Artificial Intelligence

By

Dr. R. Anandan, B.E., M.S., Ph.D.,

Chartered Engineer,

Professor,

Department of Computer Science and Engineering

School of Engineering

Vels Institute of Science, Technology and Advanced Studies (VISTAS),

Pallavaram, Chennai - 117.

and

Sindhe Phani Kumar, M.Tech

Assistant Professor,

Department of Computer Science and Engineering

PACE Institute of Technology and Sciences,

ONGOLE- 523 272 Prakasam District,

Andhra Pradesh.

Published by

Centivens Institute of Innovative Research

Private Limited

Artificial Intelligence

Copyright © 2020 by CIIR

All rights reserved. Authorized reprint of the edition published by CIIR. No part of this book may be reproduced in any form without the written permission of the publisher.

Limits of Liability/Disclaimer of Warranty: The authors are solely responsible for the contents of the paper in this volume. The publishers or editors do not take any responsibility for the same in any manner. Errors, if any, are purely unintentional and readers are required to communicate such errors to the editors or publishers to avoid discrepancies in future. No warranty may be created or extended by sales or promotional materials. The advice and strategies contained herein may not be suitable for every situation. This work is sold with the understanding that the publisher is not engaged in rendering legal, accounting, or other professional services. If professional assistance is required, the services of a competent professional person should be sought. Further, reader should be aware that internet website listed in this work may have changed or disappeared between when this was written and when it is read.

CIIR also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic books.

9 788194 293811 >

ISBN 978-81-942938-1-1

Dr. R. Anandan
Sindhe Phani Kumar

Centivens Institute of Innovative Research
302, Urban Homes, Phase 7,
Duvvada, Vizahapattinam,
Andhrapradesh- 530046

Preface

The book **"Artificial Intelligence"** is the consequence of our unyielding work. It is the versatile work that every Engineering and Computer Science students to possess. **Dr.R. Anandan**, Professor, Department of CSE, School of Engineering, VELS Institute of Science, Technology and Advanced Studies, Pallavaram, Chennai – 117 and **Mr.S. Phani Kumar**, Research Scholar in Department of CSE, School of Engineering, VELS Institute of Science, Technology and Advanced Studies, Pallavaram, Chennai – 117 and Assistant Professor in Department of CSE. PACE Institute of Technology and Sciences. Ongole, Andhra Pradesh – 523272. The idea of publishing the book **"Artificial Intelligence"** dawned in our minds which would be of great benefit to Engineering and Computer Science students.

"**Artificial Intelligence**" is one of the most significant areas of Computer Science that has its applications in almost every field. This book **"Artificial Intelligence"** has been authored to help students to learn basic concepts and techniques in with ease.

This book has been prepared as per the syllabus of most of the Indian Foreign Universities and for the courses B.E./B.Sc/BCA/M.Sc./MCA.

The book contains five units; all the units are explained intensely in a eloquent manner. An attempt has been made to keep the language as simple as possible and students will definitely be able to appreciate this book.

We hope that this book will be of great help for the students to understand the subject which in turn will help them to do well in their examination.

Suggestions for further improvement of the book will be thankfully acknowledged.

Dr.R. Anandan
Mr.S. Phani Kumar

Acknowledgement

We owe our deep gratitude to respected Chancellor of Vels Institute of Science, Technology and Advance Studies (VISTAS) **Dr. Ishari K. Ganesh**, M.Com., M.B.A., B.L., Ph.D., and **Dr. Aarthi Ganesh.**, Pro-Chancellor (Academic) for providing us with a prospect to author this book.

We extend our yawning sense of gratitude to **Dr.A. Jothi Murugan**, Vice President (Planning and Development) of Vels Institute of Science, Technology and Advance Studies (VISTAS) for his rock solid backing and generous support he has given throughout our carrier.

We extend our heartfelt thanks to Vice Chancellor **Dr.P. Swaminathan** and **Dr.S. Sriman Narayanan**, Pro Vice Chancellor for motivating us to move on into the path of novelty research and publication of books.

We wish to place on record our thankfulness to **Dr.P. Saravanan**, Registrar for being a source of inspiration to this great endeavor of ours.

We express our obligation to **Dr.E.N. Ganesh**, Dean, School of Engineering, **Mrs. K. Kalaivani**, Head, Department of CSE, School of Engineering for their sustenance and assistance.

We would like to express gratitude to **Dr.M. Venu Gopal Rao**, Chairman and **Dr.M. Sridhar**, Secretary & Correspondent, PACE Institute of Technology & Sciences, Ongole, Andhra Pradesh - 523272 for giving us an opportunity to reach our goals.

We would like to extend special thanks to the Principal **Dr.M. Sreenivasan**, PACE Institute of Technology & Sciences, who encouraged me to complete our task.

We would like to thank **Dr.S Raj Anand**, HOD of CSE, PACE Institute of Technology & Sciences, extended valuable support towards academic progress.

We take this opportunity to thank our contemporaries and our family members for their support and assistance. We thank profusely all those who have helped us in some way or other in bringing out this book.

We thank the publisher for his meticulous effort in bringing out this book. Any suggestion for the improvement of the book is most welcome.

About the Authors

Dr. R. Anandan received his Under Graduate in Computer Science and Engineering from University of Madras, Masters from Birla Institute of Technology and Science. Pilani and Doctoral degree in Computer Science and Engineering from BIHER, Chennai.

He is IBMS/390 Mainframe professional and he is recognized as Chartered Engineer from Institution of Engineers in India and received fellowship from Bose Science Society, India.

He is currently working as Professor, Department of Computing Sciences and Engineering, School of Engineering, Vels Institute of Science, Technology & Advanced Studies (VISTAS), Chennai, Tamil Nadu, India which is Pioneer Institution in Engineering. He has vast experience in corporate and all levels of Academic in Computer Science and Engineering his knowledge of interests not limited to Artificial intelligence Soft Computing, Machine Learning, High Performance Computing, Big data Analytics, Image Processing, 3D printing and knowledge Engineering. Under his eminent guidance 8 Ph.D Research Scholars are pursuing, under his Supervision.

He associated as Member in many reputed International and National societies like International Association of Engineers (IAENG) Hong Kong, Computer Science Teacher Association (CSTA) New York, Universal Association of Computer and Electronics Engineers, USA, Association of Computer Sciences and Information Technology (IACSIT), Singapore, Institution of Engineers, India, Computer Society of India and Indian Society of Technical Education (I.S.T.E).

He serves as Editorial Board Member / Technical Committee/ Reviewer of International Journal such Arab/UK/USA journals (Springer, Thomson Reuters, SCI, Elsevier). He servers as International Committee Members towards International Conference conducted in association with Springer and Scopus.

He has published more than 100 research papers in various International Journal such in Scopus, SCI and referred journal. He received best reviewer award for the year 2017-2018 from The International Arab Journal of Information Technology (SCI Journal) under Elsevier Publication, Netherland and the reviewer of the year 2018 – 2019 from Energy Review Elsevier Publication, Netherland. He received Appreciation award on **Teachers Day** towards publication of articles for the academic year 2017-2019. He is the Recipient of the awards Dr. APJ ABDUL KALAM Best Scientist Award for Computer Science Engineering from BOSE Science Society, Tamil Nadu, India, Distinguished Researcher in Artificial Intelligence Award by Research under Literal Access Awards, India, Outstanding Scientist award by International Research Awards on Engineering, Science and Management, *Award for excellence in Young Researcher in Computer Science Engineering by Global Outreach Research, New Delhi, India and Distinguished Professor Award by Vedant Academics Bangkok Awards – 2019 during International Award Conference on Multi-Disciplinary Research and Application - 2019 held in Kasetsart University, Chatuchak, Bangkok, Thailand. Outstanding Scientist Award in "1st International Scientist Awards on Engineering, Science and Medicine by VDGOOD and Best Paper Award in International Conference on Technology Innovation and Data Sciences by LINCOLN UNIVERSITY COLLEGE, SELANGOR, MALAYSIA..* He authored books in Software Engineering, Advanced Java Programming, Functions of Hardware and Internet, *Bigdata and Hadoop, Beginners Guide for OOPS Concepts in Java Programming,* Information Security, Statistics with R Programming and A Text Book on Web Technology and published Chapters in leading publications. He filed 4 patents of his work and produced 3 PhD's.

Dr.R. Anandan

S. Phani Kumar is a research scholar in the Department of CSE, VELS Institute of Science, Technology & Advanced Studies. He completed his Masters in Web Technologies from JNTUK in 2010. He is currently working as an Assistant Professor in the Department of CSE, PACE Institute of Technology & Sciences. He developed many applications for educational institutions and he received Outstanding Faculty award in 2018. His research interests Big Data Analytics, Cloud Computing and Web Technologies.

TABLE OF CONTENTS

UNIT I

1. FUNDAMENTALS

Intelligent Agents - Agents and Environment - Good Behavior - The Nature of Environments - Structure of Agents - Problem Solving - Problem Solving Agents - Example Problems - Searching for Solutions - Uniformed Search Strategies - Avoiding Repeated States - Searching with Partial Information

1.1. Introduction to AI

Artificial Intelligence (AI)

AI is that the learning of the way to create machines performs actions that, at the instant, human perform best.

It leads **4** important levels.

a) Machines that feel similar to peoples.
b) Machines that perform similar to peoples.
c) Machines that feel logically.
d) Machines that perform logically.

Performing Humanly: The Turing Assessment Approach

To conduct this assessment, we want 2 individuals and also the machine to be evaluated. One human being plays the responsibility of the questioner, who is in an exceedingly separate area from the compute and also the different human being. The questioner will raise queries of either the human being or the computer by writing queries and receiving written responses. However, the questioner is aware of them just as A and B and targets to work out that the human being is and that are the machine. The target of the machine is to fool the questioner into believing that's the human being. If the machine achieve at this, in that case we will complete that the machine is performing humanly. However program a machine to give the assessment gives lots to perform on, to posses the subsequent abilities

- **Natural Language Processing:** is to permit it to interact well in English.
- **Automated Reasoning:** is to apply the saved data to reply queries and to get latest conclusions.
- **Knowledge Representation:** is to save data supplied before or for the duration of the investigation.

- **Machine Learning:** is to undertake to latest assets and to study from practice, example etc.,

Total Turing Assessment (TTA): The assessment which contains a video signal in order that the investigator should assess the emotional capabilities of the machine. To go through the TTA, the machine requires:

- **Computer Vision:** is to recognize entities
- **Robotics:** to shift them.

Sensible Humanly: The Cognitive-modeling Method

To build a machine application to assume similar to a people, initial it needs the information regarding the particular workings of human mind.

Once finishing the study regarding human mind it is potential to precise the speculation as a machine application.

If the application‘s ip/op and temporal order attitude equals with the people attitude then we will state that the program‘s method is functioning like a human mind.

Example: General Problem Solver (GPS)

Thinking Rationally: The Laws of Thinking Approach

The correct thinking introduced the idea of logic.

Example: Phani is a student of 3rd year CSIT.

All students are excellent in 3rd year in CSE. Phani is an excellent student.

Performing Rationally: The Rational Agent Method

Performing Rationally denotes, to attain one‘s goal specified one‘s idea. Within the earlier topic laws of thought approach, accurate reasoning is chosen, conclusion is derived, but the agent acts on the conclusion defined the task of acting rationally. The learning of rational agent has **2** advantages:

1. Correct reasoning is chosen and applied.
2. It focuses on scientific improvement instead of different strategies.

The History of AI

1943:	Pitts & McCulloch: Boolean circuit of design of mind
1950:	Turing‘sassessing machines and brilliance
1950s:	Near the beginning AI applications, which includes Samuel‘s-checkers application, Gelernter‘s Geometry Engine, Simon & Newell‘s Logic Theorist
1956:	Dartmouth summit: AI accepted
1965:	Robinson‘s entire set of rules for logical analysis
1960 – 74:	AI find out computational difficulty Neural network studies nearly disappears
1969 – 79:	Near the beginning improvement of knowledge based methods
1980 – 88:	Specialist systems enterprise boom
1988 – 93:	Specialist systems enterprise busts: AI winter
1985 – 95:	Neural Networks come back to reputation
1988:	Reappearance of probability; common enhance in technical path Nouvelle AI; GA‘s, A Life, and soft computing
1995:	Agents
2003:	Human being level AI rear on the plan

1.2. Intelligent Agents

1.3. Environments and Agents

An **agent** is some thing that should be regarded as appearing up on that environment through **actuators** and recognizing its **environment** over **sensors** given away in Figure.

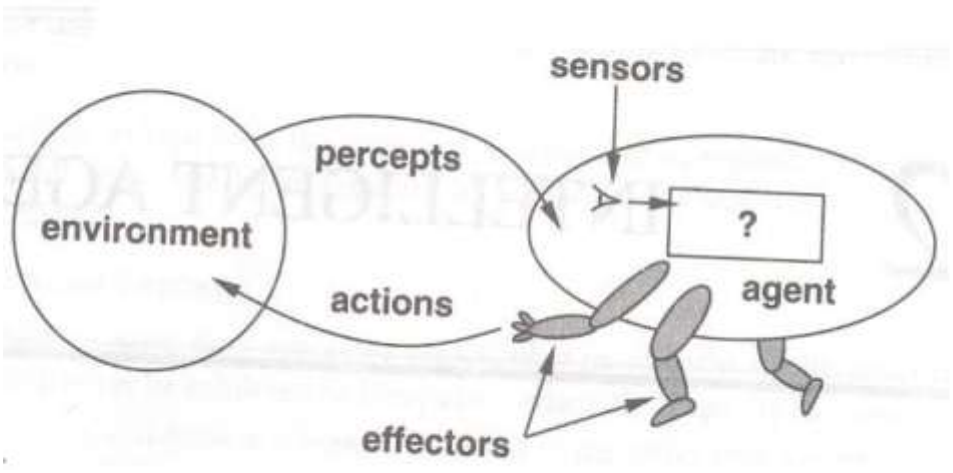


Figure: Agents connect with environment through effectors and sensors

The different categories of agent are:

- **Robotic Agent:** This agent contains infrared and cameras variety selectors for the sensors and different machines for the selectors.
- **Human Agent**: This agent contains ears, eyes and additional organs for sensors and mouth, hands, legs and additional human parts for selectors.
- **Generic Agent:** A preferred shape of an agent who connect with the environment.
- **Software Agent:** This agent contains programmed bit series as its rules and procedures.

The agent work for an agent indicates the activity extract by the agent in reaction to any rule order. It is a hypothetic mathematical explanation.

Inside, the agent work for an artificial-agent could be applied by using an agent application. An agent application is a task, which appliance the agent mapping from rules to activities. It is an actual implementation operation on the agent structural design.

1.4. Excellent Behavior: The Idea of Rationality

An agent has to act as a Rational Agent. This agent is one which performs the proper thing that is the proper actions will cause the agent to be mainly doing well within the environment.

Performance Measures

A performance measures represents the principle for achievement of an agent‘s behavior. As a standard regulation, it is improved to plan performance-measures in line with what one really desires within the environment, quite as per to how one imagines the agent could behave.

Rationality

What is a rational at all specified instance depends on 4 items:

- The **performance-measure** that describes the principle of **fulfillment**.
- The agent‘s earlier **awareness** of the environment.
- The **movements,** which the agent should execute.
- The agent‘s rule **collection** to date.

This directs to a description of a rational-agent (perfect rational agent)

- *For all potential rule series, a rational-agent need to choose an activity, which is anticipated to boost its presentation measure, specified the proof furnished by way of the rule series and anything fixed expertise the agent contain, that is the task of rational agent is to improve the performance measure depends on percept sequence.*

Omniscience, Autonomy, and Learning

An omniscient agent identifies the real final results of its performances and may work appropriately; but all-knowing is not possible actually.

A rational agent moreover to collect data, but also to study as a great deal as viable from what it perceives. The agent‘s preliminary arrangement should return some previous awareness of the environment, but because the agent gain knowledge this will be changed and improved.

Successful agents break up the function of measures the agent task into 3 dissimilar phases: while the agent is planned, a few of the calculation is achieved via its developers; while it is considering into account on its subsequent work, the agent performs higher working out; and as it study from familiarity, it performs higher working out to determine how to change its performance.

A rational-agent shall be independent – it could study what it is able to catch up on incomplete or wrong earlier information.

1.5. The Nature of Environments

Stating the Work Environment

The work environment specification consists of the outside environment, the presentation measure, the sensors, and the actuators.

In planning an agent, the primary step need to forever be to state the work environment as entirely as doable. Work environments are distinctive as a PEAS (Performance, Environment, Actuators, Sensors) or PAGE (Percept, Action, Goal, Environment) description, both methods are same.

The following table describes some of the agent types and the basic PEAS description.

Table: Patterns of Agent Categories and their PEAS Explanation

Agent category	**Actuators**	**Environment**	**Sensors**	**Performance computation**
Taxi-Driver	Accelerator, steering, signal, brake, display, horn	Roads and Further traffic, person on foot, clients	Speedometer, GPS, cameras, sonar, odometer, engine sensors, accelerometer, keyboard	Fast, legal, safe, comfortable trip, maximize profits
Health diagnosis system	Show queries, checkup, diagnoses, Recommendation, treatments	Staff, Hospital, patient,	Keyboard enter the symptoms, results, patient's responds	Lawsuits, Well patient, minimize costs
Satellite picture investigation method	Show group of picture	Down link from tracking satellite	Color pixel arrangements	Exact picture group
Refinery controller	Pumps, heaters, valves, displays	Operators, refinery,	Pressure, temperature, chemical sensors	Yield, maximize purity, safety
Part picking robot	Hand and linked arm	Bins, Conveyor belt with parts;	Camera, Joint angle sensors	proportion of parts in right bins
Interactive English trainer	Show work outs, Instructions, modifications	Assessing agency, group of students	Entry with Keyboard	Increase student's marks on assessment

Some **Software Robots** or **Software Agents** or **Soft bots** present in rich, and unlimited domains. So,

- Actions are done in real time.
- Action is performed in the complex environment.
- It is designed to scan online news origin and display the procedures with ordinary language processing.

Properties of Work Environments

1. Completely Recognizable vs. Partly Recognizable (or) Accessible vs. Inaccessible

If an agent's sensors deliver it gain to the whole condition of the environment at every end in the moment, next we state that the work environment is completely recognizable. Completely observable environments are suitable due to the fact the agent unneeded to continue any inside status to preserve path of the universe.

An environment is probably in partly observable because of inaccurate sensors and noisy or because elements of the condition are just lost from the sensor records.

2. Deterministic vs. Non-deterministic (or) Stochastic

If the subsequent condition of the environment is absolutely decided via the present condition and the operation achieved by way of the agent, next the environment is **deterministic**; or else, it is **stochastic.** In belief, an agent may not concern regarding doubt in a deterministic environment, **completely recognizable**. If the environment is **partly recognizable**, however, next it should arrive like stochastic.

3. Episodic vs. Non-episodic (or) Sequential

In an episodic environment, the agent‘s knowledge is split into atomic episodes. Each episode contains of its private percepts and activities and it does not rely on the preceding episode.

In sequential-environments, the current choice could control all expectations of decisions.

Eg: Taxi driving and Chess.

Episodic-environments are easier than sequential-environments for the reason that the agent could not required thinking in advance.

4. Dynamic vs. Static

If the environment is not changing for agent‘s action then the environment is static for that agent otherwise it is **dynamic**.

If the environment does not modify for some instance, then it modifies due to agent‘s action is called **semi-dynamic environment.**

E.g: Taxi driving is **dynamic. Chess** is **semi dynamic**. **Crossword puzzles** are **static**.

5. Continuous Vs. Discrete

If the environment has clearly defined percepts, finite number of distinct, and actions then the environment is **discrete**. E.g.: **Chess**

If the environment varies continuously with variety of value the environment is **continuous**. E.g: **Taxi driving.**

6. Single Agent vs. Multi Agent

The following Table indexes the properties of a count of known environments.

Task Environment	**Episodic**	**Deterministic**	**Agents**	**Observable**	**Discrete**	**Static**
Crossword puzzle	Sequential	Deterministic	Single	Fully	Discrete	Static
Taxi driving	Sequential	Stochastic	Multi	Partially	Continuous	Dynamic
Chess with a clock	Sequential	Strategic	Multi	Fully	Discrete	Semi
Medical diagnosis	Sequential	Stochastic	Single	Partially	Continuous	Dynamic
Backgammon	Sequential	Stochastic	Multi	Fully	Discrete	Static
Refinery controller	Sequential	Stochastic	Single	Partially	Continuous	Dynamic
Image analysis	Episodic	Deterministic	Single	Fully	Continuous	Semi
Part-picking robot	Episodic	Stochastic	Single	Partially	Continuous	Dynamic
Poker	Sequential	Strategic	Multi	Partially	Discrete	Static
Interactive English tutor	Sequential	Stochastic	Multi	Partially	Discrete	Dynamic

1.6. The Structure of Agents

An intelligent agent is a grouping of Architecture and Agent Program.

Intelligent Agent = Architecture + Agent Program

Agent Program is a task, which performs the agent mapping from rules to procedures. Their presence a range of primary agent program plan, reverse the type of records prepared explicit and utilized within the choice procedure. The proposes range in compactness, flexibility and efficiency. The suitable propose of the agent program relies upon the character of the environment.

Architecture is a computing tool utilized to run the agent program.

Four kinds of agent programs are there to execute the mapping assignment. They are:

1. Simple reflex agents
2. Utility based agents
3. Goal based agents
4. Model based reflex agents

We then clarify in common expressions **how to change** all these into **learning-agents**.

1. Simple Reflex Agents

The easiest kind of agent is the simple-reflex agent. It reacts directly to percepts i.E. those agent pick task on the source of the current rule, avoiding remaining rule record.

An agent explains in relation to how the situation – action policy permit the agent to build the connectivity from rule to operation.

Condition action rule: if condition then action

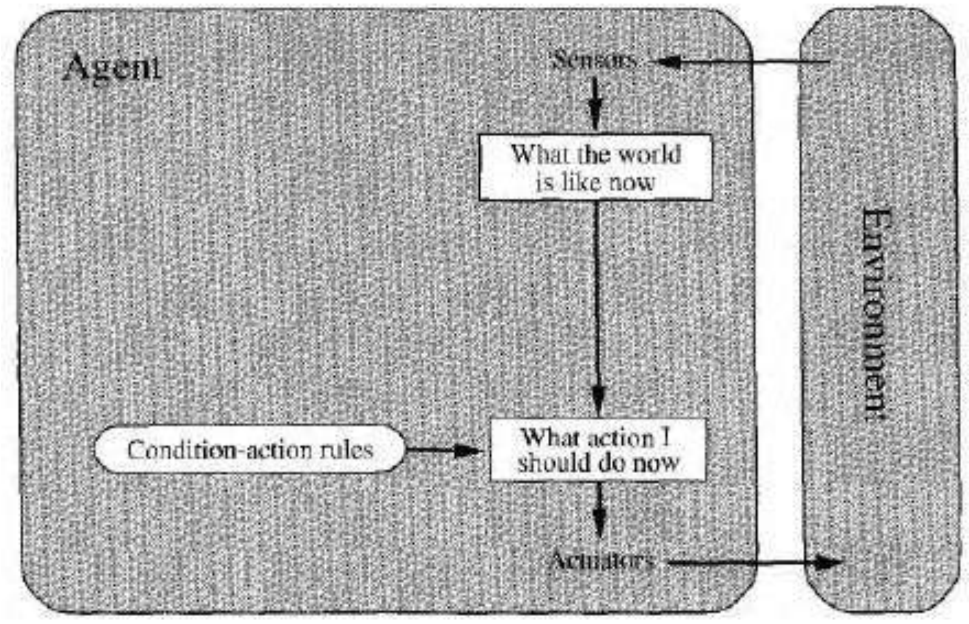


Figure: Schematic Figure of a Simple Reflex Agent

Diagram: specifies the formation of this common plan in schematic type, presenting how the state – activity rules permit the agent to build the link from rule to operation. In the above schematic figure, the shapes are described as:

Oval – to symbolize the background data in the procedure.

Rectangle – to indicate the present internal state of the agent's decision procedure.

The agent-program, that is also easy, is exposed in the following statements.

```
function SIMPLE REFLEX AGENT (percept) returns an action
    static: rules, a set of condition-action
    rules state ?   INTERPRET –
    INPUT(percept) rule ?   RULE –
    MATCH(state, rules)
    action ?   RULE – ACTION[rule]
    return action
```

INTERRUPT INPUT - Is the job of creates an absorbed explanation of the present status from the rule.

RULE MATCH - Is the job of returns the 1st regulation in the group of regulations that matching with the specified status explanation.

RULE ACTION - The preferred rule is performed as action of the specified percept.

The agent in diagram shall functions just -*if the right choice should be prepared on the source of just the present rule – which is, just if the environment is completely observable*.

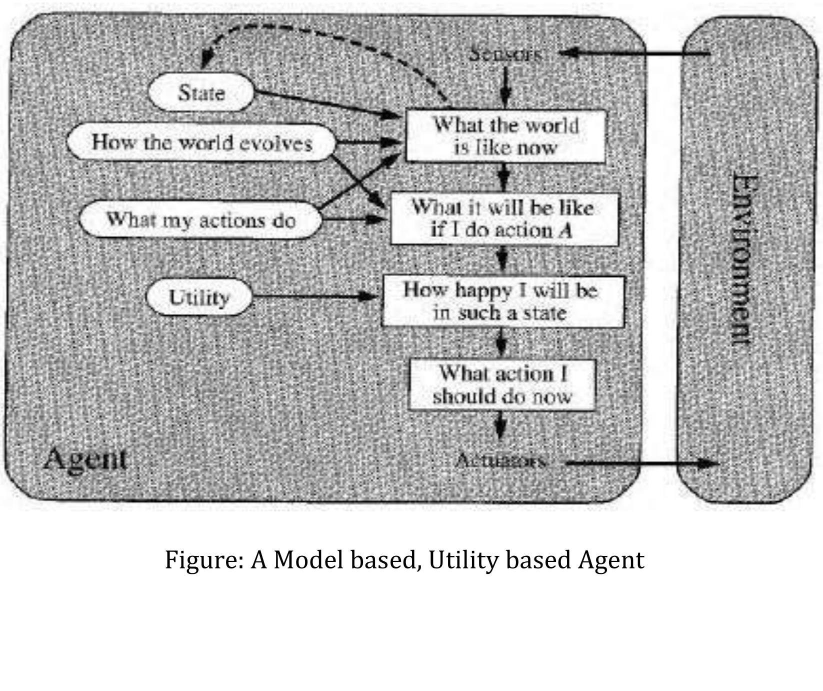
Example: Health diagnosis system

If the sufferer has reddish brown bad skin **then** begin the medication for measles.

2. Utility based Agents (Utility Mentions to the Quality of being Helpful)

An agent creates a target state with great quality manners (utility), if higher than one series contains to attain the target condition next the series with safer, greater reliable, faster and cheap than further to be selected.

A **utility function** maps a situation (or series of situations) onto an actual count, which explains the connected degree of gladness. The utility function can be utilized for two special cases: **First**, while there are incompatible targets, only a few of which should be attained (for example, safety and speed), the utility function denotes the precise transaction. **Second**, whilst the agent targets for several targets, nothing can be attained with certainty, then the achievement may be weighted up towards the significance of the goals.



Sensors
State
How the world evolves
What my actions do
Utility
What the world is like now
What it will be like if I do action A
How happy I will be in such a state
What action I should do now
Agent
Actuators
Environment

Figure: A Model based, Utility based Agent

Learning Agents

The **learning** function permits the agent to perform in primarily unfamiliar environments and to grow to be high qualified than its early awareness.

A **learning** agent should be separated into **4 theoretical elements**, as exposed in Figure.

Learning element - This is answerable for preparing developments. This makes use of the comments from the commentator on how the agent is performing and resolves how the action elements have to be changed to do well in forthcoming.

Performance element - which is answerable for opting external actions and it is equal to agent: this receives in rules and choose on operations.

Critic - It informs the learning-element how fine the agent is performing with specific to a permanent execution level.

Problem generator - This is accountable for signifying works to be able to lead to latest and informative knowledge.

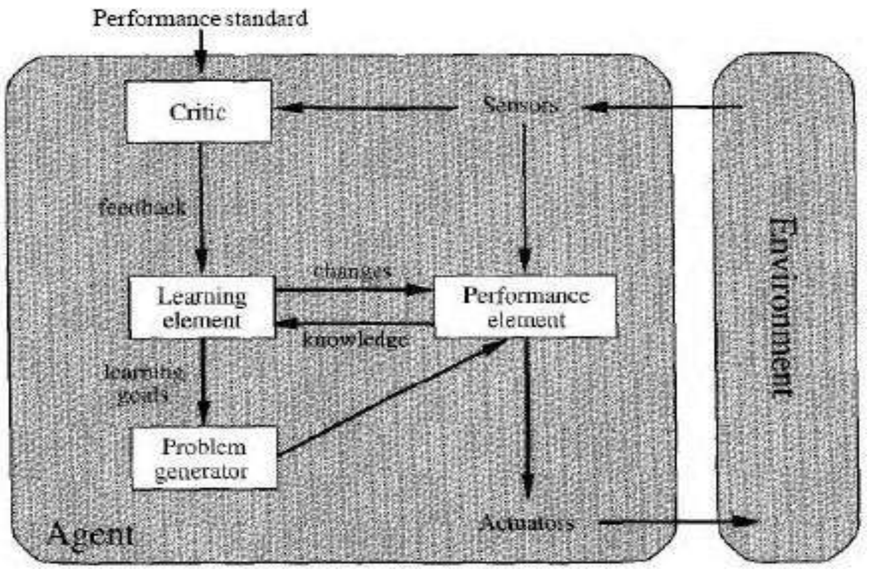


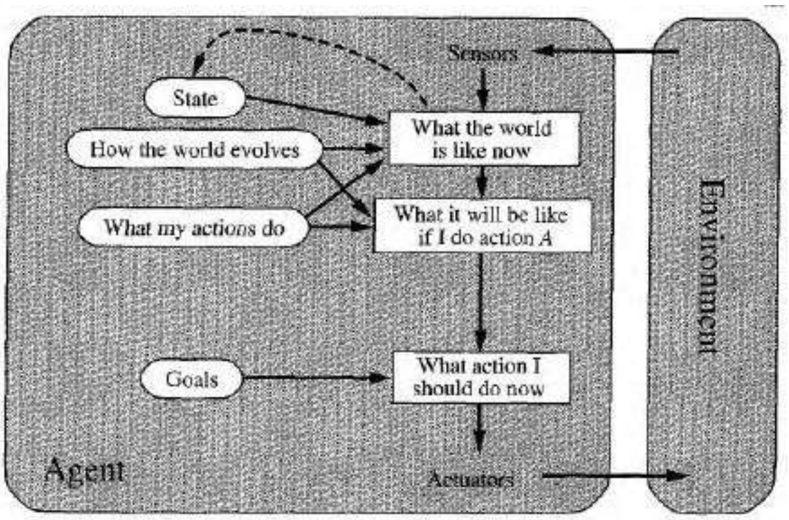
Figure: A Normal Version of Learning Agents

In review, agents contain a range of elements, and those may be presented in several methods inside the agent program, thus there seems to be high range between studying ways. **Learning** in intellectual agents might be outlining as an action of variation of all factors of the agent to carry the elements into nearer settlement with the existing elements record, there by developing on the whole action of the agent (All agents can enhance their overall performance through **learning**).

