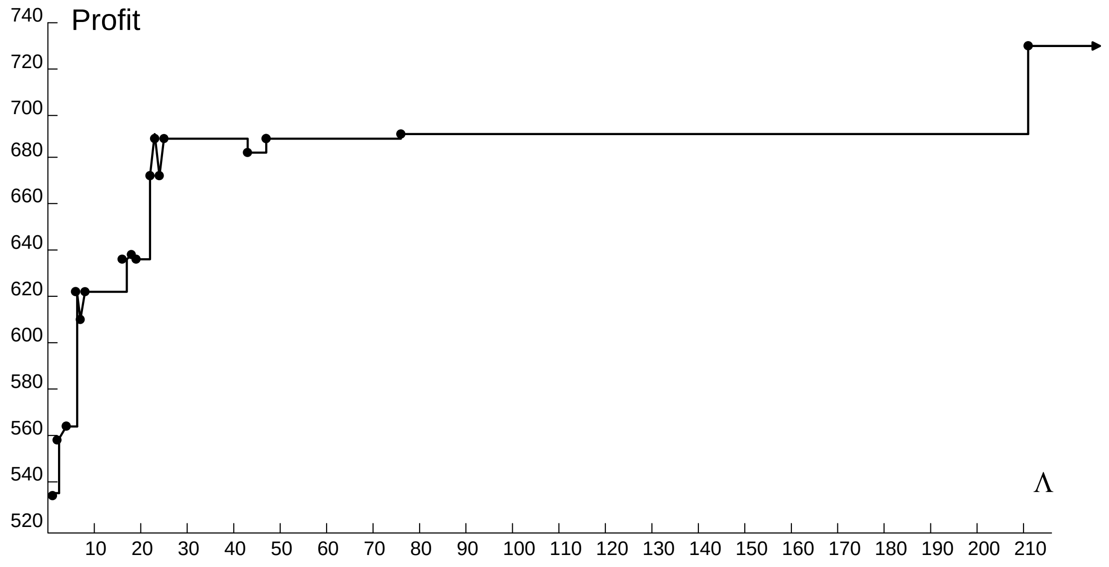

Figure 47. Profit vs. Λ curve for the PC model with N = 14.

Fortunately, there exists a method that reveals suboptimum solutions in a more efficient way. It starts with the virtual node and expands the sequences one by one. A fixed number of sequences is further investigated. This fixed number has been give the symbol Γ. It represents the number of expansions that is further investigated.
Let us illustrate this method by am example.
We select $\Lambda = 200$, and start with node 0. As a result, we obtain a list of sequence, the best of it has profit 688, which meets with figure 47. Next we expand this node with one position in the sequence, which can be node 1 only, thus attaining sequence: 0-1. This can be fixed in the program, by assigning the value 1 to the first position in the sequence and assigning the value 1 to the parameter INITCOUNT, which represents the number of predefined positions. The calculation results in an unchanged solution.
Next we expand the sequence 0-1. This reveals eight subsequences, viz.:

0-1-2; 0-1-3; 0-1-4; 0-1-5; 0-1-6; 0-1-7; 0-1-8; 0-1-9.

The parameter INITCOUNT is put to 2, and the best solutions of the subsequences have profit, respectively:

674, 729, 685, 714, 629, 730, 672, and 701.

From these, we can select a number of most promising sequences for further expansion. Let us take the best two of them. This is expressed by the expansion parameter Γ = 4.
Obviously, the optimum sequence has been detected already. It even appears when Λ ≥ 24.
Figure 48 depicts some expansions in case $\Lambda = 20$. At any step, the 4 best solutions were taken, as a consequence of the selected value $\Gamma = 4$. We stopped after 4 posi-

tions, as no major changes occurred after this. Because the output of every solution is actually a *list* of solutions, we are able to detect more solutions than indicated in the graph, which are close to optimum. Although this might appear a fairly complete list, it is not certain that it contains the full set of near optimum solutions, as some branches have been cut without investigating these to their full extent. This is the sacrifice we have to make in exchange for an extremely short CPU time.

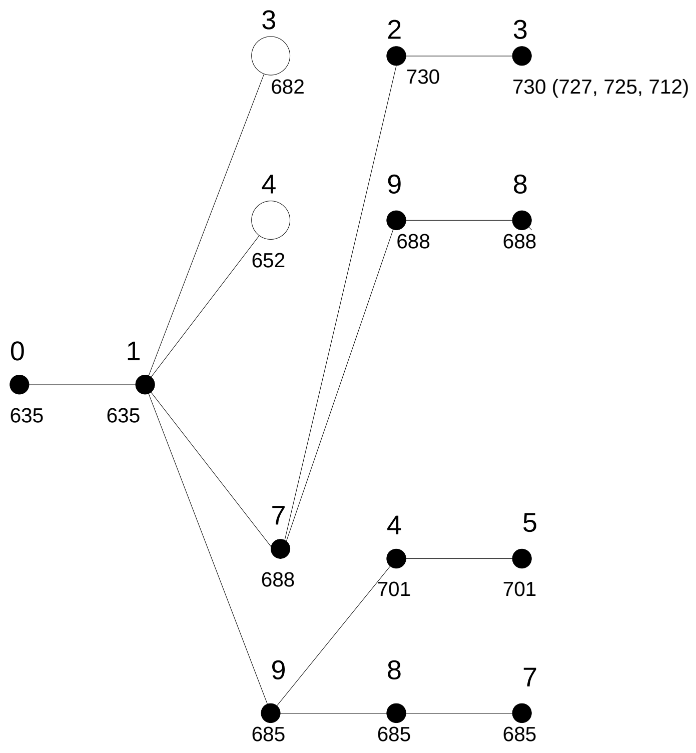

Figure 48. Partial branch and bound for the PC example (Λ = 20; Γ = 4).

We start with INITCOUNT = 0 and find the most favorable solution with profit 635. Next, we put INITCOUNT = 1 and assign the initial subsequence 0-1. We again find the same solution. Next we put INITCOUNT = 2 and expand the initial subsequence. The subsequences 0-1-2, 0-1-3, 0-1-4, 0-1-5, 0-1-6, 0-1-7, 0-1-8, and 0-1-9 must be investigated. From these, the most promising subsequences are 0-1-3, 0-1-4, 0-1-7, and 0-1-9, with objectives 682, 652, 688, and 685, respectively. We proceed with expanding the four selected subsequences one position further, and we repeat selecting the most promising four of them. Obviously, some of the promising subsequences are abandoned, which is indicated by an open circle. The other ones are further expanded. Fortunately, the optimum sequence is obtained in the third expansion. Because we obtain a list of solutions in stead of a single solution, some other favorable suboptimum solutions are also found. The objectives of these are placed within brackets. We have detected five sequences with objective exceeding 700.

Although it appears possible by this partial branch and bound to obtain the optimum solution at a fairly lower value for *Λ* than without this method (20 vs. 211), it is not possible to find the optimum at an arbitrary low value for *Λ*, unless *Γ* is increase up to an impractical value. This is demonstrated by the same exercise, with *Λ* = 10. With *Γ* = 4, the optimum is never obtained, and the maximum value for the optimum equals 688, which is not only 6% below the optimum, but also neglects a considerably populated set of solutions that are closer to optimum. The partial branch and bound picture for Λ = 10 is depicted in figure 49.

