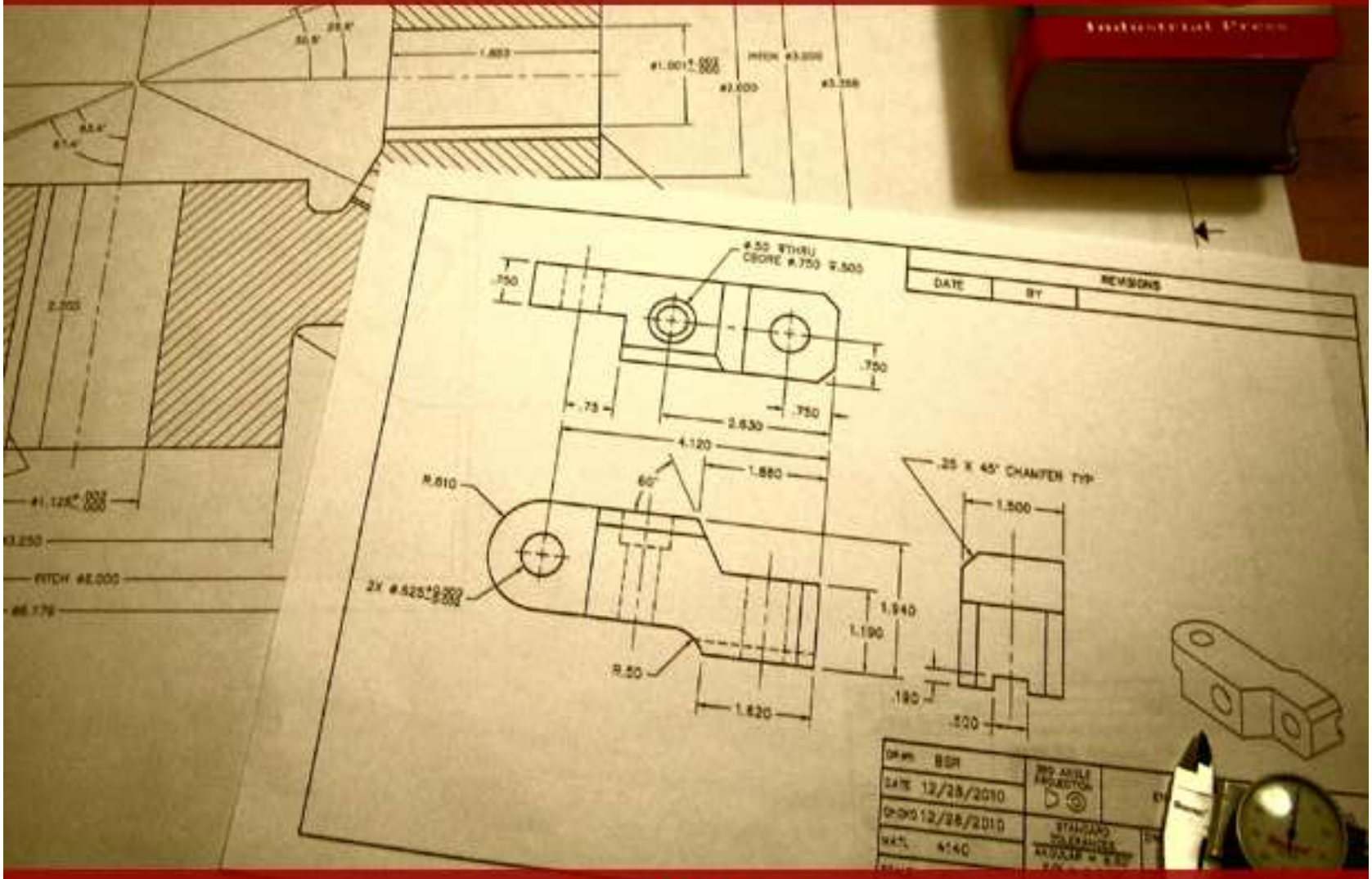
Engineering Statistics & Operations Research



Clair Moeller

Skylar Forrester

First Edition, 2012

ISBN 978-81-323-0879-9

Published by:
Academic Studio
4735/22 Prakashdeep Bldg,
Ansari Road, Darya Ganj,
Delhi - 110002
Email: info@wtbooks.com

Table of Contents

Chapter- 1

Design of Experiments

In general usage, **design of experiments (DOE)** or **experimental design** is the design of any information-gathering exercises where variation is present, whether under the full control of the experimenter or not. However, in statistics, these terms are usually used for controlled experiments. Other types of study, and their design, are discussed in opinion polls and statistical surveys (which are types of observational study), natural experiments and quasi-experiments (for example, quasi-experimental design).

In the design of experiments, the experimenter is often interested in the effect of some process or intervention (the "treatment") on some objects (the "experimental units"), which may be people, parts of people, groups of people, plants, animals, materials, etc. Design of experiments is thus a discipline that has very broad application across all the natural and social sciences.

History of development

Controlled experimentation on scurvy

In 1747, while serving as surgeon on HM Bark *Salisbury*, James Lind carried out a controlled experiment to develop a cure for scurvy.

Lind selected 12 men from the ship, all suffering from scurvy. Lind limited his subjects to men who "were as similar as I could have them", that is he provided strict entry requirements to reduce extraneous variation. He divided them into six pairs, giving each pair different supplements to their basic diet for two weeks. The treatments were all remedies that had been proposed:

- A quart of cider every day
- Twenty five gutts (drops) of *elixir vitriol* (sulphuric acid) three times a day upon an empty stomach,
- One half-pint of seawater every day
- A mixture of garlic, mustard, and horseradish in a lump the size of a nutmeg
- Two spoonfuls of vinegar three times a day
- Two oranges and one lemon every day.

The men who had been given citrus fruits recovered dramatically within a week. One of them returned to duty after 6 days and the other cared for the rest. The others experienced some improvement, but nothing was comparable to the citrus fruits, which were proved to be substantially superior to the other treatments.

Statistical experiments, following Charles S. Peirce

A theory of statistical inference was developed by Charles S. Peirce in "Illustrations of the Logic of Science" (1877–1878) and "A Theory of Probable Inference" (1883), two publications that emphasized the importance of randomization-based inference in statistics.

Randomized experiments

Charles S. Peirce randomly assigned volunteers to a blinded, repeated-measures design to evaluate their ability to discriminate weights. Peirce's experiment inspired other researchers in psychology and education, which developed a research tradition of randomized experiments in laboratories and specialized textbooks in the 1800s.

Optimal designs for regression models

Charles S. Peirce also contributed the first English-language publication on an optimal design for regression-models in 1876. A pioneering optimal design for polynomial regression was suggested by Gergonne in 1815. In 1918 Kirstine Smith published optimal designs for polynomials of degree six (and less).

Sequences of experiments

The use of a sequence of experiments, where the design of each may depend on the results of previous experiments, including the possible decision to stop experimenting, is within the scope of Sequential analysis, a field that was pioneered by Abraham Wald in the context of sequential tests of statistical hypotheses. Herman Chernoff wrote an overview of optimal sequential designs, while adaptive designs have been surveyed by S. Zacks. One specific type of sequential design is the "two-armed bandit", generalized to the multi-armed bandit, on which early work was done by Herbert Robbins in 1952.

Principles of experimental design, following Ronald A. Fisher

A methodology for designing experiments was proposed by Ronald A. Fisher, in his innovative book *The Design of Experiments* (1935). As an example, he described how to test the hypothesis that a certain lady could distinguish by flavour alone whether the milk or the tea was first placed in the cup. While this sounds like a frivolous application, it allowed him to illustrate the most important ideas of experimental design:

Comparison

In many fields of study it is hard to reproduce measured results exactly. Comparisons between treatments are much more reproducible and are usually preferable. Often one compares against a standard, scientific control, or traditional treatment that acts as baseline.

Randomization

There is an extensive body of mathematical theory that explores the consequences of making the allocation of units to treatments by means of some random mechanism such as tables of random numbers, or the use of randomization devices such as playing cards or dice. Provided the sample size is adequate, the risks associated with random allocation (such as failing to obtain a representative sample in a survey, or having a serious imbalance in a key characteristic between a treatment group and a control group) are calculable and hence can be managed down to an acceptable level. Random does *not* mean haphazard, and great care must be taken that appropriate random methods are used.

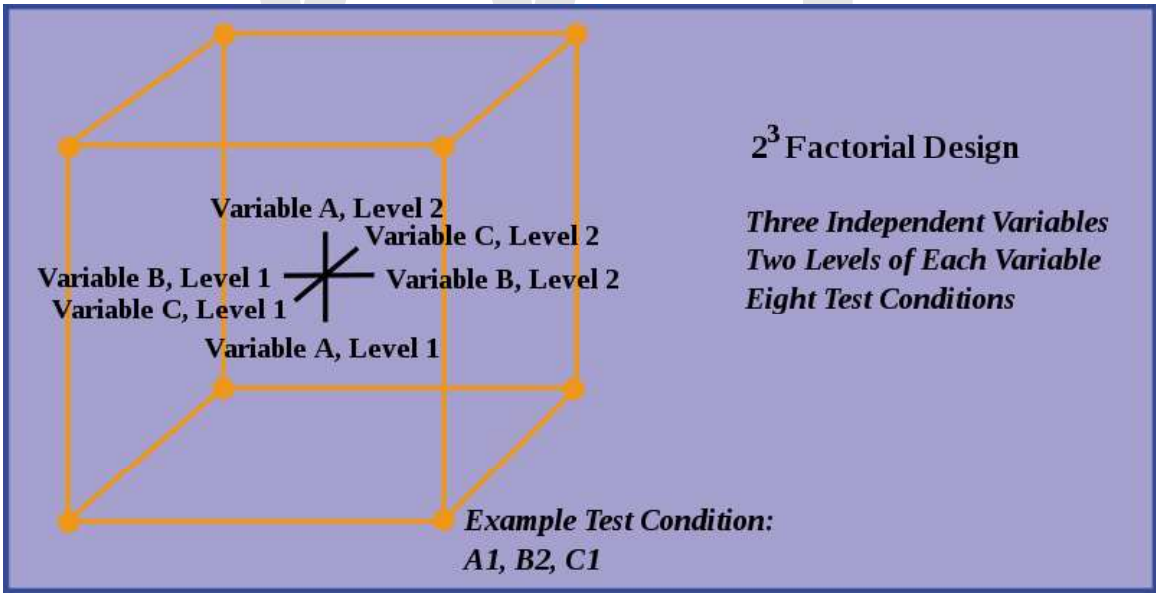
Replication

Measurements are usually subject to variation and uncertainty. Measurements are repeated and full experiments are replicated to help identify the sources of variation and to better estimate the true effects of treatments.

Blocking

Blocking is the arrangement of experimental units into groups (blocks) consisting of units that are similar to one another. Blocking reduces known but irrelevant sources of variation between units and thus allows greater precision in the estimation of the source of variation under study.

Orthogonality



Example of orthogonal factorial design

Orthogonality concerns the forms of comparison (contrasts) that can be legitimately and efficiently carried out. Contrasts can be represented by vectors

and sets of orthogonal contrasts are uncorrelated and independently distributed if the data are normal. Because of this independence, each orthogonal treatment provides different information to the others. If there are T treatments and $T - 1$ orthogonal contrasts, all the information that can be captured from the experiment is obtainable from the set of contrasts.

Factorial experiments

Use of factorial experiments instead of the one-factor-at-a-time method. These are efficient at evaluating the effects and possible interactions of several factors (independent variables).

Analysis of the design of experiments was built on the foundation of the analysis of variance, a collection of models in which the observed variance is partitioned into components due to different factors which are estimated and/or tested.

Example

This example is attributed to Harold Hotelling. It conveys some of the flavor of those aspects of the subject that involve combinatorial designs.

The weights of eight objects are to be measured using a pan balance and set of standard weights. Each weighing measures the weight difference between objects placed in the left pan vs. any objects placed in the right pan by adding calibrated weights to the lighter pan until the balance is in equilibrium. Each measurement has a random error. The average error is zero; the standard deviations of the probability distribution of the errors is the same number σ on different weighings; and errors on different weighings are independent. Denote the true weights by

$$\theta_1, \ldots, \theta_8.$$

We consider two different experiments:

1. Weigh each object in one pan, with the other pan empty. Let X_i be the measured weight of the ith object, for $i = 1, ..., 8$.
2. Do the eight weighings according to the following schedule and let Y_i be the measured difference for $i = 1, ..., 8$:

	left pan	right pan
1st weighing:	1 2 3 4 5 6 7 8	(empty)
2nd:	1 2 3 8	4 5 6 7
3rd:	1 4 5 8	2 3 6 7
4th:	1 6 7 8	2 3 4 5
5th:	2 4 6 8	1 3 5 7
6th:	2 5 7 8	1 3 4 6
7th:	3 4 7 8	1 2 5 6
8th:	3 5 6 8	1 2 4 7

Then the estimated value of the weight θ_1 is

$$\widehat{\theta}_1 = \frac{Y_1 + Y_2 + Y_3 + Y_4 - Y_5 - Y_6 - Y_7 - Y_8}{8}.$$

Similar estimates can be found for the weights of the other items. For example

$$\widehat{\theta}_2 = \frac{Y_1 + Y_2 - Y_3 - Y_4 + Y_5 + Y_6 - Y_7 - Y_8}{8}.$$

The question of design of experiments is: which experiment is better?

The variance of the estimate X_1 of θ_1 is σ^2 if we use the first experiment. But if we use the second experiment, the variance of the estimate given above is $\sigma^2/8$. Thus the second experiment gives us 8 times as much precision for the estimate of a single item, and estimates all items simultaneously, with the same precision. What is achieved with 8 weighings in the second experiment would require 64 weighings if items are weighed separately. However, note that the estimates for the items obtained in the second experiment have errors which are correlated with each other.

Many problems of the design of experiments involve combinatorial designs, as in this example.

Statistical control

It is best for a process to be in reasonable statistical control prior to conducting designed experiments. When this is not possible, proper blocking, replication, and randomization allow for the careful conduct of designed experiments.

Experimental designs after Fisher

Some efficient designs for estimating several main effects simultaneously were found by Raj Chandra Bose and K. Kishen in 1940 at the Indian Statistical Institute, but remained little known until the Plackett-Burman designs were published in *Biometrika* in 1946. About the same time, C. R. Rao introduced the concepts of orthogonal arrays as experimental designs. This was a concept which played a central role in the development of Taguchi methods by Genichi Taguchi, which took place during his visit to Indian Statistical Institute in early 1950s. His methods were successfully applied and adopted by Japanese and Indian industries and subsequently were also embraced by US industry albeit with some reservations.

In 1950, Gertrude Mary Cox and William Gemmell Cochran published the book *Experimental Designs* which became the major reference work on the design of experiments for statisticians for years afterwards.

Developments of the theory of linear models have encompassed and surpassed the cases that concerned early writers. Today, the theory rests on advanced topics in linear algebra, algebra and combinatorics.

As with other branches of statistics, experimental design is pursued using both frequentist and Bayesian approaches: In evaluating statistical procedures like experimental designs, frequentist statistics studies the sampling distribution while Bayesian statistics updates a probability distribution on the parameter space.

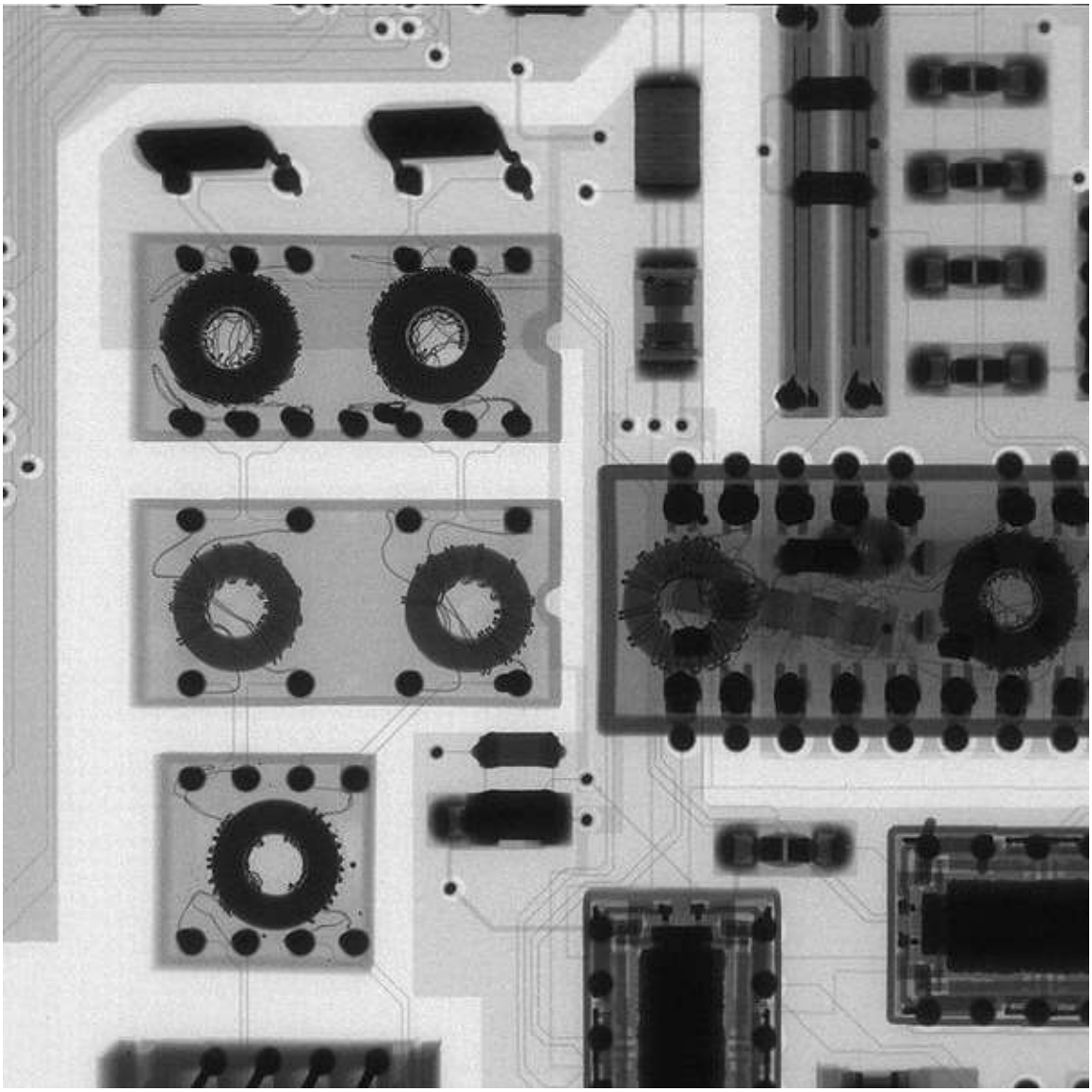
Some important contributors to the field of experimental designs are C. S. Peirce, R. A. Fisher, F. Yates, C. R. Rao, R. C. Bose, J. N. Srivastava, Shrikhande S. S., D. Raghavarao, W. G. Cochran, O. Kempthorne, W. T. Federer, A. S. Hedayat, J. A. Nelder, R. A. Bailey, J. Kiefer, W. J. Studden, F. Pukelsheim, D. R. Cox, H. P. Wynn, A. C. Atkinson, G. E. P. Box and G. Taguchi. The textbooks of D. Montgomery and R. Myers have reached generations of students and practitioners.

Chapter- 2

Quality Control and Process Control

Quality control

Maintenance check of electronic equipment on a U.S. Navy aircraft.



X-ray zoom series of a network adapter card.

Quality control is a process by which entities review the quality of all factors involved in production. This approach places an emphasis on three aspects:

1. Elements such as controls, job management, defined and well managed processes, performance and integrity criteria, and identification of records
2. Competence, such as knowledge, skills, experience, and qualifications
3. Soft elements, such as personnel integrity, confidence, organizational culture, motivation, team spirit, and quality relationships.

The quality of the outputs is at risk if any of these three aspects is deficient in any way.

Quality control emphasizes testing of products to uncover defects, and reporting to management who make the decision to allow or deny the release, whereas quality assurance attempts to improve and stabilize production, and associated processes, to avoid, or at least minimize, issues that led to the defects in the first place. For contract work, particularly work awarded by government agencies, quality control issues are among the top reasons for not renewing a contract.

Total quality control

"Total quality control" is a measure used in cases where sales decrease despite implementation of statistical quality control techniques or quality improvements. If the original specification does not reflect the correct quality requirements, quality cannot be inspected or manufactured into the product. For instance, the parameters for a pressure vessel should include not only the material and dimensions, but also operating, environmental, safety, reliability and maintainability requirements.

Quality control in project management

In project management, quality control requires the project manager and the project team to inspect the accomplished work to ensure it's alignment with the project scope. In practice, projects typically have a dedicated quality control team which focuses on this area.

Process control

Process control is a statistics and engineering discipline that deals with architectures, mechanisms and algorithms for maintaining the output of a specific process within a desired range.

For example, heating up the temperature in a room is a process that has the specific, desired outcome to reach and maintain a defined temperature (e.g. 20°C), kept constant over time. Here, the temperature is the **controlled variable**. At the same time, it is the **input variable** since it is measured by a thermometer and used to decide whether to heat or not to heat. The desired temperature (20°C) is the **setpoint**. The state of the heater (e.g. the setting of the valve allowing hot water to flow through it) is called the **manipulated variable** since it is subject to control actions.

A commonly used control device called a programmable logic controller, or a PLC, is used to read a set of digital and analog inputs, apply a set of logic statements, and generate a set of analog and digital outputs. Using the example in the previous paragraph, the room temperature would be an input to the PLC. The logical statements would compare the setpoint to the input temperature and determine whether more or less heating was necessary to keep the temperature constant. A PLC output would then either open or close the hot water valve, an incremental amount, depending on whether more or less hot

water was needed. Larger more complex systems can be controlled by a Distributed Control System (DCS) or SCADA system.

In practice, process control systems can be characterized as one or more of the following forms:

- Discrete – Found in many manufacturing, motion and packaging applications. Robotic assembly, such as that found in automotive production, can be characterized as discrete process control. Most discrete manufacturing involves the production of discrete pieces of product, such as metal stamping.

- Batch – Some applications require that specific quantities of raw materials be combined in specific ways for particular durations to produce an intermediate or end result. One example is the production of adhesives and glues, which normally require the mixing of raw materials in a heated vessel for a period of time to form a quantity of end product. Other important examples are the production of food, beverages and medicine. Batch processes are generally used to produce a relatively low to intermediate quantity of product per year (a few pounds to millions of pounds).

- Continuous – Often, a physical system is represented through variables that are smooth and uninterrupted in time. The control of the water temperature in a heating jacket, for example, is an example of continuous process control. Some important continuous processes are the production of fuels, chemicals and plastics. Continuous processes in manufacturing are used to produce very large quantities of product per year (millions to billions of pounds).

Applications having elements of discrete, batch and continuous process control are often called *hybrid* applications.

Statistical Process Control

Statistical Process Control (SPC) is an effective method of monitoring a process through the use of control charts. Much of its power lies in the ability to monitor both process center and its variation about that center. By collecting data from samples at various points within the process, variations in the process that may affect the quality of the end product or service can be detected and corrected, thus reducing waste as well as the likelihood that problems will be passed on to the customer. It has an emphasis on early detection and prevention of problems.

Multivariable Process Control is a type of Statistical Process Control where a matrix of variables (MV and CV) are created and their variations captured by doing a step test. The Dynamics captured in the model curves are used to control the plant

Examples

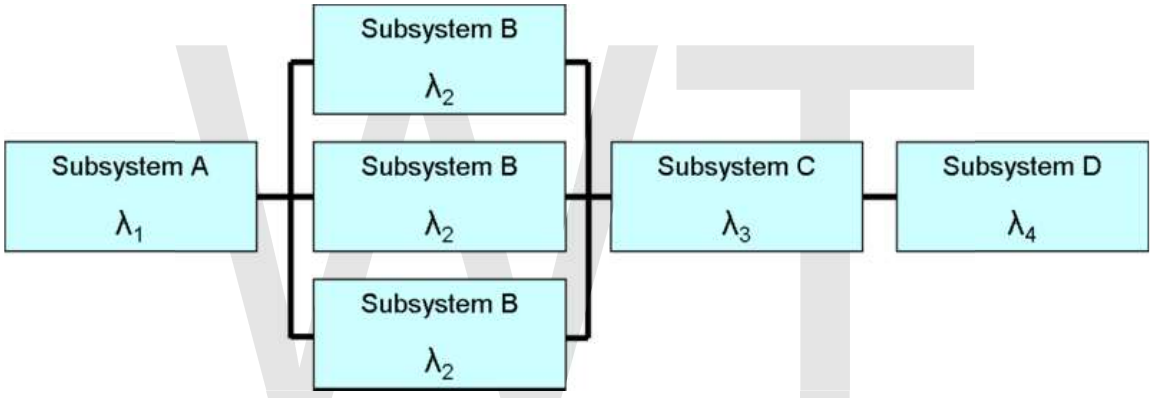
A thermostat is a simple example for a closed control loop: It constantly measures the current temperature and controls the heater's valve setting to increase or decrease the room temperature according to the user-defined setting. A simple method switches the heater either completely on, or completely off, and an overshoot and undershoot of the controlled temperature must be expected. A more expensive method varies the amount of heat provided by the heater depending on the difference between the required temperature (the "setpoint") and the actual temperature. This minimizes over/undershoot.

Chapter- 3

Reliability Engineering

Reliability engineering is an engineering field, that deals with the study of reliability: the ability of a system or component to perform its required functions under stated conditions for a specified period of time. It is often reported as a probability.

Overview



A Reliability Block Diagram

Reliability may be defined in several ways:

- The idea that something is fit for a purpose with respect to time;
- The capacity of a device or system to perform as designed;
- The resistance to failure of a device or system;
- The ability of a device or system to perform a required function under stated conditions for a specified period of time;
- The probability that a functional unit will perform its required function for a specified interval under stated conditions.
- The ability of something to "fail well" (fail without catastrophic consequences)

Reliability engineers rely heavily on statistics, probability theory, and reliability theory. Many engineering techniques are used in reliability engineering, such as reliability prediction, Weibull analysis, thermal management, reliability testing and accelerated life testing. Because of the large number of reliability techniques, their expense, and the varying degrees of reliability required for different situations, most projects develop a

reliability program plan to specify the reliability tasks that will be performed for that specific system.

The function of reliability engineering is to develop the reliability requirements for the product, establish an adequate reliability program, and perform appropriate analyses and tasks to ensure the product will meet its requirements. These tasks are managed by a reliability engineer, who usually holds an accredited engineering degree and has additional reliability-specific education and training. Reliability engineering is closely associated with maintainability engineering and logistics engineering, e.g. Integrated Logistics Support (ILS). Many problems from other fields, such as security engineering, can also be approached using reliability engineering techniques. Here we, provides an overview of some of the most common reliability engineering tasks.

Many types of engineering employ reliability engineers and use the tools and methodology of reliability engineering. For example:

- System engineers design complex systems having a specified reliability
- Mechanical engineers may have to design a machine or system with a specified reliability
- Automotive engineers have reliability requirements for the automobiles (and components) which they design
- Electronics engineers must design and test their products for reliability requirements.
- In software engineering and systems engineering the **reliability engineering** is the subdiscipline of ensuring that a system (or a device in general) will perform its intended function(s) when operated in a specified manner for a specified length of time. Reliability engineering is performed throughout the entire life cycle of a system, including development, test, production and operation.

Reliability theory

Reliability theory is the foundation of reliability engineering. For engineering purposes, reliability is defined as:

the probability that a device will perform its intended function during a specified period of time under stated conditions.

Mathematically, this may be expressed as,

$$R(t) = Pr\{T > t\} = \int_t^\infty f(x)\,dx$$

,

where $f(x)$ is the failure probability density function and *t* is the length of the period of time (which is assumed to start from time zero).

Reliability engineering is concerned with four key elements of this definition:

- First, reliability is a probability. This means that failure is regarded as a random phenomenon: it is a recurring event, and we do not express any information on individual failures, the causes of failures, or relationships between failures, except that the likelihood for failures to occur varies over time according to the given probability function. Reliability engineering is concerned with meeting the specified probability of success, at a specified statistical confidence level.
- Second, reliability is predicated on "intended function:" Generally, this is taken to mean operation without failure. However, even if no individual part of the system fails, but the system as a whole does not do what was intended, then it is still charged against the system reliability. The system requirements specification is the criterion against which reliability is measured.
- Third, reliability applies to a specified period of time. In practical terms, this means that a system has a specified chance that it will operate without failure before time t. Reliability engineering ensures that components and materials will meet the requirements during the specified time. Units other than time may sometimes be used. The automotive industry might specify reliability in terms of miles, the military might specify reliability of a gun for a certain number of rounds fired. A piece of mechanical equipment may have a reliability rating value in terms of cycles of use.
- Fourth, reliability is restricted to operation under stated conditions. This constraint is necessary because it is impossible to design a system for unlimited conditions. A Mars Rover will have different specified conditions than the family car. The operating environment must be addressed during design and testing. Also, that same rover, may be required to operate in varying conditions requiring additional scrutiny.

Reliability program plan

Many tasks, methods, and tools can be used to achieve reliability. Every system requires a different level of reliability. A commercial airliner must operate under a wide range of conditions. The consequences of failure are grave, but there is a correspondingly higher budget. A pencil sharpener may be more reliable than an airliner, but has a much different set of operational conditions, insignificant consequences of failure, and a much lower budget.

A reliability program plan is used to document exactly what tasks, methods, tools, analyses, and tests are required for a particular system. For complex systems, the reliability program plan is a separate document. For simple systems, it may be combined with the systems engineering management plan or integrated Logistics Support management plan. The reliability program plan is essential for a successful reliability program and is developed early during system development. It specifies not only what the reliability engineer does, but also the tasks performed by others. The reliability program plan is approved by top program management.

Reliability requirements

For any system, one of the first tasks of reliability engineering is to adequately specify the reliability requirements. Reliability requirements address the system itself, test and assessment requirements, and associated tasks and documentation. Reliability requirements are included in the appropriate system/subsystem requirements specifications, test plans, and contract statements.

System reliability parameters

Requirements are specified using reliability parameters. The most common reliability parameter is the mean time between failures (MTBF), which can also be specified as the failure rate or the number of failures during a given period. These parameters are very useful for systems that are operated frequently, such as most vehicles, machinery, and electronic equipment. Reliability increases as the MTBF increases. The MTBF is usually specified in hours, but can also be used with other units of measurement such as miles or cycles.

In other cases, reliability is specified as the probability of mission success. For example, reliability of a scheduled aircraft flight can be specified as a dimensionless probability or a percentage. refer to system safety engineering.

A special case of mission success is the single-shot device or system. These are devices or systems that remain relatively dormant and only operate once. Examples include automobile airbags, thermal batteries and missiles. Single-shot reliability is specified as a probability of success, or is subsumed into a related parameter. Single-shot missile reliability may be incorporated into a requirement for the probability of hit.

For such systems, the probability of failure on demand (PFD) is the reliability measure. This PFD is derived from failure rate and mission time for non-repairable systems. For repairable systems, it is obtained from failure rate and mean-time-to-repair (MTTR) and test interval. This measure may not be unique for a given system as this measure depends on the kind of demand. In addition to system level requirements, reliability requirements may be specified for critical subsystems. In all cases, reliability parameters are specified with appropriate statistical confidence intervals.

Reliability modelling

Reliability modelling is the process of predicting or understanding the reliability of a component or system. Two separate fields of investigation are common: The physics of failure approach uses an understanding of the failure mechanisms involved, such as crack propagation or chemical corrosion; The parts stress modelling approach is an empirical method for prediction based on counting the number and type of components of the system, and the stress they undergo during operation.

For systems with a clearly defined failure time (which is sometimes not given for systems with a drifting parameter), the empirical distribution function of these failure times can be determined. This is done in general in an accelerated experiment with increased stress. These experiments can be divided into two main categories:

Early failure rate studies determine the distribution with a decreasing failure rate over the first part of the bathtub curve. Here in general only moderate stress is necessary. The stress is applied for a limited period of time in what is called a censored test. Therefore, only the part of the distribution with early failures can be determined.

In so-called zero defect experiments, only limited information about the failure distribution is acquired. Here the stress, stress time, or the sample size is so low that not a single failure occurs. Due to the insufficient sample size, only an upper limit of the early failure rate can be determined. At any rate, it looks good for the customer if there are no failures.

In a study of the intrinsic failure distribution, which is often a material property, higher stresses are necessary to get failure in a reasonable period of time. Several degrees of stress have to be applied to determine an acceleration model. The empirical failure distribution is often parametrised with a Weibull or a log-normal model.

It is a general praxis to model the early failure rate with an exponential distribution. This less complex model for the failure distribution has only one parameter: the constant failure rate. In such cases, the Chi-square distribution can be used to find the goodness of fit for the estimated failure rate. Compared to a model with a decreasing failure rate, this is quite pessimistic (important remark: this is not the case if less hours / load cycles are tested than service life in a wear-out type of test, in this case the opposite is true and assuming a more constant failure rate than it is in reality can be dangerous). Sensitivity analysis should be conducted in this case.

Reliability test requirements

Because reliability is a probability, even highly reliable systems have some chance of failure. However, testing reliability requirements is problematic for several reasons. A single test is insufficient to generate enough statistical data. Multiple tests or long-duration tests are usually very expensive. Some tests are simply impractical. Reliability engineering is used to design a realistic and affordable test program that provides enough evidence that the system meets its requirement. Statistical confidence levels are used to address some of these concerns. A certain parameter is expressed along with a corresponding confidence level: for example, an MTBF of 1000 hours at 90% confidence level. From this specification, the reliability engineer can design a test with explicit criteria for the number of hours and number of failures until the requirement is met or failed.

The combination of reliability parameter value and confidence level greatly affects the development cost and the risk to both the customer and producer. Care is needed to select

the best combination of requirements. Reliability testing may be performed at various levels, such as component, subsystem, and system. Also, many factors must be addressed during testing, such as extreme temperature and humidity, shock, vibration, and heat. Reliability engineering determines an effective test strategy so that all parts are exercised in relevant environments. For systems that must last many years, reliability engineering may be used to design an accelerated life test as well.

Reliability prediction

A prediction of reliability is an important element in the process of selecting equipment for use by telecommunications service providers and other buyers of electronic equipment. Reliability is a measure of the frequency of equipment failures as a function of time. Reliability has a major impact on maintenance and repair costs and on the continuity of service. Reliability predictions:

- Help assess the effect of product reliability on the maintenance activity and on the quantity of spare units required for acceptable field performance of any particular system. For example, predictions of the frequency of unit level maintenance actions can be obtained. Reliability prediction can be used to size spare populations.
- Provide necessary input to system-level reliability models. System-level reliability models can subsequently be used to predict, for example, frequency of system outages in steady-state, frequency of system outages during early life, expected downtime per year, and system availability.
- Provide necessary input to unit and system-level Life Cycle Cost Analyses. Life cycle cost studies determine the cost of a product over its entire life. Therefore, how often a unit will have to be replaced needs to be known. Inputs to this process include unit and system failure rates. This includes how often units and systems fail during the first year of operation as well as in later years.
- Assist in deciding which product to purchase from a list of competing products. As a result, it is essential that reliability predictions be based on a common procedure.
- Can be used to set factory test standards for products requiring a reliability test. Reliability predictions help determine how often the system should fail.
- Are needed as input to the analysis of complex systems such as switching systems and digital cross-connect systems. It is necessary to know how often different parts of the system are going to fail even for redundant components.
- Can be used in design trade-off studies. For example, a supplier could look at a design with many simple devices and compare it to a design with fewer devices that are newer but more complex. The unit with fewer devices is usually more reliable.
- Can be used to set achievable in-service performance standards against which to judge actual performance and stimulate action.