3. Goal based Agents

An agent recognizes the explanation of present status and also wishes some kind of **goal** records that illustrates conditions which might be desirable. The action equivalent with the present status is selected depend on the **goal** situation.

The goal based agent is high flexible for greater than one destination. After recognizing one destination, the latest destination is stated, goal based agent is started to provide with a latest behavior. **Planning** and **Search** are the sub-fields of AI dedicated to discovery work series that attain the agent‘s goals.



Sensors
State
How the world evolves
What my actions do
What the world is like now
What it will be like if I do action A
Goals
What action I should do now
Agent
Actuators
Environment

Figure: A Goal based, Model based Agents

The goal based agent arrives fewer competent, it is high adjustable for the reason that the information that supports its choice is signified clearly and should be changed. The goal based agent‘s manners should simply be modified to go to a dissimilar position.

4. Model based Reflex Agents

The only mode to deal with partly recognizability is for the agent "*to be track of the piece of the universe it cannot observe at present*". That is, the agent that mixes the current rule with the previous internal status to make updated explanation of the current status.

The current rule is mixed with the previous internal situation and it gains a latest current situation is up to date in the situation explanation is also. This updation needs **2 types of understanding** inside the agent program. **First**, we need a little knowledge regarding how the globe evolves separately of the agent. **Second**, we need a little knowledge regarding how the agent‘s personal actions have an effect on the world.

The above 2 information applied in easy Boolean-circuits or in entire methodical hypothesis is referred as a **model** of the universe. An agent, which makes use of this kind of model is called **model based-agent.**

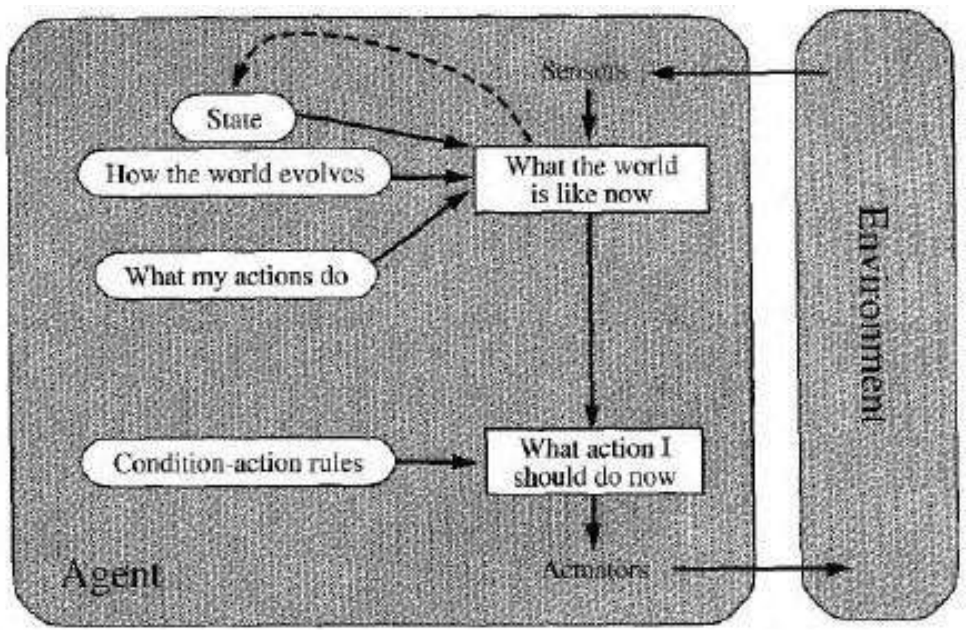


Figure: A Model-based Reflex-agent

The figure demonstrates the formation of the reflex-agent with internal condition, displaying how the present rule id jointed with the previous internal condition to produce the upgraded explanation of the present condition. The agent program is exposed in figure.

```
function REFLEX AGENT WITH STATE (percept) returns an action
    static: state, an explanation of the present world state
            rules, a set of condition action rules
            action, the latest action, primarily not

    state ?   UPDATE STATE(state, action, percept)
    rule ?   RULE MATCH(state, rules)
    action ?   RULE ACTION[rule]
return action
```

Figure: A model-based Reflex-agent

It remains catch of the present status of the globe by applying an internal design. It next selects an operation in the similar method as the reflex-agent.

UPDATE STATE - This is answerable for producing the latest internal state information by joining percept and present state information.

1.7. Solving Problems by Searching

Problem Solving Agents

It is one form of goal-based agent, in which the agent chooses what perform by selecting series of operations that result in ideal states. If the agent known the meaning of problem, it's quite simple to create a search procedure for discovering solutions, which recommends that problem solving agent need to be an intelligent agent to increase the overall performance measure.

The series of steps performed through the intelligent agent to increase the performance measure:

1. **Goal formulation -** primarily based on the agent‘s execution assess and the present state, is the initial step of problem-solving.
2. **Problem formulation** – it is the procedure of identifying what states and measures to deal, specified a goal.
3. **Search -** An agent with numerous quick alternatives of unfamiliar value should determine what to perform by using initial investigating dissimilar possible series of operations, which guide to states of identified value, and next selecting in fine series. This procedure of looking for this sort of series is known as search.
4. **Solution -** Search algorithm gets a problem as entry and arrival a result in the structure of an operation series.
5. **Execution phase -** formerly a result is found, the activities it suggests can be passed out.

The figure exposed as an easy problem-solving agent. It initially defines a problem and goal, inspect for a series of operations that universe resolve the problem, after which performs the operations individually at a time. When that is finished, it prepares one more startsover and goal.

```
function SIMPLE PROBLEM SOLVING AGENT (percept) returns an action
  inputs: per, a percept
  static: ser, an action series, primarily empty
        stat, some information of the present world state
        gl, a goal, primarily null
        prob, a problem design

  stat ?   UPDATE STATE (state, percept)
  if ser is empty then do
        gl ?   FORMULATE GOAL(stat)
        prob ?   FORMULATE PROBLEM(stat, gl)
         ser ?   SEARCH(prob)
  action ?
  FIRST(ser) ser ?
  REST(ser)
  return action
```

Figure: An Easy Problem Solving Agent

Note: First - 1st action in the series

Rest - returns the rest

Search - selecting the top one from the series of actions

Formulate problem - series of actions and situation that guide to target state

Update state - 1st state is enforced to subsequent state to reach the target state.

Well Described Problems and Solutions

A Problem should be described properly by four elements:

- P**rimary state** is the agent begins.
- Information of the available **actions (act)** offered to the agent. The general formulation utilizes a **successor (success) function**. Specified a specific state '**x**', **SUCCESSOR_FN (x)** gives a group of **< act, success >** structured pairs, accessible from **x** by any one action.

- The preliminary state and successor_function absolutely outline the **state-space** of the problem the group of each state available from the primary state. The state-space figures a diagram in which the joints are states and the arch among the joints and operations. A **path** inside the state area is a series of states joined by way of chain of operations.
- The **goal test** that regulates regardless if a specified state is a goal-state. If greater than one goal state be present, then we can test whether anyone of the goal-state is arrived at or not.
- A **path cost** utility, which allocates a numerical cost to every path. The cost of a path should be explained as the addition of the costs of all specific operations beside the path. The **step-cost** of getting action(*act)* to move from '*x'* to state '*y'* is indicated by *c(x, act, y).*

The previous elements outline a problem should be collected jointly into a separate records formation that is taken as entry to a problem-solving algorithm. A **result** to a problem is a path from the primary state to a goal state. The result value is calculated by the path-cost operation, and an **optimal-solution** contains the minimum path-cost amongst each results.

Formulating Problems

We derive a formulation of the problem in provisos of the **primary state**, **successor-function**, **goal-test**, and **path-cost**.

The procedure of eliminating feature from a depiction is known as **abstraction**. In extension to abstracting the state information, we should abstract the operations by itself.

data type PROBLEM

Components: Primary State, Successor Function, Goal Test, Path Cost.

1.8. Example Problems

Toy Problems

i. The Eight-puzzle Problem

The eight-puzzle contains of a 3×3 board with 8 valued tiles and an empty space as displayed in figure 3.2. A tile next to the empty space will move into the space. The entity is to arrive at a described target state. The principle formulation is:

- **States**: State information indicates the empty in 1 of the 9 boxes and the position of all of the 8 tiles.

- **Primary state**: Every state may be selected as the primary state.
- **Successor function**: It produces the permissible states that outcome from attempting the 4 operations (empty shifts Down, Up, Right and Left).
- **Goal test**: It tests regardless if the state fits the goal arrangement.
- **Path cost**: Every step costs one, so the steps count in the path is the length of the path (path-cost).

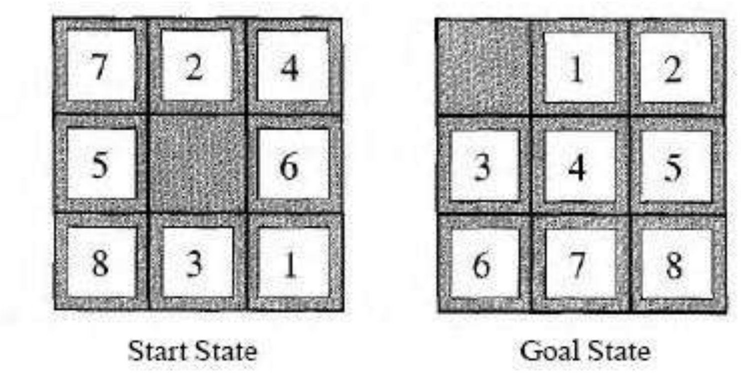


Figure: A Distinctive Instance of the Eight – puzzle

ii. The 8 – Queen's Problem

The purpose of the 8 – queen's-problem is to put 8-queens on a chess-board such that any queen does not attacks further in the same diagonal, column or row exposed in figure. There are **2 major types of formulation**. An **incremental_formulation** associates operative, which increase the state information, opening with a blank state; for the eight – queen's problem, this denotes that every operation include a queen to the state.

The incremental formulation is as follows:

- **States**: Any adjustment of zero to eight-queens on the board is a state.
- **Primary state**: The board does not contain any queen.
- **Successor function**: Any blank square inserted by a queen.
- **Goal test**: Eight-queens are at the board, no one attacked.

In this, we contain 3 × 1014 probable series to examine.

A **whole state formulation** begins with every eight queens at the board and shifts them on all sides. In each case, the path-cost is not important because just the number of last states.

A best formulation could disallow insertion a queen in any squared box, which is previously attacked:

- **States**: Adjustment of n-queens (0≤n≤8), one consistent with column within the left-most ‘n’ columns, and no queen attacking a different state.
- **Successor function**: Insert a queen to every squared box inside the left-most blank column such that it isn’t always attacked by means of further queen.

This formulation decreases the eight-queen‘s state area from 3 ×1014 to simply 2,057 and results are simple to find.

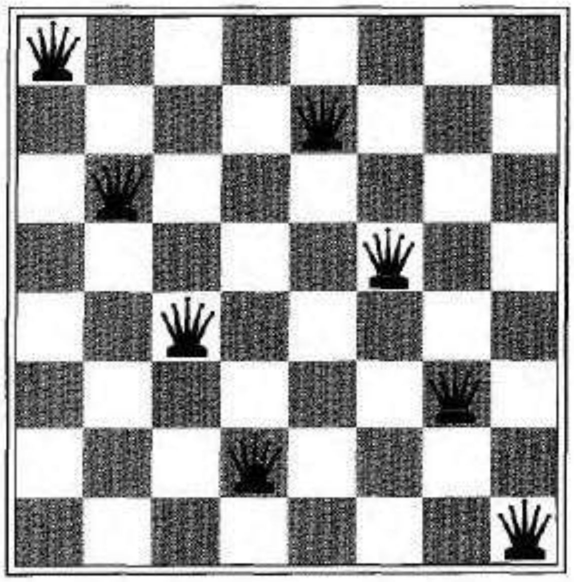


Figure: The 8 – Queen’s Problem

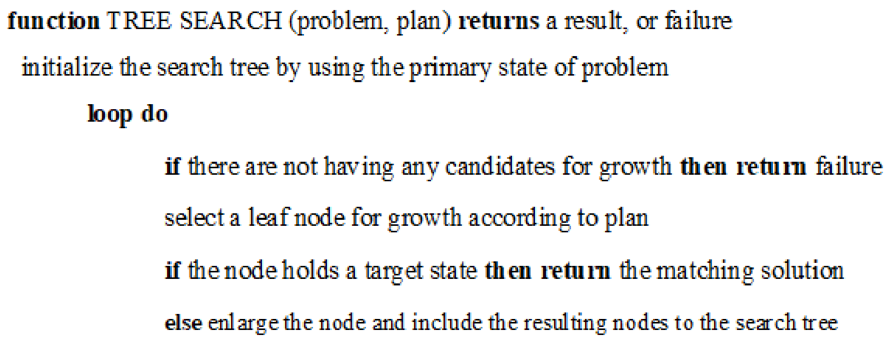
Real-world Problems

Name of the problems	**Applications**
i) Route finding	Airline journey planning system, armed operations preparation and Routing in networks.
ii) Touring and traveling salesperson problem	Shortest path tour.
iii) VLSI layout	Cell layout, channel routing
iv) Robot navigation	Route finding problem in continuous space.
v) Automated assembly ordering	Assembly of composite objects by Robot. E.g. Electric motors.

1.9. Searching for Solutions

We need to solve the formulated problems are completed by a search method throughout the state space that utilize a **search-tree** that is produced by the primary state and the successor-function that jointly describe the state-space. The root of the search-tree is a **search-node** matching to the primary state.

We keep on selecting, checking, and increasing upto either a result is observed or no states to be extended. The option of which state to develop is set by using the **search-plan**. The common search-tree algorithm is explained in the figure as follows:



```
function TREE SEARCH (problem, plan) returns a result, or failure
  initialize the search tree by using the primary state of problem
      loop do
            if there are not having any candidates for growth then return failure
            select a leaf node for growth according to plan
            if the node holds a target state then return the matching solution
            else enlarge the node and include the resulting nodes to the search tree
```

Figure: Explanation of the general search-tree algorithm

The nodes in the search tree are defined using **five components** in data structure. They are

1. **STATE**: the state location to whichever the node is correlate;
2. **PARENT NODE**: the node in the search-tree that produced another node;
3. **ACTION**: the action, which is executed to the root to produce the node;
4. **PATH COST**: the cost, generally indicated by g (n) of the path from the primary state to the node, as denoted by the parent indicator.
5. **DEPTH**: the steps count next to the path from the primary state.

The **difference among** states and nodes. A **state** correlates to the design of the world. A **node** is recording records structure applied to signify the search-tree.

To signify the group of nodes which were produced but till now not extended – this grouping is known as **fringe**. Every component of the fringe is a **leaf-node**, that is, a node without successors inside the tree. The demonstration of the fringe could be a group of nodes. The search method next could be a task that chooses the upcoming node to be accelerated from this group. It can be computationally costly, because the approach task may have to take a view at each component of the group to select the better one. Otherwise, the grouping of nodes is executed as a **queue** representation. The **queue actions** are:

- **MAKE QUEUE** (element…) – makes a queue with the specified elements.
- **EMPTY ?**(queue) – if no elements are in the queue, then it returns true.
- **FIRST**(queue) – it returns the initial element in the queue.
- **REMOVE_FIRST**(queue) – it returns initial element and eliminates that element from the queue.

- **INSERT**(element, queue) – adds an element within the queue and the resultant queue.
- **INSERT_ALL**(elements, queue) – adds a group of elements within the queue and returns the resultant queue.

The formal method of the common tree search algorithm is exposed in the bellow figure:

```
function TREE-SEARCH(problem, fringe) returns a result, or failure
    fringe ← INSERT(MAKE NODE (INITIAL STATE [problem] ), fringe )
    loop do
        if EMPTY ? ( fringe ) then return failure
        node ← REMOVE FIRST ( fringe )
        if GOAL TEST [ problem ] tested to STATE [ node ] achieves
            then return SOLUTION ( node )
        fringe ← INSERT ALL ( EXPAND ( node, problem ), fringe )


function EXPAND ( node, problem ) returns a set of nodes
    successors ←  the unfilled set
    foreach < action, result > in SUCCESSOR-FN [ problem ] ( STATE [ node ] )
    do
        s ←  a fresh NODE
        STATE[s] ← result
        PARENT NODE[s] ←
        node ACTION[s] ←
        action
        PATH COST[s] ← PATH COST [ node ] + STEP COST ( node, action, s )
        DEPTH[s] ← DEPTH [node] + 1
        inserts to successors
    return successors
```

Figure: The General Tree Search Algorithm

Evaluating Problem Solving Performance

The result of a problem_solving is any of fail or a result. We can assess an algorithm's actions in **4** methods:

- **Optimality**: If more than one way exists to derive the solution then the best one is selected.
- **Completeness**: The method definitely to discover a result when there is one.
- **Time complexity**: The Time obtained to run a result.
- **Space complexity**: Memory required executing the search.

In Artificial Intelligence, wherein the chart is pictured completely by means of the primary state and successor_function and is often endless, **difficulty** is disclosed in terms of 3 measures '**m**' is the highest length of any path in the state-space; '**d**' is the depth of the slight target node; and **b** is the branching component.

Definition of branching factor (b): The nodes count that is connected to every node in the search-tree. It is used to find space and time complexity of the search strategy.

1.10. Search Strategies

The Search Strategies are classified into **Informed search** (or) **Heuristic search** and **Uninformed search** (or) **Blind search.**

Informed Search vs. Uninformed Search

S. No	**Informed search(Heuristic search)**	**Uninformed search(Blind search)**
1.	The path-cost from the present state to target state is measured, to choose the least path-cost as the further state.	Path-cost from the present state to target state (or) no information regarding the steps count.
2.	Less effective in search method	More effective
3.	Problem to be clarified with the specified information	Additional information can be added as assumption to solve the problem
4.	E.g. 1. Depth limited search 2. Breadth first search 3. Uniform cost search 4. Depth first search 5. Interactive deepening search 6. Bi-directional search	E.g. a) Best first search b) Greedy search c) A* search

Breadth First Search (BFS)

BFS is an easy method wherein the root node is improved initially; next every incomers of the root node are extended further, afterward their incomers, and so on. Commonly, every node is extended at a specified depth within the search-tree previous to all nodes at the further level are extended.

Simple calling TREE_SEARCH (problem, FIFO-QUEUE ( )) outcomes in a breadth first search. The bellow figure displays the movement of the searching on a simple-binary tree.

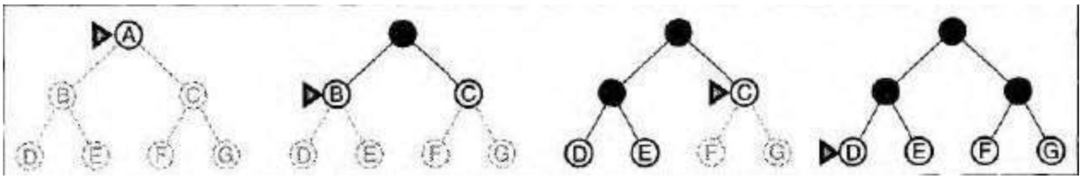


Figure: BFS on Simple-binary-tree

Time complexity is the total generated
nodes count is

b1+b2+b3+……+bd+bd+1−b= O (bd +1)

Each node that is created should present in storage. The space complexity is
same as the time complexity.

Advantages: Guaranteed to discover the single result at the shallowest depth point.

Disadvantages

1. The memory compulsion is a larger problem for BFS than the implementation moment.
2. Exponential complexity search problems can be unsolved by unaware systems for at all but only suitable for minimum instances problem (i.e.) (number of levels to be minimum (or) branching factor to be minimum)

Depth First Search (DFS)

It for all time extends the indepth node in the present fringe of the search-tree. If deadend take place, back tracking will be performed to the further quick before node for nodes to be extended.

Advantages: If greater than one result contains (or) levels count is more, next DFS is better because searching will be performed just in a little section of the total space.

Disadvantages: There is no guaranty to discover the solution.

Uniform - Cost Search (UCS)

The root node is extended initially, and then the next node also to be extended is chosen as the minimum cost-node on the fringe alternatively the minimum depth node i.e.

g(n) =path cost, breadth first search is equal to UCS when

g (n) =DEPTH (n).

Advantages: Guaranteed to discover the single result at lowest path cost.

Disadvantages: Only appropriate for minimum instances problem.

Depth Limited Search (DLS)

The boundless trees problem would be improved by giving DFS with decided depth limit '**L**'. i.e., at depth **L** nodes are considered, if they won't contain successors. This strategy is known as **depth limited search (DLS)**. The highest level of depth depends upon the states count.

Bidirectional Search (BDS)

It is an approach that it explores both the ways simultaneously i.e. frontward from the primary state and toward the back from the target, and ends when the 2 explores convene within the central point. It is applied by containing one or together of the explores verify every node earlier than it is extended to observe, if it is in the fringe of the order search-tree; if so, a result had been discovered.

Interactive Deepening Search (IDS)

It is a common approach that avoids the issue of selecting the finest depth limitation by attempting all available depth limitations. It joins the advantages of DFS and BFS.

1.11. Avoiding Repeated States

The repeated states can be avoided using **three** different ways. They are:

1. Does not return to the state we just approached from i.e. avoid any incomer, which is similar state of the node's parent.
2. Does not generate path with rotations i.e. sidestep any incomer of a node, which is similar of the node's family.
3. Does not create any state, which was continually created in advance.

1.12. Searching with Partial Information

When the information of the states or actions is unfinished about the environment, then only partial information is known to the agent. This incompleteness directs to three different problem categories:

1. **Sensor less problems** (**important problems**): If the agent does not having sensors at all, next it would be so many available primary states, and every action could lead to so many available successor states.

E.g. the Vacuum world problem

2. **Exploration problems**: Whenever the operations of the environment and the states are unidentified, the agent can find them. It may be considered as an severe case on contingency problems.
3. **Contingency Problems**: If the location is incompletely recognizable or if the operations are unsure, next the agent‘s rules give the latest record after every operation. Every probable rule describes a possibility, which should be designed for. A problem is known as adversarial, if the ambiguity is occurred by the operations of some other agent.

Question Bank

Unit - I

Part - A

1. Describe AI.
2. Describe Agent and Agent Function.
3. What is Turing Test?
4. How to evaluate the performance of an agent?
5. Write Short notes on Autonomy.
6. Give the architecture of agent in an environment.
7. What is an agent type and write the corresponding PAGE description.
8. Describe ideal Rational Agent.
9. Explain Agent Program.
10. What is Mapping. Give with an Example.
11. Explain Ideal Mapping with an example.
12. Describe Environment Program.
13. Explain the Problem Solving Agent.
14. What is meant by operators?
15. Explain the four different types of problem with one example.
16. Define State Space.
17. Define Path.
18. Define Path Cost.
19. Explain the steps in simple problem solving technique.
20. Give example for Artificial Intelligence problems.
21. Give example for real world problems in Artificial Intelligence.
22. What is the Search Tree? Explain with an example.

23. Describe Frontier or Fringe.
24. Differentiate heuristic search with blind search.
25. Describe the following terms for eight-Puzzle Problem: States, Operator, Goal-Test and Path-Cost.
26. List the steps to evaluate the performance of search strategies.
27. Write an informal explanation for the general tree search algorithm.
28. Why the problem formulation has to follow the goal formulation?
29. Differentiate Depth-First-Search, Breadth-First-Search with Best-First-Search.
30. Describe branching factor.
31. Explain the use of Heuristic Functions?

Part - B

1. Describe and Solve the bellow Problems:

   a) Missionaries and Cannibals Problem b) Water Jug Problem
2. Describe and solve the specified cryptographic mathematical problem utilizing the bellow explanations:

   i) Operator ii) State iii) Goal Test iv) Solution Process v) Initial State
3. Explain in brief about the Environment Programs with an example.
4. Describe the Vacuum World Problem and sketch the related state set-space presentations.
5. Explain in brief regarding blind-search methods with a sample example.
6. Explain the bellow uninformed search approaches with examples.
   a) Uniform Cost Search
   b) Depth first Search
   c) Depth Limited Search
   d) Breadth First Search
   e) Bidirectional Search

UNIT II

2. SEARCHING TECHNIQUES

Informed Search and Exploration, Informed (Heuristic) Search Strategies, Heuristic Function, Local Search Algorithms and Optimistic Problems, Local Search in Continuous Spaces, Online Search Agents and Unknown Environments, Constraint Satisfaction Problems (CPS), Backtracking Search and Local Search for CSP, Structure of Problems, Adversarial Search, Games – Optimal Decisions in Games, Alpha-Beta Pruning, Imperfect Real Time Decision, Games that Include an Element of Chance

2.1. Heuristic (or) Informed Search Strategies

2.2. Heuristic Search (or) Informed Search

The path-cost from the present state to target state is computed, to choose the least path-cost for the further state.

Additional information can be added as assumption to solve the problem.

E.g.

a. Greedy search
b. Best first search
c. A*search

Best First Search (BFS)

BFS is an example of the common GRAPH_SEARCH or TREE_SEARCH algorithm in which a node is chosen for extension based on a valuation function "f (n)".

A Key element of these algorithms is a heuristic_function "h (n)"

h (n)=expected cost of the least-path from node, 'n' to a target node.

Two types of valuation functions are:

Extend the node nearby to the goal state using expected cost as the evaluation is known as **Greedy Best-First-Search**.

Extend node on the minimum cost solution path by using expected cost and actual cost as the evaluation function is called **A* search**.

A* search: reducing the total expected solution cost.

Extend the node on the minimum cost solution path by using expected cost and actual cost as the evaluation function is called **A*search**. It valuates nodes by joining the cost to arrive at the node, h(n), and g(n), the cost to obtain from the node to the target:

$$f(n) = h(n)+g(n).$$

h(n) is the expected cost of the least-path from n to the target and g(n) provides the path-cost from the begin node to n node, and, we contain

f (n)=expected cost of the least result throughout 'n'.

A*search is both complete and optimal.

Greedy Best First Search

It attempts to extend the node, which is nearby to target, on the positions this is similarly to direct to a result rapidly. Therefore, it valuates nodes applying the **heuristic_function:**

$$f(n) = h(n).$$

It look like DFS in a way it wish to track an individual path each of the way to the target, but can backup when it knocks a corner. It is **not complete** and also **not optimal**. The bad case **space-complexity** and **time** is 'O (bm)', here 'm' is the highest depth of the search-space. It may be compressed with best heuristic function.

Monotonicity (consistency): In search tree every path from the root, the f-cost not at all reduces. This state is true for approximately every permissible heuristics. A heuristic which satisfies the property is called monotonicity.

Optimality: It is derived with two approaches. They are a) A* used with Tree-search b) A* used with Graph-search.

Memory Bounded Heuristic Search

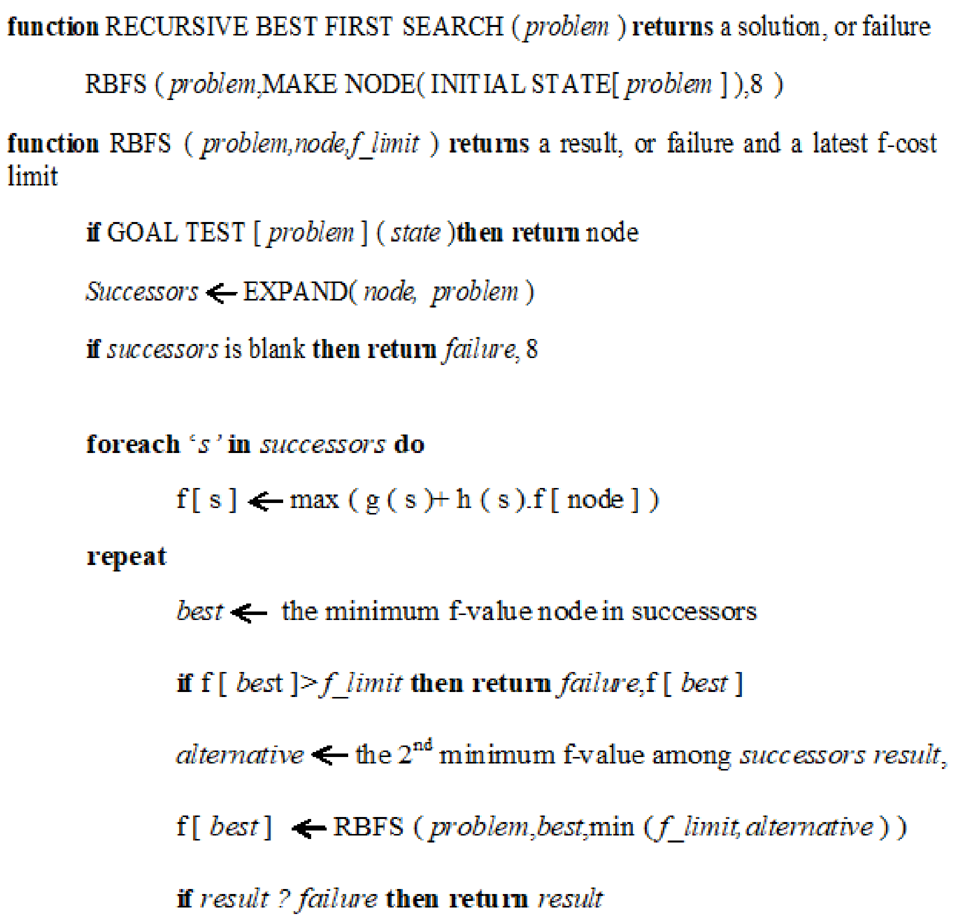
The memory necessities of A* is decreased by joining the heuristic function with iterative deepening resulting an "IDA* algorithm". The major dissimilarity between principle repetitive deepening and IDA* is the cut-off utilized is the fcost(g+h) alternatively the depth; at every repetition, the cut-off value is the minimum f-cost of each node, which go beyond the cut-off on the before repetition. The major difficulty is, it will require more storage space in complex domains.

