Soft Computing & Semantic Web

(Concepts, Components & Applications)

Bryce Noland

Lea Almond

First Edition, 2012

ISBN 978-81-323-1292-5

© All rights reserved.

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

Table of Contents

Chapter 1

Soft Computing

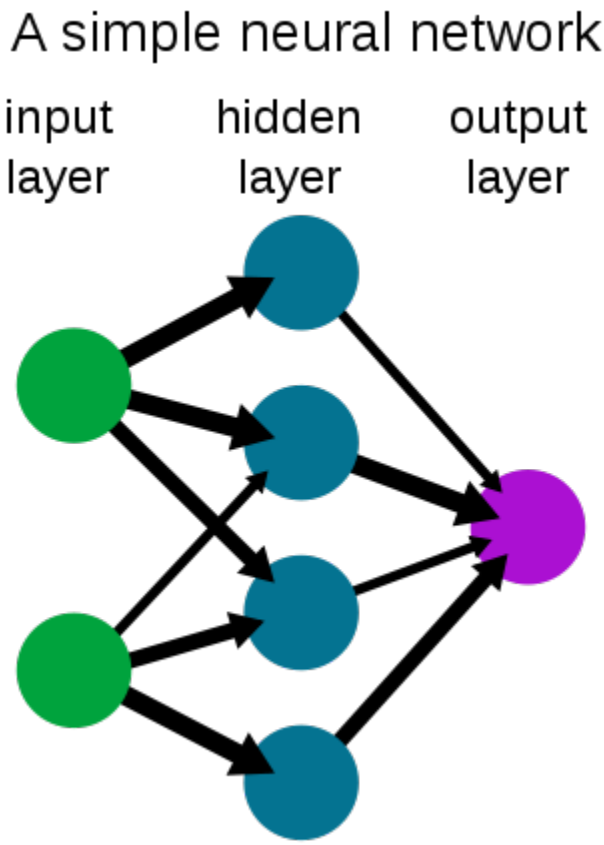
Soft computing is a term applied to a field within computer science which is characterized by the use of inexact solutions to computationally-hard tasks such as the solution of NP-complete problems, for which an exact solution cannot be derived in polynomial time.

Introduction

Soft Computing became a formal Computer Science area of study in the early 1990's. Earlier computational approaches could model and precisely analyze only relatively simple systems. More complex systems arising in biology, medicine, the humanities, management sciences, and similar fields often remained intractable to conventional mathematical and analytical methods. That said, it should be pointed out that simplicity and complexity of systems are relative, and many conventional mathematical models have been both challenging and very productive. Soft computing deals with imprecision, uncertainty, partial truth, and approximation to achieve tractability, robustness and low solution cost.

Components of soft computing include:

Neural network



Simplified view of a feedforward artificial neural network

The term **neural network** was traditionally used to refer to a network or circuit of biological neurons. The modern usage of the term often refers to artificial neural networks, which are composed of artificial neurons or nodes. Thus the term has two distinct usages:

1. Biological neural networks are made up of real biological neurons that are connected or functionally related in the peripheral nervous system or the central nervous system. In the field of neuroscience, they are often identified as groups of neurons that perform a specific physiological function in laboratory analysis.
2. Artificial neural networks are made up of interconnecting artificial neurons (programming constructs that mimic the properties of biological neurons). Artificial neural networks may either be used to gain an understanding of biological neural networks, or for solving artificial intelligence problems without necessarily creating a model of a real biological system. The real, biological

nervous system is highly complex and includes some features that may seem superfluous based on an understanding of artificial networks.

Overview

In general a biological neural network is composed of a group or groups of chemically connected or functionally associated neurons. A single neuron may be connected to many other neurons and the total number of neurons and connections in a network may be extensive. Connections, called synapses, are usually formed from axons to dendrites, though dendrodendritic microcircuits and other connections are possible. Apart from the electrical signaling, there are other forms of signaling that arise from neurotransmitter diffusion, which have an effect on electrical signaling. As such, neural networks are extremely complex.

Artificial intelligence and cognitive modeling try to simulate some properties of neural networks. While similar in their techniques, the former has the aim of solving particular tasks, while the latter aims to build mathematical models of biological neural systems.

In the artificial intelligence field, artificial neural networks have been applied successfully to speech recognition, image analysis and adaptive control, in order to construct software agents (in computer and video games) or autonomous robots. Most of the currently employed artificial neural networks for artificial intelligence are based on statistical estimation, optimization and control theory.

The cognitive modelling field involves the physical or mathematical modeling of the behaviour of neural systems; ranging from the individual neural level (e.g. modelling the spike response curves of neurons to a stimulus), through the neural cluster level (e.g. modelling the release and effects of dopamine in the basal ganglia) to the complete organism (e.g. behavioural modelling of the organism's response to stimuli). Artificial intelligence, cognitive modelling, and neural networks are information processing paradigms inspired by the way biological neural systems process data.

History of the neural network analogy

The concept of neural networks started in the late 19th century as an effort to describe how the human mind performed.

The Austrian School of economics theory of *spontaneous order* was explained by Murray Rothbard to have been first realized by Zhuangzi (Chuang Tzu) who said, "Good order results spontaneously when things are let alone." In the brain spontaneous order arises out of decentralized networks of simple units (neurons). In the late 1940s Donald Hebb made one of the first hypotheses of learning with a mechanism of neural plasticity called Hebbian learning. Hebbian learning is considered to be a 'typical' unsupervised learning rule and its later variants were early models for long term potentiation. These ideas

started being applied to computational models in 1948 with Turing's B-type machines and the perceptron.

The perceptron is essentially a linear classifier for classifying data $x \in \mathbb{R}^n$ specified by parameters $w \in \mathbb{R}^n, b \in \mathbb{R}$ and an output function $f = w'x + b$. Its parameters are adapted with an ad-hoc rule similar to stochastic steepest gradient descent. Because the inner product is a linear operator in the input space, the perceptron can only perfectly classify a set of data for which different classes are linearly separable in the input space, while it often fails completely for non-separable data. While the development of the algorithm initially generated some enthusiasm, partly because of its apparent relation to biological mechanisms, the later discovery of this inadequacy caused such models to be abandoned until the introduction of non-linear models into the field.

The cognitron (1975) designed by Kunihiko Fukushima was an early multilayered neural network with a training algorithm. The actual structure of the network and the methods used to set the interconnection weights change from one neural strategy to another, each with its advantages and disadvantages. Networks can propagate information in one direction only, or they can bounce back and forth until self-activation at a node occurs and the network settles on a final state. The ability for bi-directional flow of inputs between neurons/nodes was produced with the Hopfield's network (1982), and specialization of these node layers for specific purposes was introduced through the first hybrid network.

The parallel distributed processing of the mid-1980s became popular under the name connectionism.

The rediscovery of the backpropagation algorithm was probably the main reason behind the repopularisation of neural networks after the publication of "Learning Internal Representations by Error Propagation" in 1986 (Though backpropagation itself dates from 1969). The original network utilized multiple layers of weight-sum units of the type $f = g(w'x + b)$, where g was a sigmoid function or logistic function such as used in logistic regression. Training was done by a form of stochastic Gradient descent. The employment of the chain rule of differentiation in deriving the appropriate parameter updates results in an algorithm that seems to 'backpropagate errors', hence the nomenclature. However it is essentially a form of gradient descent. Determining the optimal parameters in a model of this type is not trivial, and steepest gradient descent methods cannot be relied upon to give the solution without a good starting point. In recent times, networks with the same architecture as the backpropagation network are referred to as Multi-Layer Perceptrons. This name does not impose any limitations on the type of algorithm used for learning.

The backpropagation network generated much enthusiasm at the time and there was much controversy about whether such learning could be implemented in the brain or not, partly because a mechanism for reverse signaling was not obvious at the time, but most importantly because there was no plausible source for the 'teaching' or 'target' signal.

The brain, neural networks and computers

Neural networks, as used in artificial intelligence, have traditionally been viewed as simplified models of neural processing in the brain, even though the relation between this model and brain biological architecture is debated, as little is known about how the brain actually works.

A subject of current research in theoretical neuroscience is the question surrounding the degree of complexity and the properties that individual neural elements should have to reproduce something resembling animal intelligence.

Historically, computers evolved from the von Neumann architecture, which is based on sequential processing and execution of explicit instructions. On the other hand, the origins of neural networks are based on efforts to model information processing in biological systems, which may rely largely on parallel processing as well as implicit instructions based on recognition of patterns of 'sensory' input from external sources. In other words, at its very heart a neural network is a complex statistical processor (as opposed to being tasked to sequentially process and execute).

Neural coding is concerned with how sensory and other information is represented in the brain by neurons. The main goal of studying neural coding is to characterize the relationship between the stimulus and the individual or ensemble neuronal responses and the relationship among electrical activity of the neurons in the ensemble. It is thought that neurons can encode both digital and analog information.

Neural networks and artificial intelligence

A *neural network* (NN), in the case of artificial neurons called *artificial neural network* (ANN) or *simulated neural network* (SNN), is an interconnected group of natural or artificial neurons that uses a mathematical or computational model for information processing based on a connectionistic approach to computation. In most cases an ANN is an adaptive system that changes its structure based on external or internal information that flows through the network.

In more practical terms neural networks are non-linear statistical data modeling or decision making tools. They can be used to model complex relationships between inputs and outputs or to find patterns in data.

However, the paradigm of neural networks - i.e., *implicit*, not *explicit* , learning is stressed - seems more to correspond to some kind of *natural intelligence* than to the traditional *Artificial Intelligence*, which would stress, instead, rule-based learning.

Background

An artificial neural network involves a network of simple processing elements (artificial neurons) which can exhibit complex global behavior, determined by the connections

between the processing elements and element parameters. Artificial neurons were first proposed in 1943 by Warren McCulloch, a neurophysiologist, and Walter Pitts, an MIT logician. One classical type of artificial neural network is the recurrent Hopfield net.

In a neural network model simple nodes, which can be called variously "neurons", "neurodes", "Processing Elements" (PE) or "units", are connected together to form a network of nodes — hence the term "neural network". While a neural network does not have to be adaptive *per se*, its practical use comes with algorithms designed to alter the strength (weights) of the connections in the network to produce a desired signal flow.

In modern software implementations of artificial neural networks the approach inspired by biology has more or less been abandoned for a more practical approach based on statistics and signal processing. In some of these systems, neural networks, or parts of neural networks (such as artificial neurons), are used as components in larger systems that combine both adaptive and non-adaptive elements.

The concept of a neural network appears to have first been proposed by Alan Turing in his 1948 paper "Intelligent Machinery".

Applications of natural and of artificial neural networks

The utility of artificial neural network models lies in the fact that they can be used to infer a function from observations and also to use it. This is particularly useful in applications where the complexity of the data or task makes the design of such a function by hand impractical.

Real life applications

The tasks to which artificial neural networks are applied tend to fall within the following broad categories:

- Function approximation, or regression analysis, including time series prediction and modeling.
- Classification, including pattern and sequence recognition, novelty detection and sequential decision making.
- Data processing, including filtering, clustering, blind signal separation and compression.

Application areas of ANNs include system identification and control (vehicle control, process control), game-playing and decision making (backgammon, chess, racing), pattern recognition (radar systems, face identification, object recognition, etc.), sequence recognition (gesture, speech, handwritten text recognition), medical diagnosis, financial applications, data mining (or knowledge discovery in databases, "KDD"), visualization and e-mail spam filtering.

- Moreover, some brain diseases, e.g. Alzheimer, are apparently, and essentially, diseases of the brain's natural NN by damaging necessary prerequisites for the functioning of the mutual interconnections between neurons and/or glia.

Neural networks and neuroscience

Theoretical and computational neuroscience is the field concerned with the theoretical analysis and computational modeling of biological neural systems. Since neural systems are intimately related to cognitive processes and behaviour, the field is closely related to cognitive and behavioural modeling.

The aim of the field is to create models of biological neural systems in order to understand how biological systems work. To gain this understanding, neuroscientists strive to make a link between observed biological processes (data), biologically plausible mechanisms for neural processing and learning (biological neural network models) and theory (statistical learning theory and information theory).

Types of models

Many models are used in the field, each defined at a different level of abstraction and trying to model different aspects of neural systems. They range from models of the short-term behaviour of individual neurons, through models of how the dynamics of neural circuitry arise from interactions between individual neurons, to models of how behaviour can arise from abstract neural modules that represent complete subsystems. These include models of the long-term and short-term plasticity of neural systems and its relation to learning and memory, from the individual neuron to the system level.

Current research

While initially research had been concerned mostly with the electrical characteristics of neurons, a particularly important part of the investigation in recent years has been the exploration of the role of neuromodulators such as dopamine, acetylcholine, and serotonin on behaviour and learning.

Biophysical models, such as BCM theory, have been important in understanding mechanisms for synaptic plasticity, and have had applications in both computer science and neuroscience. Research is ongoing in understanding the computational algorithms used in the brain, with some recent biological evidence for radial basis networks and neural backpropagation as mechanisms for processing data.

Computational devices have been created in CMOS for both biophysical simulation and neuromorphic computing. More recent efforts show promise for creating nanodevices for very large scale principal components analyses and convolution. If successful, these efforts could usher in a new era of neural computing that is a step beyond digital computing, because it depends on learning rather than programming and because it is

fundamentally analog rather than digital even though the first instantiations may in fact be with CMOS digital devices.

Criticism

A common criticism of neural networks, particularly in robotics, is that they require a large diversity of training for real-world operation. Dean Pomerleau, in his research presented in the paper "Knowledge-based Training of Artificial Neural Networks for Autonomous Robot Driving," uses a neural network to train a robotic vehicle to drive on multiple types of roads (single lane, multi-lane, dirt, etc.). A large amount of his research is devoted to (1) extrapolating multiple training scenarios from a single training experience, and (2) preserving past training diversity so that the system does not become overtrained (if, for example, it is presented with a series of right turns – it should not learn to always turn right). These issues are common in neural networks that must decide from amongst a wide variety of responses.

A. K. Dewdney, a former *Scientific American* columnist, wrote in 1997, "Although neural nets do solve a few toy problems, their powers of computation are so limited that I am surprised anyone takes them seriously as a general problem-solving tool."

Arguments for Dewdney's position are that to implement large and effective software neural networks, much processing and storage resources need to be committed. While the brain has hardware tailored to the task of processing signals through a graph of neurons, simulating even a most simplified form on Von Neumann technology may compel a NN designer to fill many millions of database rows for its connections - which can lead to abusive RAM and HD necessities. Furthermore, the designer of NN systems will often need to simulate the transmission of signals through many of these connections and their associated neurons - which must often be matched with incredible amounts of CPU processing power and time. While neural networks often yield *effective* programs, they too often do so at the cost of time and money *efficiency*.

Arguments against Dewdney's position are that neural nets have been successfully used to solve many complex and diverse tasks, ranging from autonomously flying aircraft to detecting credit card fraud .

Technology writer Roger Bridgman commented on Dewdney's statements about neural nets:

Neural networks, for instance, are in the dock not only because they have been hyped to high heaven, (what hasn't?) but also because you could create a successful net without understanding how it worked: the bunch of numbers that captures its behaviour would in all probability be "an opaque, unreadable table...valueless as a scientific resource". In spite of his emphatic declaration that science is not technology, Dewdney seems here to pillory neural nets as bad science when most of those devising them are just trying to be good engineers. An unreadable table that a useful machine could read would still be well worth having.

Some other criticisms came from believers of hybrid models (combining neural networks and symbolic approaches). They advocate the intermix of these two approaches and believe that hybrid models can better capture the mechanisms of the human mind (Sun and Bookman 1990).

Chapter 2

Fuzzy Control System

A **fuzzy control system** is a control system based on fuzzy logic—a mathematical system that analyzes analog input values in terms of logical variables that take on continuous values between 0 and 1, in contrast to classical or digital logic, which operates on discrete values of either 0 or 1 (true or false).

Overview

Fuzzy logic is widely used in machine control. The term itself inspires a certain skepticism, sounding equivalent to "half-baked logic" or "bogus logic", but the "fuzzy" part does not refer to a lack of rigour in the method, rather to the fact that the logic involved can deal with fuzzy concepts—concepts that cannot be expressed as "true" or "false" but rather as "partially true". Although genetic algorithms and neural networks can perform just as well as fuzzy logic in many cases , fuzzy logic has the advantage that the solution to the problem can be cast in terms that human operators can understand, so that their experience can be used in the design of the controller. This makes it easier to mechanize tasks that are already successfully performed by humans.

History and applications

Fuzzy logic was first proposed by Lotfi A. Zadeh of the University of California at Berkeley in a 1965 paper. He elaborated on his ideas in a 1973 paper that introduced the concept of "linguistic variables", which in this article equates to a variable defined as a fuzzy set. Other research followed, with the first industrial application, a cement kiln built in Denmark, coming on line in 1975.

Fuzzy systems were largely ignored in the U.S. because they were associated with artificial intelligence, a field that periodically oversells itself, especially in the mid-1980s, resulting in a lack of credibility within the commercial domain.

The Japanese did not have this prejudice. Interest in fuzzy systems was sparked by Seiji Yasunobu and Soji Miyamoto of Hitachi, who in 1985 provided simulations that demonstrated the superiority of fuzzy control systems for the Sendai railway. Their ideas were adopted, and fuzzy systems were used to control accelerating, braking, and stopping when the line opened in 1987.

Another event in 1987 helped promote interest in fuzzy systems. During an international meeting of fuzzy researchers in Tokyo that year, Takeshi Yamakawa demonstrated the use of fuzzy control, through a set of simple dedicated fuzzy logic chips, in an "inverted pendulum" experiment. This is a classic control problem, in which a vehicle tries to keep a pole mounted on its top by a hinge upright by moving back and forth.

Observers were impressed with this demonstration, as well as later experiments by Yamakawa in which he mounted a wine glass containing water or even a live mouse to the top of the pendulum. The system maintained stability in both cases. Yamakawa eventually went on to organize his own fuzzy-systems research lab to help exploit his patents in the field.

Following such demonstrations, Japanese engineers developed a wide range of fuzzy systems for both industrial and consumer applications. In 1988 Japan established the Laboratory for International Fuzzy Engineering (LIFE), a cooperative arrangement between 48 companies to pursue fuzzy research.

Japanese consumer goods often incorporate fuzzy systems. Matsushita vacuum cleaners use microcontrollers running fuzzy algorithms to interrogate dust sensors and adjust suction power accordingly. Hitachi washing machines use fuzzy controllers to load-weight, fabric-mix, and dirt sensors and automatically set the wash cycle for the best use of power, water, and detergent.

As a more specific example, Canon developed an autofocusing camera that uses a charge-coupled device (CCD) to measure the clarity of the image in six regions of its field of view and use the information provided to determine if the image is in focus. It also tracks the rate of change of lens movement during focusing, and controls its speed to prevent overshoot.

The camera's fuzzy control system uses 12 inputs: 6 to obtain the current clarity data provided by the CCD and 6 to measure the rate of change of lens movement. The output is the position of the lens. The fuzzy control system uses 13 rules and requires 1.1 kilobytes of memory.

As another example of a practical system, an industrial air conditioner designed by Mitsubishi uses 25 heating rules and 25 cooling rules. A temperature sensor provides input, with control outputs fed to an inverter, a compressor valve, and a fan motor. Compared to the previous design, the fuzzy controller heats and cools five times faster, reduces power consumption by 24%, increases temperature stability by a factor of two, and uses fewer sensors.

The enthusiasm of the Japanese for fuzzy logic is reflected in the wide range of other applications they have investigated or implemented: character and handwriting recognition; optical fuzzy systems; robots, including one for making Japanese flower arrangements; voice-controlled robot helicopters, this being no mean feat, as hovering is

a "balancing act" rather similar to the inverted pendulum problem; control of flow of powders in film manufacture; elevator systems; and so on.

Work on fuzzy systems is also proceeding in the US and Europe, though not with the same enthusiasm shown in Japan. The US Environmental Protection Agency has investigated fuzzy control for energy-efficient motors, and NASA has studied fuzzy control for automated space docking: simulations show that a fuzzy control system can greatly reduce fuel consumption. Firms such as Boeing, General Motors, Allen-Bradley, Chrysler, Eaton, and Whirlpool have worked on fuzzy logic for use in low-power refrigerators, improved automotive transmissions, and energy-efficient electric motors.

In 1995 Maytag introduced an "intelligent" dishwasher based on a fuzzy controller and a "one-stop sensing module" that combines a thermistor, for temperature measurement; a conductivity sensor, to measure detergent level from the ions present in the wash; a turbidity sensor that measures scattered and transmitted light to measure the soiling of the wash; and a magnetostrictive sensor to read spin rate. The system determines the optimum wash cycle for any load to obtain the best results with the least amount of energy, detergent, and water. It even adjusts for dried-on foods by tracking the last time the door was opened, and estimates the number of dishes by the number of times the door was opened.

Research and development is also continuing on fuzzy applications in software, as opposed to firmware, design, including fuzzy expert systems and integration of fuzzy logic with neural-network and so-called adaptive "genetic" software systems, with the ultimate goal of building "self-learning" fuzzy control systems.

Fuzzy sets

The input variables in a fuzzy control system are in general mapped into by sets of membership functions similar to this, known as "fuzzy sets". The process of converting a crisp input value to a fuzzy value is called "fuzzification".

A control system may also have various types of switch, or "ON-OFF", inputs along with its analog inputs, and such switch inputs of course will always have a truth value equal to either 1 or 0, but the scheme can deal with them as simplified fuzzy functions that happen to be either one value or another.

Given "mappings" of input variables into membership functions and truth values, the microcontroller then makes decisions for what action to take based on a set of "rules", each of the form:

```
IF brake temperature IS warm AND speed IS not very fast
THEN brake pressure IS slightly decreased.
```

In this example, the two input variables are "brake temperature" and "speed" that have values defined as fuzzy sets. The output variable, "brake pressure", is also defined by a

fuzzy set that can have values like "static", "slightly increased", "slightly decreased", and so on.

This rule by itself is very puzzling since it looks like it could be used without bothering with fuzzy logic, but remember that the decision is based on a set of rules:

- All the rules that apply are invoked, using the membership functions and truth values obtained from the inputs, to determine the result of the rule.
- This result in turn will be mapped into a membership function and truth value controlling the output variable.
- These results are combined to give a specific ("crisp") answer, the actual brake pressure, a procedure known as "defuzzification".

This combination of fuzzy operations and rule-based "inference" describes a "fuzzy expert system".

Traditional control systems are based on mathematical models in which the control system is described using one or more differential equations that define the system response to its inputs. Such systems are often implemented as "PID controllers" (proportional-integral-derivative controllers). They are the products of decades of development and theoretical analysis, and are highly effective.

If PID and other traditional control systems are so well-developed, why bother with fuzzy control? It has some advantages. In many cases, the mathematical model of the control process may not exist, or may be too "expensive" in terms of computer processing power and memory, and a system based on empirical rules may be more effective.

Furthermore, fuzzy logic is well suited to low-cost implementations based on cheap sensors, low-resolution analog-to-digital converters, and 4-bit or 8-bit one-chip microcontroller chips. Such systems can be easily upgraded by adding new rules to improve performance or add new features. In many cases, fuzzy control can be used to improve existing traditional controller systems by adding an extra layer of intelligence to the current control method.

Fuzzy control in detail

Fuzzy controllers are very simple conceptually. They consist of an input stage, a processing stage, and an output stage. The input stage maps sensor or other inputs, such as switches, thumbwheels, and so on, to the appropriate membership functions and truth values. The processing stage invokes each appropriate rule and generates a result for each, then combines the results of the rules. Finally, the output stage converts the combined result back into a specific control output value.

The most common shape of membership functions is triangular, although trapezoidal and bell curves are also used, but the shape is generally less important than the number of

curves and their placement. From three to seven curves are generally appropriate to cover the required range of an input value, or the "universe of discourse" in fuzzy jargon.

As discussed earlier, the processing stage is based on a collection of logic rules in the form of IF-THEN statements, where the IF part is called the "antecedent" and the THEN part is called the "consequent". Typical fuzzy control systems have dozens of rules.

Consider a rule for a thermostat:

```
IF (temperature is "cold") THEN (heater is "high")
```

This rule uses the truth value of the "temperature" input, which is some truth value of "cold", to generate a result in the fuzzy set for the "heater" output, which is some value of "high". This result is used with the results of other rules to finally generate the crisp composite output. Obviously, the greater the truth value of "cold", the higher the truth value of "high", though this does not necessarily mean that the output itself will be set to "high", since this is only one rule among many. In some cases, the membership functions can be modified by "hedges" that are equivalent to adjectives. Common hedges include "about", "near", "close to", "approximately", "very", "slightly", "too", "extremely", and "somewhat". These operations may have precise definitions, though the definitions can vary considerably between different implementations. "Very", for one example, squares membership functions; since the membership values are always less than 1, this narrows the membership function. "Extremely" cubes the values to give greater narrowing, while "somewhat" broadens the function by taking the square root.