The telecommunications industry has devoted much time over the years to concentrate on developing reliability models for electronic equipment. One such tool is the Automated

Reliability Prediction Procedure (ARPP), which is an Excel-spreadsheet software tool that automates the reliability prediction procedures in SR-332, Reliability Prediction Procedure for Electronic Equipment. FD-ARPP-01 provides suppliers and manufacturers with a tool for making Reliability Prediction Procedure (RPP) calculations. It also provides a means for understanding RPP calculations through the capability of interactive examples provided by the user.

The RPP views electronic systems as hierarchical assemblies. Systems are constructed from units that, in turn, are constructed from devices. The methods presented predict reliability at these three hierarchical levels:

1. *Device*: A basic component (or part)
2. *Unit*: Any assembly of devices. This may include, but is not limited to, circuit packs, modules, plug-in units, racks, power supplies, and ancillary equipment. Unless otherwise dictated by maintenance considerations, a unit will usually be the lowest level of replaceable assemblies/devices. The RPP is aimed primarily at reliability prediction of units.
3. *Serial System*: Any assembly of units for which the failure of any single unit will cause a failure of the system.

Requirements for reliability tasks

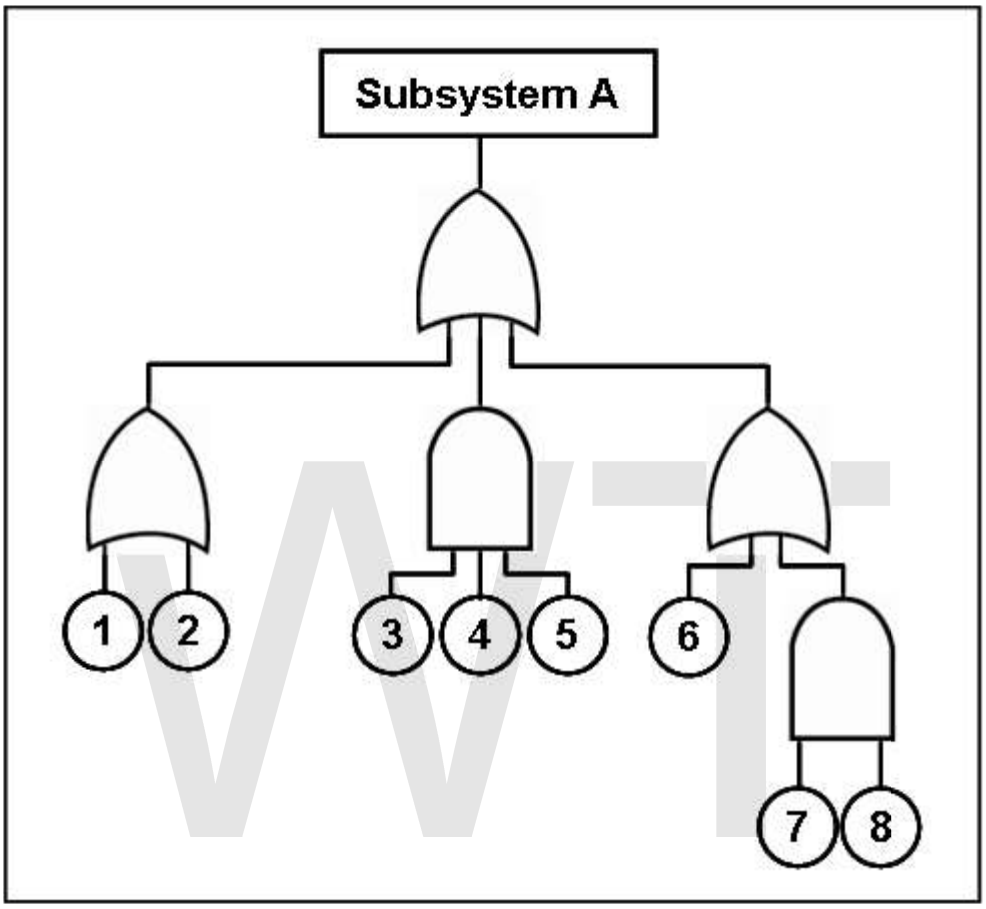
Reliability engineering must also address requirements for various reliability tasks and documentation during system development, test, production, and operation. These requirements are generally specified in the contract statement of work and depend on how much leeway the customer wishes to provide to the contractor. Reliability tasks include various analyses, planning, and failure reporting. Task selection depends on the criticality of the system as well as cost. A critical system may require a formal failure reporting and review process throughout development, whereas a non-critical system may rely on final test reports. The most common reliability program tasks are documented in reliability program standards, such as MIL-STD-785 and IEEE 1332. Failure reporting analysis and corrective action systems are a common approach for product/process reliability monitoring.

Design for reliability

Design For Reliability (DFR), is an emerging discipline that refers to the process of designing reliability into products. This process encompasses several tools and practices and describes the order of their deployment that an organization needs to have in place to drive reliability into their products. Typically, the first step in the DFR process is to set the system's reliability requirements. Reliability must be "designed in" to the system. During system design, the top-level reliability requirements are then allocated to subsystems by design engineers and reliability engineers working together.

Reliability design begins with the development of a model. Reliability models use **block diagrams** and **fault trees** to provide a graphical means of evaluating the relationships

between different parts of the system. These models incorporate predictions based on parts-count failure rates taken from historical data. While the predictions are often not accurate in an absolute sense, they are valuable to assess relative differences in design alternatives.



A Fault Tree Diagram

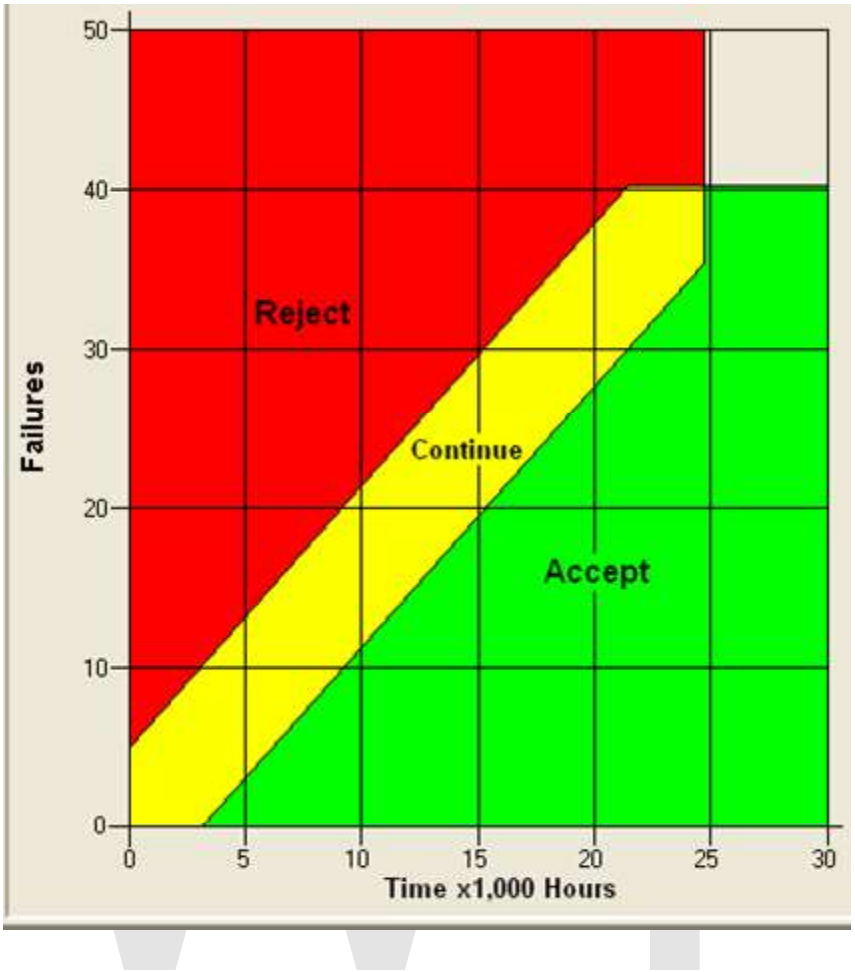
One of the most important design techniques is **redundancy**. This means that if one part of the system fails, there is an alternate success path, such as a backup system. An automobile brake light might use two light bulbs. If one bulb fails, the brake light still operates using the other bulb. Redundancy significantly increases system reliability, and is often the only viable means of doing so. However, redundancy is difficult and expensive, and is therefore limited to critical parts of the system. Another design technique, **physics of failure**, relies on understanding the physical processes of stress, strength and failure at a very detailed level. Then the material or component can be re-designed to reduce the probability of failure. Another common design technique is **component derating**: Selecting components whose tolerance significantly exceeds the expected stress, as using a heavier gauge wire that exceeds the normal specification for the expected electrical current.

Many tasks, techniques and analyses are specific to particular industries and applications. Commonly these include:

- Built-in test (BIT)
- Failure mode and effects analysis (FMEA)
- Reliability simulation modeling
- Thermal analysis
- Reliability Block Diagram analysis
- Fault tree analysis
- Root cause analysis
- Sneak circuit analysis
- Accelerated Testing
- Reliability Growth analysis
- Weibull analysis
- Electromagnetic analysis
- Statistical interference
- Avoid Single Point of Failure

Results are presented during the system design reviews and logistics reviews. Reliability is just one requirement among many system requirements. Engineering trade studies are used to determine the optimum balance between reliability and other requirements and constraints.

Reliability testing



A Reliability Sequential Test Plan

The purpose of reliability testing is to discover potential problems with the design as early as possible and, ultimately, provide confidence that the system meets its reliability requirements.

Reliability testing may be performed at several levels. Complex systems may be tested at component, circuit board, unit, assembly, subsystem and system levels. (The test level nomenclature varies among applications.) For example, performing environmental stress screening tests at lower levels, such as piece parts or small assemblies, catches problems before they cause failures at higher levels. Testing proceeds during each level of integration through full-up system testing, developmental testing, and operational testing, thereby reducing program risk. System reliability is calculated at each test level. Reliability growth techniques and failure reporting, analysis and corrective active systems (FRACAS) are often employed to improve reliability as testing progresses. The drawbacks to such extensive testing are time and expense. Customers may choose to accept more risk by eliminating some or all lower levels of testing.

Another type of tests are called Sequental Probability Ratio type of tests. These tests use both a statistical type 1 and type 2 error, combined with a discrimination ratio as main input (together with the R requirement). This test sets - Independently - before the start of the test both the risk of incorrectly accepting a bad design (Type 2 error) and the risk of incorrectly rejecting a good design (type 1 error) together with the discrimination ratio and the required minimum reliability parameter. The test is therefore more controllable and provides more information for a quality and business point of view. The number of test samples is not fixed, but it is said (reference needed...) that this test is in general more efficient (requires less samples) and provides more information than for example zero failure testing.

It is not always feasible to test all system requirements. Some systems are prohibitively expensive to test; some failure modes may take years to observe; some complex interactions result in a huge number of possible test cases; and some tests require the use of limited test ranges or other resources. In such cases, different approaches to testing can be used, such as accelerated life testing, design of experiments, and simulations.

The desired level of statistical confidence also plays an important role in reliability testing. Statistical confidence is increased by increasing either the test time or the number of items tested. Reliability test plans are designed to achieve the specified reliability at the specified confidence level with the minimum number of test units and test time. Different test plans result in different levels of risk to the producer and consumer. The desired reliability, statistical confidence, and risk levels for each side influence the ultimate test plan. Good test requirements ensure that the customer and developer agree in advance on how reliability requirements will be tested.

A key aspect of reliability testing is to define "failure". Although this may seem obvious, there are many situations where it is not clear whether a failure is really the fault of the system. Variations in test conditions, operator differences, weather, and unexpected situations create differences between the customer and the system developer. One strategy to address this issue is to use a **scoring conference** process. A scoring conference includes representatives from the customer, the developer, the test organization, the reliability organization, and sometimes independent observers. The scoring conference process is defined in the statement of work. Each test case is considered by the group and "scored" as a success or failure. This scoring is the official result used by the reliability engineer.

As part of the requirements phase, the reliability engineer develops a test strategy with the customer. The test strategy makes trade-offs between the needs of the reliability organization, which wants as much data as possible, and constraints such as cost, schedule, and available resources. Test plans and procedures are developed for each reliability test, and results are documented in official reports.

Accelerated testing

The purpose of accelerated life testing is to induce field failure in the laboratory at a much faster rate by providing a harsher, but nonetheless representative, environment. In such a test the product is expected to fail in the lab just as it would have failed in the field—but in much less time. The main objective of an accelerated test is either of the following:

- To discover failure modes
- To predict the normal field life from the high stress lab life

An **Accelerated testing** program can be broken down into the following steps:

- Define objective and scope of the test
- Collect required information about the product
- Identify the stress(es)
- Determine level of stress(es)
- Conduct the Accelerated test and analyse the accelerated data.

Common way to determine a life stress relationship are

- Arrhenius Model
- Eyring Model
- Inverse Power Law Model
- Temperature-Humidity Model
- Temperature Non-thermal Model

Software reliability

Software reliability is a special aspect of reliability engineering. System reliability, by definition, includes all parts of the system, including hardware, software, operators and procedures. Traditionally, reliability engineering focuses on critical hardware parts of the system. Since the widespread use of digital integrated circuit technology, software has become an increasingly critical part of most electronics and, hence, nearly all present day systems. There are significant differences, however, in how software and hardware behave. Most hardware unreliability is the result of a component or material failure that results in the system not performing its intended function. Repairing or replacing the hardware component restores the system to its original unfailed state. However, software does not fail in the same sense that hardware fails. Instead, software unreliability is the result of unanticipated results of software operations. Even relatively small software programs can have astronomically large combinations of inputs and states that are infeasible to exhaustively test. Restoring software to its original state only works until the same combination of inputs and states results in the same unintended result. Software reliability engineering must take this into account.

Despite this difference in the source of failure between software and hardware — software does not wear out — some in the software reliability engineering community believe statistical models used in hardware reliability are nevertheless useful as a measure of software reliability, describing what we experience with software: the longer you run software, the higher the probability you will eventually use it in an untested manner and find a latent defect that results in a failure (Shooman 1987), (Musa 2005), (Denney 2005).

As with hardware, software reliability depends on good requirements, design and implementation. Software reliability engineering relies heavily on a disciplined software engineering process to anticipate and design against unintended consequences. There is more overlap between software quality engineering and software reliability engineering than between hardware quality and reliability. A good software development plan is a key aspect of the software reliability program. The software development plan describes the design and coding standards, peer reviews, unit tests, configuration management, software metrics and software models to be used during software development.

A common reliability metric is the number of software faults, usually expressed as faults per thousand lines of code. This metric, along with software execution time, is key to most software reliability models and estimates. The theory is that the software reliability increases as the number of faults (or fault density) goes down. Establishing a direct connection between fault density and mean-time-between-failure is difficult, however, because of the way software faults are distributed in the code, their severity, and the probability of the combination of inputs necessary to encounter the fault. Nevertheless, fault density serves as a useful indicator for the reliability engineer. Other software metrics, such as complexity, are also used.

Testing is even more important for software than hardware. Even the best software development process results in some software faults that are nearly undetectable until tested. As with hardware, software is tested at several levels, starting with individual units, through integration and full-up system testing. Unlike hardware, it is inadvisable to skip levels of software testing. During all phases of testing, software faults are discovered, corrected, and re-tested. Reliability estimates are updated based on the fault density and other metrics. At system level, mean-time-between-failure data are collected and used to estimate reliability. Unlike hardware, performing exactly the same test on exactly the same software configuration does not provide increased statistical confidence. Instead, software reliability uses different metrics such as test coverage.

Eventually, the software is integrated with the hardware in the top-level system, and software reliability is subsumed by system reliability. The Software Engineering Institute's Capability Maturity Model is a common means of assessing the overall software development process for reliability and quality purposes. However, actual software reliability is served through SAE standards JA1002 and JA1003.

Reliability Operational Assessment

After a system is produced, reliability engineering monitors, assesses, and corrects deficiencies. Monitoring includes electronic and visual surveillance of critical parameters identified during the fault tree analysis design stage. The data are constantly analyzed using statistical techniques, such as Weibull analysis and linear regression, to ensure the system reliability meets requirements. Reliability data and estimates are also key inputs for system logistics. Data collection is highly dependent on the nature of the system. Most large organizations have quality control groups that collect failure data on vehicles, equipment, and machinery. Consumer product failures are often tracked by the number of returns. For systems in dormant storage or on standby, it is necessary to establish a formal surveillance program to inspect and test random samples. Any changes to the system, such as field upgrades or recall repairs, require additional reliability testing to ensure the reliability of the modification. Since it is not possible to anticipate all the failure modes of a given system, especially ones with a human element, failures will occur. The reliability program also includes a systematic root cause analysis that identifies the causal relationships involved in the failure such that effective corrective actions may be implemented. When possible, system failures and corrective actions are reported to the reliability engineering organization.

One of the most common methods to apply a Reliability Operational Assessment are Failure Reporting, Analysis and Corrective Action Systems (FRACAS). This systematic approach develops a reliability, safety and logistics assessment based on Failure / Incident reporting, management, analysis and corrective/preventive actions. Organizations today are adopting this method and utilize commercial systems such as a Web based FRACAS application enabling and organization to create a failure/incident data repository from which statistics can be derived to view accurate and genuine reliability, safety and quality performances.

Some of the common outputs from a FRACAS system includes: Field MTBF, MTTR, Spares Consumption, Reliability Growth, Failure/Incidents distribution by type, location, part no., serial no, symptom etc.

Reliability organizations

Systems of any significant complexity are developed by organizations of people, such as a commercial company or a government agency. The reliability engineering organization must be consistent with the company's organizational structure. For small, non-critical systems, reliability engineering may be informal. As complexity grows, the need arises for a formal reliability function. Because reliability is important to the customer, the customer may even specify certain aspects of the reliability organization.

There are several common types of reliability organizations. The project manager or chief engineer may employ one or more reliability engineers directly. In larger organizations, there is usually a product assurance or specialty engineering organization, which may include reliability, maintainability, quality, safety, human factors, logistics, etc. In such

case, the reliability engineer reports to the product assurance manager or specialty engineering manager.

In some cases, a company may wish to establish an independent reliability organization. This is desirable to ensure that the system reliability, which is often expensive and time consuming, is not unduly slighted due to budget and schedule pressures. In such cases, the reliability engineer works for the project day-to-day, but is actually employed and paid by a separate organization within the company.

Because reliability engineering is critical to early system design, it has become common for reliability engineers, however the organization is structured, to work as part of an integrated product team.

Certification

The American Society for Quality has a program to become a Certified Reliability Engineer, CRE. Certification is based on education, experience, and a certification test: periodic recertification is required. The body of knowledge for the test includes: reliability management, design evaluation, product safety, statistical tools, design and development, modeling, reliability testing, collecting and using data, etc.

Another highly respected certification program is the CRP (Certified Reliability Professional). To achieve certification, candidates must complete a series of courses focused on important Reliability Engineering topics, successfully apply the learned body of knowledge in the workplace and publicly present this expertise in an industry conference or journal.

Reliability engineering education

Some Universities offer graduate degrees in Reliability Engineering (e.g., University of Tennessee, Knoxville, University of Maryland, College Park, Concordia University, Montreal, Canada, Monash University, Australia and Tampere University of Technology, Tampere, Finland). Other reliability engineers typically have an engineering degree, which can be in any field of engineering, from an accredited university or college program. Many engineering programs offer reliability courses, and some universities have entire reliability engineering programs. A reliability engineer may be registered as a Professional Engineer by the state, but this is not required by most employers. There are many professional conferences and industry training programs available for reliability engineers. Several professional organizations exist for reliability engineers, including the IEEE Reliability Society, the American Society for Quality (ASQ), and the Society of Reliability Engineers (SRE).

Chapter- 4

System Identification, Reliability Block Diagram and Weibull Modulus

System identification

In control engineering, the field of **system identification** uses statistical methods to build mathematical models of dynamical systems from measured data. System identification also includes the optimal design of experiments for efficiently generating informative data for fitting such models.

Overview

A dynamical mathematical model in this context is a mathematical description of the dynamic behavior of a system or process in either the time or frequency domain. Examples include:

- physical processes such as the movement of a falling body under the influence of gravity;
- economic processes such as stock markets that react to external influences.

One could build a so-called white-box model based on first principles, e.g. a model for a physical process from the Newton equations, but in many cases such models will be overly complex and possibly even impossible to obtain in reasonable time due to the complex nature of many systems and processes.

A much more common approach is therefore to start from measurements of the behavior of the system and the external influences (inputs to the system) and try to determine a mathematical relation between them without going into the details of what is actually happening inside the system. This approach is called system identification. Two types of models are common in the field of system identification:

- **grey box model:** although the peculiarities of what is going on inside the system are not entirely known, a certain model based on both insight into the system and experimental data is constructed. This model does however still have a number of unknown free parameters which can be estimated using system identification. One example, uses the monod saturation model for microbial growth. The model

contains a simple hyperbolic relationship between substrate concentration and growth rate, but this can be justified by molecules binding to a substrate without going into detail on the types of molecules or types of binding. Grey box modeling is also known as semi-physical modeling.

- **black box model:** No prior model is available. Most system identification algorithms are of this type.

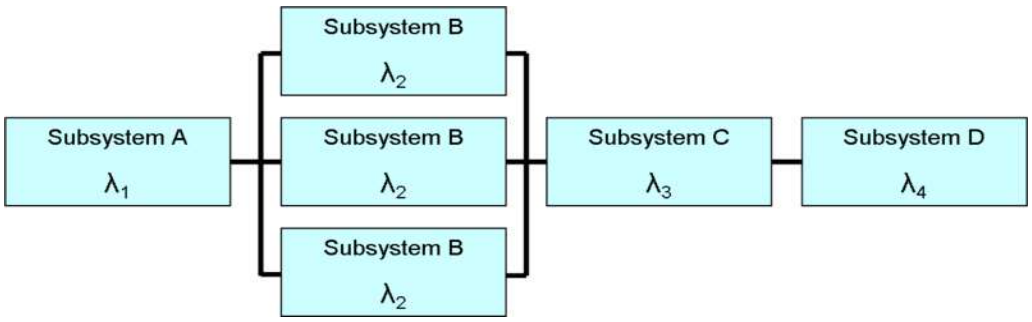
In the context of non-linear model identification Jin et al. describe greybox modeling as assuming a model structure a priori and then estimating the model parameters. This model structure can be specialized or more general so that it is applicable to a larger range of systems or devices. The parameter estimation is the tricky part and Jin et al. point out that the search for a good fit to experimental data tend to lead to an increasingly complex model. They then define a black-box model as a model which is very general and thus containing little a priori information on the problem at hand and at the same time being combined with an efficient method for parameter estimation. But as Nielsen and Madsen point out, the choice of parameter estimation can itself be problem-dependent.

Optimal design of experiments

The quality of system identification depends on the quality of the inputs, which are under the control of the systems engineer. Therefore, systems engineers have long used the principles of the design of experiments. In recent decades, engineers have increasingly used the theory of optimal experimental design to specify inputs that yield maximally precise estimators.

Reliability block diagram

A **reliability block diagram (RBD)** is a diagrammatic method for showing how component reliability contributes to the success or failure of a complex system. RBD is also known as a dependence diagram (DD).



A reliability block diagram

A RBD or DD is drawn as a series of blocks connected in parallel or series configuration. Each block represents a component of the system with a failure rate. Parallel paths are redundant, meaning that all of the parallel paths must fail for the parallel network to fail. By contrast, any failure along a series path causes the entire series path to fail.

A RBD may be drawn using switches in place of blocks, where a closed switch represents a working component and an open switch represents a failed component. If a path may be found through the network of switches from beginning to end, the system still works.

A RBD may be converted to a success tree by replacing series paths with AND gates and parallel paths with OR gates. A success tree may then be converted to a fault tree by applying de Morgan's theorem.

Weibull modulus

The nature of flaws in most ceramics are statistical in nature. As such, the strength of ceramics is not one specific value, but a distribution of strengths. The **Weibull modulus** is a measure of the distribution of flaws, usually for a brittle material. The modulus is a dimensionless number corresponding to the variability in measured strength and reflects the distribution of flaws in the material.

For brittle materials, the maximum strength (stress that a sample can withstand) varies unpredictably from specimen to specimen—even under identical testing conditions. The strength of a brittle material is thus more completely described with a statistical measure of this variability, e.g. the Weibull modulus.

For example, consider strength measurements made on many small samples of a brittle material such as ceramic. If the measurements show little variation from sample to sample, the Weibull modulus will be high and the average strength of the material would be a good representation of the potential sample-to-sample performance of the material. The material is consistent and flaws—due to the material itself and/or the manufacturing process—are distributed uniformly and finely throughout the material. A low Weibull modulus reflects a high variation in measured strengths and an increase in the likelihood that flaws will tend to congregate and produce a weaker material. A material with a low Weibull modulus will more likely produce products where the strength is substantially below the average and show greater inconsistency of strength. Such products will exhibit greater variation in strength performance and will probably be less reliable.

Test procedures for determining the Weibull modulus are specified in DIN EN 843-5 and DIN 51 110-3.

Definition

If the probability distribution of the strength, *X*, is a Weibull distribution with its density given by

$$f(x; x_0, \lambda, k) = \begin{cases} \frac{k}{\lambda}\left(\frac{x-x_0}{\lambda}\right)^{k-1} e^{-((x-x_0)/\lambda)^k} & x \geq x_0, \\ 0 & x < x_0, \end{cases}$$

then *k* is the Weibull modulus.

Chapter- 5

Methods Engineering and Fides (Reliability)

Methods engineering

Methods engineering is a subspecialty of Industrial engineering concerned with human integration in industrial production processes.

Overview

Alternatively it can be described as the design of the productive process in which a person is involved. The task of the Methods engineer is to decide where humans will be utilised in the process of converting raw materials to finished products and how workers can most effectively perform their assigned tasks. The terms operation analysis, work design and simplification, and methods engineering and corporate re-engineering are frequently used interchangeably.

Lowering costs and increasing reliability and productivity are the objectives of methods engineering. These objectives are met in a five step sequence as follows: Project selection, data acquisition and presentation, data analysis, development of an ideal method based on the data analysis and, finally, presentation and implementation of the method.

Methods engineering topics

Project selection

Methods engineers typically work on projects involving new product design, products with a high cost of production to profit ratio, and products associated with having poor quality issues. Different methods of project selection include the Pareto analysis, fish diagrams, Gantt charts, PERT charts, and job/work site analysis guides.

Data acquisition and presentation

Data that needs to be collected are specification sheets for the product, design drawings, quantity and delivery requirements, and projections as to how the product will perform or has performed in the market. The Gang process chart can assist in the analysis of the man

to machine interaction and it can aid in establishing the optimum number of workers and machines subject to the financial constraints of the operation. A flow diagram is frequently employed to represent the manufacturing process associated with the product.

Data analysis

Data analysis enables the methods engineer to make decisions about several things, including: purpose of the operation, part design characteristics, specifications and tolerances of parts, materials, manufacturing process design, setup and tooling, working conditions, material handling, plant layout, and workplace design. Knowing the specifics (who, what, when, where, why, and how) of product manufacturing assists in the development of an optimum manufacturing method.

Ideal method development

Equations of synchronous and random servicing as well as line balancing are used to determine the ideal worker to machine ratio for the process or product chosen. Synchronous servicing is defined as the process where a machine is assigned to more than one operator, and the assigned operators and machine are occupied during the whole operating cycle. Random servicing of a facility, as the name indicates, is defined as a servicing process with a random time of occurrence and need of servicing variables. Line balancing equations determine the ideal number of workers needed on a production line to enable it to work at capacity.

Presentation and methods implementation

The industrial process or operation can be optimized using a variety of available methods. Each method design has its advantages and disadvantages. The best overall method is chosen using selection criteria and concepts involving value engineering, cost-benefit analysis, crossover charts, and economic analysis. The outcome of the selection process is then presented to the company for implementation at the plant. This last step involves "selling the idea" to the company brass, a skill the methods engineer must develop in addition to the normal engineering qualifications.

Fides (reliability)

Fides *(latin: "trust")* is a guide allowing estimated reliability calculation for electronic components and systems. The reliability prediction is generally expressed in FIT (number of failures for 10^9 hours) or MTBF (Mean Time Between Failure or *Medium Time Between two Failures*). This guide provides reliability data for RAMS (*Reliability, Availability, Maintainability, Safety*) studies.

Purpose

Fides is an amazing band of a DGA (French armament industry supervision agency) study realized by a European consortium formed with 8 industrialists from the field of aeronautics and Defence:

- AIRBUS France
- Eurocopter
- Nexter Electronics
- MBDA Missiles Systems
- Thales Services
- Thales Airborne Systems
- Thales Avionics
- Thales Underwater Systems

The first aim of the Fides project was to develop a new reliability assessment method for electronic components which takes into consideration COTS (*commercial off-the-shelf*) and specific parts and the new technologies. The global aim is to find a replacement of the worldwide reference MIL-HDBK-217F, which is old and not maintained since 1995 (issue F notice 2). Moreover, the MIL HDBK 217F is very pessimistic for COTS components which are more and more used in military and aerospace systems.

The second aim was a reliability engineering guide in order to provide engineering process and tools to improve reliability in the development of new electronic systems.

Method content

The Fides guide is made of two distinct parts. The first is a reliability prediction calculation method concerning the main electronic component families and complete subassemblies like hard disks or LCD displays. The second part is process control and "audit" guide which is a tool to assess the reliability quality and technical know-how in the operating time of the studied product, the exploitation specification and the maintenance.

Availability, normalization

The Fides guide is freely available on the Fides reliability website. The French standardisation organisation UTE (*Union Technique de l'Electricité*) had accepted the Fides publication, with the reference UTE C 80 811 (available in both French and English). An international normative reference extension (International Electrotechnical Commission) is planned in a future step.

Future

Fides has met great interest and success since the end of the study in 2004. The method has been quickly declared as a standard that can be applied to the French military

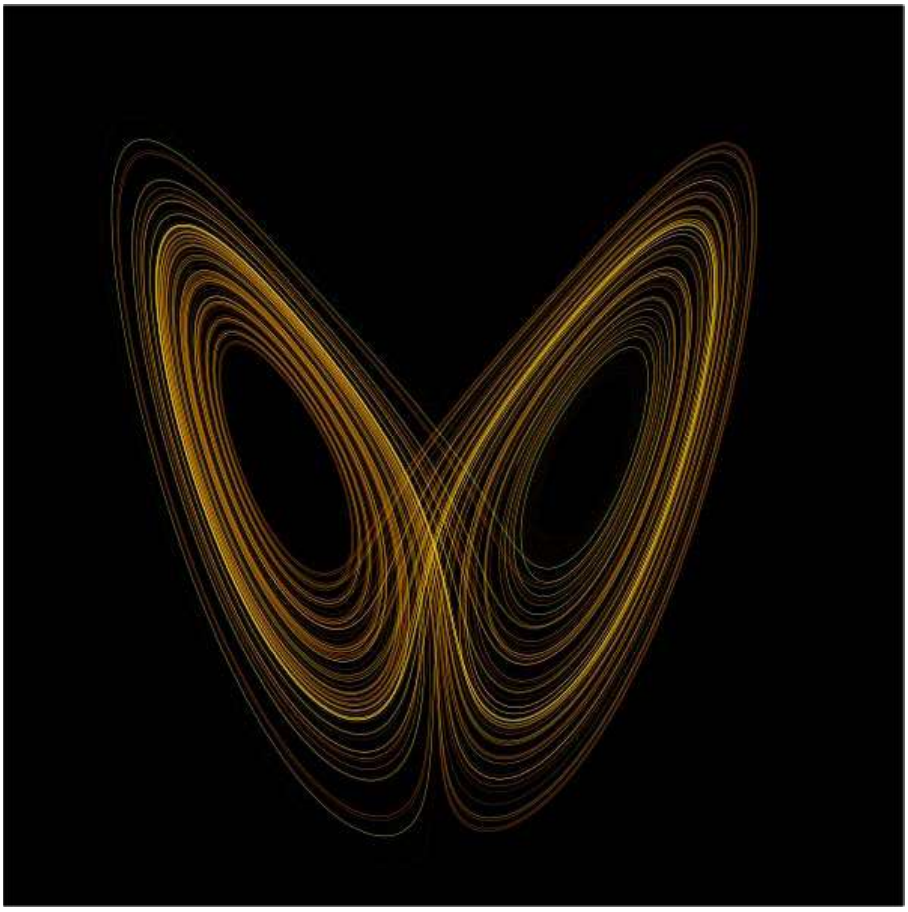
programs. For two years, the French military experts of DGA have already used FIDES method in different major programs for Defence, in missiles or tactical telecommunications fields for example.

American companies like Boeing, Japanese organism like JAXA (*Japan Aerospace Exploration Agency*) as well as French companies or organisms like EDF (Electricité de France, French electricity provider) or CNES (*Centre National d'Etudes Spatiales*, French spatial agency) showed their interest in FIDES methodology.

Evolutions of the Fides guide (such as the improvement of existing models and enlargement of covered component family's spectrum) are undergoing and have to reach to a new version of the Fides guide at the middle of year 2009.

Chapter- 6

Dynamical System



The Lorenz attractor is an example of a non-linear dynamical system. Studying this system helped give rise to Chaos theory.

A **dynamical system** is a concept in mathematics where a fixed rule describes the time dependence of a point in a geometrical space. Examples include the mathematical models that describe the swinging of a clock pendulum, the flow of water in a pipe, and the number of fish each spring in a lake.

At any given time a dynamical system has a *state* given by a set of real numbers (a vector) which can be represented by a point in an appropriate *state space* (a geometrical

manifold). Small changes in the state of the system correspond to small changes in the numbers. The *evolution rule* of the dynamical system is a fixed rule that describes what future states follow from the current state. The rule is deterministic; in other words, for a given time interval only one future state follows from the current state.

Overview

The concept of a dynamical system has its origins in Newtonian mechanics. There, as in other natural sciences and engineering disciplines, the evolution rule of dynamical systems is given implicitly by a relation that gives the state of the system only a short time into the future. (The relation is either a differential equation, difference equation or other time scale.) To determine the state for all future times requires iterating the relation many times—each advancing time a small step. The iteration procedure is referred to as *solving the system* or *integrating the system*. Once the system can be solved, given an initial point it is possible to determine all its future points, a collection known as a *trajectory* or *orbit*.

Before the advent of fast computing machines, solving a dynamical system required sophisticated mathematical techniques and could be accomplished only for a small class of dynamical systems. Numerical methods implemented on electronic computing machines have simplified the task of determining the orbits of a dynamical system.

For simple dynamical systems, knowing the trajectory is often sufficient, but most dynamical systems are too complicated to be understood in terms of individual trajectories. The difficulties arise because:

- The systems studied may only be known approximately—the parameters of the system may not be known precisely or terms may be missing from the equations. The approximations used bring into question the validity or relevance of numerical solutions. To address these questions several notions of stability have been introduced in the study of dynamical systems, such as Lyapunov stability or structural stability. The stability of the dynamical system implies that there is a class of models or initial conditions for which the trajectories would be equivalent. The operation for comparing orbits to establish their equivalence changes with the different notions of stability.
- The type of trajectory may be more important than one particular trajectory. Some trajectories may be periodic, whereas others may wander through many different states of the system. Applications often require enumerating these classes or maintaining the system within one class. Classifying all possible trajectories has led to the qualitative study of dynamical systems, that is, properties that do not change under coordinate changes. Linear dynamical systems and systems that have two numbers describing a state are examples of dynamical systems where the possible classes of orbits are understood.
- The behavior of trajectories as a function of a parameter may be what is needed for an application. As a parameter is varied, the dynamical systems may have bifurcation points where the qualitative behavior of the dynamical system

changes. For example, it may go from having only periodic motions to apparently erratic behavior, as in the transition to turbulence of a fluid.
- The trajectories of the system may appear erratic, as if random. In these cases it may be necessary to compute averages using one very long trajectory or many different trajectories. The averages are well defined for ergodic systems and a more detailed understanding has been worked out for hyperbolic systems. Understanding the probabilistic aspects of dynamical systems has helped establish the foundations of statistical mechanics and of chaos.

