

# Intelligent Decision Support Systems Based on SMILE

Marek J. Druzdziel

## On the CD:

Full documentation, the files including source code using in this article, the SMILE library and the GeNIe graphical interface can be found on the CD.

In this article I will share with the readers the basics of and the principles behind intelligent decision support systems based on the theoretically sound principles of probability theory and decision theory. Recent advances in probability theory have led to the development of graphical models that are capable of modelling the causal structure of systems, well understood by human experts, and at the same time give such structure a sound probabilistic interpretation. Graphical models can serve as a convenient basis for modelling domains that involve a high degree of uncertainty or reasoning with them. Examples of problems that are addressed by systems based on graphical models include computer vision, robotics, pattern matching, medical diagnostics and therapy planning, machine diagnostics, and even on-line help. Microsoft Corporation uses graphical models inside the Windows operating system, in troubleshooting and user interfaces – for example, on-line Office help and junk mail filtering in Outlook. The graphical models methodology is implemented in a general purpose decision modelling system SMILE and its Windows user interface, GeNIe, developed at the Decision Systems Laboratory. I will try to give the readers a flavor of GeNIe models and building systems based on the SMILE library.

## Bayesian networks

A prominent subclass of graphical probabilistic models are Bayesian networks, also called belief networks or, somewhat imprecisely, causal networks. The adjective “Bayesian” refers to the very foundation of decision theory and an increasingly popular branch of probability theory, the subjectivist *Bayesian view*, named after a brilliant English mathematician, Reverend Thomas Bayes. Rev. Bayes has become known for his views on belief revision. The Bayes theorem, accepted and used by all schools of probability theory and statistics, expresses the computation that needs to be performed in order to revise prior beliefs in the light of new evidence:

$$P(A|B) = P(A, B) / P(B) = P(B|A) / P(B) P(A)$$

Marek J. Druzdziel is an associate professor at the School of Information Sciences and Intelligent Systems Program, University of Pittsburgh, USA, and a director of the Decision Systems Laboratory, a research group devoted to building decision support systems that are based on the normative principles of probability theory and decision theory. He has written over 60 articles for leading professional journals and conferences. Contact the author: [marek@sis.pitt.edu](mailto:marek@sis.pitt.edu)

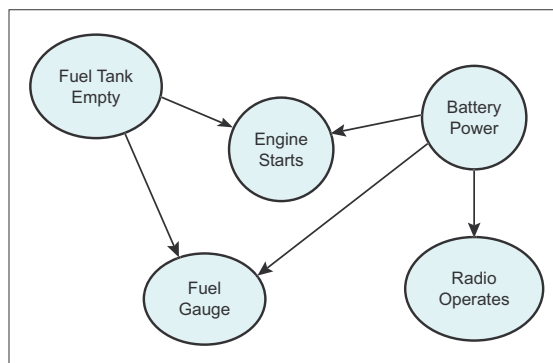


Figure 1. A simple Bayesian network for diagnosing an automobile engine

In the above formula,  $P(A|B)$  denotes the probability of an event A conditional on an event B, i.e., the probability that A is true if B happened. This probability can be expressed by a product of two factors: the likelihood ratio  $P(B|A) / P(B)$  that expresses how informative B is with respect to A and  $P(A)$ , the prior probability of A (i.e. the probability of A before observing B). The brilliance of Rev. Bayes is the interpretation of this theorem, notably that belief in a hypothesis A can be revised in strict agreement with the laws of probability.

A Bayesian network encodes knowledge about a domain by means of a graph whose vertices represent random variables and edges represent probabilistic relations among them. Typically, the structure of the graph mimics the causal structure of the domain. The Bayesian network in Figure 1 models various problems related to starting an automobile car engine.

In order for the engine to start, there must be both sufficient battery power and enough fuel. The fuel gauge is one way of observing whether there is any fuel in the tank. However, the gauge will work only if there is battery power. Battery power can be verified by a working radio. All of these relationships are probabilistic in nature. Even if there is sufficient battery power and there is enough fuel, the engine may fail to start due to other, unmodelled reasons, and the gauge may show a wrong indication. The radio itself may be broken and not work despite sufficient battery charge. These relationships are, in addition to the structure of the graph, modelled by means of numerical probabilities that express the strength of influences between the variables. These probabilities are encoded in tables that are included with every node and are indexed by the parent nodes. The node *Fuel Gauge* may contain a conditional probability table as shown in Figure 2.

