**Notes on CSC 505 (Algorithmic Techniques for Smart Systems)**

# 1. ADVERSARIAL SEARCH

An adversarial search is the type of search in which two or more players or agents with conflicting interests are trying to explore the same search space for a solution. This kind of search space where more than one agent search for solution to problem is called multi-agent environment. Here, each agent is an opponent of the other and they all play against each other. This situation usually occurs in game playing.

A game is a type of adversarial search which can be formalized with the following elements:

i. **Initial state**: It specifies how the game is set up at the start.
ii. **Player(s)**: It specifies which player has moved in the state space.
iii. **Action(s)**: It returns the set of legal moves in state space.
iv. **Result(s, a)**: It is the transition model, which specifies the result of moves in the state space.
v. **Terminal-Test(s)**: Terminal test is true if the game is over, else it is false at any case. The state where the game ends is called terminal states.
vi. **Utility(s, p)**: A utility function gives the final numeric value for a game that ends in terminal states s for player p. It is also called payoff function.

## Zero-Sum Game

A Zero-sum game is an adversarial search which involves pure competition. In Zero-sum game each agent's gain or loss of utility is exactly balanced by the losses or gains of utility of another agent. One player of the game tries to maximize one single value, while other player tries to minimize it. Each move by one player in the game is known as ply. Chess and Checkers are examples of Zero-sum game.

## Basic Concepts of Adversarial Search

### A. Game Theory

Game theory is a branch of mathematics that deals with the study of decision-making in situations where two or more agents have conflicting interests. In the context of adversarial search, game theory provides a framework for modeling two-player games, such as chess or checkers, as well as more complex, multi-agent systems. In game theory, a game is modeled as a set of players, strategies, and payoffs. The players take turns making moves, and the outcome of the game is determined by the final state of the game board and the payoffs associated with each possible outcome. Game theory provides a way to analyze the game and determine the best strategy for each player given the available information.

### B. Minimax Algorithm

The minimax algorithm is a classical method for solving two-player, zero-sum games, such as chess or checkers. The algorithm works by assuming that each player will make the move that minimizes the maximum possible payoff for the opponent. In other words, each player tries to

minimize their worst-case scenario. The minimax algorithm works by recursively exploring the game tree, starting from the current state of the game board. At each node in the tree, the algorithm calculates the minimax value, which is the minimum payoff that the maximizing player can guarantee against any move by the minimizing player. The minimax value of a node is used to determine the best move for the maximizing player. The algorithm continues exploring the tree until it reaches a terminal state, at which point the payoffs associated with that state are returned.

### C. Alpha-Beta Pruning

Alpha-beta pruning is a technique for improving the efficiency of the minimax algorithm. The basic idea behind alpha-beta pruning is to eliminate parts of the game tree that are not relevant to the current search. This is done by maintaining two values, alpha and beta, that represent the best payoff that the maximizing player and the minimizing player can guarantee, respectively. At each node in the game tree, the algorithm first checks whether the current value of beta is less than or equal to alpha. If this is the case, the algorithm can stop searching that branch of the tree, as it is guaranteed that the maximizing player will not choose a move that leads to a state with a payoff greater than beta. This significantly reduces the size of the game tree that needs to be explored, making the minimax algorithm much more efficient.

### D. Evaluation Functions

An evaluation function is a function that assigns a numerical value to the current state of a game, indicating how desirable that state is for a particular player. Evaluation functions are used to determine the best move for a player in situations where it is not possible to fully explore the game tree. Evaluation functions are designed to reflect the goals and objectives of a player. For example, in chess, an evaluation function might take into account factors such as the number of pieces on the board, the mobility of the pieces, and the control of the center of the board. The evaluation function is used to determine the best move for a player at each step of the game, without the need to explore the entire game tree.

### E. Cut-Off Strategies

Cut-off strategies are methods for limiting the depth of the game tree that is explored by the minimax algorithm. These strategies are used to reduce the computational complexity of the algorithm, as well as to make it more practical for real-world applications. Cut-off strategies are an important aspect of adversarial search, as they allow the algorithm to find a good solution in a reasonable amount of time. In many cases, a suboptimal solution that is found quickly is preferable to an optimal solution that takes an unreasonable amount of time to find. There are several different types of cut-off strategies, including depth-limited search, time-limited search, and iterative deepening search. Depth-limited search stops the search when the algorithm reaches a certain depth in the game tree, while time-limited search stops the search after a certain amount of time has passed. Iterative deepening search starts with a shallow depth limit and gradually increases the depth limit until the desired level of search is achieved.