The two recent memory bounded algorithms are:

1. Memory bounded A* search (MA*)
2. Recursive Best First Search(RBFS)

Recursive Best-first Search (RBFS)

RBFS is an easy recursive-algorithm utilizes only linear space. This algorithm is as follows:



```
function RECURSIVE BEST FIRST SEARCH (problem ) returns a solution, or failure
        RBFS (problem,MAKE NODE( INITIAL STATE[problem ] ),8 )
function RBFS ( problem,node,f_limit ) returns a result, or failure and a latest f-cost
limit
        if GOAL TEST [ problem ] ( state )then return node
        Successors ← EXPAND( node, problem )
        if successors is blank then return failure, 8

        foreach 's' in successors do
                f[ s ] ← max ( g ( s )+ h ( s ).f [ node ] )
        repeat
                best ← the minimum f-value node in successors
                if f [ best ]>f_limit then return failure,f [ best ]
                alternative ← the 2nd minimum f-value among successors result,
                f[ best ] ← RBFS ( problem,best,min (f_limit, alternative ) )
                if result ? failure then return result
```

Figure 2.1: RBFS Algorithm

Its **time complexity** depends on both the on how often the better path modifies as nodes are extended and correctness of the heuristic function. Its **space complexity** is O (bd), even though more memory is available.

Search techniques which use every accessible memory are:

1) SMA* (Simplified MA*)
2) MA* (Memory BundedA*)

SMA*

It will apply of all existing storage to reveal the search.

Properties: i) It sidesteps repetitive states as distant as its memory permit.

ii) It can use whatever memory is exists to it.

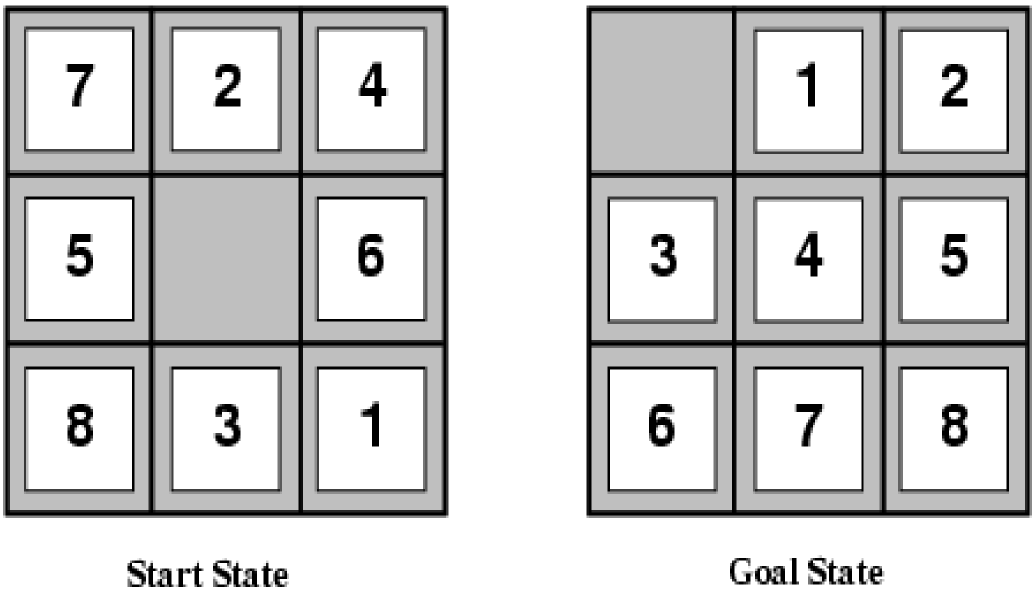
It is **complete,** if the existing storage is enough to backup the deepest resultant path.

It is **optimal,** if the sufficient storage is exists to backup the deepest resultant path or else, it comes back the better result that will be arrived at with the existing storage.

Advantage: SMA* uses only the available memory.

Disadvantage: If enough memory is not available it leads to suboptimal solution.

Time and Space complexity: depends on the available nodes count.



2.3. Heuristic Functions

The 8-puzzle Problem

Given:

Task: Discover the least solution by applying heuristic-function, which not at all approximate the steps count to the target.

Solution: To perform the given task two candidates are required, which are named as ‘h1’ and ‘h2’.

h1=the quantity of misplaced tiles. From figure, each of the 8-tiles are not in the location, so the begin state could contain h1 = 8. 'h1' is a permissible heuristic, for the reason that it is understandable, which every tile that is not in a position can be changed not less than one time.

h2 is the addition of the distances of the tiles from their target places is known as City Block Distance or Manhattan-distance:

$$h2=1+3+2+3+2+2+3+2=18$$

The result of heuristic correctness on performance:

Effectual Branching Factor (b)*

In the search tree, if the total nodes count extended by A * for an exact problem is 'N', the resultant depth is 'd', next 'b *' is the branching-factor, which is an identical tree of depth-d should have to contain in array to consist of N + 1 nodes. therefore

$$N+1=1+b^*+(b^*)2+.....+(b^*)d$$

For example: depth = 5; N = 52;

Effective branching factor = 1.91

Relaxed Problem

A Problem with lesser limitations on the operations is known as **relaxed problem**. A*n optimal result cost to be relaxed-problem is a permissible heuristic for the genuine problem*. If the specified problem is a relaxed-problem then it is available to create best heuristic function.

2.4. Optimization Problems and Local Search Algorithms

Local-search algorithms function by applying an individual present state and in general shift just to neighbors of that state. Local-search algorithms are unmethodical, they contains **2 key benefits**:

a) It utilizes small storage – regularly a stable quantity.

b) It may usually discover logical results in maximum or unlimited state-spaces for which methodical algorithms are not fitting.

To know the local-search, we can discover it extremely helpful to believe the state-space landscape as shown in the bellow figure.

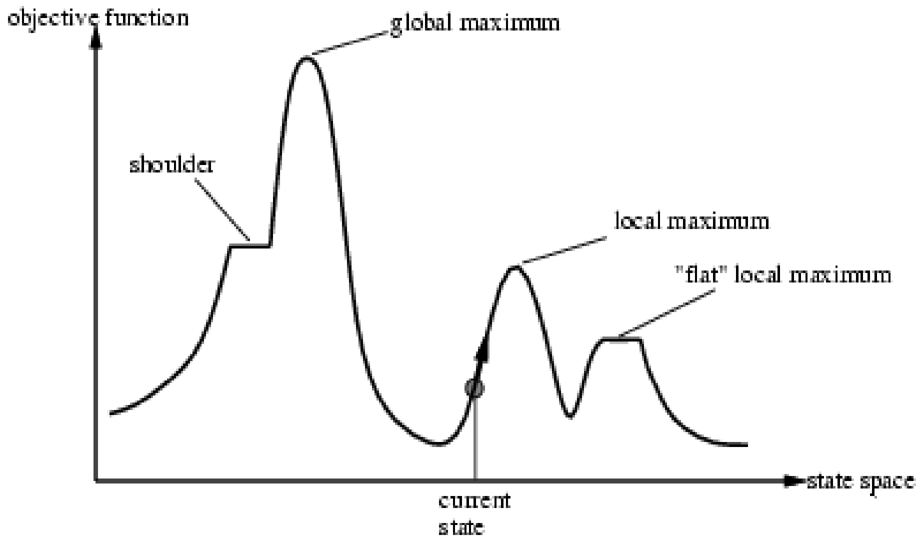


Figure: One-dimensional State Space Landscape

Applications

- Telecommunications network development
- Factory – floor layout
- Integrated –circuit plan
- Automatic programming
- Job-shop scheduling
- Vehicle routing

Advantages

- Constant search space. It is fit for online and also offline search.
- The search cost is low when matching to informed search strategies.

Some of the regional search algorithms are:

a) Genetic algorithm (GA)
b) Simulated annealing (SA)
c) Hill-climbing search (HCS)
d) Local beam search (LBS)

Hill-climbing Search (HCS)

A search technique that move in the direction of growing value to arrive at a peak state. It ends when it arrive at a dead-end where other neighbor does not have a maximum value. The Hill- climbing search algorithm as displaying in the bellow figure.

```
function HILL-CLIMBING-SEARCH ( problem ) returns a state that is local highest
    inputs:problem,a problem
    local variables:present,a node
                    neighbor,a node
    present?   MAKE –NODE ( INITIAL – STATE [ problem ] )
    loop do
        neighbor ?   a maximum - valued successor of present
        if VALUE [ neighbor ]=VALUE [ present ] then return STATE [ present ]
        present ?   neighbor
```

Figure: The Hill - climbing – Search (HCS) Algorithm

Drawbacks

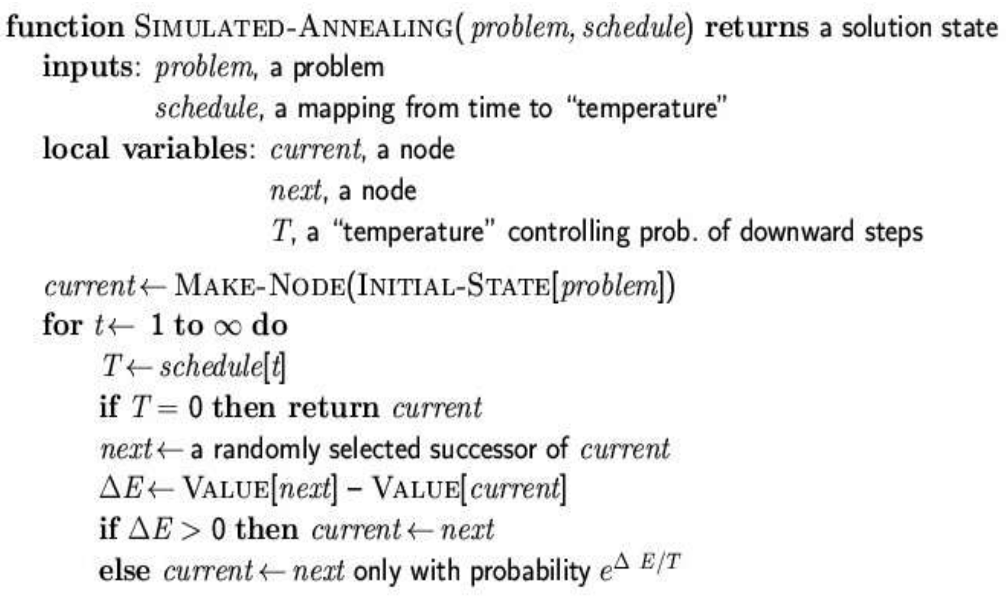
- **Local maxima** (Foot hills): a local maximum is a dead-end, which is maximum than every one of its neighboring states, but smaller than the overall highest.
- **Plateaux** (shoulder): It is a part of the state-space landscape wherever the valuation task is flat. This may be a flat-local highest.
- **Ridges**: a series of local maxima, which had a slope that gently moves to a peak.

Some of the variations of hill-climbing are:

- **First-choice hill-climbing**: applies statistic hill climbing by producing incomers at random up to one is produced, which is improved than the present state.
- **Stochastic-hill-climbing**: selects at random from between the uphill shifts.
- **Random-restart hill-climbing**: overcomes local-maximum slightly complete.

Simulated-Annealing Search

The algorithm which joins hill-climbing with unplanned walk to give up both completeness and effectiveness. This algorithm was developed using annealing process the procedure of slowly cooling a liquid up to it freezes.



```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
   inputs: problem, a problem
           schedule, a mapping from time to "temperature"
   local variables: current, a node
                    next, a node
                    T, a "temperature" controlling prob. of downward steps

   current ← MAKE-NODE(INITIAL-STATE[problem])
   for t ← 1 to ∞ do
      T ← schedule[t]
      if T = 0 then return current
      next ← a randomly selected successor of current
      ΔE ← VALUE[next] – VALUE[current]
      if ΔE > 0 then current ← next
      else current ← next only with probability $e^{\Delta E/T}$
```

Figure 2.4: The algorithm for simulated annealing search

Applications

- VLSI layout
- Airline scheduling

Local Beam Search (LBS)

The LBS algorithm remains tracking of k-states alternatively only a single state. It starts with 'k' unplanned produced states. At every step, overall incomers of all k-states are produced. If anyone is a target, the algorithm stops or else, it chooses the 'k' greatest incomers from the entire index and iterates. *In LBS, helpful data is travelled between the k related search-threads*.

This search should undergo from short of variety between k-states. Accordingly a variant labeled as statistic-beam search selects 'k' incomers at unplanned manner, with the possibility of selecting a specified incomer being a growing task of its value.

Genetic Algorithm (GA)

A GA is a variation of statistic beam-search wherein incomer states are produced by joining 2 parent-states, alternatively by changing a single state. The bellow figure explains a GA algorithm.

```
function GENETIC ALGORITHM ( population,FITNESS-FN ) returns an entity
    inputs:population, a set of entities
        FITNESS-FN,a function that evaluates the condition of an entity
    repeat
        new-population ← blank set
        loop for 'i' from one to SIZE ( population ) do
            x ← RANDOMSELECTION ( population, FITNESS - FN )
            y ←RANDOMSELECTION ( population,FITNESS – FN )
            child ← REPRODUCE ( x, y )
            if (small-random-probability ) then child ?   MUTATE ( child)


            add child to new-population
        population?   new-population
    until a few entity is fit sufficient, or sufficient time has expired
    return the greatest entity in population, according to FITNESS - FN


function REPRODUCE ( x, y ) returns an entity
    inputs: parent entities, x, y
    n?  LENGTH ( x )
    c?  random number from one to n
    return APPEND ( SUBSTRING ( x, 1, c ), SUBSTRING ( y, c+1, n ) )
```

Figure: The Genetic Algorithm

2.5. Local Search (LS) in Continuous Spaces

LS in continuous space are the one that deals with the real world problems.

- One method to resolve permanent problems is to distinct the nearer of every state.
- Simulated Annealing and Statistic hill-climbing are executed openly in the continuous space.
- Empirical gradient, line search, Newton-raphson method can be applied in this domain t find the successor state.

- It may also lead to local maxima, ridges and plateau. This situation is avoided by using random restart method.

2.6. Online Search Agents and Unknown Environments

Offline search: They calculate the whole result earlier configuration of the foot in the actual universe next apply the result with no remedy to their rules.

Online search: Agents operate by interlinking execution and activity: initially, it receives an operation next it monitors computes and the environment of the subsequent operation.

Online search is the best thought in **semi-dynamic** (or) **dynamic** domains and **statistic** domains.

Online search is a required idea for an **exploration-problem**, wherever the actions and states are not known to the agent.

Online Search Problems (OSP)

An OSP should be resolved just by an agent applying operations, alternately by a execution process. We can guess that the agent notices the bellow statements:

- The step_cost_function c(s,a,s‘) recognized the agent when it arrives at s‘.
- ACTIONS (s), whichever gives back a list of operations permitted in state ‘s’.
- GOAL-TEST(s).

Here the agent can’t access state successors, not including by trying all available actions in that state. This drawback of an agent can be avoided by:

- Possible to find Manhattan distance heuristic.
- Actions are deterministic.
- Visited states are noted to the agent.

E.g. A simple maze problem

Competitive ratio: This defines the comparison between the total path costs with the computationally shortest complete exploration path cost. This ratio, should be as small as possible for better results.

Online Search Agent

An online algorithm, increase only a node, which it physically fills. After each and every operation, an online-search agent collects a rule to identify the state. It had arrived and it may augment its map of the environment, to decide where to go next.

An online DFS agent is exposed in figure:

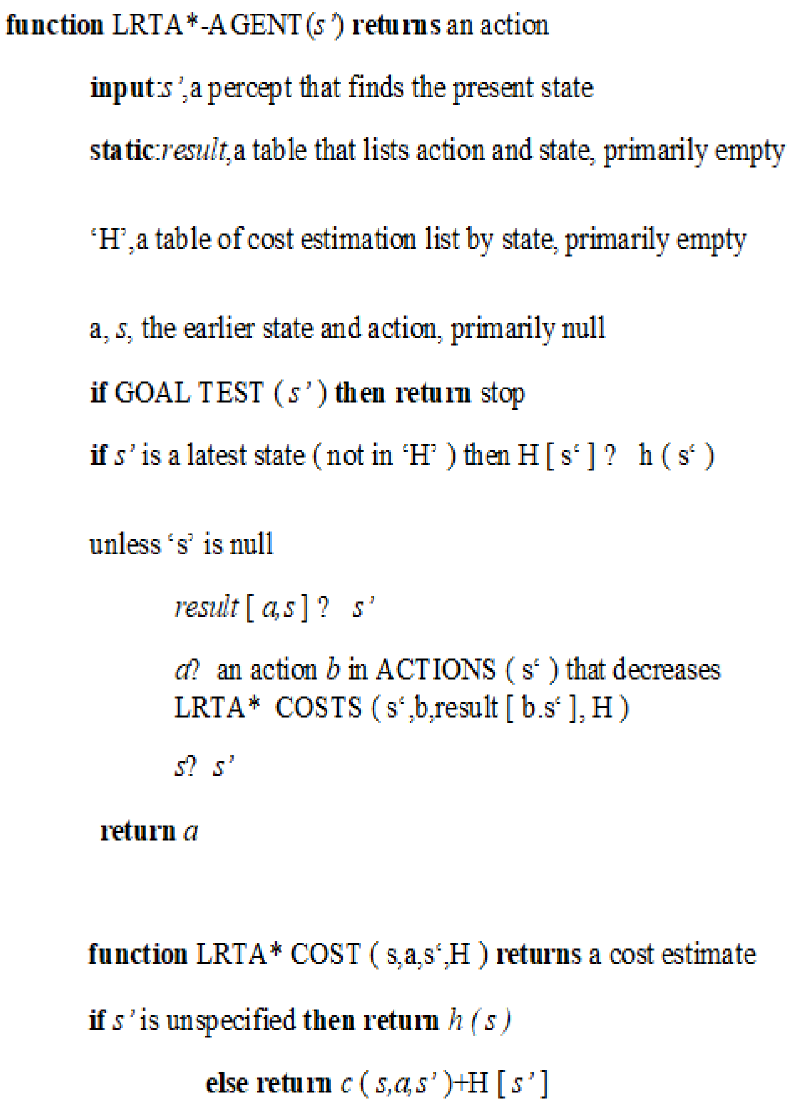
```
function ONLINE-DFS-AGENT ( s' ) returns an action
    inputs:s',a percept that recognizes the present state
    static:result, a table that lists state and action, primarily empty
        un-explored,a table that lists, each inspected state, the actions not yet attempte
        un-backtracked,a table that lists, each inspected state, the backtracks not yet
    attempted
        s, a,the earlier state and action, primarily null
    if GOAL-TEST ( s' ) then return stop
    if s' is a latest state then un-explored [ s' ] ?   ACTION ( s')
    if s is not null then do
        result [ a,s ] ?   s'
        insert 's ' to the front of un-backtracked [ s' ]
    if un-explored [ s' ] is blank then
        if un-backtracked [ s' ] is blank then return stop
        else a ?   an action b such that result [ b,s' ]=POP( un-backtracked [ s' ] )
    else a ?   POP (un-explored [ s' ] )
    s? s'
    return a
```

Figure: An Online Search Agent that Uses Depth First Exploration

Online Local Search (OLS)

Hill-climbing search technique can be utilized to perform online local search because it maintains one present state in storage. To avoid the drawback of local maxima, random walk is chosen to explore the environment instead of random restart method. The concept of hill climbing with data stores a present greatest guess H(s) of the cost to arrive at the target from all states, which have been visited is executed as Learning Real Time A*(LRTA *) algorithm.

The LRTA* agent algorithm is shown in figure:



```
function LRTA*-AGENT (s') returns an action
    input: s', a percept that finds the present state
    static: result, a table that lists action and state, primarily empty

    'H', a table of cost estimation list by state, primarily empty

    a, s, the earlier state and action, primarily null
    if GOAL TEST ( s' ) then return stop
    if s' is a latest state ( not in 'H' ) then H [ s' ] ?   h ( s' )

    unless 's' is null
        result [ a,s ] ?   s'
        a?   an action b in ACTIONS ( s' ) that decreases
        LRTA*  COSTS ( s',b,result [ b.s' ], H )
        s?  s'
     return a


    function LRTA* COST ( s,a,s',H ) returns a cost estimate
    if s' is unspecified then return h ( s )
          else return c ( s,a,s' )+H [ s' ]
```

Figure: LRTA*-Agent algorithm

2.7. Constraint Satisfaction Problems(CSP)

CSP is described by a group of **constraints**, C1, C2,....., Cm and a group of **variables**, X1,X2,X3,........, Xn. Every variable 'Xi' not having empty **domain** 'Di' of all available **values.** A whole task is a thing in which each variable is declared and a **result** to a CSP is a total task that gratifies each constraint. A few CSPs in addition need a result that increases an **objective function.**

Some examples of CSP's are:

- Then-queens problem
- A crossword problem
- A map coloring problem

Constraint graph: A CSP is generally shown as not a directed graph it is known as constraint graph wherever the corners are the binary-constraints and the nodes are the variables.

CSP should be considered as an additional formulation as an ideal inspection problem as the bellow points:

- **Primary state**: the blank set {}, in that no variables are assigned.
- **Successor function**: give an element to a not allocated variable, given that this is not dispute with earlier allocated variables.
- **Goal test**: the present set is completed.
- **Path cost**: an invariable cost for each step.

Discrete Variables

a) **Infinite domains**: For n variables with infinite domain size such as strings, integers etc. **E.g.** set of strings and set of integers.
b) **Finite domains:** For n number of variables with a limited size domain d, the complexity is O(dn). Entire assignment is available.

e.g. Map-coloring problems.

Continuous variables: Linear restrictions solvable in a polynomial point in time by linear programming. **E.g:** begin / end instants for Hubble-space telescope considerations.

Types of Constraints

a. **Unary constraints**: It limits the value of a variable.
b. **Binary constraints**: It associates with the pair of variables.
c. **Greater order constraints:** It involves more than or equal to three variables.

2.8. Backtracking Search for CSP's

Backtracking search, a form of DFS that selects values for only one variable individually and backtracks whenever a variable does not having permissible values to allocate. This algorithm is exposed in the bellow figure:

```
function BACKTRACKING_SEARCH ( csp ) returns a result, or failure
    return RECURSIVE_BACKTRACKING ( {},csp )
function RECURSIVE_BACKTRACKING ( assignmen, csp ) returns a result, or failure
    if assignment is completed then return assignment
    var?  SELECT_UNASSIGNED_VARIABLE ( VARIABLE [ csp ], assignment,csp )
    foreach value in ORDER_DOMAIN_VALUES ( var,assignment,csp ) do
    if value is reliable with assignment giving to CONSTRAINT [ csp ] then
            insert{ var = value } to assignment
            result?  RECURSIVE_BACKTRACKING( assignment,csp )
            if result? failure then return result
            delete{var = value } from assignment
    return failure
```

Figure: An Easy Backtracking Algorithm for CSP

Propagating Information through Constraints

Forward checking: The key steps of forward checking process are:

- Be track of left over permissible values for not assigned variables.
- End search when every variable does not having permissible values.

This method propagates the information from allocate to unallocated variables, but does not give early discovery for all non-success.

Constraint Propagation

Constraint propagation frequently implements constraints locally to notice inconsistencies. This propagation may be complete with dissimilar kinds of consistency methods.

1. **Node consistency**(one consistency): The node showing a variable 'V' in constraint-graph is node-consistent, if the value 'X' in the present domain of 'V', every single constraint on 'V' is fulfilled. The node inconsistency may be removed by eliminating the

values from the domain 'D' of every variable, which does not gratify single constraint on 'V'.

2. **Arc consistency** (two consistency): Arc specifies to a ordered arc in the control graph. The different versions of Arc-consistency algorithms are exist such as AC- 1, upto AC-7, but frequently used are AC-3 or AC-4.
3. **Path consistency** (K-consistency): The algorithm for creating a constraint-graph powerfully three constant that is regularly referred as path-consistency make sure that problem may be resolved without back tracking.

Handling Special Constraints

1. **Alldiff constraint**: All the variables concerned must have different values.
2. **Resource constraint**: Higher order or atmost constraint, in which consistency is achieved by removing the highest value of every domain, if it is inconsistent with least values of the further domains.

Intelligent Backtracking

Chronological backtracking: when a branch of the search be unsuccessful, backup to the previous variable and attempt a distinct value for it. At that point, the latest decision point is updated.

Conflict-directed back-jumping: It backtracks straight to the basis of the problem.

2.9. Local Search for CSP

Local search by means of the minimum variances heuristic have been tested to constraint fulfillment problems with big achievement. They apply a whole state formulation: the primary state allots a value to each variable and the successor-function regularly performs by modifying the value of 1 variable individually.

In selecting a latest value for a variable, the better noticeable heuristic is to choose the value that solution in the lower conflicts count with further variables, the **least conflicts** heuristic.

Minimum conflicts is surprisingly valuable for several CSPs, mainly when given a sensible primary state. Surprisingly, on the n-queens problem, if we do not calculate the number of primary assignment of queens, the continuation of minimum variances is approximately *autonomous of problem size*.

```
function MIN_CONFLICTS ( csp,max_steps ) returns a result or failure
    input:csp,a constraint gratification problem
        max_steps, the number of steps permitted before allowing up
    present an primary complete assignment for csp
    for i=1 to maximum steps do
        if present is a result for csp then return present
        var  'a' randomly selected, conflicted variable from VARIABLES [
        csp ] the value 'v'  for var that decreases CONFLICTS (
        var,v,present,csp )
        set var= value in present
    return failure
```

Figure: Minimum Conflicts Algorithm for Solving CSP

2.10. The Structure of Problems

The difficulty of solving CSP is powerfully correlated to the structure of its constraint-graph. If the CSP may be separated into independent sub-problems, then every sub-problem is resolved independently then the results are joined. When 'n' variables are divided as 'n/c' sub-problems, each will take dc operation to resolve. Therefore the whole operation is O(dcn /c).

Every tree structured CSP may be resolved in time limited in the variables count. The algorithm contains the bellow statements:

1. Select one variable as the origin of the tree and arrange the variables from the origin to the leafs just as all parent nodes in the tree predates it in the arranging label the variables X1,X2,…,Xn in order, all variables excluding the root has only 1 variable's parent.
2. For 'j' from one to n, allocate one value for 'Xj' consistent with the value allocated for 'Xi', wherever 'Xj' is the child of 'Xi'
3. For j from 'n' downward to two, use arc-consistency to the arc(Xi,Xj), wherever 'Xj' is the child of 'Xi', deleting values from DOMAIN[Xi] as essential.

The complete algorithm runs in time O (nd2).

General constraint graphs may be decreased to trees on 2 ways. They are:

1. Removing nodes – Cutest conditioning
2. Collapsing nodes together – Tree decomposition

2.11. Adversarial Search

Games

Game may be described by the primary state, the permissible operations in all states, a utility-function and a closing test that uses to closing states.

In game playing to select the next state, search technique is required. The **pruning technique** permits us to exclude location of the search-tree, which makes no dissimilarity to the last selection, and **heuristic-evaluation function** permit us to discover the utility of a state is not performing a total search.

2.12. Optimal Decisions in Games

Game may be officially described as a type of exploration problem with the bellow elements:

- **Primary state**: This contains the panel place and finds the performer to shift.
- A **successor_function** (operators) that arrivals a listing of pairs (move, state), all indicting a permissible shift and the resultant state.
- A **terminal-test** that decides whenever the game is completed.
- A **utility function** (payoff function or objective function) that provides a numeral values for the closing states.

The Mini-max Algorithm

This algorithm executes the mini-max selection from the present state. It executes an entire DFS of the game-tree. If the highest depth of the tree is 'm', and it has 'b' permissible shifts at all positions, next the complexity time of the mini-max algorithm is O(bm).

Minimax is for deterministic games with perfect information. The **mini-max** algorithm creates the complete game tree and uses the utility function to every closing state. Next it transmits the utility value up single level and carry on to perform so until arriving the begining node.

The mini-max algorithm is as follows:

```
function MINIMAX_DECISION ( state ) returns an action
        input:state, present state in
        gamev?  MAX_VALUE (
        state )
        return the action in SUCCESSORS ( state ) with value 'v'


function MAX_VALUE ( state ) returns a utility value
        if TERMINAL_TEST( state ) then return UTILITY ( state )
        V?  - 8
        For s,a, in SUCCESSORS ( state ) do
                v?  MAX ( v,MIN_VALUE ( s ) )
return v

        function MIN_VALUE ( state ) returns a utility value
                if TERMINAL_TEST ( state ) then return UTILITY ( state )

                v?  8
                for s,a in SUCCESSORS ( state ) do
                        v?  MIN ( v,MAX_VALUE ( s ) )
    return v.
```

Figure: An Algorithm for Calculating Minimax Decisions

2.13. Alpha Beta Pruning

Pruning: The procedure of removing a branch of the search tree from discussion without investigation is called pruning. The 2 specifications of pruning methods are mentioned bellow:

a. **Alpha(α):** Better selection for the value of MAX(maximum) along the path or lesser bound on the value, which on increasing node can be finally allocated.
b. **Beta(β)**: Better selection for the value of MIN(minimum) along the path or higher bound on the value, which a decreasing node can be finally allocated.

Alpha-Beta Pruning: The values of alpha and beta are applied on a mini-max tree, it gives back the similar shift as mini-max, but prunes left the branches that may not control the last choice is called **Cutoff or Alpha Beta pruning**.

