Module 1: Introduction to AI

What is artificial intelligence,
Well-Posed Learning Problems,
Designing a Learning System,
Choosing the training experience,
Choosing the target function,
Choosing a representation for the target function,
Choosing a function approximation algorithm,
Perspectives and issues in Machine Learning,
Problem spaces and search,
Heuristic search techniques.

Module 1: Introduction to AI

What is artificial intelligence?

- **Defining AI**: While defining artificial intelligence (AI) precisely is challenging, an approximate definition helps in discussions. AI is broadly described as the study of making computers perform tasks better than humans.

  Artificial Intelligence (AI) is the study of making computers perform tasks that humans currently do better. This definition is temporary and evolves with advancements in computer science. It avoids philosophical debates about "artificial" and "intelligence" while drawing parallels to philosophy, which historically studied knowledge before fields like mathematics and physics became independent. As AI progresses, it moves beyond tasks like playing chess to real-world applications, making its definition increasingly inadequate. AI often feels like a mirage—once achieved, it is no longer seen as AI but as standard technology, leading to the shift from "Artificial Intelligence" to "Already Implemented." Additionally, studying AI aids in understanding human intelligence, and though fully grasping AI may remain elusive, continuous exploration will gradually unveil new insights.

Understanding AI and its Challenges

- AI helps in understanding human intelligence; though fully comprehending AI may seem impossible, continuous study aids progress.
- Early AI research focused on formal tasks like game-playing and theorem proving.
- Programs such as Samuel’ s checkers-playing AI and the Logic Theorist attempted to improve performance using experience and logical reasoning.
- Early assumptions that AI could solve problems efficiently through fast computation proved incorrect due to combinatorial explosion.

THE UNDERLYING ASSUMPTIONS:

AI Problems

1. **Early AI Focus** – Initially targeted formal tasks like game playing and theorem proving (e.g., Samuel’ s checkers program).
2. **Commonsense Reasoning** – AI struggled with everyday reasoning despite efforts like the General Problem Solver (GPS).
3. **AI Expansion** – Research extended to perception, natural language processing, and expert-level problem-solving.
4. **Perception & Language Challenges** – AI faces difficulties in interpreting noisy visual/auditory data and understanding language.
5. **Expert AI Applications** – Used in engineering, medical diagnosis, finance, and scientific discovery.
6. **Categories of AI**

- **Mundane Tasks**: Perception, language, commonsense reasoning, robotics.
- **Formal Tasks**: Games, mathematics.
- **Expert Tasks**: Medicine, engineering, finance.

Physical Symbol System Hypothesis (PSSH)

1. **Definition** – A system using symbols to form structures and perform operations.
2. **Hypothesis (Newell & Simon, 1976)** – "A physical symbol system is necessary and sufficient for general intelligence."
3. **Validation** – Can’t be logically proven, only tested experimentally.
4. **Computers & Symbol Manipulation** – Computers can simulate physical symbol systems, recognized since Lady Lovelace (1842).
5. **Challenges & Counterarguments** – Neural networks and subsymbolic models question purely symbolic AI.
6. **Human-Like AI** – Some aspects, like humor, may not fit PSSH completely.
7. **Significance** – Important for understanding intelligence and building AI systems.

1. WHAT IS AN AI TECHNIQUE?

AI Problems & Techniques:

- AI problems are diverse but share a common trait: they are difficult.
- AI techniques manipulate symbols and may also be useful in non-traditional AI tasks.
- Understanding AI techniques requires analyzing their properties.

Intelligence Requires Knowledge:

- Knowledge is crucial for AI but comes with challenges:
    - It is vast and voluminous.
    - Difficult to define and characterize accurately.
    - Constantly changing.
    - Different from raw data due to its structured nature.

Characteristics of AI Techniques:

- **Generalization**: AI knowledge must capture patterns and avoid redundancy.
- **Human Interpretation**: Some AI knowledge needs to be understandable by people.
- **Modifiability**: AI systems must adapt to changes and correct errors.
- **Wide Applicability**: AI knowledge can be useful even when incomplete.
- **Efficiency**: AI should help manage large amounts of information by narrowing down possibilities.

Independence of AI Techniques:

- AI techniques should align with problem constraints but can also be applied outside traditional AI problems.

- Some AI problems can be solved without AI techniques, though the solutions may not be optimal.
- AI techniques can also be used to solve non-AI problems effectively.
- 1.3.1: TIC-TAC-TOE

Program 1: Key Features:

- **Board Representation:**
  - A 9-element vector representing the tic-tac-toe board.
  - Positions are numbered from 1 to 9.
  - Values:
    - 0 → Blank
- 1 → X
- 2 → O
- **Move Table:**
  - A large vector containing **19,683 elements (3^9)**.
  - Each entry in the table stores a possible board configuration.

Algorithm Steps:

1. Convert the board's state into a **ternary (base-3) number**, then into a **decimal number**.

2. Use this decimal value as an **index** to retrieve the next board configuration from the **Move Table**.

3. Update the board with the retrieved configuration.

Advantages:

- **Efficient in time complexity** (quick move selection).
- Can, in theory, play an optimal game.
- **Disadvantages:**
- **High space complexity** (storing all possible board states).
- **Manual effort required** to populate the move table correctly.
- **Prone to errors** in move table definition.
- **Not scalable** (e.g., for a 3D tic-tac-toe, memory requirements become infeasible).

Program 2:

Key Features:

- Uses a **9-element board vector** with values **2 (blank), 3 (X), 5 (O)**.
- A **turn counter** tracks the current move (1 to 9).

Algorithm Components:

1. **Make2:** Chooses the center if available; otherwise, picks a non-corner square.
2. **Posswin(p):** Checks if p (X or O) can win next turn.
3. **Go(n):** Places a move in square n and updates the board.

Move Strategy:

- **X moves on odd turns, O on even turns.**
- Prioritizes **winning moves, blocking opponent, or forming a fork**. **Pros:**
- **More space-efficient** than Program 1.
- **Easier to modify and understand**. **Cons:**
- **Slower** due to multiple condition checks.
- **Fixed strategy**—not adaptable or intelligent.
- **Not generalizable** beyond standard tic-tac-toe.

Program 3:

Key Features:

- Uses a **BoardPosition structure** that stores:
    - The **current board state**.
    - A **list of possible next board positions**.
    - An **evaluation score** estimating how likely a position leads to a win.

Algorithm (Minimax Procedure): Check for a winning move. If found, assign the highest rating.

1. **Consider all possible opponent moves.** Find the worst-case outcome using recursion.
2. **Assume the opponent plays optimally** to minimize the player's chances of winning.
3. **Choose the move with the best rating. Pros:**

- **More strategic** than previous programs by considering multiple moves ahead.
- **Can be extended** to handle more complex games beyond tic-tac-toe.
- **Does not require hardcoded rules**, making it more generalizable.

2. : question answering:

The section discusses question-answering programs that read English text and respond to questions based on that text. It highlights the challenge of defining "correct" answers in such programs.

Key Points:

1. **Example of Input Sentence:**
    - "Russia massed troops on the Czech border."
    - This sentence serves as the input for question-answering dialogues.
2. **Two Different Dialogues Based on Political Models:**
    - **Dialogue 1 (American Conservative Model - 1977):**
        - Q: Why did Russia do this?
        - A: To take political control of Czechoslovakia.
        - Q: What should the U.S. do?
        - A: Intervene militarily.

Dialogue 2 (American Liberal Model - 1977):

- Q: Why did Russia do this?
- A: To increase political influence over Czechoslovakia.
- Q: What should the U.S. do?
- A: Denounce the action in the United Nations.

3. Role of the POLITICS Program:

- Answers depend on both the input text and an external belief model.
- Changing the belief model alters the system’s responses.
- The first dialogue aligns with a conservative view; the second aligns with a liberal perspective.

4. Defining Correct Answers:

- Determining correctness is difficult.
- Many question-answering programs define correctness based on their computation process.
- No universally satisfactory method exists for defining correctness in question- answering programs.

3. Another Example for Comparison:

- Input text: A woman shopping for a coat finds a red one, likes it, and later realizes it matches her dress.
- Questions:

  Why did Mary go shopping?
  What did Mary find that she liked?
  Did Mary buy anything?
- The goal is to see how different programs handle such queries.

This section illustrates the complexities in automated question-answering and the influence of external knowledge models on the answers generated.

Program 1: Overview

- The program attempts to answer questions by directly matching text fragments from the input text.
- It uses predefined templates to detect patterns in both the question and the text.

Data Structures

- **QuestionPatterns**: A set of templates that map common question structures and generate patterns to match input text.
- Templates and patterns (termed *text patterns*) work together to extract relevant text from the input.

Algorithm Steps

1. Compare each question with QuestionPatterns and match templates to generate text patterns.
2. Apply a substitution process to expand verb variations (e.g., "go" → "went").
3. Apply the text patterns to the input text and extract matching results.
4. Respond with the collected answers.

Examples

1. **Q1 ("What did Mary go shopping for?")** → The system extracts "Mary went shopping for a red coat" and returns "a red coat."
2. **Q2 ("What did Mary find that she liked?")** → The system fails to provide an answer unless additional template variations exist.
3. **Q3 ("Did Mary buy anything?")** → No direct answer in the text, so no response is given.

Program 2:

1. Purpose of the Program

- The program **converts input text** into a structured internal form that captures meaning rather than just raw words.
- It **processes questions** into the same structured format.
- The program **answers questions** by comparing their structured representations with the structured form of the input text.

Data Structures Used

1. **EnglishKnowledge**
   - A **knowledge base** containing English grammar, vocabulary, and semantic rules.
   - Used for both **parsing** English input into a structured form and **generating** English output from structured data.

2. **InputText**
   - The raw input **text in character form** before processing.

3. **StructuredText**
   - A **structured representation** of the input text that preserves meaning without relying on word order or specific wording.
   - It clarifies **implicit information**, such as pronoun references.
   - Uses a **slot-and-filler structure**, where each element (e.g., subject, object, action) is placed in predefined slots.

4. **InputQuestion**
   - The **raw user query in text form**, before being processed.