### Real-life Applications of Adversarial Search

Some application areas of adversarial search algorithms include:

### A. Chess

The main challenge in building a chess program is to search through the vast number of possible moves and positions in a reasonable amount of time. This is accomplished using a combination of minimax algorithms, alpha-beta pruning, and various cut-off strategies. In addition, chess programs use sophisticated evaluation functions that take into account factors such as piece mobility, pawn structure, and king safety. For Chess, the outcomes are a win, loss, or draw and its payoff values are +1, 0, ½.

### B. Checkers

Like chess, checkers is a two-player, zero-sum game, and the goal is to find the best move for a player given the current state of the game board. Checkers programs use a combination of minimax algorithms, alpha-beta pruning, and various cut-off strategies to search through the possible moves and positions. In addition, checkers programs use sophisticated evaluation functions that take into account factors such as piece mobility and king safety.

### C. Othello

Othello is a two-player game that is played on a board with 64 squares. The game is played by placing pieces on the board, and the goal is to have the most pieces of your color on the board at the end of the game. Othello programs use a combination of minimax algorithms, alpha-beta pruning, and various cut-off strategies to search through the possible moves and positions. In addition, Othello programs use sophisticated evaluation functions that take into account factors such as piece mobility and king safety.

### D. Go

Go is an ancient board game that is played on a 19x19 grid. The game is played by placing stones on the board, and the goal is to control as much territory as possible. Go programs use a combination of minimax algorithms, Monte Carlo tree search, and artificial neural networks to search through the possible moves and positions. In addition, Go programs use sophisticated evaluation functions that take into account factors such as territory control and piece mobility.

### E. Video Games

Adversarial search algorithms are also widely used in video games, where they are used to control the behavior of non-player characters (NPCs). For example, in first-person shooter games, NPCs use adversarial search algorithms to determine the best way to attack the player, while in strategy games, NPCs use adversarial search algorithms to determine the best way to build their army and conquer territory. Video game combines minimax algorithms, Monte Carlo tree search, and reinforcement learning to implement the behavior of NPCs. In addition, video game developers often use sophisticated evaluation functions that take into account factors such as player behavior, environmental conditions, and resource availability.

## Adversarial Search-based Algorithms

Adversarial search algorithms can address specific challenges or even improve performance. Some of them are:

### A. Monte Carlo Tree Search

Monte Carlo tree search is a variant of adversarial search that uses random sampling to guide the search process. The basic idea behind Monte Carlo tree search is to simulate random playouts from each node in the game tree and use the results of these playouts to guide the search. Monte Carlo tree search has been used with great success in games such as Go, where it has been shown to outperform traditional minimax algorithms. The key advantage of Monte Carlo tree search is that it is able to handle the large branching factor and uncertainty inherent in many games, making it a more practical solution in many real-world scenarios.

### B. Evolutionary Algorithms

Evolutionary algorithms are a class of optimization algorithms that are inspired by the process of natural selection. In the context of adversarial search, evolutionary algorithms have been used to optimize the evaluation functions used by traditional minimax algorithms. The basic idea behind evolutionary algorithms is to use a population of candidate solutions, which represent different possible evaluation functions. The solutions are evaluated based on their performance in a set of games, and the best solutions are selected and recombined to form a new generation of solutions. This process is repeated until a satisfactory solution is found.

### C. Reinforcement Learning

Reinforcement learning is a branch of machine learning that deals with learning from experience. In the context of adversarial search, reinforcement learning has been used to learn the behavior of agents in complex, dynamic environments. The basic idea behind reinforcement learning is to have an agent learn from experience by taking actions in an environment and receiving rewards based on the outcomes of those actions. Over time, the agent learns to make better decisions based on the rewards it has received in the past.

Reinforcement learning has been used with great success in video games, where it has been used to learn the behavior of non-player characters (NPCs). The key advantage of reinforcement learning is that it can learn from experience in real-time, allowing it to adapt to changing conditions and opponents.

### D. Artificial Neural Networks

Artificial neural networks are a type of machine learning algorithm that are inspired by the structure and function of the human brain. In the context of adversarial search, artificial neural networks have been used to optimize the evaluation functions used by traditional minimax algorithms and to learn the behavior of agents in complex, dynamic environments. The basic idea behind artificial neural networks is to use a network of interconnected nodes, or neurons, to model complex relationships between inputs and outputs. In the context of adversarial search, artificial neural networks have been used to model the behavior of agents and to learn from experience.

Artificial neural networks have been used with great success in games such as Go, where they have been used to optimize the evaluation functions used by traditional minimax algorithms and to learn the behavior of agents. The key advantage of artificial neural networks is their ability to model complex relationships and to learn from experience, making them a powerful tool for solving real-world problems.