Fuel Tank	Full		Empty	
	Loaded	Empty	Loaded	Empty
Battery Power				
▶ Normal	0.95	0.01	0.02	0.01
Empty	0.05	0.99	0.98	0.99

Figure 2. Conditional probability table for the node Fuel Gauge

The topmost rows of the table show all combinations of the states of the parent variables. The rows of the table show the probabilities of various states of the node *Engine Starts* under all combinations of the states of the parents. The columns of the table are effectively encoding the probability distributions of the child variable conditional on the parent variables. And so, if the fuel tank is empty and the battery is loaded, the fuel gauge will show a normal reading with probability 0.95. There is a 0.05 chance that it will malfunction. Even if the battery is empty, there is a 0.01 probability that the gauge reading will be normal (columns 2 and 4).

The probability table for the node *Radio Operates* is shown in Figure 3. Here, the conditional probability distributions encode the fact that if the battery is empty, the radio will certainly not work, but even if the battery is not empty, the radio may fail with probability 0.05.

**Bayesian network inference**

The cool thing about Bayesian networks is that they can be used to compute probability distributions of variables of interest given that states of other variables have been observed. This is precisely the belief revision that Rev. Bayes has proposed. The probability of any event given a set of observations is simply probability of this event conditional on these observations. If we observe that the engine does not start and that the fuel gauge is low, the *posterior* probabilities of an empty fuel tank and an empty battery are both around one third (see Figure 4).

If, however, we make an additional observation that the car radio operates, the network infers that the battery cannot be empty and puts the blame for the engine not starting almost entirely on the possibly empty fuel tank (Figure 5).

Please note that the gauge reading was high and the network concluded that the high reading must have been erroneous. Also, please note that the probability of the fuel tank being full given the observations, is 0.01. If the fuel tank is indeed full, the cause of engine not starting may be neither empty fuel tank nor empty battery.

Inference in Bayesian networks, examples of which are given above, is performed by so called *belief updating* algorithms. The details of these algorithms are beyond the scope of this paper, although at some rough level it is fair to say that they consist of repetitive applications of the Bayes theorem that spread

Battery Power	Loaded	Empty
	▶ Operates	0.95
DoesNotOperate	0.05	1

Figure 3. Conditional probability table for the node Radio Operates

the impact of observed evidence through the network. Even though the complexity of belief-updating algorithms is exponential in the worst case, there exist a number of both exact and approximate algorithms that can handle practical networks consisting of hundreds or even thousands of variables.

**Other computations**

There is much more that a network can do for us. A mechanic performing the diagnostics can ask the question: *Which of the many available tests should I perform next?* Because we have a full specification of the underlying probability distribution, we can compute the value of observing each of the symptoms or tests. A complex medical system for diagnosing certain liver disorders developed in collaboration between University of Pittsburgh, Technical University of Bialystok, and The Medical Center of Postgraduate Education and the Institute of Biocybernetics and Biomedical Engineering, Polish Academy of Sciences is a good example. Analysis has indicated that the test *Antimitochondrial antibodies* seems the most valuable for establishing the posterior probability of *PBC* (Primary Biliary Cirrhosis).

**Learning**

Bayesian networks can learn from data in the domains where data is collected. Banks can successfully learn the joint probability distribution over variables describing their client population and use this information for the purpose of credit rating. A telephone company can perform real-time inference given characteristics of a telephone call to determine the probability that the call is fraudulent and interrupt the call if it falls above a certain threshold. A Bayesian network learned from data generated by an industrial process can subsequently be used to control this process.

**Normative systems**

The example above showed a naively simplified network consisting of only five variables. Bayesian networks can be huge and model, for example, very large engineering devices. It is

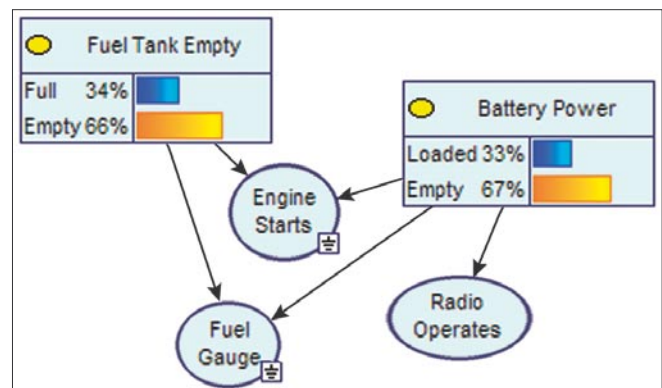
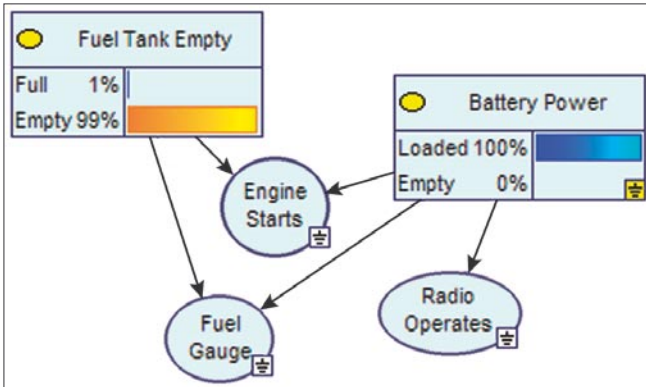