5. **StructQuestion**
   - A **structured representation of the question**, using the same slot-filling method as

3. Algorithm for Answering Questions

1. Convert the input text into a structured form

- Uses **EnglishKnow** to identify sentence components and structure the meaning.
- Ambiguities in English (e.g., pronoun references) are resolved as much as possible.

2. Convert the question into structured form

- Uses **special markers** to indicate what part of the text is being asked about.
- These markers usually align with question words (e.g., "who," "what").

3. Compare the structured question with the structured input text

- The system searches for matches between the question and the stored structured representation.

4. Return the answer

- Extracts and presents relevant parts of the structured text that match the query.

4. Examples of Question-Answering Performance

- **Q1:** The question correctly answers with **"a new coat."**

- **Q2:** Also answered correctly, providing **"a red coat."**
- **Q3:** This cannot be answered because no matching structured information is found in the input text.

5. Strengths & Limitations Strengths:

- More **meaning-oriented** than simple pattern-matching approaches.
- Flexible in answering questions reworded differently but conveying the same meaning.

Limitations:

- **Depends on structured knowledge**: The system needs **English Know** and **Structured Text** to function effectively.
- **Handling pronouns is difficult**: Resolving pronoun references is problematic without real-world knowledge.
    - Example:
        - **"Mary walked up to the salesperson. She asked where the toy department was."**
        - The system cannot determine whether **"she"** refers to Mary or the salesperson without additional knowledge.

5. Conclusion & Next Steps

- Converting text into structured representations is **better than simple pattern recognition**, but it is **not enough** for fully understanding language.
- A **more sophisticated, knowledge-rich approach** is needed, incorporating **world knowledge and context awareness** to improve question answering.

Program 3:

The text describes a program that converts input text into a structured form containing the meaning of sentences and integrates it with prior knowledge about objects and situations. This allows the system to answer questions using an augmented knowledge structure.

Key Points:

1. **WorldModel:** Represents background knowledge about objects, actions, and situations.
2. **Shopping Script Example:** A structured representation of shopping behavior, illustrating paths taken based on conditions (e.g., customer searches, buys an item, or leaves).
3. **Algorithm:**

- Convert input text into a structured form using WorldModel knowledge.
- Match question structure with IntegratedText.
- Extract answers from matched structures.

4. **Inference Limitation:** The system lacks a general reasoning mechanism, meaning it cannot deduce unstated facts unless explicitly encoded.
5. **AI Techniques & Question Answering:** The approach relies on structured knowledge but highlights the importance of computational reasoning for natural question answering.

Well-posed learning programs

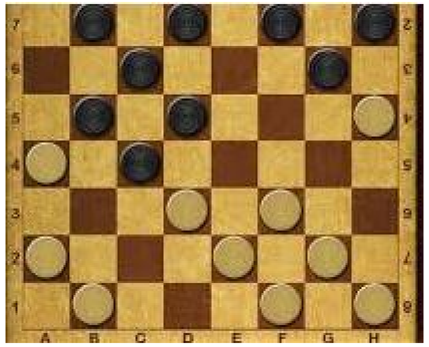
Definition of Machine Learning

- A computer program is considered to learn if its performance on a task (T), as measured by a performance metric (P), improves with experience (E).
- Example: A program that learns to play checkers improves by playing against itself.

Applications of Machine Learning

- **Speech Recognition**: Systems like SPHINX use machine learning to recognize spoken words by adapting to individual speakers, background noise, and other factors.
- **Autonomous Vehicles**: AI-driven vehicles, such as ALVINN, learn to navigate roads and have successfully driven unassisted on highways.

Examples of Machine Learning Tasks

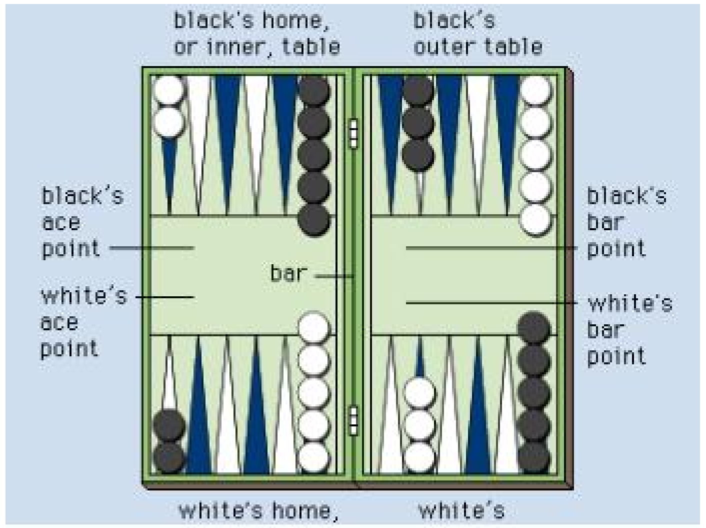


- **Checkers Learning Problem:**
  - **Task (T):** Playing checkers.
  - **Performance (P):** Percentage of games won.
  - **Experience (E):** Playing practice games against itself.
- **Handwriting Recognition:**
  - **Task (T):** Recognizing and classifying handwritten words.
  - **Performance (P):** Percentage of words correctly classified.
  - **Experience (E):** Learning from labeled handwriting samples.

- **Astronomical Classification**: Machine learning is used by NASA to classify celestial objects from large image databases.
- **Game Playing (Backgammon)**: Programs like TD-GAMMON learn by playing millions of games against themselves, reaching world-class performance levels.

Significance of Machine Learning

- Machine learning techniques are widely used in various fields, from speech processing to autonomous systems and data analysis.
- These methods enable computers to find patterns in vast amounts of data and make intelligent decisions.



1. Three Essential Features of a Learning Problem

To have a well-defined learning problem we must identify three features:

- **Task (T):** What the system is learning to do.
- **Performance Measure (P):** How well the system is improving.
- **Training Experience (E):** The data or interactions used for learning.

Contributions from Different Fields to Machine Learning

- **Artificial Intelligence (AI):**
    - Learning concepts through symbolic representations.
    - Using training data and prior knowledge for problem-solving.
- **Bayesian Methods:**
    - Using Bayes' theorem to calculate probabilities of hypotheses.
    - Naïve Bayes classifier for making predictions.

Computational Complexity Theory:

- Measuring the effort needed to learn (time, training examples, mistakes).

- **Control Theory:**
  - Learning to control processes and optimize objectives.
- **Information Theory:**
  - Concepts like entropy, optimal coding, and data compression.
- **Philosophy:**
  - Occam’ s razor (simpler hypotheses are better).
  - Justification for generalizing learning beyond observed data.
- **Psychology & Neurobiology:**
  - Understanding human learning to improve machine learning.
  - Neural networks inspired by brain functions.
- **Statistics:**
  - Error characterization (bias, variance).
  - Statistical methods for evaluating hypothesis accuracy.

Designing a Machine Learning Program

1. **Goal:** Create a program to compete in the World Checkers tournament.
2. **Performance Measure:** The percentage of games won in the tournament.
3. **Why Use Machine Learning?**
   - Instead of manually programming all possible moves, the program **learns** by playing games.
   - It improves its strategy over time based on experience.

Choosing the Training Experience:

1.Choosing the Training Experience

- **Direct vs. Indirect Feedback:**

  - **Direct:** Learned from specific board states and correct moves.
  - **Indirect:** Learns from full game outcomes, requiring **credit assignment** (determining which moves contributed to winning or losing).

Here, the learner faces the additional problem of credit assignment or determining the degree to which each move in the sequence deserves credit or blame for the outcome

Credit assignment can be a complex problem because the game can be lost even when early moves are optimal. Hence, learning from direct training feedback is typically easier than learning from indirect feedback.

2.Who Controls the Training: The second key attribute is the degree to which the learner controls the sequence of training examples.

- **Teacher-Guided:** An expert selects board states and correct moves.
- **Learner-Guided:** The system asks about complex moves.
- **Self-Learning:** The system plays against itself without external help.

3. Representativeness of Training Data

- **Issue:** If the program only plays against itself, it may not learn strategies used by human champions.
- **Ideal Case:** Training examples should match real-world game scenarios.

4. Final Design Choices

- **Training Method:** The system will train by playing against itself (unlimited data).
- **Defined Problem:**
  - **Task (T):** Playing checkers.
  - **Performance (P):** Winning percentage in the world tournament.
  - **Training Experience (E):** Self-played games.

5. Next Steps in Design

- **What knowledge to learn?** (E.g., best moves, strategies).
- **How to represent this knowledge?** (Rules, patterns, probabilities).
- **What learning mechanism to use?** (Neural networks, decision trees, reinforcement learning).

Choosing the target function:

The following design choice is to determine what knowledge will be learned and how the performance program will use this.

1. **Design Choice in Learning**: The key decision is determining what knowledge should be learned and how the performance program will use it.
2. **Checkers-playing Example**: The program can generate legal moves from a board state but needs to learn how to select the best move.
3. **Generalized Learning Task**: This problem represents a broader class where legal moves (or actions) are known. However, the best strategy for selecting them is not such as scheduling and manufacturing control. Many optimization problems fall into this class
4. **Target Function - ChooseMove**:

•Given this setting where we must learn to choose the legal moves, the most obvious choice for the type of information to be learned is a program or function that chooses the best move for any given board state. Let us call this function ***ChooseMove.***

•A natural choice is a function, ***ChooseMove*(B → M)**, that directly selects the best move M for any board state B.

•However, learning this function is challenging due to the indirect nature of training data.

5. **Alternative Target Function - Evaluation Function (V)**:

•Instead of learning ChooseMove directly, a function **V(B $\rightarrow$ $\mathbb{R}$)** is learned, which assigns a numerical score to board states.

•The system can then evaluate successor states and select the best move indirectly.

6. **Advantage of Evaluation Function (V)**:

•Learning V is easier compared to learning ChooseMove.

•Once V is learned, the best move can be determined by evaluating all successor states and choosing the one with the highest score.

7. Defining the Target Function V: The function V assigns numerical values to board states to facilitate optimal move selection.

8. Target Values for V(b):

•100 for a winning final board state.

•−100 for a losing final board state.

•0 for a drawn final board state.

•For non-final states, V(b)=V(b′), where b′ is the best achievable final board state assuming optimal play.