## 2. SEARCH UNDER UNCERTAINTY

Search under uncertainty is a critical problem which involves finding an optimal solution from a set of possible outcomes when the outcome probabilities are uncertain. The problem can be formalized as follows: given a set of possible actions, each with an associated probability distribution over outcomes, the goal is to select the action that maximizes the expected value of some objective function. The objective function could be a reward function, a cost function, or some combination of the two.

Search under uncertainty is made when we lack complete information about the state of the world and are forced to make decisions based on incomplete or imperfect information. For instance, imagine you are looking for a supermarket around an area, and you know everything about the supermarket without the exact location of the building. You can meet a random stranger on the road and ask them, and tell you to keep going down then turn left. You could reach that area and still not see the building. This kind of scenario describes searching with imprecise or uncertain details.

**Kinds of Uncertainty**

i. **Environment Uncertainty**: This uncertainty that is a property of the observable environment. Example; the coin toss with a 50:50 chance of heads or tails output.
ii. **Model Uncertainty**: This is the agents own lack of knowledge about the environment. The agent's goal is to reduce model uncertainty. For example, an agent in a maze has to choose between three paths in order to find the one that will lead to a treasure. There is no environment uncertainty in this case, as the treasure is already known to be at the end of one of the paths. The agent's uncertainty will now stem from the fact that it has incomplete knowledge about the environment.

**Causes of Uncertainty**

There are several causes of uncertainty:

i. **Noise in Observation:** Considering a set M, an observation for the domain is often referred to as an "instance" or "sample" and is one row of data. It is what is measured or collected. Observations from the domain are not crisp, instead they contain noise. Noise refers to the variability in these observations, as the results will differ from each other and will yield different results. Noise can be in the form of an error, like a slight slip of the print on the measuring instrument.
ii. **Incomplete Coverage of the Domain:** This refers to a situation in which samples from a collection are chosen to represent the opinion of the entire collection. This could lead to errors since not all domain are covered. Take for instance, you are trying to measure the size of the elephant grass from a section of a football field. The grasses are randomly selected but scope is heavily limited to just a section of the field, leading to error of incomplete coverage.

iii. **Imperfect Model of the Problem:** This refers to the imperfect nature of systems especially as it affects the development process, choice and preparation of data, as well as the choice of training hyper-parameters that could collectively result in minimal model errors. Model errors could be imperfect predictions such as predicting a quantity in a regression problem that is quite different to what was expected.

**Dynamic Programming**

Dynamic programming is a classic approach for solving sequential decision-making problems under uncertainty. The basic idea is to represent the problem as a Markov decision process (MDP) and use dynamic programming algorithms to compute an optimal policy. In an MDP, the state of the world evolves over time according to a probabilistic transition function, and the agent can take actions that influence the state transition probabilities. The goal is to find a policy that maximizes the expected sum of rewards over a finite or infinite time horizon.
Dynamic programming algorithms such as value iteration and policy iteration are computationally efficient and can handle large state spaces. However, they require complete knowledge of the transition probabilities and the reward function, which is often unrealistic in practice. Moreover, they assume that the problem can be represented as an MDP, which may not always be the case.

**How Dynamic Programming Works**

Dynamic programming works by storing the result of sub problems so that when their solutions are required, they are at hand and we do not need to recalculate them. This technique of storing the value of sub problems is called **memoization**. By saving the values in the array, we save time for computations of sub-problems we have already come across. Dynamic programming by memoization is a top-down approach to dynamic programming. By reversing the direction in which the algorithm works i.e., by starting from the base case and working towards the solution, we can also implement dynamic programming in a bottom-up manner.

## 3. LOGICAL REASONING AND PLANNING

Logical reasoning is a type of problem-solving that involves working through a set of rules that govern a scenario. This set of rules or steps is referred to as an algorithm. Logical reasoning involves testing different sets of steps or algorithms to determine which sequence of rules leads to the correct solution. In practice, it involves using given data to determine or to deduce other facts. Logical reasoning can be deductive, inductive, or abductive. Deductive reasoning starts by presenting premises and relations of the premises, which can be followed to reach a solid conclusion. There is a guaranteed certainty involved in deductive reasoning. It takes a general rule (or rules) and uses them to arrive at a specific conclusion that is always true. For example, if A = B and B = C (the premises), the inevitable conclusion is that A = C because equality is a transitive

relation. Inductive reasoning involves beginning with observations to arrive at premises as well as relations between premises, which are then used to arrive at conclusions. This conclusion has less certainty than in deductive reasoning. Example, Jimmy is a dog and Jimmy is four-legged. Therefore, all dogs have four legs. Abductive reasoning involves an incomplete observation that is used to determine the best prediction. It represents the best guess that can be made. Example, all dogs have four legs and Jimmy is four-legged. Therefore, Jimmy is a dog.