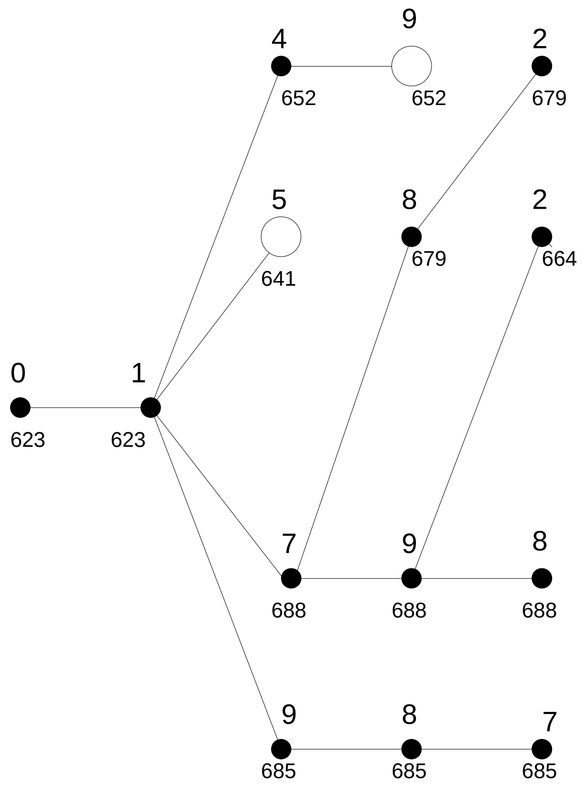

Figure 49. Partial branch and bound for the PC example (Λ = 10; Γ = 4).

At a considerably higher value for Λ, which still appears practicable, we detect many more solutions that are close to the optimum. We even obtain 8 sequences that exceed the value 720, i.e., their objective is about 1% or less removed from the optimum. Although not can be guaranteed that this is the complete set of solutions, it might be conceivable that we have obtained a considerable part of all the close-to-optimum solutions, if not all of them. With such a set of close-to-optimum solutions, we can perform multiple criteria analysis, which implies the adjustment of a modest set of sequences with respect to a set of various requirements, which might include environmental aspects as well as technical issues.
The partial branch and bound tree for Λ = 200 is depicted in figure 50.

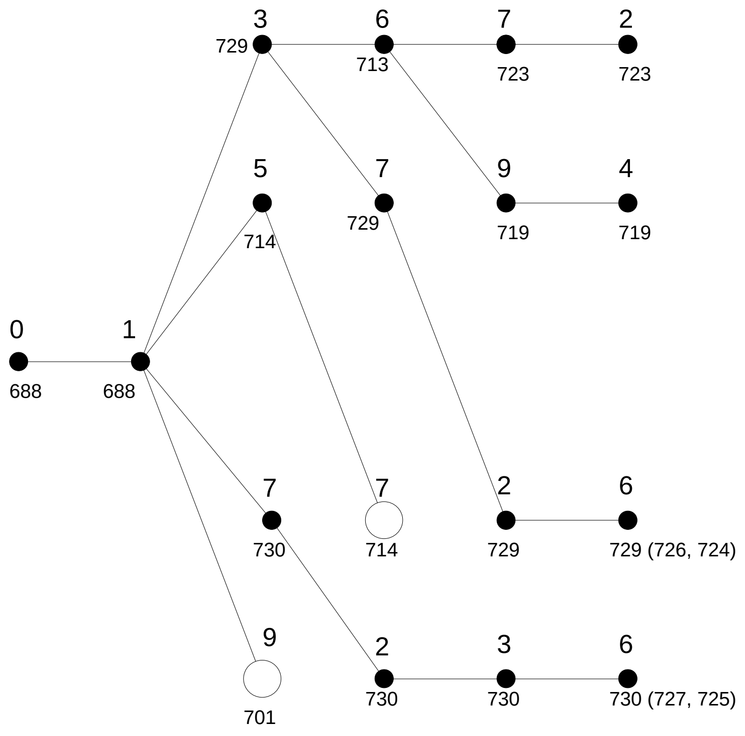

Figure 50. Partial branch and bound for the PC example (Λ = 200; Γ = 4).

Here we observe a rapid convergence to the optimum, and the appearance of alternate promising subsequences that can be investigated into more detail.
The source text of the program can be found in Appendix 3. The adaptable parameters are marked with yellow, and those parts of the program that represent the structure of the DPG and the instance of the combined costs and yields are marked with green.

9.6. Conclusions

We presented here an extremely quick and adaptable heuristic, which can be applied to a variety of problems. The exact methods that have been discussed in previous sections can be applied as a tool for evaluating the heuristics. It should be stressed that the heuristics can be modified by two parameters: the greediness parameter, *Λ*, and the depth parameter of partial branch-and bound, *Γ*. The appropriate selection of the parameter values depends on the model's structure and instance.

10. Disassembly-to order problems

10.1. Introduction

Up to present, we considered disassembly optimization only, without considering dynamical aspects. These include a demand to various components that changes over time. Apart from this, we want to extend our domain of application to families of related products, that have different structure but that might contain one or more components of the same type in common. This is called: *commonality*. Above this, it is permitted that multiple items of the same type of component appear in a single product, which might be in different places of the structure. This is called: *multiplicity*.
With different products of a family in stock, the problem consists of selecting the appropriate mix of different products aimed at meeting the demand at minimum costs. This is called the *disassembly-to order problem*.

10.2. Hierarchical tree structure

Because the problem that is discussed here appears to be more complicated than optimizing the disassembly of a single product type, we must rely on a model of the product's structure that is simpler than the full mechanical structure such as encountered in case of Bourjault's ballpoint, see figure 19, and the automatic transmission of figure 21. We will even not deal with products from which the disassembly process is represented by a rather complex disassembly precedence graph, such as the PC example in figure 46. Therefore, we restrict ourselves to products that are organized along a hierarchical tree structure, such as in the monitor example in section 3, figure 14. This approach is valid for cases that are characterized by a modular structure, such as PCs. In this case, the components that are 'harvested' are represented by *leaves* and the obstructions that have to be passed by are represented by *(sub-)roots* in a completely divergent network that resembles a branched tree. An example of such a tree is presented in figure 51. The root numbers 1 thru 5 in the figure are assigned to the roots. Obviously, the number of roots can be different in the various products of a family. Putting a root apart enables both the removal of definite leaves and the putting apart of some subroots. By this, precedence relationships are introduced.
For example, when root number 1 is put apart in the product of type $i = 1$ in figure 51, the leaf P_1 can be removed, but it has been also made possible to put apart the roots 2, 3, and 5. The subscript refers to the component type number. The figure that is placed in front of the component symbol refers to the multiplicity of this component. For instance, the component P_{11} occurs twice at the position of subroot 4 in a product of type 1. In this model, the costs are assigned to putting apart the roots. These are more or less equivalent with the operations in the earlier discussed models. The leaves represent the valuable components. Leaves meet the demand on the corresponding components. A yield is realized by the removal of a component.
The basic problem consists of finding the optimum batch of products for meeting a predefined demand on components of the different types.
Roots are equivalent with operations that must precede the removal of definite leaves and/or the putting apart of subordinate roots. The basic root might be, e.g., the removal of a casing that is obstructing the removal of the leaves. Leaves correspond to valuable components and/or modules. In the PC example this might be CD-ROM drives, power supply units, printed circuit boards, microprocessor chips, etc. The product thus is considered a collection of valuable components or modules that are arranged in some predefined way, which requires the performance of some predefined operations, such as the removal of mechanical parts that are obstructing the removal

of the valuable leaves. Obviously, some hazardous components that must be isolated from a regulation point of view, are also included in the set of leaves. The requirement for removal of these components can be considered a generalized demand as well.