Figure 4. Posterior marginal probability distributions over nodes Fuel Tank Empty and Battery Power given observation that the engine does not start and that the fuel gauge shows low level

**On the Net**

- Genie and SMILE <http://www.sis.pitt.edu/~genie/>
- Marek Druzdel homepage <http://www.pitt.edu/~druzdel/>
- Communications of the ACM <http://www.acm.org/pubs/cacm/>



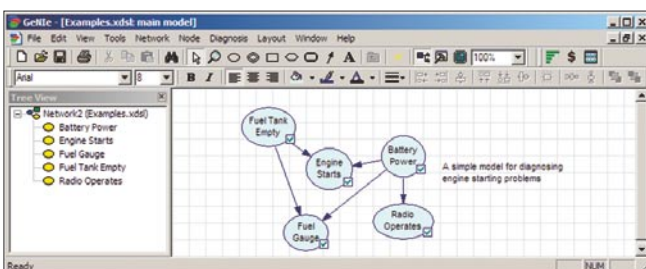
**Figure 5.** Posterior marginal probability distributions over nodes Fuel Tank Empty and Battery Power given observation that the engine does not start, that the fuel gauge does not show low and that the radio operates

not uncommon in machine diagnostics to construct networks consisting of thousands of variables. Reasoning in such networks is the same as in the above simple example: observations spread through the network and lead to deriving the posterior probability distributions of the variables of interest. The reasoning is mathematically correct and can be trusted to rely faithfully on the underlying model. Systems based on sound foundations of probability theory and decision theory are collectively known as *normative* or *prescriptive* decision support systems, as opposed to *descriptive* systems that aim at imitating reasoning of human experts (one example class of descriptive systems are rule-based expert systems).

Normative decision support systems are increasingly applied in various domains because of their sound foundations, ability to combine existing data with expert knowledge, and intuitive framework of directed graphical models, such as Bayesian networks and influence diagrams. Some of the applications of normative systems are: medical diagnostics and therapy planning, machine diagnostics, natural language processing, vision, robotics, planning, fraud detection, processing of military intelligence data in the context of battle damage assessment, and many others (March 1995 issue of *Communications of the ACM* lists several practical applications of Bayesian networks). Given the current interest in the application of decision-theoretic methods and the speed with which they are applied in practice, it can be expected that they will remain core modelling tools in intelligent systems.

## GeNIe and SMILE

The Decision Systems Laboratory at the University of Pittsburgh has been working on an implementation of a general purpose modelling environment implementing this methodology.



**Figure 6.** The GeNIe model-building interface. The left pane shows the tree view and the right pane shows the graphical view

### Listing 1. Building a Bayesian network by means of SMILE

```

void CreateEngineNetwork(void) {
    DSL_network theNet;

    // Create the node Battery
    int battery = theNet.AddNode(DSL_CPT,"Battery");

    // Set the number (and names) of states
    DSL_idArray someNames;
    someNames.Add("Loaded");
    someNames.Add("Empty");
    theNet.GetNode(battery)->Definition()->
        SetNumberOfOutcomes(someNames);

    // Create the node Radio
    int radio = theNet.AddNode(DSL_CPT,"Radio");

    // Set the number (and names) of states
    someNames.Flush();
    someNames.Add("Operates");
    someNames.Add("DoesNotOperate");
    theNet.GetNode(radio)->Definition()->
        SetNumberOfOutcomes(someNames);

    // Add an arc from Battery to Radio
    theNet.AddArc(battery,radio);

    // Create the probability distribution for the node
    // Battery using a direct method (creating the entire
    // probability table)
    DSL_doubleArray theProbs;
    theProbs.SetSize(2);
    theProbs[0] = 0.9;
    theProbs[1] = 0.1;
    theNet.GetNode(battery)->Definition()->
        SetDefinition(theProbs);

    // Create the conditional probability distribution for the
    // node Radio using a system of coordinates (an alternative
    // way of creating a probability table).
    DSL_sysCoordinates theCoordinates(
        *theNet.GetNode(radio)->Definition());
    theCoordinates.UncheckedValue() = 0.95;
    theCoordinates.Next();
    theCoordinates.UncheckedValue() = 0.05;
    theCoordinates.Next();
    theCoordinates.UncheckedValue() = 0.0;
    theCoordinates.Next();
    theCoordinates.UncheckedValue() = 1.0;

    ...

    // Save the network in a disk file.
    theNet.WriteFile("engine.dsl");
};