9. **Issue with the Definition**:

•The recursive definition requires looking ahead until the end of the game, making it **computationally infeasible**.

•This makes V a **nonoperational definition**, meaning it cannot be directly used for real- time decision-making.

10. **Learning as Function Approximation**:

•The goal is to find an **operational** (computable) approximation of V that the checkers- playing program can use efficiently.

•The learned function, denoted as ?, approximates V, as knowing the exact V is often impractical.

Choosing a Representation for the Target Function

Choice of Target Function Representation:

- The program must learn a target function **V**, but must also choose how to represent it (e.g., using a table, rules, a polynomial, or a neural network).
- **Tradeoff**: More expressive representations allow closer approximations of the target function but require more training data to learn effectively.

Chosen Representation:

- The program will represent the value of a board state (c) as a **linear combination** of predefined board features:

  - X1:Number of black pieces
  - X2: Number of red pieces
  - X3: Number of black kings
  - X4: Number of red kings
  - X5: Number of black pieces threatened by red
  - X6: Number of red pieces threatened by black

- The function will be in the form:

$$c(b) = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6$$

  - where w0 to w6 are weights to be determined by the learning algorithm.

Training Program:

- The learned weights will determine the relative importance of each board feature for the evaluation function.
- The weight w_0 is an additive constant.
- The task is reduced to learning values for the coefficients w_0 through w_6

Choosing a Function Approximation Algorithm

Training Examples:

- To learn the target function f, a set of **training examples** is required. Each example consists of a specific **board state** b and its corresponding **training value** V_{train}(b)
- A training example is an ordered pair: (b, V_{train}(b)), where b represents the board state and V_{train}(b)is the target value for that state.

Example:

- For instance, a board state b where black has won the game (with x_2 = 0, indicating no red pieces left) would have a target function value V_{train}(b) = +100.

Learning Procedure:

- The procedure involves:
  1. **Deriving training examples** from the indirect training experience available.
  2. **Adjusting the weights** w1 to w6 in the representation to fit the training examples best.

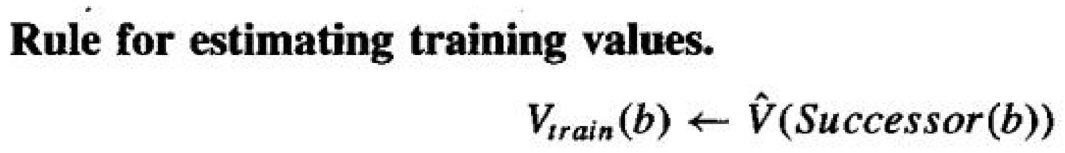
ESTIMATING TRAINING VALUES

Challenge of Training Data:

- The learner has limited training information: only whether the game was won or lost.
- However, specific training values are needed for each board state, including intermediate states, before the game ends.
- The outcome (win/loss) does not always reflect the quality of each intermediate board state, as good early-game states might contribute to a later loss.

Approach to Estimating Training Values:

- One approach is to assign the training value V_{train}(b) for any intermediate board state b based on the estimated value of the next board state, denoted as Successor(b), where the program has its turn to move.
- The training value is estimated using the current approximation V^ of the target function: Vtrain(b)≈V^(Successor(b)).



Rule for estimating training values.

$$V_{train}(b) \leftarrow \hat{V}(Successor(b))$$

- **Iterative Refinement**:
- This method may seem odd because the learner uses its current approximation of V to estimate values for intermediate states. However, this is effective because the approximation V^ tends to be more accurate for board states closer to the end of the game.
- Under certain conditions, this iterative approach can **converge** to perfect estimates of V_train.

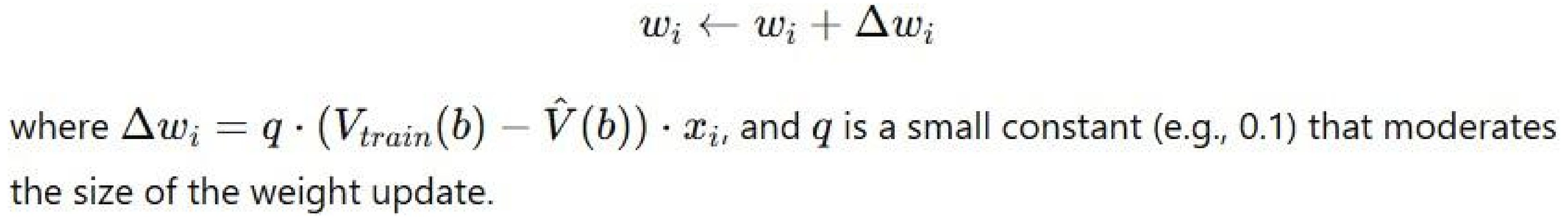
ADJUSTING THE WEIGHTS

Objective:

- The goal is to specify a learning algorithm to choose the weights w_i that best fit the training examples (b, V_{train}(b)).
- The "best fit" is often defined as minimizing the **squared error** E between the predicted values V^(b) and the actual training values V_{train}(b)

Minimizing Squared Error:

- The weights w_i are adjusted to minimize the error E for the observed training examples.
- Minimizing squared error can, in specific settings, be equivalent to finding the most probable hypothesis given the observed training data.



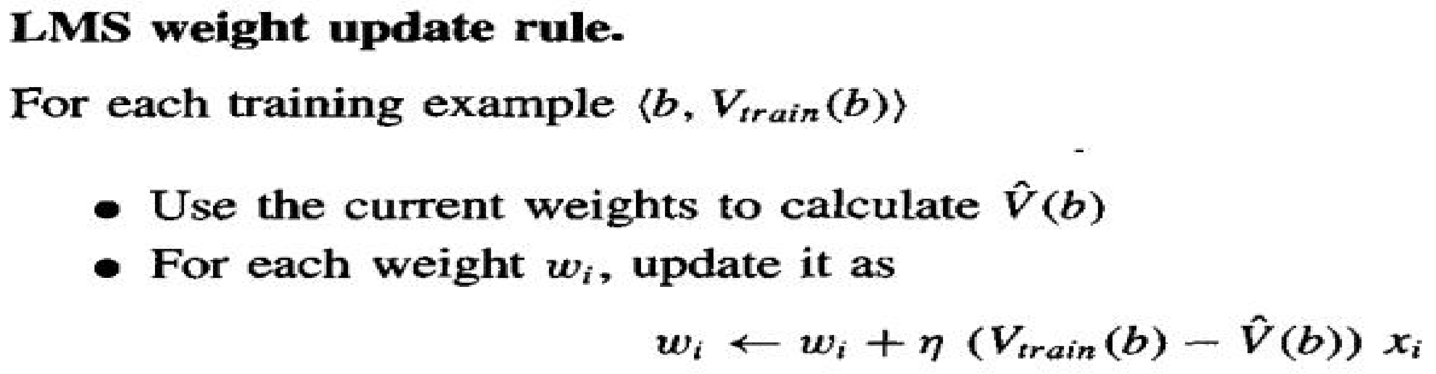
$$w_i \leftarrow w_i + \Delta w_i$$

where $\Delta w_i = q \cdot (V_{train}(b) - \hat{V}(b)) \cdot x_i$, and q is a small constant (e.g., 0.1) that moderates the size of the weight update.

Least Mean Squares (LMS) Algorithm:

- The **LMS algorithm** is used to find the weights that minimize the squared error incrementally as new training examples are introduced.
- The algorithm adjusts the weights based on each training example to reduce the error on that example.

LMS Weight Update Rule:



LMS weight update rule.

For each training example $\langle b, V_{train}(b)\rangle$

- Use the current weights to calculate $\hat{V}(b)$
- For each weight w_i, update it as

$$w_i \leftarrow w_i + \eta \; (V_{train}(b) - \hat{V}(b)) \; x_i$$

The Final Design

The checkers learning system consists of four key modules that work together to improve performance through learning:

1. **Performance System**
    - Plays the game of checkers using the learned evaluation function.
    - Takes a new game as input and generates a history of moves as output.
    - The strategy used is determined by the evaluation function, which improves over time with learning.

2.Critic

- Analyzes the game history and extracts training examples.
- Each training example includes a board state and an estimated target function value.
  In this system, the Critic follows a specific training rule to generate these examples.

3. Generalizer

- Uses training examples to generate an improved hypothesis (evaluation function).
- Generalizes from specific examples to create a function that can be applied to new situations.
- In this system, it uses the **Least Mean Squares (LMS) algorithm** to update weights and refine the function.

4. Experiment Generator

- Selects new problems (initial board states) to help the system learn more effectively.
- In this case, it always starts from the same initial board state, but more advanced versions could introduce different scenarios to enhance learning.