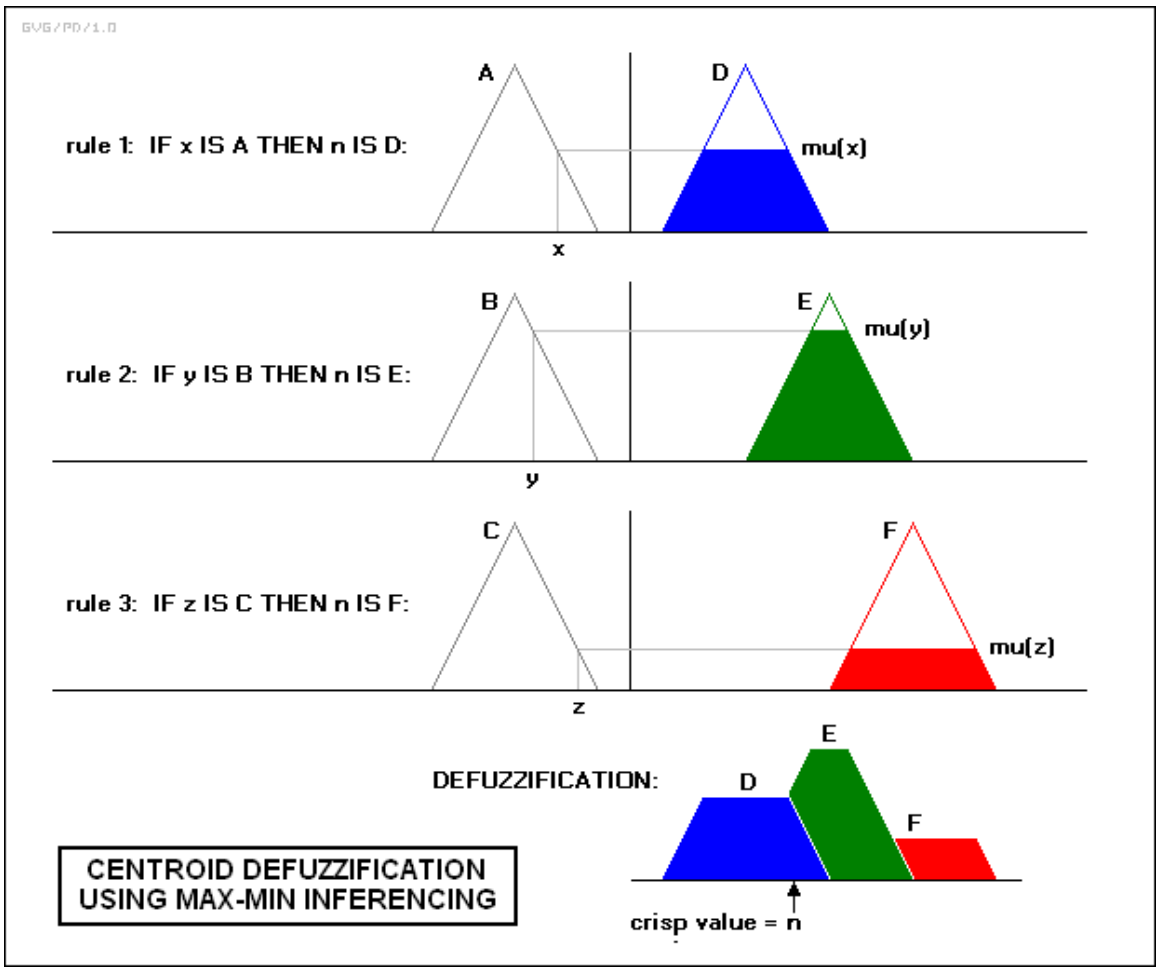
In practice, the fuzzy rule sets usually have several antecedents that are combined using fuzzy operators, such as AND, OR, and NOT, though again the definitions tend to vary: AND, in one popular definition, simply uses the minimum weight of all the antecedents, while OR uses the maximum value. There is also a NOT operator that subtracts a membership function from 1 to give the "complementary" function.

There are several different ways to define the result of a rule, but one of the most common and simplest is the "max-min" inference method, in which the output membership function is given the truth value generated by the premise.

Rules can be solved in parallel in hardware, or sequentially in software. The results of all the rules that have fired are "defuzzified" to a crisp value by one of several methods. There are dozens in theory, each with various advantages and drawbacks.

The "centroid" method is very popular, in which the "center of mass" of the result provides the crisp value. Another approach is the "height" method, which takes the value of the biggest contributor. The centroid method favors the rule with the output of greatest area, while the height method obviously favors the rule with the greatest output value.

The diagram below demonstrates max-min inferencing and centroid defuzzification for a system with input variables "x", "y", and "z" and an output variable "n". Note that "mu" is standard fuzzy-logic nomenclature for "truth value":



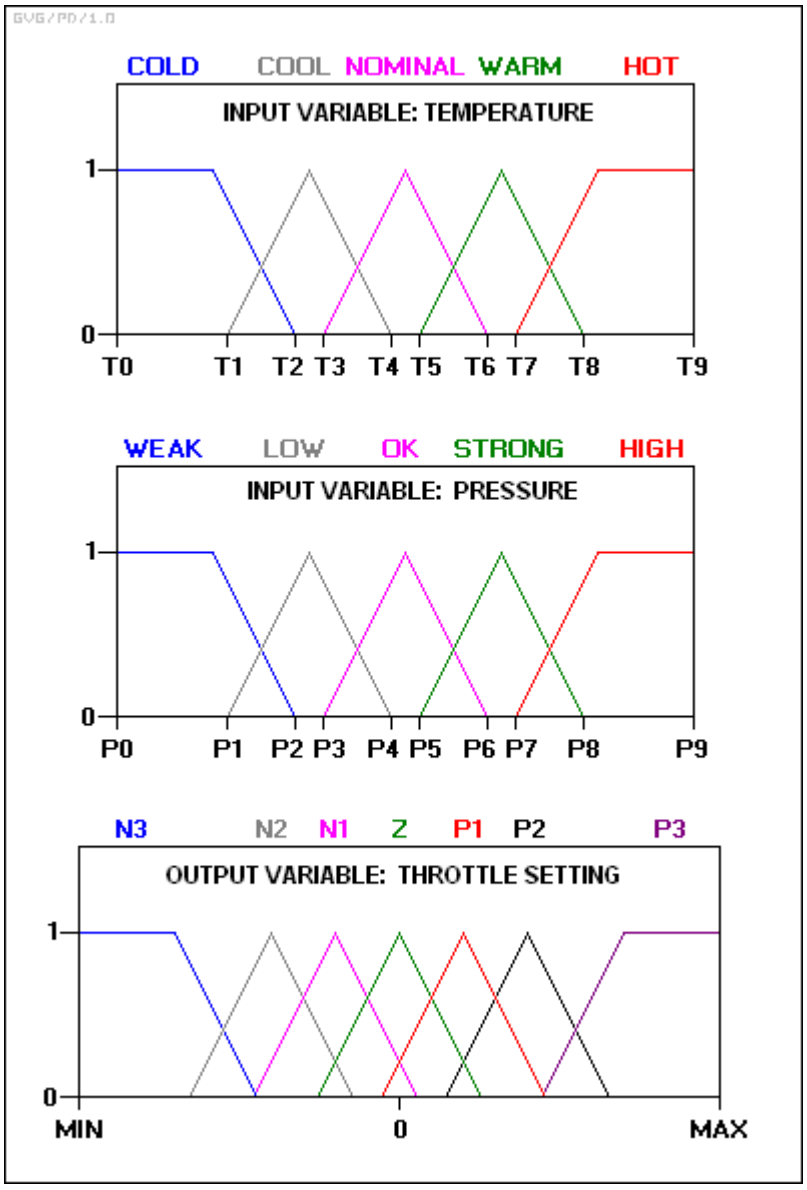
Notice how each rule provides a result as a truth value of a particular membership function for the output variable. In centroid defuzzification the values are OR'd, that is, the maximum value is used and values are not added, and the results are then combined using a centroid calculation.

Fuzzy control system design is based on empirical methods, basically a methodical approach to trial-and-error. The general process is as follows:

- Document the system's operational specifications and inputs and outputs.
- Document the fuzzy sets for the inputs.
- Document the rule set.
- Determine the defuzzification method.
- Run through test suite to validate system, adjust details as required.
- Complete document and release to production.

As a general example, consider the design of a fuzzy controller for a steam turbine. The block diagram of this control system appears as follows:

The input and output variables map into the following fuzzy set:



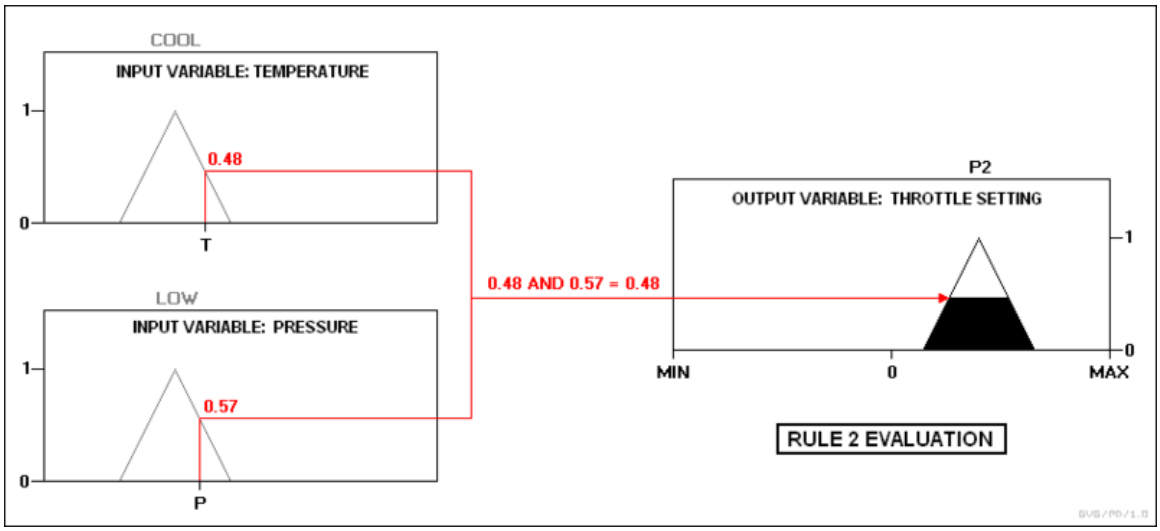
-- where:

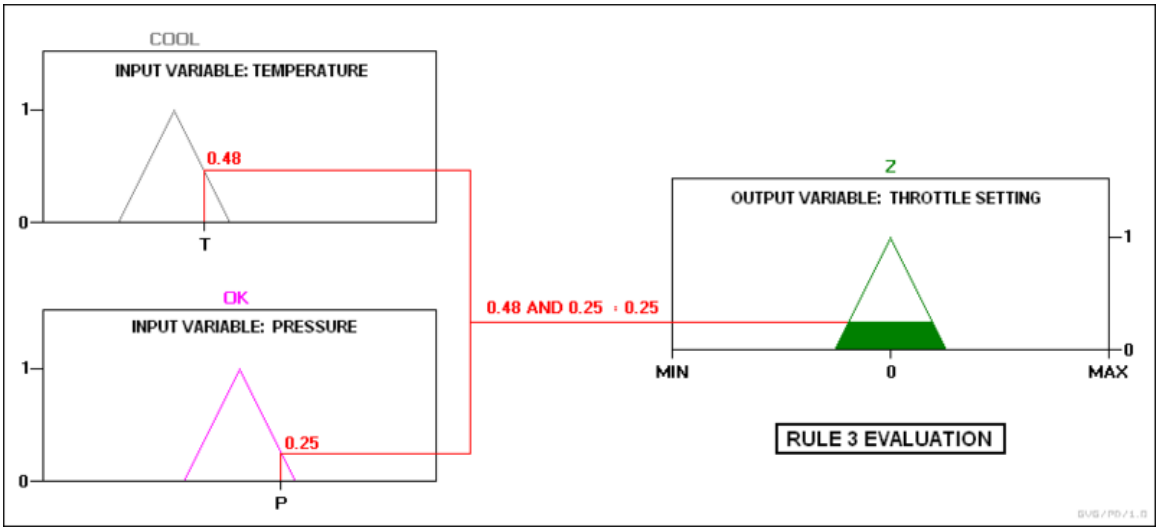
```
N3:   Large negative.
N2:   Medium negative.
N1:   Small negative.
Z:    Zero.
P1:   Small positive.
P2:   Medium positive.
P3:   Large positive.
```

The rule set includes such rules as:

```
rule 1:  IF temperature IS cool AND pressure IS weak,
         THEN throttle is P3.
rule 2:  IF temperature IS cool AND pressure IS low,
         THEN throttle is P2.
rule 3:  IF temperature IS cool AND pressure IS ok,
         THEN throttle is Z.
rule 4:  IF temperature IS cool AND pressure IS strong,
         THEN throttle is N2.
```

In practice, the controller accepts the inputs and maps them into their membership functions and truth values. These mappings are then fed into the rules. If the rule specifies an AND relationship between the mappings of the two input variables, as the examples above do, the minimum of the two is used as the combined truth value; if an OR is specified, the maximum is used. The appropriate output state is selected and assigned a membership value at the truth level of the premise. The truth values are then defuzzified. For an example, assume the temperature is in the "cool" state, and the pressure is in the "low" and "ok" states. The pressure values ensure that only rules 2 and 3 fire:





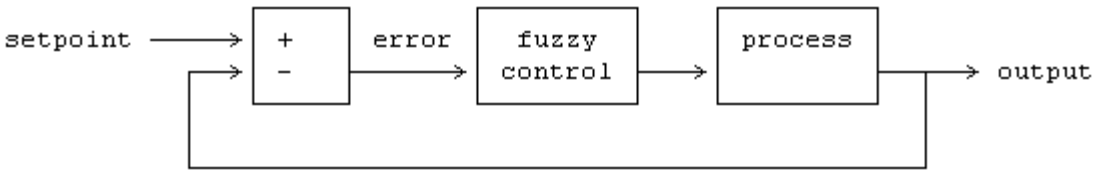
The two outputs are then defuzzified through centroid defuzzification:

```
                    |          Z       P2
                1 --+          *        *
                    |         * *      * *
                    |        *   *    *   *
                    |       *     * *     *
                    |      *       222222222
                    |     *       22222222222
                    |    3333333222222222222
                    +---333333332222222222222-->
                                    ^
                                   +150
```

The output value will adjust the throttle and then the control cycle will begin again to generate the next value.

Building a fuzzy controller

Consider implementing with a microcontroller chip a simple feedback controller:



A fuzzy set is defined for the input error variable "e", and the derived change in error, "delta", as well as the "output", as follows:

```
LP:  large positive
```

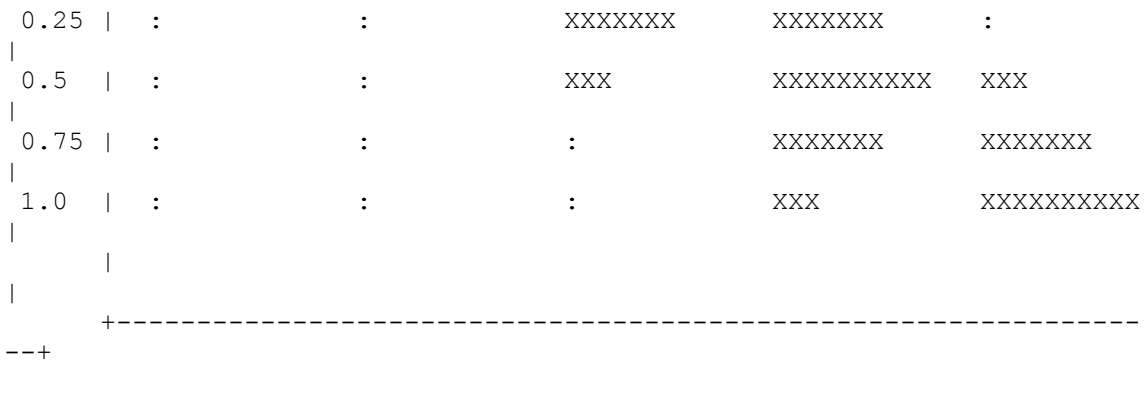
```
  SP:  small positive
  ZE:  zero
  SN:  small negative
  LN:  large negative
```

If the error ranges from -1 to +1, with the analog-to-digital converter used having a resolution of 0.25, then the input variable's fuzzy set (which, in this case, also applies to the output variable) can be described very simply as a table, with the error / delta / output values in the top row and the truth values for each membership function arranged in rows beneath:

	-1	-0.75	-0.5	-0.25	0	0.25	0.5	0.75	1
mu(LP)	0	0	0	0	0	0	0.3	0.7	1
mu(SP)	0	0	0	0	0.3	0.7	1	0.7	0.3
mu(ZE)	0	0	0.3	0.7	1	0.7	0.3	0	0
mu(SN)	0.3	0.7	1	0.7	0.3	0	0	0	0
mu(LN)	1	0.7	0.3	0	0	0	0	0	0

-- or, in graphical form (where each "X" has a value of 0.1):

```
        LN            SN            ZE            SP            LP
      +-------------------------------------------------------------
--+
      |
|
-1.0  |  XXXXXXXXXX   XXX           :             :             :
|
-0.75 |  XXXXXXX      XXXXXXX       :             :             :
|
-0.5  |  XXX          XXXXXXXXXX    XXX           :             :
|
-0.25 |  :            XXXXXXX       XXXXXXX       :             :
|
 0.0  |  :            XXX           XXXXXXXXXX    XXX           :
|
```



Suppose this fuzzy system has the following rule base:

```
  rule 1:  IF e = ZE AND delta = ZE THEN output = ZE
  rule 2:  IF e = ZE AND delta = SP THEN output = SN
  rule 3:  IF e = SN AND delta = SN THEN output = LP
  rule 4:  IF e = LP OR  delta = LP THEN output = LN
```

These rules are typical for control applications in that the antecedents consist of the logical combination of the error and error-delta signals, while the consequent is a control command output. The rule outputs can be defuzzified using a discrete centroid computation:

```
  SUM( I = 1 TO 4 OF ( mu(I) * output(I) ) ) / SUM( I = 1 TO 4 OF mu(I)
)
```

Now, suppose that at a given time we have:

```
  e     = 0.25
  delta = 0.5
```

Then this gives:

	e	delta
mu(LP)	0	0.3
mu(SP)	0.7	1
mu(ZE)	0.7	0.3
mu(SN)	0	0
mu(LN)	0	0

Plugging this into rule 1 gives:

```
rule 1:  IF e = ZE AND delta = ZE THEN output = ZE

   mu(1)     = MIN( 0.7, 0.3 ) = 0.3
   output(1) = 0
```

-- where:

- mu(1): Truth value of the result membership function for rule 1. In terms of a centroid calculation, this is the "mass" of this result for this discrete case.
- output(1): Value (for rule 1) where the result membership function (ZE) is maximum over the output variable fuzzy set range. That is, in terms of a centroid calculation, the location of the "center of mass" for this individual result. This value is independent of the value of "mu". It simply identifies the location of ZE along the output range.

The other rules give:

```
rule 2:  IF e = ZE AND delta = SP THEN output = SN

   mu(2)     = MIN( 0.7, 1 ) = 0.7
   output(2) = -0.5
rule 3: IF e = SN AND delta = SN THEN output = LP

   mu(3)     = MIN( 0.0, 0.0 ) = 0
   output(3) = 1
rule 4: IF e = LP OR  delta = LP THEN output = LN

   mu(4)     = MAX( 0.0, 0.3 ) = 0.3
   output(4) = -1
```

The centroid computation yields:

$$\frac{mu(1).output(1) + mu(2).output(2) + mu(3).output(3) + mu(4).output(4)}{mu(1) + mu(2) + mu(3) + mu(4)}$$

$$= \frac{(0.3 * 0) + (0.7 * -0.5) + (0 * 1) + (0.3 * -1)}{0.3 + 0.7 + 0 + 0.3}$$

```
=  - 0.5
```

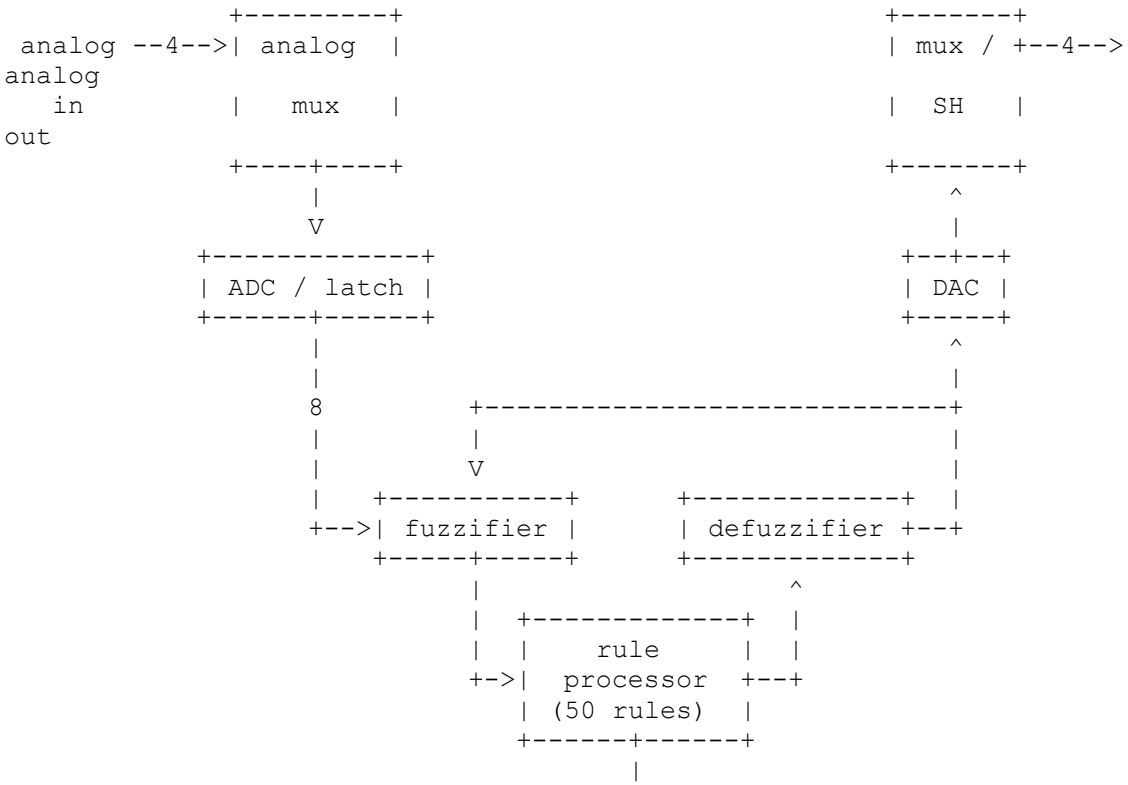
-- for the final control output. Simple. Of course the hard part is figuring out what rules actually work correctly in practice.

If you have problems figuring out the centroid equation, remember that a centroid is defined by summing all the moments (location times mass) around the center of gravity and equating the sum to zero. So if X_0 is the center of gravity, X_i is the location of each mass, and M_i is each mass, this gives:

$$0 = (X_1 - X_0) * M_1 + (X_2 - X_0) * M_2 + \ldots + (X_n - X_0) * M_n$$
$$0 = (X_1 * M_1 + X_2 * M_2 + \ldots + X_n * M_n) - X_0 * (M_1 + M_2 + \ldots + M_n)$$
$$X_0 * (M_1 + M_2 + \ldots + M_n) = X_1 * M_1 + X_2 * M_2 + \ldots + X_n * M_n$$
$$X_0 = \frac{X_1 * M_1 + X_2 * M_2 + \ldots + X_n * M_n}{M_1 + M_2 + \ldots + M_n}$$

In our example, the values of mu correspond to the masses, and the values of X to location of the masses (mu, however, only 'corresponds to the masses' if the initial 'mass' of the output functions are all the same/equivalent. If they are not the same, ie some are narrow triangles, while others maybe wide trapizoids or shouldered triangles, then the mass or area of the output function must be known or calculated. It is this mass that is then scaled by mu and multiplied by it's location X_i).

This system can be implemented on a standard microprocessor, but dedicated fuzzy chips are now available. For example, Adaptive Logic INC of San Jose, California, sells a "fuzzy chip", the AL220, that can accept four analog inputs and generate four analog outputs. A block diagram of the chip is shown below:



```
                                +------+------+
                                |  parameter  |
                                |    memory   |
                                |   256 x 8   |
                                +-------------+
```

```
ADC:  analog-to-digital converter
DAC:  digital-to-analog converter
SH:   sample/hold
```

Antilock brakes

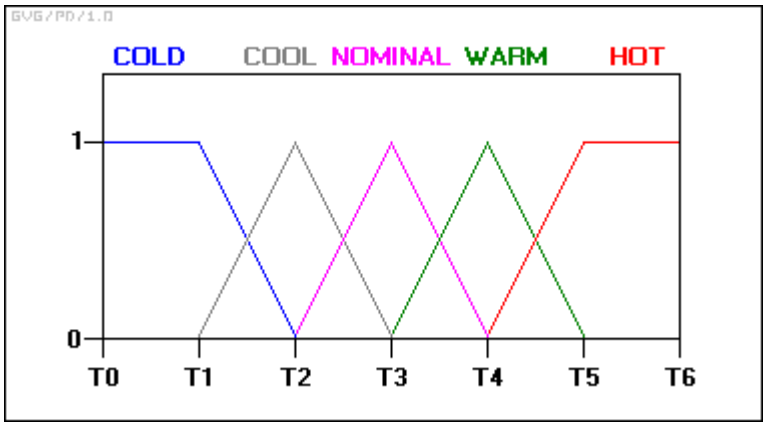
As a first example, consider an anti-lock braking system, directed by a microcontroller chip. The microcontroller has to make decisions based on brake temperature, speed, and other variables in the system.

The variable "temperature" in this system can be subdivided into a range of "states": "cold", "cool", "moderate", "warm", "hot", "very hot". The transition from one state to the next is hard to define.

An arbitrary static threshold might be set to divide "warm" from "hot". For example, at exactly 90 degrees, warm ends and hot begins. But this would result in a discontinuous change when the input value passed over that threshold. The transition wouldn't be smooth, as would be required in braking situations.

The way around this is to make the states *fuzzy*. That is, allow them to change gradually from one state to the next. In order to do this there must be a dynamic relationship established between different factors.

We start by defining the input temperature states using "membership functions":



With this scheme, the input variable's state no longer jumps abruptly from one state to the next. Instead, as the temperature changes, it loses value in one membership function while gaining value in the next. In other words, its ranking in the category of cold decreases as it becomes more highly ranked in the warmer category.

At any sampled timeframe, the "truth value" of the brake temperature will almost always be in some degree part of two membership functions: i.e.: '0.6 nominal and 0.4 warm', or '0.7 nominal and 0.3 cool', and so on.

The above example demonstrates a simple application, using the abstraction of values from multiple values. This only represents one kind of data, however, in this case, temperature.

Adding additional sophistication to this braking system, could be done by additional factors such as traction, speed, inertia, set up in dynamic functions, according to the designed fuzzy system.