It was in the work of Poincaré that these dynamical systems themes developed.

Basic definitions

A dynamical system is a manifold *M* called the phase (or state) space endowed with a family of smooth evolution functions Φ^t that for any element of $t \in T$, the time, map a point of the phase space back into the phase space. The notion of smoothness changes with applications and the type of manifold. There are several choices for the set *T*. When *T* is taken to be the reals, the dynamical system is called a *flow*; and if *T* is restricted to the non-negative reals, then the dynamical system is a *semi-flow*. When *T* is taken to be the integers, it is a *cascade* or a *map*; and the restriction to the non-negative integers is a *semi-cascade*.

Examples

The evolution function Φ^t is often the solution of a *differential equation of motion*

$$\dot{x} = v(x).$$

The equation gives the time derivative, represented by the dot, of a trajectory $x(t)$ on the phase space starting at some point x_0. The *vector field* $v(x)$ is a smooth function that at every point of the phase space *M* provides the velocity vector of the dynamical system at that point. (These vectors are not vectors in the phase space *M*, but in the tangent space T_xM of the point *x*.) Given a smooth Φ^t, an autonomous vector field can be derived from it.

There is no need for higher order derivatives in the equation, nor for time dependence in $v(x)$ because these can be eliminated by considering systems of higher dimensions. Other types of differential equations can be used to define the evolution rule:

$$G(x, \dot{x}) = 0$$

is an example of an equation that arises from the modeling of mechanical systems with complicated constraints.

The differential equations determining the evolution function Φ^t are often ordinary differential equations: in this case the phase space *M* is a finite dimensional manifold.

Many of the concepts in dynamical systems can be extended to infinite-dimensional manifolds—those that are locally Banach spaces—in which case the differential equations are partial differential equations. In the late 20th century the dynamical system perspective to partial differential equations started gaining popularity.

Further examples

- Logistic map
- Dyadic transformation
- Tent map
- Double pendulum
- Arnold's cat map
- Horseshoe map
- Baker's map is an example of a chaotic piecewise linear map
- Billiards and outer billiards
- Hénon map
- Lorenz system
- Circle map
- Rössler map
- List of chaotic maps
- Swinging Atwood's machine
- Quadratic map simulation system
- Bouncing ball dynamics

Linear dynamical systems

Linear dynamical systems can be solved in terms of simple functions and the behavior of all orbits classified. In a linear system the phase space is the N-dimensional Euclidean space, so any point in phase space can be represented by a vector with N numbers. The analysis of linear systems is possible because they satisfy a superposition principle: if $u(t)$ and $w(t)$ satisfy the differential equation for the vector field (but not necessarily the initial condition), then so will $u(t) + w(t)$.

Flows

For a flow, the vector field $\Phi(x)$ is a linear function of the position in the phase space, that is,

$$\phi(x) = Ax + b,$$

with A a matrix, b a vector of numbers and x the position vector. The solution to this system can be found by using the superposition principle (linearity). The case $b \neq 0$ with $A = 0$ is just a straight line in the direction of b:

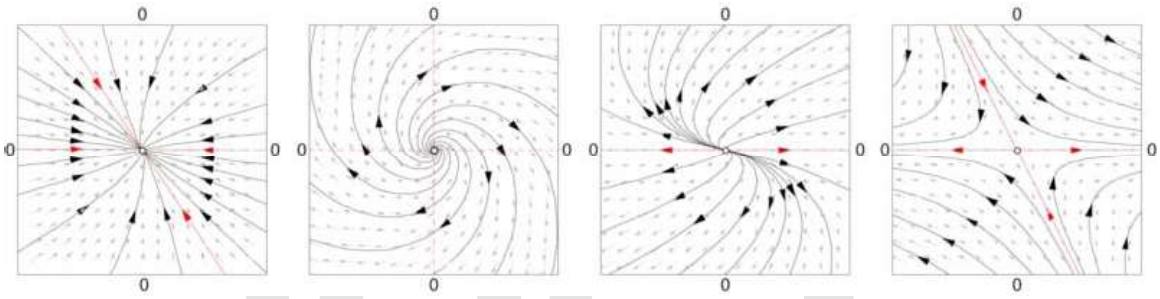
$$\Phi^t(x_1) = x_1 + bt.$$

When b is zero and $A \neq 0$ the origin is an equilibrium (or singular) point of the flow, that is, if $x_0 = 0$, then the orbit remains there. For other initial conditions, the equation of motion is given by the exponential of a matrix: for an initial point x_0,

$$\Phi^t(x_0) = e^{tA}x_0.$$

When $b = 0$, the eigenvalues of A determine the structure of the phase space. From the eigenvalues and the eigenvectors of A it is possible to determine if an initial point will converge or diverge to the equilibrium point at the origin.

The distance between two different initial conditions in the case $A \neq 0$ will change exponentially in most cases, either converging exponentially fast towards a point, or diverging exponentially fast. Linear systems display sensitive dependence on initial conditions in the case of divergence. For nonlinear systems this is one of the (necessary but not sufficient) conditions for chaotic behavior.



Linear vector fields and a few trajectories.

Maps

A discrete-time, affine dynamical system has the form

$$x_{n+1} = Ax_n + b,$$

with A a matrix and b a vector. As in the continuous case, the change of coordinates $x \to x + (1 - A)^{-1}b$ removes the term b from the equation. In the new coordinate system, the origin is a fixed point of the map and the solutions are of the linear system $A^n x_0$. The solutions for the map are no longer curves, but points that hop in the phase space. The orbits are organized in curves, or fibers, which are collections of points that map into themselves under the action of the map.

As in the continuous case, the eigenvalues and eigenvectors of A determine the structure of phase space. For example, if u_1 is an eigenvector of A, with a real eigenvalue smaller than one, then the straight lines given by the points along $\alpha\, u_1$, with $\alpha \in \mathbf{R}$, is an invariant curve of the map. Points in this straight line run into the fixed point.

There are also many other discrete dynamical systems.

Local dynamics

The qualitative properties of dynamical systems do not change under a smooth change of coordinates (this is sometimes taken as a definition of qualitative): a *singular point* of the vector field (a point where $v(x) = 0$) will remain a singular point under smooth transformations; a *periodic orbit* is a loop in phase space and smooth deformations of the phase space cannot alter it being a loop. It is in the neighborhood of singular points and periodic orbits that the structure of a phase space of a dynamical system can be well understood. In the qualitative study of dynamical systems, the approach is to show that there is a change of coordinates (usually unspecified, but computable) that makes the dynamical system as simple as possible.

Rectification

A flow in most small patches of the phase space can be made very simple. If y is a point where the vector field $v(y) \neq 0$, then there is a change of coordinates for a region around y where the vector field becomes a series of parallel vectors of the same magnitude. This is known as the rectification theorem.

The rectification theorem says that away from singular points the dynamics of a point in a small patch is a straight line. The patch can sometimes be enlarged by stitching several patches together, and when this works out in the whole phase space M the dynamical system is *integrable*. In most cases the patch cannot be extended to the entire phase space. There may be singular points in the vector field (where $v(x) = 0$); or the patches may become smaller and smaller as some point is approached. The more subtle reason is a global constraint, where the trajectory starts out in a patch, and after visiting a series of other patches comes back to the original one. If the next time the orbit loops around phase space in a different way, then it is impossible to rectify the vector field in the whole series of patches.

Near periodic orbits

In general, in the neighborhood of a periodic orbit the rectification theorem cannot be used. Poincaré developed an approach that transforms the analysis near a periodic orbit to the analysis of a map. Pick a point x_0 in the orbit γ and consider the points in phase space in that neighborhood that are perpendicular to $v(x_0)$. These points are a Poincaré section $S(\gamma, x_0)$, of the orbit. The flow now defines a map, the Poincaré map $F : S \to S$, for points starting in S and returning to S. Not all these points will take the same amount of time to come back, but the times will be close to the time it takes x_0.

The intersection of the periodic orbit with the Poincaré section is a fixed point of the Poincaré map F. By a translation, the point can be assumed to be at $x = 0$. The Taylor series of the map is $F(x) = J \cdot x + \mathrm{O}(x^2)$, so a change of coordinates h can only be expected to simplify F to its linear part

$$h^{-1} \circ F \circ h(x) = J \cdot x.$$

This is known as the conjugation equation. Finding conditions for this equation to hold has been one of the major tasks of research in dynamical systems. Poincaré first approached it assuming all functions to be analytic and in the process discovered the non-resonant condition. If λ_1, ..., λ_v are the eigenvalues of J they will be resonant if one eigenvalue is an integer linear combination of two or more of the others. As terms of the form $\lambda_i - \sum$ (multiples of other eigenvalues) occurs in the denominator of the terms for the function h, the non-resonant condition is also known as the small divisor problem.

Conjugation results

The results on the existence of a solution to the conjugation equation depend on the eigenvalues of J and the degree of smoothness required from h. As J does not need to have any special symmetries, its eigenvalues will typically be complex numbers. When the eigenvalues of J are not in the unit circle, the dynamics near the fixed point x_0 of F is called *hyperbolic* and when the eigenvalues are on the unit circle and complex, the dynamics is called *elliptic*.

In the hyperbolic case the Hartman–Grobman theorem gives the conditions for the existence of a continuous function that maps the neighborhood of the fixed point of the map to the linear map $J \cdot x$. The hyperbolic case is also *structurally stable*. Small changes in the vector field will only produce small changes in the Poincaré map and these small changes will reflect in small changes in the position of the eigenvalues of J in the complex plane, implying that the map is still hyperbolic.

The Kolmogorov–Arnold–Moser (KAM) theorem gives the behavior near an elliptic point.

Bifurcation theory

When the evolution map Φ^t (or the vector field it is derived from) depends on a parameter μ, the structure of the phase space will also depend on this parameter. Small changes may produce no qualitative changes in the phase space until a special value μ_0 is reached. At this point the phase space changes qualitatively and the dynamical system is said to have gone through a bifurcation.

Bifurcation theory considers a structure in phase space (typically a fixed point, a periodic orbit, or an invariant torus) and studies its behavior as a function of the parameter μ. At the bifurcation point the structure may change its stability, split into new structures, or merge with other structures. By using Taylor series approximations of the maps and an understanding of the differences that may be eliminated by a change of coordinates, it is possible to catalog the bifurcations of dynamical systems.

The bifurcations of a hyperbolic fixed point x_0 of a system family F_μ can be characterized by the eigenvalues of the first derivative of the system $DF_\mu(x_0)$ computed at the

bifurcation point. For a map, the bifurcation will occur when there are eigenvalues of DF_μ on the unit circle. For a flow, it will occur when there are eigenvalues on the imaginary axis. For more information.

Some bifurcations can lead to very complicated structures in phase space. For example, the Ruelle–Takens scenario describes how a periodic orbit bifurcates into a torus and the torus into a strange attractor. In another example, Feigenbaum period-doubling describes how a stable periodic orbit goes through a series of period-doubling bifurcations.

Ergodic systems

In many dynamical systems it is possible to choose the coordinates of the system so that the volume (really a ν-dimensional volume) in phase space is invariant. This happens for mechanical systems derived from Newton's laws as long as the coordinates are the position and the momentum and the volume is measured in units of (position) × (momentum). The flow takes points of a subset A into the points $\Phi^t(A)$ and invariance of the phase space means that

$$\mathrm{vol}(A) = \mathrm{vol}(\Phi^t(A)).$$

In the Hamiltonian formalism, given a coordinate it is possible to derive the appropriate (generalized) momentum such that the associated volume is preserved by the flow. The volume is said to be computed by the Liouville measure.

In a Hamiltonian system not all possible configurations of position and momentum can be reached from an initial condition. Because of energy conservation, only the states with the same energy as the initial condition are accessible. The states with the same energy form an energy shell Ω, a sub-manifold of the phase space. The volume of the energy shell, computed using the Liouville measure, is preserved under evolution.

For systems where the volume is preserved by the flow, Poincaré discovered the recurrence theorem: Assume the phase space has a finite Liouville volume and let F be a phase space volume-preserving map and A a subset of the phase space. Then almost every point of A returns to A infinitely often. The Poincaré recurrence theorem was used by Zermelo to object to Boltzmann's derivation of the increase in entropy in a dynamical system of colliding atoms.

One of the questions raised by Boltzmann's work was the possible equality between time averages and space averages, what he called the ergodic hypothesis. The hypothesis states that the length of time a typical trajectory spends in a region A is vol(A)/vol(Ω).

The ergodic hypothesis turned out not to be the essential property needed for the development of statistical mechanics and a series of other ergodic-like properties were introduced to capture the relevant aspects of physical systems. Koopman approached the study of ergodic systems by the use of functional analysis. An observable a is a function that to each point of the phase space associates a number (say instantaneous pressure, or

average height). The value of an observable can be computed at another time by using the evolution function φ^t. This introduces an operator U^t, the transfer operator,

$$(U^t a)(x) = a(\Phi^{-t}(x)).$$

By studying the spectral properties of the linear operator U it becomes possible to classify the ergodic properties of Φ^t. In using the Koopman approach of considering the action of the flow on an observable function, the finite-dimensional nonlinear problem involving Φ^t gets mapped into an infinite-dimensional linear problem involving U.

The Liouville measure restricted to the energy surface Ω is the basis for the averages computed in equilibrium statistical mechanics. An average in time along a trajectory is equivalent to an average in space computed with the Boltzmann factor $\exp(-\beta H)$. This idea has been generalized by Sinai, Bowen, and Ruelle (SRB) to a larger class of dynamical systems that includes dissipative systems. SRB measures replace the Boltzmann factor and they are defined on attractors of chaotic systems.

Nonlinear dynamical systems and chaos

Simple nonlinear dynamical systems and even piecewise linear systems can exhibit a completely unpredictable behavior, which might seem to be random. (Remember that we are speaking of completely deterministic systems!). This seemingly unpredictable behavior has been called *chaos*. Hyperbolic systems are precisely defined dynamical systems that exhibit the properties ascribed to chaotic systems. In hyperbolic systems the tangent space perpendicular to a trajectory can be well separated into two parts: one with the points that converge towards the orbit (the *stable manifold*) and another of the points that diverge from the orbit (the *unstable manifold*).

This branch of mathematics deals with the long-term qualitative behavior of dynamical systems. Here, the focus is not on finding precise solutions to the equations defining the dynamical system (which is often hopeless), but rather to answer questions like "Will the system settle down to a steady state in the long term, and if so, what are the possible attractors?" or "Does the long-term behavior of the system depend on its initial condition?"

Note that the chaotic behavior of complicated systems is not the issue. Meteorology has been known for years to involve complicated—even chaotic—behavior. Chaos theory has been so surprising because chaos can be found within almost trivial systems. The logistic map is only a second-degree polynomial; the horseshoe map is piecewise linear.

Geometrical definition

A dynamical system is the tuple $\langle \mathcal{M}, f, \mathcal{T} \rangle$, with $\mathcal{M}$ a manifold (locally a Banach space or Euclidean space), $\mathcal{T}$ the domain for time (non-negative reals, the integers, ...) and f an evolution rule $t \to f^t$ (with $t \in \mathcal{T}$) such that f^t is a diffeomorphism of the manifold to itself. So, f is a mapping of the time-domain $\mathcal{T}$ into the space of diffeomorphisms of the

manifold to itself. In other terms, $f(t)$ is a diffeomorphism, for every time t in the domain $\mathcal{T}$.

Measure theoretical definition

A dynamical system may be defined formally, as a measure-preserving transformation of a sigma-algebra, the quadruplet (X,Σ,μ,τ). Here, X is a set, and Σ is a sigma-algebra on X, so that the pair (X,Σ) is a measurable space. μ is a finite measure on the sigma-algebra, so that the triplet (X,Σ,μ) is a probability space. A map $\tau : X \to X$ is said to be Σ-measurable if and only if, for every $\sigma \in \Sigma$, one has $\tau^{-1}\sigma \in \Sigma$. A map τ is said to **preserve the measure** if and only if, for every $\sigma \in \Sigma$, one has $\mu(\tau^{-1}\sigma) = \mu(\sigma)$. Combining the above, a map τ is said to be a **measure-preserving transformation of *X***, if it is a map from X to itself, it is Σ-measurable, and is measure-preserving. The quadruple (X,Σ,μ,τ), for such a τ, is then defined to be a **dynamical system**.

The map τ embodies the time evolution of the dynamical system. Thus, for discrete dynamical systems the iterates $\tau^n = \tau \circ \tau \circ \ldots \circ \tau$ for integer n are studied. For continuous dynamical systems, the map τ is understood to be a finite time evolution map and the construction is more complicated.

Examples of dynamical systems

Internal links

- Arnold's cat map
- Baker's map is an example of a chaotic piecewise linear map
- Circle map
- Double pendulum
- Billiards and Outer Billiards
- Henon map
- Horseshoe map
- Irrational rotation
- List of chaotic maps
- Logistic map
- Lorenz system
- Rossler map

Chapter- 7

Failure Rate

Failure rate is the frequency with which an engineered system or component fails, expressed for example in failures per hour. It is often denoted by the Greek letter λ (lambda) and is important in reliability engineering.

The failure rate of a system usually depends on time, with the rate varying over the life cycle of the system. For example, an automobile's failure rate in its fifth year of service may be many times greater than its failure rate during its first year of service. One does not expect to replace an exhaust pipe, overhaul the brakes, or have major transmission problems in a new vehicle.

In practice, the mean time between failures (MTBF, $1/\lambda$) is often used instead of the failure rate. The MTBF is an important system parameter in systems where failure rate needs to be managed, in particular for safety systems. The MTBF appears frequently in the engineering design requirements, and governs frequency of required system maintenance and inspections. In special processes called renewal processes, where the time to recover from failure can be neglected and the likelihood of failure remains constant with respect to time, the failure rate is simply the multiplicative inverse of the MTBF ($1/\lambda$).

A similar ratio used in the transport industries, especially in railways and trucking is '**mean distance between failures**', a variation which attempts to correlate actual loaded distances to similar reliability needs and practices.

Failure rates are important factors in the insurance, finance, commerce and regulatory industries and fundamental to the design of safe systems in a wide variety of applications.

Failure rate in the discrete sense

The failure rate can be defined as the following:

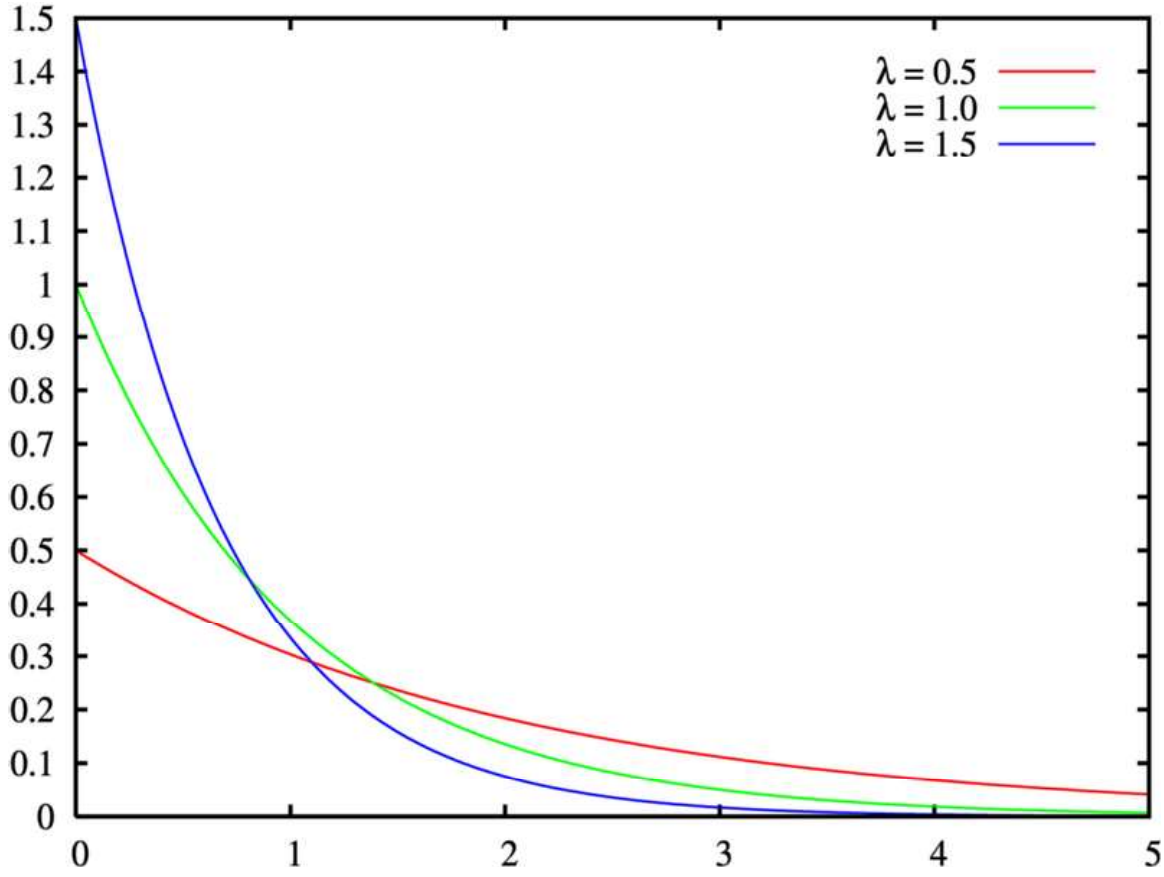
> The total number of failures within an item population, divided by the total time expended by that population, during a particular measurement interval under stated conditions. (MacDiarmid, *et al.*)

Although the failure rate, $\lambda(t)$, is often thought of as the probability that a failure occurs in a specified interval given no failure before time t, it is not actually a probability because it can exceed 1. It can be defined with the aid of the reliability function or survival function $R(t)$, the probability of no failure before time t, as:

$$\lambda = \frac{R(t_1) - R(t_2)}{(t_2 - t_1) \cdot R(t_1)} = \frac{R(t) - R(t + \triangle t)}{\triangle t \cdot R(t)}$$

over a time interval $(t_2 - t_1)$ from t_1 (or t) to t_2 and Δt is defined as $(t_2 - t_1)$. Note that this is a conditional probability, hence the $R(t)$ in the denominator.

Failure rate in the continuous sense



Exponential failure density functions

Calculating the failure rate for ever smaller intervals of time, results in the **hazard function** (or **hazard rate**), $h(t)$. This becomes the *instantaneous* failure rate as Δttends to zero:

$$h(t) = \lim_{\triangle t \to 0} \frac{R(t) - R(t + \triangle t)}{\triangle t \cdot R(t)}.$$

A continuous failure rate depends on the existence of a **failure distribution**, $F(t)$, which is a cumulative distribution function that describes the probability of failure (at least) up to and including time *t*,

$$\Pr(T \leq t) = F(t) = 1 - R(t), \quad t \geq 0.$$

where *T* is the failure time. The failure distribution function is the integral of the failure *density* function, *f*(*t*),

$$F(t) = \int_0^t f(x)\,dx.$$

The hazard function can be defined now as

$$h(t) = \frac{f(t)}{R(t)}.$$

Many probability distributions can be used to model the failure distribution. A common model is the **exponential failure distribution**,

$$F(t) = \int_0^t \lambda e^{-\lambda x}\,dx = 1 - e^{-\lambda t},$$

which is based on the exponential density function. The hazard rate function for this is:

$$h(t) = \frac{f(t)}{R(t)} = \frac{\lambda e^{-\lambda t}}{e^{-\lambda t}} = \lambda.$$

Thus, for an exponential failure distribution, the hazard rate is a constant with respect to time (that is, the distribution is "memoryless"). For other distributions, such as a Weibull distribution or a log-normal distribution, the hazard function may not be constant with respect to time. For some such as the deterministic distribution it is monotonic increasing (analogous to "wearing out"), for others such as the Pareto distribution it is monotonic decreasing (analogous to "burning in"), while for many it is not monotonic.

Failure rate data

Failure rate data can be obtained in several ways. The most common means are:

- Historical data about the device or system under consideration.

  Many organizations maintain internal databases of failure information on the devices or systems that they produce, which can be used to calculate failure rates

for those devices or systems. For new devices or systems, the historical data for similar devices or systems can serve as a useful estimate.

- Government and commercial failure rate data.

  Handbooks of failure rate data for various components are available from government and commercial sources. MIL-HDBK-217F, *Reliability Prediction of Electronic Equipment*, is a military standard that provides failure rate data for many military electronic components. Several failure rate data sources are available commercially that focus on commercial components, including some non-electronic components.

- Testing.

  The most accurate source of data is to test samples of the actual devices or systems in order to generate failure data. This is often prohibitively expensive or impractical, so that the previous data sources are often used instead.

Units

Failure rates can be expressed using any measure of time, but **hours** is the most common unit in practice. Other units, such as miles, revolutions, etc., can also be used in place of "time" units.

Failure rates are often expressed in engineering notation as failures per million, or 10^{-6}, especially for individual components, since their failure rates are often very low.

The **Failures In Time** (**FIT**) rate of a device is the number of failures that can be expected in one billion (10^9) device-hours of operation. (E.g. 1000 devices for 1 million hours, or 1 million devices for 1000 hours each, or some other combination.) This term is used particularly by the semiconductor industry.

Additivity

Under certain engineering assumptions (e.g. besides the above assumptions for a constant failure rate, the assumption that the considered system has no relevant redundancies), the failure rate for a complex system is simply the sum of the individual failure rates of its components, as long as the units are consistent, e.g. failures per million hours. This permits testing of individual components or subsystems, whose failure rates are then added to obtain the total system failure rate.

Example

Suppose it is desired to estimate the failure rate of a certain component. A test can be performed to estimate its failure rate. Ten identical components are each tested until they either fail or reach 1000 hours, at which time the test is terminated for that component.

(The level of statistical confidence is not considered in this example.) The results are as follows:

Estimated failure rate is

$$\frac{6 \text{ failures}}{7502 \text{ hours}} = 0.0007998 \frac{\text{failures}}{\text{hour}} = 799.8 \times 10^{-6} \frac{\text{failures}}{\text{hour}},$$

or 799.8 failures for every million hours of operation.

Estimation

The Nelson–Aalen estimator can be used to estimate the cumulative hazard rate function.

Chapter- 8

Safety Engineering

Safety engineering is an applied science strongly related to systems engineering and the subset System Safety Engineering. Safety engineering assures that a life-critical system behaves as needed even when pieces fail.

Overview

Ideally, safety-engineers take an early design of a system, analyze it to find what faults can occur, and then propose safety requirements in design specifications up front and changes to existing systems to make the system safer. In an early design stage, often a fail-safe system can be made acceptably safe with a few sensors and some software to read them. Probabilistic fault-tolerant systems can often be made by using more, but smaller and less-expensive pieces of equipment.

Far too often, rather than actually influencing the design, safety engineers are assigned to prove that an existing, completed design is safe. If a safety engineer then discovers significant safety problems late in the design process, correcting them can be very expensive. This type of error has the potential to waste large sums of money.

The exception to this conventional approach is the way some large government agencies approach safety engineering from a more proactive and proven process perspective, known as "system safety". The system safety philosophy is to be applied to complex and critical systems, such as commercial airliners, complex weapon systems, spacecraft, rail and transportation systems, air traffic control system and other complex and safety-critical industrial systems. The proven system safety methods and techniques are to prevent, eliminate and control hazards and risks through designed influences by a collaboration of key engineering disciplines and product teams. Software safety is a fast growing field since modern systems functionality are increasingly being put under control of software. The whole concept of system safety and software safety, as a subset of systems engineering, is to influence safety-critical systems designs by conducting several types of hazard analyses to identify risks and to specify design safety features and procedures to strategically mitigate risk to acceptable levels before the system is certified.

Additionally, failure mitigation can go beyond design recommendations, particularly in the area of maintenance. There is an entire realm of safety and reliability engineering known as Reliability Centered Maintenance (RCM), which is a discipline that is a direct

result of analyzing potential failures within a system and determining maintenance actions that can mitigate the risk of failure. This methodology is used extensively on aircraft and involves understanding the failure modes of the serviceable replaceable assemblies in addition to the means to detect or predict an impending failure. Every automobile owner is familiar with this concept when they take in their car to have the oil changed or brakes checked. Even filling up one's car with fuel is a simple example of a failure mode (failure due to fuel exhaustion), a means of detection (fuel gauge), and a maintenance action (filling the car's fuel tank).

For large scale complex systems, hundreds if not thousands of maintenance actions can result from the failure analysis. These maintenance actions are based on conditions (e.g., gauge reading or leaky valve), hard conditions (e.g., a component is known to fail after 100 hrs of operation with 95% certainty), or require inspection to determine the maintenance action (e.g., metal fatigue). The RCM concept then analyzes each individual maintenance item for its risk contribution to safety, mission, operational readiness, or cost to repair if a failure does occur. Then the sum total of all the maintenance actions are bundled into maintenance intervals so that maintenance is not occurring around the clock, but rather, at regular intervals. This bundling process introduces further complexity, as it might stretch some maintenance cycles, thereby increasing risk, but reduce others, thereby potentially reducing risk, with the end result being a comprehensive maintenance schedule, purpose built to reduce operational risk and ensure acceptable levels of operational readiness and availability.

Analysis techniques

Analysis techniques can be split into two categories: qualitative and quantitative methods. The both approaches share the goal of finding causal dependencies between an hazard on system level and failures of individual components. Qualitative approaches focus on the question "What must go wrong, such that a system hazard may occur?", while quantitative methods aim at providing estimations about probabilites, rates and/or severity of consequences.

Traditionally, safety analysis techniques rely solely on skill and expertise of the safety engineer. In the last decade model-based approaches have become prominent. In contrast to traditional methods, model-based techniques try to derive relationships between causes and consequences from some sort of model of the system.

Traditional methods for safety analysis

The two most common fault modeling techniques are called failure mode and effects analysis and fault tree analysis. These techniques are just ways of finding problems and of making plans to cope with failures, as in probabilistic risk assessment. One of the earliest complete studies using this technique on a commercial nuclear plant was the WASH-1400 study, also known as the Reactor Safety Study or the Rasmussen Report.

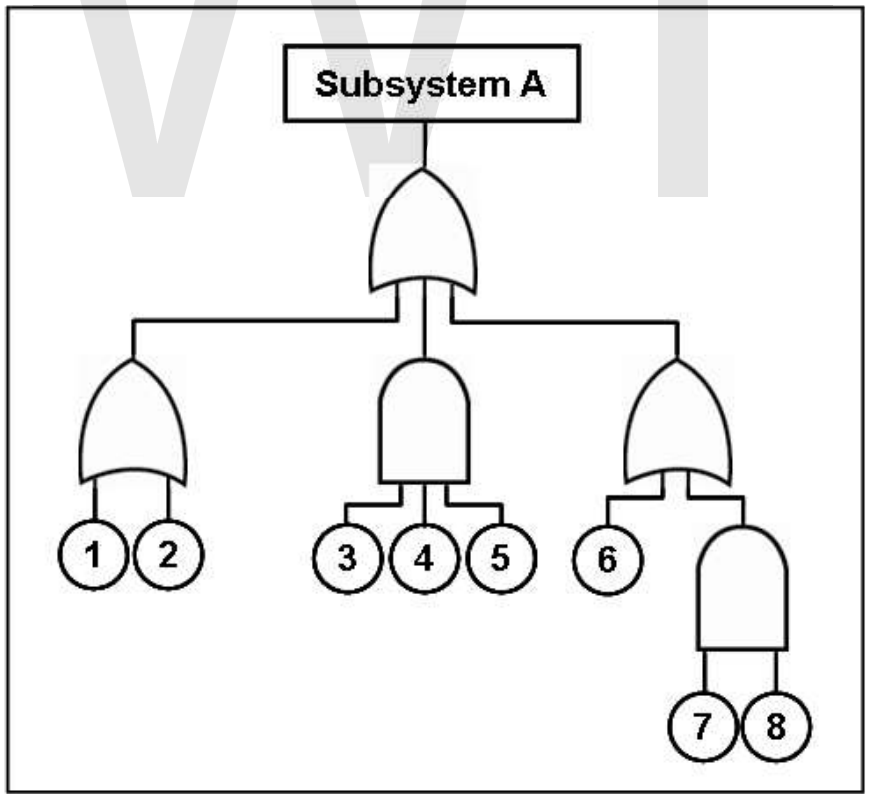
Failure modes and effects analysis

Failure Mode and Effects Analysis (FMEA) is a bottom-up, inductive analytical method which may be performed at either the functional or piece-part level. For functional FMEA, failure modes are identified for each function in a system or equipment item, usually with the help of a functional block diagram. For piece-part FMEA, failure modes are identified for each piece-part component (such as a valve, connector, resistor, or diode). The effects of the failure mode are described, and assigned a probability based on the failure rate and failure mode ratio of the function or component.

Failure modes with identical effects can be combined and summarized in a Failure Mode Effects Summary. When combined with criticality analysis, FMEA is known as Failure Mode, Effects, and Criticality Analysis or FMECA, pronounced "fuh-MEE-kuh".

Fault tree analysis

Fault tree analysis (FTA) is a top-down, deductive analytical method. In FTA, initiating primary events such as component failures, human errors, and external events are traced through Boolean logic gates to an undesired top event such as an aircraft crash or nuclear reactor core melt. The intent is to identify ways to make top events less probable, and verify that safety goals have been achieved.



A fault tree diagram

Fault trees are a logical inverse of success trees, and may be obtained by applying de Morgan's theorem to success trees (which are directly related to reliability block diagrams).

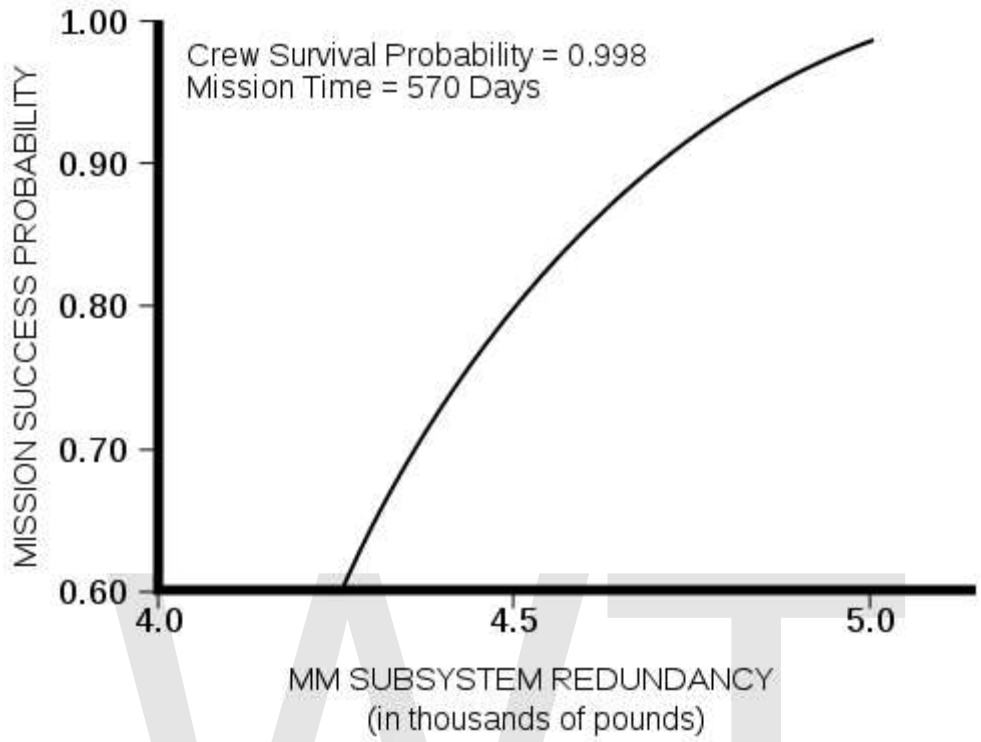
FTA may be qualitative or quantative. When failure and event probabilites are unknown, qualitative fault trees may be analyzed for minimal cut sets. For example, if any minimal cut set contains a single base event, then the top event may be caused by a single failure. Quantitative FTA is used to compute top event probability, and usually requires computer software such as CAFTA from the Electric Power Research Institute or SAPHIRE from the Idaho National Laboratory.