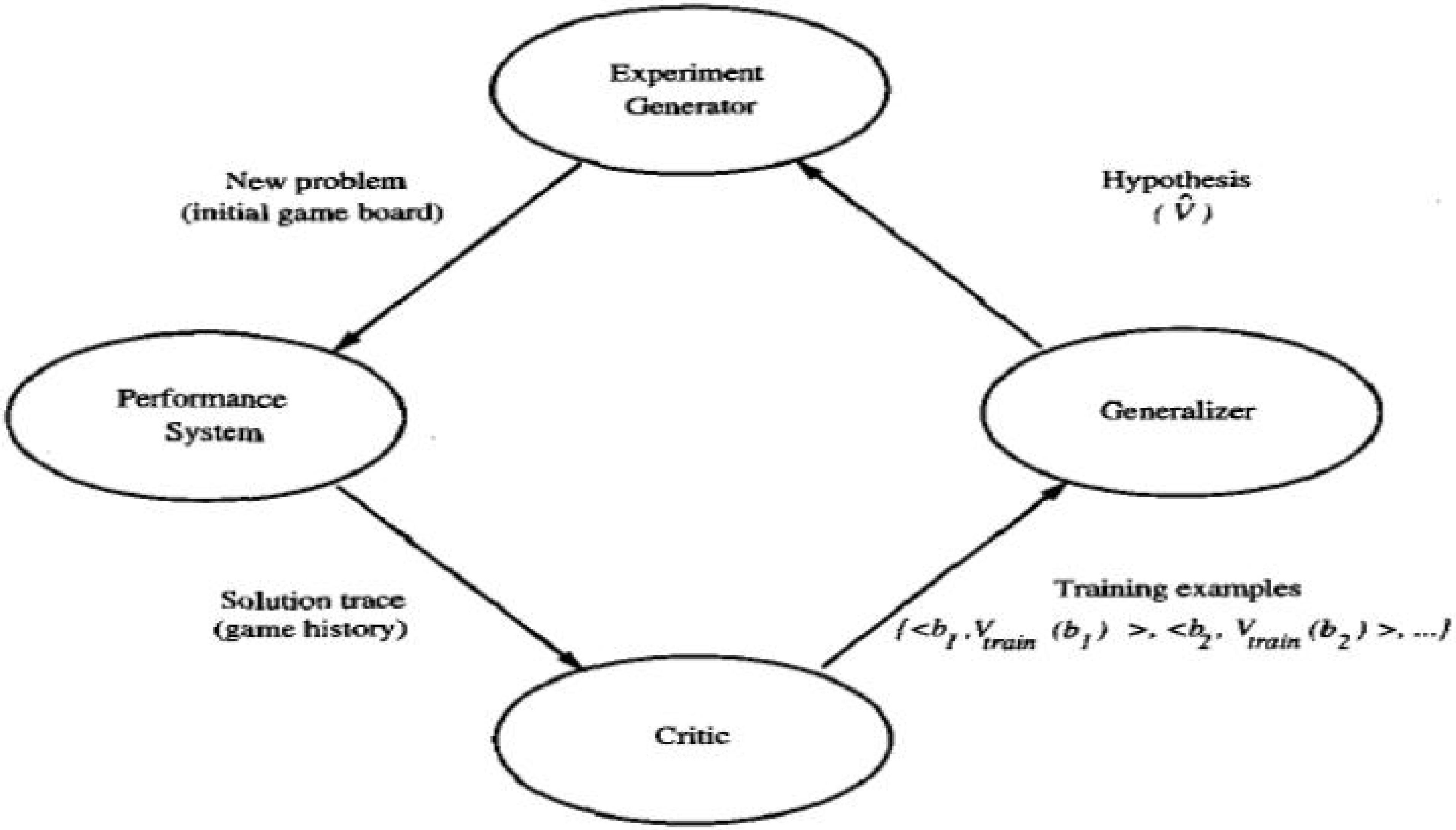

Experiment Generator
New problem (initial game board)
Hypothesis ($\hat{V}$)
Performance System
Generalizer
Solution trace (game history)
Training examples
$\{<b_1, V_{train}(b_1)>, <b_2, V_{train}(b_2)>, \ldots\}$
Critic

FIGURE 1.1
Final design of the checkers learning program.

Constraints and Limitations

- The system learns a **linear evaluation function** based on six board features.
- If the optimal function can be represented in this form, the program has a good chance of learning it. Otherwise, it can only approximate the best solution.
- Although this method improves performance, it is unlikely to reach world-champion- level play due to the **simplicity of the linear representation**.
- More advanced learning approaches, such as **neural networks**, could improve performance by considering a more detailed representation of board states.

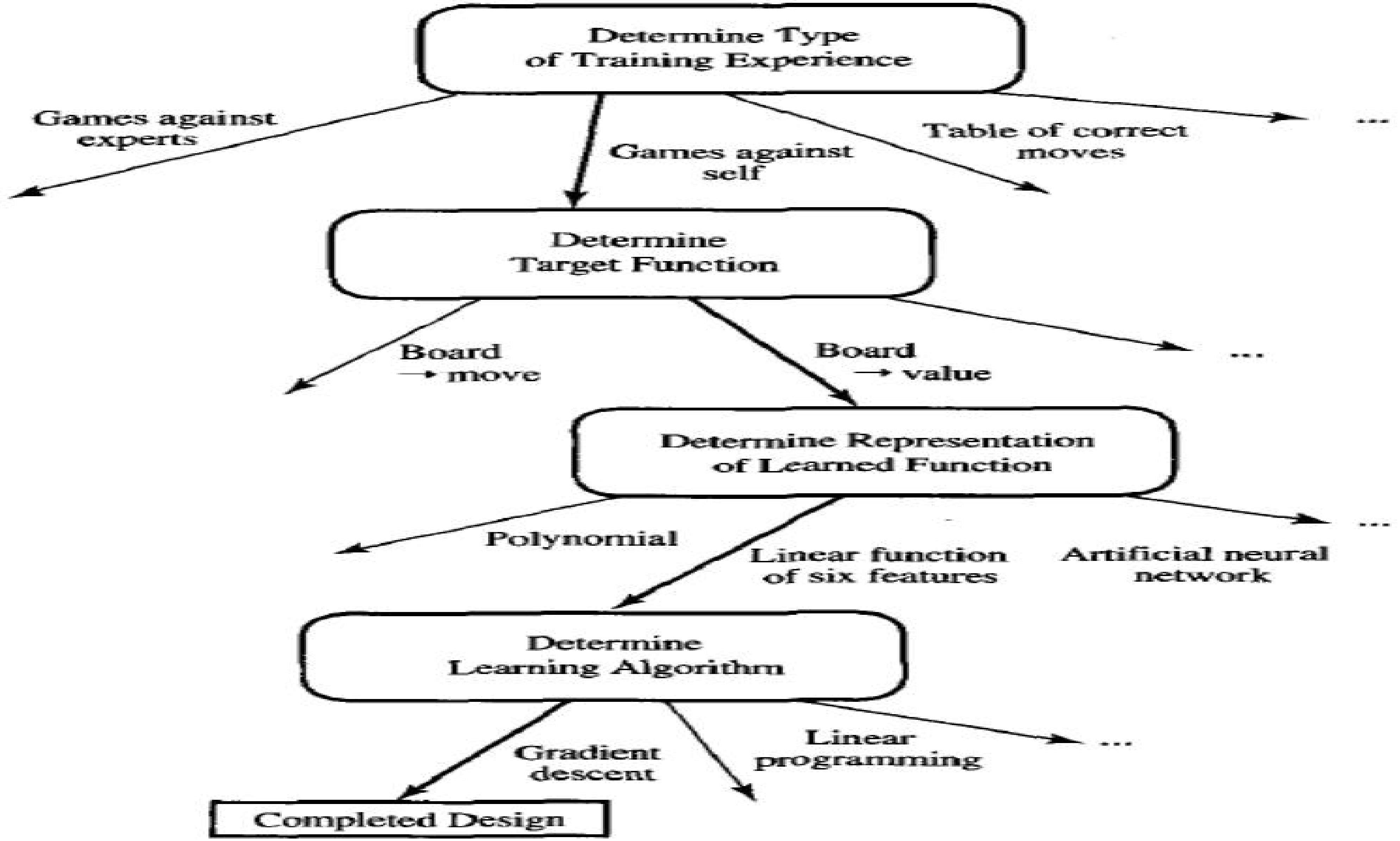

Determine Type of Training Experience
Games against experts
Games against self
Table of correct moves
...
Determine Target Function
Board → move
Board → value
...
Determine Representation of Learned Function
Polynomial
Linear function of six features
Artificial neural network
...
Determine Learning Algorithm
Gradient descent
Linear programming
...
Completed Design

FIGURE 1.2
Summary of choices in designing the checkers learning program.

Alternative Approaches

- **Nearest Neighbor Algorithm**: Store training examples and find the most similar past game state for decision-making.
- **Genetic Algorithms**: Generate multiple checkers programs, have them compete, and evolve the most successful strategies over time.
- **Explanation-Based Learning**: Analyze past successes and failures to refine strategies based on deeper reasoning.

This checker learning system is an example of how machine learning models can be structured, highlighting the importance of selecting the proper learning methods for a given task.

PERSPECTIVES AND ISSUES IN MACHINE LEARNING

Machine learning can be understood as a search process through a vast hypothesis space to find the best fit for observed data and prior knowledge. In the case of the checker learning system, this space consists of all possible evaluation functions represented by different weight values. The LMS algorithm refines the hypothesis by iteratively adjusting weights whenever there is a discrepancy between predicted and actual values. Different hypothesis representations, such as linear functions, decision trees, and neural networks, require distinct search strategies suited to their structures. Learning algorithms explore these structured spaces while balancing search efficiency, training data size, and generalization ability to ensure accurate predictions on unseen data. This search-based perspective is essential for analyzing learning methods and understanding their effectiveness across various machine-learning problems.

Issues in Machine Learning

The checkers example highlights key questions in machine learning, focusing on how algorithms learn general target functions from specific training examples and under what conditions they converge to desired results. It explores the relationship between training data size and hypothesis confidence, the role of prior knowledge in guiding learning, and strategies for selecting effective training experiences. Additionally, it examines how to frame learning as function approximation, whether this process can be automated, and how learners can refine their representations to improve learning efficiency and accuracy. These fundamental questions shape research in machine learning and influence the development of effective learning algorithms.

PROBLEMS, PROBLEM SPACES, AND SEARCH

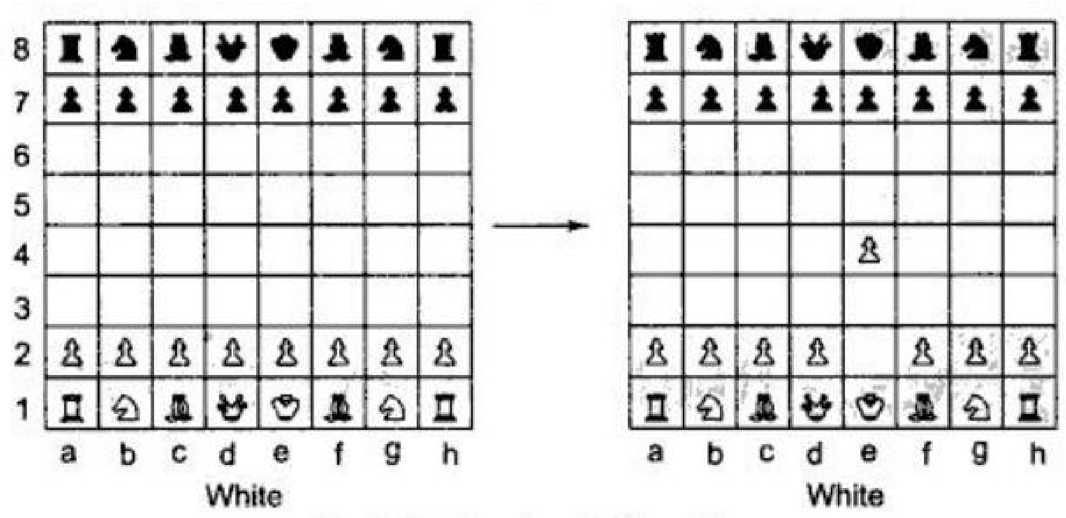
The primary focus here is on identifying and defining problems, as well as understanding the problem-solving methods used by AI. To effectively work on a problem, the following steps must be taken:

1.Defining the Problem Precisely: This involves specifying both the initial state and the goal state, along with all the constraints.
2.Analyzing the Problem: Some problems can be tackled in several ways, and analysis helps in selecting the best approach.
3.Evaluating Knowledge: The knowledge related to the problem must be identified and analyzed to effectively solve it.
4.Choosing Appropriate Problem-Solving Techniques: There are multiple ways to approach problems, and one should select the most efficient technique based on the analysis and knowledge available.