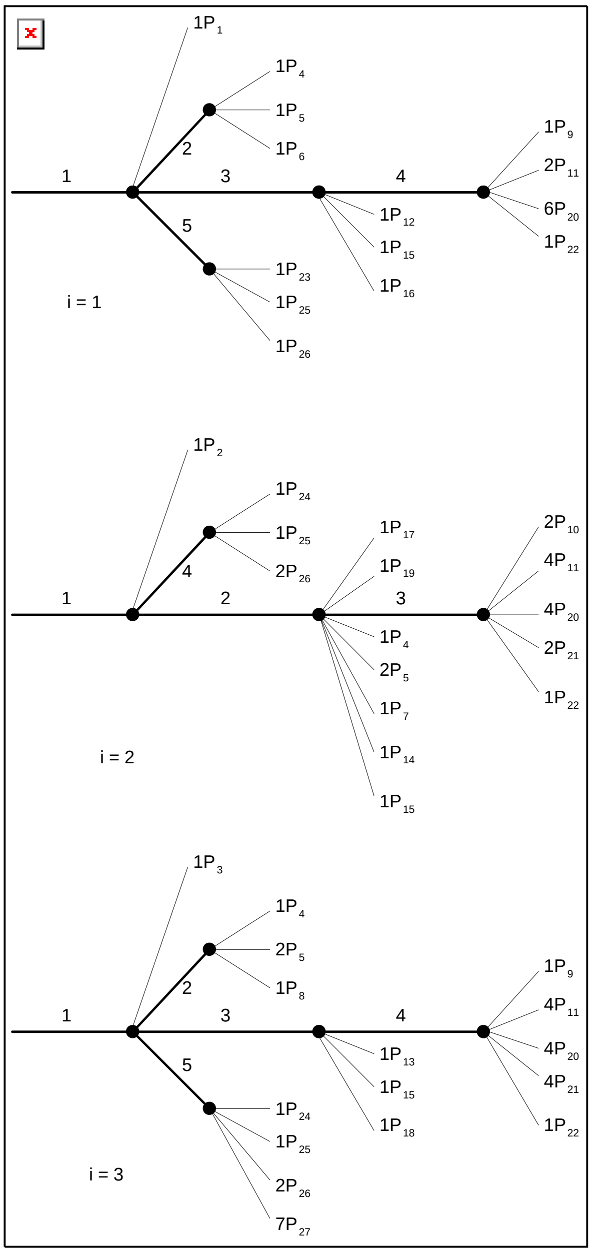

Figure 51. Hierarchical tree structures for three products in a product family.

10.3. Model description

10.3.1. General

The general problem is to find the optimum batch of products that must be disassembled to meet a predefined demand. This demand refers to all the different leaves that might be put free in disassembling the different products in a family. In this problem, there is no need to incorporate the yields of the leaves, because these depend on the demand only. Costs are assigned to the putting apart of the various roots, to the acquisition of the products, and to the storage of partly disassembled products. Costs might be put negative in appropriate cases. Constraints might be put on the amount of products of a specific type that can be acquired, on the capacity of some operations, etc. We will study a model that refers to an instance of the case with the product family of figure 51. First a single product model is discussed which will be extended to a multi-product model in a subsequent subsection.

10.3.2. Single product model

We discern three types of index.

- The index *i*, running from 1 thru *I*, refers to the product type. In figure 51, $I = 3$. In the single product model, the index *i* is omitted.
- The index *p*, which runs from 1 thru *P*, refers to the component type. In figure 51, $P = 27$.
- The index *j*, which runs from 1 thru *J*, refers to the root number. In figure 5, $J = 5$, which is the maximum number of roots in a product.

The *demand vector D* refers to the demand on any type of component at a definite point of time. Obviously, vector *D* has *P* positions, e.g.:

$D_p = [0\ \ 0\ \ 0\ \ 0\ \ 550\ \ 0\ \ 0\ \ 0\ \ 400\ \ 0\ \ 1150\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 1250\ \ 0\ \ 0\ \ 0\ \ 0\ \ 580\ \ 450\ \ 0]$

The *flow variables* x_j refer to whether or not root number *j* is put apart. Because we presently speak about a discrete number of products, the x_j are integer variables that refer to the number of products that must be disassembled to meet the demand.
If we restrict ourselves to the product of type $i = 1$, and we permit partial disassembly, the precedence relationships of the roots are given by the following constraints:

$$x_2 \le x_1; \qquad x_3 \le x_1; \qquad x_5 \le x_1; \qquad x_4 \le x_3$$

This can be condensed by means of the *tree structure matrix S*, which reads:

$$S_{j,j'} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad (10.1)$$

This matrix contains the information on the roots *j'* that can be put apart once root *j* has been put apart. The exception is the (1,1) element, which is always put to 1. Notice, e.g., that release of root 1 enables the release of roots 2, 3, and 5.
The precedence relationships condense to one single expression:

$$x_{j'} \le \sum_{j=1}^{J} S_{j,j'}\, x_j \qquad (10.2)$$

Putting $S_{1,1}$ equal to 1 is required for enabling $x_1 \ge 1$.

The *yield matrix* $Y_{j,p}$ represents the number of components of type *p* that are available once root *j* has been put apart. For the product type i = 1, it reads:

$$Y_{j,p} \quad = \quad \begin{bmatrix} 1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&1&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&1&0&2&0&0&0&0&0&0&0&0&6&0&1&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&1&0 \end{bmatrix} \tag{10.3}$$

<u>Exercise</u>:
Determine the yield matrices for the products of type $i = 2$ and $i = 3$.

In the 11th column we observe that 2 items of component $p = 11$ are available once root $j = 4$ has been put apart.
The *yield variables y_p* refer to the real yield of component p after disassembling the required batch of products. It is bound to a maximum via the *yield constraints*:

$$y_p \leq \sum_{j=1}^{J} Y_{j,p}\, x_j \tag{10.4}$$

Because the yield must meet the demand, we add this condition via:

$$y_p = d_p \tag{10.5}$$

The objective function is not the profit here, because the financial yield of the components only depend on the demand. The costs depend on the decision variables, viz., the number of products that must be disassembled. Therefore, it is the costs that must be minimized. Costs include the supply costs, the disassembly costs, and the waste disposal costs.
The *supply costs* are given by:

$$C_{s,tot} = CS \cdot x_1 \tag{10.6}$$

In this expression, the parameter C_s represents the costs for acquisition of a single item of the product. The value x_1 refers to the number of products that are processed.
The *disassembly costs* are given by:

$$C_{d,tot} \quad = \quad \sum_{j=1}^{J} C_j\, x_j\ , \tag{10.7}$$

The C_j refer to the components of a cost vector that presents the costs of putting apart root j.
The *waste disposal costs* are given by:

$$C_{w,tot} = \sum_{p=1}^{P} CW_p \cdot \left[\left(x_1 \cdot \sum_{j=1}^{J} Y_{j,p}\right) - D_p\right] \tag{10.8}$$