**Steps In Logical Reasoning**

1. **Identify the problem or issue**: Identify the question or issue that needs to be addressed.
2. **Gather information**: Collect relevant information and data to inform your analysis.
3. **Formulate a hypothesis or claim**: Based on the information you have gathered, formulate a tentative conclusion or claim about the problem or issue.
4. **Test the hypothesis or claim:** Use evidence to test the validity of the hypothesis or claim.
5. **Evaluate the evidence**: Assess the quality and relevance of the evidence, and consider alternative explanations.
6. **Draw a conclusion**: Based on the evidence and analysis, draw a conclusion or judgment about the problem or issue.
7. **Communicate your conclusion**: Communicate your conclusion clearly and succinctly, and be prepared to defend your reasoning if necessary.

**Relationship Between Logical Reasoning and Planning**

Logical reasoning and planning are closely related concepts, as both involve the use of cognitive processes to achieve a goal or solve a problem.

Logical reasoning is the process of drawing conclusions from premises, using reasoning and evidence to form an argument or solve a problem. It is a fundamental cognitive process that is necessary for effective problem-solving and decision-making. Logical reasoning involves the use of deductive, inductive, and abductive reasoning, as well as the application of critical thinking skills.

Planning, on the other hand, is the process of establishing a course of action or a set of steps that need to be taken to achieve a goal or solve a problem. Planning involves analyzing the problem, setting goals, identifying potential obstacles, and developing strategies to overcome those obstacles. Effective planning requires the use of logical reasoning, as the planner must be able to analyze the problem, identify potential solutions, and determine the best course of action.

In essence, logical reasoning is an essential component of effective planning. Logical reasoning is necessary to analyze the problem, identify potential solutions, and determine the best course of action. Planning, in turn, is a way of applying logical reasoning to achieve a specific goal. The two concepts are complementary, and effective planning requires the use of logical reasoning skills.

## 4. FUZZY LOGIC

Fuzzy logic is a form of many-valued logic in which the truth value of variables may be any real number between 0 and 1. It is employed to handle the concept of partial truth, where the truth value may range between completely true and completely false. Fuzzy logic is a unique heuristic approach to solving logical problems in computing based on “degrees of truth” rather than “true and false” that is associated with Boolean logic. A simplified way of explaining this is through the use of the image in fig. 1.


Fig 1. Boolean logic versus Fuzzy logic.

Figure 1 shows the conventional logical evaluation of if an object is hot or cold. Boolean logic involves only two possible outcomes, true or false (1 or 0). However, Fuzzy logic seeks to evaluate all degrees between the two thresholds of yes or no:

i. Is the object very much hot?
ii. Is it moderately hot?
iii. Or a little hot?

Fuzzy logic is based on the observation that people make decisions based on imprecise and inaccurate numerical information. Its models are designed to make decisions on imprecise information and situations plagued by uncertainty and vagueness (hence its name: fuzzy). Generally, the term fuzzy refers to the vast number of scenarios that can be developed in a decision tree-like system. Developing fuzzy logic protocols can require the integration of rule-based programming. These programming rules may be referred to as fuzzy sets since they are developed at the discretion of comprehensive models. Thus, on a broader scale, fuzzy logic forms the basis for artificial intelligence systems programmed through rules-based inferences.

**Fuzzy logic architecture**

The fuzzy logic architecture is shown in fig. 2.