The rest of this section introduces state space search. This approach defines a problem as a set of states with the goal of moving from the initial state to a goal state by applying a series of actions or operations.

Diagram "One Legal Chess Move," illustrates a board position and one possible move within the game.

In this example:



- The goal is to move a **white pawn** from one position (e.g., squarefile E, rank 2) to another (e.g., squarefile E, rank 4).
- The rule used here ensures the move is legal by confirming that the target square is empty.

The problem of **chess** is described as moving around in a **state space**, where each move corresponds to a legal state change. This state space consists of all possible board positions, with the objective being to find the best series of moves that leads to a goal state, such as checkmate.

The **figure** illustrates how the problem is broken down into a series of smaller steps that move through the state space by applying rules. The example of the **chess move** highlights how the system moves from one state to another by applying a rule (moving a piece) that satisfies specific conditions (like an empty target square).

1. State Space Representation:

The **state space representation** is a formal method of defining a problem in terms of a set of possible states and the operations that can be applied to move between those states. This method helps structure problems by:

1.Allowing Formal Problem Definition: The state space representation provides a structured way to convert a problem into a set of permissible operations, moving from the initial state to the goal state.
2.Defining a Process for Solving Problems: It represents the solution process as a combination of known techniques, generalizing how to move from the current state to the goal state.
Search algorithms can then be applied to explore the state space and find an optimal path to the solution.

The state space representation is a common approach in AI, and while it is illustrated using **chess**, it can be generalized to many other problem types, including puzzles and optimization tasks.

2. Types of Heuristic Techniques

(A) Greedy Best-First Search

This method selects the next step based on the most promising option at the current stage.

Example:A traveler choosing the shortest distance at each step, even if it doesn’t guarantee the shortest total journey.

(B) Hill Climbing Algorithm

Starts from an initial state and continuously moves towards a better state based on a heuristic evaluation.

Limitations: Can get stuck in **local optima**, meaning it might not find the best solution overall.

(C)Best-First Search

- Expands the node with the lowest estimated cost using a heuristic function.
- Often used in **pathfinding and AI decision-making**.

(D) A Algorithm*

Combines heuristic search with cost-based search.

Uses an evaluation function: f(n)=g(n)+h(n)f(n) = g(n) + h(n)

g(n)g(n) = actual cost from start to node **n**.

h(n)h(n) = estimated cost from **n** to the goal.

Example:Google Maps Navigation: A* helps find the fastest route by considering both travel time so far (g(n)g(n)) and estimated time remaining (h(n)h(n)).

3. Application of Heuristics in Different Domains

Examples of **heuristic functions** in different problem-solving scenarios:

(A) Chess

Chess heuristics evaluate board positions based on:

- **Material advantage** (counting the total value of pieces).
- **Positional strength** (e.g., controlling the center of the board).
- **Threat evaluation** (determining if a piece is under attack).

(B) Traveling Salesman Problem (TSP)

- A classic optimization problem where a salesperson must visit **n** cities while minimizing travel distance.
- A heuristic approach estimates:
  - **Sum of distances for unvisited cities** to prioritize paths.

(C) Tic-Tac-Toe

- Heuristic strategies focus on:
  - **Winning in the next move**.
  - **Blocking an opponent's winning move**.
  - **Maximizing future winning opportunities**

5. Limitations of Heuristics

Despite their usefulness, heuristics come with drawbacks:

- **Lack of Guarantee for Optimality**: May not always find the best solution.
- **Possibility of Getting Stuck in Local Optima**: Some methods, like hill climbing, may stop at a "good enough" solution rather than the global best.
- **Dependence on Problem-Specific Knowledge**: A poorly chosen heuristic can lead to inefficient results.

Problem Characteristics
Introduction to Heuristic Search & Problem Solving

1. Heuristic search is a widely applicable method for solving a variety of problems.
2. Different techniques work best for specific problems, so problem analysis is crucial.

Key Dimensions of Problem Analysis

1. **Decomposability**: Can the problem be broken into smaller subproblems?
2. **Solution Ignorance**: Can incorrect solutions be ignored or undone?
3. **Universality**: Is the problem universal or does it depend on the environment?
4. **Predictability**: Is the solution’s behavior predictable?
5. **Path Dependency**: Does finding a solution require comparison with all other options?
6. **Information Representation**: Does the problem require a model of the world?
7. **Knowledge Requirement**: Does solving it require significant domain knowledge?
8. **Computation vs. Interaction**: Is the problem purely computational or does it require human interaction?

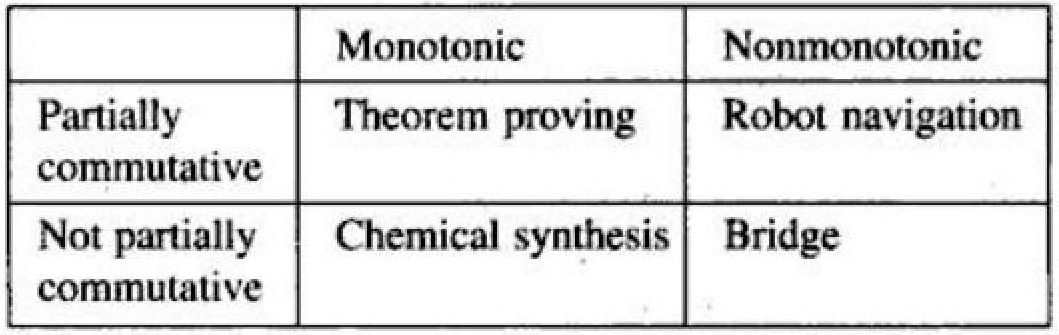
Production System Categories

Role of Production Systems in Problem Solving

1.Production systems form the basis of various problem-solving techniques.
2.Helps define solutions in terms of rules and actions.

Four Types of Production Systems

1.Monotonic: The problem-solving process does not undo previous decisions (e.g., theorem proving).
2.Partially Commutative: The order of operations matters but can sometimes be altered (e.g., chemical synthesis).
3.Nonmonotonic: Decisions can be undone, requiring backtracking (e.g., robot navigation).
4.Not Partially Commutative: Strict order of operations is required (e.g., bridge- building).



	Monotonic	Nonmonotonic
Partially commutative	Theorem proving	Robot navigation
Not partially commutative	Chemical synthesis	Bridge

Implementation Considerations for Production Systems

Commutative & Non-Commutative Systems

1. Commutative systems are easier to implement because state transitions can be reordered.
2. Non-commutative systems require careful planning, especially for irreversible changes.

Practical Applications

- **8-Puzzle & Blocks World**: Partially commutative problems where move order affects efficiency.
- **Chemical Processes**: Non-commutative because adding components in the wrong sequence changes outcomes.

Issues in the Design of Search Programs:
Search Trees: Nodes represent problem states; paths show possible moves. Exploring all paths is often impractical due to size.

Key Challenges: Choosing between forward or backward reasoning, applying rules effectively, and balancing efficiency with completeness.

Forward vs. Backward Reasoning: Forward goes from start to goal; backward starts from the goal to find starting points.

Focus: Efficient problem representation and search strategies are crucial for effective search programs

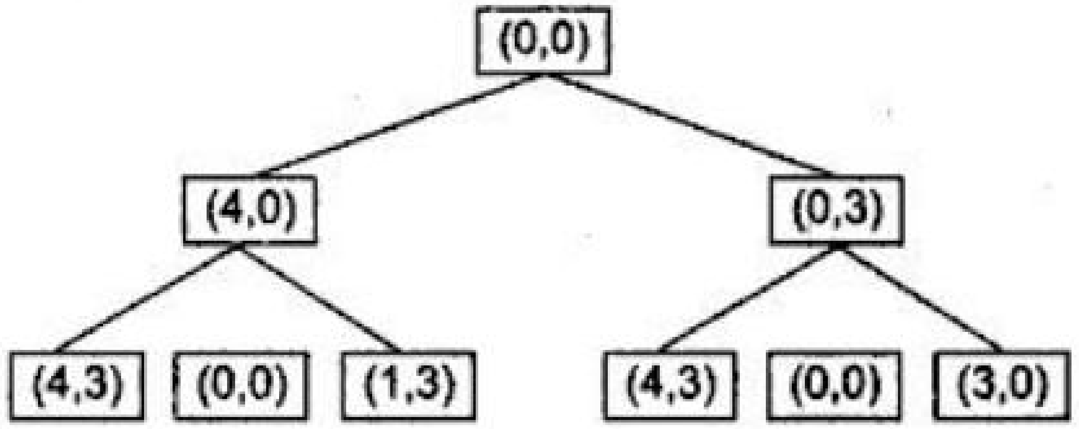


Fig. 2.18 *A Search Tree for the Water Jug Problem*

Search Trees vs. Graphs: Search trees may expand duplicate nodes, causing redundancy. Search graphs handle this by merging paths to the same node.

Redundancy Issue: In Fig. 2.18, the (0,0) state repeats unnecessarily.