Logical interpretation of fuzzy control

In spite of the appearance there are several difficulties to give a rigorous logical interpretation of the *IF-THEN* rules. As an example, interpret a rule as *IF (temperature is "cold") THEN (heater is "high")* by the first order formula $Cold(x) \rightarrow High(y)$ and assume that r is an input such that $Cold(r)$ is false. Then the formula $Cold(r) \rightarrow High(t)$ is true for any t and therefore any t gives a correct control given r. Obviously, if we consider systems of rules in which the class antecedent define a partition such a paradoxical phenomenon does not arise. In any case there is sometime of unsatisfactory in considering two variables x and y in a rule without some kind of functional dependence. A rigorous logical justification of fuzzy control is given in Hájek's book where fuzzy control is represented as a theory of Hájek's basic logic. Also in Gerla 2005 a logical approach to fuzzy control is proposed based on the following idea. Denote by f the fuzzy function associated with the fuzzy control system, i.e., given the input r, $s(y) = f(r,y)$ is the fuzzy set of possible outputs. Then given a possible output 't', we interpret f(r,t) as the truth degree of the claim *"t is a good answer given r"*. More formally, any system of IF-THEN rules can be translate into a fuzzy program in such a way that the fuzzy function f is the interpretation of a vague predicate $Good(x,y)$ in the associated least fuzzy Herbrand model. In such a way fuzzy control becomes a chapter of fuzzy logic programming. The learning process becomes a question belonging to inductive logic theory.

Chapter 3

Swarm Intelligence (Component of Soft Computing)

Swarm intelligence (SI) is the collective behaviour of decentralized, self-organized systems, natural or artificial. The concept is employed in work on artificial intelligence. The expression was introduced by Gerardo Beni and Jing Wang in 1989, in the context of cellular robotic systems.

SI systems are typically made up of a population of simple agents or boids interacting locally with one another and with their environment. The agents follow very simple rules, and although there is no centralized control structure dictating how individual agents should behave, local, and to a certain degree random, interactions between such agents lead to the emergence of "intelligent" global behavior, unknown to the individual agents. Natural examples of SI include ant colonies, bird flocking, animal herding, bacterial growth, and fish schooling.

The application of swarm principles to robots is called swarm robotics, while 'swarm intelligence' refers to the more general set of algorithms. 'Swarm prediction' has been used in the context of forecasting problems.

Example algorithms

Ant colony optimization

Ant colony optimization (ACO) is a class of optimization algorithms modeled on the actions of an ant colony. ACO methods are useful in problems that need to find paths to goals. Artificial 'ants'—simulation agents—locate optimal solutions by moving through a parameter space representing all possible solutions. Real ants lay down pheromones directing each other to resources while exploring their environment. The simulated 'ants' similarly record their positions and the quality of their solutions, so that in later simulation iterations more ants locate better solutions. One variation on this approach is the bees algorithm, which is more analogous to the foraging patterns of the honey bee.

River formation dynamics

River formation dynamics (RFD) is an heuristic method similar to ant colony optimization (ACO). In fact, RFD can be seen as a gradient version of ACO, based on

copying how water forms rivers by eroding the ground and depositing sediments. As water transforms the environment, altitudes of places are dynamically modified, and decreasing gradients are constructed. The gradients are followed by subsequent drops to create new gradients, reinforcing the best ones. By doing so, good solutions are given in the form of decreasing altitudes. This method has been applied to solve different NP-complete problems (for example, the problems of finding a minimum distances tree and finding a minimum spanning tree in a variable-cost graph). The gradient orientation of RFD makes it specially suitable for solving these problems and provides a good tradeoff between finding good results and not spending much computational time. In fact, RFD fits particularly well for problems consisting in forming a kind of covering tree.

Particle swarm optimization

Particle swarm optimization (PSO) is a global optimization algorithm for dealing with problems in which a best solution can be represented as a point or surface in an n-dimensional space. Hypotheses are plotted in this space and seeded with an initial velocity, as well as a communication channel between the particles. Particles then move through the solution space, and are evaluated according to some fitness criterion after each timestep. Over time, particles are accelerated towards those particles within their communication grouping which have better fitness values. The main advantage of such an approach over other global minimization strategies such as simulated annealing is that the large number of members that make up the particle swarm make the technique impressively resilient to the problem of local minima.

Stochastic diffusion search

Stochastic diffusion search (SDS) is an agent-based probabilistic global search and optimization technique best suited to problems where the objective function can be decomposed into multiple independent partial-functions. Each agent maintains a hypothesis which is iteratively tested by evaluating a randomly selected partial objective function parameterised by the agent's current hypothesis. In the standard version of SDS such partial function evaluations are binary, resulting in each agent becoming active or inactive. Information on hypotheses is diffused across the population via inter-agent communication. Unlike the stigmergic communication used in ACO, in SDS agents communicate hypotheses via a one-to-one communication strategy analogous to the tandem running procedure observed in some species of ant. A positive feedback mechanism ensures that, over time, a population of agents stabilise around the global-best solution. SDS is both an efficient and robust search and optimisation algorithm, which has been extensively mathematically described.

Gravitational search algorithm

Gravitational search algorithm (GSA) is constructed based on the law of Gravity and the notion of mass interactions. The GSA algorithm uses the theory of Newtonian physics and its searcher agents are the collection of masses. In GSA, we have an isolated system of masses. Using the gravitational force, every mass in the system can see the situation of

other masses. The gravitational force is therefore a way of transferring information between different masses.(Rashedi et.al, 2009)

Intelligent Water Drops

Intelligent Water Drops algorithm (IWD) is a swarm-based nature-inspired optimization algorithm, which has been inspired from natural rivers and how they find almost optimal paths to their destination.These near optimal or optimal paths follow from actions and reactions occurring among the water drops and the water drops with their riverbeds. In the IWD algorithm, several artificial water drops cooperate to change their environment in such a way that the optimal path is revealed as the one with the lowest soil on its links. The solutions are incrementally constructed by the IWD algorithm. Consequently, the IWD algorithm is generally a constructive population-based optimization algorithm.

Charged System Search

Charged System Search (CSS) is a new optimization algorithm based on some principles from physics and mechanics. CSS utilizes the governing laws of Coulomb and Gauss from electrostatics and the Newtonian laws of mechanics. CSS is a multi-agent approach in which each agent is a Charged Particle (CP). CPs can affect each other based on their fitness values and their separation distances. The quantity of the resultant force is determined by using the electrostatics laws and the quality of the movement is determined using Newtonian mechanics laws. CSS is applicable to all optimization fields; especially it is suitable for non-smooth or non-convex domains. This algorithm provides a good balance between the exploration and the exploitation paradigms of the algorithm which can considerably improve the efficiency of the algorithm and therefore the CSS also can be considered as a good global and local optimizer simultaneously.

Applications

Swarm Intelligence-based techniques can be used in a number of applications. The U.S. military is investigating swarm techniques for controlling unmanned vehicles. The European Space Agency is thinking about an orbital swarm for self assembly and interferometry. NASA is investigating the use of swarm technology for planetary mapping. A 1992 paper by M. Anthony Lewis and George A. Bekey discusses the possibility of using swarm intelligence to control nanobots within the body for the purpose of killing cancer tumors.

Crowd simulation

Artists are using swarm technology as a means of creating complex interactive systems or simulating crowds. Tim Burton's *Batman Returns* was the first movie to make use of swarm technology for rendering, realistically depicting the movements of a group of bats using the Boids system. The *Lord of the Rings* film trilogy made use of similar

technology, known as Massive, during battle scenes. Swarm technology is particularly attractive because it is cheap, robust, and simple.

There is hardly anything more crowded than the interior of an commercial aircraft. Swarm theory has been used in aircraft boarding. Over the years airlines have used computer simulations of passengers boarding a plane. Southwest Airlines researcher Douglas A. Lawson used an ant-based computer simulation employing only six interaction rules to evaluate boarding times using various boarding methods.(Miller, 2010, xii-xviii).

Ant-based routing

The use of Swarm Intelligence in Telecommunication Networks has also been researched, in the form of Ant Based Routing. This was pioneered separately by Dorigo et al. and Hewlett Packard in the mid-1990s, with a number of variations since. Basically this uses a probabilistic routing table rewarding/reinforcing the route successfully traversed by each "ant" (a small control packet) which flood the network. Reinforcement of the route in the forwards, reverse direction and both simultaneously have been researched: backwards reinforcement requires a symmetric network and couples the two directions together; forwards reinforcement rewards a route before the outcome is known (but then you pay for the cinema before you know how good the film is). As the system behaves stochastically and is therefore lacking repeatability, there are large hurdles to commercial deployment. Mobile media and new technologies have the potential to change the threshold for collective action due to swarm intelligence (Rheingold: 2002, P175).

Airlines have also used ant-based routing in assigning aircraft arrivals to airport gates. At Southwest Airlines a software program uses swarm theory, or swarm intelligence -- the idea that a colony of ants works better than one alone. Each pilot acts like an ant searching for the best airport gate. "The pilot learns from his experience what's the best for him, and it turns out that that's the best solution for the airline," Dr. Douglas A. Lawsonexplains. As a result, the "colony" of pilots always go to gates they can arrive and depart quickly. The program can even alert a pilot of plane back-ups before they happen. "We can anticipate that it's going to happen, so we'll have a gate available," Dr. Lawson says.

Chapter 4

Bayesian Network (Component of Soft Computing)

A **Bayesian network**, **belief network** or **directed acyclic graphical model** is a probabilistic graphical model that represents a set of random variables and their conditional dependencies via a directed acyclic graph (DAG). For example, a Bayesian network could represent the probabilistic relationships between diseases and symptoms. Given symptoms, the network can be used to compute the probabilities of the presence of various diseases.

Formally, Bayesian networks are directed acyclic graphs whose nodes represent random variables in the Bayesian sense: they may be observable quantities, latent variables, unknown parameters or hypotheses. Edges represent conditional dependencies; nodes which are not connected represent variables which are conditionally independent of each other. Each node is associated with a probability function that takes as input a particular set of values for the node's parent variables and gives the probability of the variable represented by the node. For example, if the parents are *m* Boolean variables then the probability function could be represented by a table of 2^m entries, one entry for each of the 2^m possible combinations of its parents being true or false.

Efficient algorithms exist that perform inference and learning in Bayesian networks. Bayesian networks that model sequences of variables (*e.g.* speech signals or protein sequences) are called dynamic Bayesian networks. Generalizations of Bayesian networks that can represent and solve decision problems under uncertainty are called influence diagrams.

Definitions and concepts

There are several equivalent definitions of a Bayesian network. For all the following, let $G = (V,E)$ be a directed acyclic graph (or DAG), and let $X = (X_v)_{v \in V}$ be a set of random variables indexed by V.

Factorization definition

X is a Bayesian network with respect to G if its joint probability density function (with respect to a product measure) can be written as a product of the individual density functions, conditional on their parent variables:

$$p(x) = \prod_{v \in V} p\left(x_v \mid x_{\mathrm{pa}(v)}\right)$$

where pa(v) is the set of parents of v (i.e. those vertices pointing directly to v via a single edge).

For any set of random variables, the probability of any member of a joint distribution can be calculated from conditional probabilities using the chain rule as follows:

$$\mathrm{P}(X_1 = x_1, \ldots, X_n = x_n) = \prod_{v=1}^{n} \mathrm{P}(X_v = x_v \mid X_{v+1} = x_{v+1}, \ldots, X_n = x_n)$$

Compare this with the definition above, which can be written as:

$$\mathrm{P}(X_1 = x_1, \ldots, X_n = x_n) = \prod_{v=1}^{n} \mathrm{P}(X_v = x_v \mid X_j = x_j \text{ for each } X_j \text{which is a parent of } X_v)$$

The difference between the two expressions is the conditional independence of the variables from any of their non-descendents, given the values of their parent variables.

Local Markov property

X is a Bayesian network with respect to G if it satisfies the *local Markov property*: each variable is conditionally independent of its non-descendants given its parent variables:

$$X_v \perp\!\!\!\perp X_{V \setminus \mathrm{de}(v)} \mid X_{\mathrm{pa}(v)} \quad \text{for all } v \in V$$

where de(v) is the set of descendants of v.

This can also be expressed in terms similar to the first definition, as

$$\mathrm{P}(X_v = x_v \mid X_i = x_i \text{for each } X_i \text{which is not a descendent of } X_v) = P(X_v = x_v \mid X_j = x_j \text{for each } X_j \text{which is a parent of } X_v)$$

Note that the set of parents is a subset of the set of non-descendants because the graph is acyclic.

Developing Bayesian Networks

To develop a Bayesian network, we often first develop a DAG G such that we believe X satisfies the local Markov property with respect to G. Sometimes this is done by creating

a causal DAG. We then ascertain the conditional probability distributions of each variable given its parents in G. In many cases, in particular in the case where the variables are discrete, if we define the joint distribution of X to be the product of these conditional distributions, then X is a Bayesian network with respect to G.

Markov blanket

The Markov blanket of a node is its set of neighboring nodes: its parents, its children, and any other parents of its children. X is a Bayesian network with respect to G if every node is conditionally independent of all other nodes in the network, given its Markov blanket.

d-separation

This definition can be made more general by defining the "d"-separation of two nodes. Let P be an undirected path (that is, a path which can go in either direction) from node u to v. Then P is said to be d-separated by a set of nodes Z if and only if (at least) one of the following holds:

1. P contains a *chain*, $i \rightarrow m \rightarrow j$, such that the middle node m is in Z,
2. P contains a *chain*, $i \leftarrow m \leftarrow j$, such that the middle node m is in Z,
3. P contains a *fork*, $i \leftarrow m \rightarrow j$, such that the middle node m is in Z, or
4. P contains an *inverted fork* (or *collider*), $i \rightarrow m \leftarrow j$, such that the middle node m is not in Z and no descendant of m is in Z.

Thus u and v are said to be d-separated by Z if all undirected paths between them are d-separated.

X is a Bayesian network with respect to G if, for any two nodes u, v:

$$X_u \perp\!\!\!\perp X_v \mid X_Z$$

where Z is a set which d-separates u and v. (The Markov blanket is the minimal set of nodes which d-separates node v from all other nodes.)

Causal networks

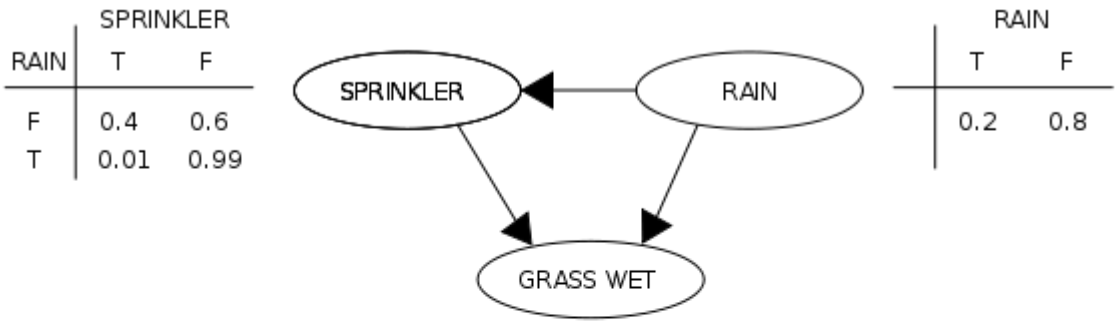
Although Bayesian networks are often used to represent causal relationships, this need not be the case: a directed edge from u to v does not require that X_v is causally dependent on X_u. This is demonstrated by the fact that Bayesian networks on the graphs:

$$a \longrightarrow b \longrightarrow c \qquad \text{and} \qquad a \longleftarrow b \longleftarrow c$$

are equivalent: that is they impose exactly the same conditional independence requirements.

A causal network is a Bayesian network with an explicit requirement that the relationships be causal. The additional semantics of the causal networks specify that if a node X is actively caused to be in a given state x (an action written as $do(X=x)$), then the probability density function changes to the one of the network obtained by cutting the links from X's parents to X, and setting X to the caused value x. Using these semantics, one can predict the impact of external interventions from data obtained prior to intervention.

Example



RAIN	SPRINKLER T	SPRINKLER F
F	0.4	0.6
T	0.01	0.99

RAIN T	RAIN F
0.2	0.8

SPRINKLER	RAIN	GRASS WET T	GRASS WET F
F	F	0.0	1.0
F	T	0.8	0.2
T	F	0.9	0.1
T	T	0.99	0.01

A simple Bayesian network

Suppose that there are two events which could cause grass to be wet: either the sprinkler is on or it's raining. Also, suppose that the rain has a direct effect on the use of the sprinkler (namely that when it rains, the sprinkler is usually not turned on). Then the situation can be modeled with a Bayesian network (shown). All three variables have two possible values, T (for true) and F (for false).

The joint probability function is:

$$\mathrm{P}(G,S,R) = \mathrm{P}(G \mid S,R)\mathrm{P}(S \mid R)\mathrm{P}(R)$$

where the names of the variables have been abbreviated to G = *Grass wet*, S = *Sprinkler*, and R = *Rain*.

The model can answer questions like "What is the probability that it is raining, given the grass is wet?" by using the conditional probability formula and summing over all nuisance variables:

$$\mathrm{P}(R = T \mid G = T) = \frac{\mathrm{P}(G = T, R = T)}{\mathrm{P}(G = T)} = \frac{\sum_{S \in \{T,F\}} \mathrm{P}(G = T, S, R = T)}{\sum_{S,R \in \{T,F\}} \mathrm{P}(G = T, S, R)}$$
$$= \frac{(0.99 \times 0.01 \times 0.2 = 0.00198_{TTT}) + (0.8 \times 0.99 \times 0.2 = 0.1584_{TFT})}{0.00198_{TTT} + 0.288_{TTF} + 0.1584_{TFT} + 0_{TFF}} \approx 35.77\%.$$

As in the example numerator is pointed out explicitly, the joint probability function is used to calculate each iteration of the summation function. In the numerator marginalizing over S and in the denominator marginalizing over S and R.

If, on the other hand, we wish to answer an interventional question: "What is the likelihood that it would rain, given that we wet the grass?" the answer would be governed by the post-intervention joint distribution function $\mathrm{P}(S,R \mid do(G = T)) = P(S \mid R)P(R)$ obtained by removing the factor $\mathrm{P}(G \mid S,R)$ from the pre-intervention distribution. As expected, the likelihood of rain is unaffected by the action: $\mathrm{P}(R \mid do(G = T)) = P(R)$.

If, moreover, we wish to predict the impact of turning the sprinkler on, we have $P(R,G \mid do(S = T)) = P(R)P(G \mid R,S = T)$ with the term $P(S = T \mid R)$ removed, showing that the action has an effect on the grass but not on the rain.

These predictions may not be feasible when some of the variables are unobserved, as in most policy evaluation problems. The effect of the action $do(x)$ can still be predicted, however, whenever a criterion called "back-door" is satisfied. It states that, if a set Z of nodes can be observed that d-separates (or blocks) all back-door paths from X to Y then $P(Y,Z \mid do(x)) = P(Y,Z,X = x) / P(X = x \mid Z)$. A back-door path is one that ends with an arrow into X. Sets that satisfy the back-door criterion are called "sufficient" or "admissible." For example, the set $Z=R$ is admissible for predicting the effect of $S=T$ on G, because R d-separate the (only) back-door path $S \leftarrow R \rightarrow G$. However, if S is not observed, there is no other set that d-separates this path and the effect of turning the sprinkler on ($S=T$) on the grass (G) cannot be predicted from passive observations. We then say that $P(G|do(S=T))$ is not "identified." This reflects the fact that, lacking interventional data, we cannot determine if the observed dependence between S and G is due to a causal connection or due to spurious created by a common cause, R.

Using a Bayesian network can save considerable amounts of memory, if the dependencies in the joint distribution are sparse. For example, a naive way of storing the conditional probabilities of 10 two-valued variables as a table requires storage space for $2^{10} = 1024$ values. If the local distributions of no variable depends on more than 3 parent variables, the Bayesian network representation only needs to store at most $10 * 2^3 = 80$ values.

One advantage of Bayesian networks is that it is intuitively easier for a human to understand (a sparse set of) direct dependencies and local distributions than complete joint distribution.

Inference and learning

There are three main inference tasks for Bayesian networks.

Inferring unobserved variables

Because a Bayesian network is a complete model for the variables and their relationships, it can be used to answer probabilistic queries about them. For example, the network can be used to find out updated knowledge of the state of a subset of variables when other variables (the *evidence* variables) are observed. This process of computing the *posterior* distribution of variables given evidence is called probabilistic inference. The posterior gives a universal sufficient statistic for detection applications, when one wants to choose values for the variable subset which minimize some expected loss function, for instance the probability of decision error. A Bayesian network can thus be considered a mechanism for automatically applying Bayes' theorem to complex problems.

The most common exact inference methods are: variable elimination, which eliminates (by integration or summation) the non-observed non-query variables one by one by distributing the sum over the product; clique tree propagation, which caches the computation so that many variables can be queried at one time and new evidence can be propagated quickly; and recursive conditioning and AND/OR search, which allow for a space-time tradeoff and match the efficiency of variable elimination when enough space is used. All of these methods have complexity that is exponential in the network's treewidth. The most common approximate inference algorithms are importance sampling, stochastic MCMC simulation, mini-bucket elimination, loopy belief propagation, generalized belief propagation, and variational methods.

Parameter learning

In order to fully specify the Bayesian network and thus fully represent the joint probability distribution, it is necessary to specify for each node X the probability distribution for X conditional upon X's parents. The distribution of X conditional upon its parents may have any form. It is common to work with discrete or Gaussian distributions since that simplifies calculations. Sometimes only constraints on a distribution are known; one can then use the principle of maximum entropy to determine a single distribution, the one with the greatest entropy given the constraints. (Analogously, in the specific context of a dynamic Bayesian network, one commonly specifies the conditional distribution for the hidden state's temporal evolution to maximize the entropy rate of the implied stochastic process.)

Often these conditional distributions include parameters which are unknown and must be estimated from data, sometimes using the maximum likelihood approach. Direct

maximization of the likelihood (or of the posterior probability) is often complex when there are unobserved variables. A classical approach to this problem is the expectation-maximization algorithm which alternates computing expected values of the unobserved variables conditional on observed data, with maximizing the complete likelihood (or posterior) assuming that previously computed expected values are correct. Under mild regularity conditions this process converges on maximum likelihood (or maximum posterior) values for parameters.

A more fully Bayesian approach to parameters is to treat parameters as additional unobserved variables and to compute a full posterior distribution over all nodes conditional upon observed data, then to integrate out the parameters. This approach can be expensive and lead to large dimension models, so in practice classical parameter-setting approaches are more common.

Structure learning

In the simplest case, a Bayesian network is specified by an expert and is then used to perform inference. In other applications the task of defining the network is too complex for humans. In this case the network structure and the parameters of the local distributions must be learned from data.

Automatically learning the graph structure of a Bayesian network is a challenge pursued within machine learning. The basic idea goes back to a recovery algorithm developed by Rebane and Pearl (1987) and rests on the distinction between the three possible types of adjacent triplets allowed in a directed acyclic graph (DAG):

1. $X \rightarrow Y \rightarrow Z$
2. $X \leftarrow Y \rightarrow Z$
3. $X \rightarrow Y \leftarrow Z$

Type 1 and type 2 represent the same dependencies (*X* and *Z* are independent given *Y*) and are, therefore, indistinguishable. Type 3, however, can be uniquely identified, since *X* and *Z* are marginally independent and all other pairs are dependent. Thus, while the *skeletons* (the graphs stripped of arrows) of these three triplets are identical, the directionality of the arrows is partially identifiable. The same distinction applies when *X* and *Z* have common parents, except that one must first condition on those parents. Algorithms have been developed to systematically determine the skeleton of the underlying graph and, then, orient all arrows whose directionality is dictated by the conditional independencies observed.

An alternative method of structural learning uses optimization based search. It requires a scoring function and a search strategy. A common scoring function is posterior probability of the structure given the training data. The time requirement of an exhaustive search returning back a structure that maximizes the score is superexponential in the number of variables. A local search strategy makes incremental changes aimed at improving the score of the structure. A global search algorithm like Markov chain Monte

Carlo can avoid getting trapped in local minima. Friedman et al. talk about using mutual information between variables and finding a structure that maximizes this. They do this by restricting the parent candidate set to *k* nodes and exhaustively searching therein.

Applications

Bayesian networks are used for modelling knowledge in computational biology and bioinformatics (gene regulatory networks, protein structure, gene expression analysis), medicine , document classification, information retrieval , image processing, data fusion, decision support systems, engineering, gaming and law.

History

The term "Bayesian networks" was coined by Judea Pearl in 1985 to emphasize three aspects:

1. The often subjective nature of the input information.
2. The reliance on Bayes's conditioning as the basis for updating information.
3. The distinction between causal and evidential modes of reasoning, which underscores Thomas Bayes' posthumously published paper of 1763.

In the late 1980s the seminal texts *Probabilistic Reasoning in Intelligent Systems* and *Probabilistic Reasoning in Expert Systems* summarized the properties of Bayesian networks and helped to establish Bayesian networks as a field of study.

Informal variants of such networks were first used by legal scholar John Henry Wigmore, in the form of Wigmore charts, to analyse trial evidence in 1913.[:66-76] Another variant, called path diagrams, was developed by the geneticist Sewall Wright and used in social and behavioral sciences (mostly with linear parametric models).

Chapter 5

Evolutionary Computation

In computer science, **evolutionary computation** is a subfield of artificial intelligence (more particularly computational intelligence) that involves combinatorial optimization problems.

Evolutionary computation uses iterative progress, such as growth or development in a population. This population is then selected in a guided random search using parallel processing to achieve the desired end. Such processes are often inspired by biological mechanisms of evolution.

History

The use of Darwinian principles for automated problem solving originated in the fifties. It was not until the sixties that three distinct interpretations of this idea started to be developed in three different places.

Evolutionary programming was introduced by Lawrence J. Fogel in the USA, while John Henry Holland called his method a genetic algorithm. In Germany Ingo Rechenberg and Hans-Paul Schwefel introduced evolution strategies. These areas developed separately for about 15 years. From the early nineties on they are unified as different representatives ("dialects") of one technology, called evolutionary computing. Also in the early nineties, a fourth stream following the general ideas had emerged – genetic programming.

These terminologies denote the field of evolutionary computing and consider evolutionary programming, evolution strategies, genetic algorithms, and genetic programming as sub-areas.