```

This modelling environment is called SMILE (Structural Modelling, Inference, and Learning Engine) and is a fully portable library of C++ classes implementing Bayesian networks and influence diagrams, suitable for direct inclusion in intelligent systems. Its Windows user interface, GeNIe, is a versatile and user-friendly development environment for graphical decision-theoretic models. SMILE comes in different shapes and sizes, including a .NET version (SMILE.NET), a Java version (jSMILE) and even a version for Pocket PC (Pocket SMILE). Our software has been made available to the community in July 1998 and, as of November 2004, we have several thousand users worldwide. Applications based on GeNIe or SMILE range from heavy-duty industrial software to academic research projects. Some examples are: battle damage assessment, group decision support models for regional conflict detection, intelligent tutoring systems, medical diagnosis and therapy planning, diagnosis of diesel locomotives, airplanes, and production processes of integrated circuits. GeNIe and SMILE have been also used in teaching statistics and decision-theoretic methods at several universities.

In the development of GeNIe, we put emphasis on accessibility and friendliness of the user interface (see the above snapshot of the program interface). The architecture of the system is flexible: SMILE is the core reasoning engine that can be embedded in dedicated user interfaces (GeNIe is in fact one such interface!). We use GeNIe in teaching and the help system includes many useful documents and tutorials. We believe that it is basically a standalone guide to decision-theoretic modelling.

### Embedding SMILE in C++ programs

SMILE can be embedded in all programming languages that interface with C++. In addition to C++ itself, the SMILE download site contains interfaces to Java and to programming environments based on the .NET standard. Listing 1 shows a fragment of a C++ program that creates two nodes of the example network for diagnosing problems with starting a car engine. Comments in the code should explain what the individual parts of the function are doing.

### Further Reading

Readers interested in learning more about the normative systems and decision-theoretic methods, are encouraged to refer to the following:

- Marek J. Druzdziel and Roger R. Flynn, *Decision Support Systems*, In Encyclopedia of Library and Information Science, Vol. 67, Suppl. 30, Allen Kent (ed.), Marcel Dekker, Inc.  
<http://www.pitt.edu/~druzdziel/abstracts/dss.html>
- Max Henrion, John S. Breese, and Eric J. Horvitz, *Decision Analysis and Expert Systems*, AI Magazine, 12(4), Winter 1991
- Eugene Charniak, *Bayesian networks without tears*, AI Magazine, 12(4), Winter 1991
- Judea Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann Publishers, Inc., 1988
- Electronic proceedings of the annual Conference on Uncertainty in Artificial Intelligence  
<http://www.sis.pitt.edu/~dsl/uai.html>
- GeNIe and SMILE manuals and on-line help  
<http://www.sis.pitt.edu/~genie/>

### Listing 2. Inference with a Bayesian network model using SMILE

```
void InferenceWithEngineNet(void) {
    DSL_network theNet;
    theNet.ReadFile("engine.dsl");

    // Choose the default algorithm
    // for all subsequent computations.
    theNet.SetDefaultBNAlgorithm(DSL_ALG_BN_LAURITZEN);

    // Introduce the observation that the radio operates
    // We happen to know that 0 is the index of state Operates
    // but we can also find it (see how this is done for the
    // node Battery below).
    theNet.GetNode(radio)->Value()->SetEvidence(0);

    // Update the network: This single call to the default
    // belief updating algorithm propagates the evidence
    // through the entire network
    theNet.UpdateBeliefs();

    // Retrieve the probability that the battery is empty.
    // First get the handle of the node Battery
    int battery = theNet.FindNode("Battery");

    // Get P(Battery = Empty)
    DSL_sysCoordinates theCoordinates(*theNet.GetNode(
        battery)->Value());
    DSL_idArray *theNames;
    theNames = theNet.GetNode(battery)->
        Definition()->GetOutcomesNames();
    int moderateIndex = theNames->FindPosition("Empty");
    theCoordinates[0] = moderateIndex;
    theCoordinates.GoToCurrentPosition();
    double P_BatteryIsEmpty =
        theCoordinates.UncheckedValue();
    printf("P(\"Battery\" = Empty) = %f\n", P_BatteryIsEmpty);
};
```

A Bayesian network saved in a disk file, whether it has been created and saved by a C++ program (like the above function `CreateEngineNetwork()`), learned from data, or created in an interactive mode using GeNIe, can be embedded in an application program. Listing 2 shows how a previously saved network can be loaded and used to compute posterior probability distribution of a network node.

The code examples in Listings 1 and 2 are oversimplified but they should give the reader a taste of what it takes to embed Bayesian networks in an application program. The entire SMILE library contains many functions that allow accomplishing programming tasks in more efficient ways. We encourage the reader to visit the Web site of the Decision Systems Laboratory, download GeNIe and SMILE, and try them out. Our software has many users and I dare to say that it is well-documented, efficient, user- and programmer-friendly, and very reliable. We are responsive to potential problems – please do not hesitate to contact us in the unlikely case that you encounter any! ■