Graph Representation: Fig. 2.19 shows a search graph for the same problem, merging paths to avoid redundancy.

Efficient Search: Graph-based search requires extra memory to track visited nodes but reduces repetitive processing.

Balanced Approach: Balancing memory use and search efficiency is crucial for effective search program design.

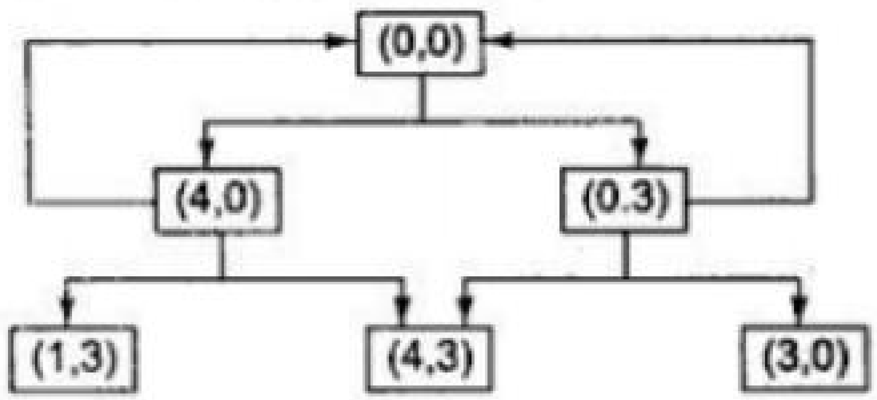
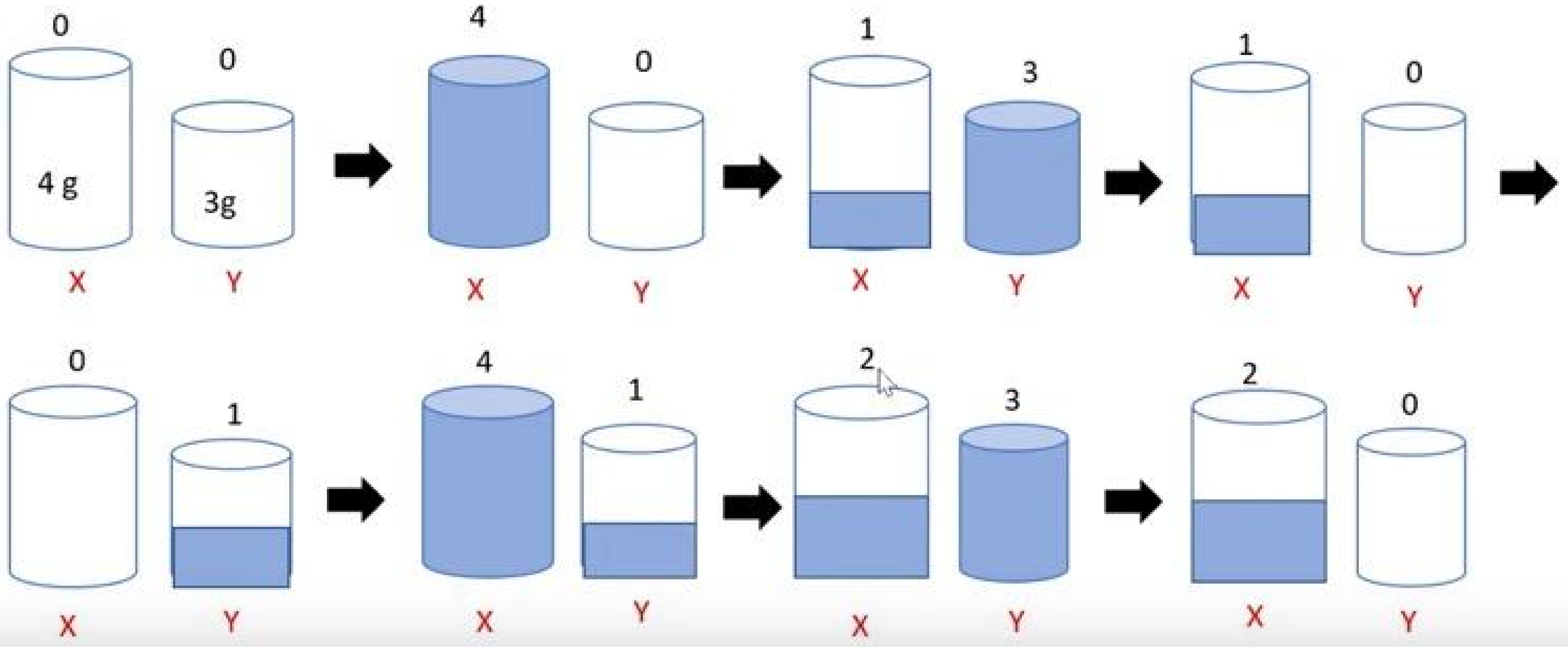


Fig. 2.19 *A Search Graph for the Water Jug Problem*

Solution



Algorithm: Check Duplicate Nodes

1. Examine the set of nodes that have been created so far to see if the new node already exists.
2. If it does not-simply add it to the graph just as for a tree.
3. If it does already exist, then do the following:
   (a) Set the node that is being expanded to point to the already existing-node corresponding to its successor rather than to the new one. The new one can simply be thrown away.
   (b) If you are keeping track of the best (shortest or otherwise least-cost) path to each node, then check to see if the new path is better or worse than the old one. If worse, do nothing. If better, record the new path as the correct path to use to get to the node and propagate the corresponding change in cost down through successor nodes as necessary.

The problems below serve as benchmarks for evaluating search techniques and heuristic approaches.

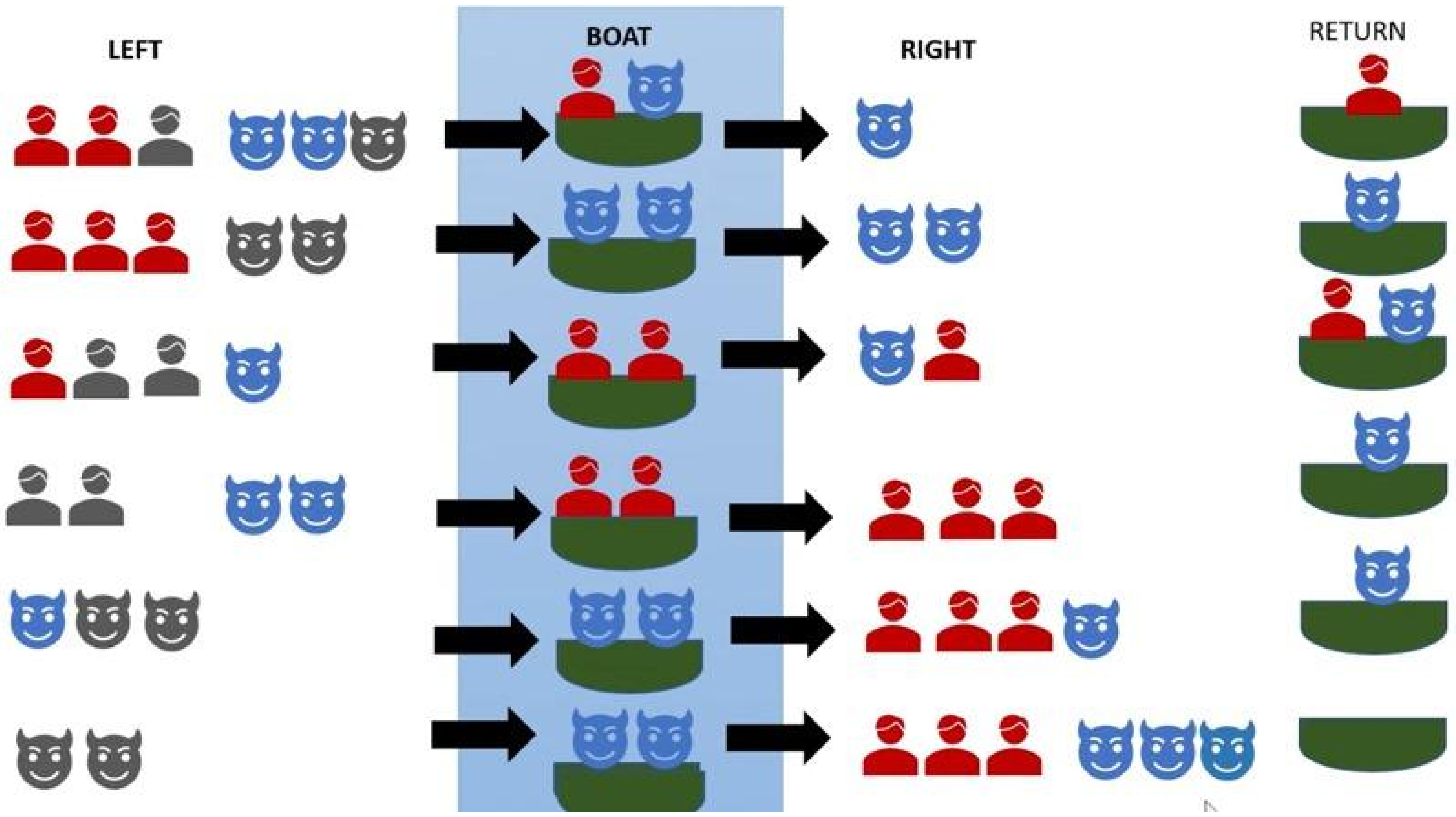
1. Missionaries and Cannibals Problem

- Three missionaries and three cannibals need to cross a river using a boat.
- The boat can only hold two people at a time.
- At no point can the number of cannibals be greater than the number of missionaries on either side of the river.
- The goal is to find a sequence of moves that ensures all safely cross without missionaries being eaten.

2. Tower of Hanoi

- A mathematical puzzle with three rods and multiple disks of different sizes.
- The goal is to move the stack of disks from one rod to another, following these rules:
  - Only one disk can be moved at a time.
  - A larger disk cannot be placed on a smaller disk.
  - The disks must be moved using an auxiliary rod.
- The minimum number of moves required for **n** disks is $\mathbf{2^n - 1}$.