Techniques

Evolutionary techniques mostly involve metaheuristic optimization algorithms such as:

Learnable Evolution Model

The **Learnable Evolution Model** (LEM) is a novel, non-Darwinian methodology for evolutionary computation that employs machine learning to guide the generation of new individuals (candidate problem solutions). Unlike standard, Darwinian-type evolutionary computation methods that use random or semi-random operators for generating new individuals (such as mutations and/or recombinations), LEM employs hypothesis generation and instantiation operators.

The hypothesis generation operator applies a machine learning program to induce descriptions that distinguish between high-fitness and low-fitness individuals in each consecutive population. Such descriptions delineate areas in the search space that most likely contain the desirable solutions. Subsequently the instantiation operator samples these areas to create new individuals.

Learning classifier system

A **learning classifier system**, or LCS, is a machine learning system with close links to reinforcement learning and genetic algorithms. First described by John Holland, his LCS consisted of a population of binary rules on which a genetic algorithm altered and selected the best rules. Rule fitness was based on a reinforcement learning technique.

Learning classifier systems can be split into two types depending upon where the genetic algorithm acts. A Pittsburgh-type LCS has a population of separate rule sets, where the genetic algorithm recombines and reproduces the best of these rule sets. In a Michigan-style LCS there is only a single population and the algorithm's action focuses on selecting the best classifiers within that ruleset. Michigan-style LCSs have two main types of fitness definitions, strength-based (e.g. **ZCS**) and accuracy-based (e.g. **XCS**). The term "learning classifier system" most often refers to Michigan-style LCSs.

Initially the classifiers or rules were binary, but recent research has expanded this representation to include real-valued, neural network, and functional (S-expression) conditions.

Learning classifier systems are not fully understood mathematically and doing so remains an area of active research. Despite this, they have been successfully applied in many problem domains.

Harmony search

In computer science and operations research, **harmony search** (HS) is a phenomenon-mimicking algorithm (also known as metaheuristic algorithm, soft computing algorithm or evolutionary algorithm) inspired by the improvisation process of musicians. In the HS algorithm, each musician (= decision variable) plays (= generates) a note (= a value) for finding a best harmony (= global optimum) all together. The Harmony Search algorithm has the following merits:

- HS does not require differential gradients, thus it can consider discontinuous functions as well as continuous functions.
- HS can handle discrete variables as well as continuous variables.
- HS does not require initial value setting for the variables.
- HS is free from divergence.
- HS may escape local optima.
- HS may overcome the drawback of GA's building block theory which works well only if the relationship among variables in a chromosome is carefully considered. If neighbor variables in a chromosome have weaker relationship than remote variables, building block theory may not work well because of crossover operation. However, HS explicitly considers the relationship using ensemble operation.
- HS has a novel stochastic derivative applied to discrete variables, which uses musician's experiences as a searching direction.
- Certain HS variants do not require algorithm parameters such as HMCR and PAR, thus novice users can easily use the algorithm.

Basic Harmony Search Algorithm

Harmony search tries to find a vector $\mathbf{x}$ that optimizes (minimizes or maximizes) a certain objective function.

The algorithm has the following steps:

Step 1: Generate random vectors ($\mathbf{x}^1 \ldots \mathbf{x}^{hms}$) as many as *hms* (harmony memory size), then store them in harmony memory (HM)

$$\mathbf{HM} = \begin{bmatrix} x_1^1 & \cdots & x_n^1 & | & f(\mathbf{x}^1) \\ \vdots & \ddots & \vdots & | & \vdots \\ x_1^{hms} & \cdots & x_n^{hms} & | & f(\mathbf{x}^{hms}) \end{bmatrix}.$$

Step 2: Generate a new vector $\mathbf{x}'$. For each component x_i',

- with probability *hmcr* (harmony memory considering rate), pick the stored value from HM, $x_i^{'} \leftarrow x_i^{int(rand(0,1)*hms)+1}$
- with probability 1 − *hmcr*, pick a random value within the allowed range.

Step 3: Perform additional work if the value in Step 2 came from HM

- with probability *par* (pitch adjusting rate), change $x_i^{'}$ by a small amount: $x_i^{'} \leftarrow x_i^{'} \pm \delta$ for discrete variable; or $x_i^{'} \leftarrow x_i^{'} \pm fw \cdot rand(0,1)$ for continuous variable.
- with probability 1 − *par*, do nothing.

Step 4: If $\mathbf{x}^{'}$ is better than the worst vector $\mathbf{x}^{Worst}$ in HM, replace $\mathbf{x}^{Worst}$ with $\mathbf{x}^{'}$.

Step 5: Repeat from Step 2 to Step 4 until termination criterion (e.g. maximum iterations) is satisfied.

The parameters of the algorithm are

- *hms* = the size of the harmony memory. A typical value ranges from 1 to 100.
- *hmcr* = the rate of choosing a value from the harmony memory. Typical value ranges from 0.7 to 0.99.
- *par* = the rate of choosing a neighboring value. Typical value ranges from 0.1 to 0.5.
- δ = the amount between two neighboring values in discrete candidate set.
- *fw* (fret width, formerly bandwidth) = the amount of maximum change in pitch adjustment.

It is possible to vary the parameter values as the search progresses, which gives an effect similar to simulated annealing.

Parameter-setting-free researches have been also performed. In the researches, algorithm users do not need tedious parameter setting process.

Differential evolution

In computer science, **differential evolution** (DE) is a method that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. Such methods are commonly known as metaheuristics as they make few or no assumptions about the problem being optimized and can search very large spaces of

candidate solutions. However, metaheuristics such as DE do not guarantee an optimal solution is ever found.

DE is used for multidimensional real-valued functions but does not use the gradient of the problem being optimized, which means DE does not require for the optimization problem to be differentiable as is required by classic optimization methods such as gradient descent and quasi-newton methods. DE can therefore also be used on optimization problems that are not even continuous, are noisy, change over time, etc.

DE optimizes a problem by maintaining a population of candidate solutions and creating new candidate solutions by combining existing ones according to its simple formulae, and then keeping whichever candidate solution has the best score or fitness on the optimization problem at hand. In this way the optimization problem is treated as a black box that merely provides a measure of quality given a candidate solution and the gradient is therefore not needed.

DE is originally due to Storn and Price. Books have been published on theoretical and practical aspects of using DE in parallel computing, multiobjective optimization, constrained optimization, and the books also contain surveys of application areas .

Algorithm

A basic variant of the DE algorithm works by having a population of candidate solutions (called agents). These agents are moved around in the search-space by using simple mathematical formulae to combine the positions of existing agents from the population. If the new position of an agent is an improvement it is accepted and forms part of the population, otherwise the new position is simply discarded. The process is repeated and by doing so it is hoped, but not guaranteed, that a satisfactory solution will eventually be discovered.

Formally, let $f: \mathbb{R}^n \to \mathbb{R}$ be the fitness or cost function which must be minimized. The function takes a candidate solution as argument in the form of a vector of real numbers and produces a real number as output which indicates the fitness of the given candidate solution. The gradient of f is not known. The goal is to find a solution $\mathbf{m}$ for which $f(\mathbf{m}) \leq f(\mathbf{p})$ for all $\mathbf{p}$ in the search-space, which would mean $\mathbf{m}$ is the global minimum. Maximization can be performed by considering the function $h = -f$ instead.

Let $\mathbf{x} \in \mathbb{R}^n$ designate a candidate solution (agent) in the population. The basic DE algorithm can then be described as follows:

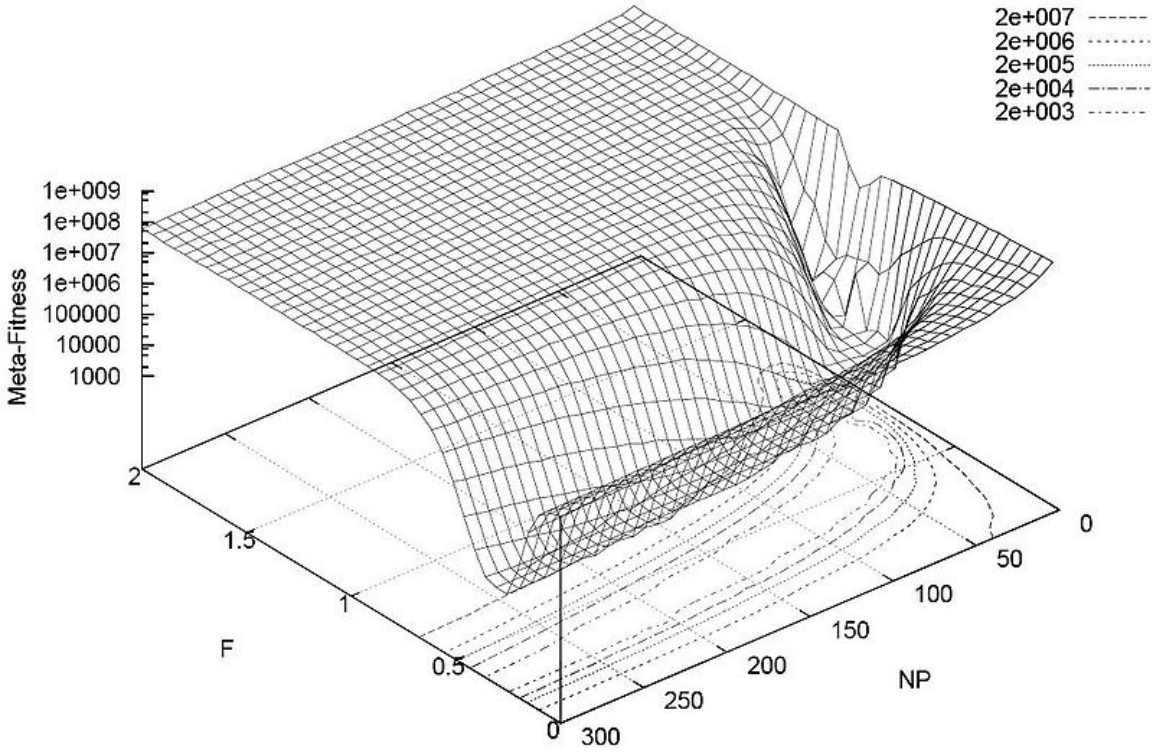
- Initialize all agents $\mathbf{x}$ with random positions in the search-space.
- Until a termination criterion is met (e.g. number of iterations performed, or adequate fitness reached), repeat the following:
    - For each agent $\mathbf{x}$ in the population do:
        - Pick three agents $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{c}$ from the population at random, they must be distinct from each other as well as from agent $\mathbf{x}$

- Pick a random index $R \in \{1, ..., n\}$, where the highest possible value n is the dimensionality of the problem to be optimized.
- Compute the agent's potentially new position $\mathbf{y} = [y_1, ..., y_n]$ by iterating over each $i \in \{1, ..., n\}$ as follows:
  - Pick $r_i \sim U(0,1)$ uniformly from the open range (0,1)
  - If $(i=R)$ or $(r_i<CR)$ let $y_i = a_i + F(b_i - c_i)$, otherwise let $y_i = x_i$
  - If $(f(\mathbf{y}) < f(\mathbf{x}))$ then replace the agent in the population with the improved candidate solution, that is, set $\mathbf{x} = \mathbf{y}$ in the population.

- Pick the agent from the population that has the lowest fitness and return it as the best found candidate solution.

Note that $F \in [0,2]$ is called the *differential weight* and $CR \in [0,1]$ is called the *crossover probability*, both these parameters are selectable by the practitioner along with the population size $NP > 3$, see below.

Parameter tuning



Performance landscape showing how the basic DE performs in aggregate on the Sphere and Rosenbrock benchmark problems when varying the two DE parameters *NP* and *F*, and keeping fixed *CR*=0.9.

The choice of DE parameters *F*, *CR* and *NP* can have a large impact on optimization performance. Selecting the DE parameters that yield good performance has therefore been the subject of much research, e.g. Storn et al. , Liu and Lampinen , and Zaharie .

Variants

Variants of the basic DE algorithm are continually being developed in an effort to improve optimization performance. A popular research trend has been to devise schemes for perturbing or adapting the DE parameters during optimization, e.g. Price et al. , Liu and Lampinen , Qin and Suganthan , and Brest et al. .

History

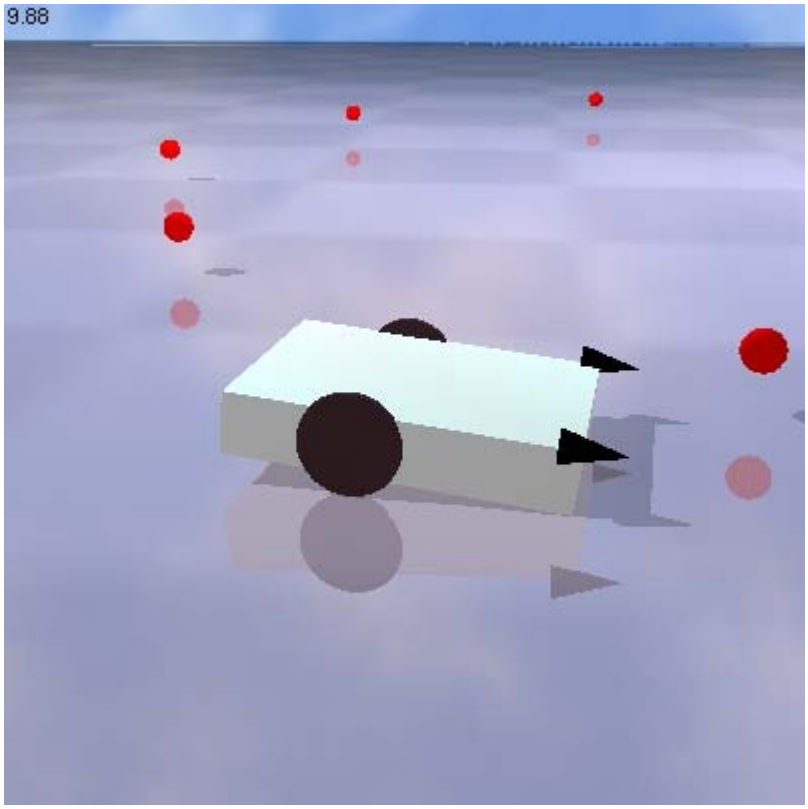
DE grew out of Kenneth Price's attempts to solve the Chebyshev polynomial fitting problem that had been posed to him by Rainer Storn. This was done by modifying Genetic Annealing originally developed by Price to use floating-point instead of bit-string encoding and arithmetic vector operations instead of logical ones.

The success of DE was demonstrated at the First International Contest on Evolutionary Optimization in May 1996, which was held in conjunction with the 1996 IEEE International Conference on Evolutionary Computation. DE won third place on the proposed benchmarks.

Chapter 6

Artificial Life

Artificial life (commonly **Alife** or **alife**) is a field of study and an associated art form which examine systems related to life, its processes, and its evolution through simulations using computer models, robotics, and biochemistry. The discipline was named by Christopher Langton, an American computer scientist, in 1986. There are three main kinds of alife, named for their approaches: *soft*, from software; *hard*, from hardware; and *wet*, from biochemistry. Artificial life imitates traditional biology by trying to *recreate* biological phenomena. The term "artificial life" is often used to specifically refer to soft alife.



A Braitenberg simulation, programmed in breve, an artificial life simulator

Overview

Artificial life studies the logic of living systems in artificial environments. The goal is to study the phenomena of living systems in order to come to an understanding of the complex information processing that defines such systems.

Also sometimes included in the umbrella term Artificial Life are agent based systems which are used to study the emergent properties of societies of agents.

While artificial life is, by definition, alive, artificial life is generally referred to as being confined to a digital environment and existence.

Philosophy

The modeling philosophy of alife strongly differs from traditional modeling, by studying not only “life-as-we-know-it”, but also “life-as-it-might-be”.

In the first approach, a traditional model of a biological system will focus on capturing its most important parameters. In contrast, an alife modeling approach will generally seek to decipher the most simple and general principles underlying life and implement them in a simulation. The simulation then offers the possibility to analyse new, different life-like systems.

Red'ko proposed to generalize this distinction not just to the modeling of life, but to any process. This led to the more general distinction of "processes-as-we-know-them" and "processes-as-they-could-be"

At present, the commonly accepted definition of life does not consider any current alife simulations or softwares to be alive, and they do not constitute part of the evolutionary process of any ecosystem. However, different opinions about artificial life's potential have arisen:

- The *strong alife* (cf. Strong AI) position states that "life is a process which can be abstracted away from any particular medium" (John von Neumann). Notably, Tom Ray declared that his program Tierra is not simulating life in a computer but synthesizing it.

- The *weak alife* position denies the possibility of generating a "living process" outside of a chemical solution. Its researchers try instead to simulate life processes to understand the underlying mechanics of biological phenomena.

International Society of Artificial Life

ISAL is a "democratic, international, professional society dedicated to promoting scientific research and education relating to artificial life, including sponsoring

conferences, publishing scientific journals and newsletters, and maintaining web sites related to artificial life."

The ISAL organizes a biannual professional conference on artificial life called the *International Conference on Artificial Life*. Each conference is uniquely identified with a roman numeral. One such conference is Alife XI, to be held in August 2008 in Winchester, England.

The ISAL also publishes the preeminent artificial life scholarly journal *Artificial Life* through MIT Press.

Biota.org

Biota.org is run by Tom Barbalet, and "promotes and assists the engineering of complete, biologically-inspired, synthetic ecosystems and organisms" . Biota.org ran an annual Digital Biota Conference Series from 1996 to 2001. More recently, Biota.org has hosted a "collection of interviews, conference lectures and conversations with artificial life developers, academics and users" through a podcast.

Grey Thumb Society

The **Grey Thumb Society** is a group of "scientists, engineers, hackers, artists, and hobbyists... with a strong interest in artificial life, artificial intelligence, biology, complex systems, and other related topics" based in Boston, Massachusetts. Members also run a blog of artificial life related topics.

Software-based - "soft"

Techniques

Cellular automaton

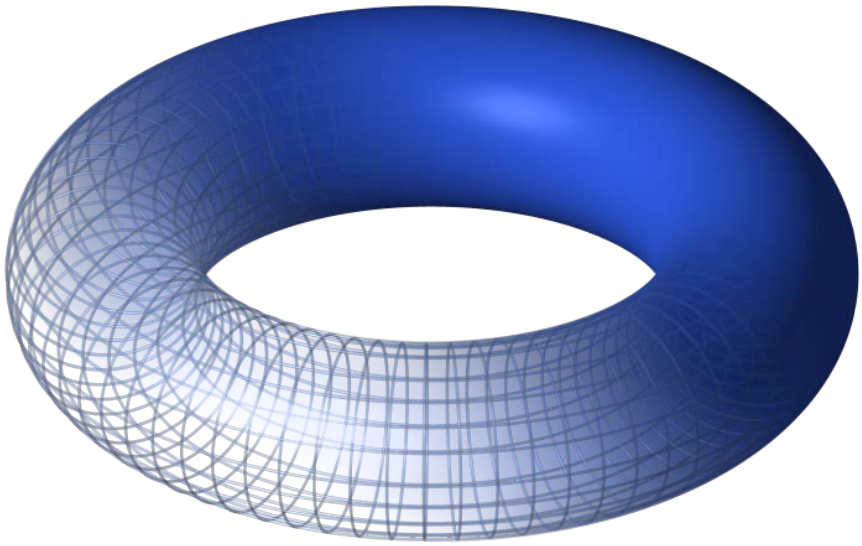
Gosper's Glider Gun creating "gliders" in the cellular automaton Conway's Game of Life.

A **cellular automaton** (pl. **cellular automata**, abbrev. **CA**) is a discrete model studied in computability theory, mathematics, physics, complexity science, theoretical biology and microstructure modeling. It consists of a regular grid of *cells*, each in one of a finite number of *states*, such as "On" and "Off" (in contrast to a coupled map lattice). The grid can be in any finite number of dimensions. For each cell, a set of cells called its *neighborhood* (usually including the cell itself) is defined relative to the specified cell. For example, the neighborhood of a cell might be defined as the set of cells a distance of 2 or less from the cell. An initial state (time t=0) is selected by assigning a state for each cell. A new *generation* is created (advancing t by 1), according to some fixed rule (generally, a mathematical function) that determines the new state of each cell in terms of the current state of the cell and the states of the cells in its neighborhood. For example, the rule might be that the cell is "On" in the next generation if exactly two of the cells in the neighborhood are "On" in the current generation, otherwise the cell is "Off" in the next generation. Typically, the rule for updating the state of cells is the same for each cell and does not change over time, and is applied to the whole grid simultaneously, though exceptions are known.

Overview

One way to simulate a two-dimensional cellular automaton is with an infinite sheet of graph paper along with a set of rules for the cells to follow. Each square is called a "cell" and each cell has two possible states, black and white. The "neighbors" of a cell are the 8 squares touching it. For such a cell and its neighbors, there are 512 (= 2^9) possible patterns. For each of the 512 possible patterns, the rule table would state whether the center cell will be black or white on the next time interval. Conway's Game of Life is a popular version of this model.

It is usually assumed that every cell in the universe starts in the same state, except for a finite number of cells in other states, often called a *configuration*. More generally, it is sometimes assumed that the universe starts out covered with a periodic pattern, and only a finite number of cells violate that pattern. The latter assumption is common in one-dimensional cellular automata.



A torus, a toroidal shape

Cellular automata are often simulated on a finite grid rather than an infinite one. In two dimensions, the universe would be a rectangle instead of an infinite plane. The obvious problem with finite grids is how to handle the cells on the edges. How they are handled will affect the values of all the cells in the grid. One possible method is to allow the values in those cells to remain constant. Another method is to define neighbourhoods differently for these cells. One could say that they have fewer neighbours, but then one would also have to define new rules for the cells located on the edges. These cells are usually handled with a *toroidal* arrangement: when one goes off the top, one comes in at the corresponding position on the bottom, and when one goes off the left, one comes in on the right. (This essentially simulates an infinite periodic tiling, and in the field of partial differential equations is sometimes referred to as *periodic* boundary conditions.) This can be visualized as taping the left and right edges of the rectangle to form a tube, then taping the top and bottom edges of the tube to form a torus (doughnut shape). Universes of other dimensions are handled similarly. This is done in order to solve boundary problems with neighborhoods, but another advantage of this system is that it is easily programmable using modular arithmetic functions. For example, in a 1-dimensional cellular automaton like the examples below, the neighborhood of a cell x_i^t—where t is the time step (vertical), and i is the index (horizontal) in one generation—is $\{x_{i-1}^{t-1}, x_i^{t-1}, x_{i+1}^{t-1}\}$. There will obviously be problems when a neighbourhood on a left border references its upper left cell, which is not in the cellular space, as part of its neighborhood.

History

Stanisław Ulam, while working at the Los Alamos National Laboratory in the 1940s, studied the growth of crystals, using a simple lattice network as his model. At the same

time, John von Neumann, Ulam's colleague at Los Alamos, was working on the problem of self-replicating systems. Von Neumann's initial design was founded upon the notion of one robot building another robot. This design is known as the kinematic model. As he developed this design, von Neumann came to realize the great difficulty of building a self-replicating robot, and of the great cost in providing the robot with a "sea of parts" from which to build its replicant. Ulam suggested that von Neumann develop his design around a mathematical abstraction, such as the one Ulam used to study crystal growth. Thus was born the first system of cellular automata. Like Ulam's lattice network, von Neumann's cellular automata are two-dimensional, with his self-replicator implemented algorithmically. The result was a universal copier and constructor working within a CA with a small neighborhood (only those cells that touch are neighbors; for von Neumann's cellular automata, only orthogonal cells), and with 29 states per cell. Von Neumann gave an existence proof that a particular pattern would make endless copies of itself within the given cellular universe. This design is known as the tessellation model, and is called a von Neumann universal constructor.

Also in the 1940s, Norbert Wiener and Arturo Rosenblueth developed a cellular automaton model of excitable media. Their specific motivation was the mathematical description of impulse conduction in cardiac systems. Their original work continues to be cited in modern research publications on cardiac arrhythmia and excitable systems.

In the 1960s, cellular automata were studied as a particular type of dynamical system and the connection with the mathematical field of symbolic dynamics was established for the first time. In 1969, G. A. Hedlund compiled many results following this point of view in what is still considered as a seminal paper for the mathematical study of cellular automata. The most fundamental result is the characterization of the set of global rules of cellular automata as the set of continuous endomorphisms of shift spaces.

In the 1970s a two-state, two-dimensional cellular automaton named Game of Life became very widely known, particularly among the early computing community. Invented by John Conway and popularized by Martin Gardner in a *Scientific American* article, its rules are as follows: If a black cell has 2 or 3 black neighbors, it stays black. If a black cell has less than 2 or more than 3 black neighbors it becomes white. If a white cell has 3 black neighbors, it becomes black. Despite its simplicity, the system achieves an impressive diversity of behavior, fluctuating between apparent randomness and order. One of the most apparent features of the Game of Life is the frequent occurrence of *gliders*, arrangements of cells that essentially move themselves across the grid. It is possible to arrange the automaton so that the gliders interact to perform computations, and after much effort it has been shown that the Game of Life can emulate a universal Turing machine. Possibly because it was viewed as a largely recreational topic, little follow-up work was done outside of investigating the particularities of the Game of Life and a few related rules.

In 1969, however, German computer pioneer Konrad Zuse published his book *Calculating Space*, proposing that the physical laws of the universe are discrete by

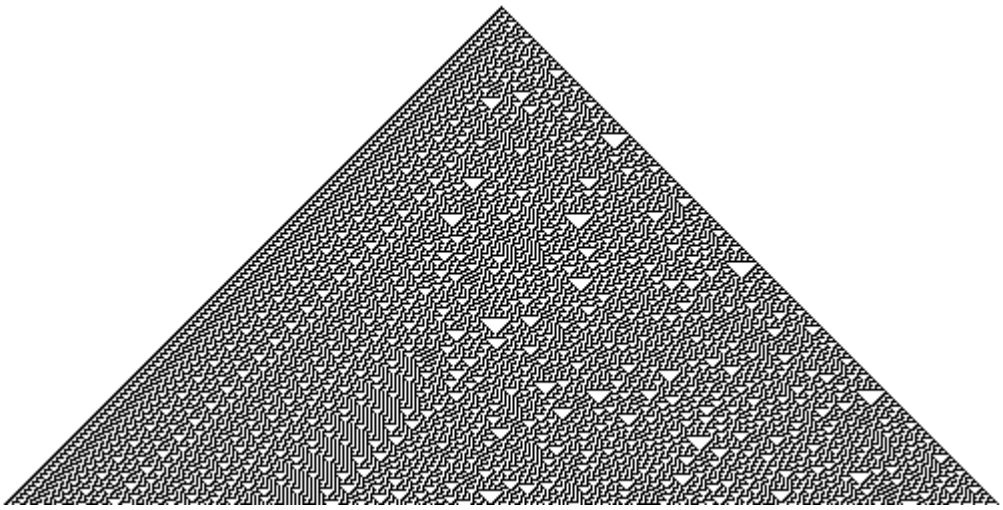
nature, and that the entire universe is the output of a deterministic computation on a giant cellular automaton. This was the first book on what today is called digital physics.