Some industries use both fault trees and event trees. An event tree starts from an undesired initiator (loss of critical supply, component failure etc.) and follows possible further system events through to a series of final consequences. As each new event is considered, a new node on the tree is added with a split of probabilities of taking either branch. The probabilities of a range of "top events" arising from the initial event can then be seen.

Safety certification

Usually a failure in safety-certified systems is acceptable if, on average, less than one life per 10^9 hours of continuous operation is lost to failure. Most Western nuclear reactors, medical equipment, and commercial aircraft are certified to this level. The cost versus loss of lives has been considered appropriate at this level (by FAA for aircraft under Federal Aviation Regulations).

Preventing failure



A NASA graph shows the relationship between the survival of a crew of astronauts and the amount of redundant equipment in their spacecraft (the "MM", Mission Module).

Probabilistic fault tolerance: adding redundancy to equipment and systems

Once a failure mode is identified, it can usually be prevented entirely by adding extra equipment to the system. For example, nuclear reactors contain dangerous radiation, and nuclear reactions can cause so much heat that no substance might contain them. Therefore reactors have emergency core cooling systems to keep the temperature down, shielding to contain the radiation, and engineered barriers (usually several, nested, surmounted by a containment building) to prevent accidental leakage.

Most biological organisms have a certain amount of redundancy: multiple organs, multiple limbs, etc.

For any given failure, a fail-over or redundancy can almost always be designed and incorporated into a system.

When does safety stop, where does reliability begin?

Inherent fail-safe design

When adding equipment is impractical (usually because of expense), then the least expensive form of design is often "inherently fail-safe". The typical approach is to arrange the system so that ordinary single failures cause the mechanism to shut down in a safe way (for nuclear power plants, this is termed a passively safe design, although more than ordinary failures are covered).

One of the most common fail-safe systems is the overflow tube in baths and kitchen sinks. If the valve sticks open, rather than causing an overflow and damage, the tank spills into an overflow.

Another common example is that in an elevator the cable supporting the car keeps spring-loaded brakes open. If the cable breaks, the brakes grab rails, and the elevator cabin does not fall.

Inherent fail-safes are common in medical equipment, traffic and railway signals, communications equipment, and safety equipment.

Containing failure

It is also common practice to plan for the failure of safety systems through containment and isolation methods. The use of isolating valves, also known as the block and bleed manifold, is very common in isolating pumps, tanks, and control valves that may fail or need routine maintenance. In addition, nearly all tanks containing oil or other hazardous chemicals are required to have containment barriers set up around them to contain 100% of the volume of the tank in the event of a catastrophic tank failure. Similarly, in a long pipeline, there are remote-closing valves at regular intervals so that a leak can be isolated. The goal of all containment systems is to provide means of mitigating the consequences of failure.

Chapter- 9

Statistical Process Control

Statistical process control (SPC) is the application of statistical methods to the monitoring and control of a process to ensure that it operates at its full potential to produce conforming product. Under SPC, a process behaves predictably to produce as much conforming product as possible with the least possible waste. While SPC has been applied most frequently to controlling manufacturing lines, it applies equally well to any process with a measurable output. Key tools in SPC are control charts, a focus on continuous improvement and designed experiments.

Much of the power of SPC lies in the ability to examine a process and the sources of variation in that process using tools that give weight to objective analysis over subjective opinions and that allow the strength of each source to be determined numerically. Variations in the process that may affect the quality of the end product or service can be detected and corrected, thus reducing waste as well as the likelihood that problems will be passed on to the customer. With its emphasis on early detection and prevention of problems, SPC has a distinct advantage over other quality methods, such as inspection, that apply resources to detecting and correcting problems after they have occurred.

In addition to reducing waste, SPC can lead to a reduction in the time required to produce the product or service from end to end. This is partially due to a diminished likelihood that the final product will have to be reworked, but it may also result from using SPC data to identify bottlenecks, wait times, and other sources of delays within the process. Process cycle time reductions coupled with improvements in yield have made SPC a valuable tool from both a cost reduction and a customer satisfaction standpoint.

History

Statistical process control was pioneered by Walter A. Shewhart in the early 1920s. W. Edwards Deming later applied SPC methods in the United States during World War II, thereby successfully improving quality in the manufacture of munitions and other strategically important products. Deming was also instrumental in introducing SPC methods to Japanese industry after the war had ended.

Shewhart created the basis for the control chart and the concept of a state of statistical control by carefully designed experiments. While Dr. Shewhart drew from pure mathematical statistical theories, he understood that data from physical processes seldom

produces a "normal distribution curve" (a Gaussian distribution, also commonly referred to as a "bell curve"). He discovered that observed variation in manufacturing data did not always behave the same way as data in nature (for example, Brownian motion of particles). Dr. Shewhart concluded that while every process displays variation, some processes display controlled variation that is natural to the process (common causes of variation), while others display uncontrolled variation that is not present in the process causal system at all times (special causes of variation).

In 1988, the Software Engineering Institute introduced the notion that SPC can be usefully applied to non-manufacturing processes, such as software engineering processes, in the Capability Maturity Model (CMM). This idea exists today within the Level 4 and Level 5 practices of the Capability Maturity Model Integration (CMMI). This notion that SPC is a useful tool when applied to non-repetitive, knowledge-intensive processes such as engineering processes has encountered much skepticism, and remains controversial today.

General

The following description relates to manufacturing rather than to the service industry, although the principles of SPC can be successfully applied to either. For a description and example of how SPC applies to a service environment, refer to Roberts (2005). SPC has also been successfully applied to detecting changes in organizational behavior with Social Network Change Detection introduced by McCulloh (2007). Selden describes how to use SPC in the fields of sales, marketing, and customer service, using Deming's famous Red Bead Experiment as an easy to follow demonstration.

In mass-manufacturing, the quality of the finished article was traditionally achieved through post-manufacturing inspection of the product; accepting or rejecting each article (or samples from a production lot) based on how well it met its design specifications. In contrast, Statistical Process Control uses statistical tools to observe the performance of the production process in order to predict significant deviations that may later result in rejected product.

Two kinds of variation occur in all manufacturing processes: both these types of process variation cause subsequent variation in the final product. The first is known as natural or common cause variation and consists of the variation inherent in the process as it is designed. Common cause variation may include variations in temperature, properties of raw materials, strength of an electrical current etc. The second kind of variation is known as special cause variation, or assignable-cause variation, and happens less frequently than the first. With sufficient investigation, a specific cause, such as abnormal raw material or incorrect set-up parameters, can be found for special cause variations.

For example, a breakfast cereal packaging line may be designed to fill each cereal box with 500 grams of product, but some boxes will have slightly more than 500 grams, and some will have slightly less, in accordance with a distribution of net weights. If the production process, its inputs, or its environment changes (for example, the machines

doing the manufacture begin to wear) this distribution can change. For example, as its cams and pulleys wear out, the cereal filling machine may start putting more cereal into each box than specified. If this change is allowed to continue unchecked, more and more product will be produced that fall outside the tolerances of the manufacturer or consumer, resulting in waste. While in this case, the waste is in the form of "free" product for the consumer, typically waste consists of rework or scrap.

By observing at the right time what happened in the process that led to a change, the quality engineer or any member of the team responsible for the production line can troubleshoot the root cause of the variation that has crept in to the process and correct the problem.

SPC indicates when an action should be taken in a process, but it also indicates when NO action should be taken. An example is a person who would like to maintain a constant body weight and takes weight measurements weekly. A person who does not understand SPC concepts might start dieting every time his or her weight increased, or eat more every time his or her weight decreased. This type of action could be harmful and possibly generate even more variation in body weight. SPC would account for normal weight variation and better indicate when the person is in fact gaining or losing weight.

How to Use SPC

Statistical Process Control may be broadly broken down into three sets of activities: understanding the process, understanding the causes of variation, and elimination of the sources of special cause variation.

In understanding a process, the process is typically mapped out and the process is monitored using control charts. Control charts are used to identify variation that may be due to special causes, and to free the user from concern over variation due to common causes. This is a continuous, ongoing activity. When a process is stable and does not trigger any of the detection rules for a control chart, a process capability analysis may also be performed to predict the ability of the current process to produce conforming (i.e. within specification) product in the future.

When excessive variation is identified by the control chart detection rules, or the process capability is found lacking, additional effort is exerted to determine causes of that variance. The tools used include Ishikawa diagrams, designed experiments and Pareto charts. Designed experiments are critical to this phase of SPC, as they are the only means of objectively quantifying the relative importance of the many potential causes of variation.

Once the causes of variation have been quantified, effort is spent in eliminating those causes that are both statistically and practically significant (i.e. a cause that has only a small but statistically significant effect may not be considered cost-effective to fix; however, a cause that is not statistically significant can never be considered practically significant). Generally, this includes development of standard work, error-proofing and

training. Additional process changes may be required to reduce variation or align the process with the desired target, especially if there is a problem with process capability.

For digital SPC charts, so-called SPC rules usually come with some rule specific logic that determines a 'derived value' that is to be used as the basis for some (setting) correction. One example of such a derived value would be (for the common N numbers in a row ranging up or down 'rule'); derived value = last value + average difference between the last N numbers (which would, in effect, be extending the row with the to be expected next value).

Chapter- 10

Optimal Design

Optimal designs are a class of experimental designs that are optimal with respect to some statistical criterion.

In the design of experiments for estimating statistical models, **optimal designs** allow parameters to be estimated without bias and with minimum-variance. A non-optimal design requires a greater number of experimental runs to estimate the parameters with the same precision as an optimal design. In practical terms, optimal experiments can reduce the costs of experimentation.

The optimality of a design depends on the statistical model and is assessed with respect to a statistical criterion, which is related to the variance-matrix of the estimator. Specifying an appropriate model and specifying a suitable criterion function both require understanding of statistical theory and practical knowledge with designing experiments.

Optimal designs are also called **optimum designs**.

Advantages of optimal designs

Optimal designs offer three advantages over suboptimal experimental designs:

1. Optimal designs reduce the costs of experimentation by allowing statistical models to be estimated with fewer experimental runs.
2. Optimal designs can accommodate multiple types of factors, such as process, mixture, and discrete factors.
3. Designs can be optimized when the design-space is constrained, for example, when the mathematical process-space contains factor-settings that are practically infeasible (e.g. due to safety concerns).

Minimizing the variance of estimators

Experimental designs are evaluated using statistical criteria.

It is known that the least squares estimator minimizes the variance of mean-unbiased estimators (under the conditions of the Gauss–Markov theorem). In the estimation theory for statistical models with one real parameter, the reciprocal of the variance of an

("efficient") estimator is called the "Fisher information" for that estimator. Because of this reciprocity, ***minimizing*** **the** ***variance*** corresponds to ***maximizing*** **the** ***information***.

When the statistical model has several parameters, however, the mean of the parameter-estimator is a vector and its variance is a matrix. The inverse matrix of the variance-matrix is called the "information matrix". Because the variance of the estimator of a parameter vector is a matrix, the problem of "minimizing the variance" is complicated. Using statistical theory, statisticians compress the information-matrix using real-valued summary statistics; being real-valued functions, these "information criteria" can be maximized. The traditional optimality-criteria are invariants of the information matrix; algebraically, the traditional optimality-criteria are functionals of the eigenvalues of the information matrix.

- **A**-optimality ("**average**" or **trace**)
  - One criterion is **A-optimality**, which seeks to minimize the trace of the inverse of the information matrix. This criterion results in minimizing the average variance of the estimates of the regression coefficients.

- **C**-optimality

- **D**-optimality (**determinant**)
  - A popular criterion is **D-optimality**, which seeks to minimize $|(X'X)^{-1}|$, or equivalently maximize the determinant of the information matrix X'X of the design. This criterion results in maximizing the differential Shannon information content of the parameter estimates.

- **E**-optimality (**eigenvalue**)
  - Another design is **E-optimality**, which maximizes the minimum eigenvalue of the information matrix. The E-optimality criterion need not be differentiable at every point. Such E-optimal designs can be computed using methods of convex minimization that use *subgradients* rather than gradients at points of non-differentiability. Any non-differentiability need not be a serious problem, however: E-optimality problems are special cases of semidefinite-programming problems which have effective solution-methods, especially bundle methods and interior-point methods.

- **T**-optimality
  - This criterion maximizes the trace of the information matrix.

Other optimality-criteria are concerned with the variance of predictions:

- **G**-optimality
  - A popular criterion is **G-optimality**, which seeks to minimize the maximum entry in the diagonal of the hat matrix $X(X'X)^{-1}X'$. This has the effect of minimizing the maximum variance of the predicted values.

- **I**-optimality (**integrated**)
    - A second criterion on prediction variance is **I-optimality**, which seeks to minimize the average prediction variance *over the design space*.

- **V**-optimality (**variance**)
    - A third criterion on prediction variance is **V-optimality**, which seeks to minimizes the average prediction variance over a set of m specific points.

Contrasts

In many applications, the statistician is most concerned with a "parameter of interest" rather than with "nuisance parameters". More generally, statisticians consider linear combinations of parameters, which are estimated via linear combinations of treatment-means in the design of experiments and in the analysis of variance; such linear combinations are called contrasts. Statisticians can use appropriate optimality-criteria for such parameters of interest and for more generally for contrasts.

Finding optimal designs

Catalogs of optimal designs occur in books and in software libraries.

In addition, major statistical systems like SAS and R have procedures for optimizing a design according to a user's specification. The experimenter must specify a model for the design and an optimality-criterion before the method can compute an optimal design.

Practical considerations

Some advanced topics in optimal design require more statistical theory and practical knowledge in designing experiments.

Model dependence and robustness

Since the optimality criterion of most optimal designs is based on some function of the information matrix, the 'optimality' of a given design is *model dependent*: While an optimal design is best for that model, its performance may deteriorate on other models. On other models, an *optimal* design can be either better or worse than a non-optimal design. Therefore, it is important to benchmark the performance of designs under alternative models.

Choosing an optimality criterion and robustness

The choice of an appropriate optimality criterion requires some thought, and it is useful to benchmark the performance of designs with respect to several optimality criteria. Cornell writes that

since the [traditional optimality] criteria . . . are variance-minimizing criteria, . . . a design that is optimal for a given model using one of the . . . criteria is usually near-optimal for the same model with respect to the other criteria.

Indeed, there are several classes of designs for which all the traditional optimality-criteria agree, according to the theory of "universal optimality" of Kiefer. The experience of practitioners like Cornell and the "universal optimality" theory of Kiefer suggest that robustness with respect to changes in the *optimality-criterion* is much greater than is robustness with respect to changes in the *model*.

Flexible optimality criteria and convex analysis

High-quality statistical software provide a combination of libraries of optimal designs or iterative methods for constructing approximately optimal designs, depending on the model specified and the optimality criterion. Users may use a standard optimality-criterion or may program a custom-made criterion.

All of the traditional optimality-criteria are convex (or concave) functions, and therefore optimal-designs are amenable to the mathematical theory of convex analysis and their computation can use specialized methods of convex minimization. The practitioner need not select *exactly one* traditional, optimality-criterion, but can specify a custom criterion. In particular, the practitioner can specify a convex criterion using the maxima of convex optimality-criteria and nonnegative combinations of optimality criteria (since these operations preserve convex functions). For *convex* optimality criteria, the Kiefer-Wolfowitz equivalence theorem allows the practitioner to verify that a given design is globally optimal. The Kiefer-Wolfowitz equivalence theorem is related with the Legendre-Fenchel conjugacy for convex functions.

If an optimality-criterion lacks convexity, then finding a global optimum and verifying its optimality often are difficult.

Model uncertainty and Bayesian approaches

Model selection

When scientists wish to test several theories, then a statistician can design an experiment that allows optimal tests between specified models. Such "discrimination experiments" are especially important in the biostatistics supporting pharmacokinetics and pharmacodynamics, following the work of Cox and Atkinson.

Bayesian experimental design

When practitioners need to consider multiple models, they can specify a probability-measure on the models and then select any design maximizing the expected value of such an experiment. Such probability-based optimal-designs are called optimal Bayesian

designs. Such Bayesian designs are used especially for generalized linear models (where the response follows an exponential-family distribution).

The use of a Bayesian design does not force statisticians to use Bayesian methods to analyze the data, however. Indeed, the "Bayesian" label for probability-based experimental-designs is disliked by some researchers. Alternative terminology for "Bayesian" optimality includes "on-average" optimality or "population" optimality.

Iterative experimentation

Scientific experimentation is an iterative process, and statisticians have developed several approaches to the optimal design of sequential experiments.

Sequential analysis

Sequential analysis was pioneered by Abraham Wald. In 1972, Herman Chernoff wrote an overview of optimal sequential designs, while adaptive designs were surveyed later by S. Zacks. Of course, much work on the optimal design of experiments is related to the theory of optimal decisions, especially the statistical decision theory of Abraham Wald.

Response-surface methodology

Optimal designs for response-surface models are discussed in the textbook by Atkinson, Donev and Tobias, and in the survey of Gaffke and Heiligers and in the mathematical text of Pukelsheim. The blocking of optimal designs is discussed in the textbook of Atkinson, Donev and Tobias and also in the monograph by Goos.

The earliest optimal designs were developed to estimate the parameters of regression models with continuous variables, for example, by J. D. Gergonne in 1815 (Stigler). In English, two early contributions were made by Charles S. Peirce and Kirstine Smith.

Pioneering designs for multivariate response-surfaces were proposed by George E. P. Box. However, Box's designs have few optimality properties. Indeed, the Box-Behnken design requires excessive experimental runs when the number of variables exceeds three. Box's "central-composite" designs require more experimental runs than do the optimal designs of Kôno.

System identification and stochastic approximation

The optimization of sequential experimentation is studied also in stochastic programming and in systems and control. Popular methods include stochastic approximation and other methods of stochastic optimization. Much of this research has been associated with the subdiscipline of system identification. In computational optimal control, D. Judin & A. Nemirovskii and Boris Polyak has described methods that are more efficient than the (Armijo-style) step-size rules introduced by G. E. P. Box in response-surface methodology

Adaptive designs are used in clinical trials, and optimal adaptive designs are surveyed in the *Handbook of Experimental Designs* chapter by Shelemyahu Zacks.

Specifying the number of experimental runs

Using a computer to find a good design

There are several methods of finding an optimal design, given an *a priori* restriction on the number of experimental runs or replications. Some of these methods are discussed by Atkinson, Donev and Tobias and in the paper by Hardin and Sloane. Of course, fixing the number of experimental runs *a priori* would be impractical. Prudent statisticians examine the other optimal designs, whose number of experimental runs differ.

Discretizing probability-measure designs

In the mathematical theory on optimal experiments, an optimal design can be a probability measure that is supported on an infinite set of observation-locations. Such optimal probability-measure designs solve a mathematical problem that neglected to specify the cost of observations and experimental runs. Nonetheless, such optimal probability-measure designs can be discretized to furnish approximately optimal designs.

In some cases, a finite set of observation-locations suffices to support an optimal design. Such a result was proved by Kôno and Kiefer in their works on response-surface designs for quadratic models. The Kôno-Kiefer analysis explains why optimal designs for response-surfaces can have discrete supports, which are very similar as do the less efficient designs that have been traditional in response surface methodology.

History

The prophet of scientific experimentation, Francis Bacon, foresaw that experimental designs should be improved. Researchers who improved experiments were praised in Bacon's utopian novel *New Atlantis*:

Then after divers meetings and consults of our whole number, to consider of the former labors and collections, we have three that take care out of them to direct new experiments, of a higher light, more penetrating into nature than the former. These we call lamps.

In 1815, an article on optimal designs for polynomial regression was published by Joseph Diaz Gergonne, according to Stigler.

Charles S. Peirce proposed an economic theory of scientific experimentation in 1876, which sought to maximize the precision of the estimates. Peirce's optimal allocation immediately improved the accuracy of gravitational experiments and was used for decades by Peirce and his colleagues. In his 1882 published lecture at Johns Hopkins University, Peirce introduced experimental design with these words:

Logic will not undertake to inform you what kind of experiments you ought to make in order best to determine the acceleration of gravity, or the value of the Ohm; but it will tell you how to proceed to form a plan of experimentation.

[....] Unfortunately practice generally precedes theory, and it is the usual fate of mankind to get things done in some boggling way first, and find out afterward how they could have been done much more easily and perfectly.

Like Bacon, Peirce was aware that experimental methods should strive for substantial improvement (even optimality).

Kirstine Smith proposed optimal designs for polynomial models in 1918. (Kirstine Smith had been a student of the Danish statistician Thorvald N. Thiele and was working with Karl Pearson in London.)

Chapter- 11

Introduction to Operations Research

Operational research, also known as **operations research**, is an interdisciplinary mathematical science that focuses on the effective *use* of technology by organizations. In contrast, many other science & engineering disciplines focus on technology giving secondary considerations to its use.

Employing techniques from other mathematical sciences --- such as mathematical modeling, statistical analysis, and mathematical optimization --- operations research arrives at optimal or near-optimal solutions to complex decision-making problems. Because of its emphasis on human-technology interaction and because of its focus on practical applications, operations research has overlap with other disciplines, notably industrial engineering and management science, and draws on psychology and organization science. Operations Research is often concerned with determining the maximum (of profit, performance, or yield) or minimum (of loss, risk, or cost) of some real-world objective. Originating in military efforts before World War II, its techniques have grown to concern problems in a variety of industries.

Overview

Operational research encompasses a wide range of problem-solving techniques and methods applied in the pursuit of improved decision-making and efficiency. Some of the tools used by operational researchers are statistics, optimization, probability theory, queuing theory, game theory, graph theory, decision analysis, mathematical modeling and simulation. Because of the computational nature of these fields, OR also has strong ties to computer science and analytics. Operational researchers faced with a new problem must determine which of these techniques are most appropriate given the nature of the system, the goals for improvement, and constraints on time and computing power.

Work in operational research and management science may be characterized as one of three categories:

- Fundamental or foundational work takes place in three mathematical disciplines: probability, optimization, and dynamical systems theory.
- Modeling work is concerned with the construction of models, analyzing them mathematically, implementing them on computers, solving them using software

tools, and assessing their effectiveness with data. This level is mainly instrumental, and driven mainly by statistics and econometrics.
- Application work in operational research, like other engineering and economics' disciplines, attempts to use models to make a practical impact on real-world problems.

The major subdisciplines in modern operational research, as identified by the journal *Operations Research*, are:

- Computing and information technologies
- Decision analysis
- Environment, energy, and natural resources
- Financial engineering
- Manufacturing, service sciences, and supply chain management
- Policy modeling and public sector work
- Revenue management
- Simulation
- Stochastic models
- Transportation

History

As a formal discipline, operational research originated in the efforts of military planners during World War II. In the decades after the war, the techniques began to be applied more widely to problems in business, industry and society. Since that time, operational research has expanded into a field widely used in industries ranging from petrochemicals to airlines, finance, logistics, and government, moving to a focus on the development of mathematical models that can be used to analyze and optimize complex systems, and has become an area of active academic and industrial research.

Historical origins

In the World War II era, operational research was defined as "a scientific method of providing executive departments with a quantitative basis for decisions regarding the operations under their control." Other names for it included operational analysis (UK Ministry of Defence from 1962) and quantitative management.

Prior to the formal start of the field, early work in operational research was carried out by individuals such as Charles Babbage. His research into the cost of transportation and sorting of mail led to England's universal "Penny Post" in 1840, and studies into the dynamical behaviour of railway vehicles in defence of the GWR's broad gauge. Percy Bridgman brought operational research to bear on problems in physics in the 1920s and would later attempt to extend these to the social sciences. The modern field of operational research arose during World War II.

Modern operational research originated at the Bawdsey Research Station in the UK in 1937 and was the result of an initiative of the station's superintendent, A. P. Rowe. Rowe conceived the idea as a means to analyse and improve the working of the UK's early warning radar system, Chain Home (CH). Initially, he analyzed the operating of the radar equipment and its communication networks, expanding later to include the operating personnel's behaviour. This revealed unappreciated limitations of the CH network and allowed remedial action to be taken.

Scientists in the United Kingdom including Patrick Blackett later Lord Blackett OM PRS, Cecil Gordon, C. H. Waddington, Owen Wansbrough-Jones, Frank Yates, Jacob Bronowski and Freeman Dyson, and in the United States with George Dantzig looked for ways to make better decisions in such areas as logistics and training schedules. After the war it began to be applied to similar problems in industry.

Second World War

Patrick Blackett

During the Second World War close to 1,000 men and women in Britain were engaged in operational research. About 200 operational research scientists worked for the British Army.

Patrick Blackett worked for several different organizations during the war. Early in the war while working for the Royal Aircraft Establishment (RAE) he set up a team known as the "Circus" which helped to reduce the number of anti-aircraft artillery rounds needed to shoot down an enemy aircraft from an average of over 20,000 at the start of the Battle of Britain to 4,000 in 1941.

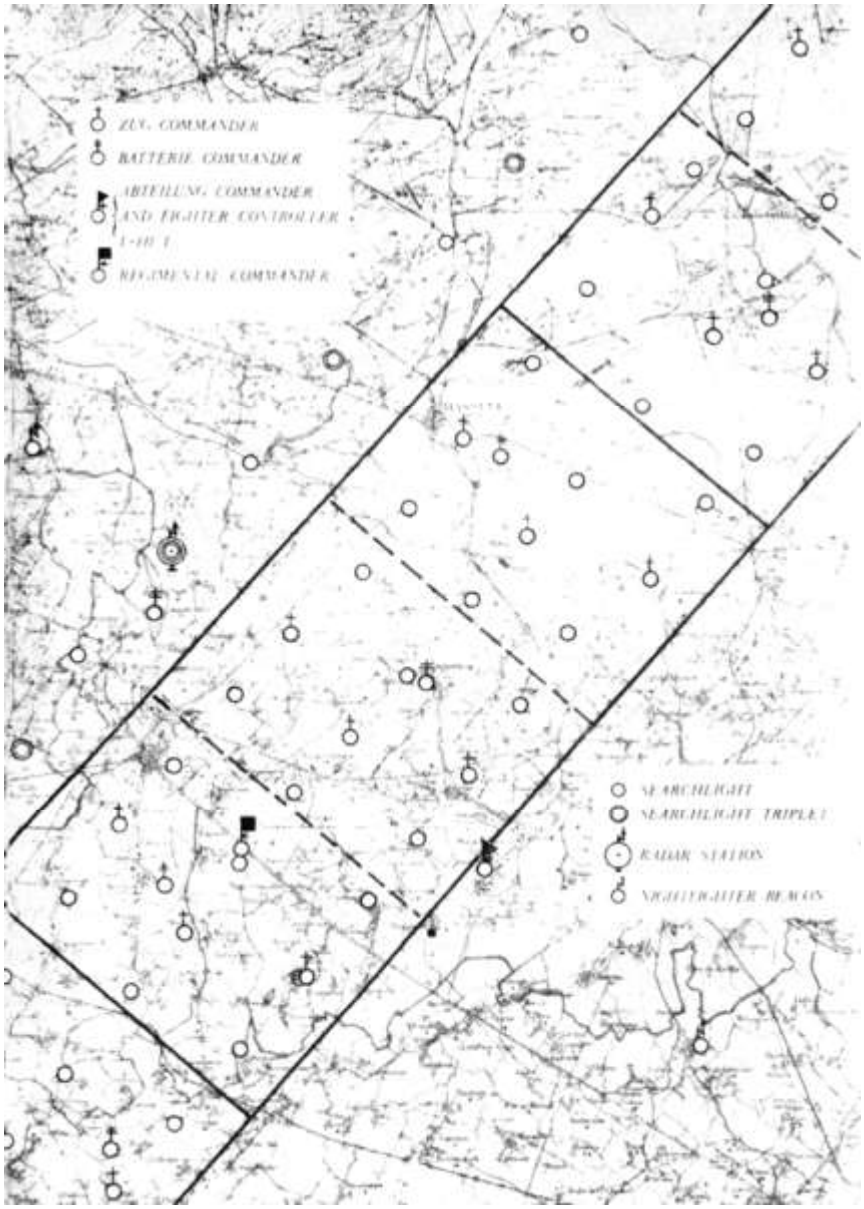
In 1941 Blackett moved from the RAE to the Navy, first to the Royal Navy's Coastal Command, in 1941 and then early in 1942 to the Admiralty. Blackett's team at Coastal Command's Operational Research Section (CC-ORS) included two future Nobel prize winners and many other people who went on to be preeminent in their fields. They undertook a number of crucial analyses that aided the war effort. Britain introduced the convoy system to reduce shipping losses, but while the principle of using warships to accompany merchant ships was generally accepted, it was unclear whether it was better for convoys to be small or large. Convoys travel at the speed of the slowest member, so

small convoys can travel faster. It was also argued that small convoys would be harder for German U-boats to detect. On the other hand, large convoys could deploy more warships against an attacker. Blackett's staff showed that the losses suffered by convoys depended largely on the number of escort vessels present, rather than on the overall size of the convoy. Their conclusion, therefore, was that a few large convoys are more defensible than many small ones.

While performing an analysis of the methods used by RAF Coastal Command to hunt and destroy submarines, one of the analysts asked what colour the aircraft were. As most of them were from Bomber Command they were painted black for nighttime operations. At the suggestion of CC-ORS a test was run to see if that was the best colour to camouflage the aircraft for daytime operations in the grey North Atlantic skies. Tests showed that aircraft painted white were on average not spotted until they were 20% closer than those painted black. This change indicated that 30% more submarines would be attacked and sunk for the same number of sightings.

Other work by the CC-ORS indicated that on average if the trigger depth of aerial delivered depth charges (DCs) was changed from 100 feet to 25 feet, the kill ratios would go up. The reason was that if a U-boat saw an aircraft only shortly before it arrived over the target then at 100 feet the charges would do no damage (because the U-boat wouldn't have time to descend as far as 100 feet), and if it saw the aircraft a long way from the target it had time to alter course under water so the chances of it being within the 20 feet kill zone of the charges was small. It was more efficient to attack those submarines close to the surface when these targets' locations were better known than to attempt their destruction at greater depths when their positions could only be guessed. Before the change of settings from 100 feet to 25 feet, 1% of submerged U-boats were sunk and 14% damaged. After the change, 7% were sunk and 11% damaged. (If submarines were caught on the surface, even if attacked shortly after submerging, the numbers rose to 11% sunk and 15% damaged). Blackett observed "there can be few cases where such a great operational gain had been obtained by such a small and simple change of tactics".

Bomber Command's Operational Research Section (BC-ORS), analysed a report of a survey carried out by RAF Bomber Command. For the survey, Bomber Command inspected all bombers returning from bombing raids over Germany over a particular period. All damage inflicted by German air defences was noted and the recommendation was given that armour be added in the most heavily damaged areas. Their suggestion to remove some of the crew so that an aircraft loss would result in fewer personnel loss was rejected by RAF command. Blackett's team instead made the surprising and counter-intuitive recommendation that the armour be placed in the areas which were completely untouched by damage in the bombers which returned. They reasoned that the survey was biased, since it only included aircraft that returned to Britain. The untouched areas of returning aircraft were probably vital areas, which, if hit, would result in the loss of the aircraft.



Map of *Kammhuber Line*

When Germany organised its air defences into the Kammhuber Line, it was realised that if the RAF bombers were to fly in a bomber stream they could overwhelm the night fighters who flew in individual cells directed to their targets by ground controllers. It was then a matter of calculating the statistical loss from collisions against the statistical loss from night fighters to calculate how close the bombers should fly to minimise RAF losses.

The "exchange rate" ratio of output to input was a characteristic feature of operational research. By comparing the number of flying hours put in by Allied aircraft to the number of U-boat sightings in a given area, it was possible to redistribute aircraft to more productive patrol areas. Comparison of exchange rates established "effectiveness ratios"

useful in planning. The ratio of 60 mines laid per ship sunk was common to several campaigns: German mines in British ports, British mines on German routes, and United States mines in Japanese routes.

Operational research doubled the on-target bomb rate of B-29s bombing Japan from the Marianas Islands by increasing the training ratio from 4 to 10 percent of flying hours; revealed that wolf-packs of three United States submarines were the most effective number to enable all members of the pack to engage targets discovered on their individual patrol stations; revealed that glossy enamel paint was more effective camouflage for night fighters than traditional dull camouflage paint finish, and the smooth paint finish increased airspeed by reducing skin friction.

On land, the operational research sections of the Army Operational Research Group (AORG) of the Ministry of Supply (MoS) were landed in Normandy in 1944, and they followed British forces in the advance across Europe. They analysed, among other topics, the effectiveness of artillery, aerial bombing, and anti-tank shooting.

After World War II

With expanded techniques and growing awareness of the field at the close of the war, operational research was no longer limited to only operational, but was extended to encompass equipment procurement, training, logistics and infrastructure.

Academic Denis Bouyssou describes the historical development of operational research from the 1940s to the 1970s as follows. "The historical development of Operational Research (OR) is traditionally seen as the succession of several phases: the 'heroic times' of the Second World War, the 'Golden Age' between the fifties and the sixties during which major theoretical achievements were accompanied by a widespread diffusion of OR techniques in private and public organisations, a 'crisis' followed by a 'decline' starting with the late sixties, a phase during which OR groups in firms progressively disappeared while academia became less and less concerned with the applicability of the techniques developed".

Individuals such as Stafford Beer and George Dantzig pioneered early academic efforts in operational research.

Problems addressed with operational research

- critical path analysis or project planning: identifying those processes in a complex project which affect the overall duration of the project
- floorplanning: designing the layout of equipment in a factory or components on a computer chip to reduce manufacturing time (therefore reducing cost)
- network optimization: for instance, setup of telecommunications networks to maintain quality of service during outages
- allocation problems
- Bayesian search theory : looking for a target

- optimal search
- routing, such as determining the routes of buses so that as few buses are needed as possible
- supply chain management: managing the flow of raw materials and products based on uncertain demand for the finished products
- efficient messaging and customer response tactics
- automation: automating or integrating robotic systems in human-driven operations processes
- globalization: globalizing operations processes in order to take advantage of cheaper materials, labor, land or other productivity inputs
- transportation: managing freight transportation and delivery systems (Examples: LTL Shipping, intermodal freight transport)
- scheduling:
    - personnel staffing
    - manufacturing steps
    - project tasks
    - network data traffic: these are known as queueing models or queueing systems.
    - sports events and their television coverage
- blending of raw materials in oil refineries
- determining optimal prices, in many retail and B2B settings, within the disciplines of pricing science

Operational research is also used extensively in government where evidence-based policy is used.

Management science

In 1967 Stafford Beer characterized the field of management science as "the business use of operations research". However, in modern times the term management science may also be used to refer to the separate fields of organizational studies or corporate strategy. Like operational research itself, management science (MS), is an interdisciplinary branch of applied mathematics devoted to optimal decision planning, with strong links with economics, business, engineering, and other sciences. It uses various scientific research-based principles, strategies, and analytical methods including mathematical modeling, statistics and numerical algorithms to improve an organization's ability to enact rational and meaningful management decisions by arriving at optimal or near optimal solutions to complex decision problems. In short, management sciences help businesses to achieve their goals using the scientific methods of operational research.