Missionaries and Cannibals Problem



3. Monkey and Bananas Problem

- A monkey is in a room where bananas are hanging from the ceiling.
- A chair and a stick are present in the room, but the bananas are out of reach.
- The challenge is for the monkey to find a way to use the objects to reach and obtain the bananas.
- This problem is often used to study **planning and reasoning** in artificial intelligence.

4.Cryptarithmetic Problems

- These are arithmetic problems in which digits are replaced with unique letters.
- Examples:
    - SEND + MORE = MONEY
    - DONALD + GERALD = ROBERT
    - CROSS + ROADS = DANGER
- The goal is to assign a unique decimal digit (0-9) to each letter so that the arithmetic equation holds true.
- No two different letters can have the same digit.

```
  S E N D
+ M O R E
---------
M O N E Y
```

Character	Code
S	9
E	5
N	6
D	7
M	1
O	0
R	8
Y	2

HEURISTIC SEARCH TECHNIQUES:

Definition & Importance of Heuristic Search

- Many AI problems are **too complex** to be solved by **direct techniques**.
- They require **appropriate search methods** with heuristic techniques to **guide** the search efficiently.
- Heuristic search methods **estimate the best direction** to reach a solution faster.
- These methods form the **core of most AI systems** despite their limitations.

Why Heuristic Search is Needed?

- **Direct search methods** are impractical due to **combinatorial explosion** (exponential growth of possibilities).
- Heuristic search **overcomes complexity** by applying **domain-specific knowledge** to **reduce search space**.
- Although heuristic methods are not always **guaranteed to find the optimal solution**, they are often **effective enough** in practice.

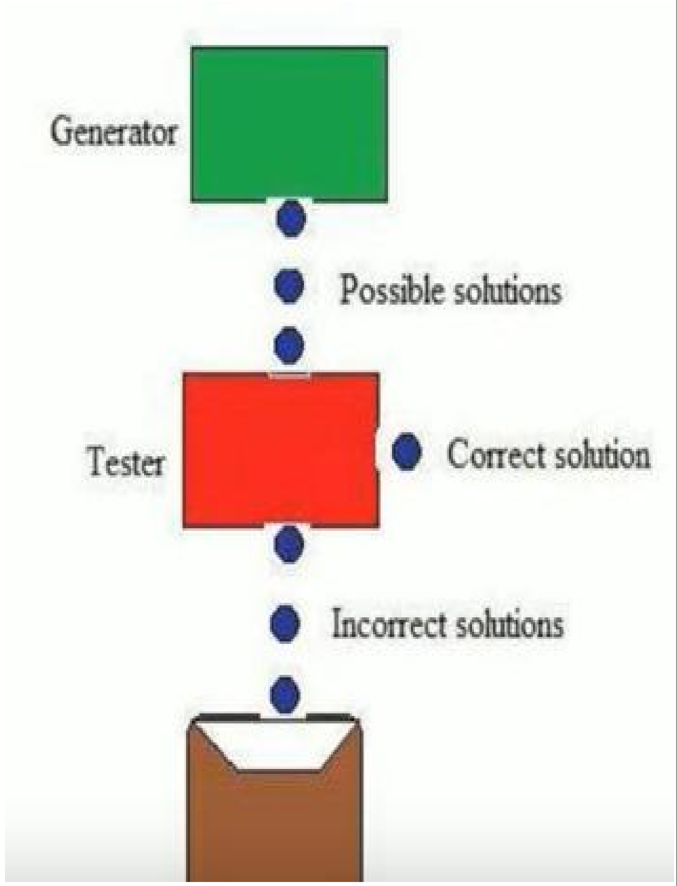
Heuristic Search Methods

- **Generate-and-Test** – Generates possible solutions and tests them for correctness.
- **Hill Climbing** – Moves towards increasing improvement (local optimization).
- **Constraint Satisfaction** – Solves problems by **eliminating** invalid possibilities based on constraints.
- **Best-First Search** – Uses a heuristic to **choose the most promising node** first.
- **Means-Ends Analysis** – Reduces the gap between the current state and goal by taking intermediate steps.

Generate-and-Test Method

Algorithm Steps:

1. **Generate a possible solution**
   - The solution can be a **random guess** or a **systematic approach** based on a given problem.
   - In some cases, it starts **from a predefined state** rather than a random guess.
2. **Test the solution**
   - Check if the solution **meets the desired conditions**.
3. **Repeat if necessary**
   - If the solution **fails**, generate a new one and test again until the correct solution is found.



Hill Climbing

2.Hill Climbing Overview

- **Hill Climbing** is a **variant of generate-and-test** where **feedback** is used to decide the next move.
- It **improves efficiency** by **choosing moves** that lead to a **better state** rather than random generation.
- Instead of **blindly testing solutions**, a **heuristic function** helps **evaluate each move**.
- This technique is particularly useful when:
    - **Perfect domain knowledge is unavailable**.
    - A **heuristic function can estimate the quality** of a state.

1. Simple Hill Climbing Algorithm

Steps:

1. **Evaluate the initial state**. If it's the **goal**, return it; otherwise, continue.
2. **Loop until a solution is found or no more moves are available:**

- o Apply **all possible operators** to generate **new states**.
- o Choose the **best state** based on the heuristic function.
- o If no better state exists, **terminate the process**.

Decision Cases:

- If **new state = goal**, return it.
- If **new state is better than the current one**, move to it.
- If **no better state exists**, terminate (local maximum).

Example Use Case:

- **Four-colored block puzzle**
  - Each **state transformation** changes the **color** of a block.
  - The **heuristic function** evaluates how **close** the arrangement is to the goal
  - The algorithm **selects moves** that **increase the value of the function**.

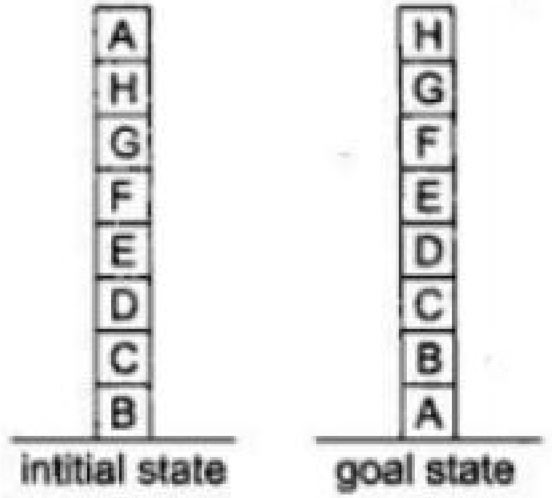


Fig. 3.1 *A Hill-Climbing Problem*

Limitations of Hill Climbing

- **Local Maxima:** Gets stuck in **suboptimal solutions** if no better move is available.
- **Plateau Problem:** Flat areas with **no clear direction** can cause the search to **halt**.
- **Ridges:** Paths to the solution may require **temporary backtracking**, which hill climbing **does not allow**.

Steepest-Ascent Hill Climbing

Steepest-Ascent Hill Climbing (also called **Gradient Search**) is an improvement over **Simple Hill Climbing**. Instead of selecting the first better move, it considers **all possible moves** and picks the best one.

Algorithm:

1. Evaluate the initial state:
   1. If it is a **goal state**, return it and quit.
   2. Otherwise, continue.

2. Loop until a solution is found or no better state exists:

   1. For each possible **operator** applied to the current state:
      1. Apply the operator and generate a new state.
      2. Evaluate the new state.
   2. **Select the best state** among the generated ones.
   3. If this new state is **better than the current state**, move to it.
   4. If no improvement is possible, terminate.

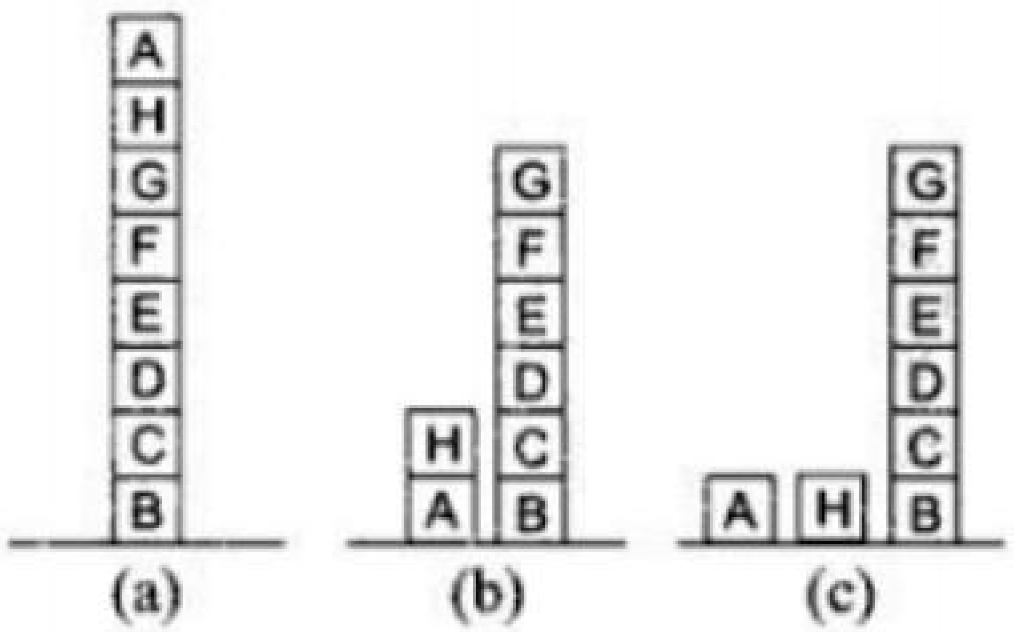


Fig. 3.2 *Three Possible Moves*

Example Problem:

- Consider a **colored block puzzle**.
- To solve it, different **perturbations** (small changes) are applied to see which move gives the best improvement.
- **Trade-off**: Steepest-Ascent Hill Climbing often requires more **evaluation steps** but finds better solutions than simple hill climbing.

Challenges & Solutions in Hill Climbing

1.Local Maxima

- A state that is better than its neighboring states but **not the global best**.
- **Solution:** Introduce **random jumps** to escape local maxima.

2.Plateaus

- A **flat region** where all neighboring states have the **same heuristic value**.
- **Solution:** Use **random exploration** or **backtracking**.