In 1983 Stephen Wolfram published the first of a series of papers systematically investigating a very basic but essentially unknown class of cellular automata, which he terms *elementary cellular automata* (see below). The unexpected complexity of the behavior of these simple rules led Wolfram to suspect that complexity in nature may be due to similar mechanisms. Additionally, during this period Wolfram formulated the concepts of intrinsic randomness and computational irreducibility, and suggested that rule 110 may be universal—a fact proved later by Wolfram's research assistant Matthew Cook in the 1990s.

In 2002 Wolfram published a 1280-page text *A New Kind of Science*, which extensively argues that the discoveries about cellular automata are not isolated facts but are robust and have significance for all disciplines of science. Despite much confusion in the press and academia, the book did not argue for a fundamental theory of physics based on cellular automata, and although it did describe a few specific physical models based on cellular automata, it also provided models based on qualitatively different abstract systems.

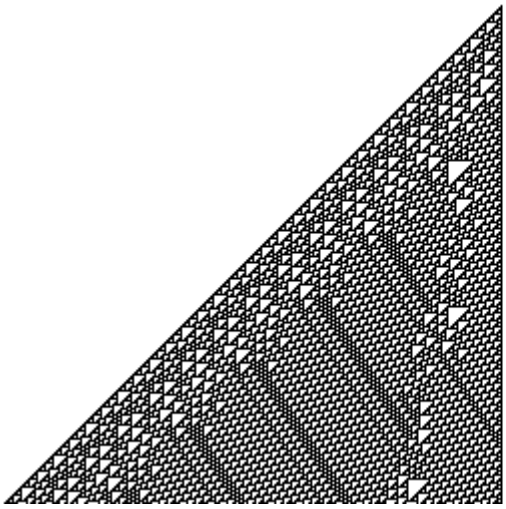
Elementary cellular automata

The simplest nontrivial CA would be one-dimensional, with two possible states per cell, and a cell's neighbors defined to be the adjacent cells on either side of it. A cell and its two neighbors form a neighborhood of 3 cells, so there are 2^3=8 possible patterns for a neighborhood. A rule consists of deciding, for each pattern, whether the cell will be a 1 or a 0 in the next generation. There are then 2^8=256 possible rules. These 256 CAs are generally referred to by their Wolfram code, a standard naming convention invented by Stephen Wolfram which gives each rule a number from 0 to 255. A number of papers have analyzed and compared these 256 CAs. The rule 30 and rule 110 CAs are particularly interesting. The images below show the history of each when the starting configuration consists of a 1 (in the center of each image) surrounded by 0's. Each row of pixels represents a generation in the history of the automation, with t=0 being the top row. Each pixel is colored white for 0 and black for 1.



Rule 30 cellular automaton

current pattern	111	110	101	100	011	010	001	000
new state for center cell	0	0	0	1	1	1	1	0



Rule 110 cellular automaton

current pattern	111	110	101	100	011	010	001	000
new state for center cell	0	1	1	0	1	1	1	0

Rule 30 generates exhibits *class 3* behavior, meaning even simple input patterns such as that shown lead to chaotic, seemingly random histories.

Rule 110, like the Game of Life, exhibits what Wolfram calls *class 4* behavior, which is neither completely random nor completely repetitive. Localized structures appear and interact in various complicated-looking ways. In the course of the development of *A New Kind of Science*, as a research assistant to Stephen Wolfram in 1994, Matthew Cook proved that some of these structures were rich enough to support universality. This result

is interesting because rule 110 is an extremely simple one-dimensional system, and one which is difficult to engineer to perform specific behavior. This result therefore provides significant support for Wolfram's view that class 4 systems are inherently likely to be universal. Cook presented his proof at a Santa Fe Institute conference on Cellular Automata in 1998, but Wolfram blocked the proof from being included in the conference proceedings, as Wolfram did not want the proof to be announced before the publication of *A New Kind of Science*. In 2004, Cook's proof was finally published in Wolfram's journal Complex Systems (Vol. 15, No. 1), over ten years after Cook came up with it. Rule 110 has been the basis over which some of the smallest universal Turing machines have been built, inspired on the breakthrough concepts that the development of the proof of rule 110 universality produced.

Reversible

A CA is said to be *reversible* if for every current configuration of the CA there is exactly one past configuration (preimage). If one thinks of a CA as a function mapping configurations to configurations, reversibility implies that this function is bijective.

For one dimensional CA there are known algorithms for deciding whether a rule is reversible or irreversible . For CA of two or more dimensions it has been proved that the reversibility is undecidable for arbitrary rules. The proof by Jarkko Kari is related to the tiling problem by Wang tiles.

Reversible CA are often used to simulate such physical phenomena as gas and fluid dynamics, since they obey the laws of thermodynamics. Such CA have rules specially constructed to be reversible. Such systems have been studied by Tommaso Toffoli, Norman Margolus and others.

For finite CAs that are not reversible, there must exist patterns for which there are no previous states. These patterns are called *Garden of Eden patterns*. In other words, no pattern exists which will develop into a Garden of Eden pattern.

Several techniques can be used to explicitly construct reversible CA with known inverses. Two common ones are the second order technique and the partitioning technique, both of which involve modifying the definition of a CA in some way. Although such automata do not strictly satisfy the definition given above, it can be shown that they can be emulated by conventional CAs with sufficiently large neighborhoods and numbers of states, and can therefore be considered a subset of conventional CA.

Totalistic

A special class of CAs are *totalistic* CAs. The state of each cell in a totalistic CA is represented by a number (usually an integer value drawn from a finite set), and the value of a cell at time t depends only on the *sum* of the values of the cells in its neighborhood (possibly including the cell itself) at time $t-1$. If the state of the cell at time t does depend

on its own state at time $t-1$ then the CA is properly called *outer totalistic*, although the distinction is not always made. Conway's Game of Life is an example of an outer totalistic CA with cell values 0 and 1.

A notation exists to describe rulesets of two-state totalistic CAs consisting of an initial indicating the neighbourhood of each cell and sums following the letters S (for survival) and B (for birth) for which those changes occur. In this notation Conway's Game of Life is M:S23/B3. This notation has been extended for non-totalistic CAs, where a letter or letters follow each sum indicating what patterns of neighbours cause survival or birth events.

Wolfram classes

Stephan Wolfram, in A New Kind of Science and in several papers dating from the mid-1980s, defined four classes into which cellular automata and several other simple computational models can be divided depending on their behavior. While earlier studies in cellular automata tended to try to identify type of patterns for specific rules, Wolfram's classification was the first attempt to classify the rules themselves. In order of complexity the classes are:

- Class 1: Nearly all initial patterns evolve quickly into a stable, homogeneous state. Any randomness in the initial pattern disappears.
- Class 2: Nearly all initial patterns evolve quickly into stable or oscillating structures. Some of the randomness in the initial pattern may filter out, but some remains. Local changes to the initial pattern tend to remain local.
- Class 3: Nearly all initial patterns evolve in a pseudo-random or chaotic manner. Any stable structures that appear are quickly destroyed by the surrounding noise. Local changes to the initial pattern tend to spread indefinitely.
- Class 4: Nearly all initial patterns evolve into structures that interact in complex and interesting ways. Class 2 type stable or oscillating structures may be the eventual outcome, but the number of steps required to reach this state may be very large, even when the initial pattern is relatively simple. Local changes to the initial pattern may spread indefinitely. Wolfram has conjectured that many, if not all class 4 cellular automata are capable of universal computation. This has been proven for Rule 110 and Conway's game of life.

These definitions are qualitative in nature and there is some room for interpretation. According to Wolfram,

...with almost any general classification scheme there are inevitably cases which get assigned to one class by one definition and another class by another definition. And so it is with cellular automata: there are occasionally rules...that show some features of one class and some of another.

Evolving cellular automata using genetic algorithms

Recently there has been a keen interest in building decentralized systems, be they sensor networks or more sophisticated micro level structures designed at the network level and aimed at decentralized information processing. The idea of emergent computation came from the need of using distributed systems to do information processing at the global level. The area is still in its infancy, but some people have started taking the idea seriously. Melanie Mitchell who is Professor of Computer Science at Portland State University and also the Santa Fe Institute External professor has been working on the idea of using self-evolving cellular arrays to study emergent computation and distributed information processing. Mitchell and colleagues are using evolutionary computation to program cellular arrays. Computation in decentralized systems is very different from classical systems, where the information is processed at some central location depending on the system's state. In decentralized system, the information processing occurs in the form of global and local pattern dynamics.

The inspiration for this approach comes from complex natural systems like insect colonies, nervous system and economic systems. The focus of the research is to understand how computation occurs in an evolving decentralized system. In order to model some of the features of these systems and study how they give rise to emergent computation, Mitchell and collaborators at the SFI have applied Genetic Algorithms to evolve patterns in cellular automata. They have been able to show that the GA discovered rules that gave rise to sophisticated emergent computational strategies. Mitchell's group used a single dimensional binary array where each cell has two neighbors. The array can be thought of as a circle where the first and last cells are neighbors. The evolution of the array was tracked through the number of ones and zeros after each iteration. The results were plotted to show clearly how the network evolved and what sort of emergent computation was visible.

The results produced by Mitchell's group are interesting, in that a very simple array of cellular automata produced results showing coordination over global scale, fitting the idea of emergent computation. Future work in the area may include more sophisticated models using cellular automata of higher dimensions, which can be used to model complex natural systems.

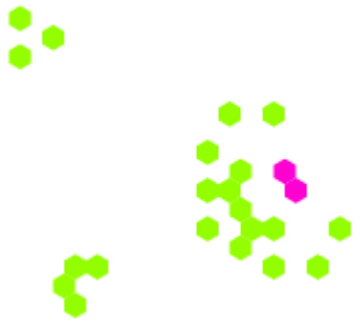
Cryptography use

Rule 30 was originally suggested as a possible Block cipher for use in cryptography.

Cellular automata have been proposed for public key cryptography. The one way function is the evolution of a finite CA whose inverse is believed to be hard to find. Given the rule, anyone can easily calculate future states, but it appears to be very difficult to calculate previous states. However, the designer of the rule can create it in such a way as to be able to easily invert it. Therefore, it is apparently a trapdoor function, and can be used as a public-key cryptosystem. The security of such systems is not currently known.

Related automata

There are many possible generalizations of the CA concept.



A cellular automaton based on hexagonal cells instead of squares (rule 34/2)

One way is by using something other than a rectangular (cubic, *etc.*) grid. For example, if a plane is tiled with regular hexagons, those hexagons could be used as cells. In many cases the resulting cellular automata are equivalent to those with rectangular grids with specially designed neighborhoods and rules.

Also, rules can be probabilistic rather than deterministic. A probabilistic rule gives, for each pattern at time t, the probabilities that the central cell will transition to each possible state at time t+1. Sometimes a simpler rule is used; for example: "The rule is the Game of Life, but on each time step there is a 0.001% probability that each cell will transition to the opposite color."

The neighborhood or rules could change over time or space. For example, initially the new state of a cell could be determined by the horizontally adjacent cells, but for the next generation the vertical cells would be used.

The grid can be finite, so that patterns can "fall off" the edge of the universe.

In CA, the new state of a cell is not affected by the new state of other cells. This could be changed so that, for instance, a 2 by 2 block of cells can be determined by itself and the cells adjacent to itself.

There are *continuous automata*. These are like totalistic CA, but instead of the rule and states being discrete (*e.g.* a table, using states {0,1,2}), continuous functions are used, and the states become continuous (usually values in [0,1]). The state of a location is a finite number of real numbers. Certain CA can yield diffusion in liquid patterns in this way.

Continuous spatial automata have a continuum of locations. The state of a location is a finite number of real numbers. Time is also continuous, and the state evolves according to

differential equations. One important example is reaction-diffusion textures, differential equations proposed by Alan Turing to explain how chemical reactions could create the stripes on zebras and spots on leopards. When these are approximated by CA, such CAs often yield similar patterns. MacLennan considers continuous spatial automata as a model of computation.

There are known examples of continuous spatial automata which exhibit propagating phenomena analogous to gliders in the Game of Life.

Natural biotic types

Conus textile exhibits a cellular automaton pattern on its shell

Some living things use naturally occurring cellular automata in their functioning.

Patterns of some seashells, like the ones in *Conus* and *Cymbiola* genus, are generated by natural CA. The pigment cells reside in a narrow band along the shell's lip. Each cell secretes pigments according to the activating and inhibiting activity of its neighbour pigment cells, obeying a natural version of a mathematical rule. The cell band leaves the colored pattern on the shell as it grows slowly. For example, the widespread species *Conus textile* bears a pattern resembling the Rule 30 CA described above.

Plants regulate their intake and loss of gases via a CA mechanism. Each stoma on the leaf acts as a cell.

Neural networks can be used as cellular automata, too. The complex moving wave patterns on the skin of cephalopods are a good display of corresponding activation patterns in the animals' brain.

Chemical types

The Belousov-Zhabotinsky reaction is a spatio-temporal chemical oscillator which can be simulated by means of a cellular automaton. In the 1950s A. M. Zhabotinsky (extending the work of B. P. Belousov) discovered that when a thin, homogenous layer of a mixture of malonic acid, acidified bromate and a ceric salt were mixed together and left undisturbed, fascinating geometric patterns such as concentric circles and spirals propagate across the medium. In the "Computer Recreations" section of the August 1988 issue of Scientific American, A. K. Dewdney discussed a cellular automaton which was developed by Martin Gerhardt and Heike Schuster of the University of Bielefeld (West Germany). This automaton produces wave patterns resembling those in the Belousov-Zhabotinsky reaction.

Computer processors

CA processors are a physical, not software only, implementation of CA concepts, which can process information computationally. Processing elements are arranged in a regular grid of identical cells. The grid is usually a square tiling, or tessellation, of two or three dimensions; other tilings are possible, but not yet used. Cell states are determined only by interactions with the small number of adjoining cells. Cells interact, communicate, directly only with adjoining, adjacent, neighbor cells. No means exists to communicate directly with cells farther away.

One such CA processor array configuration is the systolic array.

Cell interaction can be via electric charge, magnetism, vibration (phonons at quantum scales), or any other physically useful means. This can be done in several ways so no wires are needed between any elements.

This is very unlike processors used in most computers today, von Neumann designs, which are divided into sections with elements that can communicate with distant elements, over wires.

Error correction coding

CA have been applied to design error correction codes in the paper "Design of CAECC – Cellular Automata Based Error Correcting Code", by D. Roy Chowdhury, S. Basu, I. Sen

Gupta, P. Pal Chaudhuri. The paper defines a new scheme of building SEC-DED codes using CA, and also reports a fast hardware decoder for the code.

CA as model of the fundamental physical reality

As Andrew Ilachinski points out in his *Cellular Automata*, many scholars with some interest for fundamental issues have raised the question: 'is the universe a CA?' Ilachinsky argues that the importance of this question may be better appreciated with a simple observation, which can be stated as follows. Consider the evolution of Rule 110: if it were some kind of "alien physics," what would be a reasonable description of the observed patterns? If you didn't know how the images were generated, you might end up conjecturing about the movement of some particle-like objects (indeed, physicist Jim Crutchfield made a rigorous mathematical theory out of this idea proving the statistical emergence of "particles" from CA). Then, as the argument goes, one might wonder if *our* world, which is currently well described by physics with particle-like objects, could be a CA at its most fundamental level?

While a complete theory along this line is still to be developed, entertaining and developing this hypothesis led scholars to interesting speculation and fruitful intuitions on how can we make sense of our world within a discrete framework. Marvin Minsky – the AI pioneer – investigated how to understand particle interaction with a four-dimensional CA lattice; Konrad Zuse – the first who actually built a universal PC – developed an irregularly organized lattice to address the question of the information content of particles. More recently, Edward Fredkin exposed what he terms the "finite nature hypothesis", i.e. the idea that 'ultimately every quantity of physics, including space and time, will turn out to be discrete and finite.' Fredkin and Stephen Wolfram are strong proponents of a CA-based physics.

In recent years, other suggestions along these lines emerged from the literature in non-standard computation. Stephen Wolfram's A New Kind of Science considers CA to be the key to understand a variety of subjects, physics included. The *Mathematics Of the Models of Reference* – created by iLabs founder Gabriele Rossi and developed with Francesco Berto and Jacopo Tagliabue – features an original 2D/3D universe based on a new "rhombic dodecahedron-based" lattice and a unique rule: this model satisfies universality (it is equivalent to a Turing Machine) and perfect reversibility (a *desideratum* if one wants to conserve various quantities easily and never lose information), and it comes embedded in a first-order theory allowing computable, qualitative statements on the universe evolution.

Notable simulators

This is a list of Artificial life/Digital organism simulators, organized by the method of creature definition.

Program-based

These contain organisms with a complex DNA language, usually Turing complete. This language is more often in the form of a computer program than actual biological DNA. Assembly derivatives are the most common languages used. Use of cellular automata is common but not required.

- Tierra (early 1990s)
- Avida (1993-present)
- Evolve 4.0 (1996–2007)
- Darwinbots (2003–2008)
- Framsticks (1996–present)
- breve, a multi-purpose simulation environment (2006–present)
- DigiHive (2006–2009)

Module-based

Individual modules are added to a creature. These modules modify the creature's behaviors and characteristics either directly, by hard coding into the simulation (leg type A increases speed and metabolism), or indirectly, through the emergent interactions between a creature's modules (leg type A moves up and down with a frequency of X, which interacts with other legs to create motion). Generally these are simulators which emphasize user creation and accessibility over mutation and evolution.

- TechnoSphere

Parameter-based

Organisms are generally constructed with pre-defined and fixed behaviors that are controlled by various parameters that mutate. That is, each organism contains a collection of numbers or other *finite* parameters. Each parameter controls one or several aspects of an organism in a well-defined way.

- Kyresoo Plants

Neural net–based

These simulations have creatures that learn and grow using neural nets or a close derivative. Emphasis is often, although not always, more on learning than on natural selection.

- Creatures (a game)
- Noble Ape
- Polyworld

Hardware-based - "hard"

Hardware-based artificial life mainly consist of *robots*, that is, automatically guided machines, able to do tasks on their own.

Biochemical-based - "wet"

Biochemical-based life is studied in the field of synthetic biology. It involves e.g. the creation of synthetic DNA. The term "wet" is an extension of the term "wetware".

Related subjects

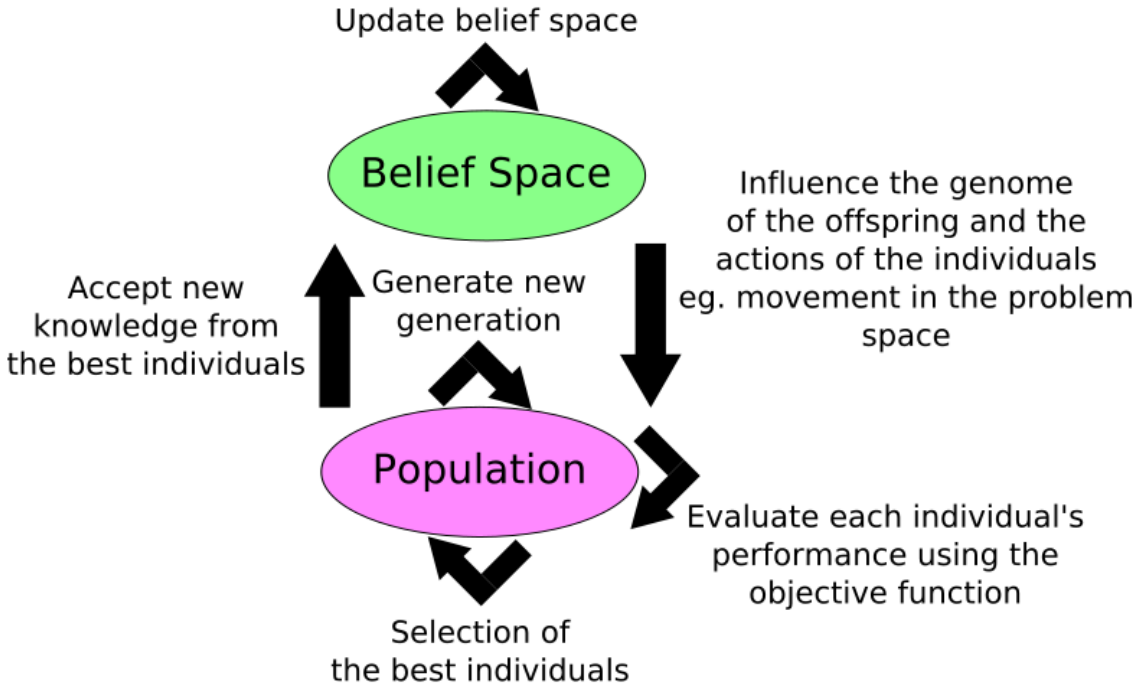
1. Artificial intelligence has traditionally used a top down approach, while alife generally works from the bottom up.
2. Artificial chemistry started as a method within the alife community to abstract the processes of chemical reactions.
3. Evolutionary algorithms are a practical application of the weak alife principle applied to optimization problems. Many optimization algorithms have been crafted which borrow from or closely mirror alife techniques. The primary difference lies in explicitly defining the fitness of an agent by its ability to solve a problem, instead of its ability to find food, reproduce, or avoid death. The following is a list of evolutionary algorithms closely related to and used in alife:
    - Ant colony optimization
    - Evolutionary algorithm
    - Genetic algorithm
    - Genetic programming
    - Swarm intelligence
4. Evolutionary art uses techniques and methods from artificial life to create new forms of art.
5. Evolutionary music uses similar techniques, but applied to music instead of visual art.

Chapter 7

Techniques of Evolutionary Computation

Cultural Algorithm

Cultural algorithms (CA) are a branch of evolutionary computation where there is a knowledge component that is called the belief space in addition to the population component. In this sense, cultural algorithms can be seen as an extension to a conventional genetic algorithm. Cultural algorithms were introduced by Reynolds.



Belief space

The belief space of a cultural algorithm is divided into distinct categories. These categories represent different domains of knowledge that the population has of the search space.

The belief space is updated after each iteration by the best individuals of the population. The best individuals can be selected using a fitness function that assesses the performance of each individual in population much like in genetic algorithms.

List of belief space categories

- **Normative knowledge** A collection of desirable value ranges for the individuals in the population component eg. acceptable behavior for the agents in population.
- **Domain specific knowledge** Information about the domain of the problem CA is applied to.
- **Situational knowledge** Specific examples of important events - eg. successful/unsuccessful solutions
- **Temporal knowledge** History of the search space - eg. the temporal patterns of the search process
- **Spatial knowledge** Information about the topography of the search space

Population

The population component of the cultural algorithm is approximately the same as that of the genetic algorithm.

Communication protocol

Cultural algorithms require an interface between the population and belief space. The best individuals of the population can update the belief space via the update function. Also, the knowledge categories of the belief space can affect the population component via the influence function. The influence function can affect population by altering the genome or the actions of the individuals.

Pseudo-code for cultural algorithms

1. Initialize **population space** (choose initial population)
2. Initialize **belief space** (eg. set domain specific knowledge and normative value-ranges)
3. Repeat until termination condition is met
    1. Perform actions of the individuals in **population space**
    2. Evaluate each individual by using the **fitness function**
    3. Select the parents to reproduce a new generation of offspring

4. Let the belief space alter the genome of the offspring by using the **influence function**
5. Update the belief space by using the **accept function** (this is done by letting the best individuals to affect the belief space)

Applications

Social simulation

Social simulation is a research field that applies computational methods to study issues in the social sciences. The issues explored include problems in sociology, political science, economics, anthropology, geography, archaeology and linguistics (Takahashi, Sallach & Rouchier 2007).

Social simulation aims to cross the gap between the descriptive approach used in the social sciences and the formal approach used in the hard sciences, by moving the focus on the processes/mechanisms/behaviors that build the social reality.

In social simulation, computers supports human reasoning activities by executing these mechanisms. This field explores the simulation of societies as complex non-linear systems, which are difficult to study with classical mathematical equation-based models. Robert Axelrod regards social simulation as a third way of doing science, differing from both the deductive and inductive approach; generating data that can be analysed inductively, but coming from a rigorously specified set of rules rather than from direct measurement of the real world. Thus, simulating a phenomenon is akin to generating it - constructing artificial societies. These ambitious aims have encountered several criticisms.

The social simulation approach to the social sciences is promoted and coordinated by three regional associations, ESSA for Europe, North America (reorganizing under the new CSSS name), and PAAA Pacific Asia.

History and development

The history of the agent based model can be traced back to the Von Neumann machine, a theoretical machine capable of reproduction. The device von Neumann proposed would follow precisely detailed instructions to fashion a copy of itself. The concept was then improved by von Neumann's friend Stanislaw Ulam, also a mathematician; Ulam suggested that the machine be built on paper, as a collection of cells on a grid. The idea intrigued von Neumann, who drew it up—creating the first of the devices later termed cellular automata.

Another improvement was brought by mathematician, John Conway. He constructed the well-known Game of Life. Unlike the von Neumann's machine, Conway's Game of Life operated by simple rules in a virtual world in the form of a 2-dimensional checkerboard.

The birth of the agent based model as a model for social systems was primarily brought about by a computer scientist, Craig Reynolds. He tried to model the reality of lively biological agents, known as the artificial life, a term coined by Christopher Langton.

Joshua M. Epstein and Robert Axtell developed the first large scale agent model, the Sugarscape, to simulate and explore the role of social phenomena such as seasonal migrations, pollution, sexual reproduction, combat, transmission of disease, and even culture.