This is the difference between the components that are actually present in the processed product, and those that are disassembled aimed at meeting the demand.
Self-evidently, the objective function reads:

$$\boldsymbol{Obj} = C_{tot} = C_{s,tot} + C_{d,tot} + C_{w,tot} \tag{10.9}$$

10.3.3. Multiple product model

Extending the above mentioned theories to multiple products can be straightforwardly done by adding index i to the different expressions. The both the flow and yield variables thus have two indices now. The constraints read:

$$x_{i,j'} \leq \sum_{j=1}^{J} S_{i,j,j'} \, x_{i,j} \tag{10.2a}$$

$$y_{i,p} \leq \sum_{j=1}^{J} Y_{i,j,p} \, x_{i,j} \tag{10.4a}$$

$$\sum_{i=1}^{I} y_{i,p} = d_p \tag{10.5a}$$

The partial costs read:

$$C_{s,tot} = \sum_{i=1}^{I} CS_i \cdot x_{i,1} \tag{10.6a}$$

$$C_{d,tot} = \sum_{i=1}^{I}\sum_{j=1}^{J} C_{i,j} \, x_{i,j} \, , \tag{10.7a}$$

$$C_{w,tot} = \sum_{p=1}^{P} CW_p \cdot \left[\sum_{i=1}^{I} \left(x_{i,1} \cdot \sum_{j=1}^{J} Y_{i,j,p} \right) - D_p \right] \tag{10.8a}$$

The model, presented in AIMMS 2.20 code, is elaborated in Appendix 4. As can be seen, the disassembly costs and the disposal costs are multiplied by the factors 0.2 and 0.25 respectively, which has been done to be consistent with the case discussed by Veerakamolmal and Gupta (1999). The results of the calculation are outlined in tables 10.1 and 10.2.

Table 10.1: Components yield of the demanded components obtained by the tree structure model.

component (p)	5	9	10	11	19	20	21	25	26	27
$i = 1$	0	237	-	474	-	1250	-	237	237	-
$i = 2$	400	-	120	24	200	0	398	180	213	-
$i = 3$	150	163	-	652	-	0	652	163	0	350

The solution, based on the instance of the model given in appendix 4, requires 273 units of product type 1, 200 units of product type 2, and 163 units of product type 3.

Table 10.2: Number of roots that has to be taken apart in the solution of table 6.

root number (j)	$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$
$i = 1$	237	0	237	237	237
$i = 2$	200	200	199	180	-
$i = 3$	163	75	163	163	163

It should be noticed that the range of the integer variables has been predefined in the code, because the maximum default value is 100, which is exceeded in the instance

discussed here. Surprisingly for this integer linear programming problem, the required CPU time is modest, i.e. far below 1 sec.
In case of larger models, with huge demand figures, complex product structures, and/or many types of product in a family, in which it can be expected that the CPU time increases, one can always make an attempt to approximate the integer linear programming problem by a linear one.

References and recommended literature

The theory that has been discussed in this course is extensively discussed and applied in a number of papers and in a book. Papers of the author are listed below.

Journal papers and Book Contributions on Industrial Ecology and Bulk Recycling

WOLTERS, W.T.M, LAMBERT, A.J.D. AND CLAUS, J., 1995, *Sequencing problems in designing energy efficient production systems*, Int. J. on Production Economics, **41**, 405-410.

LAMBERT, A.J.D, JANSEN, M.H., SCHUWER, R.V. AND SPLINTER, M.A.M., 1997, *Environmental information systems based on physical flows*, in: J. Goossenaerts, F. Kimura and H. Wortmann (eds.), Information Infrastructure Systems for Manufacturing, Chapman & Hall, London, p. 383-394.

STOOP, M.L.M. AND LAMBERT, A.J.D., 1998, *Processing of Discarded Refrigerators in the Netherlands*, Technovation, **18**(2) 101-110.

KLEINEIDAM, U., LAMBERT, A.J.D., BLANSJAAR, J., KOK, J.J., AND VAN HEIJNINGEN, R.J.J., 2000, *Optimising product recycling chains by control theory*. International Journal of Production Economics, **66**(2), pp. 185-195.

LAMBERT, A.J.D. AND STOOP, M.L.M., 2001, *Processing of discarded household refrigerators: lessons from the Dutch example*, Journal of Cleaner Production, **9**(3), p. 243-252.

LAMBERT, A.J.D., 2001, *Life cycle analysis including recycling*, in: J. Sarkis (Ed.) Greener Maniufacturing and Operations: From Design to Delivery and Back. Greenleaf Publishing, Sheffield UK, Chapter 2, p. 36-55.

LAMBERT, A.J.D. AND BOONS, F.A., 2002, *Eco-industrial parks*, Technovation, **22**(8), 471-484.

LAMBERT, A.J.D., BOELAARTS, H.M. AND SPLINTER, M.A.M., 2004, *Optimal recycling system desgn with an application to sophisticated packaging tools*, Environmental and Resource Economics, **28**(3) 273-299.

Congress papers on Industrial Ecology and Bulk Recycling

LAMBERT, A.J.D., 1994, *Production Chain Optimisation*, in: Abstracts of EURO XIII/OR 36, held at Glasgow, UK, july 19-22, 1994, p. 111-112.

LAMBERT, A.J.D, 1994, *Dismantling and Refurbishing of Cars in a Chain Perspective*, in Proceedings for the Dedicated Conference on Lean/Agile Manufacturing in the Automotive Industry, Aachen, D, oct/nov 1994, p. 741-748.

LAMBERT, A.J.D, JANSEN, M.H., SCHUWER, R.V. AND SPLINTER, M.A.M., 1996, *Environmental Information Systems, based on Physical Flows*, in: Goossenaerts J, Kimura, F., and Wortmann, H. (eds.), Pre-prints Proc. of the 2nd International Conference on Design of Information Infrastructure Systems for Manufacturing (DIISM), 16-18 september 1996, Eindhoven, 34-1..34-12.

LAMBERT, A.J.D. AND SPLINTER M.A.M., 1996, *Reuse of Sophisticated Product Carriers*, in: Flapper, S.D. and De Ron, A.J. (eds.), Proc. of 1st Working Seminar on Reuse, 11-13 november 1996, Eindhoven, pp. 213-222

LAMBERT, A.J.D., 1999 *Disassembly process modelling: problems and solutions concerning their adaptation to practice*, in: P.G. Maropoulos and J.A. McGeouch (eds.): Proc. of the 15th International Conference on Computer-Aided Production engineering (CAPE'99), Durham, England, 19-21 april 1999, p. 485-492.

LAMBERT, A.J.D., JANSEN, M.H. AND SPLINTER, M.A.M., 1997, *Environmental Information Sytems Based on Enterprise Resource Planning*, Integrated Manufacturing Systems: The International Journal of Manufacturing Technology Management, special issue: selected papers of 3rd International Symposium on Logistics, Padua, Italia, 1997. 2000, **11**(2), p. 105-111.

Journal papers and Book Contributions on Disassembly Sequencing

LAMBERT, A.J.D, 1997, *Optimal Disassembly of Complex Products*, International Journal of Production Research, 35 (1997), 2509-2523

LAMBERT, A.J.D., 1999, *Mathematical programming in disassembly/clustering sequence generation*, Computers and Industrial Engineering (CAIE) reuse special. **36** (4) 1999, p. 723-738.

LAMBERT, A.J.D. AND GUPTA, S.M., 2002, *Demand-driven disassembly optimisation for electronic products*, Journal of Electronics Manufacturing, **11**(2), 121-135.

LAMBERT, A.J.D., 2002, *Determining optimum disassembly sequences in electronic equipment*, Computers and Industrial Engineering, **43**(3), 553-575.

KANG, J.G., LEE, D.H., XIROUCHAKIS, P. AND LAMBERT, A.J.D., 2002, *Optimal disassembly sequencing with sequence-dependent operation times based on the directed graph of assembly states*, Journal of the Korean Institute of Industrial Engineers, **28**, 264-273.