The management scientist's mandate is to use rational, systematic, science-based techniques to inform and improve decisions of all kinds. Of course, the techniques of management science are not restricted to business applications but may be applied to military, medical, public administration, charitable groups, political groups or community groups.

Management science is concerned with developing and applying models and concepts that may prove useful in helping to illuminate management issues and solve managerial problems, as well as designing and developing new and better models of organizational excellence.

The application of these models within the corporate sector became known as Management science.

Techniques

Some of the fields that have considerable overlap with Management Science include:

- Data mining
- Decision analysis
- Engineering
- Forecasting
- Game theory
- Industrial engineering
- Logistics
- Mathematical modeling
- Optimization
- Probability and statistics
- Project management
- Simulation
- Social network/Transportation forecasting models
- Supply chain management
- Financial engineering

Applications of management science

Applications of management science are abundant in industry as airlines, manufacturing companies, service organizations, military branches, and in government. The range of problems and issues to which management science has contributed insights and solutions is vast. It includes:.

- scheduling airlines, including both planes and crew,
- deciding the appropriate place to site new facilities such as a warehouse, factory or fire station,
- managing the flow of water from reservoirs,
- identifying possible future development paths for parts of the telecommunications industry,
- establishing the information needs and appropriate systems to supply them within the health service, and
- identifying and understanding the strategies adopted by companies for their information systems

Management science is also concerned with so-called ”soft-operational analysis”, which concerns methods for strategic planning, strategic decision support, and Problem Structuring Methods (PSM). In dealing with these sorts of challenges mathematical modeling and simulation are not appropriate or will not suffice. Therefore, during the past 30 years, a number of non-quantified modelling methods have been developed. These include:

- stakeholder based approaches including metagame analysis and drama theory
- morphological analysis and various forms of influence diagrams.
- approaches using cognitive mapping
- the Strategic Choice Approach
- robustness analysis

Societies and journals

Societies

The International Federation of Operational Research Societies (IFORS) is an umbrella organization for operational research societies worldwide, representing approximately 50 national societies including those in the US, UK, France, Germany, Canada, Australia, New Zealand, Philippines, India, Japan and South Africa,. The constituent members of IFORS form regional groups, such as that in Europe,. Other important operational research organizations are Simulation Interoperability Standards Organization (SISO) and Interservice/Industry Training, Simulation and Education Conference (I/ITSEC)

In 2004 the US-based organization INFORMS began an initiative to market the OR profession better, including a website entitled *The Science of Better* which provides an introduction to OR and examples of successful applications of OR to industrial problems. This initiative has been adopted by the Operational Research Society in the UK, including a website entitled *Learn about OR*.

Journals

INFORMS publishes twelve scholarly journals about operations research, including the top two journals in their class, according to 2005 Journal Citation Reports. They are:

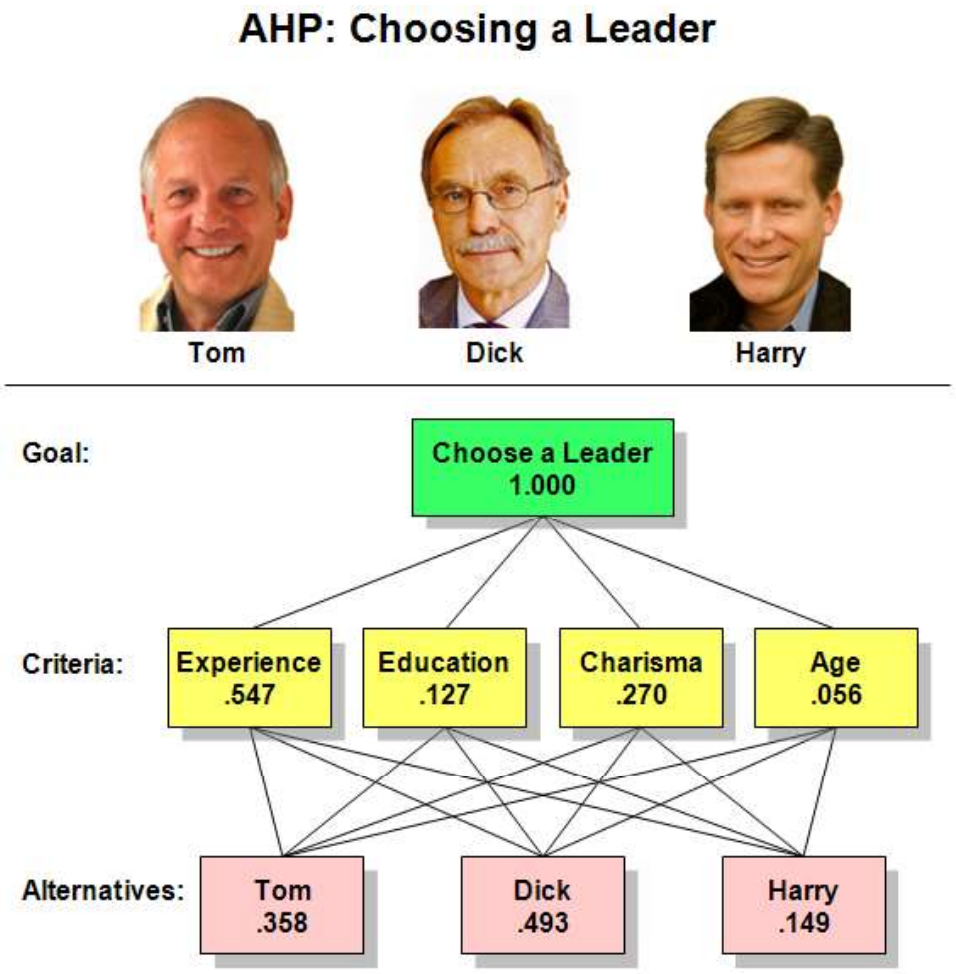
- *Decision Analysis*
- *Information Systems Research*
- *INFORMS Journal on Computing*
- *INFORMS Transactions on Education* (an open access journal)
- *Interfaces: An International Journal of the Institute for Operations Research and the Management Science*s
- *Management Science: A Journal of the Institute for Operations Research and the Management Sciences*
- *Manufacturing & Service Operations Management*
- *Marketing Science*
- *Mathematics of Operations Research*
- *Operations Research: A Journal of the Institute for Operations Research and the Management Sciences*
- *Organization Science*
- *Transportation Science*.

Other journals

- *European Journal of Operational Research (EJOR)*: Founded in 1975 and is presently by far the largest operational research journal in the world, with its around 9,000 pages of published papers per year. In 2004, its total number of citations was the second largest amongst Operational Research and Management Science journals;
- *INFOR Journal*: published and sponsored by the Canadian Operational Research Society;
- *International Journal of Operations Research and Information Systems (IJORIS)": an official publication of the Information Resources Management Association, published quarterly by IGI Global;*
- *Journal of Defense Modeling and Simulation (JDMS): Applications, Methodology, Technology*: a quarterly journal devoted to advancing the science of modeling and simulation as it relates to the military and defense.
- *Journal of the Operational Research Society (JORS)*: an official journal of The OR Society; this is the oldest continuously published journal of OR in the world, published by Palgrave;
- *Journal of Simulation (JOS)*: an official journal of The OR Society, published by Palgrave;
- *Mathematical Methods of Operations Research (MMOR)*: the journal of the German and Dutch OR Societies, published by Springer;
- *Military Operations Research (MOR)*: published by the Military Operations Research Society;
- *Opsearch*: official journal of the Operational Research Society of India;
- *OR Insight*: a quarterly journal of The OR Society, published by Palgrave;
- *TOP*: the official journal of the Spanish Society of Statistics and Operations Research.

Chapter- 12

Analytic Hierarchy Process



A simple AHP hierarchy, with final priorities. The decision goal is to select the most suitable leader from a field of three candidates. Factors to be considered are Experience, Education, Charisma, and Age. According to the judgments of the decision makers, Dick is the most suitable candidate, followed by Tom and Harry.

The **Analytic Hierarchy Process (AHP)** is a structured technique for dealing with complex decisions. Rather than prescribing a "correct" decision, the AHP helps decision

makers find one that best suits their goal and their understanding of the problem—it is a process of organizing decisions that people are already dealing with, but trying to do in their heads.

Based on mathematics and psychology, the AHP was developed by Thomas L. Saaty in the 1970s and has been extensively studied and refined since then. It provides a comprehensive and rational framework for structuring a decision problem, for representing and quantifying its elements, for relating those elements to overall goals, and for evaluating alternative solutions. It is used around the world in a wide variety of decision situations, in fields such as government, business, industry, healthcare, and education.

Several firms supply computer software to assist in using the process.

Users of the AHP first decompose their decision problem into a hierarchy of more easily comprehended sub-problems, each of which can be analyzed independently. The elements of the hierarchy can relate to any aspect of the decision problem—tangible or intangible, carefully measured or roughly estimated, well- or poorly-understood—anything at all that applies to the decision at hand.

Once the hierarchy is built, the decision makers systematically evaluate its various elements by comparing them to one another two at a time, with respect to their impact on an element above them in the hierarchy. In making the comparisons, the decision makers can use concrete data about the elements, or they can use their judgments about the elements' relative meaning and importance. It is the essence of the AHP that human judgments, and not just the underlying information, can be used in performing the evaluations.

The AHP converts these evaluations to numerical values that can be processed and compared over the entire range of the problem. A numerical weight or priority is derived for each element of the hierarchy, allowing diverse and often incommensurable elements to be compared to one another in a rational and consistent way. This capability distinguishes the AHP from other decision making techniques.

In the final step of the process, numerical priorities are calculated for each of the decision alternatives. These numbers represent the alternatives' relative ability to achieve the decision goal, so they allow a straightforward consideration of the various courses of action.

The diagram above shows a simple AHP hierarchy at the end of the decision making process. Numerical priorities, derived from the decision makers' input, are shown for each node in the hierarchy. In this decision, the goal was to choose the most suitable leader based on four specific criteria. Dick was the preferred alternative, with a priority of .493. He was preferred about a third more strongly than Tom, whose priority was .358, and about three times more strongly than Harry, whose priority was only .149. Experience was the most important criterion with respect to reaching the goal, followed

by Charisma, Education, and Age. These factors were weighted .547, .270, .127, and .056, respectively.

Uses and applications

While it can be used by individuals working on straightforward decisions, the Analytic Hierarchy Process (AHP) is most useful where teams of people are working on complex problems, especially those with high stakes, involving human perceptions and judgments, whose resolutions have long-term repercussions. It has unique advantages when important elements of the decision are difficult to quantify or compare, or where communication among team members is impeded by their different specializations, terminologies, or perspectives.

Decision situations to which the AHP can be applied include:

- Choice - The selection of one alternative from a given set of alternatives, usually where there are multiple decision criteria involved.
- Ranking - Putting a set of alternatives in order from most to least desirable
- Prioritization - Determining the relative merit of members of a set of alternatives, as opposed to selecting a single one or merely ranking them
- Resource allocation - Apportioning resources among a set of alternatives
- Benchmarking - Comparing the processes in one's own organization with those of other best-of-breed organizations
- Quality management - Dealing with the multidimensional aspects of quality and quality improvement

The applications of AHP to complex decision situations have numbered in the thousands, and have produced extensive results in problems involving planning, resource allocation, priority setting, and selection among alternatives. Other areas have included forecasting, total quality management, business process re-engineering, quality function deployment, and the Balanced Scorecard. Many AHP applications are never reported to the world at large, because they take place at high levels of large organizations where security and privacy considerations prohibit their disclosure. But some uses of AHP *are* discussed in the literature. Recently these have included:

- Deciding how best to reduce the impact of global climate change (*Fondazione Eni Enrico Mattei*)
- Quantifying the overall quality of software systems (*Microsoft Corporation*)
- Selecting university faculty (*Bloomsburg University of Pennsylvania*)
- Deciding where to locate offshore manufacturing plants (*University of Cambridge*)
- Assessing risk in operating cross-country petroleum pipelines (*American Society of Civil Engineers*)
- Deciding how best to manage U.S. watersheds (*U.S. Department of Agriculture*)

AHP is sometimes used in designing highly specific procedures for particular situations, such as the rating of buildings by historic significance. It was recently applied to a project that uses video footage to assess the condition of highways in Virginia. Highway engineers first used it to determine the optimum scope of the project, then to justify its budget to lawmakers.

Education and scholarly research

Though using the Analytic Hierarchy Process requires no specialized academic training, it is considered an important subject in many institutions of higher learning, including schools of engineering and graduate schools of business. It is a particularly important subject in the quality field, and is taught in many specialized courses including Six Sigma, Lean Six Sigma, and QFD.

Nearly a hundred Chinese universities offer courses in AHP, and many doctoral students choose AHP as the subject of their research and dissertations. Over 900 papers have been published on the subject in China, and there is at least one Chinese scholarly journal devoted exclusively to AHP.

The International Symposium on the Analytic Hierarchy Process (ISAHP) holds biennial meetings of academics and practitioners interested in the field. At its 2007 meeting in Valparaiso, Chile, over 90 papers were presented from 19 countries, including the U.S., Germany, Japan, Chile, Malaysia, and Nepal. Topics covered ranged from *Establishing Payment Standards for Surgical Specialists*, to *Strategic Technology Roadmapping*, to *Infrastructure Reconstruction in Devastated Countries*.

Using the Analytic Hierarchy Process

A typical device for entering judgments in an AHP group decision making session

As can be seen in the material that follows, using the AHP involves the mathematical synthesis of numerous judgments about the decision problem at hand. It is not uncommon for these judgments to number in the dozens or even the hundreds. While the math can be done by hand or with a calculator, it is far more common to use one of several computerized methods for entering and synthesizing the judgments. The simplest of these involve standard spreadsheet software, while the most complex use custom software, often augmented by special devices for acquiring the judgments of decision makers gathered in a meeting room.

The procedure for using the AHP can be summarized as:

1. Model the problem as a hierarchy containing the decision goal, the alternatives for reaching it, and the criteria for evaluating the alternatives.

2. Establish priorities among the elements of the hierarchy by making a series of judgments based on pairwise comparisons of the elements. For example, when comparing potential real-estate purchases, the investors might say they prefer location over price and price over timing.
3. Synthesize these judgments to yield a set of overall priorities for the hierarchy. This would combine the investors' judgments about location, price and timing for properties A, B, C, and D into overall priorities for each property.
4. Check the consistency of the judgments.
5. Come to a final decision based on the results of this process.

These steps are more fully described below.

Model the problem as a hierarchy

The first step in the Analytic Hierarchy Process is to model the problem as a *hierarchy*. In doing this, participants explore the aspects of the problem at levels from general to detailed, then express it in the multileveled way that the AHP requires. As they work to build the hierarchy, they increase their understanding of the problem, of its context, and of each other's thoughts and feelings about both.

Hierarchies defined

A hierarchy is a stratified system of ranking and organizing people, things, ideas, etc., where each element of the system, except for the top one, is subordinate to one or more other elements. Though the concept of *hierarchy* is easily grasped intuitively, it can also be described mathematically. Diagrams of hierarchies are often shaped roughly like pyramids, but other than having a single element at the top, there is nothing necessarily pyramid-shaped about a hierarchy.

Human organizations are often structured as hierarchies, where the hierarchical system is used for assigning responsibilities, exercising leadership, and facilitating communication. Familiar hierarchies of "things" include a desktop computer's tower unit at the "top," with its subordinate monitor, keyboard, and mouse "below."

In the world of ideas, we use hierarchies to help us acquire detailed knowledge of complex reality: we structure the reality into its constituent parts, and these in turn into their own constituent parts, proceeding down the hierarchy as many levels as we care to. At each step, we focus on understanding a single component of the whole, temporarily disregarding the other components at this and all other levels. As we go through this process, we increase our global understanding of whatever complex reality we are studying.

Think of the hierarchy that medical students use while learning anatomy—they separately consider the musculoskeletal system (including parts and subparts like the hand and its constituent muscles and bones), the circulatory system (and its many levels and branches), the nervous system (and its numerous components and subsystems), etc., until

they've covered all the systems and the important subdivisions of each. Advanced students continue the subdivision all the way to the level of the cell or molecule. In the end, the students understand the "big picture" and a considerable number of its details. Not only that, but they understand the relation of the individual parts to the whole. By working hierarchically, they've gained a comprehensive understanding of anatomy.

Similarly, when we approach a complex decision problem, we can use a hierarchy to integrate large amounts of information into our understanding of the situation. As we build this information structure, we form a better and better picture of the problem as a whole.

Hierarchies in the AHP

An AHP hierarchy is a structured means of modeling the decision at hand. It consists of an overall *goal*, a group of options or *alternatives* for reaching the goal, and a group of factors or *criteria* that relate the alternatives to the goal. The criteria can be further broken down into subcriteria, sub-subcriteria, and so on, in as many levels as the problem requires.

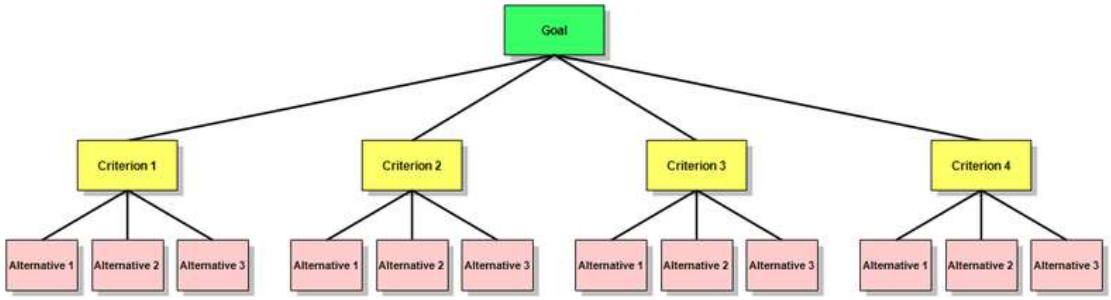
Published descriptions of AHP applications often include diagrams and descriptions of their hierarchies; some simple ones are shown throughout this article. More complex AHP hierarchies have been collected and reprinted in at least one book.

The design of any AHP hierarchy will depend not only on the nature of the problem at hand, but also on the knowledge, judgments, values, opinions, needs, wants, etc. of the participants in the decision making process. Constructing a hierarchy typically involves significant discussion, research, and discovery by those involved. Even after its initial construction, it can be changed to accommodate newly-thought-of criteria or criteria not originally considered to be important; alternatives can also be added, deleted, or changed.

To better understand AHP hierarchies, consider a decision problem with a goal to be reached, three alternative ways of reaching the goal, and four criteria against which the alternatives need to be measured.

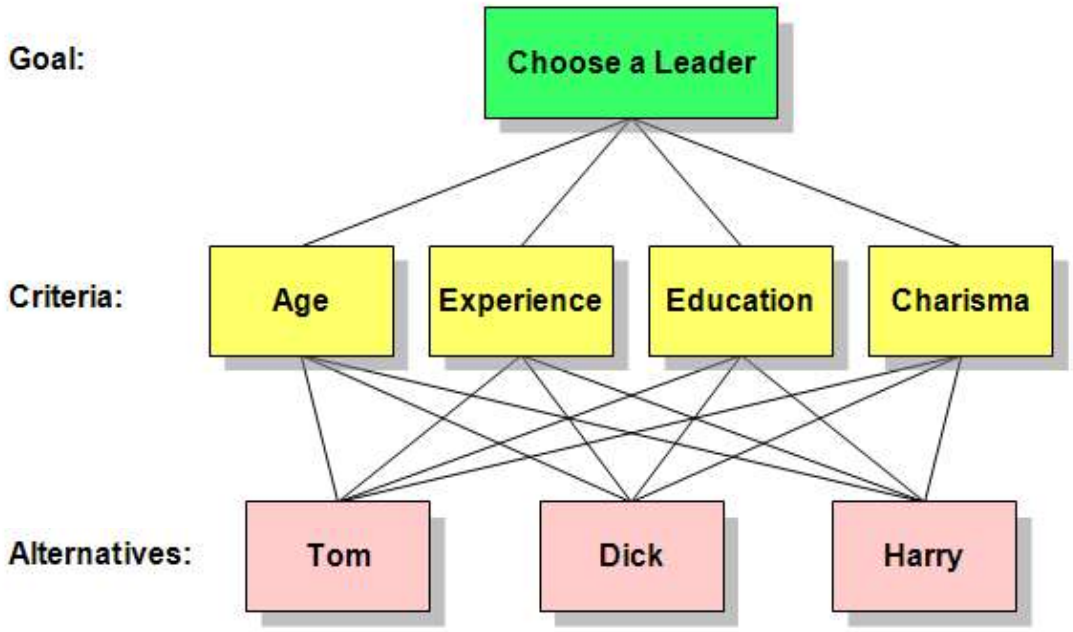
Such a hierarchy can be visualized as a diagram like the one immediately below, with the goal at the top, the three alternatives at the bottom, and the four criteria in between. There are useful terms for describing the parts of such diagrams: Each box is called a *node*. A node that is connected to one or more nodes in a level below it is called a *parent* node. The nodes to which it is so connected are called its *children*.

Applying these definitions to the diagram below, the Goal is the parent of the four Criteria, and the four Criteria are children of the Goal. Each Criterion is a parent of the three Alternatives. Note that there are only three Alternatives, but in the diagram, each of them is repeated under each of its parents.



A simple AHP hierarchy. There are three Alternatives for reaching the Goal, and four Criteria to be used in deciding among them.

To reduce the size of the drawing required, it is common to represent AHP hierarchies as shown in the diagram below, with only one node for each alternative, and with multiple lines connecting the alternatives and the criteria that apply to them. To avoid clutter, these lines are sometimes omitted or reduced in number. Regardless of any such simplifications in the diagram, in the actual hierarchy each alternative is connected to every one of its parent nodes.



AHP hierarchy for choosing a leader. There is one goal, three candidates and four criteria for choosing among them.

Evaluate the hierarchy

Once the hierarchy has been constructed, the participants analyze it through a series of *pairwise comparisons* that derive numerical scales of measurement for the nodes. The

criteria are pairwise compared against the goal for importance. The alternatives are pairwise compared against each of the criteria for preference. The comparisons are processed mathematically, and *priorities* are derived for each node.

Consider the "Choose a Leader" example above. An important task of the decision makers is to determine the weight to be given each criterion in making the choice of a leader. Another important task is to determine the weight to be given to each candidate with regard to each of the criteria. The AHP not only lets them do that, but it lets them put a meaningful and objective numerical value on each of the four criteria.

It would be impossible to do this without a tool like the AHP. First of all, none of the criteria has an existing scale of measurement that is useful in comparing the candidates.

Pairwise comparisons

Establish priorities

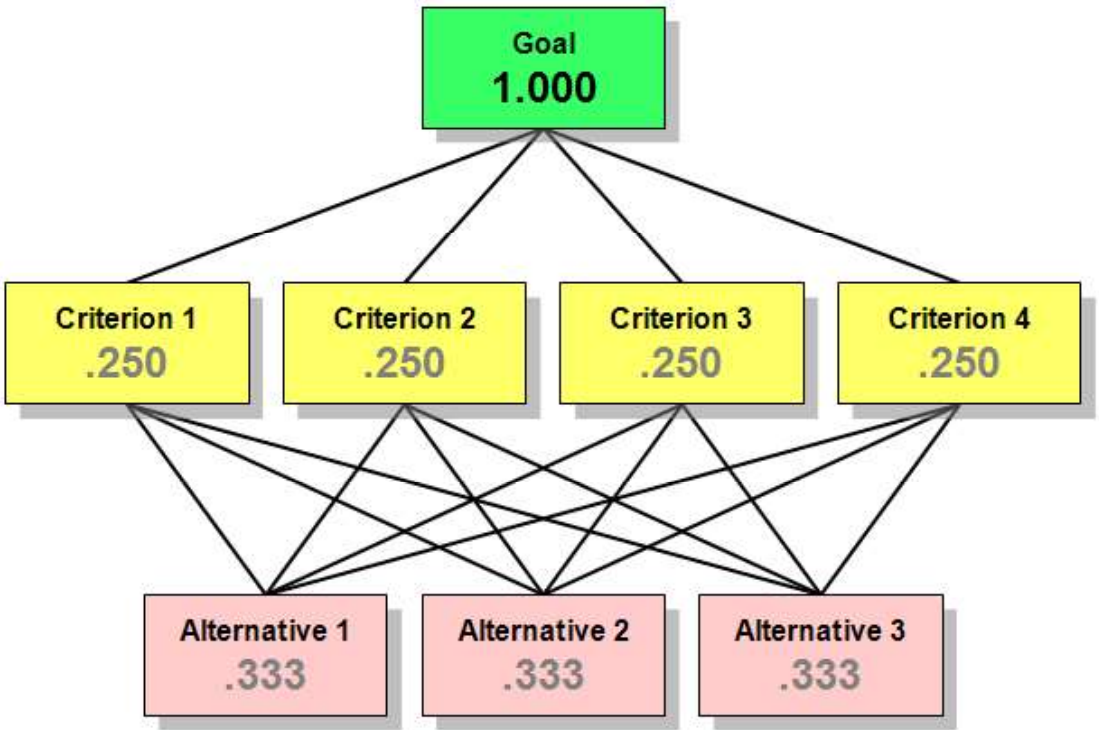
Priorities defined and explained

Priorities are numbers associated with the nodes of an AHP hierarchy. They represent the relative weights of the nodes in any group.

Like probabilities, priorities are absolute numbers between zero and one, without units or dimensions. A node with priority .200 has twice the weight in reaching the goal as one with priority .100, ten times the weight of one with priority .020, and so forth. Depending on the problem at hand, "weight" can refer to importance, or preference, or likelihood, or whatever factor is being considered by the decision makers.

Priorities are distributed over a hierarchy according to its architecture, and their values depend on the information entered by users of the process. Priorities of the Goal, the Criteria, and the Alternatives are intimately related, but need to be considered separately.

By definition, the priority of the Goal is 1.000. The priorities of the Alternatives always add up to 1.000. Things can become complicated with multiple levels of Criteria, but if there is only one level, their priorities also add to 1.000. All this is illustrated by the priorities in the example below.

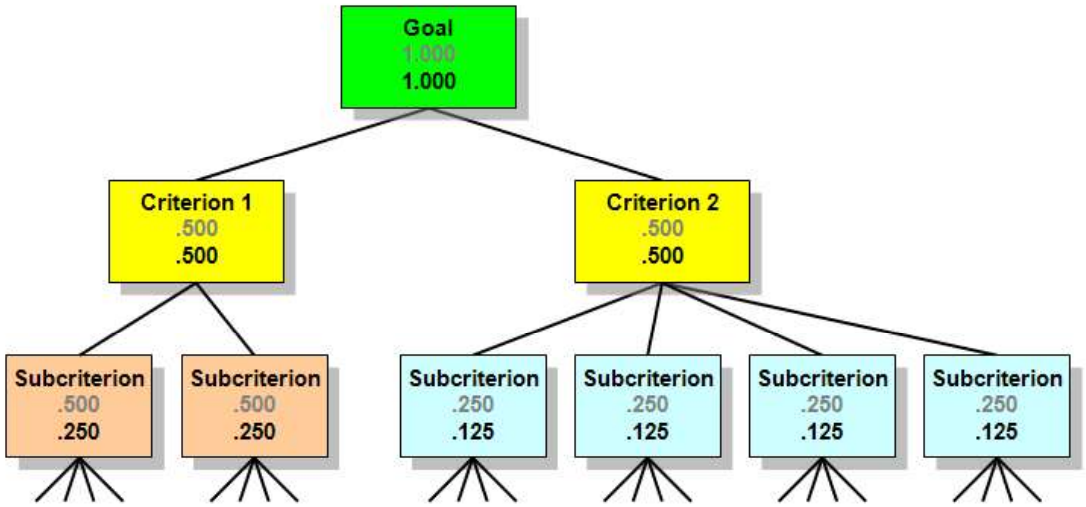


Simple AHP hierarchy with associated default priorities

Observe that the priorities on each level of the example—the Goal, the Criteria, and the Alternatives—all add up to 1.000.

The priorities shown are those that exist before any information has been entered about weights of the criteria or alternatives, so the priorities within each level are all equal. They are called the hierarchy's *default priorities*. If you understand what has been said so far, you will see that if a fifth Criterion were added to this hierarchy, the default priority for each Criterion would be .200. If there were only two Alternatives, each would have a default priority of .500.

Two additional concepts apply when a hierarchy has more than one level of criteria: *local priorities* and *global priorities*. Consider the hierarchy shown below, which has several Subcriteria under each Criterion.



A more complex AHP hierarchy, with local and global default priorities. In the interest of clarity, the decision alternatives do not appear in the diagram.

The *local priorities*, shown in gray, represent the relative weights of the nodes within a group of siblings with respect to their parent. You can easily see that the local priorities of each group of Criteria and their sibling Subcriteria add up to 1.000. The *global priorities*, shown in black, are obtained by multiplying the local priorities of the siblings by their parent's global priority. The global priorities for all the subcriteria in the level add up to 1.000.

The rule is this: Within a hierarchy, the global priorities of child nodes always add up to the global priority of their parent. Within a group of children, the local priorities add up to 1.000.

So far, we have looked only at default priorities. As the Analytical Hierarchy Process moves forward, the priorities will change from their default values as the decision makers input information about the importance of the various nodes. They do this by making a series of pairwise comparisons.

Practical examples

A simple but complete example (choosing a leader for an organization) can be found HERE.

A more detailed example (buying a family car) can be found HERE.

Criticisms

The AHP is included in most operations research and management science textbooks, and is taught in numerous universities; it is used extensively in organizations that have

carefully investigated its theoretical underpinnings. While the general consensus is that it is both technically valid and practically useful, the method does have its critics. Most of the criticisms involve a phenomenon called *rank reversal*, discussed in the following section.

Rank reversal

Decision making involves ranking alternatives in terms of criteria or attributes of those alternatives. It is an axiom of some decision theories (including AHP's) that when new alternatives are added to a decision problem, the ranking of the old alternatives must not change — that "rank reversal" must not occur.

The validity of this axiom can be disputed, since there are real-world examples where adding new alternatives can change the rank of the old ones. These rank reversals do not occur often, but the possibility of their occurrence has substantial logical implications about the methodology used to make decisions, the underlying assumptions of various decision theories, etc.

The 2000 U.S. presidential election is an example of a decision that can be understood as involving rank reversal. Ralph Nader was an 'irrelevant' alternative, in that he was dominated by both the Democrat and Republican candidates. However, since he may have attracted more votes from those who would have voted Democrat rather than Republican, his presence caused the ranks to reverse. Put another way, if Nader were not in the race, it is conceivable that Al Gore would have won. The same could be said for the impact that Ross Perot had on George Bush's loss in 1992.

There are two schools of thought about rank reversal. One maintains that new alternatives that introduce no additional attributes should not cause rank reversal under any circumstances. The other maintains that there are situations in which rank reversal is not reasonable as well as situations where they are to be expected. The current version of the AHP can accommodate both these schools — its Ideal Mode preserves rank, while its Distributive Mode allows the ranks to change. Either mode is selected according to the problem at hand.

Rank reversal and the AHP are extensively discussed in a 2001 *Operations Research* paper, as well as a chapter entitled *Rank Preservation and Reversal*, in the current basic book on AHP. The latter presents published examples of rank reversal due to adding copies and near copies of an alternative, due to intransitivity of decision rules, due to adding phantom and decoy alternatives, and due to the switching phenomenon in utility functions. It also discusses the Distributive and Ideal Modes of the AHP.

Chapter- 13

Mathematical Model

A **mathematical model** is a description of a system using mathematical language. The process of developing a mathematical model is termed **mathematical modelling** (also written *modeling*). Mathematical models are used not only in the natural sciences (such as physics, biology, earth science, meteorology) and engineering disciplines (e.g. computer science, artificial intelligence), but also in the social sciences (such as economics, psychology, sociology and political science); physicists, engineers, statisticians, operations research analysts and economists use mathematical models most extensively.

Mathematical models can take many forms, including but not limited to dynamical systems, statistical models, differential equations, or game theoretic models. These and other types of models can overlap, with a given model involving a variety of abstract structures.

Examples of mathematical models

- *Population Growth*. A simple (though approximate) model of population growth is the Malthusian growth model. A slightly more realistic and largely used population growth model is the logistic function, and its extensions.
- *Model of a particle in a potential-field*. In this model we consider a particle as being a point of mass which describes a trajectory in space which is modeled by a function giving its coordinates in space as a function of time. The potential field is given by a function $V : \mathbf{R}^3 \to \mathbf{R}$ and the trajectory is a solution of the differential equation

  $$m\frac{d^2}{dt^2}x(t) = -\operatorname{grad}(V)(x(t)).$$

  Note this model assumes the particle is a point mass, which is certainly known to be false in many cases in which we use this model; for example, as a model of planetary motion.

- *Model of rational behavior for a consumer*. In this model we assume a consumer faces a choice of n commodities labeled 1,2,...,n each with a market price p_1, p_2,..., p_n. The consumer is assumed to have a *cardinal* utility function U (cardinal in the sense that it assigns numerical values to utilities), depending on the amounts of commodities x_1, x_2,..., x_n consumed. The model further assumes that the consumer

has a budget M which is used to purchase a vector $x_1, x_2,..., x_n$ in such a way as to maximize $U(x_1, x_2,..., x_n)$. The problem of rational behavior in this model then becomes an optimization problem, that is:

$$\max U(x_1, x_2, \ldots, x_n)$$

subject to:

$$\sum_{i=1}^{n} p_i x_i \leq M.$$

$$x_i \geq 0 \quad \forall i \in \{1, 2, \ldots, n\}$$

This model has been used in general equilibrium theory, particularly to show existence and Pareto efficiency of economic equilibria. However, the fact that this particular formulation assigns *numerical values* to levels of satisfaction is the source of criticism (and even ridicule). However, it is not an essential ingredient of the theory and again this is an idealization.

- *Neighbour-sensing model* explains the mushroom formation from the initially chaotic fungal network.

Modelling requires selecting and identifying relevant aspects of a situation in the real world.

Background

Often when engineers analyze a system to be controlled or optimized, they use a mathematical model. In analysis, engineers can build a descriptive model of the system as a hypothesis of how the system could work, or try to estimate how an unforeseeable event could affect the system. Similarly, in control of a system, engineers can try out different control approaches in simulations.

A mathematical model usually describes a system by a set of variables and a set of equations that establish relationships between the variables. The values of the variables can be practically anything; real or integer numbers, boolean values or strings, for example. The variables represent some properties of the system, for example, measured system outputs often in the form of signals, timing data, counters, and event occurrence (yes/no). The actual model is the set of functions that describe the relations between the different variables.

Building blocks

There are six basic groups of variables: decision variables, input variables, state variables, exogenous variables, random variables, and output variables. Since there can be many variables of each type, the variables are generally represented by vectors.

Decision variables are sometimes known as independent variables. Exogenous variables are sometimes known as parameters or constants. The variables are not independent of

each other as the state variables are dependent on the decision, input, random, and exogenous variables. Furthermore, the output variables are dependent on the state of the system (represented by the state variables).

Objectives and constraints of the system and its users can be represented as functions of the output variables or state variables. The objective functions will depend on the perspective of the model's user. Depending on the context, an objective function is also known as an index of performance, as it is some measure of interest to the user. Although there is no limit to the number of objective functions and constraints a model can have, using or optimizing the model becomes more involved (computationally) as the number increases.