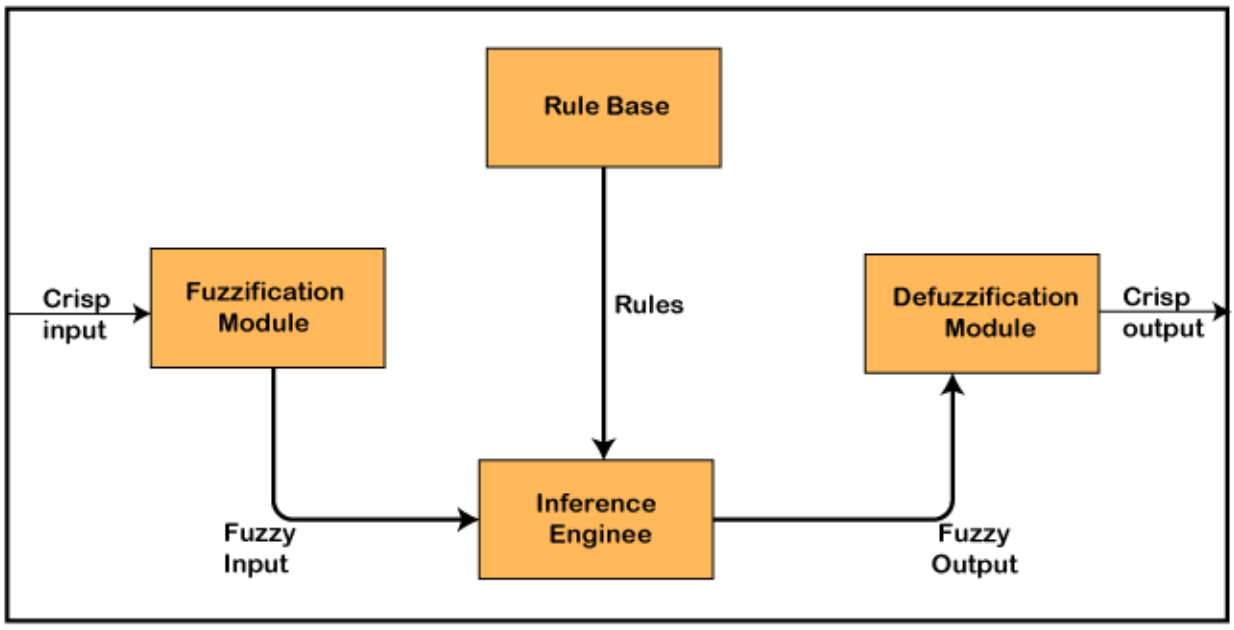


Fig. 2. Fuzzy Logic Architecture.

The fuzzy logic consists of the following components:

i. **Rule Base**: This contains the rules and membership functions that regulate or control decision-making in the fuzzy logic system. It also contains the IF-THEN conditions used for conditional programming and controlling the system.
ii. **Fuzzifier**: This component transforms raw inputs into fuzzy sets. The fuzzy sets proceed to the control system, where they undergo further processing.
iii. **Inference Engine**: This is a tool that establishes the ideal rules for a specific input. It then applies these rules to the input data to generate a fuzzy output.
iv. **Defuzzifier**: This component transforms the fuzzy sets into an explicit output (in the form of crisp inputs). Defuzzification is the final stage of a fuzzy logic system.

Implementation Steps of Fuzzy Logic

**Step 1**: In a fuzzy logic system, we can use linguistic terms to describe different categories of a situation, say, temperature. Some of the linguistic terms used in temperature case include very cold, hot, very warm, warm, cold, and very hot. Temperature can be termed as a fuzzy set t consisting of the aforementioned linguistic terms.

**Step 2**: After defining the linguistic terms, we should create membership functions. A membership function is a graphical representation of a fuzzy set. In our case, fuzzy set (t) is graphically represented. The input temperature is on the x-axis, while the degrees of membership are on the y-axis. The membership function computes the degrees of membership of various temperature elements.

**Step 3**: In the third step, we will construct rules for controlling the air conditioner's temperature. Here, we can apply the IF-THEN logic to set effective rules. For example, the following IF-THEN conditions can be made.

i. IF the room temperature is very cold, and the desired temperature is very warm, THEN heat.
ii. IF the room temperature is very hot, and the required temperature is cold, THEN cool.
iii. IF the room temperature is warm, and the required temperature is warm, THEN no action should be taken.

In the three IF-THEN conditions, heat, cool, and no action represent the actions that need to be taken after the IF-THEN conditions have been met.

**Step 4**: After setting the system's rules, the fuzzifier uses them to transform the raw input into fuzzy sets. This is done through fuzzy operations (e.g., Max and Min). These fuzzy sets are used to generate a membership function output.

**Step 5**: Defuzzification: This is the last algorithm step. In this step, the defuzzifier uses the membership function to establish the output temperature.

## 5. QUALIFYING UNCERTAINTY (IN PROBABILISTIC REASONING)

Probabilistic reasoning is a form of knowledge representation in which the concept of probability is used to indicate the degree of uncertainty in knowledge. It is a reasoning that deals with uncertainty and likelihoods of something happening based on available evidence and prior knowledge. It involves using probability theory to quantify uncertainty and to calculate the likelihood of different outcomes. Probabilistic reasoning is important because many real-world problems involve uncertainty and incomplete information. It helps to make better decisions, reduce risk, and improve the understanding of complex systems. Probabilistic models are used to examine data using statistical codes.

**Levels of Uncertainty Reasoning**

There are four levels of uncertainty reasoning:

Level one: A clear enough future.

This involves developing single forecast that sufficiently provides a precise basis for planning. To help generate this usefully precise prediction of the future, managers can use the standard strategy tool kit: market research, analyses of competitors' costs and capacity, value chain analysis, Michael Porter's five-forces framework, etc.

Level two: Alternative futures.

The future can be described as one of a few discrete scenarios. An analysis cannot identify which of them that will materialize. Instead, certain factors determine which scenario to be implemented. Therefore, the scenario or strategy that eventually come to pass depend on the factor that first emerged.

Level three: A range of futures.