LAMBERT, A.J.D., 2003, *Disassembly sequencing: a survey*. International Journal of Production Research, **41**(16), 3721-3759.

LAMBERT, A.J.D., 200x, *Optimizing disassembly processes with sequence dependent cost*. Paper submitted to Computers & Operations Research, Special Issue on Reverse Logistics. **Accepted**.

INDERFURTH, K., FLAPPER, S.D.P., LAMBERT, A.J.D., PAPPIS, C.P. AND VOUTSINAS, T.G., 2004, *Production planning for product recovery management*. In: Dekker, R., Fleischmann, M., Inderfurth, K., and Wassenhove, L.N. van (eds.), Reverse Logistics: Quantitative models for closed-loop supply chains. Berlin: Springer Verlag. Ch. 10, p. 249-274.

LAMBERT, A.J.D. AND GUPTA, S.M., 2005. *Disassembly modeling for assembly, maintenance, reuse, and recycling*. Boca Raton FL: CRC Press. Book, 419 p.

LAMBERT, A.J.D., 2006, *Generation of assembly graphs by systematic analysis of assembly structures*. EJOR special issue on Balancing Assembly and Transfer Lines. European Journal of Operational Research, **168**(3), 932-951.

LAMBERT, A.J.D., 2005, *Exact method for disassembly sequence optimization subjected to sequence dependent costs*. International Journal of Operations and Quantitative Management, **11**(2), 75-89.

LAMBERT, A.J.D., 2006, *Exact methods in optimum disassembly sequence search for problems subject to sequence dependent costs*. Omega special issue on Reverse Production Systems: Disassembly and other reverse manufacturing related practices. **34**, 538-549.

Congress papers on Disassembly Sequencing

LAMBERT, A.J.D., 1994, *Optimal Dissassembly of Complex Products*, in Proceedings of 23th Annual Meeting of Northeast Decision Sciences Institute, Portsmouth NH, USA, pp. 74-80.

LAMBERT, A.J.D., 1999, *Optimal disassembly sequence generation for combined material recycling and part reuse.* in: Proc. of IEEE international symposium on assembly and task planning (ISATP '99), july 21-24, Porto, Portugal, p. 146-151.

LAMBERT, A.J.D., 2000, *Optimum Disassembly Sequence Generation*, Conference 4193 on Environmentally Conscious Manufacturing, held at 6-8 November 2000, Boston MA. Proceedings of SPIE, Volume 4193, S.M. Gupta (*ed.*) Environmentally Conscious Manufacturing, p. 56-67.

LAMBERT, A.J.D., 2001, *Automatic determination of transition matrices in optimal disassembly sequence generation*, Proc. of ISATP'01, 4th IEEE International Symposium on Assembly and Task Planning, Fukuoka, Japan, May 28-30, p. 220-225.

LAMBERT, A.J.D., 2001, *Disassembly aimed at product remanufacturing*. ASME DETC2001 21st Computers and Information in Engineering Conference (CIE), Pittsburgh, september 9-12, 2001, DETC2001/CIE-21252 (Proc. on CD-ROM).

LAMBERT, A.J.D., 2002, *Generation of assembly graphs by systematic analysis of assembly structures*, 15th IFAC World Congress, Barcelona. Session: Assembly line Design and Balancing, 6p. CD-ROM.

LAMBERT, A.J.D., 2003, *Optimum disassembly sequence with sequence-dependent disassembly costs*. Proceedings of 5th IEEE International symposium on Assembly and Task Planning (ISATP'03), Besançon F, 151-156.

LAMBERT, A.J.D., 2004, *Exact methods in disassembly sequencing as a benchmark for heuristic algorithms*, in: Proceedings of SPIE Conference 5584 on Environmentally Conscious Manufacturing, October 2004, Philadelphia, pp. 1-11.

LAMBERT, A.J.D., 2005, *Generating disassembly sequences using exact and heuristic methods, applied to disassembly precedence graphs*. Proceedings of 6th IEEE International Symposium on Assembly and Task Planning, ISATP'05, 19-21 July 2005, Montréal, on CD-ROM.

LAMBERT, A.J.D. AND GUPTA, S.M., 2005, *Determining optimum and suboptimum disassembly sequences with an aplication to a cell phone*. Proceedings of 6th IEEE International Symposium on Assembly and Task Planning, ISATP'05, 19-21 July 2005, Montréal, on CD-ROM.

LAMBERT, A.J.D. AND GUPTA, S.M., 2005, *A Heuristic Solution for the Disassembly Line Balancing Problem Incorporating Sequence Dependent Costs*. Proceedings of SPIE International Conference on Environmentally Conscious Manufacturing, Boston, October 23-26

LAMBERT, A.J.D., AND GUPTA, S.M., 2005, *Heuristic Algorithm for Disassembly Line Balancing*. INFORMS Annual Meeting, New Orleans/San Fransisco, november 2005.

References

BALDWIN, D.F., ABELL, T.E., LUI, M.M., DE FAZIO, T.L. AND WHITNEY, D.E., 1991, *An integrated computer aid for generating and evaluating assembly sequences for mechanical products*. IEEE Transactions on Robotics and Automation **7**, 78-94.

BISSCHOP, J. AND ENTRIKEN, R., 1993, *AIMMS, the modeling system*. Haarlem: Paragon Decision Technology.

BOOTHROYD, G., POLI, C., AND MURCH L.E., 1982, *Automatic assembly. Manufacturing, Engineering and Materials Processing.* New York: Marcel Dekker Inc. Vol. **6**, Chapter 9.

BOURJAULT, A., 1984, *Contribution à une approche méthodologique de l'assemblage automatisé: elaboration automatique des séquencs opératoires* (contribution to a methodological approach of automatic assembly: automatic determination of operation sequences). Ph.-D. Thesis. Besançon, France: Université de Franche-Comté (in French).

BRUNDTLAND, G.H., 1987, *Our common future*. report of the U.N. World Commission on Environment and Development. Published by Oxford University Press, 1988.

DE FAZIO, T.L. AND WHITNEY, D.E., 1987, *Simplified generation of all mechanical assembly sequences*. IEEE Journal of Robotics and Automation RA-3, **6**, 640-658.

FROSCH, R.A. AND GALLOPOULOS, N.E., 1989, *Strategies for Manufacturing*. Scientific American, **261**(3), 144-152.

GUNGOR, A. AND GUPTA, S.M., 1997, *An evaluation methodology for disassembly processes*. Computers and Industrial Engineering **33**, 329-332.

GUNGOR, A. AND GUPTA, S.M., 1999, *Issues in environmentally conscious manufacturing and product recovery: A survey*. Computers and Industrial Engineering **36**, 811-853.

GUNGOR, A. AND GUPTA, S.M., 2001, *Disassembly sequence plan generation using a branch-and-bound algorithm*. International Journal of Production Research, **39**, 481-509.

HOMEM DE MELLO, L.S. AND SANDERSON, A.C., 1990, *And/or graph representation of assembly plans*. IEEE Transactions on Robotics and Automation, **6**, 188-199.

HOMEM DE MELLO, L.S. AND SANDERSON, A.C., 1991, *A correct and complete algorithm for the generation of mechanical assembly sequences*. IEEE Transactions on Robotics and Automation, **7**, 228-240.

JOHNSON, M.R. AND WANG, M.H., 1998, *Economical evaluation of disassembly operations for recycling, manufacturing and reuse*. International Journal of Production Research **36**, 3227-3252.