Nigel Gilbert (with Klaus G. Troitzsch) published the first textbook on Social Simulation: Simulation for the Social Scientist (1999) and established its most relevant journal: the Journal of Artificial Societies and Social Simulation.

More recently, Ron Sun developed methods for basing agent based simulation on models of human cognition, known as cognitive social simulation.

Types of Simulation and Modeling

Social simulation can refer to a general class of strategies for understanding social dynamics using computers to simulate social systems. Social simulation allows for a more systematic way of viewing the possibilities of outcomes.

There are four major types of social simulation:

1. System level simulation.
2. System level modeling.
3. Agent-based simulation.
4. Agent-based modeling.

A social simulation may fall within the rubric of computational sociology which is a recently developed branch of sociology that uses computation to analyze social phenomena. The basic premise of computational sociology is to take advantage of computer simulations (Polhill & Edmonds 2007) in the construction of social theories. It involves the understanding of social agents, the interaction among these agents, and the effect of these interactions on the social aggregate. Although the subject matter and methodologies in social science differ from those in natural science or computer science, several of the approaches used in contemporary social simulation originated from fields such as physics and artificial intelligence.

System Level Simulation

System Level Simulation (SLS) is the oldest level of social simulation. System level simulation looks at the situation as a whole. This theoretical outlook on social situations uses a wide range of information to determine what should happen to society and its members if certain variables are present. Therefore, with specific variables presented, society and its members should have a certain response to the new situation. Navigating through this theoretical simulation will allow researchers to develop educated ideas of what will happen under some specific variables.

For example if NASA were to conduct a system level simulation it would benefit the organization by providing a cost effective research method to navigate through the simulation. This allows the researcher to steer through the virtual possibilities of the given simulation and develop safety procedures, and to produce proven facts about how a certain situation will play out. (National Research 2006)

System Level Modeling

System level modeling (SLM) aims to specifically predict (unlike system level simulation's generalization in prediction) and convey any number of actions, behaviors, or other theoretical possibilities of nearly any person, object, construct et cetera within a system using a large set of mathematical equations and computer programming in the form of models.

A model is a representation of a specific thing ranging from objects and people to structures and products created through mathematical equations and are designed, using computers, in such a way that they are able to stand-in as the aforementioned things in a study. Models can be either simplistic or complex, depending on the need for either; however, models are intended to be simpler than what they are representing while remaining realistically similar in order to be used accurately. They are built using a collection of data that is translated into computing languages that allow them to represent the system in question. These models, much like simulations, are used to help us better understand specific roles and actions of different things so as to predict behavior and the like.

Agent Based Simulation

Agent-based social simulation (ABSS) consists of modeling different societies after artificial agents, (varying on scale) and placing them in a computer simulated society to observe the behaviors of the agents. From this data it is possible to learn about the reactions of the artificial agents and translate them into the results of non-artificial agents and simulations. Three main fields in ABSS are agent based computing, social science, and computer simulation.

Agent based computing is the design of the model and agents, while the computer simulation is the part of the simulation of the agents in the model and the outcomes. The

social science is a mixture of sciences and social part of the model. It is where the social phenomena is developed and theorized. The main purpose of ABSS is to provide models and tools for agent based simulation of social phenomena. With ABSS we can explore different outcomes for phenomena where we might not be able to view the outcome in real life. It can provide us valuable information on society and the outcomes of social events or phenomenas.

Agent-Based Modeling

Agent-based modeling (ABM) is a system in which a collection of agents independently interact on networks. Each individual agent is responsible for different behaviors that result in collective behaviors. These behaviors as a whole help to define the workings of the network. ABM focuses on human social interactions and how people work together and communicate with one another without having one, single "group mind". This essentially means that it tends to focus on the consequences of interactions between people (the agents) in a population. Researchers are better able to understand this type of modeling by modeling these dynamics on a smaller, more localized level. Essentially, ABM helps to better understand interactions between people (agents) who, in turn, influence one another (in response to these influences). Simple individual rules or actions can result in coherent group behavior. Changes in these individual acts can effect the collective group in any given population.

Agent-based modeling is an experimental tool for theoretical research. It enables one to deal with more complex individual behaviors, such as adaptation. Overall, through this type of modeling, the creator, or researcher, aims to model behavior of agents and the communication between them in order to better understand how these individual interactions impact an entire population. In essence, ABM is a way of modeling and understanding different global patterns.

Current Research

There are several current research projects that relate directly to modeling and agent-based simulation the following are listed below with a brief overview.

- “Generative e-Social Science for Socio-Spatial Simulation” or (GENESIS) is a research node of the UK National Centre for e-Social Science funded by the UK research council ESRC.

- “National e-Infrastructure for Social Simulation” or (NeISS) is a UK based project funded by JISC.

- “Network Models Governance and R&D collaboration networks” or (N.E.M.O) is a research centre whose main focus is to identify ways to create and to assess desirable network structures for typical functions; (e.g. knowledge, creation, transfer, and distribution.) This research will ultimately aid policy-makers at all

political levels in improving the effectiveness and efficiency of network-based policy instruments at promoting the knowledge economy in Europe.

- "Agent based simulations of Market and Consumer Behavior" is another research group that is funded by the Unilever Corporate Research. The current research that is being conducted is investigating the usefulness of agent based simulations for modeling consumer behavior and to show the potential value and insights it can add to long-established marketing methods.

- "New and Emergent World Models Through Individual, Evolutionary and Social Learning" or (New Ties) is a three year project that will ultimately create a virtual society developed by agent-based simulation. The project will develop a simulated society capable of exploring the environment and developing its own image of this environment and the society through interaction. The goal of the research project is for the simulated society to exhibit individual learning, evolutionary learning and social learning.

- Bruch and Mare's project on neighborhood segregation: The purpose of the study is to figure out the reasoning for neighborhood segregation based on race, and to figure out the tipping point or when people become uncomfortable with the integration levels into their neighborhood, and decide to flee from the neighborhood. They set up a model using flash cards, and put the agents house in the middle and put houses of different races surrounding the agents house. They asked people how comfortable they would feel with different situations, if they were okay with one situation they asked another until the neighborhood was fully integrated. Bruch and Mare's results showed that the tipping point was at 50%. When a neighborhood became 50% minority and 50% white, people of both races began to become uncomfortable and white flight began to rise. The use of agent based modeling showed how useful it can be in the world of sociology, people did not have to answer why they would become uncomfortable just in which situation they were uncomfortable with.

- The MAELIA Program (Multi-Agent Emergent Norms Assessment) is a project dealing with the relationships between the users and managers of a natural resource, in that case water, and the related norms and laws that are to be built within them (conventions) or are imposed to them by other actors (institutions). The purpose of the project is to build a generic multiscale platform which is planned to deal with water conflict -related issues.

Agent based modeling is most useful in providing a bridge between micro and macro levels, which is a large part of what sociology studies. Agent based models are most appropriate for studying processes that lack central coordination, including the emergence of institutions that, once established, impose order from the top down. The models focus on how simple and predictable local interactions generate familiar but highly detailed global patterns, such as emergence of norms and participation of collective action. Michael W. Macy and Robert Willer researched a recent survey of

applications and found that there were two main problems with agent based modeling the self-organization of social structure and the emergence of social order (Macy & Willer 2002). Below is a brief description of each problem Macy and Willer believe there to be;

1. "*Emergent structure*. In these models, agents change location or behavior in response to social influences or selection pressures. Agents may start out undifferentiated and then change location or behavior so as to avoid becoming different or isolated (or in some cases, overcrowded). Rather than producing homogeneity, however, these conformist decisions aggregate to produce global patterns of cultural differentiation, stratification, and homophilic clustering in local networks. Other studies reverse the process, starting with a heterogeneous population and ending in convergence: the coordination, diffusion, and sudden collapse of norms, conventions, innovations, and technological standards."
2. "*Emergent social order*. These studies show how egoistic adaptation can lead to successful collective action without either altruism or global (top down) imposition of control. A key finding across numerous studies is that the viability of trust, cooperation, and collective action depends decisively on the embeddedness of interaction."

These examples simply show the complexity of our environment and that agent based models are designed to explore the minimal conditions, the simplest set of assumptions about human behavior, required for a given social phenomenon to emerge at a higher level of organization.

Criticisms of Social Simulation

Since its creation, computerized social simulation has been the target of some criticism in regard to its practicality and accuracy. Social simulation's simplification of the complex to form models from which we can better understand the latter is sometimes seen as a draw back, as using fairly simple models to simulate real life with computers is not always the best way to predict behavior.

Most of the criticism seems to be aimed at agent-based models and simulation and how they work:

1. Simulations, being man-made from mathematical interfaces, predict human behavior in a far too simple manner in regard to the complexities of humanity and our actions.
2. Simulations cannot enlighten researchers as to how people interact or behave in ways not programmed into their models. For this reason, the scope of simulations are limited in that the researchers must already know what they are going to find (to a degree, for they cannot find anything they themselves did not place in the model) at least vaguely, possibly skewing the results.
3. Due to the complexities of what is being measured, simulations must be analyzed in unbiased ways; however, with the model running on a pre-made set of instructions coded into it by a modeler, biases exist almost universally.

4. It is highly difficult and often impractical to attempt to link the findings from the abstract world the simulation creates and our complex society and all of its variation.

Researchers working in social simulation might respond that the competing theories from the social sciences are far simpler than those achieved through simulation and therefore suffer the aforementioned drawbacks much more strongly. Theories in social science tend to be linear models that are not dynamic, and are generlaly inferred from small laboratory experiments. The behavior of populations of agents under these models is rarely tested or verified against empirical observation.

Process optimization

Process optimization is the discipline of adjusting a process so as to optimize some specified set of parameters without violating some constraint. The most common goals are minimizing cost, maximizing throughput, and/or efficiency. This is one of the major quantitative tools in industrial decision making.

When optimizing a process, the goal is to maximize one or more of the process specifications, while keeping all others within their constraints.

Areas

Fundamentally, there are three parameters that can be adjusted to affect optimal performance. They are:

- Equipment optimization

The first step is to verify that the existing equipment is being used to its fullest advantage by examining operating data to identify equipment bottlenecks.

- Operating procedures

Operating procedures may vary widely from person-to-person or from shift-to-shift. Automation of the plant can help significantly. But automation will be of no help if the operators take control and run the plant in manual.

- Control optimization

In a typical processing plant, such as a chemical plant or oil refinery, there are hundreds or even thousands of control loops. Each control loop is responsible for controlling one part of the process, such as maintaining a temperature, level, or flow.

If the control loop is not properly designed and tuned, the process runs below its optimum. The process will be more expensive to operate, and equipment will wear out prematurely. For each control loop to run optimally, identification of sensor, valve, and tuning problems is important. It has been well documented that over 35% of control loops typically have problems.

The process of continuously monitoring and optimizing the entire plant is sometimes called performance supervision.

Chapter 8

Introduction to Semantic Web

W3C's Semantic Web logo

Semantic Web is a group of methods and technologies to allow machines to understand the meaning - or "semantics" - of information on the World Wide Web. The term was coined by World Wide Web Consortium (W3C) director Tim Berners-Lee. He defines the *Semantic Web* as “a web of data that can be processed directly and indirectly by machines.”

While the term "Semantic Web" is not formally defined it is mainly used to describe the model and technologies proposed by the W3C. These technologies include the Resource Description Framework (RDF), a variety of data interchange formats (e.g. RDF/XML, N3, Turtle, N-Triples), and notations such as RDF Schema (RDFS) and the Web Ontology Language (OWL), all of which are intended to provide a formal description of concepts, terms, and relationships within a given knowledge domain.

The key element is that the application in context will try to determine the meaning of the text or other data and then create connections for the user. The evolution of Semantic Web will specifically make possible scenarios that were not otherwise, such as allowing customers to share and utilize computerized applications simultaneously in order to cross reference the time frame of activities with documentation and/or data. According to the original vision, the availability of machine-readable metadata would enable automated agents and other software to access the Web more intelligently. The agents would be able to perform tasks automatically and locate related information on behalf of the user.

Many of the technologies proposed by the W3C already exist and are used in various projects. The Semantic Web as a global vision, however, has remained largely unrealized and its critics have questioned the feasibility of the approach.

In addition other technologies with similar goals, such as microformats, have evolved, which are not always described as "Semantic Web".

Purpose

The main purpose of the **Semantic Web** is driving the evolution of the current Web by allowing users to use it to its full potential thus allowing users to find,share, and combine information more easily. Humans are capable of using the Web to carry out tasks such as finding the Irish word for "folder," reserving a library book, and searching for a low price for a DVD. However, machines cannot accomplish all of these tasks without human direction, because web pages are designed to be read by people, not machines. The semantic web is a vision of information that can be interpreted by machines, so machines can perform more of the tedious work involved in finding, combining, and acting upon information on the web.

Tim Berners-Lee originally expressed the vision of the semantic web as follows:

I have a dream for the Web [in which computers] become capable of analyzing all the data on the Web – the content, links, and transactions between people and computers. A ‘Semantic Web’, which should make this possible, has yet to emerge, but when it does, the day-to-day mechanisms of trade, bureaucracy and our daily lives will be handled by machines talking to machines. The ‘intelligent agents’ people have touted for ages will finally materialize.

– Tim Berners-Lee, 1999

Semantic Web application areas are experiencing intensified interest due to the rapid growth in the use of the Web, together with the innovation and renovation of information content technologies. The Semantic Web is regarded as an integrator across different content,information applications and systems,it also provides mechanisms for the realisation of Enterprise Information Systems. The rapidity of the growth experienced provides the impetus for researchers to focus on the creation and dissemination of innovative Semantic Web technologies, where the envisaged ’Semantic Web’ is long overdue. Often the terms ’Semantics’, ’metadata’, ’ontologies’ and ’Semantic Web’ are used inconsistently. In particular, these terms are used as everyday terminology by researchers and practitioners, spanning a vast landscape of different fields, technologies, concepts and application areas. Furthermore, there is confusion with regards to the current status of the enabling technologies envisioned to realise the Semantic Web. In a paper presented by Gerber, Barnard and Van der Merwe the Semantic Web landscape is charted and a brief summary of related terms and enabling technologies is presented. The architectural model proposed by Tim Berners-Lee is used as basis to present a status model that reflects current and emerging technologies.

Semantic Publishing

Semantic publishing will greatly benefit from the semantic web. In particular, the semantic web is expected to revolutionize scientific publishing, such as real-time publishing and sharing of experimental data on the Internet. This simple but radical idea is now being explored by W3C HCLS group's Scientific Publishing Task Force.

Semantic Blogging

Semantic blogging, like semantic publishing, will change the way blogs are read. Currently "the process of blogging inherently emphasizes metadata creation more than traditional Web publishing methodologies". Some blog users already tag their entries with topics, allowing for easier migration into a semantic web environment. It is intentionally saved in not only a human-readable format, but also in a machine-readable format as the tags can be linked easily to other blogs containing similar information. When a release of a game or movie occurs, bloggers tend to rate them using their own system. If there were to be a unified system, these blogs could easily become assimilated using similar semantics and give a user a score when searching using a semantic search. RSS feeds are another way that blogs already have machine-readable data that is easily accessible by the semantic web.

Web 3.0

Tim Berners-Lee has described the semantic web as a component of 'Web 3.0'.

The internet community as a whole tends to find the two terms "Semantic Web" and "Web 3.0" to be at least synonymous in concept if not completely interchangeable. The definition continues to vary depending on to whom you speak. The overwhelming consensus is that Web 3.0 is most assuredly the "next big thing" but there only lies speculation as to just what that might be. It will be an improvement in the respect that it will still contain Web 2.0 properties while continuing to add to its ever expanding lexicon and library of applications. There are some who claim that Web 3.0 will be more application based and center its efforts towards more graphically capable environments, "non-browser applications and non-computer based devices...geographic or location-based information retrieval" and even more applicable use and growth of Artificial Intelligence. For example, Conrad Wolfram, has argued that Web 3.0 is where "the computer is generating new information", rather than humans.

Others simply state their belief that Web 3.0 will primarily focus on dramatically improving the functionality and usability of search engines. An important factor that users must continue to keep in mind is that the transition to Web 2.0 from "Web 1.0" took approximately ten years. Given the same time frame, this next transition will not be complete until around the year 2015.

People keep asking what Web 3.0 is. I think maybe when you've got an overlay of scalable vector graphics - everything rippling and folding and looking misty — on Web 2.0 and access to a semantic Web integrated across a huge space of data, you'll have access to an unbelievable data resource..."

– Tim Berners-Lee, 2006

Highly specialized information silos, moderated by a cult of personality, validated by the community, and put into context with the inclusion of meta-data through widgets.

– Steve Spalding, 2007

Relationship to the hypertext web

Limitations of HTML

Many files on a typical computer can be loosely divided into documents and data. Documents like mail messages, reports, and brochures are read by humans. Data, like calendars, addressbooks, playlists, and spreadsheets are presented using an application program which lets them be viewed, searched and combined in many ways.

Currently, the World Wide Web is based mainly on documents written in Hypertext Markup Language (HTML), a markup convention that is used for coding a body of text interspersed with multimedia objects such as images and interactive forms. Metadata tags, for example

```
<meta name="keywords" content="computing, computer studies, computer">
<meta name="description" content="Cheap widgets for sale">
<meta name="author" content="John Doe">
```

provide a method by which computers can categorise the content of web pages.

With HTML and a tool to render it (perhaps web browser software, perhaps another user agent), one can create and present a page that lists items for sale. The HTML of this catalog page can make simple, document-level assertions such as "this document's title is 'Widget Superstore'", but there is no capability within the HTML itself to assert unambiguously that, for example, item number X586172 is an Acme Gizmo with a retail price of €199, or that it is a consumer product. Rather, HTML can only say that the span of text "X586172" is something that should be positioned near "Acme Gizmo" and "€199", etc. There is no way to say "this is a catalog" or even to establish that "Acme Gizmo" is a kind of title or that "€199" is a price. There is also no way to express that these pieces of information are bound together in describing a discrete item, distinct from other items perhaps listed on the page.

Semantic HTML refers to the traditional HTML practice of markup following intention, rather than specifying layout details directly. For example, the use of `<em>` denoting "emphasis" rather than `<i>`, which specifies italics. Layout details are left up to the browser, in combination with Cascading Style Sheets. But this practice falls short of specifying the semantics of objects such as items for sale or prices.

Microformats represent unofficial attempts to extend HTML syntax to create machine-readable semantic markup about objects such as retail stores and items for sale.

Semantic Web solutions

The Semantic Web takes the solution further. It involves publishing in languages specifically designed for data: Resource Description Framework (RDF), Web Ontology Language (OWL), and Extensible Markup Language (XML). HTML describes documents and the links between them. RDF, OWL, and XML, by contrast, can describe arbitrary things such as people, meetings, or airplane parts. Tim Berners-Lee calls the resulting network of Linked Data the Giant Global Graph, in contrast to the HTML-based World Wide Web.

These technologies are combined in order to provide descriptions that supplement or replace the content of Web documents. Thus, content may manifest itself as descriptive data stored in Web-accessible databases, or as markup within documents (particularly, in Extensible HTML (XHTML) interspersed with XML, or, more often, purely in XML, with layout or rendering cues stored separately). The machine-readable descriptions enable content managers to add meaning to the content, i.e., to describe the structure of the knowledge we have about that content. In this way, a machine can process knowledge itself, instead of text, using processes similar to human deductive reasoning and inference, thereby obtaining more meaningful results and helping computers to perform automated information gathering and research.

An example of a tag that would be used in a non-semantic web page:

```
<item>cat</item>
```

Encoding similar information in a semantic web page might look like this:

```
<item rdf:about="http://dbpedia.org/resource/Cat">Cat</item>
```

Skeptical reactions

Practical feasibility

Critics (e.g. Which Semantic Web?) question the basic feasibility of a complete or even partial fulfillment of the semantic web. Cory Doctorow's critique ("metacrap") is from the perspective of human behavior and personal preferences. For example, people lie: they may include spurious metadata into Web pages in an attempt to mislead Semantic Web engines that naively assume the metadata's veracity. This phenomenon was well-known with metatags that fooled the AltaVista ranking algorithm into elevating the ranking of certain Web pages: the Google indexing engine specifically looks for such attempts at manipulation. Peter Gärdenfors and Timo Honkela point out that logic-based semantic web technologies cover only a fraction of the relevant phenomena related to semantics.

Where semantic web technologies have found a greater degree of practical adoption, it has tended to be among core specialized communities and organizations for intra-company projects. The practical constraints toward adoption have appeared less

challenging where domain and scope is more limited than that of the general public and the World-Wide Web.

The potential of an idea in fast progress

The original 2001 Scientific American article by Berners-Lee described an expected evolution of the existing Web to a Semantic Web. A complete evolution as described by Berners-Lee has yet to occur. In 2006, Berners-Lee and colleagues stated that: "This simple idea, however, remains largely unrealized." While the idea is still in the making, it seems to evolve quickly and inspire many. Between 2007-2010 several scholars have already explored first applications and the social potential of the semantic web in the business and health sectors, and for social networking and even for the broader evolution of democracy, specifically, how a society forms its common will in a democratic manner through a semantic web

Censorship and privacy

Enthusiasm about the semantic web could be tempered by concerns regarding censorship and privacy. For instance, text-analyzing techniques can now be easily bypassed by using other words, metaphors for instance, or by using images in place of words. An advanced implementation of the semantic web would make it much easier for governments to control the viewing and creation of online information, as this information would be much easier for an automated content-blocking machine to understand. In addition, the issue has also been raised that, with the use of FOAF files and geo location meta-data, there would be very little anonymity associated with the authorship of articles on things such as a personal blog.

Doubling output formats

Another criticism of the semantic web is that it would be much more time-consuming to create and publish content because there would need to be two formats for one piece of data: one for human viewing and one for machines. However, many web applications in development are addressing this issue by creating a machine-readable format upon the publishing of data or the request of a machine for such data. The development of microformats has been one reaction to this kind of criticism. Another argument in defense of the feasibility of semantic web is the likely falling price of human intelligence tasks in digital labor markets like the Amazon Mechanical Turk.

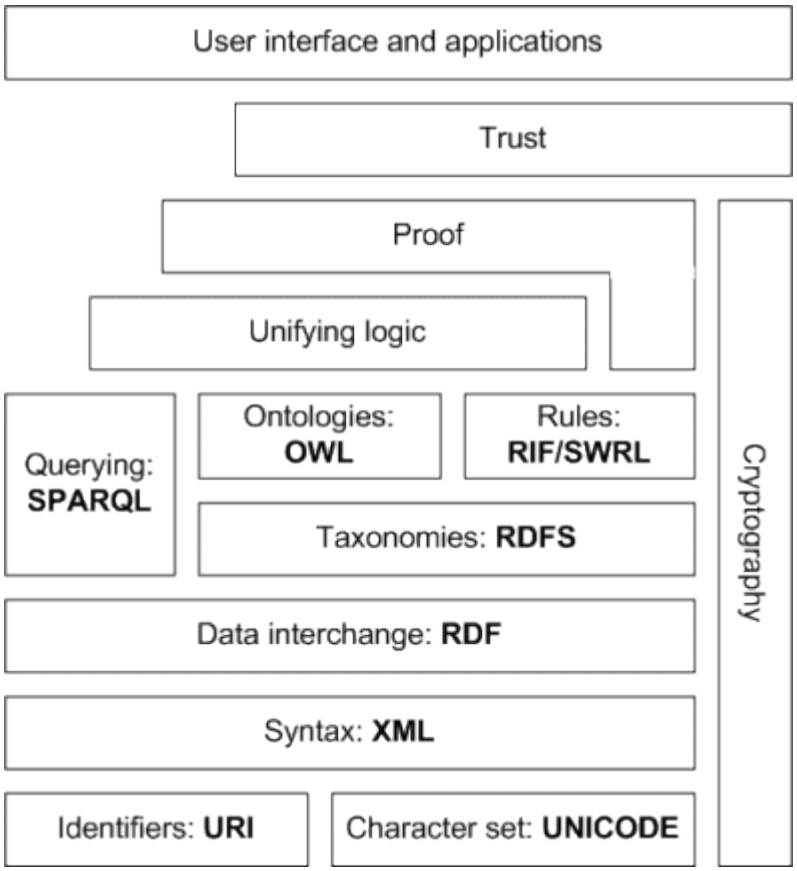
Specifications such as eRDF and RDFa allow arbitrary RDF data to be embedded in HTML pages. The GRDDL (Gleaning Resource Descriptions from Dialects of Language) mechanism allows existing material (including microformats) to be automatically interpreted as RDF, so publishers only need to use a single format, such as HTML.

Need

The idea of a *semantic web*, able to describe and associate meaning with data necessarily involves more than simple XHTML mark-up code. It is based on an assumption that in order for it to be possible to endow machines with an ability to accurately interpret web homed content, far more than the mere ordered relationships involving letters and words, is necessary as underlying infrastructure (attendant to semantic issues). Otherwise, most of the supportive functionality would have been available in Web 2.0 (and before) and it would have been possible to derive a semantically capable Web with minor, incremental additions.

Additions to the infrastructure to support semantic functionality include latent dynamic network models that can, under certain conditions, be 'trained' to appropriately 'learn' meaning based on order data, in the process 'learning' relationships with order (a kind of rudimentary working grammar).

Components



The Semantic Web Stack

The semantic web comprises the standards and tools of XML, XML Schema, RDF, RDF Schema and OWL that are organized in the Semantic Web Stack. The OWL Web Ontology Language Overview describes the function and relationship of each of these components of the semantic web:

- XML provides an elemental syntax for content structure within documents, yet associates no semantics with the meaning of the content contained within.
- XML Schema is a language for providing and restricting the structure and content of elements contained within XML documents.
- RDF is a simple language for expressing data models, which refer to objects ("resources") and their relationships. An RDF-based model can be represented in XML syntax.
- RDF Schema extends RDF and is a vocabulary for describing properties and classes of RDF-based resources, with semantics for generalized-hierarchies of such properties and classes.
- OWL adds more vocabulary for describing properties and classes: among others, relations between classes (e.g. disjointness), cardinality (e.g. "exactly one"), equality, richer typing of properties, characteristics of properties (e.g. symmetry), and enumerated classes.
- SPARQL is a protocol and query language for semantic web data sources.