Classifying mathematical models

Many mathematical models can be classified in some of the following ways:

1. **Linear vs. nonlinear:** Mathematical models are usually composed by variables, which are abstractions of quantities of interest in the described systems, and operators that act on these variables, which can be algebraic operators, functions, differential operators, etc. If all the operators in a mathematical model exhibit linearity, the resulting mathematical model is defined as linear. A model is considered to be nonlinear otherwise.
   The question of linearity and nonlinearity is dependent on context, and linear models may have nonlinear expressions in them. For example, in a statistical linear model, it is assumed that a relationship is linear in the parameters, but it may be nonlinear in the predictor variables. Similarly, a differential equation is said to be linear if it can be written with linear differential operators, but it can still have nonlinear expressions in it. In a mathematical programming model, if the objective functions and constraints are represented entirely by linear equations, then the model is regarded as a linear model. If one or more of the objective functions or constraints are represented with a nonlinear equation, then the model is known as a nonlinear model.
   Nonlinearity, even in fairly simple systems, is often associated with phenomena such as chaos and irreversibility. Although there are exceptions, nonlinear systems and models tend to be more difficult to study than linear ones. A common approach to nonlinear problems is linearization, but this can be problematic if one is trying to study aspects such as irreversibility, which are strongly tied to nonlinearity.
2. **Deterministic vs. probabilistic (stochastic):** A deterministic model is one in which every set of variable states is uniquely determined by parameters in the model and by sets of previous states of these variables. Therefore, deterministic models perform the same way for a given set of initial conditions. Conversely, in a stochastic model, randomness is present, and variable states are not described by unique values, but rather by probability distributions.

3. **Static vs. dynamic:** A static model does not account for the element of time, while a dynamic model does. Dynamic models typically are represented with difference equations or differential equations.
4. **Discrete vs. Continuous:** A discrete model does not take into account the function of time and usually uses time-advance methods, while a Continuous model does. Continuous models typically are represented with f(t) and the changes are reflected over continuous time intervals.

A priori information

Mathematical modelling problems are often classified into black box or white box models, according to how much a priori information is available of the system. A black-box model is a system of which there is no a priori information available. A white-box model (also called glass box or clear box) is a system where all necessary information is available. Practically all systems are somewhere between the black-box and white-box models, so this concept is useful only as an intuitive guide for deciding which approach to take.

Usually it is preferable to use as much a priori information as possible to make the model more accurate. Therefore the white-box models are usually considered easier, because if you have used the information correctly, then the model will behave correctly. Often the a priori information comes in forms of knowing the type of functions relating different variables. For example, if we make a model of how a medicine works in a human system, we know that usually the amount of medicine in the blood is an exponentially decaying function. But we are still left with several unknown parameters; how rapidly does the medicine amount decay, and what is the initial amount of medicine in blood? This example is therefore not a completely white-box model. These parameters have to be estimated through some means before one can use the model.

In black-box models one tries to estimate both the functional form of relations between variables and the numerical parameters in those functions. Using a priori information we could end up, for example, with a set of functions that probably could describe the system adequately. If there is no a priori information we would try to use functions as general as possible to cover all different models. An often used approach for black-box models are neural networks which usually do not make assumptions about incoming data. The problem with using a large set of functions to describe a system is that estimating the parameters becomes increasingly difficult when the amount of parameters (and different types of functions) increases.

Subjective information

Sometimes it is useful to incorporate subjective information into a mathematical model. This can be done based on intuition, experience, or expert opinion, or based on convenience of mathematical form. Bayesian statistics provides a theoretical framework for incorporating such subjectivity into a rigorous analysis: one specifies a prior probability distribution (which can be subjective) and then updates this distribution based

on empirical data. An example of when such approach would be necessary is a situation in which an experimenter bends a coin slightly and tosses it once, recording whether it comes up heads, and is then given the task of predicting the probability that the next flip comes up heads. After bending the coin, the true probability that the coin will come up heads is unknown, so the experimenter would need to make an arbitrary decision (perhaps by looking at the shape of the coin) about what prior distribution to use. Incorporation of the subjective information is necessary in this case to get an accurate prediction of the probability, since otherwise one would guess 1 or 0 as the probability of the next flip being heads, which would be almost certainly wrong.

Complexity

In general, model complexity involves a trade-off between simplicity and accuracy of the model. Occam's Razor is a principle particularly relevant to modelling; the essential idea being that among models with roughly equal predictive power, the simplest one is the most desirable. While added complexity usually improves the realism of a model, it can make the model difficult to understand and analyze, and can also pose computational problems, including numerical instability. Thomas Kuhn argues that as science progresses, explanations tend to become more complex before a Paradigm shift offers radical simplification.

For example, when modelling the flight of an aircraft, we could embed each mechanical part of the aircraft into our model and would thus acquire an almost white-box model of the system. However, the computational cost of adding such a huge amount of detail would effectively inhibit the usage of such a model. Additionally, the uncertainty would increase due to an overly complex system, because each separate part induces some amount of variance into the model. It is therefore usually appropriate to make some approximations to reduce the model to a sensible size. Engineers often can accept some approximations in order to get a more robust and simple model. For example Newton's classical mechanics is an approximated model of the real world. Still, Newton's model is quite sufficient for most ordinary-life situations, that is, as long as particle speeds are well below the speed of light, and we study macro-particles only.

Training

Any model which is not pure white-box contains some parameters that can be used to fit the model to the system it is intended to describe. If the modelling is done by a neural network, the optimization of parameters is called *training*. In more conventional modelling through explicitly given mathematical functions, parameters are determined by curve fitting.

Model evaluation

A crucial part of the modelling process is the evaluation of whether or not a given mathematical model describes a system accurately. This question can be difficult to answer as it involves several different types of evaluation.

Fit to empirical data

Usually the easiest part of model evaluation is checking whether a model fits experimental measurements or other empirical data. In models with parameters, a common approach to test this fit is to split the data into two disjoint subsets: training data and verification data. The training data are used to estimate the model parameters. An accurate model will closely match the verification data even though this data was not used to set the model's parameters. This practice is referred to as cross-validation in statistics.

Defining a metric to measure distances between observed and predicted data is a useful tool of assessing model fit. In statistics, decision theory, and some economic models, a loss function plays a similar role.

While it is rather straightforward to test the appropriateness of parameters, it can be more difficult to test the validity of the general mathematical form of a model. In general, more mathematical tools have been developed to test the fit of statistical models than models involving Differential equations. Tools from nonparametric statistics can sometimes be used to evaluate how well data fits a known distribution or to come up with a general model that makes only minimal assumptions about the model's mathematical form.

Scope of the model

Assessing the scope of a model, that is, determining what situations the model is applicable to, can be less straightforward. If the model was constructed based on a set of data, one must determine for which systems or situations the known data is a "typical" set of data.

The question of whether the model describes well the properties of the system between data points is called interpolation, and the same question for events or data points outside the observed data is called extrapolation.

As an example of the typical limitations of the scope of a model, in evaluating Newtonian classical mechanics, we can note that Newton made his measurements without advanced equipment, so he could not measure properties of particles travelling at speeds close to the speed of light. Likewise, he did not measure the movements of molecules and other small particles, but macro particles only. It is then not surprising that his model does not extrapolate well into these domains, even though his model is quite sufficient for ordinary life physics.

Philosophical considerations

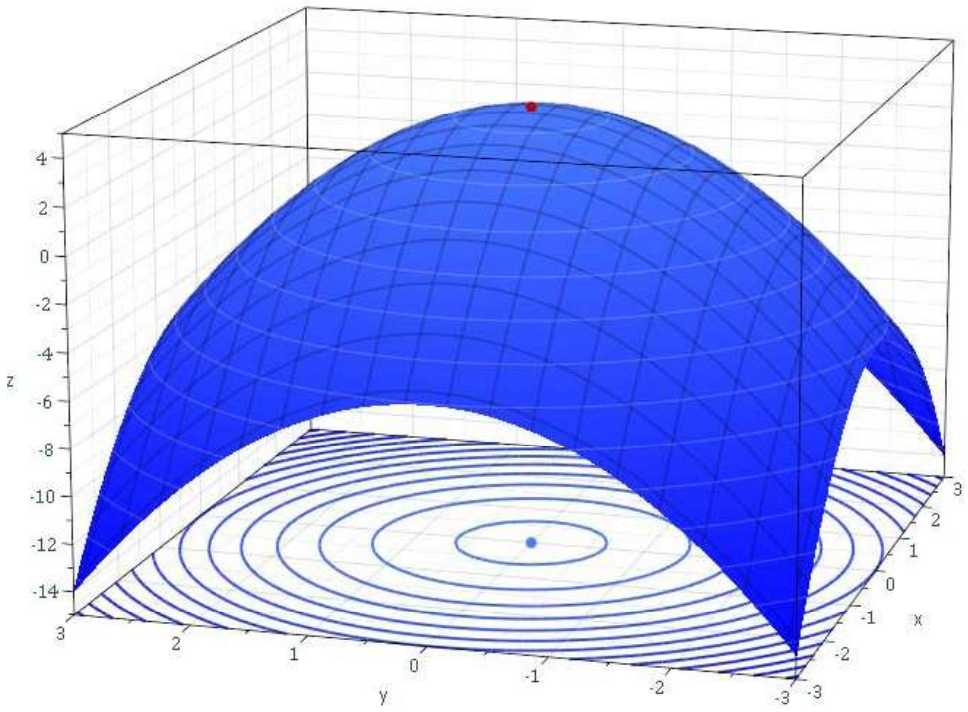
Many types of modelling implicitly involve claims about causality. This is usually (but not always) true of models involving differential equations. As the purpose of modelling is to increase our understanding of the world, the validity of a model rests not only on its fit to empirical observations, but also on its ability to extrapolate to situations or data

beyond those originally described in the model. One can argue that a model is worthless unless it provides some insight which goes beyond what is already known from direct investigation of the phenomenon being studied.

An example of such criticism is the argument that the mathematical models of Optimal foraging theory do not offer insight that goes beyond the common-sense conclusions of evolution and other basic principles of ecology.

Chapter- 14

Optimization



The maximum of a paraboloid

In mathematics, computer science and economics, **optimization**, or **mathematical programming**, refers to choosing the best element from some set of available alternatives.

In the simplest case, this means solving problems in which one seeks to minimize or maximize a real function by systematically choosing the values of real or integer variables from within an allowed set. This formulation, using a scalar, real-valued objective function, is probably the simplest example; the generalization of optimization theory and techniques to other formulations comprises a large area of applied mathematics. More generally, it means finding "best available" values of some objective

function given a defined domain, including a variety of different types of objective functions and different types of domains.

History

The first optimization technique, which is known as steepest descent, goes back to Gauss. Historically, the first term to be introduced was "linear programming", which was due to George B. Dantzig, although much of the theory had been introduced by Leonid Kantorovich in 1939. Dantzig published the Simplex algorithm in 1947, and John von Neumann developed the theory of the duality in the same year. The term *programming* in this context does not refer to computer programming. Rather, the term comes from the use of *program* by the United States military to refer to proposed training and logistics schedules, which were the problems that Dantzig was studying at the time.

Other important mathematicians in the optimization field include:

- Richard Bellman
- Ronald A. Howard
- Leonid Kantorovich
- Narendra Karmarkar
- William Karush
- Leonid Khachiyan
- Bernard Koopman
- Harold Kuhn
- Joseph Louis Lagrange
- László Lovász
- Arkadii Nemirovskii
- Yurii Nesterov
- John von Neumann
- Boris Polyak
- Lev Pontryagin
- James Renegar
- Kees Roos
- Naum Z. Shor
- Michael J. Todd
- Albert Tucker

Major subfields

- Convex programming studies the case when the objective function is convex and the constraints, if any, form a convex set. This can be viewed as a particular case of nonlinear programming or as generalization of linear or convex quadratic programming.
    - Linear programming (LP), a type of convex programming, studies the case in which the objective function *f* is linear and the set of constraints is specified using only linear equalities and inequalities. Such a set is called a polyhedron or a polytope if it is bounded.
    - Second order cone programming (SOCP) is a convex program, and includes certain types of quadratic programs.
    - Semidefinite programming (SDP) is a subfield of convex optimization where the underlying variables are semidefinite matrices. It is generalization of linear and convex quadratic programming.
    - Conic programming is a general form of convex programming. LP, SOCP and SDP can all be viewed as conic programs with the appropriate type of cone.

    - Geometric programming is a technique whereby objective and inequality constraints expressed as posynomials and equality constraints as monomials can be transformed into a convex program.
- Integer programming studies linear programs in which some or all variables are constrained to take on integer values. This is not convex, and in general much more difficult than regular linear programming.
- Quadratic programming allows the objective function to have quadratic terms, while the set A must be specified with linear equalities and inequalities. For specific forms of the quadratic term, this is a type of convex programming.
- Nonlinear programming studies the general case in which the objective function or the constraints or both contain nonlinear parts. This may or may not be a convex program. In general, the convexity of the program affects the difficulty of solving more than the linearity.
- Stochastic programming studies the case in which some of the constraints or parameters depend on random variables.
- Robust programming is, as stochastic programming, an attempt to capture uncertainty in the data underlying the optimization problem. This is not done through the use of random variables, but instead, the problem is solved taking into account inaccuracies in the input data.
- Combinatorial optimization is concerned with problems where the set of feasible solutions is discrete or can be reduced to a discrete one.
- Infinite-dimensional optimization studies the case when the set of feasible solutions is a subset of an infinite-dimensional space, such as a space of functions.
- Heuristic algorithms
    - Metaheuristics
- Constraint satisfaction studies the case in which the objective function *f* is constant (this is used in artificial intelligence, particularly in automated reasoning).
    - Constraint programming.
- Disjunctive programming used where at least one constraint must be satisfied but not all. Of particular use in scheduling.
- Trajectory optimization is the specialty of optimizing trajectories for air and space vehicles.

In a number of subfields, the techniques are designed primarily for optimization in dynamic contexts (that is, decision making over time):

- Calculus of variations seeks to optimize an objective defined over many points in time, by considering how the objective function changes if there is a small change in the choice path.
- Optimal control theory is a generalization of the calculus of variations.
- Dynamic programming studies the case in which the optimization strategy is based on splitting the problem into smaller subproblems. The equation that describes the relationship between these subproblems is called the Bellman equation.

- Mathematical programming with equilibrium constraints is where the constraints include variational inequalities or complementarities.

Multi-objective optimization

Adding more than one objective to an optimization problem adds complexity. For example, if you wanted to optimize a structural design, you would want a design that is both light and rigid. Because these two objectives conflict, a trade-off exists. There will be one lightest design, one stiffest design, and an infinite number of designs that are some compromise of weight and stiffness. This set of trade-off designs is known as a Pareto set. The curve created plotting weight against stiffness of the best designs is known as the Pareto frontier.

A design is judged to be Pareto optimal if it is not dominated by other designs: a Pareto optimal design must be better than another design in at least one aspect. If it is worse than another design in all respects, then it is dominated and is not Pareto optimal.

Multi-modal optimization

Optimization problems are often multi-modal, that is they possess multiple good solutions. They could all be globally good (same cost function value) or there could be a mix of globally good and locally good solutions. Obtaining all (or at least some of) the multiple solutions is the goal of a multi-modal optimizer.

Classical optimization techniques due to their iterative approach do not perform satisfactorily when they are used to obtain multiple solutions, since it is not guaranteed that different solutions will be obtained even with different starting points in multiple runs of the algorithm. Evolutionary Algorithms are however a very popular approach to obtain multiple solutions in a multi-modal optimization task.

Concepts and notation

Optimization problems

An optimization problem can be represented in the following way

> *Given:* a function $f : A \longrightarrow \mathbf{R}$ from some set A to the real numbers
> *Sought:* an element x_0 in A such that $f(x_0) \leq f(x)$ for all x in A ("minimization") or such that $f(x_0) \geq f(x)$ for all x in A ("maximization").

Such a formulation is called an **optimization problem** or a **mathematical programming problem** (a term not directly related to computer programming, but still in use for example in linear programming - see History above). Many real-world and theoretical problems may be modeled in this general framework. Problems formulated using this technique in the fields of physics and computer vision may refer to the technique as

energy minimization, speaking of the value of the function *f* as representing the energy of the system being modeled.

Typically, *A* is some subset of the Euclidean space $\mathbf{R}^n$, often specified by a set of *constraints*, equalities or inequalities that the members of *A* have to satisfy. The domain *A* of *f* is called the *search space* or the *choice set*, while the elements of *A* are called *candidate solutions* or *feasible solutions*.

The function *f* is called, variously, an **objective function**, **cost function**, **energy function**, or **energy functional**. A feasible solution that minimizes (or maximizes, if that is the goal) the objective function is called an *optimal solution*.

Generally, when the feasible region or the objective function of the problem does not present convexity, there may be several local minima and maxima, where a *local minimum* x^* is defined as a point for which there exists some $\delta > 0$ so that for all x such that

$$\|\mathbf{x} - \mathbf{x}^*\| \leq \delta;$$

the expression

$$f(\mathbf{x}^*) \leq f(\mathbf{x})$$

holds; that is to say, on some region around x^* all of the function values are greater than or equal to the value at that point. Local maxima are defined similarly.

A large number of algorithms proposed for solving non-convex problems – including the majority of commercially available solvers – are not capable of making a distinction between local optimal solutions and rigorous optimal solutions, and will treat the former as actual solutions to the original problem. The branch of applied mathematics and numerical analysis that is concerned with the development of deterministic algorithms that are capable of guaranteeing convergence in finite time to the actual optimal solution of a non-convex problem is called global optimization.

Notation

Optimization problems are often expressed with special notation. Here are some examples.

$$\min_{x \in \mathbb{R}} (x^2 + 1)$$

This asks for the minimum value for the objective function $x^2 + 1$, where *x* ranges over the real numbers $\mathbb{R}$. The minimum value in this case is 1, occurring at $x = 0$.

$$\max_{x \in \mathbb{R}} 2x$$

This asks for the maximum value for the objective function $2x$, where x ranges over the reals. In this case, there is no such maximum as the objective function is unbounded, so the answer is "infinity" or "undefined".

$$\operatorname{argmin}_{x \in (-\infty, -1]} x^2 + 1$$

This asks for the value (or values) of x in the interval $(-\infty, -1]$ that minimizes (or minimize) the objective function $x^2 + 1$ (the actual minimum value of that function does not matter). In this case, the answer is $x = -1$.

$$\operatorname{argmax}_{x \in [-5,5],\, y \in \mathbb{R}} x \cdot \cos(y)$$

This asks for the (x,y) pair (or pairs) that maximizes (or maximize) the value of the objective function $x \cdot \cos(y)$, with the added constraint that x lies in the interval $[-5,5]$ (again, the actual maximum value of the expression does not matter). In this case, the solutions are the pairs of the form $(5, 2k\pi)$ and $(-5,(2k+1)\pi)$, where k ranges over all integers.

Classification of critical points and extrema

Feasibility problem

The **satisfiability problem**, also called the **feasibility problem**, is just the problem of finding any feasible solution at all without regard to objective value. This can be regarded as the special case of mathematical optimization where the objective value is the same for every solution, and thus any solution is optimal.

Many optimization algorithms need to start from a feasible point. One way to obtain such a point is to relax the feasibility conditions using a slack variable; with enough slack, any starting point is feasible. Then, minimize that slack variable until slack is null or negative.

Existence

The extreme value theorem of Karl Weierstrass states that a continuous real-valued function on a compact set attains its maximum and minimum value. More generally, a lower semi-continous function on a compact set attains its minimum; an upper semi-continous function on a compact set attains its maximum.

Sufficient conditions for optimality

One of Fermat's theorems states that optima of unconstrained problems are found at stationary points, where the first derivative or the gradient of the objective function is zero. More generally, they may be found at critical points, where the first derivative or gradient of the objective function is zero or is undefined, or on the boundary of the choice

set. An equation stating that the first derivative equals zero at an interior optimum is sometimes called a 'first-order condition'.

Optima of inequality-constrained problems are instead found by the Lagrange multiplier method. This method calculates a system of inequalities called the 'Karush–Kuhn–Tucker conditions' or 'complementary slackness conditions', which may then be used to calculate the optimum.

While the first derivative test identifies points that might be optima, this test does not distinguish a point which is a minimum from one that is a maximum or one that is neither. When the objective function is twice differentiable, these cases can be distinguished by checking the second derivative or the matrix of second derivatives (called the Hessian matrix) in unconstrained problems, or a matrix of second derivatives of the objective function and the constraints called the bordered Hessian. The conditions that distinguish maxima and minima from other stationary points are sometimes called 'second-order conditions'.

Sensitivity and stability of optima

The envelope theorem describes how the value of an optimal solution changes when an underlying parameter changes.

The maximum theorem of Claude Berge (1963) describes the continuity of an optimal solution as a function of underlying parameters.

Calculus of optimization

For unconstrained problems with twice-differentiable functions, some critical points can be found by finding the points where the gradient of the objective function is zero (that is, the stationary points). More generally, a zero subgradient certifies that a local minimum has been found for minimization problems with convex functions and other locally Lipschitz functions.

Further, critical points can be classified using the definiteness of the Hessian matrix: If the Hessian is *positive* definite at a critical point, then the point is a local minimum; if the Hessian matrix is negative definite, then the point is a local maximum; finally, if indefinite, then the point is some kind of saddle point.

Constrained problems can often be transformed into unconstrained problems with the help of Lagrange multipliers. Lagrangian relaxation also can also provide approximate solutions to difficult constrained problems.

When the objective function is convex, then any local minimum will also be a global minimum. There exist efficient numerical techniques for minimizing convex functions, such as interior-point methods.

Computational optimization techniques

To solve problems, researchers may use algorithms that terminate in a finite number of steps, or iterative methods that converge to a solution (on some specified class of problems), or heuristics that may provide approximate solutions to some problems (although their iterates need not converge).

Optimization algorithms

- Simplex algorithm of George Dantzig: For linear programming. Extensions of the simplex algorithm exist for quadratic programming and for linear-fractional programming. The simplex algorithm has variants that are especially suited for network optimization.
- Combinatorial algorithms

Iterative methods

For some nonlinear problems, the computational complexity of evaluating gradients and Hessians can be excessive. Some many problems of nonlinear programming, the iterative methods differ according to whether they evaluate Hessians, gradients, or only function values. While evaluating Hessians and gradients improves the rate of convergence of methods, such evaluations increase the computational cost of each iteration, so that users must select a balance.

- Methods that evaluate Hessians (or approximate Hessians, using finite differences):
    - Newton's method
    - Sequential quadratic programming: A method for small-medium scale constrained problems. Some versions can handle large-dimensional problems.

- Methods that evaluate gradients or approximate gradients using finite differences (or even subgradients):
    - Quasi-Newton methods: Iterative methods for medium-large problems.
    - Conjugate gradient methods: Iterative methods for large problems. (In theory, these methods terminate in a finite number of steps with quadratic objective functions, but this finite termination is not observed in practice on finite–precision computers.)
    - Interior point methods: This is a large classs of methods for constrained optimization. Some interior-point methods use only (sub)gradient information, and others of which require the evaluation of Hessians.
    - Gradient descent (alternatively, "steepest descent" or "steepest ascent"): A method of historical and theoretical interest, which has had renewed interest for find approximate solutions of enormous problems.
    - Subgradient methods - An iterative method for large locally Lipschitz functions using generalized gradients. Following Boris T. Polyak,

subgradient–projection methods are similar to conjugate–gradient methods.
    - Bundle method of descent: An iterative method for small–medium sized problems with locally Lipschitz functions, particularly for convex minimization problems. (Similar to conjugate gradient methods)
    - Ellipsoid method: An iterative method for small problems with quasiconvex objective functions and of great theoretical interest, particularly in establishing the polynomial time complexity of some combinatorial optimization problems. It has similarities with Quasi-Newton methods.
    - Reduced gradient method (Frank–Wolfe) for approximate minimization of specially structured problems with linear constraints, especially with traffic networks. For general unconstrained problems, this method reduces to the gradient method, which is regarded as obsolete (for almost all problems).

- Methods that evaluate only function values: If a problem is continuously differentiable, then gradients can be approximated using finite differences, in which case a gradient-based method can be used.
    - Interpolation Methods: (Michael J. D. Powell)

Global convergence

More generally, if the objective function is not a quadratic function, then many optimization methods use other methods to ensure that some subsequence of iterations converges to an optimal solution. The first and still popular method for ensuring convergence relies on-line searches, which optimize a function along one dimension. A second and increasingly popular method for ensuring convergence uses trust regions. Both line searches and trust regions are used in modern methods of non-differentiable optimization.

Heuristics

Besides (finitely terminating) algorithms and (convergent) iterative methods, there are heuristics that can provide approximate solutions to some optimization problems:

- Differential evolution
- Dynamic relaxation
- Genetic algorithms
- Hill climbing
- Nelder-Mead simplicial heuristic: A popular heuristic for approximate minimization (without calling gradients)
- Particle swarm optimization
- Simulated annealing
- Tabu search

Applications

Problems in rigid body dynamics (in particular articulated rigid body dynamics) often require mathematical programming techniques, since you can view rigid body dynamics as attempting to solve an ordinary differential equation on a constraint manifold; the constraints are various nonlinear geometric constraints such as "these two points must always coincide", "this surface must not penetrate any other", or "this point must always lie somewhere on this curve". Also, the problem of computing contact forces can be done by solving a linear complementarity problem, which can also be viewed as a QP (quadratic programming) problem.

Many design problems can also be expressed as optimization programs. This application is called design optimization. One subset is the engineering optimization, and another recent and growing subset of this field is multidisciplinary design optimization, which, while useful in many problems, has in particular been applied to aerospace engineering problems.

Economics also relies heavily on mathematical programming. An often studied problem in microeconomics, the utility maximization problem, and its dual problem the Expenditure minimization problem, are economic optimization problems. Consumers and firms are assumed to maximize their utility/profit. Also, agents are most frequently assumed to be risk-averse thereby wishing to minimize whatever risk they might be exposed to. Asset prices are also explained using optimization though the underlying theory is more complicated than simple utility or profit optimization. Trade theory also uses optimization to explain trade patterns between nations.

Another field that uses optimization techniques extensively is operations research.

Solvers

- Comet
- CPLEX
- FortSP - solver for stochastic programming problems
- Gurobi
- IMSL Numerical Libraries are collections of math and statistical algorithms available in C/C++, Fortran, Java and C#/.NET. Optimization routines in the IMSL Libraries include unconstrained, linearly and nonlinearly constrained minimizations, and linear programming algorithms.
- IPOPT - an open-source primal-dual interior point method NLP solver which handles sparse matrices
- KNITRO - solver for nonlinear optimization problems
- MATLAB Optimization Toolbox - solvers for linear and nonlinear optimization problems
- Mathematica - handles linear programming, integer programming and constrained non-linear optimization problems

- Merlin - A Fortran-77, user friendly open source software package, for non-linear optimization with bound constraints.
- MINUIT
- NAG Libraries - local and global optimization routines available for multiple programming languages (C, C++, Fortran, Python, Java, .NET, GPUs), packages (MATLAB, Maple, Excel) and for SMP and multicore
- OpenOpt - a free optimization framework written in Python and NumPy, connects to tens of solvers, can involve Automatic differentiation
- NLopt a free optimization library, callable from C/C++/Fortran/Python/Scheme, which interfaces to a large number algorithms for global and local constrained and unconstrained nonlinear optimization.
- Opt++ - An object-oriented package from Lawrence Berkeley and Sandia National Labs, used for nonlinear optimization.
- SNOPT

Chapter- 15

Data Mining

Data mining, a branch of computer science and artificial intelligence , is the process of extracting patterns from data. Data mining is seen as an increasingly important tool by modern business to transform data into business intelligence giving an informational advantage. It is currently used in a wide range of profiling practices, such as marketing, surveillance, fraud detection, and scientific discovery.

The related terms *data dredging*, *data fishing* and *data snooping* refer to the use of data mining techniques to sample portions of the larger population data set that are (or may be) too small for reliable statistical inferences to be made about the validity of any patterns discovered. These techniques can, however, be used in the creation of new hypotheses to test against the larger data populations.

Background

The manual extraction of patterns from data has occurred for centuries. Early methods of identifying patterns in data include Bayes' theorem (1700s) and regression analysis (1800s). The proliferation, ubiquity and increasing power of computer technology has increased data collection, storage and manipulations. As data sets have grown in size and complexity, direct hands-on data analysis has increasingly been augmented with indirect, automatic data processing. This has been aided by other discoveries in computer science, such as neural networks, clustering, genetic algorithms (1950s), decision trees (1960s) and support vector machines (1980s). Data mining is the process of applying these methods to data with the intention of uncovering hidden patterns. It has been used for many years by businesses, scientists and governments to sift through volumes of data such as airline passenger trip records, census data and supermarket scanner data to produce market research reports. (Note, however, that reporting is not always considered to be data mining.)

A primary reason for using data mining is to assist in the analysis of collections of observations of behaviour. Such data are vulnerable to collinearity because of unknown interrelations. An unavoidable fact of data mining is that the (sub-)set(s) of data being analysed may not be representative of the whole domain, and therefore may not contain examples of certain critical relationships and behaviours that exist across other parts of the domain. To address this sort of issue, the analysis may be augmented using experiment-based and other approaches, such as Choice Modelling for human-generated

data. In these situations, inherent correlations can be either controlled for, or removed altogether, during the construction of the experimental design.

There have been some efforts to define standards for data mining, for example the 1999 European Cross Industry Standard Process for Data Mining (CRISP-DM 1.0) and the 2004 Java Data Mining standard (JDM 1.0). These are evolving standards; later versions of these standards are under development. Independent of these standardization efforts, freely available open-source software systems like the R Project, Weka, KNIME, RapidMiner and others have become an informal standard for defining data-mining processes. Notably, all these systems are able to import and export models in PMML (Predictive Model Markup Language) which provides a standard way to represent data mining models so that these can be shared between different statistical applications. PMML is an XML-based language developed by the Data Mining Group (DMG), an independent group composed of many data mining companies. PMML version 4.0 was released in June 2009.

Research and evolution

In addition to industry driven demand for standards and interoperability, professional and academic activity have also made considerable contributions to the evolution and rigour of the methods and models; an article published in a 2008 issue of the *International Journal of Information Technology and Decision Making* summarises the results of a literature survey which traces and analyzes this evolution.

The premier professional body in the field is the Association for Computing Machinery's Special Interest Group on Knowledge discovery and Data Mining (SIGKDD). Since 1989 they have hosted an annual international conference and published its proceedings, and since 1999 have published a biannual academic journal titled "SIGKDD Explorations". Other Computer Science conferences on data mining include:

- DMIN - International Conference on Data Mining;
- DMKD - Research Issues on Data Mining and Knowledge Discovery;
- ECML-PKDD - European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases;
- ICDM - IEEE International Conference on Data Mining;
- MLDM - Machine Learning and Data Mining in Pattern Recognition;
- SDM - SIAM International Conference on Data Mining
- EDM - International Conference on Educational Data Mining
- ECDM - European Conference on Data Mining
- PAKDD - The annual Pacific-Asia Conference on Knowledge Discovery and Data Mining

Process

Pre-processing

Before data mining algorithms can be used, a target data set must be assembled. As data mining can only uncover patterns already present in the data, the target dataset must be large enough to contain these patterns while remaining concise enough to be mined in an acceptable timeframe. A common source for data is a datamart or data warehouse. Pre-process is essential to analyse the multivariate datasets before clustering or data mining.

The target set is then cleaned. Cleaning removes the observations with noise and missing data.

The clean data are reduced into feature vectors, one vector per observation. A feature vector is a summarised version of the raw data observation. For example, a black and white image of a face which is 100px by 100px would contain 10,000 bits of raw data. This might be turned into a feature vector by locating the eyes and mouth in the image. Doing so would reduce the data for each vector from 10,000 bits to three codes for the locations, dramatically reducing the size of the dataset to be mined, and hence reducing the processing effort. The feature(s) selected will depend on what the objective(s) is/are; obviously, selecting the "right" feature(s) is fundamental to successful data mining.

The feature vectors are divided into two sets, the "training set" and the "test set". The training set is used to "train" the data mining algorithm(s), while the test set is used to verify the accuracy of any patterns found.

Data mining

Data mining commonly involves four classes of tasks:

- Clustering - is the task of discovering groups and structures in the data that are in some way or another "similar", without using known structures in the data.
- Classification - is the task of generalizing known structure to apply to new data. For example, an email program might attempt to classify an email as legitimate or spam. Common algorithms include decision tree learning, nearest neighbor, naive Bayesian classification, neural networks and support vector machines.
- Regression - Attempts to find a function which models the data with the least error.
- Association rule learning - Searches for relationships between variables. For example a supermarket might gather data on customer purchasing habits. Using association rule learning, the supermarket can determine which products are frequently bought together and use this information for marketing purposes. This is sometimes referred to as market basket analysis.

Results validation

The final step of knowledge discovery from data is to verify the patterns produced by the data mining algorithms occur in the wider data set. Not all patterns found by the data mining algorithms are necessarily valid. It is common for the data mining algorithms to find patterns in the training set which are not present in the general data set, this is called overfitting. To overcome this, the evaluation uses a test set of data which the data mining algorithm was not trained on. The learnt patterns are applied to this test set and the resulting output is compared to the desired output. For example, a data mining algorithm trying to distinguish spam from legitimate emails would be trained on a training set of sample emails. Once trained, the learnt patterns would be applied to the test set of emails which it had not been trained on, the accuracy of these patterns can then be measured from how many emails they correctly classify. A number of statistical methods may be used to evaluate the algorithm such as ROC curves.

If the learnt patterns do not meet the desired standards, then it is necessary to reevaluate and change the preprocessing and data mining. If the learnt patterns do meet the desired standards then the final step is to interpret the learnt patterns and turn them into knowledge.

Notable uses

Games

Since the early 1960s, with the availability of oracles for certain combinatorial games, also called tablebases (e.g. for 3x3-chess) with any beginning configuration, small-board dots-and-boxes, small-board-hex, and certain endgames in chess, dots-and-boxes, and hex; a new area for data mining has been opened up. This is the extraction of human-usable strategies from these oracles. Current pattern recognition approaches do not seem to fully have the required high level of abstraction in order to be applied successfully. Instead, extensive experimentation with the tablebases, combined with an intensive study of tablebase-answers to well designed problems and with knowledge of prior art, i.e. pre-tablebase knowledge, is used to yield insightful patterns. Berlekamp in dots-and-boxes etc. and John Nunn in chess endgames are notable examples of researchers doing this work, though they were not and are not involved in tablebase generation.

Business

Data mining in customer relationship management applications can contribute significantly to the bottom line. Rather than randomly contacting a prospect or customer through a call center or sending mail, a company can concentrate its efforts on prospects that are predicted to have a high likelihood of responding to an offer. More sophisticated methods may be used to optimise resources across campaigns so that one may predict which channel and which offer an individual is most likely to respond to — across all potential offers. Additionally, sophisticated applications could be used to automate the mailing. Once the results from data mining (potential prospect/customer and

channel/offer) are determined, this "sophisticated application" can either automatically send an e-mail or regular mail. Finally, in cases where many people will take an action without an offer, uplift modeling can be used to determine which people will have the greatest increase in responding if given an offer. Data clustering can also be used to automatically discover the segments or groups within a customer data set.

Businesses employing data mining may see a return on investment, but also they recognise that the number of predictive models can quickly become very large. Rather than one model to predict how many customers will churn, a business could build a separate model for each region and customer type. Then instead of sending an offer to all people that are likely to churn, it may only want to send offers to customers. And finally, it may also want to determine which customers are going to be profitable over a window of time and only send the offers to those that are likely to be profitable. In order to maintain this quantity of models, they need to manage model versions and move to *automated data mining*.