KANEHARA, T., SUZUKI, T., INABA, A. AND OKUMA, S., 1992, *On algebraic and graph structural properties of assembly Petri net*. Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, Yokohama, Japan. pp. 2286-2293.

KANG, J.G., LEE, D.H., XIROUCHAKIS, P. AND PERSSON, J.G., 2001, *Parallel disassembly sequencing with sequence-dependent operation times*. Annals of the CIRP, **50**, 343-346.

LAPERRIERE, L. AND ELMARAGHY, H.A., 1992, *Planning of products assembly and disasembly*. Annals of the CIRP, **41**(1), 5-9.

MOSEMANN, H. AND WAHL, F.M., 2001, *Automatic decomposition of planned assembly sequences into skill primitives*. IEEE Transactions on Robotics and Automation,**17**, 709-718.

SANTOCHI, M., DINI, G. AND FAILLI, F., 2001, *Computer aided disassembly planning: State of the art and perspectives*. Annals of the CIRP, **50**, 507-529.

SPENGLER, TH., PLOOG, M., AND SCHRÖTER, M., 2003, *Integrated planning of acquisition, disassembly and bulk recycling: a case study on electronic scrap recovery*. OR Spectrum, **25**(3), 413-442.

UCHIYAMA, N., ARAI, E. AND IGOSHI, M., 1994, *Generation of mechanical assembly sequences considering different evaluation viewpoints*. In: E. Usui (ed.): Advancement of Intelligent Production, pp. 701-706.

VEERAKAMOLMAL, P., AND GUPTA, S.M., 1999, *Analysis of design efficiency for the disassembly of modular electronic products*. Journal of Electronics Manufacturing, **9**(1), 79-95.

WOLTER, J.D., 1992, *A combinatorial analysis of enumerative data structures for assembly planning*. Journal of Design and Manufacturing, **2**(2), 93-104.

YEE, S.T. AND VENTURA, J.A., 1999, *A Petri net model to determine optimal assembly sequences with assembly operation constraints*. Journal of Manufacturing Systems, **18**, 203-213.

Further literature on disassembly planning and scheduling is mentioned in the book on Disassembly Modeling by Lambert and Gupta (2005), and in the survey paper by Lambert (2003). A further list of hundreds of papers up to present, is on the site of S.M. Gupta:
http://www1.coe.neu.edu/~smgupta/disassembly.htm

APPENDIX 1

Program for the iterative of Bourjault's ballpoint disassembly. AIMMS 2.20 code

```
! PROGRAM salesm4, incomplete disassembly, simple example
SETS:
        nodes := {0 .. 6},
!       0 is both the source and the sink
        nonvirtualnodes SUBSET nodes := {1 .. 6};
INDICES:
        j,k in nodes,
        l in nonvirtualnodes;
PARAMETERS:
        c(j,k)                        "cost of operation",
        y(l)                          "yield of component";
INTEGER VARIABLES:
        w(j,k)                        "partial flow variables";
VARIABLES:
        x(j)                          "flow variables",
        obj                           "objective function or profit";
CONSTRAINTS:
        flow_composition(k) ..
                x(k) = sum[j,w(j,k)],

!incomplete sequences not permitted
        partial_node_constraint (j) ..
                sum[k,w(k,j)]=sum[k,w(j,k)],
        all_nodes_visited_once(j) ..
                x(j)<=1,
        inhibit_1_cycles (j) ..
                w(j,j)=0,
        struc_1 ..
                w("0","1")+w("0","2")+w("0","3")+w("0","4")=0,
        struc_2 ..
                w("1","5")+w("3","5")+w("4","5")+w("2","6")=0,
        constr_1 ..
                w("2","5")+w("5","2")<=1,
        constr_2 ..
                w("4","2")+w("2","5")<=1,
        constr_3 ..
                w("2","4")+w("4","2")<=1,
        constr_4 ..
                w("3","6")+w("6","2")+w("2","5")<=2,
        constr_5 ..
                w("1","3")+w("3","6")+w("6","1")<=2,

        profit ..
                obj = sum[l,y(l)*x(l)] -
sum[k,sum[j,(c(j,k)*w(j,k))]];
MODEL:
        optimal_disassembly_sequence
                maximise: obj
                subject to: all
                method: mip;
SOLVE optimal_disassembly_sequence;
```

```
! DATA SECTION
PARAMETERS:
y := TABLE
          1   2   3   4   5   6
!        --  --  --  --  --  --
1        99  40  93   4  60  70,

c := TABLE
          1   2   3   4   5   6
!        --  --  --  --  --  --
0        40  84  25  69  73  80
1         0  46   9  63  42  86
2        74   0  16  13  36  51
3        74  29   0  26  73  16
4        72  21  80   0   1  42
5        16  42  60  26   0  60
6        26  73  42  36  73   0
;
```

APPENDIX 2

Program for the rigorous solution of Bourjault's ballpoint disassembly. AIMMS 2.20 code.

```
! PROGRAMMA salesm5, complete disassembly, unconstrained, rigorous
SETS:
        nodes := {0 .. 6},
!       0 is both the source and the sink
        nonvirtualnodes SUBSET nodes := {1 .. 6};
INDICES:
        j,k in nodes,
        l in nonvirtualnodes;
PARAMETERS:
        c(j,k)                                "cost of operation",
        y(l)                                  "yield of component",
        N                                     "number of nodes";
INTEGER VARIABLES:
        p(j,k)                                "partial counter";
BINARY VARIABLES:
        w(j,k)                                "partial flow vari-
ables";
VARIABLES:
        a(l)                                  "aggregate counter",
        x(j)                                  "flow variables",
        obj                                   "objective function
or profit";
        N := 6;
CONSTRAINTS:
        flow_aggregation(k) ..
                x(k) = sum[j,w(j,k)],
        partial_node_constraint (j) ..        !incomplete sequences
not permitted
                sum[k,w(k,j)]=sum[k,w(j,k)],
        all_nodes_visited_not_more_than_once(j) ..
                x(j)<=1,
        inhibit_1_cycles (l) ..
                w(l,l)=0,
        initialization_of_partial_counters ..
                sum[j,p("0",j)]=N,
        counter_aggregation (l) ..
                a(l)=sum[j,p(j,l)],
        counter_decrement (l) ..
                sum[j,p(l,j)]=a(l)-x(l),
        upper_bound_of_counter (j,k) ..
                p(j,k)<=(N+1)*w(j,k),
! PRECEDENCE RELATIONSHIPS
        struc_1 ..
                a("5")>=a("3"),
        struc_2 ..
                a("6")>=a("2"),
        struc_3 ..
                a("5")>=a("4"),
        struc_4 ..
                a("5")>=a("1"),
        profit ..
                obj = sum[l,y(l)*x(l)] -
sum[k,sum[j,(c(j,k)*w(j,k))]];
```

```
MODEL:
        optimal_disassembly_sequence
                maximise: obj
                subject to: all
                method: mip;
SOLVE optimal_disassembly_sequence;
! DATA SECTION
PARAMETERS:
! YIELDS OF COMPONENTS
y := TABLE
        1   2   3   4   5   6
!      --  --  --  --  --  --
1      99  40  93   4  60  70,

! COSTS OF OPERATIONS
c := TABLE
        1   2   3   4   5   6
!      --  --  --  --  --  --
0      40  84  25  69  73  80
1       0  46   9  63  42  86
2      74   0  16  13  36  51
3      74  29   0  26  73  16
4      72  21  80   0   1  42
5      16  42  60  26   0  60
6      26  73  42  36  73   0
;
```