Current ongoing standardizations include:

- Rule Interchange Format (RIF) as the Rule Layer of the Semantic Web Stack

Not yet fully realized layers include:

- Unifying Logic and Proof layers are undergoing active research.

The intent is to enhance the usability and usefulness of the Web and its interconnected resources through:

- Servers which expose existing data systems using the RDF and SPARQL standards. Many converters to RDF exist from different applications. Relational databases are an important source. The semantic web server attaches to the existing system without affecting its operation.
- Documents "marked up" with semantic information (an extension of the HTML `<meta>` tags used in today's Web pages to supply information for Web search engines using web crawlers). This could be machine-understandable information about the human-understandable content of the document (such as the creator, title, description, etc., of the document) or it could be purely metadata representing a set of facts (such as resources and services elsewhere in the site). (Note that *anything* that can be identified with a *Uniform Resource Identifier* (URI) can be described, so the semantic web can reason about animals, people, places, ideas, etc.) Semantic markup is often generated automatically, rather than manually.

- Common metadata vocabularies (ontologies) and maps between vocabularies that allow document creators to know how to mark up their documents so that agents can use the information in the supplied metadata (so that *Author* in the sense of 'the Author of the page' won't be confused with *Author* in the sense of a book that is the subject of a book review).
- Automated agents to perform tasks for users of the semantic web using this data
- Web-based services (often with agents of their own) to supply information specifically to agents (for example, a Trust service that an agent could ask if some online store has a history of poor service or spamming)

Challenges

Some of the challenges for the Semantic Web include vastness, vagueness, uncertainty, inconsistency, and deceit. Automated reasoning systems will have to deal with all of these issues in order to deliver on the promise of the Semantic Web.

- Vastness: The World Wide Web contains at least 24 billion pages as of this writing (June 13, 2010). The SNOMED CT medical terminology ontology contains 370,000 class names, and existing technology has not yet been able to eliminate all semantically duplicated terms. Any automated reasoning system will have to deal with truly huge inputs.

- Vagueness: These are imprecise concepts like "young" or "tall". This arises from the vagueness of user queries, of concepts represented by content providers, of matching query terms to provider terms and of trying to combine different knowledge bases with overlapping but subtly different concepts. Fuzzy logic is the most common technique for dealing with vagueness.

- Uncertainty: These are precise concepts with uncertain values. For example, a patient might present a set of symptoms which correspond to a number of different distinct diagnoses each with a different probability. Probabilistic reasoning techniques are generally employed to address uncertainty.

- Inconsistency: These are logical contradictions which will inevitably arise during the development of large ontologies, and when ontologies from separate sources are combined. Deductive reasoning fails catastrophically when faced with inconsistency, because "anything follows from a contradiction". Defeasible reasoning and paraconsistent reasoning are two techniques which can be employed to deal with inconsistency.

- Deceit: This is when the producer of the information is intentionally misleading the consumer of the information. Cryptography techniques are currently utilized to alleviate this threat.

This list of challenges is illustrative rather than exhaustive, and it focuses on the challenges to the "unifying logic" and "proof" layers of the Semantic Web. The World

Wide Web Consortium (W3C) Incubator Group for Uncertainty Reasoning for the World Wide Web (URW3-XG) final report lumps these problems together under the single heading of "uncertainty". Many of the techniques mentioned here will require extensions to the Web Ontology Language (OWL) for example to annotate conditional probabilities. This is an area of active research.

Projects

This section lists some of the many projects and tools that exist to create Semantic Web solutions.

FOAF

A popular application of the semantic web is Friend of a Friend (or FoaF), which uses RDF to describe the relationships people have to other people and the "things" around them. FOAF permits intelligent agents to make sense of the thousands of connections people have with each other, their jobs and the items important to their lives; connections that may or may not be enumerated in searches using traditional web search engines. Because the connections are so vast in number, human interpretation of the information may not be the best way of analyzing them.

FOAF is an example of how the Semantic Web attempts to make use of the relationships within a social context.

GoodRelations for e-commerce

A huge potential for Semantic Web technologies lies in adding data structure and typed links to the vast amount of offer data, product model features, and tendering / request for quotation data.

The GoodRelations ontology is a popular vocabulary for expressing product information, prices, payment options, etc. It also allows expressing demand in a straightforward fashion.

GoodRelations has been adopted by Google, BestBuy, Overstock, Yahoo, OpenLink Software, O'Reilly Media, the Book Mashup, and many others.

SIOC

The Semantically-Interlinked Online Communities project (SIOC, pronounced "shock") provides a vocabulary of terms and relationships that model web data spaces. Examples of such data spaces include, among others: discussion forums, blogs, blogrolls / feed subscriptions, mailing lists, shared bookmarks and image galleries.

SIMILE

Semantic **I**nteroperability of **M**etadata and **I**nformation in un**L**ike **E**nvironments

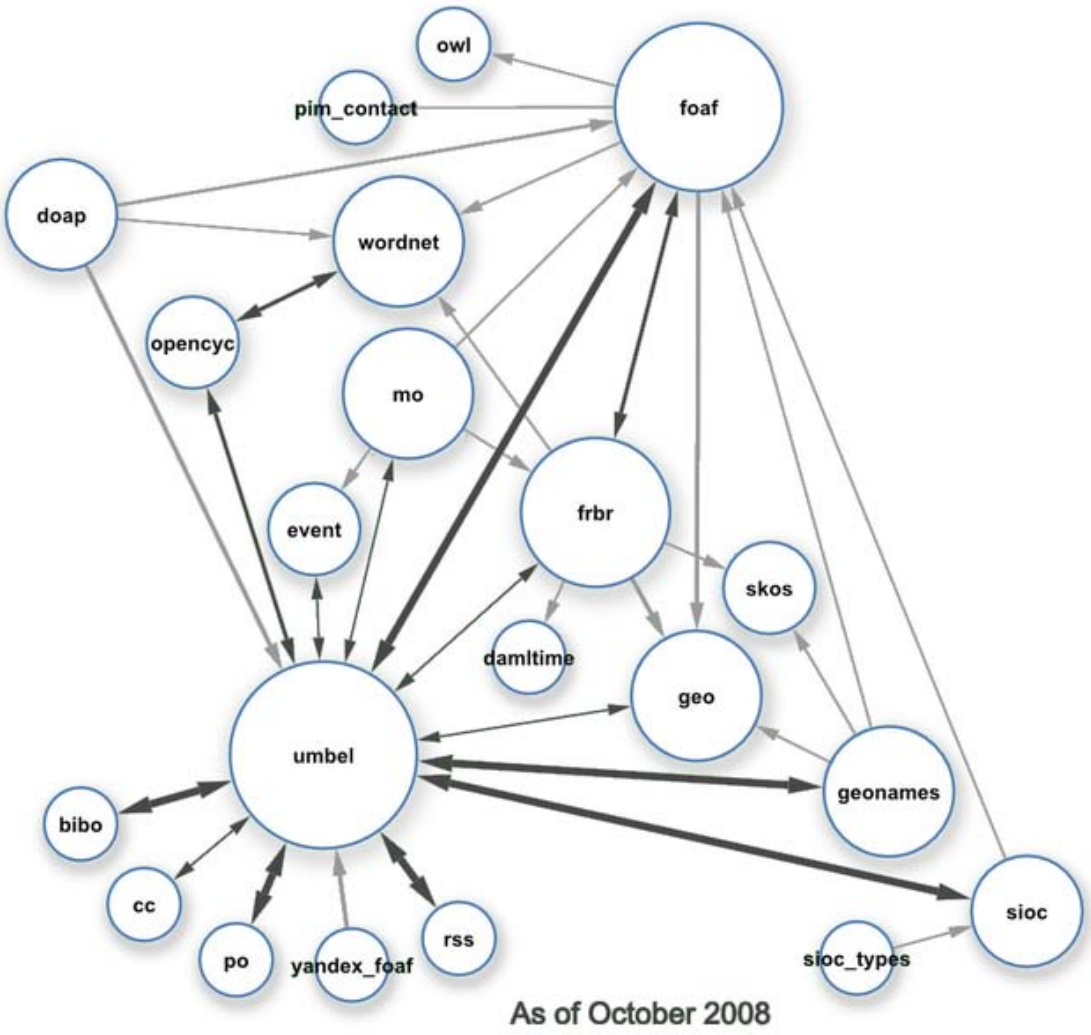
SIMILE is a joint project, conducted by the MIT Libraries and MIT CSAIL, which seeks to enhance interoperability among digital assets, schemata/vocabularies/ontologies, meta data, and services.

NextBio

A database consolidating high-throughput life sciences experimental data tagged and connected via biomedical ontologies. Nextbio is accessible via a search engine interface. Researchers can contribute their findings for incorporation to the database. The database currently supports gene or protein expression data and is steadily expanding to support other biological data types.

Linking Open Data

Datasets in the Linking Open Data project, as of Sept 2008



Class linkages within the Linking Open Data datasets

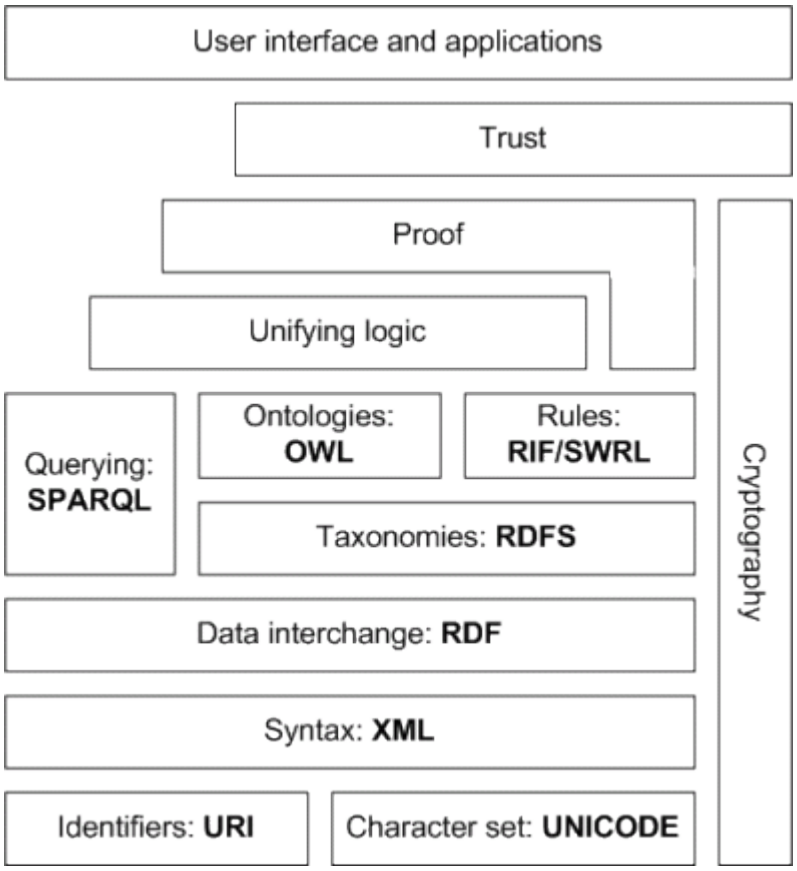
The Linking Open Data project is a W3C-led effort to create openly accessible, and interlinked, RDF Data on the Web. The data in question takes the form of RDF Data Sets drawn from a broad collection of data sources. There is a focus on the Linked Data style of publishing RDF on the Web.

OpenPSI

OpenPSI the (OpenPSI project) is a community effort to create a UK government linked data service that supports research. It is a collaboration between the University of Southampton and the UK government, led by OPSI at The National Archives and is supported by JISC funding.

Chapter 9

Semantic Web Stack



Semantic Web Stack

The *Semantic Web Stack*, also known as *Semantic Web Cake* or *Semantic Web Layer Cake*, illustrates the architecture of the Semantic Web.

Overview

The *Semantic Web Stack* is an illustration of the hierarchy of languages, where each layer exploits and uses capabilities of the layers below. It shows how technologies that are

standardized for Semantic Web are organized to make the Semantic Web possible. It also shows how Semantic Web is an extension (not replacement) of classical hypertext web.

The illustration was created by Tim Berners-Lee. The stack is still evolving as the layers are concretized.

Semantic Web Technologies

As shown in the Semantic Web Stack, the following languages or technologies are used to create Semantic Web. The technologies from the bottom of the stack up to OWL are currently standardized and accepted to build Semantic Web applications. It is still not clear how the top of the stack is going to be implemented. All layers of the stack need to be implemented to achieve full visions of the Semantic Web.

Hypertext Web technologies

The bottom layers contain technologies that are well known from hypertext web and that without change provide basis for the semantic web.

- Internationalized Resource Identifier (IRI), generalization of URI, provides means for uniquely identifying semantic web resources. Semantic Web needs unique identification to allow provable manipulation with resources in the top layers.
- Unicode serves to represent and manipulate text in many languages. Semantic Web should also help to bridge documents in different human languages, so it should be able to represent them.
- XML is a markup language that enables creation of documents composed of structured data. Semantic web gives meaning (semantics) to structured data.
- XML Namespaces provides a way to use markups from more sources. Semantic Web is about connecting data together, and so it is needed to refer more sources in one document.

Standardized Semantic Web technologies

Middle layers contain technologies standardized by W3C to enable building semantic web applications.

- Resource Description Framework (RDF) is a framework for creating statements in a form of so-called triples. It enables to represent information about resources in the form of graph - the semantic web is sometimes called Giant Global Graph.
- RDF Schema (RDFS) provides basic vocabulary for RDF. Using RDFS it is for example possible to create hierarchies of classes and properties.
- Web Ontology Language (OWL) extends RDFS by adding more advanced constructs to describe semantics of RDF statements. It allows stating additional constraints, such as for example cardinality, restrictions of values, or characteristics of properties such as transitivity. It is based on description logic and so brings reasoning power to the semantic web.

- SPARQL is a RDF query language - it can be used to query any RDF-based data (i.e., including statements involving RDFS and OWL). Querying language is necessary to retrieve information for semantic web applications.

Unrealized Semantic Web technologies

Top layers contain technologies that are not yet standardized or contain just ideas that should be implemented in order to realize Semantic Web.

- RIF or SWRL will bring support of rules. This is important for example to allow describing relations that cannot be directly described using description logic used in OWL.
- Cryptography is important to ensure and verify that semantic web statements are coming from trusted source. This can be achieved by appropriate digital signature of RDF statements.
- Trust to derived statements will be supported by (a) verifying that the premises come from trusted source and by (b) relying on formal logic during deriving new information.
- User interface is the final layer that will enable humans to use semantic web applications.

Chapter 10

Resource Description Framework

Resource Description Framework

Current Status	Published, W3C Recommendation
Editors	Frank Manola, Eric Miller
Base Standards	XML, URI
Related Standards	RDFS, OWL, RIF, RDFa
Domain	Semantic Web
Abbreviation	RDF

The **Resource Description Framework (RDF)** is a family of World Wide Web Consortium (W3C) specifications originally designed as a metadata data model. It has come to be used as a general method for conceptual description or modeling of information that is implemented in web resources, using a variety of syntax formats.

Overview

The RDF data model is similar to classic conceptual modeling approaches such as Entity-Relationship or Class diagrams, as it is based upon the idea of making statements about resources (in particular Web resources) in the form of subject-predicate-object expressions. These expressions are known as *triples* in RDF terminology. The subject denotes the resource, and the predicate denotes traits or aspects of the resource and expresses a relationship between the subject and the object. For example, one way to represent the notion "The sky has the color blue" in RDF is as the triple: a subject denoting "the sky", a predicate denoting "has the color", and an object denoting "blue". RDF is an abstract model with several serialization formats (i.e., file formats), and so the particular way in which a resource or triple is encoded varies from format to format.

This mechanism for describing resources is a major component in what is proposed by the W3C's Semantic Web activity: an evolutionary stage of the World Wide Web in which automated software can store, exchange, and use machine-readable information

distributed throughout the Web, in turn enabling users to deal with the information with greater efficiency and certainty. RDF's simple data model and ability to model disparate, abstract concepts has also led to its increasing use in knowledge management applications unrelated to Semantic Web activity.

A collection of RDF statements intrinsically represents a labeled, directed multi-graph. As such, an RDF-based data model is more naturally suited to certain kinds of knowledge representation than the relational model and other ontological models. However, in practice, RDF data is often persisted in relational database or native representations also called Triplestores, or Quad stores if context (i.e. the named graph) is also persisted for each RDF triple. As RDFS and OWL demonstrate, additional ontology languages can be built upon RDF.

History

There were several ancestors to the W3C's RDF. Technically the closest was MCF, a project initiated by Ramanathan V. Guha while at Apple Computer and continued, with contributions from Tim Bray, during his tenure at Netscape Communications Corporation. Ideas from the Dublin Core community, and from PICS, the Platform for Internet Content Selection (the W3C's early Web content labelling system) were also key in shaping the direction of the RDF project.

The W3C published a specification of RDF's data model and XML syntax as a Recommendation in 1999. Work then began on a new version that was published as a set of related specifications in 2004. While there are a few implementations based on the 1999 Recommendation that have yet to be completely updated, adoption of the improved specifications has been rapid since they were developed in full public view, unlike some earlier technologies of the W3C. Most newcomers to RDF are unaware that the older specifications even exist.

In June 2010, W3C organized a workshop to gather feedback from the Web community and discuss possible revisions and improvements to RDF.

RDF Topics

RDF Vocabulary

The vocabulary defined by the RDF specification is:

- rdf:type - a predicate used to state that a resource is an instance of a class
- rdf:XMLLiteral - the class of typed literals
- rdf:Property - the class of properties
- rdf:Alt, rdf:Bag, rdf:Seq - containers of alternatives, unordered containers, and ordered containers (rdfs:Container is a super-class of the three)
- rdf:List - the class of RDF Lists

- rdf:nil - an instance of rdf:List representing the empty list
- rdf:Statement, rdf:subject, rdf:predicate, rdf:object – used for reification (see below)

This vocabulary is used as a foundation for RDF Schema where it is extended.

Serialization formats

Two common serialization formats are in use.

The first is an XML format. This format is often called simply RDF because it was introduced among the other W3C specifications defining RDF. However, it is important to distinguish the XML format from the abstract RDF model itself. Its MIME media type, application/rdf+xml, was registered by RFC 3870. It recommends RDF documents to follow the new 2004 specifications.

In addition to serializing RDF as XML, the W3C introduced Notation 3 (or N3) as a non-XML serialization of RDF models designed to be easier to write by hand, and in some cases easier to follow. Because it is based on a tabular notation, it makes the underlying triples encoded in the documents more easily recognizable compared to the XML serialization. N3 is closely related to the Turtle and N-Triples formats.

Triples may be stored in a triplestore.

Resource identification

The subject of an RDF statement is either a Uniform Resource Identifier (URI) or a blank node, both of which denote resources. Resources indicated by blank nodes are called anonymous resources. They are not directly identifiable from the RDF statement. The predicate is a URI which also indicates a resource, representing a relationship. The object is a URI, blank node or a Unicode string literal.

In Semantic Web applications, and in relatively popular applications of RDF like RSS and FOAF (Friend of a Friend), resources tend to be represented by URIs that intentionally denote, and can be used to access, actual data on the World Wide Web. But RDF, in general, is not limited to the description of Internet-based resources. In fact, the URI that names a resource does not have to be dereferenceable at all. For example, a URI that begins with "http:" and is used as the subject of an RDF statement does not necessarily have to represent a resource that is accessible via HTTP, nor does it need to represent a tangible, network-accessible resource — such a URI could represent absolutely anything. However, there is broad agreement that a bare URI (without a # symbol) which returns a 300-level coded response when used in an HTTP GET request should be treated as denoting the internet resource that it succeeds in accessing.

Therefore, producers and consumers of RDF statements must agree on the semantics of resource identifiers. Such agreement is not inherent to RDF itself, although there are

some controlled vocabularies in common use, such as Dublin Core Metadata, which is partially mapped to a URI space for use in RDF. The intent of publishing RDF-based ontologies on the Web is often to establish, or circumscribe, the intended meanings of the resource identifiers used to express data in RDF. Note that this is not a 'bare' resource identifier, but is rather a URI reference, containing the '#' character and ending with a fragment identifier.

Statement reification and context

The body of knowledge modeled by a collection of statements may be subjected to reification, in which each *statement* (that is each triple *subject-predicate-object* altogether) is assigned a URI and treated as a resource about which additional statements can be made, as in "*Jane says that* John is the author of document X". Reification is sometimes important in order to deduce a level of confidence or degree of usefulness for each statement.

In a reified RDF database, each original statement, being a resource, itself, most likely has at least three additional statements made about it: one to assert that its subject is some resource, one to assert that its predicate is some resource, and one to assert that its object is some resource or literal. More statements about the original statement may also exist, depending on the application's needs.

Borrowing from concepts available in logic (and as illustrated in graphical notations such as conceptual graphs and topic maps), some RDF model implementations acknowledge that it is sometimes useful to group statements according to different criteria, called *situations*, *contexts*, or *scopes*, as discussed in articles by RDF specification co-editor Graham Klyne. For example, a statement can be associated with a context, named by a URI, in order to assert an "is true in" relationship. As another example, it is sometimes convenient to group statements by their source, which can be identified by a URI, such as the URI of a particular RDF/XML document. Then, when updates are made to the source, corresponding statements can be changed in the model, as well.

Implementation of scopes does not necessarily require fully reified statements. Some implementations allow a single scope identifier to be associated with a statement that has not been assigned a URI, itself. Likewise *named graphs* in which a set of triples is named by a URI can represent context without the need to reify the triples.

Query and inference languages

The predominant query language for RDF graphs is SPARQL. SPARQL is an SQL-like language, and a recommendation of the W3C as of January 15, 2008.

An example of a SPARQL query to show country capitals in Africa, using a fictional ontology.

```
PREFIX abc: <nul://sparql/exampleOntology#> .
SELECT ?capital ?country
```

```
WHERE {
  ?x abc:cityname ?capital ;
     abc:isCapitalOf ?y.
  ?y abc:countryname ?country ;
     abc:isInContinent abc:Africa.
}
```

Other ways to query RDF graphs include:

- RDQL, precursor to SPARQL, SQL-like
- Versa, compact syntax (non–SQL-like), solely implemented in 4Suite (Python)
- RQL, one of the first declarative languages for uniformly querying RDF schemas and resource descriptions, implemented in RDFSuite.
- SeRQL, part of Sesame
- XUL has a template element in which to declare rules for matching data in RDF. XUL uses RDF extensively for databinding.

Examples

The postal abbreviation for New York

Certain concepts in RDF are taken from logic and linguistics, where subject-predicate and subject-predicate-object structures have meanings similar to, yet distinct from, the uses of those terms in RDF. This example demonstrates:

In the English language statement *'New York has the postal abbreviation NY'*, *'New York'* would be the subject, *'has the postal abbreviation'* the predicate and *'NY'* the object.

Encoded as an RDF triple, the subject and predicate would have to be resources named by URIs. The object could be a resource or literal element. For example, in the Notation 3 form of RDF, the statement might look like:

```
<urn:x-states:New%20York> <http://purl.org/dc/terms/alternative> "NY" .
```

In this example, "urn:x-states:New%20York" is the URI for a resource that denotes the U.S. state New York, "http://purl.org/dc/terms/alternative" is the URI for a predicate (whose human-readable definition can be found at here), and "NY" is a literal string. Note that the URIs chosen here are not standard, and don't need to be, as long as their meaning is known to whatever is reading them.