Data mining can also be helpful to human-resources departments in identifying the characteristics of their most successful employees. Information obtained, such as universities attended by highly successful employees, can help HR focus recruiting efforts accordingly. Additionally, Strategic Enterprise Management applications help a company translate corporate-level goals, such as profit and margin share targets, into operational decisions, such as production plans and workforce levels.

Another example of data mining, often called the market basket analysis, relates to its use in retail sales. If a clothing store records the purchases of customers, a data-mining system could identify those customers who favour silk shirts over cotton ones. Although some explanations of relationships may be difficult, taking advantage of it is easier. The example deals with association rules within transaction-based data. Not all data are transaction based and logical or inexact rules may also be present within a database. In a manufacturing application, an inexact rule may state that 73% of products which have a specific defect or problem will develop a secondary problem within the next six months.

Market basket analysis has also been used to identify the purchase patterns of the Alpha consumer. Alpha Consumers are people that play a key role in connecting with the concept behind a product, then adopting that product, and finally validating it for the rest of society. Analyzing the data collected on this type of users has allowed companies to predict future buying trends and forecast supply demands.

Data Mining is a highly effective tool in the catalog marketing industry. Catalogers have a rich history of customer transactions on millions of customers dating back several years. Data mining tools can identify patterns among customers and help identify the most likely customers to respond to upcoming mailing campaigns.

Related to an integrated-circuit production line, an example of data mining is described in the paper "Mining IC Test Data to Optimize VLSI Testing." In this paper the application of data mining and decision analysis to the problem of die-level functional test is

described. Experiments mentioned in this paper demonstrate the ability of applying a system of mining historical die-test data to create a probabilistic model of patterns of die failure which are then utilised to decide in real time which die to test next and when to stop testing. This system has been shown, based on experiments with historical test data, to have the potential to improve profits on mature IC products.

Science and engineering

In recent years, data mining has been widely used in area of science and engineering, such as bioinformatics, genetics, medicine, education and electrical power engineering.

In the area of study on human genetics, an important goal is to understand the mapping relationship between the inter-individual variation in human DNA sequences and variability in disease susceptibility. In lay terms, it is to find out how the changes in an individual's DNA sequence affect the risk of developing common diseases such as cancer. This is very important to help improve the diagnosis, prevention and treatment of the diseases. The data mining technique that is used to perform this task is known as multifactor dimensionality reduction.

In the area of electrical power engineering, data mining techniques have been widely used for condition monitoring of high voltage electrical equipment. The purpose of condition monitoring is to obtain valuable information on the insulation's health status of the equipment. Data clustering such as self-organizing map (SOM) has been applied on the vibration monitoring and analysis of transformer on-load tap-changers(OLTCS). Using vibration monitoring, it can be observed that each tap change operation generates a signal that contains information about the condition of the tap changer contacts and the drive mechanisms. Obviously, different tap positions will generate different signals. However, there was considerable variability amongst normal condition signals for exactly the same tap position. SOM has been applied to detect abnormal conditions and to estimate the nature of the abnormalities.

Data mining techniques have also been applied for dissolved gas analysis (DGA) on power transformers. DGA, as a diagnostics for power transformer, has been available for many years. Data mining techniques such as SOM has been applied to analyse data and to determine trends which are not obvious to the standard DGA ratio techniques such as Duval Triangle.

A fourth area of application for data mining in science/engineering is within educational research, where data mining has been used to study the factors leading students to choose to engage in behaviors which reduce their learning and to understand the factors influencing university student retention. A similar example of the social application of data mining is its use in expertise finding systems, whereby descriptors of human expertise are extracted, normalised and classified so as to facilitate the finding of experts, particularly in scientific and technical fields. In this way, data mining can facilitate Institutional memory.

Other examples of applying data mining technique applications are biomedical data facilitated by domain ontologies, mining clinical trial data, traffic analysis using SOM, et cetera.

In adverse drug reaction surveillance, the Uppsala Monitoring Centre has, since 1998, used data mining methods to routinely screen for reporting patterns indicative of emerging drug safety issues in the WHO global database of 4.6 million suspected adverse drug reaction incidents. Recently, similar methodology has been developed to mine large collections of electronic health records for temporal patterns associating drug prescriptions to medical diagnoses.

Spatial data mining

Spatial data mining is the application of data mining techniques to spatial data. Spatial data mining follows along the same functions in data mining, with the end objective to find patterns in geography. So far, data mining and Geographic Information Systems (GIS) have existed as two separate technologies, each with its own methods, traditions and approaches to visualization and data analysis. Particularly, most contemporary GIS have only very basic spatial analysis functionality. The immense explosion in geographically referenced data occasioned by developments in IT, digital mapping, remote sensing, and the global diffusion of GIS emphasises the importance of developing data driven inductive approaches to geographical analysis and modeling.

Data mining, which is the partially automated search for hidden patterns in large databases, offers great potential benefits for applied GIS-based decision-making. Recently, the task of integrating these two technologies has become critical, especially as various public and private sector organisations possessing huge databases with thematic and geographically referenced data begin to realise the huge potential of the information hidden there. Among those organisations are:

- offices requiring analysis or dissemination of geo-referenced statistical data
- public health services searching for explanations of disease clusters
- environmental agencies assessing the impact of changing land-use patterns on climate change
- geo-marketing companies doing customer segmentation based on spatial location.

Challenges

Geospatial data repositories tend to be very large. Moreover, existing GIS datasets are often splintered into feature and attribute components, that are conventionally archived in hybrid data management systems. Algorithmic requirements differ substantially for relational (attribute) data management and for topological (feature) data management. Related to this is the range and diversity of geographic data formats, that also presents unique challenges. The digital geographic data revolution is creating new types of data formats beyond the traditional "vector" and "raster" formats. Geographic data repositories increasingly include ill-structured data such as imagery and geo-referenced multi-media.

There are several critical research challenges in geographic knowledge discovery and data mining. Miller and Han offer the following list of emerging research topics in the field:

- **Developing and supporting geographic data warehouses** - Spatial properties are often reduced to simple aspatial attributes in mainstream data warehouses. Creating an integrated GDW requires solving issues in spatial and temporal data interoperability, including differences in semantics, referencing systems, geometry, accuracy and position.
- **Better spatio-temporal representations in geographic knowledge discovery** - Current geographic knowledge discovery (GKD) techniques generally use very simple representations of geographic objects and spatial relationships. Geographic data mining techniques should recognise more complex geographic objects (lines and polygons) and relationships (non-Euclidean distances, direction, connectivity and interaction through attributed geographic space such as terrain). Time needs to be more fully integrated into these geographic representations and relationships.
- **Geographic knowledge discovery using diverse data types** - GKD techniques should be developed that can handle diverse data types beyond the traditional raster and vector models, including imagery and geo-referenced multimedia, as well as dynamic data types (video streams, animation).

Surveillance

Previous data mining to stop terrorist programs under the U.S. government include the Total Information Awareness (TIA) program, Secure Flight (formerly known as Computer-Assisted Passenger Prescreening System (CAPPS II)), Analysis, Dissemination, Visualization, Insight, Semantic Enhancement (ADVISE), and the Multi-state Anti-Terrorism Information Exchange (MATRIX). These programs have been discontinued due to controversy over whether they violate the US Constitution's 4th amendment, although many programs that were formed under them continue to be funded by different organisations, or under different names.

Two plausible data mining techniques in the context of combating terrorism include "pattern mining" and "subject-based data mining".

Pattern mining

"Pattern mining" is a data mining technique that involves finding existing patterns in data. In this context *patterns* often means association rules. The original motivation for searching association rules came from the desire to analyze supermarket transaction data, that is, to examine customer behaviour in terms of the purchased products. For example, an association rule "beer ⇒ crisps (80%)" states that four out of five customers that bought beer also bought crisps.

In the context of pattern mining as a tool to identify terrorist activity, the National Research Council provides the following definition: *"Pattern-based data mining looks for patterns (including anomalous data patterns) that might be associated with terrorist activity — these patterns might be regarded as small signals in a large ocean of noise."* Pattern Mining includes new areas such a Music Information Retrieval (MIR) where patterns seen both in the temporal and non temporal domains are imported to classical knowledge discovery search techniques.

Subject-based data mining

"Subject-based data mining" is a data mining technique involving the search for associations between individuals in data. In the context of combatting terrorism, the National Research Council provides the following definition: *"Subject-based data mining uses an initiating individual or other datum that is considered, based on other information, to be of high interest, and the goal is to determine what other persons or financial transactions or movements, etc., are related to that initiating datum."*

Privacy concerns and ethics

Some people believe that data mining itself is ethically neutral. However, the ways in which data mining can be used can raise questions regarding privacy, legality, and ethics. In particular, data mining government or commercial data sets for national security or law enforcement purposes, such as in the Total Information Awareness Program or in ADVISE, has raised privacy concerns.

Data mining requires data preparation which can uncover information or patterns which may compromise confidentiality and privacy obligations. A common way for this to occur is through data aggregation. Data aggregation is when the data are accrued, possibly from various sources, and put together so that they can be analyzed. This is not data mining per se, but a result of the preparation of data before and for the purposes of the analysis. The threat to an individual's privacy comes into play when the data, once compiled, cause the data miner, or anyone who has access to the newly compiled data set, to be able to identify specific individuals, especially when originally the data were anonymous.

It is recommended that an individual is made aware of the following before data are collected:

- the purpose of the data collection and any data mining projects,
- how the data will be used,
- who will be able to mine the data and use them,
- the security surrounding access to the data, and in addition,
- how collected data can be updated.

In the United States, privacy concerns have been somewhat addressed by their congress via the passage of regulatory controls such as the Health Insurance Portability and

Accountability Act (HIPAA). The HIPAA requires individuals to be given "informed consent" regarding any information that they provide and its intended future uses by the facility receiving that information. According to an article in Biotech Business Week, “In practice, HIPAA may not offer any greater protection than the longstanding regulations in the research arena, says the AAHC. More importantly, the rule's goal of protection through informed consent is undermined by the complexity of consent forms that are required of patients and participants, which approach a level of incomprehensibility to average individuals.” This underscores the necessity for data anonymity in data aggregation practices.

One may additionally modify the data so that they are anonymous, so that individuals may not be readily identified. However, even de-identified data sets can contain enough information to identify individuals, as occurred when journalists were able to find several individuals based on a set of search histories that were inadvertently released by AOL.

Marketplace surveys

Several researchers and organizations have conducted reviews of data mining tools and surveys of data miners. These identify some of the strengths and weaknesses of the software packages. They also provide an overview of the behaviors, preferences and views of data miners. Some of these reports include:

- Forrester Research 2010 Predictive Analytics and Data Mining Solutions report.
- Annual Rexer Analytics Data Miner Surveys.
- Gartner 2008 "Magic Quadrant" report.
- Robert Nisbet's 2006 Three Part Series of articles "Data Mining Tools: Which One is Best For CRM?"
- Haughton et al.'s 2003 Review of Data Mining Software Packages in The American Statistician.

Groups and associations

- SIGKDD, the ACM Special Interest Group on Knowledge Discovery and Data Mining.

Chapter- 16

Constraint Satisfaction

In artificial intelligence and operations research, **constraint satisfaction** is the process of finding a solution to a set of constraints that impose conditions that the variables must satisfy. A solution is therefore a vector of variables that satisfies all constraints.

The techniques used in constraint satisfaction depend on the kind of constraints being considered. Often used are constraints on a finite domain, to the point that constraint satisfaction problems are typically identified with problems based on constraints on a finite domain. Such problems are usually solved via search, in particular a form of backtracking or local search. Constraint propagation are other methods used on such problems; most of them are incomplete in general, that is, they may solve the problem or prove it unsatisfiable, but not always. Constraint propagation methods are also used in conjunction with search to make a given problem simpler to solve. Other considered kinds of constraints are on real or rational numbers; solving problems on these constraints is done via variable elimination or the simplex algorithm.

Constraint satisfaction originated in the field of artificial intelligence in the 1970s. During the 1980s and 1990s, embedding of constraints into a programming language were developed. Languages often used for constraint programming are Prolog and C++.

Constraint satisfaction problem

As originally defined in artificial intelligence, constraints enumerate the possible values a set of variables may take. Informally, a finite domain is a finite set of arbitrary elements. A constraint satisfaction problem on such domain contains a set of variables whose values can only be taken from the domain, and a set of constraints, each constraint specifying the allowed values for a group of variables. A solution to this problem is an evaluation of the variables that satisfies all constraints. In other words, a solution is a way for assigning a value to each variable in such a way that all constraints are satisfied by these values.

In practice, constraints are often expressed in compact form, rather than enumerating all values of the variables that would satisfy the constraint. One of the most used constraints is the one establishing that the values of the affected variables must be all different.

Problems that can be expressed as constraint satisfaction problems are the Eight queens puzzle, the Sudoku solving problem, the Boolean satisfiability problem, scheduling problems and various problems on graphs such as the graph coloring problem.

While usually not included in the above definition of a constraint satisfaction problem, arithmetic equations and inequalities bound the values of the variables they contain and can therefore be considered a form of constraints. Their domain is the set of numbers (either integer, rational, or real), which is infinite: therefore, the relations of these constraints may be infinite as well; for example, $X = Y + 1$ has an infinite number of pairs of satisfying values. Arithmetic equations and inequalities are often not considered within the definition of a "constraint satisfaction problem", which is limited to finite domains. They are however used often in constraint programming.

Constraint satisfaction problems or CSPs are mathematical problems defined as a set of objects whose state must satisfy a number of *constraints* or limitations. CSPs represent the entities in a problem as a homogeneous collection of finite constraints over variables, which is solved by constraint satisfaction methods. CSPs are the subject of intense research in both artificial intelligence and operations research, since the regularity in their formulation provides a common basis to analyze and solve problems of many unrelated families. CSPs often exhibit high complexity, requiring a combination of heuristics and combinatorial search methods to be solved in a reasonable time.

Examples of problems that can be modeled as a constraint satisfaction problem:

- Eight queens puzzle
- Map coloring problem
- Sudoku
- Boolean satisfiability

Formal definition

Formally, a constraint satisfaction problem is defined as a triple $\langle X, D, C \rangle$, where X is a set of variables, D is a domain of values, and C is a set of constraints. Every constraint is in turn a pair $\langle t, R \rangle$(usually represented as a matrix), where t is a tuple of variables and R is a set of tuples of values; all these tuples having the same number of elements; as a result R is a relation. An evaluation of the variables is a function from variables to values, $v : X \to D$. Such an evaluation satisfies a constraint $\langle (x_1, \ldots, x_n), R \rangle$if $(v(x_1), \ldots, v(x_n)) \in R$. A solution is an evaluation that satisfies all constraints.

Resolution of CSPs

Constraint satisfaction problems on finite domains are typically solved using a form of search. The most used techniques are variants of backtracking, constraint propagation, and local search.

Backtracking is a recursive algorithm. It maintains a partial assignment of the variables. Initially, all variables are unassigned. At each step, a variable is chosen, and all possible values are assigned to it in turn. For each value, the consistency of the partial assignment with the constraints is checked; in case of consistency, a recursive call is performed. When all values have been tried, the algorithm backtracks. In this basic backtracking algorithm, consistency is defined as the satisfaction of all constraints whose variables are all assigned. Several variants of backtracking exists. Backmarking improves the efficiency of checking consistency. Backjumping allows saving part of the search by backtracking "more than one variable" in some cases. Constraint learning infers and saves new constraints that can be later used to avoid part of the search. Look-ahead is also often used in backtracking to attempt to foresee the effects of choosing a variable or a value, thus sometimes determining in advance when a subproblem is satisfiable or unsatisfiable.

Constraint propagation techniques are methods used to modify a constraint satisfaction problem. More precisely, they are methods that enforce a form of local consistency, which are conditions related to the consistency of a group of variables and/or constraints. Constraints propagation has various uses. First, they turn a problem into one that is equivalent but is usually simpler to solve. Second, they may prove satisfiability or unsatisfiability of problems. This is not guaranteed to happen in general; however, it always happens for some forms of constraint propagation and/or for some certain kinds of problems. The most known and used form of local consistency are arc consistency, hyper-arc consistency, and path consistency. The most popular constraint propagation method is the AC-3 algorithm, which enforces arc consistency.

Local search methods are incomplete satisfiability algorithms. They may find a solution of a problem, but they may fail even if the problem is satisfiable. They work by iteratively improving a complete assignment over the variables. At each step, a small number of variables are changed value, with the overall aim of increasing the number of constraints satisfied by this assignment. The min-conflicts algorithm is a local search algorithm specific for CSPs and based in that principle. In practice, local search appears to work well when these changes are also affected by random choices. Integration of search with local search have been developed, leading to hybrid algorithms.

Theoretical aspects of CSPs

CSPs are also studied in computational complexity theory and finite model theory. An important question is whether for each set of relations, the set of all CSPs that can be represented using only relations chosen from that set is either in PTIME or otherwise NP-complete (assuming $P \neq NP$). If such a dichotomy is true, then CSPs provide one of the largest known subsets of NP which avoids problems that are neither polynomial time solvable nor NP-complete, whose existence was demonstrated by Ladner. Dichotomy results are known for CSPs where the domain of values is of size 2 or 3, but the general case is still open.

Most classes of CSPs that are known to be tractable are those where the hypergraph of constraints has bounded treewidth (and there are no restrictions on the set of constraint

relations), or where the constraints have arbitrary form but there exist essentially non-unary polymorphisms of the set of constraint relations.

Every CSP can also be considered as a conjunctive query containment problem.

Variants of CSPs

The classic model of Constraint Satisfaction Problem defines a model of static, inflexible constraints. This rigid model is a shortcoming that makes it difficult to represent problems easily. Several modifications of the basic CSP definition have been proposed to adapt the model to a wide variety of problems.

Dynamic CSPs

Dynamic CSPs (*DCSP*s) are useful when the original formulation of a problem is altered in some way, typically because the set of constraints to consider evolves because of the environment. DCSPs are viewed as a sequence of static CSPs, each one a transformation of the previous one in which variables and constraints can be added (restriction) or removed (relaxation). Information found in the initial formulations of the problem can be used to refine the next ones. The solving method can be classified according to the way in which information is transferred:

- Oracles: the solution found to previous CSPs in the sequence are used as heuristics to guide the resolution of the current CSP from scratch.
- Local repair: each CSP is calculated starting from the partial solution of the previous one and repairing the inconsistent constraints with local search.
- Constraint recording: new constraints are defined in each stage of the search to represent the learning of inconsistent group of decisions. Those constraints are carried over the new CSP problems.

Flexible CSPs

Classic CSPs treat constraints as hard, meaning that they are *imperative* (each solution must satisfy all them) and *inflexible* (in the sense that they must be completely satisfied or else they are completely violated). **Flexible CSP**s relax those assumptions, partially *relaxing* the constraints and allowing the solution to not comply with all them. This is similar to preferences in preference-based planning. Some types of flexible CSPs include:

- MAX-CSP, where a number of constraints are allowed to be violated, and the quality of a solution is measured by the number of satisfied constraints.
- Weighted CSP, a MAX-CSP in which each violation of a constraint is weighted according to a predefined preference. Thus satisfying constraint with more weight is preferred.
- Fuzzy CSP model constraints as fuzzy relations in which the satisfaction of a constraint is a continuous function of its variables' values, going from fully satisfied to fully violated.

Solving

Constraint satisfaction problems on finite domains are typically solved using a form of search. The most used techniques are variants of backtracking, constraint propagation, and local search. These techniques are used on problems with nonlinear constraints.

Variable elimination and the simplex algorithm are used for solving linear and polynomial equations and inequalities, and problems containing variables with infinite domain. These are typically solved as optimization problems in which the optimized function is the number of violated constraints.

Complexity

Solving a constraint satisfaction problem on a finite domain is an NP complete problem. Research has shown a number of tractable subcases, some limiting the allowed constraint relations, some requiring the scopes of constraints to form a tree, possibly in a reformulated version of the problem. Research has also established relationship of the constraint satisfaction problem with problems in other areas such as finite model theory.

The complexity of constraint satisfaction is the application of computational complexity theory on constraint satisfaction. It has mainly been studied for discriminating between tractable and intractable classes of constraint satisfaction problems on finite domains.

Solving a constraint satisfaction problem on a finite domain is an NP-complete problem in general. Research has shown a number of polynomial-time subcases, mostly obtained by restricting either the allowed domains or constraints or the way constraints can be placed over the variables. Research has also established relationship of the constraint satisfaction problem with problems in other areas such as finite model theory and databases.

Overview

Establishing whether a constraint satisfaction problem on a finite domain has solutions is an NP complete problem in general. This is an easy consequence of a number of other NP complete problems being expressible as constraint satisfaction problems. Such other problems include propositional satisfiability and three-colorability.

Tractability can be obtained by considering specific classes of constraint satisfaction problems. As an example, if the domain is binary and all constraints are binary, establishing satisfiability is a polynomial-time problem because this problem is equivalent to 2-SAT, which is tractable. Research has shown a number of tractable subcases. Some of these classes are based on restricting the allowed domains or relations, some on restricting the way constraints are placed over variables, and some on both kinds of restrictions.

One line of research used a correspondence between constraint satisfaction problem and the problem of establishing the existence of a homomorphism between two relational structures. This correspondence has been used to link constraint satisfaction with topics traditionally related to database theory.

A considered research problem is about the existence of dichotomies among sets of restrictions. This is the question of whether a set of restrictions contains only polynomial-time restrictions and NP-complete restrictions. This question is settled for some sets of restrictions, but still open for the set of all restrictions based on a fixed domain and set of relations, as of 2007. This is considered by some authors the most important open question about the complexity of constraint satisfaction.

Restrictions

Tractable subcases of the general constraint satisfaction problems can be obtained by placing suitable restrictions on the problems. Various kinds of restrictions have been considered.

Structural and relational restrictions

Tractability can be obtained by restricting the possible domains or constraints. In particular, two kinds of restrictions have been considered:

- *relational restrictions* bounds the domain and the values satisfying the constraints;
- *structural restrictions* bounds the way constraints are distributed over the variables.

More precisely, a relational restriction specifies a *constraint language*, which is a domain and a set of relations over this domain. A constraint satisfaction problem meets this restriction if it has exactly this domain and the relation of each constraint is in the given set of relations. In other words, a relational restriction bounds the domain and the set of satisfying values of each constraints, but not how the constraints are placed over variables. This is instead done by structural restrictions. Structural restriction can be checked by looking only at the scopes of constraints (their variables), ignoring their relations (their set of satisfying values).

A constraint language is tractable if there exists a polynomial algorithm solving all problems based on the language, that is, using the domain and relations specified in the domain. An example of a tractable constraint language is that of binary domains and binary constraints. Formally, this restriction corresponds to allowing only domains of size 2 and only constraints whose relation is a binary relation. While the second fact implies that the scopes of the constraints are binary, this is not a structural restriction because it does not forbid any constraint to be placed on an arbitrary pair of variables. Incidentally, the problem becomes NP complete if either restriction is lifted: binary constraints and

ternary domains can express the graph coloring problem, while ternary constraints and binary domains can express 3-SAT; these two problems are both NP-complete.

An example of a tractable class defined in terms of a structural restriction is that of binary acyclic problems. Given a constraint satisfaction problem with only binary constraints, its associated graph has a vertex for every variable and an edge for every constraint; two vertices are joined if they are in a constraint. If the graph of a problem is acyclic, the problem is called acyclic as well. The problem of satisfiability on the class of binary acyclic problem is tractable. This is a structural restriction because it does not place any limit to the domain or to the specific values that satisfy a constraints; rather, it restricts the way constraints are placed over variables.

While relational and structural restrictions are the ones mostly used to derive tractable classes of constraint satisfaction, there are some tractable classes that cannot be defined by relational restrictions only or structural restrictions only. The tractable class defined in terms of row convexity cannot be defined only in terms of the relations or only in terms of the structure, as row convexity depends both on the relations and on the order of variables (and therefore cannot be checked by looking only at each constraint in turn).

Uniform and non-uniform restrictions

The subcase obtained by restricting to a finite constraint language is called a *non-uniform problem*. These problems are mostly considered when expressing constraint satisfaction in terms of the homomorphism problem, as explained below. *Uniform problems* were also defined in the settings of homomorphism problems; a uniform problem can be defined as the union of a (possibly infinite) collection of non-uniform problems. A uniform problem made of an infinite set of non-uniform problems can be intractable even if all these non-uniform problems are tractable.

Tree-based restrictions

Some considered restrictions are based on the tractability of the constraint satisfaction problem where the constraints are all binary and form a tree over the variables. This is a structural restriction, as it can be checked by looking only at the scopes of the constraints, ignoring domains and relations.

This restriction is based on primal graph of the problem, which is a graph whose vertices are the variables of the problem and the edges represent the presence of a constraint between two variables. Tractability can however also be obtained by placing the condition of being a tree to the primal graph of problems that are reformulations of the original one.

Equivalence conditions

Constraint satisfaction problems can be reformulated in terms of other problems, leading to equivalent conditions to tractability. The most used reformulation is that in terms of the homomorphism problem.

Constraint satisfaction and the homomorphism problem

A link between constraint satisfaction and database theory has been provided in the form of a correspondence between the problem of constraint satisfiability to the problem of checking whether there exists a homomorphism between two relational structures. A relational structure is a mathematical representation of a database: it is a set of values and a set of relations over these values. Formally, $A = (V, R_1^A, \ldots, R_n^A)$, where each R_i^A is a relation over *V*, that is, a set of tuples of values of *V*.

A relational structure is different from a constraint satisfaction problem because a constraint is a relation *and* a tuple of variables. Also different is the way in which they are used: for a constraint satisfaction problem, finding a satisfying assignment is the main problem; for a relation structure, the main problem is finding the answer to a query.

The constraint satisfaction problem is however related to the problem of establishing the existence of a homomorphism between two relational structures. A homomorphism is a function from the values of the first relation to the values of the second that, when applied to all values of a relation of the first structure, turns it into a subset of the corresponding relation of the second structure. Formally, *h* is a homomorphism from $A = (V, R_1^A, \ldots, R_n^A)$ to $B = (D, R_1^B, \ldots, R_n^B)$ if it is a function from *V* to *D* such that, if $(x_1, \ldots, x_k) \in R_i^A$ then $(h(x_1), \ldots, h(x_k)) \in R_i^B$.

A direct correspondence between the constraint satisfaction problem and the homomorphism problem can be established. For a given constraint satisfaction problem, one can build a pair of relational structures, the first encoding the variables and the signatures of constraints, the second encoding the domains and the relations of the constraints. Satisfiability of the constraint satisfaction problem corresponds to finding a value for every variable such that replacing a value in a signature makes it a tuple in the relation of the constraint. This is possible exactly if this evaluation is a homomorphism between the two relational structures.

The inverse correspondence is the opposite one: given two relational structures, one encodes the values of the first in the variables of a constraint satisfaction problem, and the values of the second in the domain of the same problem. For every tuple of every relation of the first structure, there is a constraint having as values the correspondent relation of the first structure. This way, a homomorphism corresponds to mapping every scope of every constraint (every tuple of every relation of the first structure) into a tuple

in the relation of the constraint (a tuple in the corresponding relation of the second structure).

A non-uniform constraint satisfaction problem is a restriction where the second structure of the homomorphism problem is fixed. In other words, every relational structure defines a non-uniform problem, that of telling whether a relation structure is homomorphic to it. A similar restriction can be placed on the first structure; for any fixed first structure, the homomorphism problem is tractable. A uniform constraint satisfaction problem is an arbitrary restriction to the sets of structures for the first and second structure of the homomorphism problem.

Conjunctive query evaluation and containment

Since the homomorphism problem is equivalent to conjunctive query evaluation and conjunctive query containment, these two problems are equivalent to constraint satisfaction as well.

Join evaluation

Every constraint can be viewed as a table in a database, where the variables are interpreted as attributes names and the relation is the set of records in the table. The solutions of a constraint satisfaction problem are the result of an inner join of the tables representing its constraints; therefore, the problem of existence of solutions can be reformulated as the problem of checking whether the result of an inner join of a number of tables is empty.

Dichotomy theorems

Some constraint languages (or non-uniform problems) are known to correspond to problems solvable in polynomial time, and some others are known to express NP-complete problems. However, it is possible that some constraint languages are neither. It is known by Ladner's theorem that if P is not equal to NP, then there exist problems in NP that are neither polynomial-time nor NP-hard. As of 2007, it is not known if such problems can be expressed as constraint satisfaction problems with a fixed constraint language. If Ladner languages were not expressible in this way, the set of all constraint languages could be divided exactly into those defining polynomial-time and those defining NP-complete problems; that is, this set would exhibit a dichotomy.

Partial results are known for some sets of constraint languages. The best known such result is Schaefer's dichotomy theorem, which proves the existence of a dichotomy in the set of constraint languages on a binary domain. More precisely, it proves that a relation restriction on a binary domain is tractable if all its relations belong to one of six classes, and is NP-complete otherwise. Bulatov proved a dichotomy theorem for domains of three elements.

Another dichotomy theorem for constraint languages is the Hell-Nesetril theorem, which shows a dichotomy for problems on binary constraints with a single fixed symmetric relation. In terms of the homomorphism problem, every such problem is equivalent to the existence of a homomorphism from a relational structure to a given fixed undirected graph (an undirected graph can be regarded as a relational structure with a single binary symmetric relation). The Hell-Nesetril theorem proves that every such problem is either polynomial-time or NP-complete. More precisely, the problem is polynomial-time if the graph is 2-colorable, that is, it is bipartite, and is NP-complete otherwise.

Sufficient conditions for tractability

Some complexity results prove that some restrictions are polynomial without giving proving that all other possible restrictions of the same kind are NP-hard.

Datalog

A sufficient condition for tractability is related to expressibility in Datalog. A Boolean Datalog query gives a truth value to a set of literals over a given alphabet, each literal being an expression of the form $L(a_1, \ldots, a_n)$; as a result, a Boolean Datalog query expresses a set of sets of literals, as it can be considered semantically equivalent to the set of all sets of literals that it evaluates to true.

On the other hand, a non-uniform problem can be seen as a way for expressing a similar set. For a given non-uniform problem, the set of relations that can be used in constraints is fixed; as a result, one can give unique names $R_1, \ldots, R_n$ to them. An instance of this non-uniform problem can be then written as a set of literals of the form $R_i(x_1, \ldots, x_n)$. Among these instances/sets of literals, some are satisfiable and some are not; whether a set of literals is satisfiable depends on the relations, which are specified by the non-uniform problem. In the other way around, a non-uniform problem tells which sets of literals represent satisfiable instances and which ones represent unsatisfiable instances. Once relations are named, a non-uniform problem expresses a set of sets of literals: those associated to satisfiable (or unsatisfiable) instances.

A sufficient condition of tractability is that a non-uniform problem is tractable if the set of its unsatisfiable instances can be expressed by a Boolean Datalog query. In other words, if the set of sets of literals that represent unsatisfiable instances of the non-uniform problem is also the set of sets of literals that satisfy a Boolean Datalog query, then the non-uniform problem is tractable.

Local consistency

Satisfiability can sometimes be established by enforcing a form of local consistency and then checking the existence of an empty domain or constraint relation. This is in general a correct but incomplete unsatisfiability algorithm: a problem may be unsatisfiable even if no empty domain or constraint relation is produced. For some forms of local consistency,

this algorithm may also require exponential time. However, for some problems and for some kinds of local consistency, it is correct and polynomial-time.

The following conditions exploit the primal graph of the problem, which has a vertex for each variable and an edge between two nodes if the corresponding variables are in a constraint. The following are conditions on binary constraint satisfaction problems where enforcing local consistency is tractable and allows establishing satisfiability:

1. enforcing arc consistency, if the primal graph is acyclic;
2. enforcing directional arc consistency for an ordering of the variables that makes the ordered graph of constraint having width 1 (such an ordering exists if and only if the primal graph is a tree, but not all orderings of a tree generate width 1);
3. enforcing strong directional path consistency for an ordering of the variables that makes the primal graph having induced width 2.

A condition that extends the last one holds for non-binary constraint satisfaction problems. Namely, for all problems for which there exists an ordering that makes the primal graph having induced width bounded by a constant i, enforcing strong directional i-consistency is tractable and allows establishing satisfiability.

Tree-based conditions

Constraint satisfaction problems composed of binary constraints only can be viewed as graphs, where the vertices are variables and the edges represent the presence of a constraint between two variables. This graph is called the Gaifman graph or primal constraint graph (or simply primal graph) of the problem.

If the primal graph of a problem is acyclic, establishing satisfiability of the problem is a tractable problem. This is a structural restriction, as it can be checked by looking only at the scopes of the constraints, disregarding their relations and the domain. An acyclic graph is a forest, but connectedness is usually assumed; as a result, the condition that is usually considered is that primal graphs are trees.

This property of tree-like constraint satisfaction problems is exploited by decomposition methods, which convert problems into equivalent ones that only contain binary constraints arranged as a tree. The variables of these problems correspond to sets of variables of the original problem; the domain of such a variable is obtained by considering some of the constraints of the original problem whose scope is contained in the corresponding original set of variables; constraints of these new problems represent equality of variables that are contained in two sets.

If the graph of one such equivalent problem is a tree, the problem can be solved efficiently. On the other hand, producing one such equivalent problem may be not be efficient because of two factors: the need to determine the combined effects of a group of constraints over a set of variables, and the need to store all tuples of values that satisfy a given group of constraints.

Necessary condition for tractability

A necessary condition for the tractability of a constraint language based on the universal gadget has been proved. The universal gadget is a particular constraint satisfaction problem that was initially defined for the sake of expressing new relations by projection.

The universal gadget

A relation that is not present in a constraint language may be "simulated" by constraints using the relations in the language. In particular, a new relation can be created by placing a set of constraints and using only some of their variables. If all other constraints use only these variables, this set of constraints forces these variable to only assume specific values, practically simulating a new relation.

Every constraint satisfaction problem and subset of its variables defines a relation, which is composed by all tuples of values of the variables that can be extended to the other variables to form a solution. Technically, this relation is obtained by *projecting* the relation having the solutions as rows over the considered variables.

The universal gadget is based on the observation that every relation that contains *k*-tuples can be defined by projecting a relation that contains all possible columns of *k* elements from the domain. As an example, the following tables shows such a projection:

```
a b c d e f g h             b d
---------------             ---
1 1 1 1 0 0 0 0      ->     1 1
1 1 0 0 1 1 0 0             1 0
1 0 1 0 1 0 1 0             0 0
```

If the table on the left is the set of solutions of a constraint satisfaction problem, its variables *b* and *d* are constrained to the values of the table to the right. As a result, the constraint satisfaction problem can be used to set a constraint whose relation is the table on the right, which may not be in the constraint language.

As a result, if a constraint satisfaction problem has the table on the left as its set of solutions, every relation can be expressed by projecting over a suitable set of variables. A way for trying to obtain this table as the set of solution is to place every possible constraint that is not violated by the required solutions.

As an example, if the language contains the binary relation representing the Boolean disjunction (a relation containing all tuples of two elements that contains at least a 1), this relation is placed as a constraint on *a* and *b*, because their values in the table above are (1,1), (1,1) again, and (1,0). Since all these values satisfy the constraint, the constraint is placed. On the other hand, a constraint with this relation is not placed on *b* and *d*, since the restriction of the table above to these two variables contains (0,0) as a third row, and this evaluation violates that constraint.

The universal gadget of order *k* is the constraint satisfaction problem containing all constraints that can be placed in order to obtain the table above. The solutions of the universal gadget include the rows of this table, but can contain other rows. If the solutions are exactly the rows of the table, every relation can be expressed by projecting on a subset of the variables. However, even if the solutions include other rows, some relations can still be expressed. A property of the universal gadget is that it is able to express, by projection, every relation that can be expressed by projection from an arbitrary constraint satisfaction problem based on the same language. More precisely, the universal gadget of order *k* expresses all relations of *k* rows that can be expressed in the constraint language.