APPENDIX 3

Program for the Heuristic in VisualBasic 6.0

```
'PROGRAM 'PC' ACCORDING TO GUNGOR-GUPTA PC EXAMPLE
'CLEARED VERSION
'########################################################################
'DECLARATION OF CONSTANTS
Const FINSIZE = 100          'Length of finlist
Const LAMBDA = 200           'Index of greediness, length of intlist and provlist
Const MODELSIZE = 14         'Maximum number of nonvirtual operations (maximum
N)
'number of vectors and express in precedence relations
Const INITCOUNT = 4          'Number of predefined nonvirtual operations; zero if no
predefined ones

'TYPE DEFINITIONS
Private Type LongRecord
'Contains a string of task numbers, each of 4 positions
'Contains a string of flow binaries, each of 1 position
'Contains an objective value
'Length of Op and Flow is MODELSIZE
   Op(0 To MODELSIZE) As Integer
   Flow(0 To MODELSIZE) As Boolean
   Obj As Integer
End Type

Private Type Record
   Op(0 To MODELSIZE) As Integer
   Obj As Integer
End Type

Private Type List
   Rec(1 To LAMBDA) As Record
End Type

Private Type Fin
   Rec(1 To FINSIZE) As Record
End Type

'MAIN MODULE
Private Sub Start_Click()
Dim s1, s2, dummy, counter As Integer
Dim Provlist As List
Dim Intlist As List
Dim LRec1 As LongRecord
Dim Rec1 As Record
'Initialization of final and provisional list
Dim Finlist As Fin
```

```
    For s1 = 1 To FINSIZE
        Finlist.Rec(s1).Obj = -999
    Next s1
    For s1 = 1 To LAMBDA
        Intlist.Rec(s1).Obj = -999
    Next s1
    For s1 = 1 To LAMBDA
        Provlist.Rec(s1).Obj = -999
    Next s1
Finlist.Rec(1) = Init
Intlist.Rec(1) = Init
'Print Makestring(Init)
LRec1 = ConvFlow(Init)
'Print Makelongstring(LRec1)
counter = INITCOUNT + 1
dummy = Expand(counter, Intlist, Provlist)
dummy = Putintlist(Provlist, Intlist)
dummy = Putfinlist(Intlist, Finlist)
For counter = INITCOUNT + 2 To MODELSIZE
    For s1 = 1 To LAMBDA
        Provlist.Rec(s1).Obj = -999
    Next s1
    dummy = Expand(counter, Intlist, Provlist)
    dummy = Putintlist(Provlist, Intlist)
    dummy = Putfinlist(Intlist, Finlist)
Next counter
For s1 = 1 To FINSIZE
Print Makestring(Finlist.Rec(s1))
Next s1
End Sub

'SUPPORTS ENFORCED INITIALIZATION
Private Function Init() As Record
    Dim i1, i2 As Integer
    Dim Rec1 As Record
    Rec1.Obj = 0
    For i1 = 0 To MODELSIZE
        Rec1.Op(i1) = 0
    Next i1
'   Insert an amount of INITCOUNT enforced operations'
    Rec1.Op(1) = 1
    Rec1.Op(2) = 3
    Rec1.Op(3) = 7
    Rec1.Op(4) = 2
For i1 = 0 To INITCOUNT + 1
        Rec1.Obj = Rec1.Obj + Readmatrix(Rec1.Op(i1), Rec1.Op(i1 + 1))
    Next i1
    Init = Rec1
End Function
```

```
'READS THE PROFIT MATRIX
'NUMBER OF ENABLED VECTORS EQUALS MODELSIZE
Private Function Readmatrix(j As Integer, k As Integer) As Integer
    Dim Vector(0 To MODELSIZE) As String
    Vector(0) = "   0  59 -44  68 -65 -13 -10 -23  63  20   0   1   3  62  -2"
    Vector(1) = "   0   0  -6  84 -59  18 -16   7  34  29  19  35 -20  -2  55"
    Vector(2) = "   0  25   0  77  -9  24  19 -40  38  -6  43  19 -21  56   8"
    Vector(3) = "   0  25  11   0 -22 -13  54  34  -4   0 -12  21  -3  38 -19"
    Vector(4) = "   0  27  19  13   0  59  28 -14  63  79 -16  46  50  82 -31"
    Vector(5) = "   0  83  -2  33 -22   0  10  20  10   7  52  18   6  -4 -25"
    Vector(6) = "   0  73 -33  51 -32 -13   0  -4  23  44  47  57  23  90 -34"
    Vector(7) = "   0  30  24  10 -37   8  -5   0  47  38 -29  34  43  30 -16"
    Vector(8) = "   0  33  22   1 -41 -15   5  11   0   7  10  -9  57  79  47"
    Vector(9) = "   0  73 -20  51 -12   9 -16 -28  78   0  45  21  66  35 -30"
    Vector(10) = "   0   2   1  30 -40  50  -4 -27  68  64   0  18  75  45  53"
    Vector(11) = "   0  63 -46  41 -64  -6  29 -17  49  28 -33   0   0  41   0"
    Vector(12) = "   0  24 -56  -4 -75  34   0  -1  46  61  53  86   0   0  -2"
    Vector(13) = "   0  12   1  36  -9 -39  65 -13  63  20  13   0  60   0  55"
    Vector(14) = "   0  44 -42  -1 -84 -37  -9  36 -13  60  55  71  15  28   0"

    Readmatrix = Val(Mid(Vector(j), (4 * k) + 1, 4))
End Function

'CREATES AN ADDITIONAL VECTOR ACCOUNTING FOR THE
PRECEDENCE RELATIONS
Private Function ConvFlow(CRec As Record) As LongRecord
'Expands the Flow part of the Longrecord
'By making use of the set of precedence relations
Dim c1 As Integer
Dim LRec1 As LongRecord
Dim LRec2 As LongRecord
    For c1 = 0 To MODELSIZE
        LRec1.Op(c1) = CRec.Op(c1)
    Next c1
    LRec1.Obj = CRec.Obj
    LRec1.Flow(0) = True
    For c1 = 1 To MODELSIZE
        LRec1.Flow(c1) = False
    Next c1
    For c1 = 1 To MODELSIZE
        If LRec1.Op(c1) > 0 Then
            LRec1.Flow(LRec1.Op(c1)) = True
        End If
    Next c1
    For c1 = 0 To MODELSIZE
        LRec2.Flow(c1) = LRec1.Flow(c1)
    Next c1
'Print Makelongstring(LRec1)
'Print Makelongstring(LRec2)
```

```
'List of Precedence Relations
   If LRec2.Flow(0) = True Then
      LRec1.Flow(1) = True
   End If
   If LRec2.Flow(1) = True Then
      LRec1.Flow(2) = True
      LRec1.Flow(3) = True
      LRec1.Flow(4) = True
      LRec1.Flow(5) = True
      LRec1.Flow(6) = True
      LRec1.Flow(7) = True
      LRec1.Flow(8) = True
      LRec1.Flow(9) = True
   End If
   If LRec2.Flow(2) = True And LRec2.Flow(3) = True Then
      LRec1.Flow(10) = True
   End If
   If LRec2.Flow(3) = True And LRec2.Flow(4) = True Then
      LRec1.Flow(11) = True
   End If
   If LRec2.Flow(5) = True And LRec2.Flow(6) = True Then
      LRec1.Flow(12) = True
   End If
   If LRec2.Flow(6) = True And LRec2.Flow(7) = True Then
      LRec1.Flow(13) = True
   End If
   If LRec2.Flow(10) = True And LRec2.Flow(11) = True Then
      LRec1.Flow(14) = True
   End If
  'Print Makelongstring(LRec2)
   For c1 = 0 To MODELSIZE
      If LRec2.Flow(c1) = True Then
         LRec1.Flow(c1) = False
      End If
   Next c1
'Print Makelongstring(LRec1)
'Print '-'
ConvFlow = LRec1
End Function
Private Function Expand(Count1 As Integer, List1 As List, List2 As List) As Integer
'Reads from Intlist, converts, expands, and puts in Provlist
Dim e1, e2, e3, e4, Pointer As Integer
Dim Rec1 As Record
Dim LRec1 As LongRecord
Pointer = 1
For e1 = 1 To LAMBDA
   Rec1 = List1.Rec(e1)
   If Rec1.Obj = -999 Then
Exit For
   End If
```

```
    LRec1 = ConvFlow(Rec1)
    For e2 = 0 To MODELSIZE
        If LRec1.Flow(e2) = True Then
            Rec1.Op(Count1) = e2
            Rec1.Obj = LRec1.Obj + Readmatrix(Rec1.Op(Count1 - 1),
Rec1.Op(Count1))
            For e3 = 1 To LAMBDA
                If Rec1.Obj > List2.Rec(e3).Obj Then
                    Pointer = e3
                        For e4 = 1 To (LAMBDA - Pointer)
                            List2.Rec(LAMBDA + 1 - e4) = List2.Rec(LAMBDA - e4)
                        Next e4
                        List2.Rec(Pointer) = Rec1
                    Exit For
                End If
            Next e3
        End If
    Next e2
Next e1
End Function
Private Function Putintlist(List1 As List, List2 As List) As Integer
Dim p1 As Integer
Dim Rec1 As Record
    For p1 = 1 To LAMBDA
        Rec1 = List1.Rec(p1)
        If Rec1.Obj = -999 Then Exit For
        List2.Rec(p1) = List1.Rec(p1)
    Next p1
End Function
Private Function Putfinlist(List1 As List, FList1 As Fin) As Integer
Dim f1, f2, f3, counter, Obj1 As Integer
Dim Rec1 As Record
counter = 1
For f1 = 1 To FINSIZE
    For f2 = counter To LAMBDA
        Obj1 = FList1.Rec(f1).Obj
        If List1.Rec(f2).Obj > Obj1 Then
            For f3 = 1 To FINSIZE - f1
                FList1.Rec(FINSIZE + 1 - f3) = FList1.Rec(FINSIZE - f3)
            Next f3
            FList1.Rec(f1) = List1.Rec(f2)
            counter = counter + 1
        End If
    Next f2
Next f1
End Function

Private Function Makestring(Rec1 As Record) As String
'Converts a record to a string, for representation
Dim m1 As Integer
Dim Str1 As String
   For m1 = 0 To MODELSIZE
      Str1 = Str1 + Str(Rec1.Op(m1))
   Next m1
   Str1 = Str1 & "#"
   Str1 = Str1 + Str(Rec1.Obj)
   Makestring = Str1
End Function
Private Function Makelongstring(LRec1 As LongRecord) As String
'Converts a record to a string, for representation
Dim m1 As Integer
Dim Str1, Str2 As String
   For m1 = 0 To MODELSIZE
      Str1 = Str1 + Str(LRec1.Op(m1))
   Next m1
   Str1 = Str1 & "#"
   For m1 = 0 To MODELSIZE
      If LRec1.Flow(m1) = True Then
         Str1 = Str1 + "1"
      Else
         Str1 = Str1 + "0"
      End If
   Next m1
   Str1 = Str1 & "#"
   Str1 = Str1 + Str(LRec1.Obj)
   Makelongstring = Str1
End Function
```