A limited number of key variables define a range of potential futures, but the actual outcome may lie anywhere within it. There are no natural discrete scenarios.

Level four: True ambiguity.

A number of dimensions of uncertainty interact to create an environment that is virtually impossible to predict. It is impossible to identify a range of potential outcomes, let alone scenarios within a range. Level four situations are quite rare, and they tend to migrate toward one of the others over time. Nevertheless, they do exist.

**Rules In Qualification of Probabilistic Reasoning**

1. All relevant facts and evidence must be considered.
2. Empirical evidence should be considered to support claims.
3. Probabilistic reasoning should account for invisible and varying facts and opinions.
4. The results must be objectively derived using appropriate means.
5. Probability estimates must be consistent with past experience and current scientific knowledge.
6. The accuracy of the results must be verifiable.
7. The results should be tested against real-world data.
8. Models should incorporate feedback loops to improve accuracy over time.
9. Where possible, data should be analyzed using multiple models for comparison and validation.
10. The results need to be replicable and verifiable.

## 6. BAYESIAN NETWORKS

A Bayesian network is a compact, flexible and interpretable representation of a joint probability distribution. They belong to the family of probabilistic graphical models and are used to represent knowledge about an uncertain domain. Bayesian networks involve updating our beliefs about a situation based on new evidence. Hence, they are also called Belief Reasoning. They consist of

two parts: a structure and parameters. The structure is a directed acyclic graph (DAG) that expresses conditional independencies and dependencies among random variables associated with nodes. The parameters consist of conditional probability distributions associated with each node. Each node in the graph represents a random variable, while the edges between the nodes represent probabilistic dependencies among the corresponding random variables. In other words, a Bayesian network can be described as a type of graphical model that uses probability to determine the occurrence of an event.

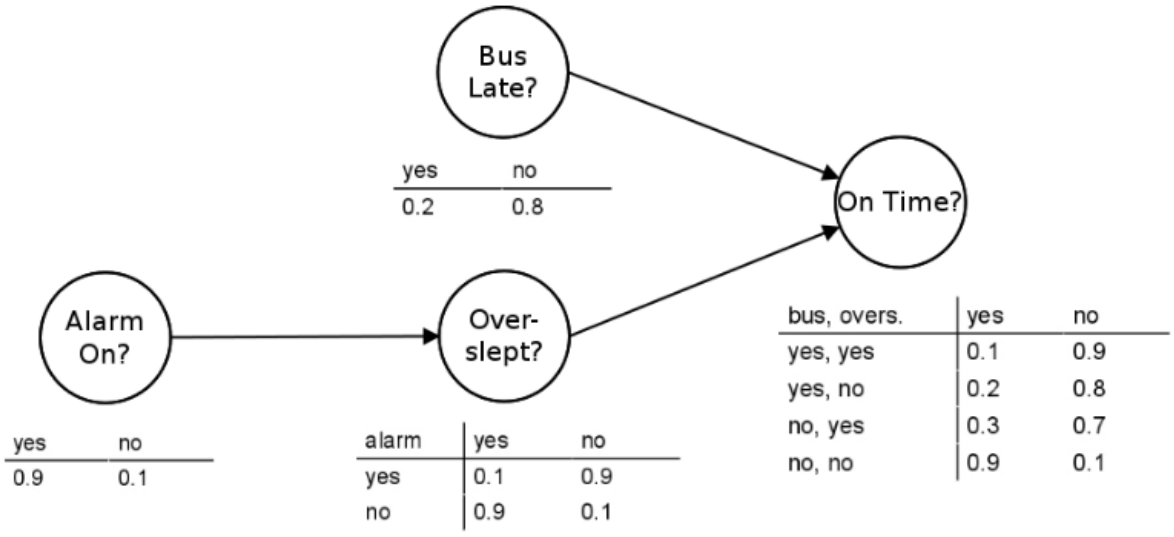


Fig3. A typical Bayesian Network.

**Basic Concepts of Bayesian Networks**

**i. Joint probability**

Joint probability is a probability of two or more events happening together. For example, the joint probability of two events A and B is the probability that both events occur, P(A∩B).

**ii. Conditional probability**

Conditional probability defines the probability that event B will occur, given that event A has already occurred. There are two ways joint probability can be represented:

```
If A and B are dependent events  p(B/A) = P( A and B)/P(A)
If A and B are independent events : P(B/A) = P(B)
```

The conditional probability distribution of each node is represented by a table called the "node table". It contains two columns, one for each possible state of the parent node (or "parent random variable") and one for each possible state of the child node (or "child random variable"). The rows in this table correspond to all possible combinations of parent and child states. In order to find out

how likely it is that a certain event will happen, we need to sum up the probabilities from all paths of that event.