Consider the 2 play game tree from the bellow figure:

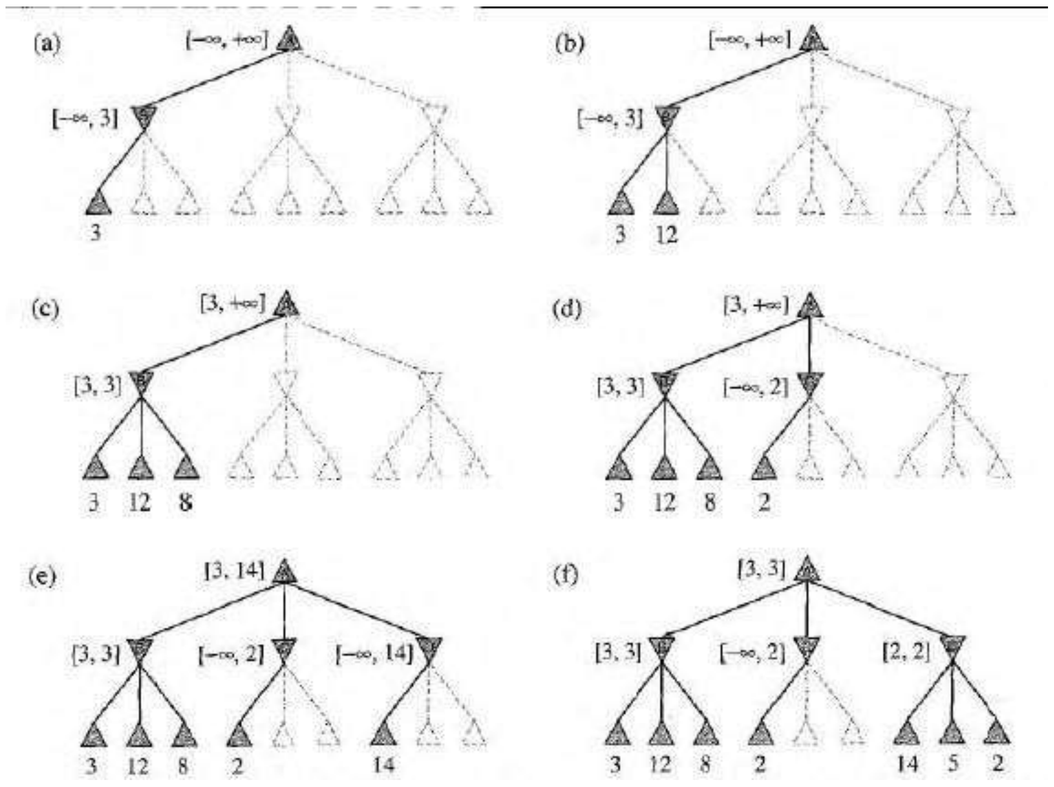


Figure 6.5 Stages in the calculation of the optimal decision for the game tree in Figure 6.2. At each point, we show the range of possible values for each node. (a) The first leaf below B has the value 3. Hence, B, which is a MIN node, has a value of *at most* 3. (b) The second leaf below B has a value of 12; MIN would avoid this move, so the value of B is still at most 3. (c) The third leaf below B has a value of 8; we have seen all B's successors, so the value of *B* is exactly 3. Now, we can infer that the value of the root is *at least* 3, because MAX has a choice worth 3 at the root. (d) The first leaf below C has the value 2. Hence, *C*, which is a MIN node, has a value of *at most* 2. But we know that *B* is worth 3, so MAX would never choose C. Therefore, there is no point in looking at the other successors of C. This is an example of alpha–beta pruning. (e) The first leaf below D has the value 14, so D is worth *at most* 14. This is still higher than MAX'S best alternative(i.e., 3), so we need to keep exploring D's successors. Notice also that we now have bounds on all of the successors of the root, so the root's value is also at most 14. (f) The second successor of D is worth 5, so again we need to keep exploring. The third successor is worth 2, so now D is worth exactly 2. MAX's decision at the root is to move to B, giving a value of 3.

Alpha-Beta pruning may be practiced to trees of all depths and this is frequently probable to prune complete sub trees in place of leaves.

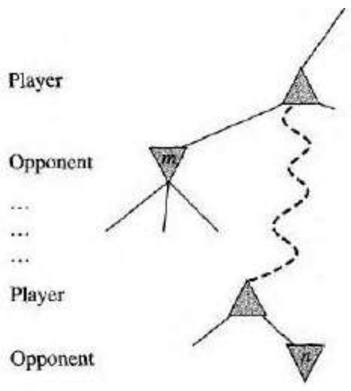


Figure Alpha Beta pruning, the common case. If 'm' is improved than 'n' for player, we can never obtain to 'n' in play.

```
function ALPHA-BETA-SEARCH(state) returns an action
  inputs: state, current state in game

  $v \leftarrow$ MAX-VALUE(state, $-\infty$, $+\infty$)
  return the action in SUCCESSORS(state) with value $v$
```

```
function MAX-VALUE(state, $a$, $\beta$) returns a utility value
  inputs: state, current state in game
          $a$, the value of the best alternative for MAX along the path to state
          $\beta$, the value of the best alternative for MIN along the path to state

  if TERMINAL-TEST(state) then return UTILITY(state)
  $v \leftarrow -\infty$
  for $a$, $s$ in SUCCESSORS(state) do
    $v \leftarrow$ MAX($v$, MIN-VALUE($s$, $a$, $\beta$))
    if $v \geq \beta$ then return $v$
    $a \leftarrow$ MAX($\alpha$, $v$)
  return $v$
```

```
function MIN-VALUE(state, $a$, $\beta$) returns a utility value
  inputs: state, current state in game
          $\alpha$, the value of the best alternative for MAX along the path to state
          $\beta$, the value of the best alternative for MIN along the path to state

  if TERMINAL-TEST(state) then return UTILITY(state)
  $v \leftarrow +\infty$
  for $a$, $s$ in SUCCESSORS(state) do
    $v \leftarrow$ MIN($v$, MAX-VALUE(%$a$, $\beta$))
    if $v \leq \alpha$ then return $v$
    $\beta \leftarrow$ MIN($\beta$, $v$)
  return $v$
```

Figure: An Alpha-beta Search Algorithm

2.14. Imperfect, Real Time Decisions

Effectiveness of Alpha Beta Pruning

It requires to observe only $O(b^{m/2})$ nodes to select the better move.

Futility cut-off

Finishing the searching of a sub tree, which attempt slight possibility for enhancement over other recognized path, is called futility cut off.

2.15. Games that Contain an Element of Possibility

Backgammon problem is a common game that joins the skill and luck. Dice are rotated at the starting of a player turn to decide the group of permissible shifts that is accessible to the player.

The bellow figure, white has rotated a 5-6, and has 4 probable shifts.

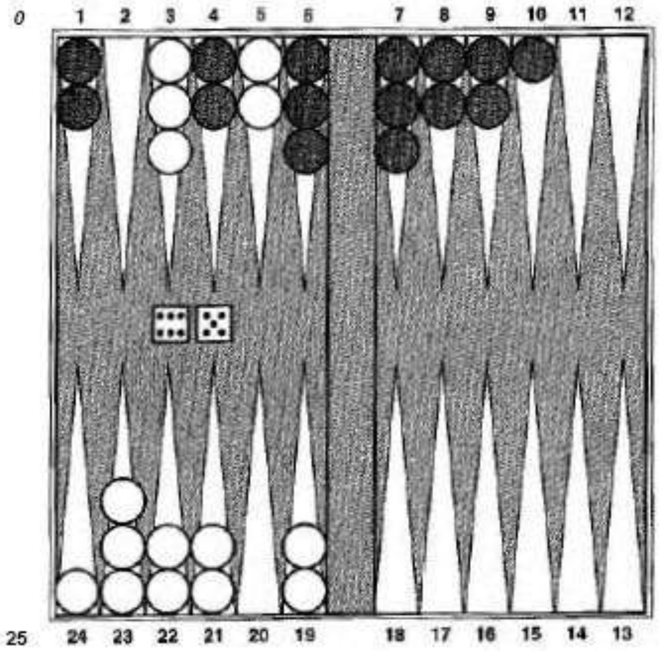


Figure A typical backgammon position. The goal of the game is to move all one's pieces off the board. White moves clockwise toward 25, and black moves counterclockwise toward 0. A piece can move to any position unless there are multiple opponent pieces there; if there is one opponent, it is captured and must start over. In the position shown, White has rolled 6–5 and must choose among four legal moves: (5–10,5–11), (5–11,19–24), (5–10,10–16), and (5–11,11–16).

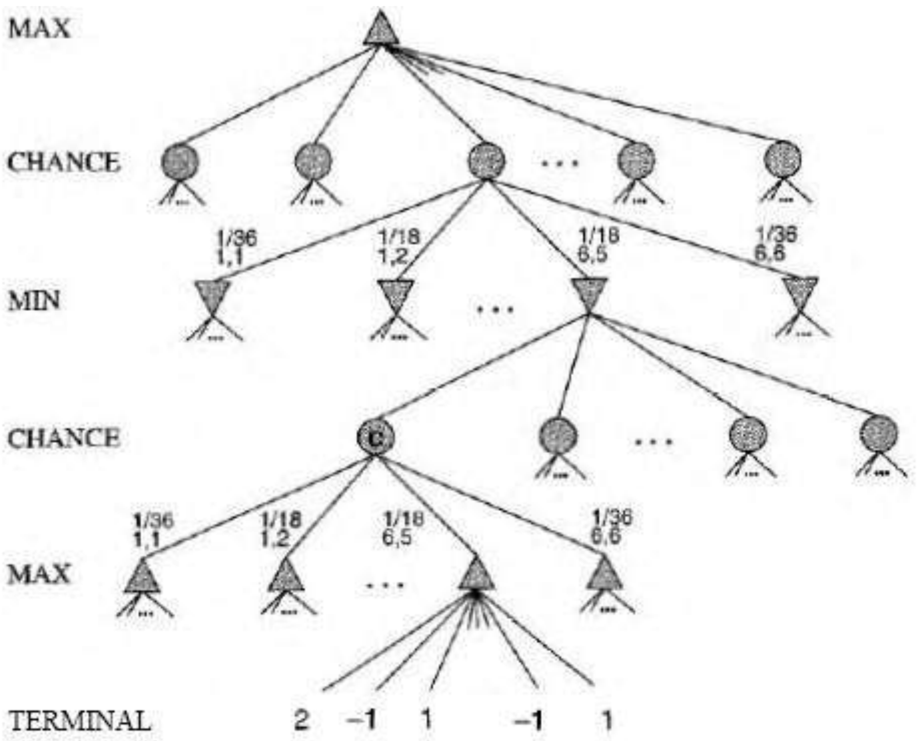


Figure: The Schematic Game Trees for a Backgammon Location

Question Bank

Unit - II

Part - A

1. Explain the local-minima problem.
2. How to increase the efficiency of a search-based problem-solving method?
3. Identify the complexity of mini-max.
4. What is the purpose of online search agents in unidentified environments?
5. Explain the difference between A*search with greedy search.
6. Describe relaxed problem.
7. Explain the 3 benefits of hill-climbing.
8. Define pruning.
9. How the search technique is perform powerful in continuous space.
10. Explain in detail about chance nodes.
11. Describe cycle cut set.
12. What are constraint-satisfaction problems? How can we prepare them as search problems?

13. Explain the effective branching factor with an example.
14. Write the step by step process of mini-max algorithm.
15. State the purpose to avoid the drawback of mini-max algorithm.
16. Explain the alpha and beta cutoff with example.
17. What is the requirement of memory delimited in heuristic search?
18. Describe CSP.
19. Define constraint graph.
20. List the different types of constraints.
21. Define backtracking search.
22. Define constraint propagation
23. What is a constraint-satisfaction problem?
24. Explain the game playing problems with an example.
25. What is local-search algorithm and explain the different types and applications.
26. Explain the memory bounded search with two examples.
27. How free-decomposition is attained?
28. Differentiate online search with offline search.
29. Describe the horizon problem with an example.
30. Explain the optimality of A* search and monotonocity.
31. What is the requirement of arc-consistency?
32. Explain the different kinds of consistency methods.
33. How does alpha beta pruning method works?
34. Describe the conflict-direct back jumping.

Part – B

1. Explain back-tracking search of CSP along with algorithm.
2. Explain in brief regarding memory-bounded search algorithms with one example for all searches.
3. How does hill-climbing make sure greedy-local-search?
4. Discuss the different issues related with the backtracking search from CSPs. How they addressed?
5. Explain in brief about heuristic-search methods with one example for every search.
6. What is genetic algorithm? Explain with one example.
7. Explain mini-max algorithm with its process with one example.
8. Explain the performance of A* search algorithm with one example.
9. Explain wheatear the problem structure affects on the solving method?

10. Describe the structure of problem using CSP?
11. What is the requirement of online search agent? Discuss with algorithm.
12. Describe the alpha beta pruning and provide the order changes to the mini-max process to increase its performance.
13. Explain the alpha-beta pruning with its process and with an example.

UNIT III

3. KNOWLEDGE REPRESENTATION

First Order Logic, Representation Revisited, Syntax and Semantics for First Order Logic - Using First Order Logic, Knowledge Engineering in First Order Logic, Inference in First Order Logic, Prepositional Versus First Order Logic, Lifting and Unification, Forward Chaining and Backward Chaining, Resolution, Knowledge Representation, Ontological Engineering, Categories and Objects, Actions, Simulation and Events, Mental Events and Mental Objects

3.1. First Order Logic(FOL)

3.2. Representation Revisited

FOL or **First Order Predicate Calculus** (FOPC), which creates a powerful set of ontological commitments i.e. properties, objects, functions and relations of the world belongings to be presented.

- **Properties** : It is used to differentiate one object with another object. (E.g. big, small, round etc.)
- **Objects** : Items with separate identities. (E.g. home, people, college, colors etc.)
- **Functions** : One type of relation, which has just a single value from a specified entry. (E.g. mother of, greatest brother etc.)
- **Relations** : relation exists between objects. (E.g. owns, bigger than etc.)

The initial variation between FOL logic and propositional lies in the existential obligation prepared by all languages – which are, what it suppose regarding the scenery of actuality.

Language	Ontological Commitment (What exists in the world)	Epistemological Commitment (What an agent believes about facts)
Propositional logic	facts	true/false/unknown
First-order logic	facts, objects, relations	true/false/unknown
Temporal logic	facts, objects, relations, times	true/false/unknown
Probability theory	facts	degree of belief $\in [0, 1]$
Fuzzy logic	facts with degree of truth $\in [0, 1]$	known interval value

Figure: Formal Languages with their Epistemological and Ontological Commitments

3.3. Syntax and Semantics for FOL

FOL has sentences, and also has terms, which signify objects. Variables, functions and Constant symbols are utilized to create terms, predicate symbols and quantifiers are utilized to create sentences.

Models for FOL

The domain of the representation is a group of objects it have; these objects are occasionally known as domain elements. Figure: displays a pattern with 5 objects.

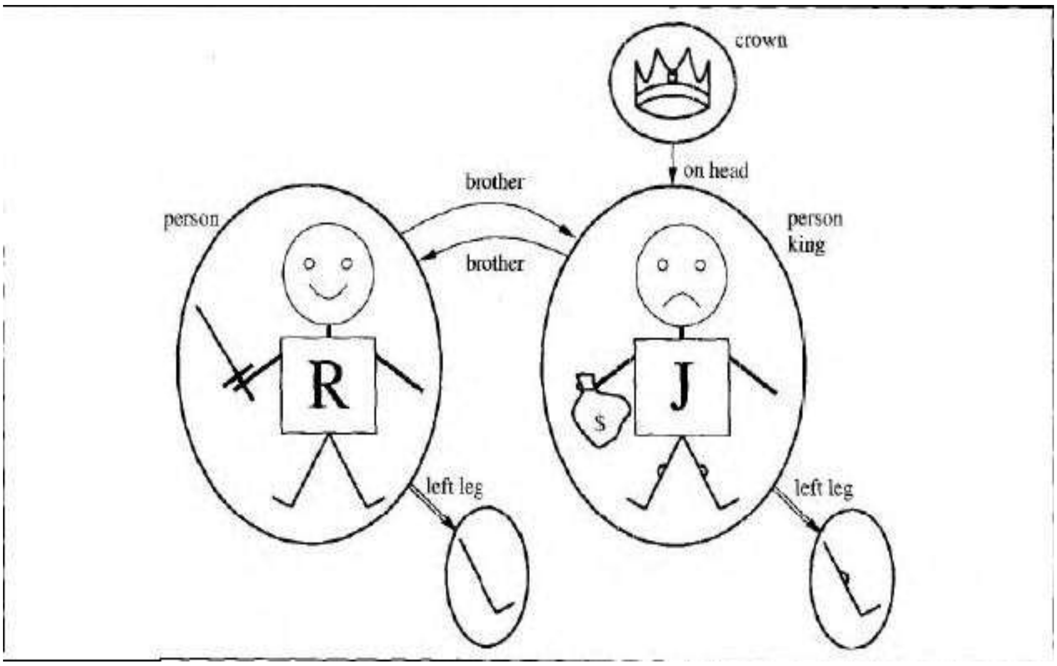


Figure: A Model Having 5 Objects Three Unary Relations, Two Binary Relations and One Unary Function

Symbols and Interpretations

Figure shows the formal grammar of FOL.

$$\begin{array}{rcl}
Sentence & \rightarrow & AtomicSentence \\
 & \mid & (\ Sentence\ Connective\ Sentence\ ) \\
 & \mid & Quantifier\ Variable, \ldots\ Sentence \\
 & \mid & \neg\ Sentence \\
 & & \\
AtomicSentence & \rightarrow & Predicate(Term, \ldots) \mid Term = Term \\
 & & \\
Term & \rightarrow & Function(Term, \ldots) \\
 & \mid & Constant \\
 & \mid & Variable \\
 & & \\
Connective & \rightarrow & \Rightarrow \mid \wedge \mid \vee \mid \Leftrightarrow \\
Quantifier & \rightarrow & \forall \mid \exists \\
Constant & \rightarrow & A \mid X_1 \mid John \mid \ldots \\
Variable & \rightarrow & a \mid x \mid s \mid \ldots \\
Predicate & \rightarrow & Before \mid HasColor \mid Raining \mid \ldots \\
Function & \rightarrow & Mother \mid LeftLeg \mid \ldots
\end{array}$$

Figure: The Syntax of FOL with Equality, Specified in BNF

The basic semantic components of the FOL are the signs that apply for functions, relations and objects. The signs arrive in 3 types: **function symbols** apply for functions; **constant symbols** apply for objects; and **predicate symbols** apply for relations.

Terms are a logical statement, which indicates to an object.

An **atomic statement** is created from a predicate symbol pursued by a parenthesized listing of expressions.

The **atomic statement** is correct in a specified model, beneath a specified clarification, if the relationship indicated by the predicate symbol keeps amongst the objects indicated by the arguments.

Quantifiers: Theses are used to state properties of total collection of objects, than presenting the objects by the name.

First order logic (FOL) contains 2 quantifiers:

1. Existential quantifiers
2. Universal quantifiers

Equality

FOL consists of one additional way to build atomic sentences, excluding a terms and predicate as described in advance. We may apply the **equality sign** to create sentences to work that 2 expressions indicate to the similar object.

3.4. Knowledge Engineering in FOL

Knowledge Engineering (KE): The common procedure of knowledge base planning procedure is called knowledge engineering (KE).

The Knowledge Engineering Method

KE projects differ usually in difficulty, scope and content, but suchlike projects contain the bellow steps:

a. Discover the task.
b. Collect the appropriate knowledge.
c. Choose on expressions of constants, functions and predicates.
d. Encode broad awareness regarding the domain.
e. Encode an explanation of the detailed problem instance.
f. Pose questions to the deduction process and obtain response.
g. Debug the intelligence base.

Electronic Circuits' Domain

We can establish a knowledge base and ontology that permit us to cause regarding digital circuits of the type displayed in the bellow Figure. We follow the 7 step procedure for information engineering.

a. Discover the job.
b. Collect the appropriate knowledge.
c. Choose on expressions of constants, predicates and functions.
d. Encode broad awareness regarding the domain.
e. Encode an explanation of the detailed problem instance.

f. Pose questions to the deduction process and obtain response.

g. Debug the intelligence base.

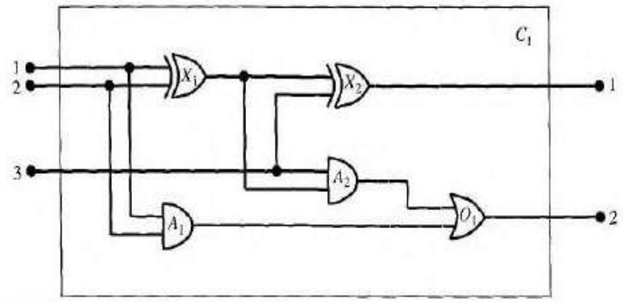


Figure: A digital-circuit 'C1', purporting to be a 1-bit. Full adder. The primary 2 inputs are the 2 bits to be inserted and the 3rd input is a carry bit. The 1st output is the addition and the 2nd output is a carry bit for the subsequent adder. The circuit has 2 XOR gates, 2 AND gates and 1 OR gate.

3.5. Inference in FOL

FOL vs. Propositional

Inference Rules for Quantifiers

The rule of **Universal Instantiation**(UI) declares that, we may assume every statement gained by replacing a **ground-term** for the variable.

SUBST(@*a*, *)* indicate the outcome of using the replace *eight* to the statement '*a'.* Afterward the rule is drafted

Vv*a*

SUBST({ V/~4),

for every ground term '*g'* and variable 'v'*.*

The subsequent **Existential-Instantiation** rule: the existential quantifier is somewhat most complex. For every variable-v, constant symbol-*k* and sentence '*a',* which doesn't arrive somewhere in the intelligence base.

3.6. Lifting and Unification

A First Order Inference Rule

This inference procedure may be confined as a distinct inference rule, which we term General.

GENERALIZEPONEDN S **sized Modus-Ponens:** The atomic statement *q* and *p,,,, p,',*

Wherever there is a replacement *eight* such *Suesr(Q,pil)=Suesr(B,p,),* for all *i,*

$$\frac{p_1{}',\ p_2{}',\ \ldots,\ p_n{}',\ (p_1 \land p_2 \land \ldots \land p_n \Rightarrow q)}{\text{SUBST}(\theta, q)}$$

Unification

Lifted-inference rules need discovering replacements, which create dissimilar reasonable statements.

UNIFICATION looks the same. This procedure is known as **unification** and it is a crucial element of every first-order UNIFIER inference algorithms.

The ***UNIFY* algorithm** obtain 2 statements and give back a **unifier** for them if one be present:

UNIFY(^,q)=**0** wherever *SUBST(O,= ~)SUBST(@ q,).*

The problem happens because the 2 sentences occur to utilize the similar name of a variable. The problem may be keep away by **regularizing separately** one of the 2 statements is combined that means changing the name of its APART variables to keep away name conflicts.

```
function UNIFY($x$, $y$, $\theta$) returns a substitution to make $x$ and $y$ identical
  inputs: $x$, a variable, constant, list, or compound
          $y$, a variable, constant, list, or compound
          $\theta$, the substitution built up so far (optional, defaults to empty)

  if $0$ = failure then return failure
  else if $x = y$ then return $\theta$
  else if VARIABLE?($x$) then return UNIFY-VAR($x, y, 0$)
  else if VARIABLE?($y$) then return UNIFY-VAR($y$, x, $0$)
  else if COMPOUND?($x$) and COMPOUND?($y$) then
      return UNIFY(ARGS[$x$], ARGS[$y$], UNIFY(OP[$x$], OP[$y$], $\theta$))
  else if LIST?($x$) and LIST?($y$) then
      return UNIFY(REST[$x$], REST[$y$], UNIFY(FIRST[$x$], FIRST[$y$], $\theta$))
  else return failure
```

```
function UNIFY-VAR($var$, $x$, 0) returns a substitution
  inputs: $var$, a variable
          $x$, any expression
          $0$, the substitution built up so far

  if $\{var/val\} \in 6$ then return UNIFY($val$, $x$, $\theta$)
  else if $\{x/val\} \in \theta$ then return UNIFY($var$, $val$, $\theta$)
  else if OCCUR-CHECK?($var$, $x$) then return failure
  else return add $\{varlx)$ to 6
```

Figure: The UNIFY Algorithm

Save and Retrieval

Primarily the SAY and INQUIRE functions are used to tell and investigate a information base are the additional primary SAVE and RETRIEVAL functions. SAVE ‘S’ saves a statement ‘*s*’ into the information base and RETRIEVAL (^) gives back every unifiers such the question ‘q’ unifies with a few statement in the information base.

3.7. Forward Chaining

Forward chaining algorithm for hypothesis definitive sections was already specified. The proposal is easy: initiate with the atomic statements in the information base and perform ModusPonens in the frontward direction, inserting latest atomic statements, upto no more inferences may be created.

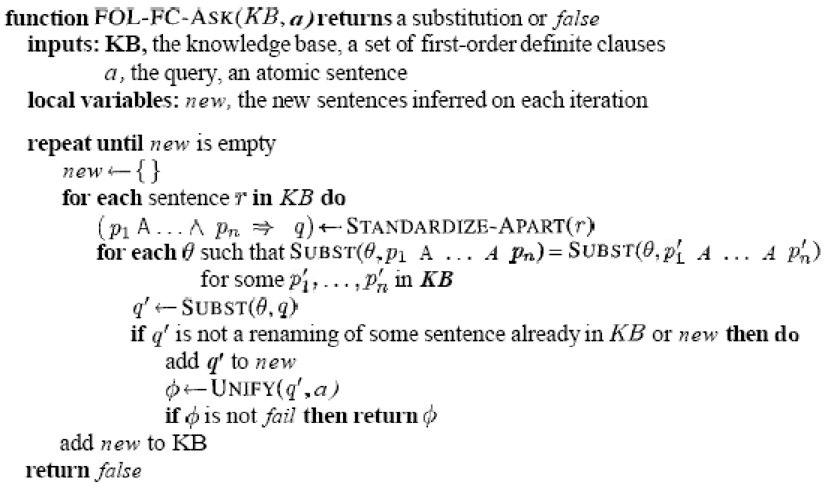
First Order Definite Sections

It directly resembles propositional definitive sections these are disunion of accurate of which *precisely 1 is positive. An* exact sections whichever is atomic or it’s an inference whose predecessor is a combination of optimistic literals and who’s consequential is an individual optimistic literal.

This data base contain no action symbols and therefore an example of the class “DATALOG” of **Data-log** databases, which is, the groups of first-order definitive sections without function symbols.

An Simplest Forward Chaining Algorithm

Initial Forward-Chaining algorithm can believe a simplest one, as displayed in the bellow Figure.



```
function FOL-FC-ASK($KB$, $a$) returns a substitution or $false$
  inputs: KB, the knowledge base, a set of first-order definite clauses
          $a$, the query, an atomic sentence
  local variables: $new$, the new sentences inferred on each iteration

  repeat until $new$ is empty
     $new \leftarrow \{\}$
     for each sentence $r$ in $KB$ do
         $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \leftarrow$ STANDARDIZE-APART($r$)
         for each $\theta$ such that SUBST($\theta, p_1 \wedge \ldots \wedge p_n$) = SUBST($\theta, p'_1 \wedge \ldots \wedge p'_n$)
                      for some $p'_1, \ldots, p'_n$ in $KB$
             $q' \leftarrow$ SUBST($\theta, q$)
             if $q'$ is not a renaming of some sentence already in $KB$ or $new$ then do
                 add $q'$ to $new$
                 $\phi \leftarrow$ UNIFY($q', a$)
                 if $\phi$ is not $fail$ then return $\phi$
     add $new$ to KB
  return $false$
```

Figure: Efficient Forward Chaining

There are 3 available basis of difficulty. Initially, the inner loop of the algorithm includes discovering every available unifier such the basis of a unifies rule with an appropriate group of particulars in the data base. This is generally known as **pattern matching** and will be high costly. Second, this algorithm again checks each rule on each round to observe regardless if its locations are fulfilled, yet if little add-ons are completed to the data base on every round. Lastly, the algorithm could produce several details that are unnecessary to the target.

Matching Rules against Unknown Facts

We will convey each limited domain CSP as an individually best section jointly with a few related ground particulars.

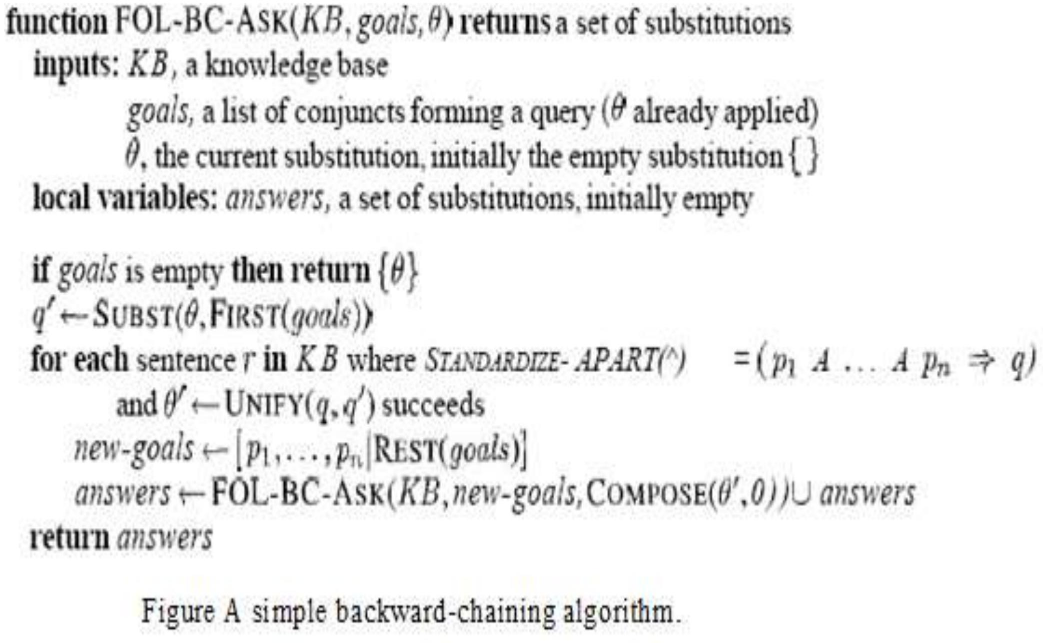
Incremental Forward Chaining

Redundant rule matching will be keep away if we create the following studies: ***Each latest fact deduced on repetition 't' should be resulting from no less than one latest fact deduced on repetition 't–1'.***

3.8. Backward Chaining

Backward-Chaining Algorithm

The bellow figure explains an easy backward-chaining algorithm:



```
function FOL-BC-ASK(KB, goals, $\theta$) returns a set of substitutions
  inputs: KB, a knowledge base
          goals, a list of conjuncts forming a query ($\theta$ already applied)
          $\theta$, the current substitution, initially the empty substitution { }
  local variables: answers, a set of substitutions, initially empty

  if goals is empty then return {$\theta$}
  $q'$ ← SUBST($\theta$, FIRST(goals))
  for each sentence r in KB where STANDARDIZE-APART(^)   = ($p_1$ A ... A $p_n$ ⇒ q)
        and $\theta'$ ← UNIFY(q, $q'$) succeeds
     new-goals ← [$p_1$, ..., $p_n$|REST(goals)]
     answers ← FOL-BC-ASK(KB, new-goals, COMPOSE($\theta'$, 0)) ∪ answers
  return answers
```

Figure A simple backward-chaining algorithm.