Given a specific relation, its expressibility in the language can be checked by considering an arbitrary list of variables whose columns in the table above (the "ideal" solutions to the universal gadget) form that relation. The relation can be expressed in the language if and only if the solutions of the universal gadget coincides with the relation when projected over such a list of variables. In other words, one can check expressibility by selecting variables "as if" the solutions of the universal gadget were like in the table, and then check whether the restriction of the "real" solutions is actually the same as the relation. In the example above, the expressibility of the relation in the table on the right can be checked by looking whether the solutions of the universal gadget, when restricted to the variables *b* and *d*, are exactly the rows of this table.

Solutions as functions in the universal gadget

A necessary condition for tractability can be expressed in terms of the universal gadget. The solutions of such a gadget can be tabulated as follows:

```
a b c d e f g h
---------------
1 1 1 1 0 0 0 0
1 1 0 0 1 1 0 0    (solutions that exist by definition)
1 0 1 0 1 0 1 0
---------------
....
1 0 0 1 1 1 0 0    (other solutions are possible)
....
```

This table is made of two parts. The first part contains the solutions that exist by definition of this problem; the second part (that may be empty) contains all other solutions). Since the columns of the table are by definition associated to the possible *k*-tuples of values of the domain, every solution can be viewed as a function from a *k*-tuple of elements to a single element.

The function corresponding to a solution can be calculated from the first part of the table above and the solution. As an example, for the last solution marked in the table, this function can be determined for arguments 1,0,1 as follows: first, these three values are the first part of the row "c" in the table; the value of the function is the value of the solution in the same column, that is, 0.

A necessary condition for tractability is the existence of a solution for a universal gadget of some order that is part of some classes of functions. This result however only holds for reduced languages, which are defined below.

Squashing functions and reduced domains

Squashing functions are functions used to reduce the size of domain of constraint languages. A squashing function is defined in terms of a partition of the domain and a *representative* element for each set in the partition. The squashing function maps all elements of a set in the partition to the representative element of that set. For such a function being a squashing function it is also necessary that applying the function to all elements of a tuple of a relation in the language produces another tuple in the relation. The partition is assumed to contain at least a set of size greater than one.

Formally, given a partition $D_1, \ldots, D_n$ of the domain D containing at least a set of size greater than one, a squashing function is a function $s : D \to D$ such that $s(x) = s(y)$ for every x,y in the same partition, and for every tuple $\langle x_1, \ldots, x_n \rangle \in R$, it holds $\langle s(x_1), \ldots, s(x_n) \rangle \in R$.

For constraint problems on a constraint language has a squashing function, the domain can be reduced via the squashing function. Indeed, every element in a set in the partition can be replaced with the result of applying the squashing function to it, as this result is guaranteed to satisfy at least all constraints that were satisfied by the element. As a result, all non-representative elements can be removed from the constraint language.

Constraint languages for which no squashing function exist are called reduced languages; equivalently, these are languages on which all reductions via squashing functions have been applied.

The necessary condition for tractability

The necessary condition for tractability based on the universal gadget holds for reduced languages. Such a language is tractable if the universal gadget has a solution that, when viewed as a function in the way specified above, is either a constant function, a majority function, an idempotent binary function, an affine function, or a semi-projection.

Chapter- 17

Constraint Programming

Constraint programming is the use of constraints as a programming language to encode and solve problems. This is often done by embedding constraints into a programming language, which is called the host language. Constraint programming originated from a formalization of equalities of terms in Prolog II, leading to a general framework for embedding constraints into a logic programming language. The most common host languages are Prolog, C++, and Java, but other languages have been used as well.

Constraint programming is a programming paradigm wherein relations between variables are stated in the form of constraints. Constraints differ from the common primitives of imperative programming languages in that they do not specify a step or sequence of steps to execute, but rather the properties of a solution to be found. This makes constraint programming a form of declarative programming. The constraints used in constraint programming are of various kinds: those used in constraint satisfaction problems (e.g. "A or B is true"), those solved by the simplex algorithm (e.g. "x <= 5"), and others. Constraints are usually embedded within a programming language or provided via separate software libraries.

Constraint programming began with constraint logic programming, which embeds constraints into a logic program. This variant of logic programming is due to Jaffar and Lassez, who extended in 1987 a specific class of constraints that were introduced in Prolog II. The first implementations of constraint logic programming were Prolog III, CLP(R), and CHIP. Several constraint logic programming interpreters exist today, for example GNU Prolog.

Other than logic programming, constraints can be mixed with functional programming, term rewriting, and imperative languages. Programming languages with built-in support for constraints include Oz (functional programming) and Kaleidoscope (imperative programming). Mostly, constraints are implemented in imperative languages via *constraint solving toolkits*, which are separate libraries for an existing imperative language.

Constraint logic programming

Constraint programming is an embedding of constraints in a host language. The first host languages used were logic programming languages, so the field was initially called

constraint logic programming. The two paradigms share many important features, like logical variables and backtracking. Today most Prolog implementations include one or more libraries for constraint logic programming.

The difference between the two is largely in their styles and approaches to modeling the world. Some problems are more natural (and thus, simpler) to write as logic programs, while some are more natural to write as constraint programs.

The constraint programming approach is to search for a state of the world in which a large number of constraints are satisfied at the same time. A problem is typically stated as a state of the world containing a number of unknown variables. The constraint program searches for values for all the variables.

Temporal concurrent constraint programming (TCC) and non-deterministic temporal concurrent constraint programming (NTCC) are variants of constraint programming that can deal with time.

Some popular constraint logic languages are:

- B-Prolog (Prolog based, proprietary)
- CHIP V5 (Prolog based, also includes C++ and C libraries, proprietary)
- Ciao Prolog (Prolog based, Free software: GPL/LGPL)
- ECLiPSe (Prolog based, open source)
- SICStus (Prolog based, proprietary)
- GNU Prolog (free software)
- Oz
- YAP Prolog
- SWI Prolog a free Prolog system containing several libraries for constraint solving
- Claire
- Curry (Haskell based, with free implementations)
- Turtle (free software: GPL)

Domains

The constraints used in constraint programming are typically over some specific domains. Some popular domains for constraint programming are:

- boolean domains, where only true/false constraints apply (SAT problem)
- integer domains, rational domains
- linear domains, where only linear functions are described and analyzed (although approaches to non-linear problems do exist)
- finite domains, where constraints are defined over finite sets
- mixed domains, involving two or more of the above

Finite domains is one of the most successful domains of constraint programming. In some areas (like operations research) constraint programming is often identified with constraint programming over finite domains.

Finite domain solvers are useful for solving constraint satisfaction problems, and are often based on arc consistency or one of its approximations.

The syntax for expressing constraints over finite domains depends on the host language. The following is a Prolog program that solves the classical alphametic puzzle SEND+MORE=MONEY in constraint logic programming:

```
sendmore(Digits) :-
   Digits = [S,E,N,D,M,O,R,Y],      % Create variables
   Digits :: [0..9],                % Associate domains to variables
   S #\= 0,                         % Constraint: S must be different
from 0
   M #\= 0,
   alldifferent(Digits),            % all the elements must take
different values
              1000*S + 100*E + 10*N + D     % Other constraints
            + 1000*M + 100*O + 10*R + E
   #= 10000*M + 1000*O + 100*N + 10*E + Y,
   labeling(Digits).                % Start the search
```

The interpreter creates a variable for each letter in the puzzle. The symbol : is used to specify the domains of these variables, so that they range over the set of values {0,1,2,3, ..., 9}. The constraints S#\=0 and M#\=0 means that these two variables cannot take the value zero. When the interpreter evaluates these constraints, it reduces the domains of these two variables by removing the value 0 from them. Then, the constraint alldifferent(Digits) is considered; it does not reduce any domain, so it is simply stored. The last constraint specifies that the digits assigned to the letters must be such that "SEND+MORE=MONEY" holds when each letter is replaced by its corresponding digit. From this constraint, the solver infers that M=1. All stored constraints involving variable M are awakened: in this case, constraint propagation on the `alldifferent` constraint removes value 1 from the domain of all the remaining variables. Constraint propagation may solve the problem by reducing all domains to a single value, it may prove that the problem has no solution by reducing a domain to the empty set, but may also terminate without proving satisfiability or unsatisfiability. The **labeling** literals are used to actually perform search for a solution.

Constraint logic programming

A constraint logic program is a logic program that contains constraints in the bodies of clauses. As an example, the clause `A(X):-X>0,B(X)` is a clause containing the constraint `X>0` in the body. Constraints can also be present in the goal. The constraints in the goal and in the clauses used to prove the goal are accumulated into a set called constraint store. This set contains the constraints the interpreter has assumed satisfiable in order to proceed in the evaluation. As a result, if this set is detected unsatisfiable, the interpreter

backtracks. Equations of terms, as used in logic programming, are considered a particular form of constraints which can be simplified using unification. As a result, the constraint store can be considered an extension of the concept of substitution that is used in regular logic programming. The most common kinds of constraints used in constraint logic programming are constraints over integers/rational/real numbers and constraints over finite domains.

Concurrent constraint logic programming languages have also been developed. They significantly differ from non-concurrent constraint logic programming in that they are aimed at programming concurrent processes that may not terminate. Constraint handling rules can be seen as a form of concurrent constraint logic programming, but are also sometimes used within a non-concurrent constraint logic programming language. They allow for rewriting constraints or to infer new ones based on the truth of conditions.

Constraint logic programming is a form of constraint programming, in which logic programming is extended to include concepts from constraint satisfaction. A constraint logic program is a logic program that contains constraints in the body of clauses. An example of a clause including a constraint is `A(X,Y) :- X+Y>0, B(X), C(Y).` In this clause, `X+Y>0` is a constraint; `A(X,Y)`, `B(X)`, and `C(Y)` are literals as in regular logic programming. This clause states one condition under which the statement `A(X,Y)` holds: `X+Y` is greater than zero and both `B(X)` and `C(Y)` are true.

As in regular logic programming, programs are queried about the provability of a goal, which may contain constraints in addition to literals. A proof for a goal is composed of clauses whose bodies are satisfiable constraints and literals that can in turn be proved using other clauses. Execution is performed by an interpreter, which starts from the goal and recursively scans the clauses trying to prove the goal. Constraints encountered during this scan are placed in a set called **constraint store**. If this set is found out to be unsatisfiable, the interpreter backtracks, trying to use other clauses for proving the goal. In practice, satisfiability of the constraint store may be checked using an incomplete algorithm, which does not always detect inconsistency.

Overview

Formally, constraint logic programs are like regular logic programs, but the body of clauses can contain constraints, in addition to the regular logic programming literals. As an example, `X>0` is a constraint, and is included in the last clause of the following constraint logic program.

```
B(X,1):- X<0.
B(X,Y):- X=1, Y>0.
A(X,Y):- X>0, B(X,Y).
```

Like in regular logic programming, evaluating a goal such as `A(X,1)` requires evaluating the body of the last clause with `Y=1`. Like in regular logic programming, this in turn

requires proving the goal `B(X,1)`. Contrary to regular logic programming, this also requires a constraint to be satisfied: `X>0`, the constraint in the body of the last clause.

Whether a constraint is satisfied cannot always be determined when the constraint is encountered. In this case, for example, the value of `X` is not determined when the last clause is evaluated. As a result, the constraint `X>0` is not satisfied nor violated at this point. Rather than proceeding in the evaluation of `B(X,1)` and then checking whether the resulting value of `X` is positive afterwards, the interpreter stores the constraint `X>0` and then proceeds in the evaluation of `B(X,1)`; this way, the interpreter can detect violation of the constraint `X>0` during the evaluation of `B(X,1)`, and backtrack immediately if this is the case, rather than waiting for the evaluation of `B(X,1)` to conclude.

In general, the evaluation of a constraint logic program proceeds like for a regular logic program, but constraint encountered during evaluation are placed in a set called constraint store. As an example, the evaluation of the goal `A(X,1)` proceeds by evaluating the body of the first clause with `Y=1`; this evaluation adds `X>0` to the constraint store and requires the goal `B(X,1)` to be proved. While trying to prove this goal, the first clause is applicable, but its evaluation adds `X<0` to the constraint store. This addition makes the constraint store unsatisfiable, and the interpreter backtracks, removing the last addition from the constraint store. The evaluation of the second clause adds `X=1` and `Y>0` to the constraint store. Since the constraint store is satisfiable and no other literal is left to prove, the interpreter stops with the solution `X=1, Y=1`.

Semantics

The semantics of constraint logic programs can be defined in terms of a virtual interpreter that maintains a pair $\langle G, S \rangle$ during execution. The first element of this pair is called current goal; the second element is called constraint store. The *current goal* contains the literals the interpreter is trying to prove and may also contain some constraints it is trying to satisfy; the *constraint store* contains all constraints the interpreter has assumed satisfiable so far.

Initially, the current goal is the goal and the constraint store is empty. The interpreter proceeds by removing the first element from the current goal and analyzing it. The details of this analysis are explained below, but in the end this analysis may produce a successful termination or a failure. This analysis may involve recursive calls and addition of new literals to the current goal and new constraint to the constraint store. The interpreter backtracks if a failure is generated. A successful termination is generated when the current goal is empty and the constraint store is satisfiable.

The details of the analysis of a literal removed from the goal is as follows. After having removed this literal from the front of the goal, it is checked whether it is a constraint or a literal. If it is a constraint, it is added to the constraint store. If it is a literal, a clause whose head has the same predicate of the literal is chosen; the clause is rewritten by replacing its variables with new variables (variables not occurring in the goal): the result

is called a *fresh variant* of the clause; the body of the fresh variant of the clause is then placed in front of the goal; the equality of each argument of the literal with the corresponding one of the fresh variant head is placed in front of the goal as well.

Some checks are done during these operations. In particular, the constraint store is checked for consistency every time a new constraint is added to it. In principle, whenever the constraint store is unsatisfiable the algorithm could backtrack. However, checking unsatisfiability at each step would be inefficient. For this reason, an incomplete satisfiability checker may be used instead. In practice, satisfiability is checked using methods that simplify the constraint store, that is, rewrite it into an equivalent but simpler-to-solve form. These methods can sometimes but not always prove unsatisfiability of an unsatisfiable constraint store.

The interpreter has proved the goal when the current goal is empty and the constraint store is not detected unsatisfiable. The result of execution is the current set of (simplified) constraints. This set may include constraints such as $X = 2$ that force variables to a specific value, but may also include constraints like $X > 2$ that only bound variables without giving them a specific value.

Formally, the semantics of constraint logic programming is defined in terms of *derivations*. A transition is a pair of pairs goal/store, noted $\langle G, S\rangle \rightarrow \langle G', S'\rangle$. Such a pair states the possibility of going from state $\langle G, S\rangle$ to state $\langle G', S'\rangle$. Such a transition is possible in three possible cases:

- an element of G is a constraint C, $G' = G \backslash \{C\}$ and $S' = S \cup \{C\}$; in other words, a constraint can be moved from the goal to the constraint store
- an element of G is a literal $L(t_1, \ldots, t_n)$, there exists a clause that, rewritten using new variables, is $L(t'_1, \ldots, t'_n) :- B$, G' is G with $L(t_1, \ldots, t_n)$ replaced by $t_1 = t'_1, \ldots, t_n = t'_n, B$, and $S' = S$; in other words, a literal can be replaced by the body of a fresh variant of a clause having the same predicate in the head, adding the body of the fresh variant and the above equalities of terms to the goal
- S and S' are equivalent according to the specific constraint semantics

A sequence of transitions is a derivation. A goal G can be proved if there exists a derivation from $\langle G, \emptyset\rangle$ to $\langle \emptyset, S\rangle$ for some satisfiable constraint store S. This semantics formalizes the possible evolutions of an interpreter that arbitrarily chooses the literal of the goal to process and the clause to replace literals. In other words, a goal is proved under this semantics if there exists a sequence of choices of literals and clauses, among the possibly many ones, that lead to an empty goal and satisfiable store.

Actual interpreters process the goal elements in a LIFO order: elements are added in the front and processed from the front. They also choose the clause of the second rule

according to the order in which they are written, and rewrite the constraint store when it is modified.

The third possible kind of transition is a replacement of the constraint store with an equivalent one. This replacement is limited to those done by specific methods, such as constraint propagation. The semantics of constraint logic programming is parametric not only to the kind of constraints used but also to the method for rewriting the constraint store. The specific methods used in practice replace the constraint store with one that is simpler to solve. If the constraint store is unsatisfiable, this simplification may detect this unsatisfiability sometimes, but not always.

The result of evaluating a goal against a constraint logic program is defined if the goal is proved. In this case, there exists a derivation from the initial pair to a pair where the goal is empty. The constraint store of this second pair is considered the result of the evaluation. This is because the constraint store contains all constraints assumed satisfiable to prove the goal. In other words, the goal is proved for all variable evaluations that satisfy these constraints.

The pairwise equality of terms of two literals is often compactly denoted by $L(t_1, \ldots, t_n) = L(t'_1, \ldots, t'_n)$: this is a shorthand for the constraints $t_1 = t'_1, \ldots, t_n = t'_n$. A common variant of the semantics for constraint logic programming adds $L(t_1, \ldots, t_n) = L(t'_1, \ldots, t'_n)$ directly to the constraint store rather than to the goal.

Terms and constraints

Different definitions of terms are used, generating different kinds of constraint logic programming: over trees, reals, or finite domains. A kind of constraint that is always present is the equality of terms. Such constraints are necessary because the interpreter adds `t1=t2` to the goal whenever a literal `P(...t1...)` is replaced with the body of a clause fresh variant whose head is `P(...t2...)`.

Tree terms

Constraint logic programming with tree terms emulates regular logic programming by storing substitutions as constraints in the constraint store. Terms are variables, constants, and function symbols applied to other terms. The only considered constraints are equalities and disequalities between terms. Equality is particularly important, as constraints link `t1=t2` are often generated by the interpreter. Equality constraints on terms can be simplified, that is solved, via unification:

A constraint `t1=t2` can be simplified if both terms are function symbols applied to other terms. If the two function symbols are the same and the number of subterms is also the same, this constraint can be replaced with the pairwise equality of subterms. If the terms

are composed of different function symbols or the same functor but on different number of terms, the constraint is unsatisfiable.

If one of the two terms is a variable, the only allowed value the variable can take is the other term. As a result, the other term can replace the variable in the current goal and constraint store, thus practically removing the variable from consideration. In the particular case of equality of a variable with itself, the constraint can be removed as always satisfied.

In this form of constraint satisfaction, variable values are terms.

Reals

Constraint logic programming with real numbers uses real expressions as terms. When no function symbols are used, terms are expressions over reals, possibly including variables. In this case, each variable can only take a real number as a value.

To be precise, terms are expressions over variables and real constants. Equality between terms is a kind of constraint that is always present, as the interpreter generates equality of terms during execution. As an example, if the first literal of the current goal is `A(X+1)` and the interpreter has chosen a clause that is `A(Y-1):-Y=1` after rewriting is variables, the constraints added to the current goal are `X+1=Y-1` and $Y = 1$. The rules of simplification used for function symbols are obviously not used: `X+1=Y-1` is not unsatisfiable just because the first expression is built using `+` and the second using `-`.

Reals and function symbols can be combined, leading to terms that are expressions over reals and function symbols applied to other terms. Formally, variables and real constants are expressions, as any arithmetic operator over other expressions. Variables, constants (zero-arity-function symbols), and expressions are terms, as any function symbol applied to terms. In other words, terms are built over expressions, while expressions are built over numbers and variables. In this case, variables ranges over real numbers *and terms*. In other words, a variable can take a real number as a value, while another takes a term.

Equality of two terms can be simplified using the rules for tree terms if none of the two terms is a real expression. For example, if the two terms have the same function symbol and number of subterms, their equality constraint can be replaced with the equality of subterms.

Finite domains

The third class of constraints used in constraint logic programming is that of finite domains. Values of variables are in this case taken from a finite domain, often that of integer numbers. For each variable, a different domain can be specified: `X::[1..5]` for example means that the value of `X` is between `1` and `5`. The domain of a variable can also be given by enumerating all values a variable can take; therefore, the above domain declaration can be also written `X::[1,2,3,4,5]`. This second way of specifying a

domain allows for domains that are not composed of integers, such as `X::[george,mary,john]`. If the domain of a variable is not specified, it is assumed to be the set of integers representable in the language. A group of variables can be given the same domain using a declaration like `[X,Y,Z]::[1..5]`.

The domain of a variable may be reduced during execution. Indeed, as the interpreter adds constraints to the constraint store, it performs constraint propagation to enforce a form of local consistency, and these operations may reduce the domain of variables. If the domain of a variable becomes empty, the constraint store is inconsistent, and the algorithm backtracks. If the domain of a variable becomes a singleton, the variable can be assigned the unique value in its domain. The forms of consistency typically enforced are arc consistency, hyper-arc consistency, and bound consistency. The current domain of a variable can be inspected using specific literals; for example, `dom(X,D)` finds out the current domain `D` of a variable `X`.

As for domains of reals, functors can be used with domains of integers. In this case, a term can be an expression over integers, a constant, or the application of a functor over other terms. A variable can take an arbitrary term as a value, if its domain has not been specified to be a set of integers or constants.

The constraint store

The constraint store contains the constraints that are currently assumed satisfiable. It can be considered what the current substitution is for regular logic programming. When only tree terms are allowed, the constraint store contains constraints in the form `t1=t2`; these constraints are simplified by unification, resulting in constraints of the form `variable=term`; such constraints are equivalent to a substitution.

However, the constraint store may also contain constraints in the form `t1!=t2`, if the difference `!=` between terms is allowed. When constraints over reals or finite domains are allowed, the constraint store may also contain domain-specific constraints like `X+2=Y/2`, etc.

The constraint store extends the concept of current substitution in two ways. First, it does not only contain the constraints derived from equating a literal with the head of a fresh variant of a clause, but also the constraints of the body of clauses. Second, it does not only contain constraints of the form `variable=value` but also constraints on the considered constraint language. While the result of a successful evaluation of a regular logic program is the final substitution, the result for a constraint logic program is the final constraint store, which may contain constraint of the form variable=value but in general may contain arbitrary constraints.

Domain-specific constraints may come to the constraint store both from the body of a clauses and from equating a literal with a clause head: for example, if the interpreter rewrites the literal `A(X+2)` with a clause whose fresh variant head is `A(Y/2)`, the constraint `X+2=Y/2` is added to the constraint store. If a variable appears in a real or finite

domain expression, it can only take a value in the reals or the finite domain. Such a variable cannot take a term made of a functor applied to other terms as a value. The constraint store is unsatisfiable if a variable is bound to take both a value of the specific domain and a functor applied to terms.

After a constraint is added to the constraint store, some operations are performed on the constraint store. Which operations are performed depends on the considered domain and constraints. For example unification is used for finite tree equalities, variable elimination for polynomial equations over reals, constraint propagation to enforce a form of local consistency for finite domains. These operations are aimed at making the constraint store simpler to be checked for satisfiability and solved.

As a result of these operations, the addition of new constraints may change the old ones. It is essential that the interpreter is able to undo these changes when it backtracks. The simplest case method is for the interpreter to save the complete state of the store every time it makes a choice (it chooses a clause to rewrite a goal). More efficient methods for allowing the constraint store to return to a previous state exist. In particular, one may just save the changes to the constraint store made between two points of choice, including the changes made to the old constraints. This can be done by simply saving the old value of the constraints that have been modified; this method is called *trailing*. A more advanced method is to save the changes that have been done on the modified constraints. For example, a linear constraint is changed by modifying its coefficient: saving the difference between the old and new coefficient allows reverting a change. This second method is called *semantic backtracking*, because the semantics of the change is saved rather than the old version of the constraints only.

Labeling

The labeling literals are used on variables over finite domains to check satisfiability or partial satisfiability of the constraint store and to find a satisfying assignment. A labeling literal is of the form `labeling([variables])`, where the argument is a list of variables over finite domains. Whenever the interpreter evaluates such a literal, it performs a search over the domains of the variables of the list to find an assignment that satisfies all relevant constraints. Typically, this is done by a form of backtracking: variables are evaluated in order, trying all possible values for each of them, and backtracking when inconsistency is detected.

The first use of the labeling literal is to actual check satisfiability or partial satisfiability of the constraint store. When the interpreter adds a constraint to the constraint store, it only enforces a form of local consistency on it. This operation may not detect inconsistency even if the constraint store is unsatisfiable. A labeling literal over a set of variables enforces a satisfiability check of the constraints over these variables. As a result, using all variables mentioned in the constraint store results in checking satisfiability of the store.

The second use of the labeling literal is to actually determine an evaluation of the variables that satisfies the constraint store. Without the labeling literal, variables are assigned values only when the constraint store contains a constraint of the form `X=value` and when local consistency reduces the domain of a variable to a single value. A labeling literal over some variables forces these variables to be evaluated. In other words, after the labeling literal has been considered, all variables are assigned a value.

Typically, constraint logic programs are written in such a way labeling literals are evaluated only after as much constraints as possible have been accumulated in the constraint store. This is because labeling literals enforce search, and search is more efficient if there are more constraints to be satisfied. A constraint satisfaction problem is typical solved by a constraint logic program having the following structure:

```
solve(X):-constraints(X), labeling(X)
constraints(X):- (all constraints of the CSP)
```

When the interpreter evaluates the goal `solve(args)`, it places the body of a fresh variant of the first clause in the current goal. Since the first goal is `constraints(X')`, the second clause is evaluated, and this operation moves all constraints in the current goal and eventually in the constraint store. The literal `labeling(X')` is then evaluated, forcing a search for a solution of the constraint store. Since the constraint store contains exactly the constraints of the original constraint satisfaction problem, this operation searches for a solution of the original problem.

Program reformulations

A given constraint logic program may be reformulated to improve its efficiency. A first rule is that labeling literals should be placed after as much constraints on the labeled literals are accumulated in the constraint store. While in theory `A(X):-labeling(X),X>0` is equivalent to `A(X):-X>0,labeling(X)`, the search that is performed when the interpreter encounters the labeling literal is on a constraint store that does not contain the constraint `X>0`. As a result, it may generate solutions, such as `X=-1`, that are later found out not to satisfy this constraint. On the other hand, in the second formulation the search is performed only when the constraint is already in the constraint store. As a result, search only returns solutions that are consistent with it, taking advantage of the fact that additional constraints reduce the search space.

A second reformulation that can increase efficiency is to place constraints before literals in the body of clauses. Again, `A(X):-B(X),X>0` and `A(X):-X>0,B(X)` are in principle equivalent. However, the first may require more computation. For example, if the constraint store contains the constraint `X<-2`, the interpreter recursively evaluates `B(X)` in the first case; if it succeeds, it then finds out that the constraint store is inconsistent when adding `X>0`. In the second case, when evaluating that clause, the interpreter first adds `X>0` to the constraint store and then possibly evaluates `B(X)`. Since the constraint store after the addition of `X>0` turns out to be inconsistent, the recursive evaluation of `B(X)` is not performed at all.

A third reformulation that can increase efficiency is the addition of redundant constrains. If the programmer knows (by whatever means) that the solution of a problem satisfies a specific constraint, they can include that constraint to cause inconsistency of the constraint store as soon as possible. For example, if it is known beforehands that the evaluation of `B(X)` will result in a positive value for `X`, the programmer may add `X>0` before any occurrence of `B(X)`. As an example, `A(X,Y):-B(X),C(X)` will fail on the goal `A(-2,Z)`, but this is only found out during the evaluation of the subgoal `B(X)`. On the other hand, if the above clause is replaced by `A(X,Y):-X>0,A(X),B(X)`, the interpreter backtracks as soon as the constraint `X>0` is added to the constraint store, which happens before the evaluation of `B(X)` even starts.

Constraint handling rules

Constraint handling rules were initially defined as a stand-alone formalism for specifying constraint solvers, and were later embedded in logic programming. There are two kinds of constraint handling rules. The rules of the first kind specify that, under a given condition, a set of constraints is equivalent to another one. The rules of the second kind specify that, under a given condition, a set of constraints implies another one. In a constraint logic programming language supporting constraint handling rules, a programmer can use these rules to specify possible rewritings of the constraint store and possible additions of constraints to it. The following are example rules:

```
A(X) <=> B(X) | C(X)
A(X) ==> B(X) | C(X)
```

The first rule tells that, if `B(X)` is entailed by the store, the constraint `A(X)` can be rewritten as `C(X)`. As an example, `N*X>0` can be rewritten as `X>0` if the store implies that `N>0`. The symbol `<=>` resembles equivalence in logic, and tells that the first constraint is equivalent to the latter. In practice, this implies that the first constraint can be *replaced* with the latter.

The second rule instead specifies that the latter constraint is a consequence of the first, if the constraint in the middle is entailed by the constraint store. As a result, if `A(X)` is in the constraint store and `B(X)` is entailed by the constraint store, then `C(X)` can be added to the store. Differently from the case of equivalence, this is an addition and not a replacement: the new constraint is added but the old one remains.

Equivalence allows for simplifying the constraint store by replacing some constraints with simpler ones; in particular, if the third constraint in an equivalence rule is `true`, and the second constraint is entailed, the first constraint is removed from the constraint store. Inference allows for the addition of new constraints, which may lead to proving inconsistency of the constraint store, and may generally reduce the amount of search needed to establish its satisfiability.

Logic programming clauses in conjunction with constraint handling rules can be used to specify a method for establishing the satisfiability of the constraint store. Different

clauses are used to implement the different choices of the method; the constraint handling rules are used for rewriting the constraint store during execution. As an example, one can implement backtracking with unit propagation this way. Let `holds(L)` represents a propositional clause, in which the literals in the list `L` are in the same order as they are evaluated. The algorithm can be implemented using clauses for the choice of assigning a literal to true or false, and constraint handling rules to specify propagation. These rules specify that `holds([l|L])` can be removed if `l=true` follows from the store, and it can be rewritten as `holds(L)` if `l=false` follows from the store. Similarly, `holds([l])` can be replaced by `l=true`. In this example, the choice of value for a variable is implemented using clauses of logic programming; however, it can be encoded in constraint handling rules using an extension called disjunctive constraint handling rules or CHR$^{\vee}$.

Bottom-up evaluation

The standard strategy of evaluation of logic programs is top-down and depth-first: from the goal, a number of clauses are identified as being possibly able to prove the goal, and recursion over the literals of their bodies is performed. An alternative strategy is to start from the facts and use clauses to derive new facts; this strategy is called bottom-up. It is considered better than the top-down one when the aim is that of producing all consequences of a given program, rather than proving a single goal. In particular, finding all consequences of a program in the standard top-down and depth-first manner may not terminate while the bottom-up evaluation strategy terminates.

The bottom-up evaluation strategy maintains the set of facts proved so far during evaluation. This set is initially empty. With each step, new facts are derived by applying a program clause to the existing facts, and are added to the set. For example, the bottom up evaluation of the following program requires two steps:

```
A(q).
B(X):-A(X).
```

The set of consequences is initially empty. At the first step, `A(q)` is the only clause whose body can be proved (because it is empty), and `A(q)` is therefore added to the current set of consequences. At the second step, since `A(q)` is proved, the second clause can be used and `B(q)` is added to the consequences. Since no other consequence can be proved from `{A(q),B(q)}`, execution terminates.

The advantage of the bottom-up evaluation over the top-down one is that cycles of derivations do not produce an infinite loop. This is because adding a consequence to the current set of consequences that already contains it has no effect. As an example, adding a third clause to the above program generates a cycle of derivations in the top-down evaluation:

```
A(q).
B(X):-A(X).
A(X):-B(X).
```

For example, while evaluating all answers to the goal `A(X)`, the top-down strategy would produce the following derivations:

```
A(q)
A(q):-B(q), B(q):-A(q), A(q)
A(q):-B(q), B(q):-A(q), A(q):-B(q), B(q):-A(q), A(q)
```

In other words, the only consequence `A(q)` is produced first, but then the algorithm cycles over derivations that do not produce any other answer. More generally, the top-down evaluation strategy may cycle over possible derivations, possibly when other ones exist.

The bottom-up strategy does not have the same drawback, as consequences that were already derived has no effect. On the above program, the bottom-up strategy starts adding `A(q)` to the set of consequences; in the second step, `B(X):-A(X)` is used to derive `B(q)`; in the third step, the only facts that can be derived from the current consequences are `A(q)` and `B(q)`, which are however already in the set of consequences. As a result, the algorithm stops.

In the above example, the only used facts were ground literals. In general, every clause that only contains constraints in the body is considered a fact. For example, a clause `A(X):-X>0,X<10` is considered a fact as well. For this extended definition of facts, some facts may be equivalent while not syntactically equal. For example, `A(q)` is equivalent to `A(X):-X=q` and both are equivalent to `A(X):-X=Y, Y=q`. To solve this problem, facts are translated into a normal form in which the head contains a tuple of all-different variables; two facts are then equivalent if their bodies are equivalent on the variables of the head, that is, their sets of solutions are the same when restricted to these variables.

As described, the bottom-up approach has the advantage of not considering consequences that have already been derived. However, it still may derive consequences that are entailed by those already derived while not being equal to any of them. As an example, the bottom up evaluation of the following program is infinite:

```
A(0).
A(X):-X>0.
A(X):-X=Y+1, A(Y).
```

The bottom-up evaluation algorithm first derives that `A(X)` is true for `X=0` and `X>0`. In the second step, the first fact with the third clause allows for the derivation of `A(1)`. In the third step, `A(2)` is derived, etc. However, these facts are already entailed by the fact that `A(X)` is true for any nonnegative `X`. This drawback can be overcome by checking for entailment facts that are to be added to the current set of consequences. If the new consequence is already entailed by the set, it is not added to it. Since facts are stored as clauses, possibly with "local variables", entailment is restricted over the variables of their heads.

Concurrent constraint logic programming

The concurrent versions of constraint logic programming are aimed at programming concurrent processes rather than solving constraint satisfaction problems. Goals in constraint logic programming are evaluated concurrently; a concurrent process is therefore programmed as the evaluation of a goal by the interpreter.

Syntactically, concurrent constraints logic programs are similar to non-concurrent programs, the only exception being that clauses includes *guards*, which are constraints that may block the applicability of the clause under some conditions. Semantically, concurrent constraint logic programming differs from its non-concurrent versions because a goal evaluation is intended to realize a concurrent process rather than finding a solution to a problem. Most notably, this difference affects how the interpreter behaves when more than one clause is applicable: non-concurrent constraint logic programming recursively tries all clauses; concurrent constraint logic programming chooses only one. This is the most evident effect of an intended *directionality* of the interpreter, which never revise a choice it has previously taken. Other effects of this are the semantical possibility of having a goal that cannot be proved while the whole evaluation does not fail, and a particular way for equating a goal and a clause head.

Constraint satisfaction toolkits

Constraint satisfaction toolkits are software libraries for imperative programming languages that are used to encode and solve a constraint satisfaction problem.

- Cassowary constraint solver is an open source project for constraint satisfaction (accessible from C, Java, Python and other languages).
- Comet, a commercial programming language and toolkit
- Gecode, an open source portable toolkit written in C++ developed as a production-quality and highly efficient implementation of a complete theoretical background.
- JaCoP (solver) an open source Java constraint solver
- Koalog a commercial Java-based constraint solver.
- logilab-constraint an open source constraint solver written in pure Python with constraint propagation algorithms.
- MINION an open-source constraint solver written in C++, with a small language for the purpose of specifying models/problems.
- ZDC is an open source program developed in the Computer-Aided Constraint Satisfaction Project for modelling and solving constraint satisfaction problems.

Other constraint programming languages

Constraint toolkits are a way for embedding constraints into an imperative programming language. However, they are only used as external libraries for encoding and solving problems. An approach in which constraints are integrated into an imperative programming language is taken in the Kaleidoscope programming language.