### iii. Bayes' Theorem

Bayesian Networks are based on Bayes' theorem, which states that the probability of a hypothesis (an event or a state of the world) given some evidence is proportional to the product of the prior probability of the hypothesis and the likelihood of the evidence given the hypothesis. Bayesian Networks use Bayes' theorem to update the probabilities of hypotheses as new evidence is observed.

### iv. Directed Acyclic Graphs (DAGs)

Bayesian Networks are represented by directed acyclic graphs (DAGs), which are graphs with directed edges between nodes that do not form cycles. The DAG specifies the structure of the network, with each node representing a random variable and each edge representing a probabilistic relationship between the connected nodes.

### v. Conditional Probability Tables (CPTs)

Each node in a Bayesian Network has an associated conditional probability table (CPT) that specifies the probability distribution of that node given its parent nodes in the DAG. CPTs represent the conditional dependencies between the variables, which are necessary for inference and prediction.

### vi. Inference

Inference in Bayesian Networks involves computing the posterior probabilities of the nodes given observed evidence. This can be done using various inference algorithms, such as exact inference algorithms, approximate inference algorithms, or sampling methods

**Application of Bayesian Networks**

Bayesian networks find applications in a variety of tasks such as:

1. **Spam filtering**: A spam filter is a program that helps in detecting unsolicited and spam mails. Bayesian spam filters check whether a mail is spam or not. They use filtering to learn from spam and ham messages.
2. **Biomonitoring**: This involves the use of indicators to quantify the concentration of chemicals in the human body. Blood or urine is used to measure the same.
3. **Information retrieval**: Bayesian networks assist in information retrieval for research, which is a constant process of extracting information from databases. It works in a loop. Hence, we have to continuously reconsider and redefine our research problem to avoid data overload.
4. **Image processing**: A form of signal processing, image processing uses mathematical operations to convert images into digital format. Once images are converted, their quality

can be enhanced with more operations. The input image doesn't necessarily have to be in the form of an image; it could be a photograph or a video frame.

5. **Gene regulatory network**: A Bayesian network is an algorithm that can be applied to gene regulatory networks in order to make predictions about the effects of genetic variations on cellular phenotypes. Gene regulatory networks are a set of mathematical equations that describe the interactions between genes, proteins, and metabolites. They are used to study how genetic variations affect the development of a cell or organism.
6. **Turbo code**: Turbo codes are a type of error correction code capable of achieving very high data rates and long distances between error correcting nodes in a communications system. They have been used in satellites, space probes, deep-space missions, military communications systems, and civilian wireless communication systems, including WiFi and 4G LTE cellular telephone systems.
7. **Document classification**: This is a problem often encountered in computer science and information science. Here, the main issue is to assign a document multiple classes. The task can be achieved manually and algorithmically. Since manual effort takes too much time, algorithmic documentation is done to complete it quickly and effectively.

## 7. DYNAMIC BAYESIAN NETWORKS

Dynamic Bayesian Networks are an extension of the traditional Bayesian Networks (BNs) that model static systems. They are probabilistic models that are widely used for modelling dynamic systems that evolve over time. Dynamic Bayesian Networks are a generalization of Bayesian Networks that can model systems with time-dependent variables, such as financial markets, weather patterns, and medical diagnoses.

Dynamic Bayesian Networks extend the structure of static Bayesian Networks to include a temporal component, allowing for the representation of time-varying systems. The structure of a Dynamic Bayesian Network is composed of two components: the dynamic model and the observation model.

1. **Dynamic Model**: The dynamic model of a Dynamic Bayesian Network specifies the evolution of the system over time. It is represented by a series of interconnected Bayesian Networks, where each network represents the state of the system at a particular time step. The structure of the dynamic model is similar to that of a static Bayesian Network, with each node representing a random variable and each edge representing a probabilistic relationship between the connected nodes. However, in a Dynamic Bayesian Network, the nodes are divided into two sets: the hidden variables and the state variables. The hidden variables represent the unobserved variables that drive the evolution of the system, while the state variables represent the variables that are observed at each time step.
2. **Observation Model**: The observation model of a Dynamic Bayesian Network specifies the relationship between the hidden variables and the observed variables. It is represented

by a set of conditional probability tables (CPTs) that describe the probability distribution of the observed variables given the values of the hidden variables. The observation model allows for the incorporation of noisy or incomplete observations into the Dynamic Bayesian Network.

The combination of the dynamic model and the observation model allows for the representation of complex time-varying systems and the prediction of future states of the system based on observed evidence.