N-Triples is just one of several standard serialization formats for RDF. The triple above can also be equivalently represented in the standard RDF/XML format as:

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dcterms="http://purl.org/dc/terms/">
        <rdf:Description rdf:about="urn:x-states:New%20York">
                        <dcterms:alternative>NY</dcterms:alternative>
```

```
        </rdf:Description>
</rdf:RDF>
```

However, because of the restrictions on the syntax of QNames (such as `dcterms:alternative` above), there are some RDF graphs that are not representable with RDF/XML.

Applications

- Sigma - Application from DERI in National University of Ireland, Galway(NUIG).
- Creative Commons - Uses RDF to embed license information in web pages and mp3 files.
- DOAC (Description of a Career) - supplements FOAF to allow the sharing of résumé information.
- FOAF (Friend of a Friend) - designed to describe people, their interests and interconnections.
- Haystack client - Semantic web browser from MIT CS & AI lab.
- IDEAS Group - developing a formal 4D Ontology for Enterprise Architecture using RDF as the encoding.
- Microsoft shipped a product, Connected Services Framework, which provides RDF-based Profile Management capabilities.
- MusicBrainz - Publishes information about Music Albums.
- NEPOMUK, an open-source software specification for a Social Semantic desktop uses RDF as a storage format for collected metadata. NEPOMUK is mostly known because of its integration into the KDE SC 4 desktop environment.
- RDF Site Summary - one of several "RSS" languages for publishing information about updates made to a web page; it is often used for disseminating news article summaries and sharing weblog content.
- ResumeRDF - developed to express information contained in a personal Resume or Curriculum Vitae (CV) on the Semantic Web. This includes information about work and academic experience, skills, etc.
- Simple Knowledge Organization System (SKOS) - a KR representation intended to support vocabulary/thesaurus applications
- SIOC (Semantically-Interlinked Online Communities) - designed to describe online communities and to create connections between Internet-based discussions from message boards, weblogs and mailing lists.
- Smart-M3 - provides an infrastructure for using RDF and specifically uses the ontology agnostic nature of RDF to enable heterogeneous mashing-up of information
- Many other RDF schemas are available by searching SchemaWeb.

Some uses of RDF include research into social networking. This is important because it could help governments keep track of terrorists cells. It will also help people in business fields understand better their relationships with members of industries that could be of

use for product placement. It will also help scientists understand how people are connected to one another.

RDF is being used to have a better understanding of traffic patterns. This is because the information regarding traffic patterns is on different websites, and RDF is used to integrate information from different sources on the web. Before, the common methodology was using keyword searching, but this method is problematic because it does not consider synonyms. This is why ontologies are useful in this situation. But one of the issues that comes up when trying to efficiently study traffic is that to fully understand traffic, concepts related to people, streets, and roads must be well understood. Since these are human concepts, they require the addition of fuzzy logic. This is because values that are useful when describing roads, like slipperiness, are not precise concepts and cannot be measured. This would imply that the best solution would incorporate both fuzzy logic and ontology.

Chapter 11

Web Ontology Language

OWL Web Ontology Language

Current Status	Published
Year Started	2002
Editors	Mike Dean, Guus Schreiber
Base Standards	Resource Description Framework, RDFS
Domain	Semantic Web
Abbreviation	OWL
Website	OWL Reference

OWL 2 Web Ontology Language

Current Status	Published
Year Started	2008
Editors	W3C OWL Working Group
Base Standards	Resource Description Framework, RDFS
Domain	Semantic Web
Abbreviation	OWL 2

The **Web Ontology Language** (**OWL**) is a family of knowledge representation languages for authoring ontologies. The languages are characterised by formal semantics and RDF/XML-based serializations for the Semantic Web. OWL is endorsed by the World Wide Web Consortium (W3C) and has attracted academic, medical and commercial interest.

In October 2007, a new W3C working group was started to extend OWL with several new features as proposed in the OWL 1.1 member submission. This new version, called

OWL 2, soon found its way into semantic editors such as Protégé and semantic reasoners such as Pellet, RacerPro, FaCT++ and HermiT. W3C announced the new version on 27 October 2009.

The OWL family contains many species, serializations, syntaxes and specifications with similar names. This may be confusing unless a consistent approach is adopted. OWL and OWL2 will be used to refer to the 2004 and 2009 specifications, respectively. Full species names will be used, including specification version (for example, OWL2 EL). When referring more generally, *OWL Family* will be used.

History

Early ontology languages

There is a long history of ontological development in philosophy and computer science. Since the 1990s, a number of research efforts have explored how the idea of knowledge representation (KR) from artificial intelligence (AI) could be made useful on the World Wide Web. These included languages based on HTML (called SHOE), based on XML (called XOL, later OIL), and various frame-based KR languages and knowledge acquisition approaches.

Ontology languages for the web

In 2000 in the United States, DARPA started development of DAML led by James Hendler. In March 2001, the *Joint EU/US Committee on Agent Markup Languages* decided that DAML should be merged with OIL. The *EU/US ad hoc Joint Working Group on Agent Markup Languages* was convened to develop DAML+OIL as a web ontology language. This group was jointly funded by the DARPA (under the DAML program) and the European Union's Information Society Technologies (IST) funding project. DAML+OIL was intended to be a thin layer above RDFS, with formal semantics based on a description logic (DL).

OWL started as a research-based revision of DAML+OIL aimed at the semantic web.

Semantic web standards

The Semantic Web provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries.
—World Wide Web Consortium, *W3C Semantic Web Activity*

RDF schema

a declarative representation language influenced by ideas from knowledge representation
—World Wide Web Consortium, *Metadata Activity*

In the late 1990s, the World Wide Web Consortium (W3C) *Metadata Activity* started work on RDF Schema (RDFS), a language for RDF vocabulary sharing. The RDF

became a W3C Recommendation in February 1999, and RDFS a Candidate Recommendation in March 2000. In February 2001, the *Semantic Web Activity* replaced the Metadata Activity. In 2004 (as part of a wider revision of RDF) RDFS became a W3C Recommendation. Though RDFS provides some support for ontology specification, the need for a more expressive ontology language had become clear.

Web-Ontology Working Group

As of Monday, the 31st of May, our working group will officially come to an end. We have achieved all that we were chartered to do, and I believe our work is being quite well appreciated.
—James Hendler and Guus Schreiber, *So Long and thanks for all the fish*

The World Wide Web Consortium (W3C) created the *Web-Ontology Working Group* as part of their Semantic Web Activity. It began work on November 1, 2001 with co-chairs James Hendler and Guus Schreiber. The first working drafts of the abstract syntax, reference and synopsis were published in July 2002. OWL became a formal W3C recommendation on February 10, 2004 and the working group was disbanded on May 31, 2004.

OWL Working Group

In 2005, at the *OWL Experiences And Directions Workshop* a consensus formed that recent advances in description logic would allow a more expressive revision to satisfy user requirements more comprehensively whilst retaining good computational properties. In December 2006, the OWL1.1 Member Submission was made to the W3C. The W3C chartered the *OWL Working Group* as part of the Semantic Web Activity in September 2007. In April 2008, this group decided to call this new language OWL2, indicating a substantial revision.

OWL 2 became a W3C recommendation in October 2009. OWL 2 introduces profiles to improve scalability in typical applications.

Acronym

Why not be inconsistent in at least one aspect of a language which is all about consistency?
—Guus Schreiber, *Why OWL and not WOL?*

The natural acronym for *Web Ontology Language* would be *WOL* instead of *OWL*. Although the character Owl from Winnie the Pooh wrote his name *WOL*, the acronym *OWL* was proposed without reference to that character, as an easily pronounced acronym that would yield good logos, suggest wisdom, and honor William A. Martin's *One World Language* knowledge representation project from the 1970s.

Adoption

A survey (published in 2006) of ontologies available on the web collected 688 OWL ontologies. Of these, 199 were OWL Lite, 149 were OWL DL and 337 OWL Full (by syntax). They found that 19 ontologies had in excess of 2,000 classes, and that 6 had more than 10,000. The same survey collected 587 RDFS vocabularies.

Ontologies

Introduction

An ontology is an explicit specification of a conceptualization.
—Tom Gruber, *A Translation Approach to Portable Ontology Specifications*

The data described by an ontology in the OWL family is interpreted as a set of "individuals" and a set of "property assertions" which relate these individuals to each other. An ontology consists of a set of axioms which place constraints on sets of individuals (called "classes") and the types of relationships permitted between them. These axioms provide semantics by allowing systems to infer additional information based on the data explicitly provided. A full introduction to the expressive power of the OWL is provided in the W3C's *OWL Guide*.

Example

An ontology describing families might include axioms stating that a "hasMother" property is only present between two individuals when "hasParent" is also present, and individuals of class "HasTypeOBlood" are never related via "hasParent" to members of the "HasTypeABBlood" class. If it is stated that the individual Harriet is related via "hasMother" to the individual Sue, and that Harriet is a member of the "HasTypeOBlood" class, then it can be inferred that Sue is not a member of "HasTypeABBlood".

Species

OWL sublanguages

The W3C-endorsed OWL specification includes the definition of three variants of OWL, with different levels of expressiveness. These are OWL Lite, OWL DL and OWL Full (ordered by increasing expressiveness). Each of these sublanguages is a syntactic extension of its simpler predecessor. The following set of relations hold. Their inverses do not.

- Every legal OWL Lite ontology is a legal OWL DL ontology.
- Every legal OWL DL ontology is a legal OWL Full ontology.
- Every valid OWL Lite conclusion is a valid OWL DL conclusion.
- Every valid OWL DL conclusion is a valid OWL Full conclusion.

OWL Lite

OWL Lite was originally intended to support those users primarily needing a classification hierarchy and simple constraints. For example, while it supports cardinality constraints, it only permits cardinality values of 0 or 1. It was hoped that it would be simpler to provide tool support for OWL Lite than its more expressive relatives, allowing quick migration path for systems utilizing thesauri and other taxonomies. In practice, however, most of the expressiveness constraints placed on OWL Lite amount to little more than syntactic inconveniences: most of the constructs available in OWL DL can be built using complex combinations of OWL Lite features. Development of OWL Lite tools has thus proven almost as difficult as development of tools for OWL DL, and OWL Lite is not widely used.

OWL DL

OWL DL was designed to provide the maximum expressiveness possible while retaining computational completeness (either φ or $\neg\varphi$ belong), decidability (there is an effective procedure to determine whether φ is derivable or not), and the availability of practical reasoning algorithms. OWL DL includes all OWL language constructs, but they can be used only under certain restrictions (for example, number restrictions may not be placed upon properties which are declared to be transitive). OWL DL is so named due to its correspondence with description logic, a field of research that has studied the logics that form the formal foundation of OWL.

OWL Full

OWL Full is based on a different semantics from OWL Lite or OWL DL, and was designed to preserve some compatibility with RDF Schema. For example, in OWL Full a class can be treated simultaneously as a collection of individuals and as an individual in its own right; this is not permitted in OWL DL. OWL Full allows an ontology to augment the meaning of the pre-defined (RDF or OWL) vocabulary. It is unlikely that any reasoning software will be able to support complete reasoning for OWL Full.

OWL2 profiles

In OWL 2, there are three sublanguages of the language. OWL 2 EL is a fragment that has polynomial time reasoning complexity; OWL 2 QL is designed to enable easier access and query to data stored in databases; OWL 2 RL is a rule subset of OWL 2.

Syntax

The OWL family of languages support a variety of syntaxes. It is useful to distinguish *high level* syntaxes aimed at specification from *exchange* syntaxes more suitable for general use.

High level

These are close to the ontology structure of languages in the OWL family.

OWL abstract syntax

This high level syntax is used to specify the OWL ontology structure and semantics.

The OWL abstract syntax presents an ontology as a sequence of *annotations*, *axioms* and *facts*. Annotations carry machine and human oriented meta-data. Information about the classes, properties and individuals that compose the ontology is contained in axioms and facts only. Each class, property and individual is either *anonymous* or identified by an URI reference. Facts state data either about an individual or about a pair of individual identifiers (that the objects identified are distinct or the same). Axioms specify the characteristics of classes and properties. This style is similar to frame languages, and quite dissimilar to well known syntaxes for description logics (DLs) and Resource Description Framework (RDF).

Sean Bechhofer, et. al. argue that though this syntax is hard to parse, it is quite concrete. They conclude that the name *abstract syntax* may be somewhat misleading.

OWL2 functional syntax

This syntax closely follows the structure of an OWL2 ontology. It is used by OWL2 to specify semantics, mappings to exchange syntaxes and profiles.

Exchange syntaxes

RDF syntaxes

Syntactic mappings into RDF are specified for languages in the OWL family. Several RDF serialization formats have been devised. Each leads to a syntax for languages in the OWL family through this mapping. RDF/XML is normative.

OWL2 XML syntax

OWL2 specifies an XML serialization that closely models the structure of an OWL2 ontology.

Manchester Syntax

The Manchester Syntax is a compact, human readable syntax with a style close to frame languages. Variations are available for OWL and OWL2. Not all OWL and OWL2 ontologies can be expressed in this syntax.

Examples

- The W3C OWL 2 Web Ontology Language provides syntax examples.

Tea ontology

Consider an ontology for tea based on a Tea class. But first, an ontology is needed. Every OWL ontology must be identified by an URI. This is enough to get a flavour of the syntax. To save space below, preambles and prefix definitions have been skipped.

OWL2 Functional Syntax

```
Ontology(<http://example.com/tea.owl>
  Declaration( Class( :Tea ) )
)
```

OWL2 XML Syntax

```
 <Ontology ontologyIRI="http://example.com/tea.owl" ...>
   <Prefix name="owl" IRI="http://www.w3.org/2002/07/owl#"/>
   <Declaration>
     <Class IRI="Tea"/>
   </Declaration>
 </Ontology>
```

Manchester Syntax

```
Ontology: <http://example.com/tea.owl>
Class: Tea
```

RDF/XML syntax

```
<rdf:RDF ...>
    <owl:Ontology rdf:about=""/>
    <owl:Class rdf:about="#Tea"/>
</rdf:RDF>
```

RDF/Turtle

```
 <http://example.com/tea.owl> rdf:type owl:Ontology .
 :Tea  rdf:type            owl:Class .
```

Semantics

Relation to description logic

In the beginning, IS-A was quite simple. Today, however, there are almost as many meanings for this inheritance link as there are knowledge-representation systems.
—Ronald J. Brachman, *What ISA is and isn't*

Early attempts to build large ontologies were plagued by a lack of clear definitions. Members of the OWL family have model theoretic formal semantics, and so have strong logical foundations.

Description logics (DLs) are a family of logics that are decidable fragments of first-order logic with attractive and well-understood computational properties. OWL DL and OWL Lite semantics are based on DLs. They combine a syntax for describing and exchanging ontologies, and formal semantics that gives them meaning. For example, OWL DL corresponds to the SHOIN (D) description logic, while OWL 2 corresponds to the SROIQ(D) logic. Sound, complete, terminating reasoners (i.e. systems which are guaranteed to derive every consequence of the knowledge in an ontology) exist for these DLs.

Relation To RDFS

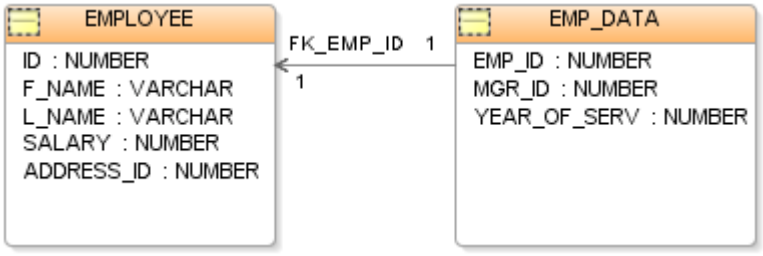
OWL Full is intended to be compatible with RDF Schema (RDFS), and to be capable of augmenting the meanings of existing Resource Description Framework (RDF) vocabulary. A model theory describes the formal semantics for RDF. This interpretation provides the meaning of RDF and RDFS vocabulary. So, the meaning of OWL Full ontologies are defined by extension of the RDFS meaning, and OWL Full is a semantic extension of RDF.

Open world assumption

the closed world assumption implies that everything we don't know is false, while the open world assumption states that everything we don't know is undefined
—Stefano Mazzocchi, *Closed World vs. Open World: the First Semantic Web Battle*

The languages in the OWL family use the open world assumption. Under this open world assumption, if a statement cannot be proved to be true using current knowledge, we cannot draw the conclusion that the statement is false.

Relational database



A visual diagram showing the relationship between the two tables, as indicated by the arrow

A **relational database** matches data by using common characteristics found within the data set. The resulting groups of data are organized and are much easier for many people to understand.

For example, a data set containing all the real-estate transactions in a town can be grouped by the year the transaction occurred; or it can be grouped by the sale price of the transaction; or it can be grouped by the buyer's last name; and so on.

Such a grouping uses the relational model (a technical term for this is schema). Hence, such a database is called a "relational database."

The software used to do this grouping is called a relational database management system (RDBMS). The term "relational database" often refers to this type of software.

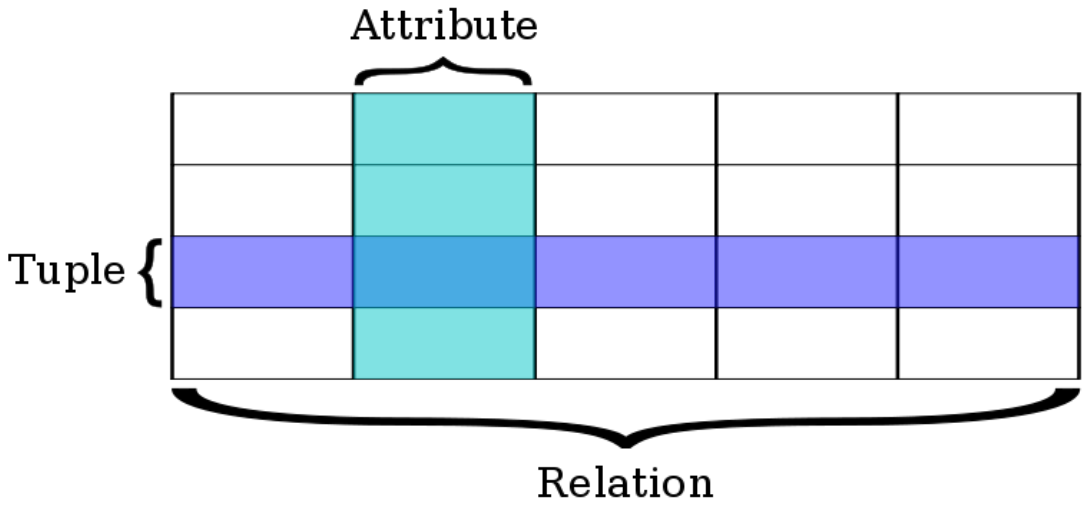
Relational databases are currently the predominant choice in storing financial records, medical records, manufacturing and logistical information, personnel data and much more.

Contents

Strictly, a relational database is a collection of relations (frequently called tables). Other items are frequently considered part of the database, as they help to organize and structure the data, in addition to forcing the database to conform to a set of requirements.

Terminology

The term *relational database* was originally defined and coined by Edgar Codd at IBM Almaden Research Center in 1970.



Relational database terminology

Relational database theory uses a set of mathematical terms, which are roughly equivalent to SQL database terminology. The table below summarizes some of the most important relational database terms and their SQL database equivalents.

Relational term	SQL equivalent
relation, base relvar	table
derived relvar	view, query result, result set
tuple	row
attribute	column

Relations or Tables

A *relation* is defined as a set of tuples that have the same attributes. A tuple usually represents an object and information about that object. Objects are typically physical objects or concepts. A relation is usually described as a table, which is organized into rows and columns. All the data referenced by an attribute are in the same domain and conform to the same constraints.

The relational model specifies that the tuples of a relation have no specific order and that the tuples, in turn, impose no order on the attributes. Applications access data by specifying queries, which use operations such as *select* to identify tuples, *project* to identify attributes, and *join* to combine relations. Relations can be modified using the *insert*, *delete*, and *update* operators. New tuples can supply explicit values or be derived from a query. Similarly, queries identify tuples for updating or deleting. It is necessary for each tuple of a relation to be uniquely identifiable by some combination (one or more) of its attribute values. This combination is referred to as the primary key.

Base and derived relations

In a relational database, all data are stored and accessed via relations. Relations that store data are called "base relations", and in implementations are called "tables". Other relations do not store data, but are computed by applying relational operations to other relations. These relations are sometimes called "derived relations". In implementations these are called "views" or "queries". Derived relations are convenient in that though they may grab information from several relations, they act as a single relation. Also, derived relations can be used as an abstraction layer.

Domain

A domain describes the set of possible values for a given attribute, and can be considered a constraint on the value of the attribute. Mathematically, attaching a domain to an attribute means that any value for the attribute must be an element of the specified set.

The character data value 'ABC', for instance, is not in the integer domain. The integer value 123, satisfies the domain constraint.

Constraints

Constraints make it possible to further restrict the domain of an attribute. For instance, a constraint can restrict a given integer attribute to values between 1 and 10. Constraints provide one method of implementing business rules in the database. SQL implements constraint functionality in the form of check constraints.

Constraints restrict the data that can be stored in relations. These are usually defined using expressions that result in a boolean value, indicating whether or not the data satisfies the constraint. Constraints can apply to single attributes, to a tuple (restricting combinations of attributes) or to an entire relation.

Since every attribute has an associated domain, there are constraints (**domain constraints**). The two principal rules for the relational model are known as **entity integrity** and **referential integrity**.

Primary keys

A primary key uniquely defines a relationship within a database. In order for an attribute to be a good primary key it must not repeat. While natural attributes are sometimes good primary keys, Surrogate keys are often used instead. A surrogate key is an artificial attribute assigned to an object which uniquely identifies it (for instance, in a table of information about students at a school they might all be assigned a Student ID in order to differentiate them). The surrogate key has no intrinsic (inherent) meaning, but rather is useful through its ability to uniquely identify a tuple.

Another common occurrence, especially in regards to N:M cardinality is the composite key. A composite key is a key made up of two or more attributes within a table that (together) uniquely identify a record. (For example, in a database relating students, teachers, and classes. Classes *could* be uniquely identified by a composite key of their room number and time slot, since no other class could have exactly the same combination of attributes. In fact, use of a composite key such as this can be a form of data verification, albeit a weak one.)

Foreign keys

A foreign key is a reference to a key in another relation, meaning that the referencing table has, as one of its attributes, the values of a key in the referenced table. Foreign keys need not have unique values in the referencing relation. Foreign keys effectively use the values of attributes in the referenced relation to restrict the domain of one or more attributes in the referencing relation.

A foreign key could be described formally as: "For all tables in the referencing relation projected over the referencing attributes, there must exist a table in the referenced relation projected over those same attributes such that the values in each of the referencing attributes match the corresponding values in the referenced attributes."

Stored procedures

A stored procedure is executable code that is associated with, and generally stored in, the database. Stored procedures usually collect and customize common operations, like inserting a tuple into a relation, gathering statistical information about usage patterns, or encapsulating complex business logic and calculations. Frequently they are used as an application programming interface (API) for security or simplicity. Implementations of stored procedures on SQL DBMSs often allow developers to take advantage of procedural extensions (often vendor-specific) to the standard declarative SQL syntax.

Stored procedures are not part of the relational database model, but all commercial implementations include them.

Indices

An index is one way of providing quicker access to data. Indices can be created on any combination of attributes on a relation. Queries that filter using those attributes can find matching tuples randomly using the index, without having to check each tuple in turn. This is analogous to using the index of a book to go directly to the page on which the information you are looking for is found i.e. you do not have to read the entire book to find what you are looking for. Relational databases typically supply multiple indexing techniques, each of which is optimal for some combination of data distribution, relation size, and typical access pattern. B+ trees, R-trees, and bitmaps.

Indices are usually not considered part of the database, as they are considered an implementation detail, though indices are usually maintained by the same group that maintains the other parts of the database.

Relational operations

Queries made against the relational database, and the derived relvars in the database are expressed in a relational calculus or a relational algebra. In his original relational algebra, Codd introduced eight relational operators in two groups of four operators each. The first four operators were based on the traditional mathematical set operations:

- The union operator combines the tuples of two relations and removes all duplicate tuples from the result. The relational union operator is equivalent to the SQL UNION operator.
- The intersection operator produces the set of tuples that two relations share in common. Intersection is implemented in SQL in the form of the INTERSECT operator.
- The difference operator acts on two relations and produces the set of tuples from the first relation that do not exist in the second relation. Difference is implemented in SQL in the form of the EXCEPT or MINUS operator.
- The cartesian product of two relations is a join that is not restricted by any criteria, resulting in every tuple of the first relation being matched with every

tuple of the second relation. The cartesian product is implemented in SQL as the CROSS JOIN join operator.

The remaining operators proposed by Codd involve special operations specific to relational databases:

- The selection, or restriction, operation retrieves tuples from a relation, limiting the results to only those that meet a specific criteria, i.e. a subset in terms of set theory. The SQL equivalent of selection is the SELECT query statement with a WHERE clause.
- The projection operation extracts only the specified attributes from a tuple or set of tuples.
- The join operation defined for relational databases is often referred to as a natural join. In this type of join, two relations are connected by their common attributes. SQL's approximation of a natural join is the INNER JOIN join operator.
- The relational division operation is a slightly more complex operation, which involves essentially using the tuples of one relation (the dividend) to partition a second relation (the divisor). The relational division operator is effectively the opposite of the cartesian product operator (hence the name).

Other operators have been introduced or proposed since Codd's introduction of the original eight including relational comparison operators and extensions that offer support for nesting and hierarchical data, among others.

Normalization

Normalization was first proposed by Codd as an integral part of the relational model. It encompasses a set of best practices designed to eliminate the duplication of data, which in turn prevents data manipulation anomalies and loss of data integrity. The most common forms of normalization applied to databases are called the normal forms. Normalization trades reducing redundancy for increased information entropy. Normalization is criticised because it increases complexity and processing overhead required to join multiple tables representing what are conceptually a single item.

Relational database management systems

Relational databases, as implemented in relational database management systems, have become a predominant choice for the storage of information in new databases used for financial records, manufacturing and logistical information, personnel data and much more. Relational databases have often replaced legacy hierarchical databases and network databases because they are easier to understand and use, even though they are much less efficient. As computer power has increased, the inefficiencies of relational databases, which made them impractical in earlier times, have been outweighed by their ease of use. However, relational databases have been challenged by Object Databases, which were

introduced in an attempt to address the object-relational impedance mismatch in relational database, and XML databases.

The three leading commercial relational database vendors are Oracle, Microsoft, and IBM.. The three leading open source implementations are MySQL, PostgreSQL, and SQLite.

Terminology

Languages in the OWL family are capable of creating classes, properties, defining instances and its operations.

Instances

An *instance* is an object. It corresponds to a description logic *individual*.

Classes

A *class* is a collection of objects. It corresponds to a description logic (DL) *concept*. A class may contain individuals, *instances* of the class. A class may have any number of instances. An instance may belong to none, one or more classes.

A class may be a *subclass* of another, inheriting characteristics from its parent *superclass*. This corresponds to logical subsumption and DL *concept inclusion* notated $\sqsubseteq$.

All classes are subclasses of owl:Thing (DL *top* notated $\top$), the *root* class.

All classes are subclassed by owl:Nothing (DL *bottom* notated $\bot$), the *empty* class. No instances are members of owl:Nothing. Modelers use owl:Thing and owl:Nothing to assert facts about all or no instances.

Example

For example, Employee could be the subclass of class owl:Thing while Dealer, Manager, and Labourer all subclass of Employee.

Properties

A property is a directed binary relation that specifies class characteristics. It corresponds to a description logic *role*. They are attributes of instances and sometimes act as data values or link to other instances. Properties may possess logical capabilities such as being transitive, symmetric, inverse and functional. Properties may also have domains and ranges.

Datatype properties

Datatype properties are relations between instances of classes and RDF literals or XML schema datatypes. For example, modelName (String datatype) is the property of Manufacturer class. They are formulated using *owl:DatatypeProperty* type.

Object properties

Object properties are relations between instances of two classes. For example, ownedBy may be an object type property of the Vehicle class and may have a range which is the class Person. They are formulated using *owl:ObjectProperty*.

Operators

Languages in the OWL family support various operations on classes such as union, intersection and complement. They also allow class enumeration, cardinality, and disjointness.

Public ontologies

Libraries

Biomedical

- OBO Foundry
- NCBO BioPortal
- NCI Enterprise Vocabulary Services

Standards

- Suggested Upper Merged Ontology
- TDWG

Browsers

The following tools include public ontology browsers:

- Protégé OWL

Search

- Swoogle

Limitations

- Relationships are directed
- No direct language support for n-ary relationships. For example modelers may wish to describe the qualities of a relation, to relate more than 2 individuals or to relate an individual to a list. This cannot be done within OWL. They may need to adopt a pattern instead which encodes the meaning outside the formal semantics.