Logic Programming

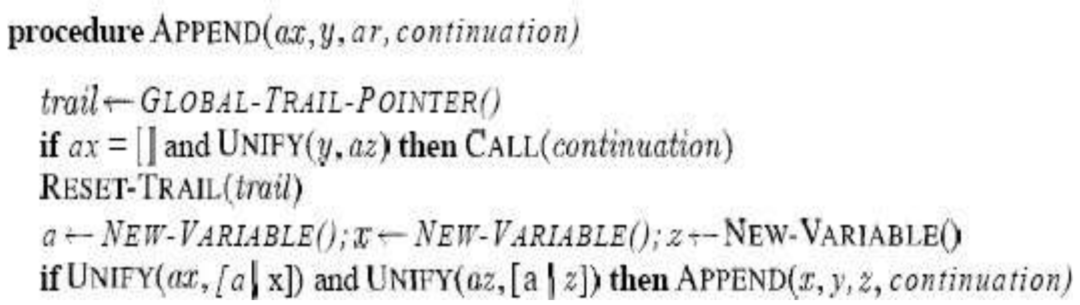
$$Algorithm = Logic + Control$$

Prolog is by distant the almost extensively utilized logic programming language. Its clients count in the hundreds of thousands. It is utilized initially as a fast prototyping language and for sign modification work such as scripting compilers(1990, VanRoy) and interpret to natural language. Prolog-programs are groups of definitive sections drafted in a document somewhat dissimilar from normal first-order. Logic Program utilizes small letters for constants and capital letters for variables. Sections are drafted with the head earlier the body:- it is utilized for a period marks the ending of a statement, comma divide literals in the body and left implication:

The implementation of Prolog-programs is complete via depth first backward-chaining, where sections are attempted in the order, and then they are drafted in the knowledge base. Few portions of Prolog drop outside normal logical inference:

Proficient Execution of Logic Programs

The implementation of a Prolog-program may occur in 2 forms interpret and compiled. Interpretation fundamentally quantities to operating the FOL_ BC_ ASK algorithm from the bellow figure.



```
procedure APPEND(ax, y, ar, continuation)

    trail ← GLOBAL-TRAIL-POINTER()
    if ax = [ ] and UNIFY(y, az) then CALL(continuation)
    RESET-TRAIL(trail)
    a ← NEW-VARIABLE(); x ← NEW-VARIABLE(); z ← NEW-VARIABLE()
    if UNIFY(ax, [a | x]) and UNIFY(az, [a | z]) then APPEND(x, y, z, continuation)
```

Figure: APPEND Algorithm

3.9. Resolution

Conjunctive Normal Form (CNF) for First-order Logic

In the propose case, first-order decision needs that statements be in **CNF**, which is, a conjunction of sections, wherever every section is a disconnection of literal. Literals may have variables that are supposed to be globally evaluated.

e.g., the statement

V x American(x) A Weapon(y) A Sells(z, y , z) A Hostile(z) +- Criminal (x)

becomes, in CNF,

l A m e r i c a n (x) V -1 Weapon(y) V l S e l l s (x , y , z) V 1 Hostile(z) V Criminal (x) .

Each statement of first-order logic may be transformed into a priory similar CNF statement.

We can demonstrate the process by interpreting the sentence "Every person who feels affection for all animals is loved by everyone,"

Or

'dx ['d y Animal(y) J Loves(x, y)] + [3 y Loves(y, x)] .

The steps are as follows:

1. Eliminate implications:

ti x [i ' d y 1 Animal (y) V Loves(x, y j] v [3 y Loves(y, xj]

2. Shift *for* inmost: Adding to the common rules for invalid combinative, we require rules for invalid quantifiers. Therefore, we contain *l'dxp* turn into *3x1p*73*xp* turn into *'dx1p*.

Our sentence moves between the following conversions:

'd x [3 y l(lAnimal (Y) V Loves(x, y))] V [3 y Loves(y, x)] .

'dx[3 y iiAnimal(y) A lLoves(x, y)] V [3 y Loves(y, x)] .

t i x [3 y Animal(y) A lLoves(x, y)] v [3 y Loves(y, x)] .

Normalize Variables

Distribute V over A

Drop global quantifiers

Solemnize

Completeness of Resolution

Resolution is **refusal-complete** that means, which is a group of statements is not classifiable, and then resolution can forever be capable to obtain a challenge. Resolution may not be utilized to produce all rational outcomes of a group of statements, but it may be utilized to create that a specified statement is involved by the group of statements.

Our target is to demonstrate the following: if 'S' ***is an unsatisfied group of sections next the request of limited count of resolution steps to 'S' can give up a negation.***

The fundamental structure of the evidence is exposed in the bellow Figure.

It continues as follows:

1. Initially, we examine that if 'S' is unclassified, next there continues an appropriate group of ***ground_instances*** of the sections of 'S' thus, this group is unclassified (Her brand's principle).
2. We request to the **ground resolution theorem** that situations that proposal decision is finished for ground statements.
3. We utilize a **lifting lemma** to demonstrate that, for every proposal decision evidence utilizing the group of ground statements, there is matching first-order decision evidence utilizing the first-order statements from which the ground statements are gained.

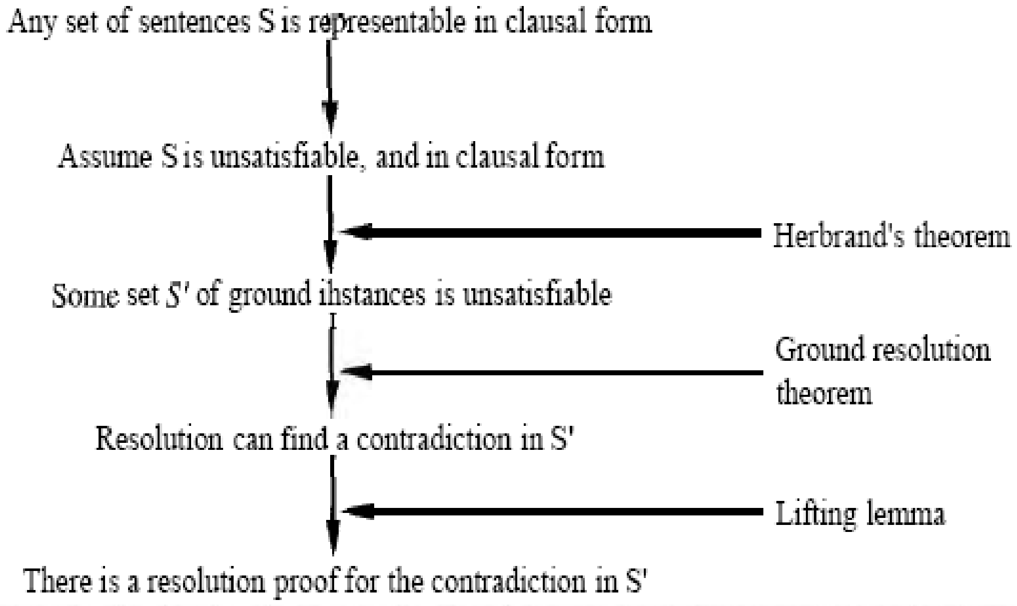

Any set of sentences S is representable in clausal form
Assume S is unsatisfiable, and in clausal form
Herbrand's theorem
Some set S' of ground instances is unsatisfiable
Ground resolution theorem
Resolution can find a contradiction in S'
Lifting lemma
There is a resolution proof for the contradiction in S'

Figure: Structure of a Completeness Proof for Resolution

```
procedure OTTER(sos, usable)
   inputs: sos, a set of support—clauses defining the problem (a global variable)
           usable, background knowledge potentially relevant to the problem

   repeat
       clause ← the lightest member of sos
       move clause from sos to usable
       PROCESS(INFER(clause, usable), sos)
   until sos = [ ] or a refutation has been found
```

```
function INFER(clause, usable) returns clauses

   resolve clause with each member of usable
   return the resulting clauses after applying FILTER
```

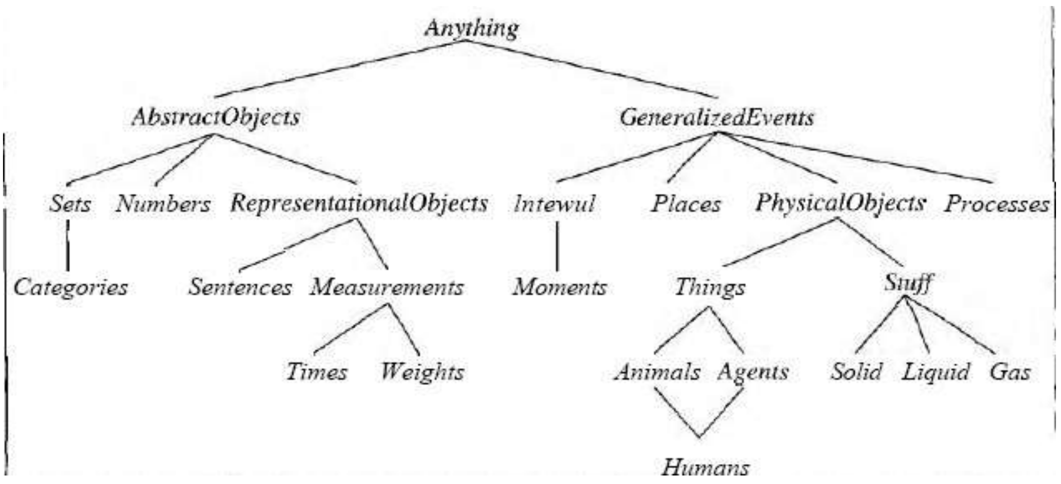
```
procedure PROCESS(clauses, sos)

   for each clause in clauses do
       clause ← SIMPLIFY(clause)
       merge identical literals
       discard clause if it is a tautology
       sos ← [clause | sos]
       if clause has no literals then a refutation has been found
       if clause has one literal then look for unit refutation
```

Figure : Drawing of the OTTER theorem prover. Heuristic-control is executed in the choice of the "lightest" section and in the FILTER function that deletes not interesting sections from deliberation.

3.10. Knowledge Representation

3.11. Ontological Engineering

This chapter demonstrates how to make these presentations, focusing on common ideas, such as Time, *Actions,* Beliefs-that, *and Physical Objects* occur in most dissimilar domains. Presenting these abstract ideas is occasionally known as ontological-engineering. It is linked to the data engineering procedure, but works on a grander-scale. The expectation of presenting *all* in the universe is frightening.



For example, we can describe the information of dissimilar kinds of books, televisions, objects robots, or what should be filled afterward.

The common structure of ideas is known as **upper ontology,** for the reason that of the gathering of sketch charts with the common ideas at the top and the most precise ideas.

3.12. Objects and Categories

The management of objects into **categories** is a very important element of data presentation. Even if communications with the universe take position at the stage of single objects, *a lot analysis gets position at the stage of categories.*

There are 2 chances for presenting categories in first order logic: **objects** and **predicates.**

Measurements

The both commonsense and scientific theories of the universe, cost, mass, objects contain height, and so on. The values, which we allocate for the properties are known as **measures.** Normal quantitative-measures are really simple to present. We assume that the world contains abstract objects measure as the *length,* which is the length of this line-segment.

3.13. Events, Actions, and Situations

The Idea of Situation Calculation

One noticeable way to keep away from several duplicates of axioms is easily to count more time to declare, *"'Vt',* is the outcome at *t*+1 of performing the operation at *'t'".* Rather than association with accurate times similar to *t*+1, we can focus on this clause on ***situations*** that represent the resultant states from performing operations. This method is known as **situation calculation** and associates with the idea:

Activities are logical expressions such as ***frontward*** and ***TurnRight***. *C*urrently, we can imagine that the situation involves just a single agent. (If there is greater than one, an extra argument may be added to convey. Which, agent is performing the activity).

Situations are logical expressions containing the primary state (regularly known as *So)* and each situation that are created by using an activity to a state. The function *Result(a,s)* (occasionally known as *Do)* names the state that outcomes whenever activity *a* is applied in state *s.* The bellow Figure demonstrates this thought.

Fluent are tasks and declares that differ from one state to the other, the position of the agent or the attention of the wumpus. The dictionaries convey a fluent is a thing that flow similar to a fluid. It means flowing or shifting crossways states. By showing, the state is at all times the previous argument of a fluent. example, *l Holdzng (G 1,So)* coveys that the agent is not containing the gold-GI in the primary state *'So'. Age (Wumpus,So)* indicates to the wumpus's age in *'So'.*

A temporary or **continual** builds and performs are also permitted. Examples contain the predicate_Gold(GI) and the function *Left_Leg_Of* (Wumpus *).*

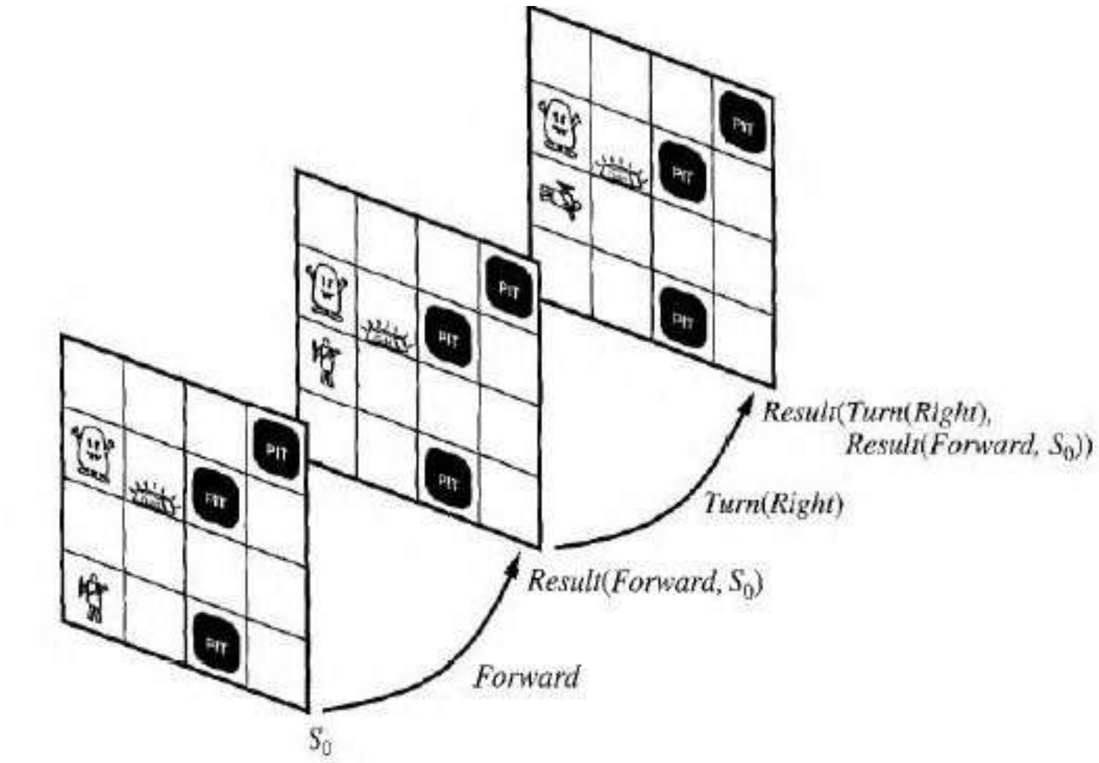


Figure: In Situation Calculation, Every Situation (excluding 'So') is the Outcome of an Action

A state calculation agent would be capable to deduct the result of a specified series of PROJECTION activity; this is the **projection** work, with a appropriate constructive_inference algorithm, it would also be capable to connect a series that attains a preferred outcome; it is the **planning** job.

Defining Actions in Situation Calculations

In the easiest edition of situation calculation, every action is defined by 2 axioms: a **possibility axiom,** which conveys when it is probable to perform the operation, and an **effect axiom,** which conveys EFFECT-AXIOM what occurs whenever an available action is performed.

The axioms contain the bellow method:

EFFECT_AXIOM: *Poss (a,s)+Modifications that outcome from doing action.*

POSSIBILITY_AXIOM: *Preconditions+Poss (a,s)*.

The problem is the effect-axioms conveys what modifications, but does not convey what continues the same.

Presenting all the things, which continue the similar, is known as the **frame problem.** We can discover an adequate result to the frame-problem for the reason that, the actual universe, approximately all continues the similar approximately everything at the time. Every operation changes just a small part of every fluent.

A method is to draft accurate **frame_axioms,** which conveys what continues the similar.

Resolving the Presentational Frame Problem

The result to the presentational frame problem contains only a small modification in point of view on how to draft the axioms. As an alternative of drafting out the outcome of every operation, we look at how every fluent predicate derives above the time. The axioms we utilize are known as **successor state axioms.**

It contains the bellow method:

AXIOM SUCCESSOR-STATE AXIOM:

Operation is available+(Fluent is true in solution state#Action 'S'
reaction completed it true. It was true previous to and operation is left).

The *Identical names-axiom* states are not qualifies for each pair of not changeable values in the knowledge base.

Resolving the Inferential-frame Problem

To resolve the inferential-frame problem, we contain 2 chances. Initially, we wo*uld* reject situation calculation and discover a latest formality for drafting axioms. That has been completed with formalities such as *'1'* has *fluent-calculus.* Second, we would change the inference process to control frame axioms rnose precisely.

Time and Event Calculus

The starting and ending associations play a responsibility same to the outcome association in state calculation; Initiates(e,f,t) means, which the happening of event 'e' at time 't' source fluent 'f' to turn into true, while Terminates(e,f,t_1) means, which 'f' halts to be true. We utilize Happens(e,t) to denote that event 'e' occurs at time 't', and we utilize Clipped(f,t,t_2) to denote that 'f' is ended by few events for a time among 't_1' and 't_2'. Properly, the axiom is:

EVENT CALCULUS AXIOM:

$$\begin{aligned} T(f,t_2) \;\Leftrightarrow\; & \exists\, e,t \;\; Happens(e,t) \wedge Initiates(e,f,t) \wedge (t < t_2) \\ & \wedge \neg Clipped(f,t,t_2) \\ Clipped(f,t,t_2) \;\Leftrightarrow\; & \exists\, e,t_1 \;\; Happens(e,t_1) \wedge Terminates(e,f,t_1) \\ & \wedge (t < t_1) \wedge (t_1 < t_2)\;. \end{aligned}$$

Generalized Events

A generalized event is collected from features of a few "space time-chunk". It is a portion of this multi-dimensional space time world. This removal simplifies the majority of the ideas we have observed so far, locations, fluent, time, physical objects, and including actions.

3.14. Mental Events and Mental Objects

A traditional theory of principles, we start with the connection among mental objects and agents, associations such as Wants, Knows, and Believes. Associations of this type are known as propositional attitudes, be-ATTITUDE because they explain an approach, which an agent may obtain near a theorem. Rotating a theorem into an object is called reification.

Professionally, the property of being capable to replacement a expression openly for an equivalent expression is known as **Referential Transparency**.

There are 2 approaches to attain this.

The **primary** is to utilize a dissimilar type of logic known as **Modal Logic**, in which propositional approaches such as Knows and Believes turn into Modal Operators, which is attentively unclear. This method is enclosed in the old comments part.

The **secondary** method that we can follow is to attain efficient dullness inside a Referentially-Transparent language by utilizing a syntactic-theory of mental-objects. This denotes that mental-objects are presented by a group of characters.

Knowledge and Belief

The relations among knowing and believing have been prepared broadly in beliefs. It is generally conveyed that intelligence is explained true belief. Action, time, and knowledge.

Actions may contain knowledge effects and knowledge preconditions.

Workout to collect and utilize knowledge is often presented by using a shorthand document called **runtime variables**.

Question Bank

Unit - III

Part - A

1. Write short notes on knowledge based agent.
2. Write the two functions of KB agent.
3. Define inference.
4. Explain three levels of knowledge based agent with an example.
5. Define logic.
6. Define entailment.
7. Define truth preserving in logic.
8. Give example for syntax and semantic representation.
9. Define inference procedure.
10. Define logical inference or deduction.
11. Define validity with one example.
12. Describe satisfiablity with one example.
13. List the names of five different types of logic.
14. Differentiate propositional logic with FOL.
15. Write the BNF grammar representation for propositional logic.
16. List the names of inference rules of propositional logic.
17. Write the BNF grammar representation for FOL.
18. Define predicate symbol with an example.
19. Define WFF with an example.
20. Differentiate function and relation in FOL with an example.
21. Define ground term.

22. Define horn clause.
23. Write short notes on higher order logic.
24. Write short notes on uniqueness quantifier.
25. Write short notes on uniqueness operator.
26. Define domain. Give example.
27. Define axiom.
28. Define independent axiom.
29. List the names of logical agents for wumpus world problem.
30. List the names of inference rules with quantifiers.
31. State the advantage of generalized modus ponen rule.
32. Write short notes on canonical form.
33. Write short notes on unification.
34. Differentiate forward and backward chaining.
35. Define refutation.
36. Define skolimization with an example.
37. Differentiate CNF and implicative normal form.
38. Write short notes on resolution strategies.
39. Covert the given sentence into propositional form.
40. Convert the given sentence into FOL form.
41. Convert the given sentence into CNF & INF form.
42. Define production system. Give example.
43. Define binding list with an example.
44. Write short notes on subsumption method.
45. How parallelization can be achieved in logic programming?
46. Define ontological engineering with an example.
47. Write short notes on situation calculus.
48. Differentiate suff nouns with count nouns with an example.
49. Define frame problem.
50. List the types of frame problem.
51. List the predicates of time intervals.
52. Define verification.

Part - B

1. Define the problem of wumpus world environment and derive the steps to reach a goal Test with corresponding structure representation.

2. Solve the wumpus world problem using propositional logic.
3. Discuss the logical agents of wumpus world problem.
4. Solve the given KB problem using FOL representation.
5. Explain the inference rules of FOL with an example for each.
6. A). Write the rules to convert a FOL sentence into Normal form

   B). Solve the given KB problem using resolution with refutation technique in CNF & INF form.
7. Solve the given KB problem using resolution with refutation technique in INF & CNF.
8. Explain knowledge engineering in FOL with an example.
9. Explain Forward chaining algorithm with one example.
10. Explain Backward algorithm with one example.
11. Explain with suitable example how the real world happening are represented in FOL.
12. Describe unification algorithm with one example.

UNIT IV

4. LEARNING

Learning form Observations – Forms of Learning – Inductive Learning – Learning Decision Trees – Ensemble Learning – Knowledge in Learning – Logical Formulation of Learning – Explanation based Learning – Learning Using Relevant Information – Inductive Logic Programming – Statistical Learning Methods – Learning with Complete Data – Learning with Hidden Variable – EM Algorithm – Instance based Learning – Neural Networks – Reinforcement Learning – Passive Reinforcement Learning – Active Reinforcement Learning – Generalization in Reinforcement Learning.

4.1. Learning from Observations

Studying takes position as the agent seeing its communications with the universe and its individual choice making procedure. Studying takes several varieties depending on the environment of the presentation element, the element to be enhanced, and the existing comments.

4.2. Forms of Learning

Learning agent is a performance agent that chooses what activities to be taken and a learning component that changes the performance component so that best choices can be taken in the upcoming. The plan of a learning element is influenced by 3 main issues:

- What *presentation* is utilized for the elements?
- What *feedback* is possible to learn these elements?
- Which *elements* of the performance element to be learned?

The components of those agents contain the following:

1. A straight mapping from situation on the present situation to activities.
2. A way to assume related properties of the universe from the percept-sequence.
3. Details regarding the method the universe develop and regarding the outcomes of available operations the agent may get.
4. Function details representing the importance of the global situation
5. Operation value details representing the importance of operations.
6. A target that explains classes of status whose success increases the agent's function.

The kind of comments applicable for studying is typically the most essential factor in resolving the environment of the studying problem, which the agent deals with.

The domain of machine-learning typically differentiates with 3 cases: **reinforcement, supervised, and unsupervised** learning.

Reinforcement Learning

Here learning arrangement is rather than being advised by a tutor, it learns from supplementing, i.e. by occasional rewards. The presentation of the learned knowledge just performs very imperative job in deciding how the learning algorithm should perform. The final main feature in the plan of learning methods is the possibility of earlier knowl*edge.*

Supervised Learning

1. A correct answer for each example or instance is available.
2. Learning is done from known sample input and output.

Unsupervised Learning

It is a learning pattern is which correct answers are not given for the input. It is mainly used in probabilistic learning system.

4.3. Inductive Learning

An **example** is a couple of values '*x' and 'f* (z)', wherever 'x' is the entered value and '*f(x)'* is the output of the function tested to '*x'.* The job of **pure-inductive-inference** or **induction** is this: Specified a set of instances of '*f',* returns a function '*h'* that estimates 'f'. The function 'h' is known as a **hypothesis.**

For the functions which are nondeterministic, there is a predictable tradeoff between the degrees of jit to the information and the difficulty of the hypothesis. There is an exchange between the difficulty of selecting simple, reliable hypotheses inside that space and the expressiveness of a hypothesis space.

4.4. Learning Decision Trees

Decision tree introduction is the easiest and most effective variety of learning-algorithm. It provides a better introduction to the field of introductory learning, and is simple to apply.

Decision Trees as Performance Elements

A decision tree gets as entry an item or scenario defined by a group of attributes and gives back a selection the expected resultant value for the entered value. The input attributes may be continuous or discrete. At this time, we imagine distinct inputs. The resultant value also can be continuous or distinct; studying a distinct valued characteristic is known as **classification** learning, learning a continual function is known as **regression**.

A decision tree arrive its selection by executing a series of assessments. Every internal node inside the tree correlates to check of the value of one of the properties, and the branches from the node are categorized with the available values of the check. Every leaf node inside the tree gives the value to be revisit if that leaf is arrived. The decision-tree presentation appears to be most normal for peoples; certainly, several "How To" guidelines (e.g., for vehicle repair) are drafted totally as a single-decision-tree extending more than 100s of leafs.

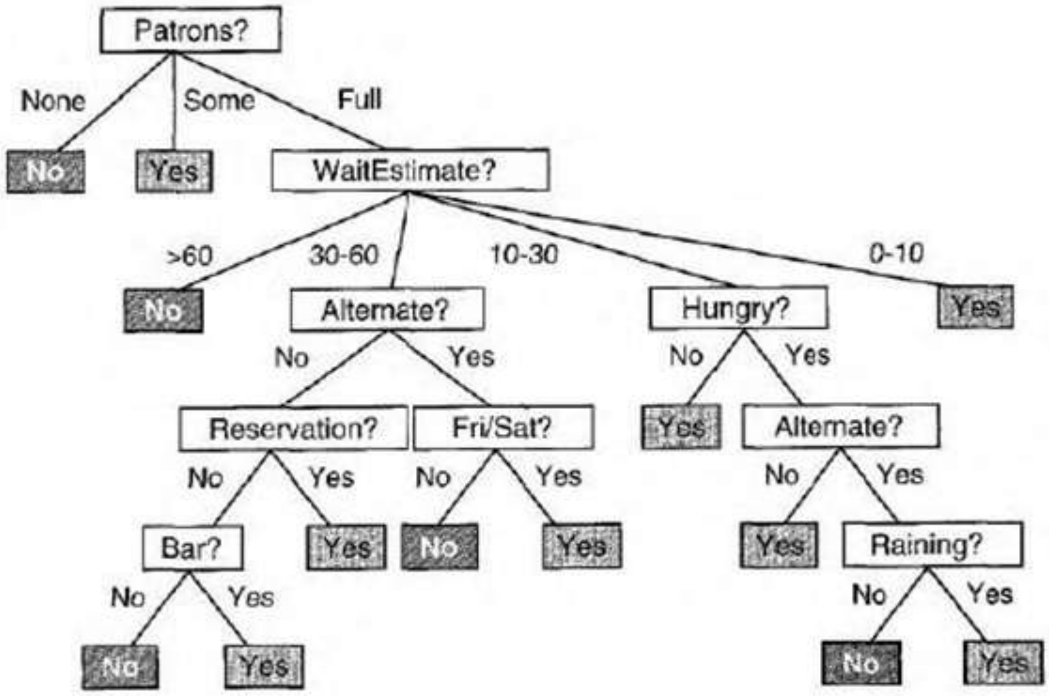


Figure: The Decision Tree for Choosing Whether to Stay for a Table

Expressiveness of Decision Trees

The relationship between the conclusion and the logical combination of attribute values the decision tree expressed as:

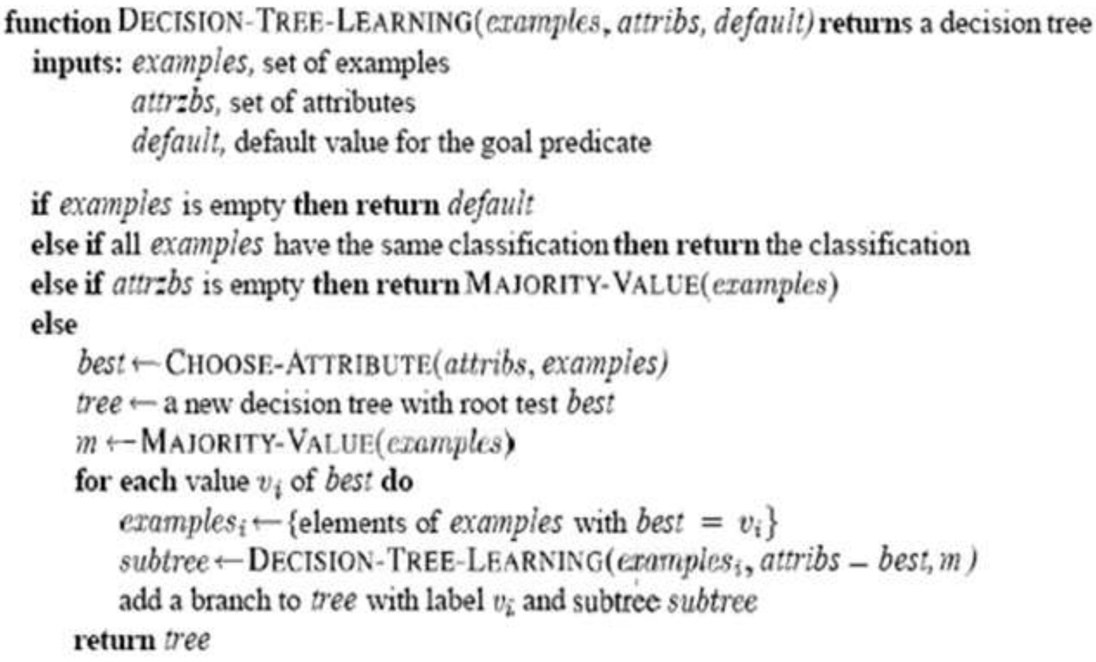
- **Propositional logic –** one variable and other predicates are unary.
- **Parity function -** returns one if an even number of input are one.
- **Majority function -** returns one if greater than half of its input is one.