**Inference In Dynamic Bayesian Networks**

Inference in Dynamic Bayesian Networks (DBNs) involves computing the posterior probability distribution over the current state of the system, given the observed evidence and the previous state of the system. The posterior distribution is used to make predictions about the future state of the system and to update the belief state of the system over time. Inference in Dynamic Bayesian Networks can be performed using various techniques, including filtering, smoothing, prediction, and control.

i. **Filtering:** Filtering is the process of computing the posterior probability distribution over the current state of the system given the observed evidence up to the current time step. Filtering is often used in real-time applications, where the goal is to estimate the current state of the system based on the available evidence. Filtering can be performed using the Forward algorithm, which is a dynamic programming algorithm that computes the posterior distribution over the current state of the system based on the evidence up to the current time step. The Forward algorithm is based on the Bayesian rule and the Markov property of Dynamic Bayesian Networks. The Forward algorithm works by iteratively computing the posterior probability distribution over the current state of the system at each time step. The algorithm starts by initializing the posterior distribution over the initial state of the system. Then, for each time step, the algorithm updates the posterior distribution over the current state of the system based on the evidence and the transition probabilities of the DBN. The posterior distribution over the current state of the system can be used to make predictions about the future behaviour of the system and to update the belief state of the system over time.

ii. **Smoothing:** Smoothing is the process of computing the posterior probability distribution over the past and present states of the system given all of the available evidence. Smoothing is often used in offline applications, where the goal is to estimate the complete trajectory of the system over a period of time. Smoothing can be performed using the Forward-Backward algorithm, which is a dynamic programming algorithm that computes the posterior probability distribution over the past and present states of the system based on all available evidence. The Forward-Backward algorithm is based on the Bayesian rule and the Markov property of Dynamic Bayesian Networks. The Forward-Backward algorithm works by first computing the posterior distribution over the current state of the system using the Forward algorithm. Then, it computes the posterior distribution over the past and

present states of the system by iteratively smoothing the posterior distribution over the current state of the system backward in time. The posterior distribution over the past and present states of the system can be used to estimate the complete trajectory of the system over a period of time and to update the belief state of the system over time.

iii. **Prediction:** Prediction is the process of computing the posterior probability distribution over the future state of the system given the observed evidence up to the current time step. Prediction is often used in real-time applications, where the goal is to predict the future behaviour of the system based on the available evidence. Prediction can be performed using the Forward algorithm, which is a dynamic programming algorithm that computes the posterior probability distribution over the future state of the system based on the evidence up to the current time step. The Forward algorithm is based on the Bayesian rule and the Markov property of Dynamic Bayesian Networks. The Forward algorithm works by iteratively computing the posterior probability distribution over the future state of the system at each time step. The algorithm starts by initializing the posterior distribution over the initial state of the system. Then, for each time step, the algorithm updates the posterior distribution over the future state of the system based on the evidence and the transition probabilities of the DBN. The posterior distribution over the future state of the system can be used to make predictions about the future behaviour of the system and to update the belief state of the system over time.

iv. **Control:** Control involves using the posterior distribution over the current state of the system to make decisions that optimize some objective function. Control is often used in applications such as robotics and finance, where the goal is to control the behaviour of a system to achieve some desired outcome. The most common control algorithm used in Dynamic Bayesian Networks is the Reinforcement Learning algorithm, which uses the posterior distribution over the current state of the system to learn a policy that maximizes some reward function.

**Applications of Dynamic Bayesian Networks**

Some examples of the applications of Dynamic Bayesian Networks are:

i. **Robotics:** Dynamic Bayesian Networks are used in robotics to model the evolution of the environment and the robot's sensors over time, and to make decisions about how to act based on that information. For example, a DBN can be used to model the movement of objects in a room and to predict where they will be in the future, allowing the robot to plan its movements accordingly.

ii. **Finance:** Dynamic Bayesian Networks are used in finance to model the evolution of financial markets and to make predictions about future market trends. For example, a DBN can be used to model the relationship between economic indicators such as interest rates and inflation, and to predict the future performance of stocks and other financial instruments.

iii. **Epidemiology:** Dynamic Bayesian Networks are used in epidemiology to model the spread of infectious diseases over time and to make predictions about the future course of an outbreak. For example, a DBN can be used to model the transmission of a disease between individuals and to predict the likelihood of an epidemic occurring.

iv. **Traffic Management:** Dynamic Bayesian Networks are used in traffic management to model the flow of traffic over time and to make decisions about how to optimize traffic flow. For example, a DBN can be used to model the relationship between traffic volume, road capacity, and travel times, and to predict the impact of changes in traffic patterns or road conditions.

v. **Natural Language Processing:** Dynamic Bayesian Networks are used in natural language processing to model the evolution of language over time and to make predictions about the meaning of a sentence or a document. For example, a DBN can be used to model the relationships between words and phrases in text, and to predict the meaning of a sentence based on the context in which it appears.