APPENDIX 4

Program for the disassembly-to order problem in section 10. AIMMS 2.20 code

! PROGRAM: 'TREE'. This is a DISASSEMBLY-TO ORDER model(3 product types with extended costs, tree structure model)
! this deals with the multiciplity of parts

```
ORDERED SETS:
        product_type := {1 .. 3},
        operations := {1 .. 5},
        components := {1 .. 27};
INDICES:
        i in product_type,
        p in components,
        j,k in actions;
PARAMETERS:
        C (i,j)                         "cost of action",
        ct(i)                           "supply costs",
        cw(p)                           "waste costs",
        D (p)                           "demand for component",
        S(i,k,j)                        "structure matrix",
        Y(i,j,p)                        "yield matrix";
INTEGER VARIABLES:
        x(i,j) -> {0 .. 250}            "flow variable";
POSITIVE VARIABLES:
        y(p)                            "yield",
        yield(i,p)                      "partial yield",
        cdiss                           "disassembly costs",
        csupp                           "supply costs",
        cwaste                          "disposal costs";
VARIABLES:
        obj                             "objective";
CONSTRAINTS:
        node_constraint(i,j) ..
                x(i,j) <= sum[k,S(i,k,j)*x(i,k)],
        partial_yield(i,p) ..
                yield(i,p) <= sum[j,Y(i,j,p)*x(i,j)],
        demand_constraint(p) ..
                sum[i,yield(i,p)] = D(p),
        disassembly_costs ..
                cdiss = 0.2*sum[i,sum[j,C(i,j)*x(i,j)]],
        supply_costs ..
                csupp = sum[i,ct(i)*x(i,"1")],
        disposal_costs ..
                 cwaste = 0.25*sum[p,cw(p)*(sum[i,x(i,"1")*sum[j,Y(i,j,p)]]-D(p))],
        objective ..
                obj = cdiss + csupp + cwaste;
MODEL:
        optimal_disassembly_sequence
                minimise: obj
                subject to: all
                method: mip;
SOLVE optimal_disassembly_sequence;
```

```
! DATA SECTION
PARAMETERS:
S:= TABLE
            1  2  3  4  5
!          -- -- -- -- --
(1,1)       1  1  1     1
(1,3)                1
(2,1)       1  1     1
(2,2)             1
(3,1)       1  1  1     1
(3,3)                1   ,

Y:= TABLE
            1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
!          -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
(1,1)       1
(1,2)                1  1  1
(1,3)                                        1        1  1
(1,4)                               1     2                          6     1
(1,5)                                                                         1     1  1
(2,1)          1
(2,2)                1  2     1                    1  1     1     1
(2,3)                                  2  4                          4  2  1
(2,4)                                                                            1  1  2
(3,1)             1
(3,2)                1  2        1
(3,3)                                           1     1        1
(3,4)                               1     4                          4  4  1
(3,5)                                                                            1  1  2  7,

C:= TABLE
           1    2    3    4    5
!         --   --   --   --   --
1        3.7  2.5  1.9  2.1  1.7
2        4.5  3.1  2.3  2.3
3        3.7  2.6  2.0  2.1  1.7,

D:= {(5):550,(9):400,(10):120,(11):1150,(19):200,(20):1250,(21):1050,(25):580,(26):450,(27):350},

ct:= {(1):140,(2):120,(3):135},

cw:={(1):5,(2):9,(3):6,(4):2,(5):10,(6):3,(7):3,(8):4,(9):5,(10):5,(11):2,(12):1,(13):1,(14):1,
(15):1,(16):3,(17):3,(18):3,(19):2,(20):1,(21):1,(22):1,(23):6,(24):6,(25):5,(26):7,(27):3};
```