The truth-table contains two n rows, for the reason that every input case is defined by n-attributes. If it receives two n bits to describe the function, then there are twenty two n dissimilar functions on ‘n’ Attributes.

Inducing Decision Trees from Examples

An instance for a Boolean-decision tree contains of a vector of giving attributes, 'X' and a single-Boolean resultant value is *'y'.*

DECISION TREE LEARNING algorithm is displayed in the bellow Figure:



```
function DECISION-TREE-LEARNING(examples, attribs, default) returns a decision tree
  inputs: examples, set of examples
          attrzbs, set of attributes
          default, default value for the goal predicate

  if examples is empty then return default
  else if all examples have the same classification then return the classification
  else if attrzbs is empty then return MAJORITY-VALUE(examples)
  else
      best ← CHOOSE-ATTRIBUTE(attribs, examples)
      tree ← a new decision tree with root test best
      m ← MAJORITY-VALUE(examples)
      for each value $v_i$ of best do
          $examples_i$ ← {elements of examples with best = $v_i$}
          subtree ← DECISION-TREE-LEARNING($examples_i$, attribs − best, m)
          add a branch to tree with label $v_i$ and subtree subtree
      return tree
```

Figure: Decision tree Learning Algorithm

The performance of learning algorithm depends on,

- Choosing attribute tests.
- Data set for training and testing without noise and over lifting.

Selecting Attribute Tests

The method utilized in decision-tree learning for choosing attributes is planned to reduce the depth of the last tree. The plan is to choose the attribute that moves as distant as available near giving a precise arrangement of the examples. A best attribute splits the examples into groups, which are all negative or positive.

CHOOSE-ATTRIBUTE Function

The measure could contain its highest value whenever the attribute is best and its least value whenever the attribute is of not usable at all.

In common, if the available solution ‘vi’ contain probabilities “P(vi)”, which the knowledge content ‘I’ of the real solution is specified by

$$I(P(v_1), \ldots, P(v_n)) = \sum_{i=1} -P(v_i) \log_2 P(v_i)$$

Evaluating the Work of the Learning Algorithm

A learning algorithm is best, if it gives suggestion that perform a better work of estimating the categorizations of not seen examples. It is most suitable to accept the following methods:

1. Gather a huge group of examples.
2. Separate it into 2 not joint groups: the **training group** and the **test group.**
3. Apply the learning algorithm to the training group, producing theories.
4. Determine the percentage of instances in the test group, which is properly arranged by ‘h’.
5. Replicate steps two to four for dissimilar sizes of training groups and dissimilar at random chosen training groups of all size.

The outcome of this process is a dataset that may be prepared to provide the average forecast excellence as a function of the size of the training group. This function may be designed on a chart, offering what is termed the **learning-curve** for the algorithm on the appropriate field.

Noise and Over Fitting

Noise: more than two examples with the similar explanation in attributes but dissimilar classifications.

Over lifting: When there is a large group of probable hypothesis tree result with meaningless regularity in the data. To overcome the problem of over lifting decision tree pruning or cross-validation can be applied.

Improving the Appropriateness of Decision Trees

The decision tree induction in applied on variety of problems, to address the following issues:

1. Missing data.
2. Multi-valued attributes.
3. Numerical valued and Continuous input attributes.
4. Repeated value output attributes.

4.5. Ensemble Learning

The intention of **ensemble learning** process is to choose an entire group, or **ensemble** of hypotheses from joining their predictions and the hypothesis space.

The most popular ensemble learning methods are:

1. Boosting
2. Bagging

Boosting: Most generally utilized ensemble approach is known as **boosting.** To know how it process, "WEIGTHTED-TRAINING" require initially defining the thought of a **weighted training set.** In such a training group, every example has a related weight "wJ>0". The maximum weight of an instance, the greater is the import connected to it throughout the studying of a hypothesis. It is sincere to change the learning algorithms. We contain observed so far to work with weighted training groups; Boosting begins with w=1 for every examples (i.e., a general training group). From this group, it produces the primary hypothesis-*hl.* This hypothesis can analyze a few of the training examples properly and a few improperly. We could similar to the later hypothesis to perform well on the misclassified instances, so we maximize their weights while minimizing the weights of the properly classified examples. From this latest weighted training group, we produce hypothesis-*h2.* The procedure carries on in this method upto we have produced 'M' hypotheses, where 'M' is an entry to the boosting-algorithm. The last ensemble hypothesis is a weighted popular grouping of all the 'M' hypotheses, all weighted-corresponding to how can it worked on the training group.

```
function ADABOOST(examples, L, M) returns a weighted-majority hypothesis
   inputs: examples, set of $N$ labelled examples $(x_1, y_1), \ldots, (x_N, y_N)$
           L, a learning algorithm
           M, the number of hypotheses in the ensemble
   local variables: w, a vector of $N$ example weights, initially $1/N$
                    h, a vector of $M$ hypotheses
                    z, a vector of M hypothesis weights

   for m = 1 to M do
      h[m] ← L(examples, w)
      error ← 0
      for $j = 1$ to $N$ do
         if $h[m](x_j) \neq y_j$ then error ← error + w[j]
      for $j = 1$ to $N$ do
         if $h[m](x_j) = y_j$ then w[j] ← w[j] . error/(1 − error)
      w ← NORMALIZE(w)
      z[m] ← log (1 − error)/ error
   return WEIGHTED-MAJORITY(~z)
```

Figure: The ADABOOST Alternative of the Boosting Process for Ensemble Learning

4.6. Knowledge in Learning

This easy knowledge free image of inductive learning continued upto the before time 1980s.

The recent method is to propose agents that previously recognize a bit and attempting to study a few most these cannot noise similar to an extremely deep insight, but it creates entirely a different way we plan the agents. It can also contain a few relevance to our hypothesis regarding how knowledge itself working. The common thought is displayed schematically in the bellow figure.

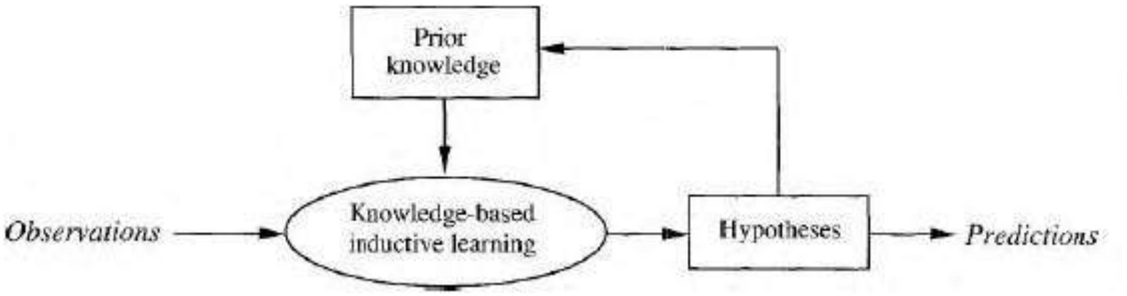


Figure: A cumulative-learning Procedure Utilizes, and Inserts to its Store of Background Information Over Time

4.7. Logical Grouping of Learning

Pure inductive learning is a procedure of discovering a theory that acknowledges with the noticed instances. We focus on this description to the case wherever the theory is characterized by a group of logical statements.

Examples of Hypotheses

The restaurant-learning problem: Learning a regulation for selecting regardless if to wait for a table. Instances are explained by an **attributes** like FrilSat, Bar, Alternate, and so on.

For example, a decision-tree declares that the target estimate is correct of an object if any branches noted to *true* is fulfilled. Every hypothesis forecast that certain group of instances; namely, these which gratify its applicant meaning shall be instances of the target estimate. This group is known as the **extension** of the estimate. 2 theories with dissimilar add-ons are then logically incompatible with one another, for the reason that they oppose on their forecasts for no less than 1 example. Logically, it is precisely similar to the decision rule of conclusion. We may describe inductive-learning in a logical situation as a procedure of regularly removing theories, which are incompatible with the instances, reducing the probabilities.

Current Best Hypothesis Search (CBHS)

The thought beyond the **CBHS** is to alter it as latest examples appear in order to continue consistency and to continue a single hypothesis. The hypothesis states it could be negative but it is really positive. The expansion of the hypothesis should be improved to contain it. It is known as **generalization;** the expansion of the hypothesis should be reduced to keep out the instance. It is known as **specialization.**

Define the **CURRENT-BEST LEARNING** algorithm:

```
function CURRENT-BEST-LEARNING(examples) returns a hypothesis

  H ← any hypothesis consistent with the first example in examples
  for each remaining example in examples do
      if e is false positive for H then
          H ← choose a specialization of H consistent with examples
      else if e is false negative for H then
          H ← choose a generalization of H consistent with examples
      if no consistent specialization/generalization can be found then fail
  return H
```

Figure: The current-best hypothesis learning algorithm. It looks for a reliable hypothesis and backtracks whenever no reliable generalization/specialization may be detected.

We described specialization and simplification as process that alter the **expansion** of a hypothesis. Currently we required to decide precisely how they may be executing as syntactic procedures that alter the person description connected with the hypothesis, so that a program may take them away. This is completed by initial mentioning that simplification. Specialization is also logical associations among hypotheses. If hypothesis-HI with explanation 'C1' is a simplification of hypothesis-H2 with explanation 'C2', next we should contain *t*xC2 (x)=+ C l(x). Then in order to build a simplification of '*Hz*', we just required discovering an explanation 'Cl', which is logically implicated by 'C2'. This is simply completed. For example, if C2(x) is an *Alternate(x)Patrons(x,Some),* next one probable simplification is specified by *Cl(x) Patrons(x, Some).* This is known as dropping **conditions.**

In such situations, the program should retrack to an earlier selection point. The recent better learning algorithm and its alternates have been utilized in several machine learning methods, beginning with PatrickWinston's(1970) arch-learning program.

With a huge count of instances and a huge space, though, a few complications happen:

1. The search procedure can involve a big deal of back-tracking.
2. Inspecting all the earlier instances another time for all change is highly expensive.

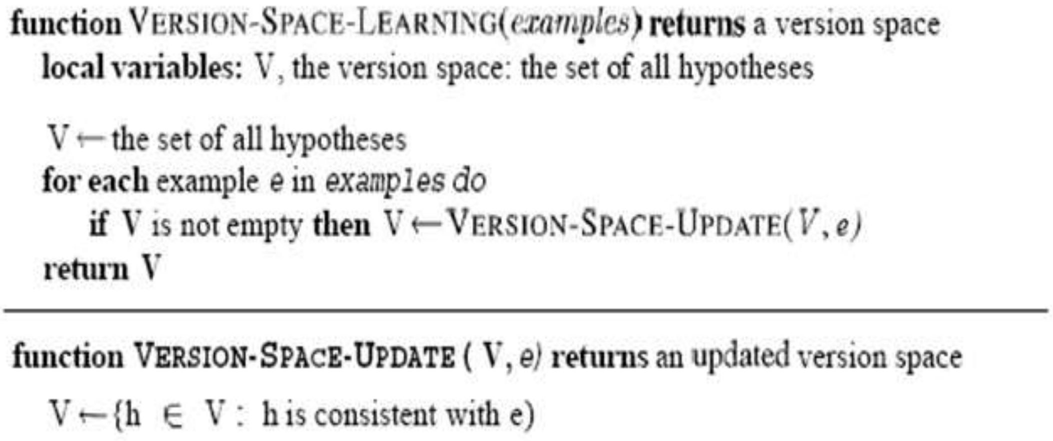
Hypothesis space may be a twice as exponentially big place.

Smallest Commitment Search

Retracking happens for the reason that the present better hypothesis method has to select a specific hypothesis as its better estimate even if it could not contain sufficient records still to be of the selection.

$$H1 \vee H2 \vee H3 \ldots \vee Hn.$$

The group of hypotheses continuing is known as the **version-space,** and the learning algorithm is known as the candidate elimination algorithm or version space learning algorithm.



```
function VERSION-SPACE-LEARNING(examples) returns a version space
  local variables: V, the version space: the set of all hypotheses

  V ← the set of all hypotheses
  for each example e in examples do
     if V is not empty then V ← VERSION-SPACE-UPDATE(V, e)
  return V
```

```
function VERSION-SPACE-UPDATE ( V, e) returns an updated version space
  V ← {h ∈ V : h is consistent with e)
```

Figure: The Version Space Learning Algorithm

4.8. Explanation based Learning (EBL)

EBL is a process for extract common rules from separate review. The method of **memorization** has extended been utilized in computer science to accelerate programs by storing the outcomes of computing. The essential thought of memo functions is to build up a data base of output/input pairs; whenever the function is called, it initially verifies the data base to observe regardless if, it may avoid resolving the problem from scrape. EBL obtains a best deal after, by generating common rules that wrap a complete class of instances.

Extracting General Rules from Examples

The constraint can include the Background information, in inclusion to the Hypothesis and the noticed Classifications and Explanations. In the instance of lizard warming, the caveman simplify by describing the victory of the pointed attach: it supports the lizard whilst putting the hand aside from the fire. From this clarification, they may collect a common rule: any sharp, stiff, long object may be utilized to soft-bodied, warm little edibles. This type of simplification procedure known as **EBL.**

"The agent cannot really study anything exactly latest from the case".

We may simplify this instance to occur with the residual limitation:

Background Hypothesis explanations categorizations.

In ILP systems, earlier awareness plays 2 key functions in decreasing the difficulty of learning:

1. Any hypothesis created should be reliable with the earlier awareness also as with the latest inspections; the successful hypothesis space range is decreased. To contain just these hypotheses, which are reliable with what is previously recognized?

2. For any specified group of inspections, the range of the hypothesis necessary to build a clarification for the inspections may be a lot decreased, for the reason that the previous awareness can be accessible to assist the latest rules in clarifying the inspections. The slighter the hypothesis, the simpler it is to discover.

The primary **EBL** procedure performs as follows:

1. Specified an instance, build evidence that the target predicate executes to the instance by utilizing the existing background awareness.

2. In parallel, build a simplified evidence tree for the mobilized target by applying the similar conclusion steps in the primary evidence.

3. Build a latest rule whose left side contains the ranges of the evidence tree and whose right side is the mobilized target (once using the required binding from the derived evidence).

4. Leave any situations, which are true anyway of the variables values in the target.

Enhancing Efficiency

There are 3 things concerned in the study of efficiency increases from EBL:

1. Inserting huge number of rules may drop the logic procedure; for the reason that the deduction method should yet to verify these rules still in cases wherever they perform disturbance give up a result. In other way, it raises the **Branching Factor** in the search space.
2. To recompense for the drop in logic, the resulting rules should propose important raises in velocity for the cases, which they make wrap. These raises come regarding mostly for the reason that the derivative rules keep away from last points otherwise it could be taken, because they cut down the evidence itself.
3. Obtained rules could be as common as probable, SCI, which they use to the biggest probable group of cases.

By simplifying from previous instance problems, EBL builds the database more proficient for the variety of problems, which it is sensible to imagine.

4.9. Learning Using Relevant Information

Statements state a strict form of connection: specified language, nationality is completely decided. Language is a meaning of nationality. These statements are known as **determinations** or **Functional Dependencies.** They happen so generally in definite varieties of requests (e.g., describing data base design) that a particular statement applied to draft them. We accept the document of Davies(1985):

Language (x, 1)+Nationality (x,n)

Determining the Hypothesis Space

Determinations state an enough basis language from which to build hypotheses relating to the goal predicate. This sentence may be confirmed by viewing that a specified resolution is rationally the same to a sentence that the proper explanations of the goal predicate is one of the groups of each explanation expressible by applying the predicates on the left part of the resolution.

Learning and Using Relevance Information

```
function MINIMAL-CONSISTENT-DET($E, A$) returns a set of attributes
  inputs: E, a set of examples
          $A$, a set of attributes, of size n

  for $i \leftarrow 0, \ldots,$ n do
     for each subset $A_i$ of $A$ of size $i$ do
        if CONSISTENT-DET?($A_i$, E) then return $A_i$
```

```
function CONSISTENT-DET?($A$, E) returns a truth-value
  inputs: A, a set of attributes
          E, a set of examples
  local variables: H. a hash table

  for each example e in E do
     if some example in H has the same values as e for the attributes $A$
        but a different classification then return false
     store the class of e in H, indexed by the values for attributes $A$ of the example e
  return true
```

Figure: Algorithm for Discovering a Smallest Consistent Determination

4.10. Inductive Logic Programming(ILP)

ILP joins inductive process with the capacity of initial order representation, focusing on the representation of hypothesis as logic program. It has increased status for 3 causes. Initially ILP proposes an accurate method to the common data based inductive- learning problem. Second it proposes entire algorithms for inducing common initial order hypothesis from instances that should then learn effectively in fields wherever the attribute-based algorithms are tough to execute.

Example

The common record based-induction problem is to "solve" the consequence restriction

$$\textit{Background A Hypothesis A Descriptions} = \textit{classifications}$$

For the unspecified theories, specified the Background awareness and instances defined by Classifications and Descriptions.

The equivalent explanations are as follows:

Father (Philip, Charles) Father (Philip, Anne) . . .

Mother (Mum, Margaret) Mother (Mum, Elizabeth) . . .

Married (Diana, Charles) Married (Elixabeth, Philip) . .

Male (Philip) Male (Charles) . . .

Female (Beatrice) Female (Margaret:) . . .

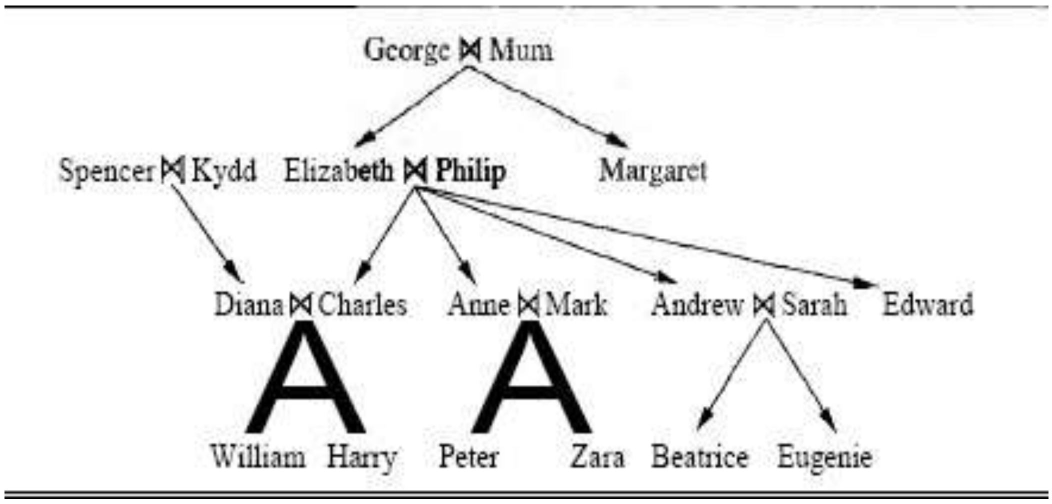


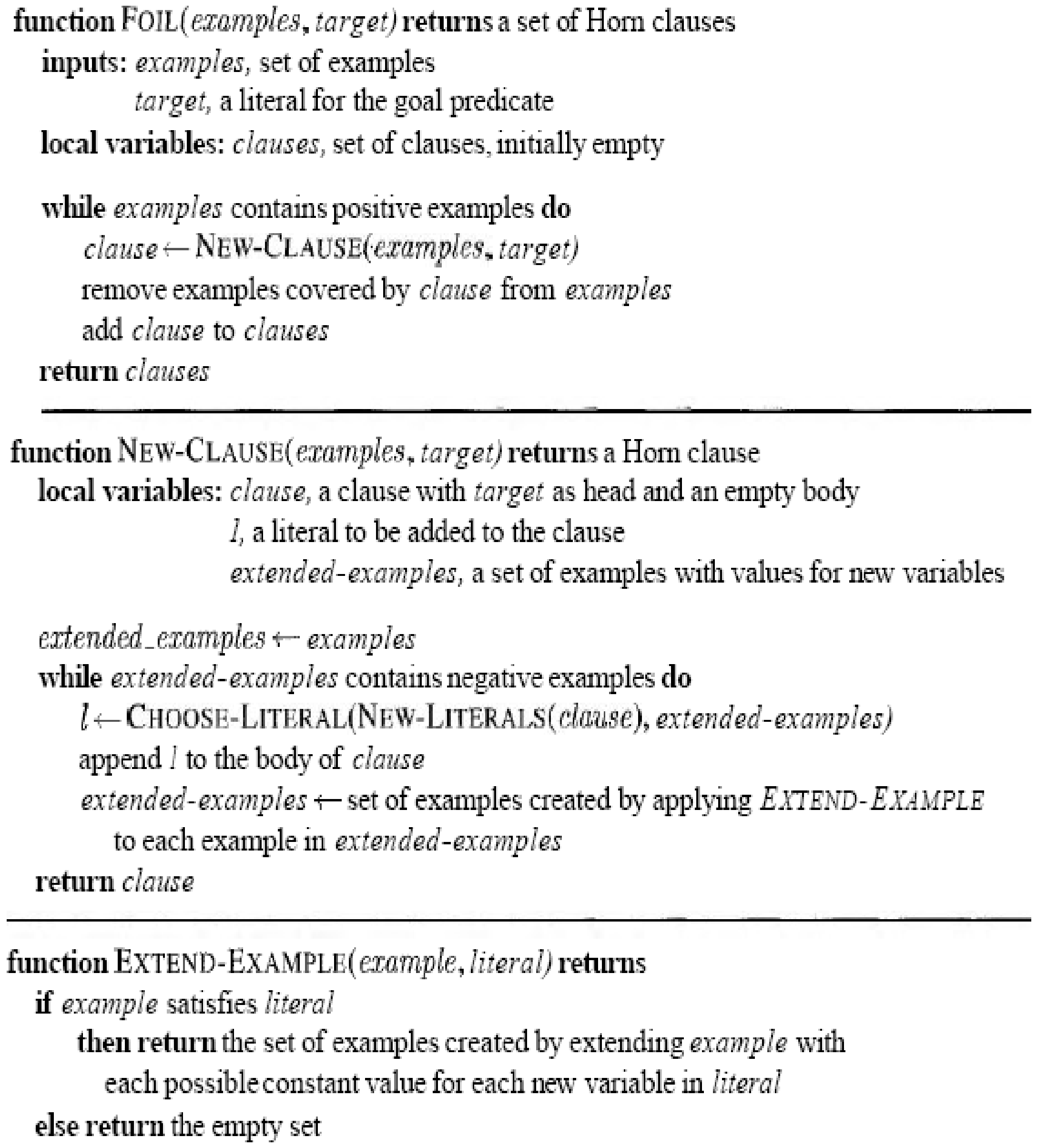
Figure: A Typical Family Tree

The goal of an Inductive Learning program is to occur with a group of statements for the theories such that the consequence limitation is fulfilled. Assume, for the instant, which the agent does not have background awareness: Background is unfilled.

Attribute Based Learning algorithms are unqualified of studying related predicates*.*

Top Down Inductive Learning Approaches

The initial method to ILP performs by beginning with a extremely common rule and regularly specialising it, so that it fix the information. It is basically what occurs in Decision Tree studying, wherever a Decision Tree is slowly developed upto it is reliable with the considerations. To perform ILP, we apply first-order literals rather than of attributes, and the theories is a group of sections rather than of a Decision Tree.



```
function FOIL(examples, target) returns a set of Horn clauses
  inputs: examples, set of examples
          target, a literal for the goal predicate
  local variables: clauses, set of clauses, initially empty

  while examples contains positive examples do
      clause ← NEW-CLAUSE(examples, target)
      remove examples covered by clause from examples
      add clause to clauses
  return clauses
```

```
function NEW-CLAUSE(examples, target) returns a Horn clause
  local variables: clause, a clause with target as head and an empty body
                   l, a literal to be added to the clause
                   extended-examples, a set of examples with values for new variables

  extended_examples ← examples
  while extended-examples contains negative examples do
      l ← CHOOSE-LITERAL(NEW-LITERALS(clause), extended-examples)
      append l to the body of clause
      extended-examples ← set of examples created by applying EXTEND-EXAMPLE
          to each example in extended-examples
  return clause
```

```
function EXTEND-EXAMPLE(example, literal) returns
  if example satisfies literal
      then return the set of examples created by extending example with
          each possible constant value for each new variable in literal
  else return the empty set
```

Figure: Drawing of the FOIL Algorithm for Learning Groups of First-order

Inductive Learning with Opposite Inference

The secondary major way to ILP involves reversing the common inferable evidence procedure.

Opposite declaration is based on the monitoring that if the instance Classification goes after from *Background* A *Hypothesis A Descriptions,* next one should be capable to confirm this reality by declaration. ***Twelve*** numbers of methods to educating the search have been attempted in applying ILP methods:

1. Too many selections should be removed
2. The proof strategy can be restricted.
3. The representation language can be restricted,
4. Inference may be completed with model inspection instead theorem confirming.
5. Inference may be completed with ground proposal sections instead in first-order logic.

Preparing Selections with Inductive-logic-programming

An inverse-resolution process, which reverses an entire resolution approach, it is a standard, an entire algorithm for studying first-order hypothesis.

4.11. Statistical Learning Methods

Bayesian learning easily computes the probability of all hypotheses, specified the information, and prepares forecasts on that basis. That is, the forecasts are prepared by applying *each* hypothesis, weighted by their possibilities instead by utilizing only a single "better" hypothesis.

The key measures in the Bayesian method are the hypothesis earlier P(hi*),* and the possibility of the information beneath all hypotheses, P *(* dJhi *).*

4.12. Learning with Entire Information

A statistical learning process starts with the easiest work: parameter learning with entire information. A parameter learning work contains discovering the arithmetic parameters for a possibility model, whose configuration is permanent.

Maximum Probability Parameter Learning: Discrete Models

Actually, while we have placed out a single standard process for most probability parameter learning:

1. List a declaration for the possibility of the information as a meaning of the parameter(s).
2. List the copied of the log possibility with regard to all parameter.
3. Discover the values of parameter such the copies are 0.

An important problem with most probability studying in common: -*when the information group is less sufficient that a few actions have not yet been watched for example, no cherry confections the most probability theory allots zero likelihood to those actions".*

The most significant point is that, *with entire records, the most probability parameter learning problem for a Bayesian system disintegrates into independent learning problems, single for every parameter.* The subsequent point is the parameter esteems for a variable, specified its parents are only the watched recurrences of the values of a variable for every configuration of the values of a parent. As in the past, we should be mindful to keep away from 0s when the records group is less.

Naive Bayes Models

Possibly the regular Bayesian system method utilized in AI is the **naïveBayes** method. In this method, the variable of a class-C is the root and the variables of an attribute-*Xi* are the ranges. The method is "naive7' on the grounds that it accepts that the attributes are restrictively free of one another, specified the class.

Most Possibly Parameter Learning Continuous Methods

Continuous Probability method acting as the **linear_Gaussian** method. The standards for most possibility learning are equal to the separate case. Let us start with an easy case: parameter learning of a Gaussian_Density function on one variable. That is the information is created as well as follows:

$$P(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

The parameter of the mean of this model is 'Y' and the standard-deviation is '*a'.*

The weight (yj-(B1 xj+02)) is the **mistake** for (zj,yj), which is, the dissimilarity among the real value "y_j" and the expected value (1xj+$2) - SOE is the popular **Sum of Squared Errors(SSE).** It is the weight, which is reduced by the regular **LinearRegression** method. At present we should know why reducing the SSE specifies the most probability continuous process, *given that the information is created with GaussianNoise of* per*manent variation.*

Bayesian Parameter Learning

The Bayesian method to learning parameter positions a theory earlier above the probable parameters values and improves this sharing as record enter. This expression of guess and learning generates this obvious that Bayesian-learning needs no more "standards of learning". In addition, *there is a real meaning, only a single algorithm for learning,* i.e., the algorithm of inference for BayesianNetworks.

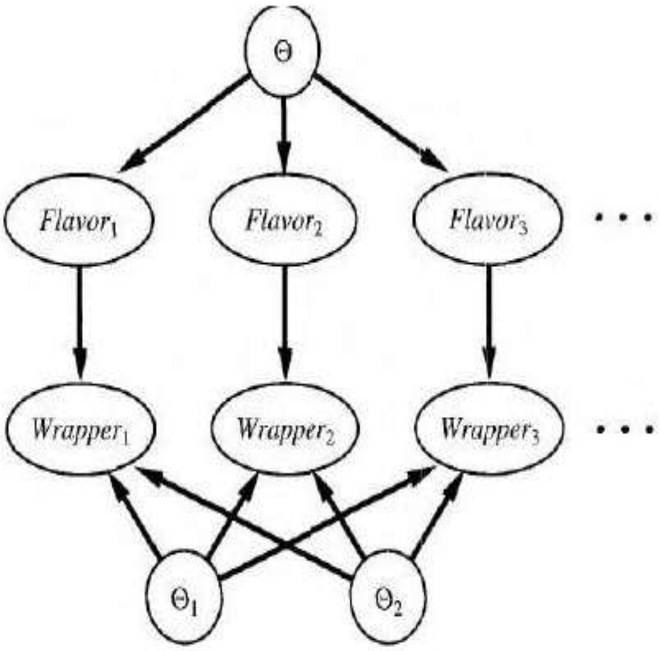


Figure: Bayesian network, which matches to a Bayesian learning procedure. Posterior distributions for the parameter variables e$_0$, e$_1$, and *e$_2$* should be deduced from their earlier distributions and the proof in the *Flavor*, and the *Wrapper variables*.

There are 2 different approaches for determining whenever best designs have been selected.

The primary is to check regardless if the provisional freedom statements implied in the design are really fulfilled in the record.

4.13. Learning with Hidden Variable

Several genuine problems contain unseen variables (sometimes termed as hidden variables) that are unrecognizable in the record that are accessible for learning.

Ex: health data frequently contain the noticed symptoms, the treatment apply, and possibly the result of the treatment, however they rarely have a straight examination of the illness itself.

Thus, *latent variables should noticeably decrease the number of parameters necessary to state a Bayesian network.*

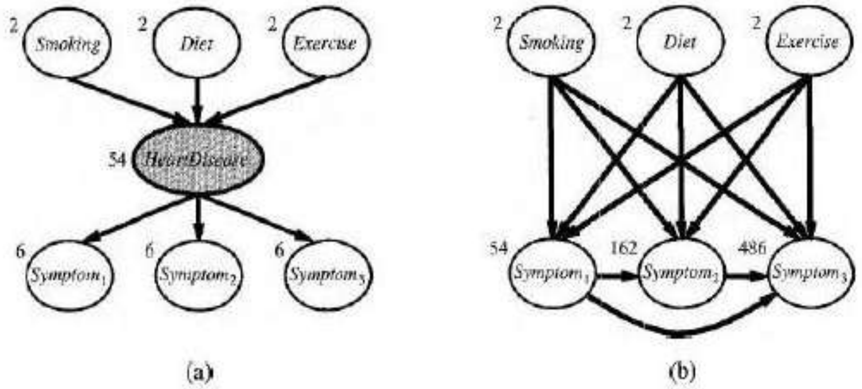


Figure: **(a) an easy investigative network for heart illness, which is supposed to be a unseen variable. Every variable has 3 probable values and is tagged with the number of independent-parameters in its conditional sharing; the entire number is 78. (b) The corresponding network with *Heart illness* detached. Note that the sign variables are no long-lasting conditionally autonomous specified their parents. This network needs "708" parameters.**

Unsupervised Clustering: Learning Combinations of Gaussians

Unsupervised clustering is the difficulty of discriminating several kinds in a group of elements. The difficulty is unsupervised for the reason that the group tags are unspecified.

Clustering assumes that the records are created from a **mixture sharing** 'P'. Such sharing contains '*k'* **elements,** all of which is a sharing of itself.

A data position is created by initially selecting a component and then creating an easy from that element. C is a random-variable indicate the element, with the value 1*K;* afterward the mixture sharing is specified:

$$P(\mathbf{x}) = \sum_{i=1}^{k} P(C=i)\ P(\mathbf{x}|C=i)\ ,$$

where 'x' mentions, the attributes value for a data position.

For the Gaussians mixture, we initialise the mixture approach parameters randomly and after that repeat the bellow 2 steps:

A. **E-step**: calculates the probabilities *pij=P(C=*i*lxj),* the probability that datum *"xj"* are created by an element 'i'. By Bayes' rule, we contain *P (C=**I**) and pij=aP(xj*IC=*i).* The expression *P (C=*I) is only the weight parameter for the 'ith' Gaussian and the expression *P(xj*IC=i) is only the probability of the 'ith' Gaussian at *"xj"*. Describe *pi=Cjpij.*

B. **M-step**: calculate the latest covariance, mean and element as follows:

$$\mu_i \leftarrow \sum_j p_{ij}\mathbf{x}_j / p_i$$

$$xi+ Cpii\ (x j - pi)\ (xi - P i)\ / p i\ W i + p i .$$

The E-step, should be observing as calculating the values predictable is *'p'* of the unseen **pointer-variables** *'Z'*, where *'Z'* is one if daturn-**x3** are created by the 'ith' element and zero or else.

Learning Bayesian Networks with Hidden Variables

The message from this instance is that *the constraint upgrades for Bayesian Network learning with variables of unseen are openly accessible from the outcomes of deduction on every instance. In addition simply* local *posterior-probabilities are required for every constraint.*

Learning of Hidden Markov Models (HMMs)

Our last program of EM includes learning the changeover possibilities in HMMs. This model should be presented by a Dynamic Bayes Network with one distinct situation variable. Every data position includes of a study series of limited length, so the problem is to study the changeover possibilities from a group of study series.

The Common Structure of the EM Algorithm

We noticed multiple cases of the EM algorithm. All includes calculating predictable values of the variables of variables for every case and then calculating the constraints, utilizing the values of predictable as if they are the values of experiential. Let '**x**' be the entire values of experiential in each of the case let '**Z**' indicate each of the unseen variable for each of the case and let **eight** be the entire elements for the probability method. The EM algorithm is:

$$\boldsymbol{\theta}^{(i+1)} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \sum_{\mathbf{z}} P(\mathbf{Z}=\mathbf{z}|\mathbf{x}, \boldsymbol{\theta}^{(i)}) L(\mathbf{x}, \mathbf{Z}=\mathbf{z}|\boldsymbol{\theta})$$

This expression is the EM algorithm in a shell of nut.

Learning Bayes Network Structures with Hidden Variables

In the easiest instance, the unseen variables are indexed besides the experimental variables; while their values aren't experimental, the learning algorithm is informed that they be present and should discover a position for them in the grid formation. The final approach should be applied by including latest change selections in the structural search: additionally to changing connections, the algorithm should insert or remove an unseen variable or alter its parity.

One probable development is the so-referred to as **structure** EM algorithm that functions in greatly the similar method as regular parametric EM excluding. The algorithm should upgrade the design in addition to the considerations.

4.14. Instance based Learning

Parametric Learning systems are frequently effective and easy, but suppose a selected constrained family of methods frequently restraint what is occurrence inside the actual world, from where the records appear. In comparison to nonparametric and parametric learning process permit the hypothesis difficulty to grow with the records.

Two extremely easy relations of nonparametric **memory-based learning** or **example-based learning** process, so known as for the reason that they make hypotheses honestly from the preparation examples itself.

Nearest Neighbor Models (NNM)

The major concept of **NNM** is that the property of every specific entry point 'x' is probable to be equal to these of objects inside the neighborhood of x.

The k-NNM learning algorithm is quite easy to apply, needs slight in the method of altering, and frequently executes very fine. This is a great factor to attempt initially on a latest learning problem. For huge record groups, still, we need an competent method for discovering the nearby neighbors of a question factor 'x' easily computing the gap to each factor might get lengthy. A change of inventive strategies had been planned to build this step competent by pre processing the preparation records. Unfortunately, maximum of those strategies do longer size properly with the measurement of the space (i.e., the characteristics count).

Kernel Models

In a kernel model, we saw all preparation examples as creating a small weight function of its own. The weight guesses as an entire is simply the distributed addition of the whole small kernel functions. A preparation example at ‘xi’ can create a kernel_function K(x,xi) that allocates a possibility to every factor-x inside the space. Therefore, the weight

guesses is one. P (x)=*NC*~(xxi,). The kernel function commonly depends just on the *distance-**D(x,xi)*** from ‘***x’*** to the example ‘***xi’.*** The great trendy kernel function is the Gaussian. For minimalism, we will anticipate spherical Gaussians with general difference ‘w’ next to all axes, i.e.

$$K(\mathbf{x}, \mathbf{x}_i) = \frac{1}{(w^2\sqrt{2\pi})^d} e^{-\frac{D(\mathbf{x},\mathbf{x}_i)^2}{2w^2}} ,$$

The dimensions count in ‘x’ is ‘d’.

Supervised learning with kernels is completed by picking a *weighted* grouping of *each of* the forecasts from the preparation examples.

4.15. Neural Networks

A neuron is one of the cells in the brain whose major task is the dissemination, procedure, and collection of electrical signals. The brain's data working capability is an idea to appear initially from *networks* of like neurons. For this cause, a few of the initial ‘**A1’** process tried to generate Artificial **Neural Networks.** (Additional names for the area contain **neural computation, parallel distributed processing,** and **connectionism.)**

Elements in Neural Networks

Neural networks are collected of nodes or **elements** linked by directed **links. A** link from element ‘j’ to element ‘i’ provide to spread the **activation-**a_j from ‘**j**’ to ‘*i’.* Every connection just had a numerical **weight-**W_j linked with this that decides sign and power of WEIGHT the link. Every element ‘i’ initially calculates a weighted addition of its entries: ‘*N*’ Next, it gives an **activation function-**g to this addition to get the result: Note that we can incorporated a **bias-weight-***Wo, ‘i’* linked to a permanent entry a_o=*-1.*

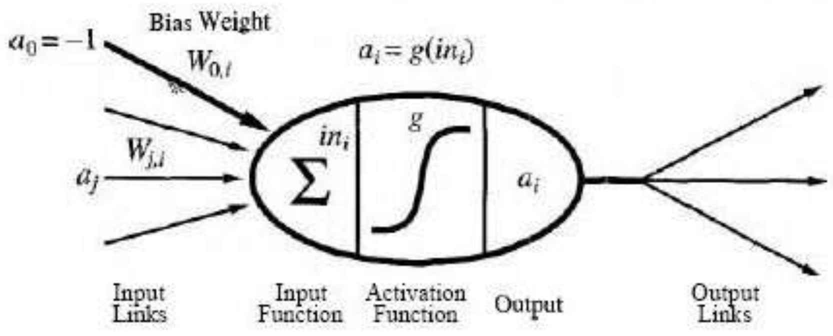


Figure: An Easy Mathematical Approach for a Neuron

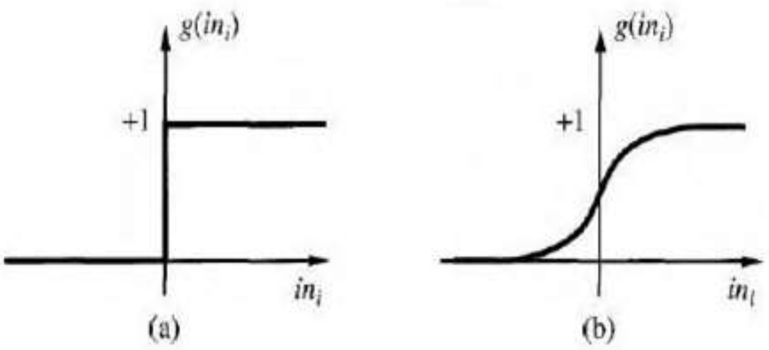


Figure 4.13 (a) The threshold activation-function, which yields '1' when the input is positive value and zero if not. (b) The sigmoid-function 1 /(1+*e* -").

Figure demonstrates how the Boolean tasks NOT, AND and OR should be signified by threshold elements with appropriate weights. It is essential for the reason that it denotes we should utilize those elements to generate a network to calculate any Boolean task of the entries.

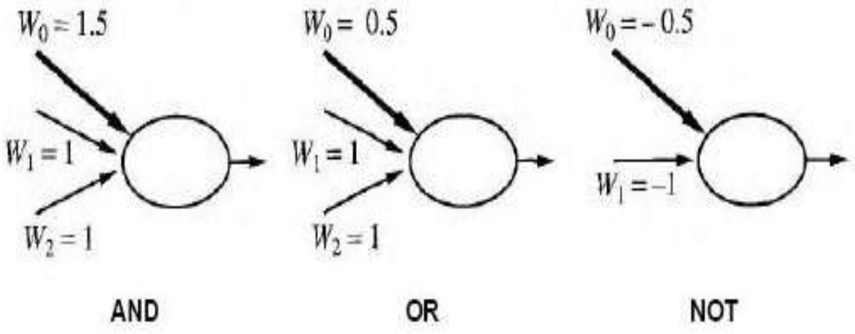


Figure: Elements with a threshold activation-function should work as logic-gates, specified proper input and bias-weights.

Network Structures

There are 2 major types of Neural Network structures: acyclic or feed forward networks and cyclic or recurrent networks. An acyclic system signifies a task of its present input; therefore, it had no internal status excluding the weights itself. A repeated network, any further way, feeds its results back into one its own inputs.

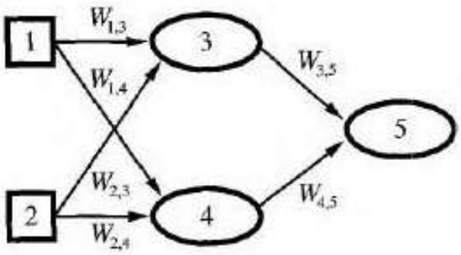


Figure: An easy neural-network with 2 inputs, one unseen layer of two elements, and one output.

A neural-network should be utilized for regression or classification.

Feed-forward structures are regularly ordered in **layers,** so every element obtains input just from elements in the instantly earlier layer.

Single Layer Feed Forward Neural Networks (Perceptrons)

A network with each input linked directly to the outputs is known as a **Single Layer Neural Network,** or a **perceptron** network. While every output element is autonomous of the furthers, every weight influences just 1 of the outputs.

In common, *threshold perceptrons should signify just linearly divisible functions.*

In spite of their partial communicative power, threshold perceptrons had a few benefits.

In specific, *there is an easy learning algorithm, which can specify a threshold perceptron to a few limitedly divisible preparation groups.*

The entire algorithm is exposed in Figure

```
function PERCEPTRON-LEARNING(examples, network) returns a perceptron hypothesis
  inputs: examples, a set of examples, each with input x = $x_1, \ldots, x_n$ and output $y$
          network, a perceptron with weights $W_j$, $j = 0 \ldots n$, and activation function $g$

  repeat
     for each e in examples do
        $in \leftarrow \sum_{j=0}^{n} W_j \, x_j[e]$
        $Err \leftarrow y[e] - g(in)$
        $W_j \leftarrow W_j + \alpha \times Err \times g'(in) \times x_j[e]$
  until some stopping criterion is satisfied
  return NEURAL-NET-HYPOTHESIS(network)
```

Figure: The grade descending learning algorithm for perceptrons, guessing a diverse activation function 'g'.

Observe that *the weight upgrade vector for highest possibility studying in sigmoid-perceptrons is basically the same to the upgrade vector for squared error reduction.*

Multilayer Feed-forward Neural Networks

The benefit of inserting unseen layers is that it increases the space of hypotheses that the network should signify.

Learning-algorithms for multi-layer systems are equal to the perceptron-learning algorithm. The back transmission procedure should be outlined as given:

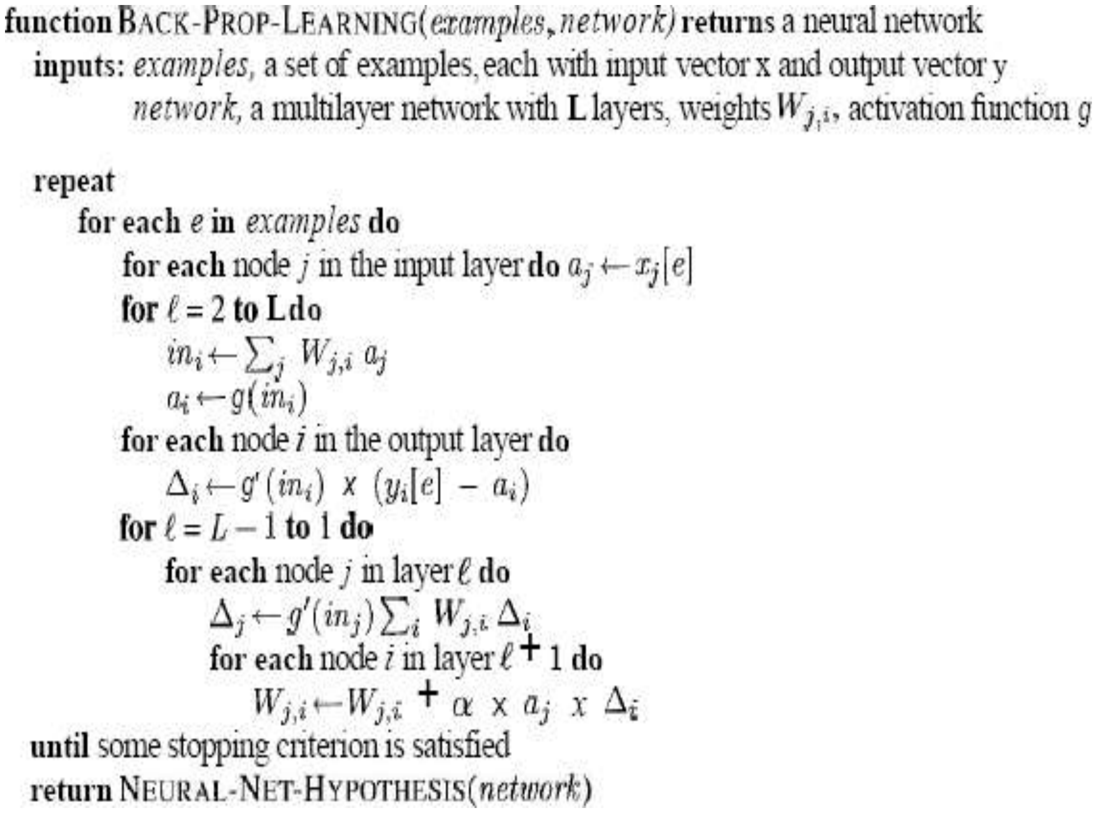
Calculate the ‘A’ values for the outcome elements by utilizing the experimental error.

Beginning with outcome layer, replicate the subsequent for every layers in the network, till the initially unseen layer is attained:

a) Transmit the ‘A’ values backwards to the earlier layer.

b) Upgrade the weights among the 2 layers.

The complete algorithm is exposed in Figure



```
function BACK-PROP-LEARNING(examples, network) returns a neural network
  inputs: examples, a set of examples, each with input vector x and output vector y
          network, a multilayer network with L layers, weights $W_{j,i}$, activation function $g$

  repeat
     for each $e$ in examples do
        for each node $j$ in the input layer do $a_j \leftarrow x_j[e]$
        for $\ell = 2$ to L do
           $in_i \leftarrow \sum_j W_{j,i}\, a_j$
           $a_i \leftarrow g(in_i)$
        for each node $i$ in the output layer do
           $\Delta_i \leftarrow g'(in_i) \times (y_i[e] - a_i)$
        for $\ell = L - 1$ to 1 do
           for each node $j$ in layer $\ell$ do
              $\Delta_j \leftarrow g'(in_j) \sum_i W_{j,i}\, \Delta_i$
              for each node $i$ in layer $\ell + 1$ do
                 $W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i$
  until some stopping criterion is satisfied
  return NEURAL-NET-HYPOTHESIS(network)
```

Figure: The Back Transmission Algorithm for Learning in Multi-layer Networks

Learning Neural Network Structures

We need to observe networks that are not entirely linked, and then we require locating a few valuable search methods through the huge space of probable link topologies. The best Brain Damage algorithm starts with a entirely linked network and eliminates links from it. Once the network is skilled for the primary time, a data-theoretic method recognizes a best choice of links that should be removed. The network is re-trained, and if its presentation had not minimized next the procedure is do again. Additionally to eliminating links, it is just likely to eliminate elements that are not giving more to the outcome.

4.16. Reinforcement Learning

The problem is *without a few feedbacks regarding what is fine and what is poor; the agent shall contain no grounds for selecting which go to build.* The agent requires knowing that a little fine has occurred when it succeeds and that a little poor has occurred when it drops. This type of comment is known as a **reward** or **reinforcement.**

The job of **Reinforcement Learning** is to apply experimental rewards to study a best strategy for the environment.

Three of the agent designs:

a. **Q-learning** agent studies a Q-function or an **action-value** function, specifying the estimated value of obtaining a specified operation in a specified condition.
b. **Utility-based agent** studies a utility task on positions and applies it to choose operations that increase the estimated result utility.
c. **Reflex agent** studies a strategy, which maps directly from status to operations.

Kinds of reinforcements learning.

1. Active reinforcement learning
2. Passive reinforcement learning

4.17. Passive Reinforcement Learning

To keep objects easy, we begin with the instance of a PassiveLearning agent by utilizing a status based presentation in a completely evident situation. In PassiveLearning, the agent's strategy-T is permanent: in status-s, it forever performs the operation (s), its target is just to study how the best strategy is to study the Utility_Function UT(s)?

The major dissimilarity is that the PassiveLearning agent is unrecognized the **transition_Model** T(s,a,s'), that gives the likelihood of success states' from states later than performing action-a or does not recognize the **reward_Function** R(s) that gives the reward for every status.

The utility is described to be the predictable addition of (not counted) rewards gained if the strategy is followed. This expression is shown as:

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^{t} R(s_t) \mid \pi, s_0 = s\right]$$

Direct Utility Estimation (DUE)

A simplest process for **DUE** was innovated in the late 1950s in the field of **Adaptive Control Theory.** The thought is that the utility of a THEORY condition is the predictable entire reward from that condition forward, and every test gives a *model* of this value for every condition is looked up.

Adaptive Dynamic Programming (AIDP)

An **AIDP** agent performs by studying the Transition method of the situation as it moves next to and resolving the matching Markov Decision procedure by applying a dynamic programming system.

The complete agent program for a passive-ADP agent is represented in the Figure:

```
function PASSIVE-ADP-AGENT(percept) returns an action
  inputs: percept, a percept indicating the current state $s'$ and reward signal $r'$
  static: $\pi$, a fixed policy
          mdp, an MDP with model T, rewards R, discount $\gamma$
          $U$, a table of utilities, initially empty
          $N_{sa}$, a table of frequencies for state-action pairs, initially zero
          $N_{sas'}$, a table of frequencies for state-action-state triples, initially zero
          $s$, a, the previous state and action, initially null

  if $s$ is new then do $U[s] \leftarrow r$ ; $R[s] \leftarrow r'$
  if $s$ is not null then do
      increment $N_{sa}[s, a]$ and $N_{sas'}[s, a, s]$
      for each $t$ such that $N_{sas'}[s, a, t]$ is nonzero do
          $T[s,a,t] \leftarrow N_{sas'}[s,a,t] / N_{sa}[s,a]$
  $U \leftarrow$ POLICY- EVALUATION(^,  $U$, mdp)
  if TERMINAL?[$s'$] then $s, a \leftarrow$ null else $s$, a $\leftarrow s$ , $\pi[s']$
  return $a$
```

Figure: A Passive-reinforcement-learning agent-based on ADP

```
function PASSIVE-TD-AGENT(percept) returns an action
    inputs: percept, a percept indicating the current state $s'$ and reward signal $r'$
    static: n-, a fixed policy
            $U$, a table of utilities, initially empty
            $N_s$, a table of frequencies for states, initially zero
            $s$, a, $r$, the previous state, action, and reward, initially null

    if $s'$ is new then $U[s'] \leftarrow r'$
    if $s$ is not null then do
        increment $N_s[s]$
        $U[s] \leftarrow U[s] + \alpha(N_s[s])(r + \gamma U[s'] - U[s])$
    if TERMINAL?[$s'$] then $s, a, r \leftarrow$ null else $s$, a, $r \leftarrow s, \pi[s'], r'$
    return $a$
```

Figure: A passive-reinforcement-learning agent that learns utility guesses by applying temporal difference.

Temporal Difference Learning (TDL)

It is probable to contain the best of both worlds; which is, one should estimated the restriction expressions displayed previous with not resolving them for each probable conditions. *The major thing is to apply the experimental moves to change the experimental conditions values so that they accept* with the restriction expressions.

4.18. Active Reinforcement Learning

A passive-learning agent had permanent strategies that determine its performance. An active-agent should choose what actions to obtain.

Exploration

EXPLORATION maximizes its reward as mirrored in its present utility approximates and **exploration** to increase its lifelong success. Pure utilization threats receiving fixed in a track. Pure exploration to upgrade one's data is not usable if one not at all sets that information into procedure.

The following expression performs this:

$$U^{+}(s) \leftarrow R(s) + \gamma \max_{a} f\left(\sum_{s'} T(s,a,s')U^{+}(s'),\ N(a,s)\right)$$

at this point, f(u,n) is known as the **exploration function.**

Learning an Action-Value Function

Q-learning at learns an action-value presentation in place of learning utilities. A key property: *A TD agent, which learns a Q-function, performs without requirement a method **for** either action selection or learning.*

```
function Q-LEARNING-AGENT(percept) returns an action
    inputs: percept, a percept indicating the current state $s'$ and reward signal $r'$
    static: $Q$, a table of action values index by state and action
            $N_{sa}$, a table of frequencies for state-action pairs
            $s, a, r$, the previous state, action, and reward, initially null

    if $s$ is not null then do
        increment $N_{sa}[s, a]$
        $Q[a,s] \leftarrow Q[a,s] + \alpha(N_{sa}[s,a])(r + \gamma \max_{a'} Q[a',s'] - Q[a,s])$
    if TERMINAL?[$s'$] then $s, a, r \leftarrow$ null
    else $s, a, r \leftarrow s', \operatorname{argmax}_{a'} f(Q[a',s'], N_{sa}[s',a']), r'$
    return $a$
```

```
function Q-LEARNING-AGENT(percept) returns an action
    inputs: percept, a percept indicating the current state $s'$ and reward signal $r'$
    static: $Q$, a table of action values index by state and action
            $N_{sa}$, a table of frequencies for state-action pairs
            $s, a, r$, the previous state, action, and reward, initially null

    if $s$ is not null then do
        increment $N_{sa}[s, a]$
        $Q[a,s] \leftarrow Q[a,s] + \alpha(N_{sa}[s,a])(r + \gamma \max_{a'} Q[a',s'] - Q[a,s])$
    if TERMINAL?[$s'$] then $s, a, r \leftarrow$ null
    else $s, a, r \leftarrow s', \operatorname{argmax}_{a'} f(Q[a',s'], N_{sa}[s',a']), r'$
    return $a$
```

Figure: An Exploratory Q-learning Agent

4.19. Generalization in Reinforcement Learning

Function estimation creates it realistic to present utility tasks for huge situation spaces, but it is not the primary advantage. *The compression attained by a task estimated permits the learning-agent to situates it is not visited and to simplification it is visited.*

Question Bank

Unit - IV

Part - A

1. Differentiate supervised learning with unsupervised learning?
2. What is meant by reinforcement learning?
3. Define decision tree with an example.
4. Define regression.
5. Define over fitting.
6. What is meant by cross validation?
7. List the general uses of decision tree learning system?
8. what is meant by boosting?
9. Define the terms false negative and false positive for the hypothesis.
10. What is meant by memorization?
11. What is meant by functional dependencies or determinations?
12. Define Baye‘s rule.
13. What is meant by Artificial Neural Network(ANN)?
14. What is need of activation functions in neural network?
15. Differentiate real neuron with simulated neuron.
16. Differentiate feed forward with recurrent network.
17. What is meant by learning rate?
18. Differentiate passive learning with active learning.
19. Write short notes on Q-functions.
20. Write short notes on perception.
21. Differentiate nearest-neighbor method with Kernal method.
22. What is meant by cumulative learning?
23. List some of the applications of learning.
24. Define learning.
25. Explain the different kinds of learning methods.

Part - B

1. How the decision-tree-learning is done? Explain with an example and algorithm.
2. Explain the ensemble learning with boosting algorithm.
3. Explain the version space/candidate elimination algorithm with an example.
4. How to extract general rules from an example? Explain.
5. How learning with complete data is achieved?
6. Explain in detail EM algorithm.
7. Explain the back propagation-algorithm of learning in a multi-layer neural-network.
8. Explain passive reinforcement learning agent with

   (i) Adaptive Dynamic Programming (ADP) (ii) Temporal Difference (TD)

UNIT V

5. APPLICATIONS

Communication – Communication as Action – Formal Grammar for a Fragment of English – Syntactic Analysis – Augmented Grammars – Semantic Interpretation – Ambiguity and Disambiguation – Discourse Understanding – Grammar Induction – Probabilistic Language Processing – Probabilistic Language Models – Information Retrieval – Information Extraction – Machine Translation

5.1. Introduction of Communication

Communication is the planned replace of data brought regarding by the making and awareness of **symbols** warn from a collective method of traditional symbols.

What groups' persons separately from a distinct animal is the difficult method of prearranged communication called **language** that allows us to spread most of what we aware regarding the universe.

5.2. Communication as Action

The actions existing to an agent is to make language. This is known as **speech-act**. The general expressions mentioning to every form of communication is

Speaker → Utterance → Hearer

The different types of terms used in speech act are:

- Inform
- Query
- Request
- Acknowledge
- Promise

Basics of Language

A **formal language** is described as a group of strings. Apiece of string is a order of terminating signs is known as **words**.

A **grammar** is a fixed group of regulations that gives a language. The grammar is a group of modify the rules.

Example

Sentence - S

Noun phrase - NP

Verb phrase - VP

These are called **Non Terminal Symbols.**

The Constituent Steps of Conversation

A distinctive conversation event, in which spokesman 'S' needs to notify listener 'L' regarding proposition 'P' by applying words 'W', is collected of **7** steps.

1. Intention
2. Generation
3. Synthesis
4. Perception
5. Analysis
6. Disambiguation
7. Incorporation

The bellow figure explains the 7 methods mixed up in communication by applying the example statement: The wumpus is expired.

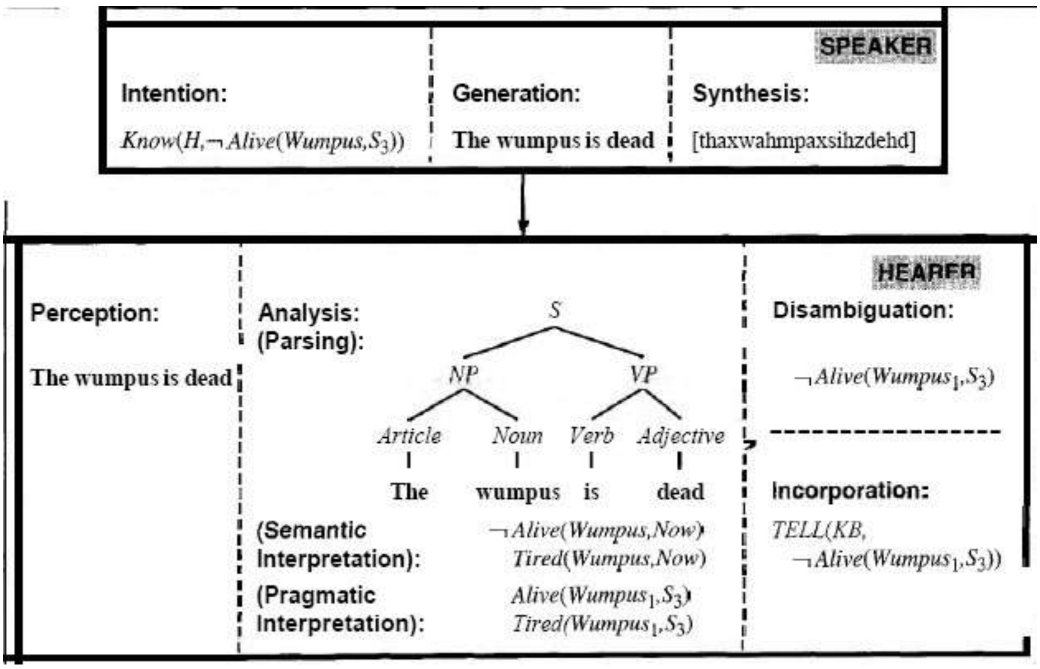


Figure: Seven Processes Involved in Communication

5.3. Proper Grammar for a Portion of English

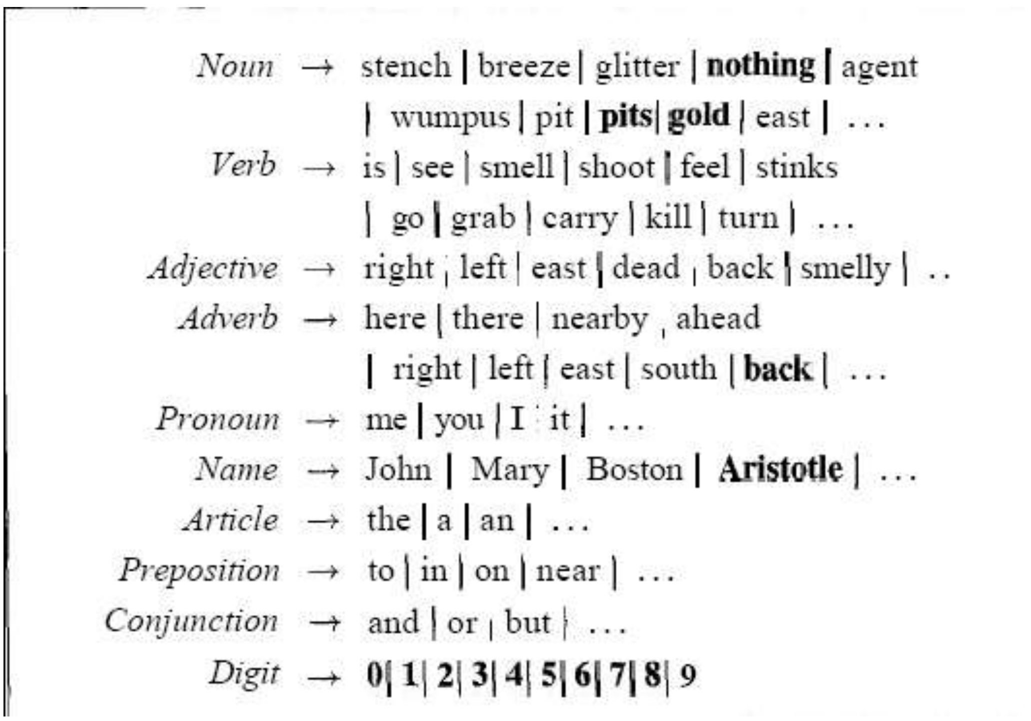
A proper grammar for a portion of English to create syntaxes regarding the wumpus world is build and this language. $\mathcal{E}_0$

The Lexicon of: $\mathcal{E}_0$

A index of permissible words, collected into groups such as

- Nouns
- Pronouns
- Verbs to indicate events
- Names to things
- Adverbs to change verbs
- Adjectives to change nouns
- Preposition
- Conjunction
- Articles

The following figure shows a small lexicon.



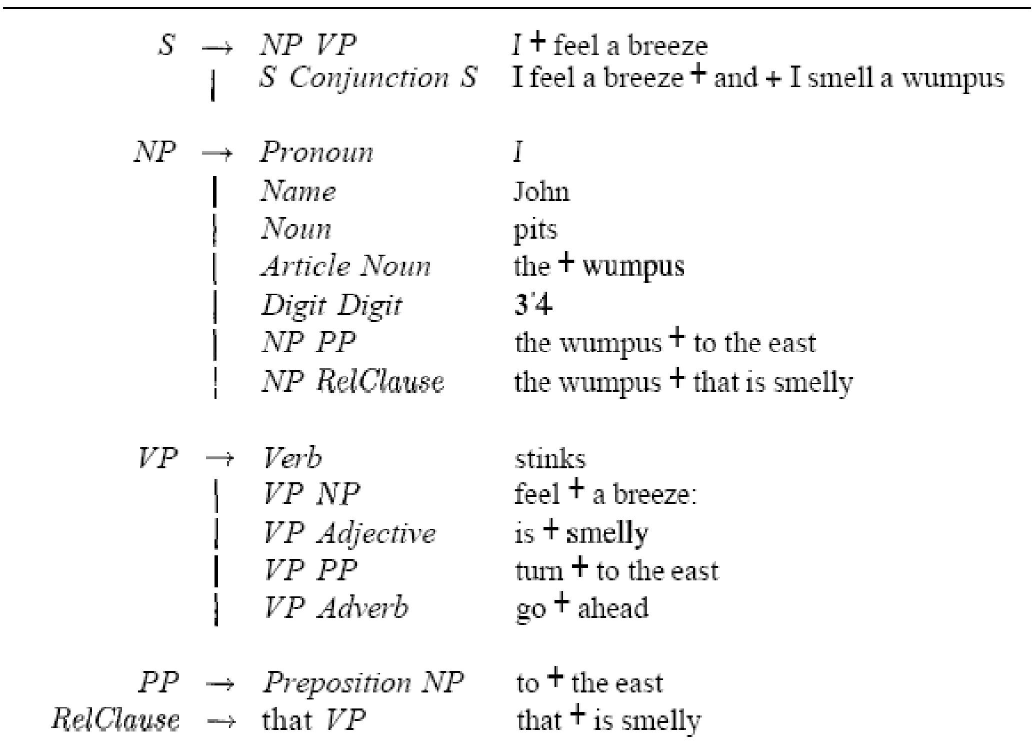
Noun	→	stench | breeze | glitter | **nothing** | agent | wumpus | pit | **pits** | **gold** | east | ...
Verb	→	is | see | smell | shoot | feel | stinks | go | grab | carry | kill | turn | ...
Adjective	→	right | left | east | dead | back | smelly | ..
Adverb	→	here | there | nearby | ahead | right | left | east | south | **back** | ...
Pronoun	→	me | you | I | it | ...
Name	→	John | Mary | Boston | **Aristotle** | ...
Article	→	the | a | an | ...
Preposition	→	to | in | on | near | ...
Conjunction	→	and | or | but | ...
Digit	→	**0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | 9

The Grammar of $\mathcal{E}_0$.

It defines how to combine the word and phrases, using 5 non-terminal symbols. The dissimilar types of expressions are:

- Sentence
- Verb phrase(VB)
- Noun phrase (NP)
- Relative clause(Reclause)
- Prepositional phrase(PP)

The figure explains a grammar for $\mathcal{E}_0$.



S	→	*NP VP*	I + feel a breeze
	|	*S Conjunction S*	I feel a breeze + and + I smell a wumpus
NP	→	*Pronoun*	I
	|	*Name*	John
	|	*Noun*	pits
	|	*Article Noun*	the + wumpus
	|	*Digit Digit*	3'4
	|	*NP PP*	the wumpus + to the east
	|	*NP RelClause*	the wumpus + that is smelly
VP	→	*Verb*	stinks
	|	*VP NP*	feel + a breeze:
	|	*VP Adjective*	is + smelly
	|	*VP PP*	turn + to the east
	|	*VP Adverb*	go + ahead
PP	→	*Preposition NP*	to + the east
RelClause	→	that *VP*	that + is smelly

Figure: The Grammar for $\mathcal{E}_0$.

5.4. Syntactic Analysis (Parsing)

Syntactic analysis is the phase in which an input statement is exchanged into a hierarchical-structure that communicates to the elements of meaning in the statement. This procedure is known as **parsing**.

Parsing should be observed as a procedure of discovering for a parse-tree. There are 2 methods for identifying the search-space.

1. Top-down parsing
2. Bottom-up parsing

1. Top-down Parsing

Initiate with the begin symbol and execute the grammar rules forward up to the symbols at the ends of the tree communicate to the elements of the statement initiating parsed.

2. Bottom-up Parsing

Initiate with the statement to be parsed and execute the grammar rules backward up to an individual tree whose ends are the words of the statement and whose top node is the institute symbol has been formed.

5.5. Augmented Grammars

The procedure of expanding the presented rules of the grammar in place of initiating latest rules. This formality for expansion is known as **definite clause grammar** or **DCG**. The major advantage of DCG is which we should expand the group signs with extra expand.

We define DCG as follows:

- The notation $\mathrm{X} \rightarrow \mathrm{Y}\ Z \ldots$ translates as $\mathrm{Y}(s_1)\ \mathrm{A}\ Z(s_2)\ A \ldots \Rightarrow X(s_1 + s_2 + \ldots)$.
- The notation $\mathrm{X} \rightarrow \mathrm{Y} \mid Z \mid \ldots$ translates as $\mathrm{Y}(s) \vee Z(s) \vee \ldots \Rightarrow X(s)$.
- In either of the preceding rules, any nonterminal symbol Y can be augmented with one or more arguments. Each argument can be a variable, a constant, or a function of arguments. In the translation, these arguments precede the string argument (e.g., $NP(case)$ translates as $NP(case, s_1)$).
- The notation $\{P(\ldots)\}$ can appear on the right-hand side of a rule and translates verbatim into $\mathbf{P}(\ldots)$. This allows the grammar writer to insert a test for $\mathbf{P}(\ldots)$ without having the automatic string argument added.
- The notation $\mathrm{X} \rightarrow$ **word** translates as $X([word])$.

5.6. Semantic Interpretation

Semantics – the origin of the meaning of statements. Semantic interpretation is the procedure linked an FOL statement with an expression.

$$
\begin{array}{l}
Exp(x) \rightarrow Exp(x_1)\ Operator(op)\ Exp(x_2)\ \{x = Apply(op, x_1, x_2)\} \\
Exp(x) \rightarrow (\ Exp(x)\ ) \\
Exp(x) \rightarrow Number(x) \\
Number(x) \rightarrow Digit(x) \\
Number(x) \rightarrow Number(x_1)\ Digit(x_2)\ \{x = 10 \times x_1 + x_2\} \\
Digit(x) \rightarrow x\ \{0 \leq x \leq 9\} \\
Operator(x) \rightarrow x\ \{x \in \{+, -, \div, \times\}\}
\end{array}
$$

Figure: A Grammar for Arithmetic Operations, Augmented with Semantics

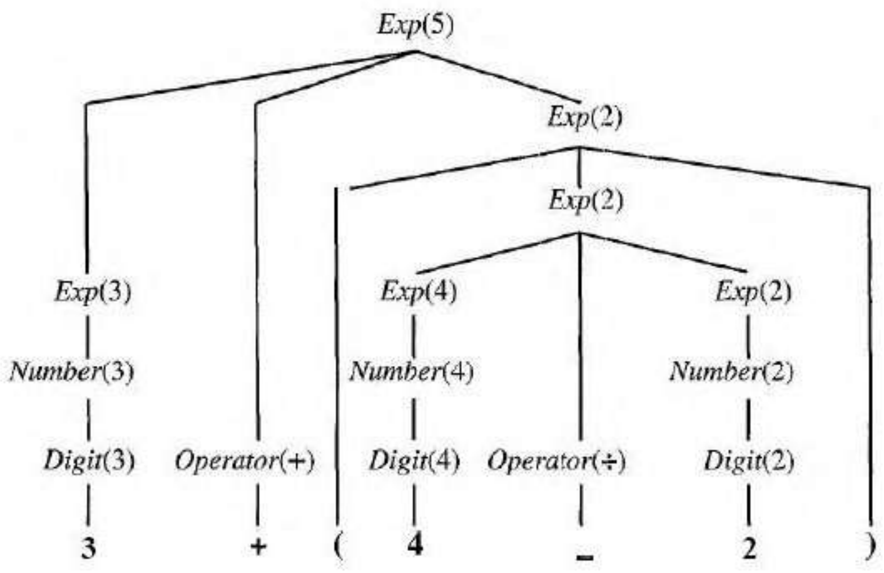


Figure: Parse-tree with Semantic-interpretations for the Expression 3 + (4 / 2)

The Semantics of an English Section

$S(rel(obj)) \rightarrow NP(obj)\ VP(rel)$
$VP(rel(obj)) \rightarrow Verb(rel)\ NP(obj)$
$NP(obj) \rightarrow Name(obj)$

$Name(John) \rightarrow$ **John**
$Name(Mary) \rightarrow$ **Mary**
$Verb(\lambda y\ Ax\ Loves(x,y)) \rightarrow$ **loves**

Figure: A grammar that should obtain a parse-tree and semantic-interpretation for "John loves Mary".

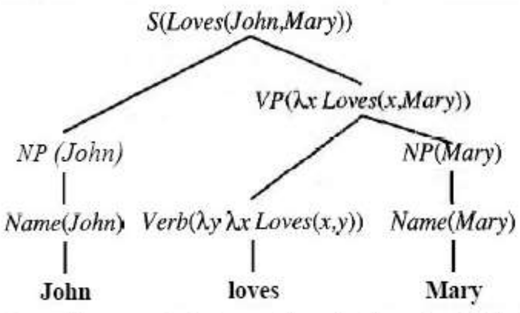


Figure: A parse-tree with semantic interpretations for the string: "John loves Mary"".

5.7. Ambiguity and Disambiguation

A statement, phrase or word is ambiguous, if it contains greater than 1 sense. Categories of ambiguity:

1. **Lexical ambiguity**: in which a statement contains greater than 1 sense. It is absolutely general;
2. **Syntactic ambiguity**: should happen with or without lexical-ambiguity.
3. **Semantic ambiguity**: The Syntactic-ambiguity directs to a semantic-ambiguity, for the reason that 1 translate signifies that the wumpus is in 2.2 and diversely that a stink is in 2.2. In this instance, receiving the incorrect interpretation should be a dangerous fault. It should happen still in phrases without syntactic or lexical ambiguity.

A **metonymy** is a form of talking in which one entity is utilized to set from one more.

A **metaphor** is a form of talking in which an expression with 1 truthful definition is utilized to propose s dissimilar meaning ay method of likeness.

Disambiguation

Disambiguation is a question of diagnosis. It is probable by joining proof, utilizing all the methods from uncertain reasoning and knowledge representation.

We should divide the knowledge down into **4** models:

1. Acoustic model
2. Mental model
3. World model
4. Language model

5.8. Discourse Understanding

A discourse is some statement of words – frequently one that is greater than a single statement extended. Text books, conversations, climate reports and novels are each discourse. We will look at two particular sub problems:

1. Reference Resolution

It is the explanation of a definite noun or a pronoun expression, which mention to an object in the universe. The resolution is the earlier parts of the discourse and based on knowledge of the universe.

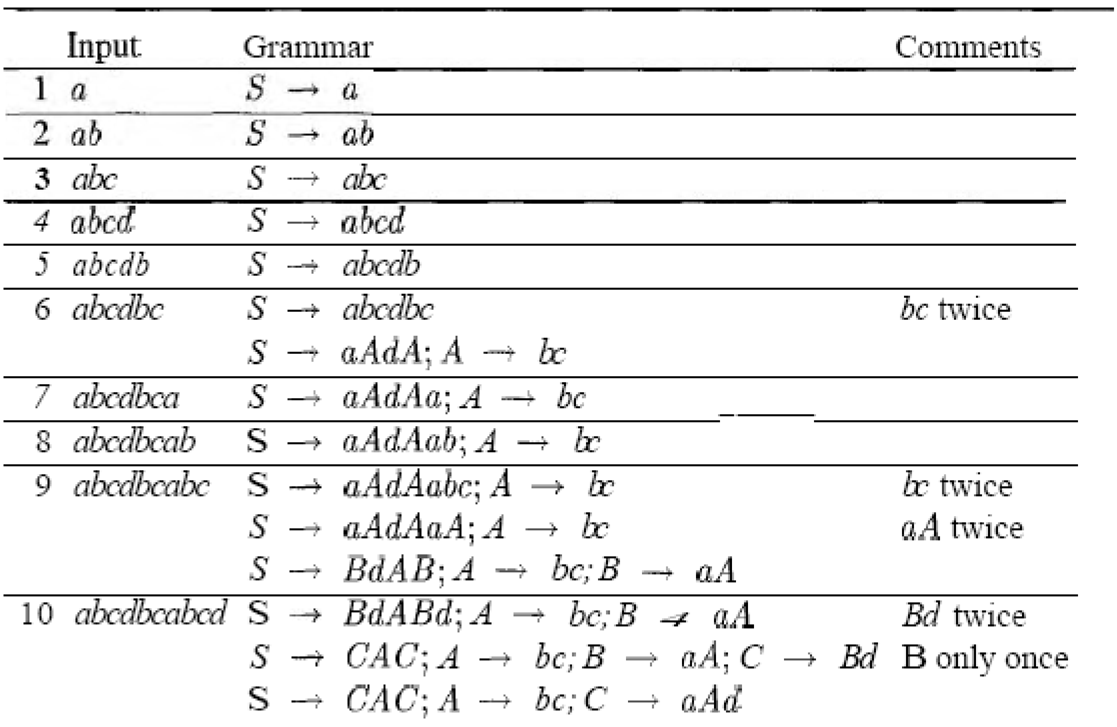
2. Structure of Coherent Discourse

A discourse contains structure over the level of statement. The grammar states that a discourse is collected of sections which every section is a collection of statements or either a statement and where sections are collected by coherence relations.

Grammar Induction

It is the work of studying a grammar from records. It makes a grammar of a particular from a grammar, which creates just a single statement namely the original text.

Figure displays the algorithm in process of the text:abcdbcabcd



	Input	Grammar	Comments
1	*a*	$S \rightarrow a$	
2	*ab*	$S \rightarrow ab$	
3	*abc*	$S \rightarrow abc$	
4	*abcd*	$S \rightarrow abcd$	
5	*abcdb*	$S \rightarrow abcdb$	
6	*abcdbc*	$S \rightarrow abcdbc$	*bc* twice
		$S \rightarrow aAdA; A \rightarrow bc$	
7	*abcdbca*	$S \rightarrow aAdAa; A \rightarrow bc$	
8	*abcdbcab*	$S \rightarrow aAdAab; A \rightarrow bc$	
9	*abcdbcabc*	$S \rightarrow aAdAabc; A \rightarrow bc$	*bc* twice
		$S \rightarrow aAdAaA; A \rightarrow bc$	*aA* twice
		$S \rightarrow BdAB; A \rightarrow bc; B \rightarrow aA$	
10	*abcdbcabcd*	$S \rightarrow BdABd; A \rightarrow bc; B \rightarrow aA$	*Bd* twice
		$S \rightarrow CAC; A \rightarrow bc; B \rightarrow aA; C \rightarrow Bd$	B only once
		$S \rightarrow CAC; A \rightarrow bc; C \rightarrow aAd$	

Figure: The Algorithm in Operation in the Text "abcdbcabcd"

Probabilistic Language Processing

Introduction

Probabilistic language models deployed on 'n' grams improve an unexpected quantity of data regarding a language.

In probabilistic-language model, this should learn from data. It is simpler than **Define Clause Grammar** (**DCG**). It has three specific asks.

1. Information extraction
2. Information retrieval
3. Machine translation

5.9. Probabilistic Language Models

A Probabilistic-language model describes a probability division above a group of strings.

- It can be conveniently trained from records.
- Learning is only counting number of incidences.
- It is robust
- It is applied for disambiguation probability should be utilized to select the most probable understanding.

E.g. Trigram and Bigram language model applied in speech identification.

A **unigram method** allocates a probability "P(w)" to every word in the word list. This method imagines that words are selected in separately, so the probability of a string is only the creation of the probability of words.

A **bigram method** allocates a probability $P(w_i|w_{i-1})$ to every word, specified the earlier word.

In common an n-gram method states on the earlier "n-1" words, allocates a probability for $P(w_i|w_{i-(n-1)}\ldots w_{i-1})$. We need someway of rectifying above the nil counts. The easiest method to perform this is known as AddOne smoothing; we insert a single to the count of each probable bigram.

One more method is **linear-interpolation-smoothing**, that joins bigram, unigram, and trigram methods by limited interruption.

Evaluating a Language Model

Split the information into training information and test information. Verify the restrictions of the model from the training information. Next compute the probability allotted to test information by the model. If the probability is bigger, then it is improved.

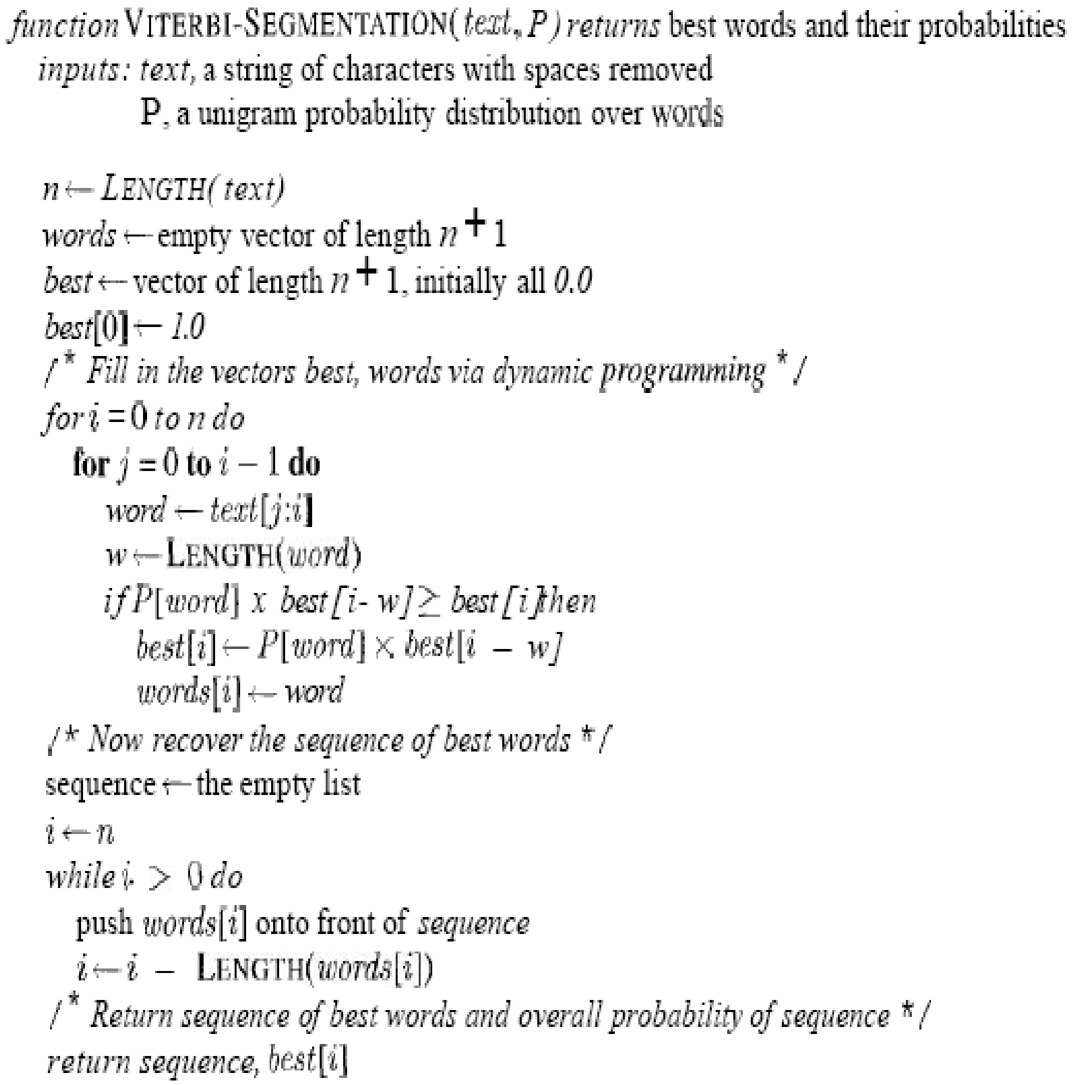
The second approach from evaluating a model is by computing the confusion of the model on test string of words.

Segmentation: It is used to find word margins in a text without spaces.

It is used to read words without spaces.

It is easy to read by human because they have full awareness of pragmatics, semantics and English sentences.

It is done by Viter bi algorithm specifically designed for segmentation problem.



```
function VITERBI-SEGMENTATION(text, P) returns best words and their probabilities
  inputs: text, a string of characters with spaces removed
          P, a unigram probability distribution over words

  n ← LENGTH(text)
  words ← empty vector of length n + 1
  best ← vector of length n + 1, initially all 0.0
  best[0] ← 1.0
  /* Fill in the vectors best, words via dynamic programming */
  for i = 0 to n do
     for j = 0 to i − 1 do
        word ← text[j:i]
        w ← LENGTH(word)
        if P[word] x best[i- w] ≥ best[i] then
           best[i] ← P[word] × best[i − w]
           words[i] ← word
  /* Now recover the sequence of best words */
  sequence ← the empty list
  i ← n
  while i > 0 do
     push words[i] onto front of sequence
     i ← i − LENGTH(words[i])
  /* Return sequence of best words and overall probability of sequence */
  return sequence, best[i]
```

Figure: A Viterbi-based Word Segmentation Algorithms

Probabilistic Context-free Grammars (PCFG)

Rams method get benefit of co-occurrences character in the corpra, but they does not contain notion of grammar at lengths larger than 'n'. PCFG contains of a CFG where every rule contain a connected probability.

The addition of probability of all rules is 1.

Learning Probability from PCFG

To create a PCFG, we have to combine probability from each CFG rule.

This proposes that learning the grammar from records could be improved than data-engineering method.

Two types of data are given:

1. Parsed
2. Unparsed

The 'E' step approximates the probabilities, which every subsequence is created by all rules. The 'M' step next approximates the probability of all rules. It is done by Inside-Outside algorithm.

Inside-Outside Algorithm

It induces grammar from unparsed tree. It is slow.

Learning Rule Structure from PCFG

If the grammar rules structure is not recognized, then make use of **Chomsky Normal Form (CNF)**.

$X \rightarrow Yz$

$X \rightarrow t$

Where t is a terminal and X, Y, Z is non terminals.

5.10. Information Retrieval (IR)

It is a work of selecting documents that are related to user's requirement for data.

E.g. Google, Yahoo etc.

An IR is characterized as

A. A manuscript gathering

B. Query in a query language

C. A result set

D. A demonstration of the result set.

The initial IR systems performed on a Boolean word type. Every word in this manuscript gathering is considered as a Boolean attribute, which is true of a manuscript if the word take places in the manuscript and false if it is not.

Evaluating IR Model

Here two measures to assess whether the IR method is performing well or not. They are:

1. Recall
2. Precision

Recall: is the percentage of all the related manuscripts in the gathering that are in the outcome.

Precision: is the percentage of manuscripts in the outcome set that are truly related.

Other Measures to Evaluate IR

The other measures are

1. Reciprocal rank.
2. Time to answer

IR Refinements

The uni-gram method considers all words as totally autonomous, but we recognize that a few words are associated.

E.g. the word -Couch|| has two closely related words

Couches

Sofa

So IR systems do refinements from these types of word by the following methods.

1. Case folding
2. Streaming
3. Recognize synonyms
4. Metadata

Presentation of Result Sets

There are three mechanisms to attain performance development. They are

1. Document classification
2. Relevance feedback
3. Document clustering
   - Agglomerative clustering
   - K-means clustering

Implementing IR Systems

IR systems are made efficient by two data structures. They are

1. Lexicon:
    - It indexes each and every word in the manuscript gathering.
    - It supports one operation
    - It is implemented by hash table.
2. Inverted index:
    - It is similar to the index at back side of a book. It consists of a set of hit lists, which is the place where the word occurs.
    - In uni-gram method, it is an index of pairs.

5.11. Information Extraction (IE)

IE is the procedure of generating database entries by removing content and viewing for connections between those events and objects and for incidents of a specific class of event or object.

E.g. To extract addresses from WebPages with database fields as road, village, pin code and state.

Categories of IE System

1. Attribute based System

This system imagines that the total text as an individual object and attempt to remove attributes of that object.

If standard expression equals the text closely once, next that section of text is extracted. It is the value of the attribute.

If there is no equal, not anything will happen.

2. Relational based System

It observes the connections between them and greater than one object.

This system is made by applying cascaded limited state generators, i.e. it contains of a sequence of Finite State Automata (FSA).

An example of this system is FASTUS.

It contains of 5 phases. They are:

- A. Tokenization
- B. Complex word handling
- C. Basic groups
- D. Complex phrases
- E. Merger structures

5.12. Machine Translation

It is the automated translation of text from one language (source) to another (target).

Types of Translation

1. Rough translation
2. Restricted source translation
3. Pre-edited translation
4. Literary translation

Machine Translation System

If translation is to be done fluently and perfectly then the interpreter (machine or human) should study the genuine text, be aware of the condition to which it is mentioned and discovers a better equivalent text in the target language reporting same or similar situation.

These systems differ in the stage to which they evaluate the text.

Statistical Machine Translation

It is a new approach proposed during last decade. Here the whole translation process is based on discovering the best possible translation of a statement, applying records collected from 's' bilingual process.

The method for p(F/E) has 4 set of limitations.

1. Translation method
2. Language method
3. Fertility method
4. Word choice method
5. Offset method

Learning Probabilities for Machine-translation

- Align sentences
- Segment into sentence
- Approximate the French language model
- Approximate the primary fertility model
- Approximate the primary choice model
- Approximate the primary offset model
- Improve estimates

Question Bank

Unit - V

Part - A

1. What is meant by communication?
2. Define utterance.
3. Define parsing.
4. List the types of grammar with its rules.
5. Give example for open classes and closed classes.
6. Differentiate bottom up parsing with top-down parsing.
7. What is meant by chart parsers?
8. Give example for chart parsing systems.
9. What is meant by define clause grammar.
10. Give example for augmented grammar.
11. List the types of ambiguity with example.
12. What is meant by probabilistic language model?
13. Define smoothing? What are the types of smoothing?
14. Differentiate information retrieval with information extraction.
15. List the stages of FASTOS.
16. What is meant by language model and translation model.
17. Write the steps for K means clustering.
18. Differentiate document classification with document clustering.
19. What is meant by stemming.
20. Define interlingua.

Part - B

1. Explain the seven processes involved in communication with one example.
2. Explain the two kinds of parsing with suitable example.
3. Explain the chart parsing algorithm with suitable example.
4. How semantic interpretation is done for a sentence? Explain with an example.
5. Write short notes on:
    - (a) Discourse understanding.
    - (b) Grammar induction
6. Explain the segmentation algorithm in detail.
7. Explain the steps to build an IR system.
8. List the different methods to do machine translation. Explain in detail about the statistical machine translation.
9. How IE is done using FASTOS system, explain the stages of it.