3.Ridges

- A narrow peak where small movements **do not improve the state**, making it hard to climb.
- **Solution:** Apply **multiple rules simultaneously** or allow **sideways movement**.

Additional Techniques for Improvement

Backtracking

- Saves previous states and moves back when stuck.
- Helps in escaping **plateaus and local maxima**

Using Randomness

- Instead of **always picking the best move**, sometimes choose **random moves**.
- Can prevent getting stuck in **local maxima**.

Multiple Rule Application

- Instead of checking **one rule at a time**, apply **several** before deciding.
- Helps when there are complex interdependencies in the problem space.

SIMULATED ANNEALING (SA)

Simulated annealing is a variation of **hill climbing** in which, at the beginning of the process, some **downhill moves** are allowed. This helps to **explore the solution space** early on, so the final solution does not depend too much on the **starting state**. This reduces the chances of getting trapped in **local maxima**, **plateaus**, or **ridges**.

Concept & Inspiration

SA is based on an analogy with the **annealing process** in metallurgy.

- In physical annealing, a metal is heated to a **high energy state** and then gradually cooled.
- During cooling, the **energy of the system decreases**, and the system settles into a low-energy **stable state** (crystal structure).
- In the simulated version, this translates to **gradually reducing the acceptance of worse solutions** as the process progresses.

Mathematical Model

The probability of accepting a transition to a higher energy state is given by:

ΔE = Change in energy (or cost function)
T = Temperature (a control parameter)
k = Boltzmann’ s constant

$$P(\Delta E) = e^{-\frac{\Delta E}{k * t}}$$

This equation implies:

- **At high temperatures**, the probability of accepting a bad move is high.
- **At low temperatures**, the probability is low, leading to a more refined search.

Working of Simulated Annealing

1. The system starts at a **high temperature**.
2. Moves are **randomly selected** and accepted based on their energy difference.
3. **Bad moves are allowed** with a probability that decreases over time.
4. Over time, the **temperature is reduced**, leading to a **more refined search**.
5. The process converges to a **global optimum**.

Algorithm: Simulated Annealing

Evaluate the initial state:

- If it is the goal state, return it.
- Otherwise, continue with the current state.

Initialize:

- **BEST-SO-FAR** = Current state.
- **Set initial temperature** using an annealing schedule.

Iterate until solution is found:

- If no new operators exist, terminate.
- Choose a **random state** from available operators.
- Compute the energy difference:$\Delta E = f(\text{value of new state}) - f(\text{value of current state})$
- If **ΔE is positive**, update the current state.
- If **ΔE is negative**, accept the state with probability **$e^{(-\Delta E/T)}$**.

Reduce temperature according to the **annealing schedule**. **Continue until temperature approaches zero**.

Key Differences from Hill Climbing

1. **Move selection**: SA can accept worse moves, unlike hill climbing.
2. **Temperature-dependent decisions**: SA uses an annealing schedule to reduce temperature.
3. **Randomized behavior**: SA is more exploratory, avoiding local optima.

Choosing an Annealing Schedule

- A good annealing schedule is **crucial** for performance.
- **Temperature T should decrease gradually**.
- As **$T \to 0$**, SA behaves like hill climbing.
- The best schedules balance **exploration and convergence**.

Applications of Simulated Annealing

- Traveling Salesman Problem
- Circuit Design Optimization
- Scheduling Problems
- Machine Learning (e.g., Neural Networks)

BEST-FIRST SEARCH

- Best-First Search is an advanced search method that combines the benefits of depth-first and breadth-first search.
- It efficiently determines the best node to explore next using heuristic functions.

OR Graphs

- Depth-First Search is efficient when a solution exists and does not get trapped in infinite paths.
- Breadth-First Search is useful for problems with dead-end paths, as it explores all possibilities at a given level.
- Best-First Search merges these approaches, selecting the most promising node at each step.

Best-First Search Process

1. At each step, the algorithm chooses the most promising node using a heuristic function.
2. If the node is a solution, the search ends.
3. Otherwise, the node generates successor nodes which are evaluated and prioritized.
4. The most promising successor is explored first, while other nodes remain in memory for future exploration.

Example (Figure 3.3)

- The diagram illustrates a step-by-step execution of Best-First Search on a graph.
- The algorithm selects the most promising node based on the heuristic values.
- Unlike hill climbing, where only the best move is chosen at each step, Best-First Search keeps track of alternative paths for future exploration.

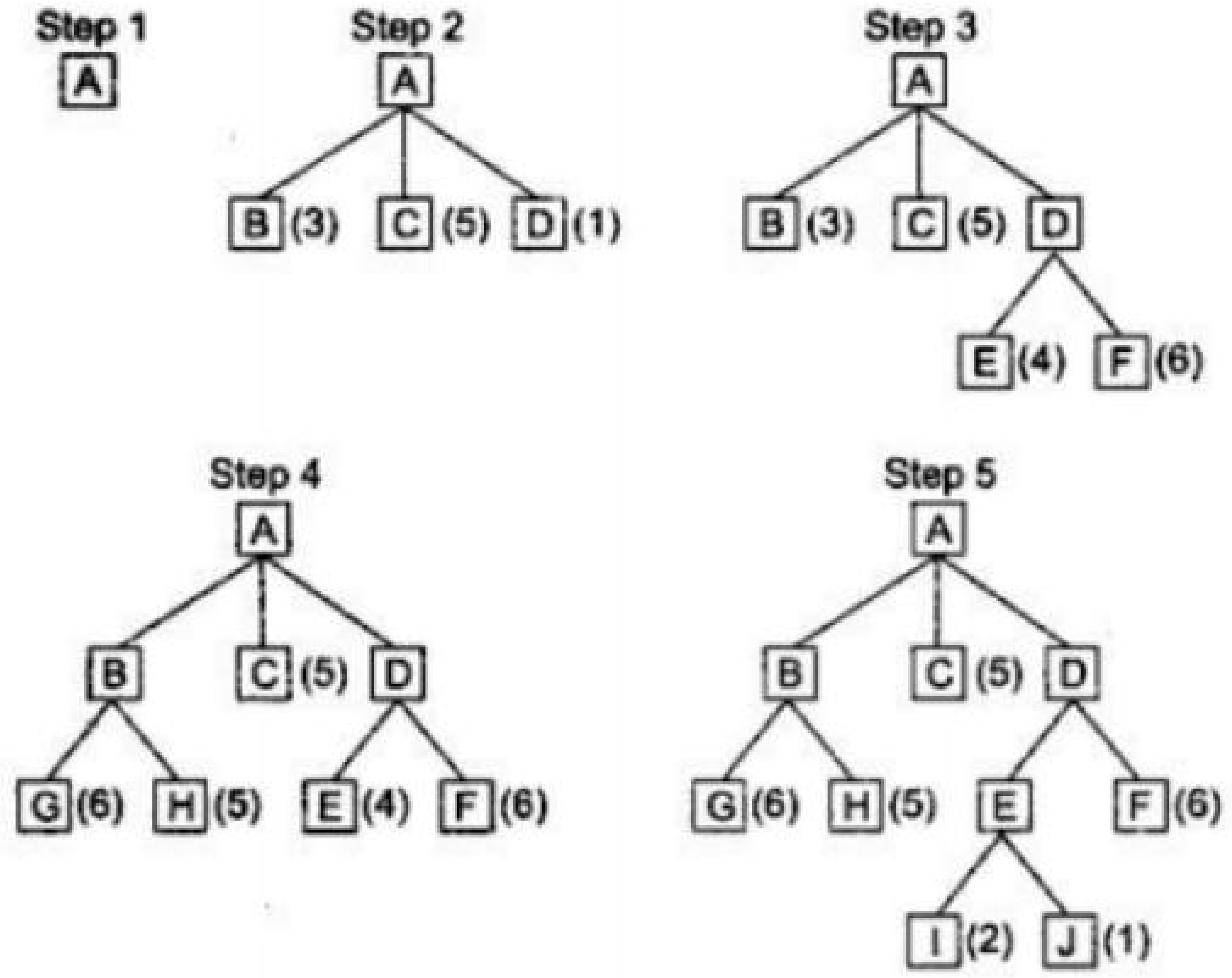


Fig. 3.3 *A Best-First Search*

Implementation of Best-First Search in Graphs

- The graph-search method ensures nodes are not revisited unnecessarily.
- Two lists are maintained:

    1. **OPEN**: Nodes yet to be explored.
    2. **CLOSED**: Nodes that have already been expanded.

- **OPEN** list prioritizes nodes with better heuristic values.

Algorithm: Best-First Search

1. Start with OPEN containing the initial state.
2. Until a goal is found or OPEN is empty:

    - Select the best node from OPEN.
    - If it is a goal, return success.
    - Otherwise, generate its successors and:

        - Evaluate each successor.
        - If it has been encountered before, update parent information.
        - Otherwise, add it to OPEN and record its heuristic value.

The A Algorithm*

Introduction

The *A algorithm** is an improved version of the **Best-First Search**. It was introduced

by **Hart et al. (1968, 1972)**.

The algorithm uses a combination of:

1. **g(n)** → The cost from the **starting node** to the current node.
2. **h(n)** → A heuristic function that estimates the cost from the **current node** to the **goal**.
3. **f(n) = g(n) + h(n)** → The estimated total cost from the start node to the goal.

The **main objective** of A* is to balance between:

1. **Greedy search** (which follows the heuristic h(n) to the goal).
2. **Uniform Cost Search** (which explores all possibilities optimally).

*Steps of the A Algorithm**

The A* algorithm follows these **systematic steps**:

Step 1: Initialization

- **Create two lists**:
  - OPEN: Contains the nodes that need to be explored.
  - CLOSED: Contains the nodes that have already been visited.
- **Start with the initial node**:
  - Set g(initial) = 0.
  - Set h(initial) = estimated cost to goal.
  - Compute f(initial) = g + h.
  - Add the initial node to OPEN.

Step 2: Main Search Process

- Repeat the following **until the goal node is found** or OPEN is empty:

1. Pick the node in OPEN with the **lowest f-value** (lowest estimated total cost). Call this node BESTNODE.
2. If BESTNODE is the **goal**, return the solution.
3. Otherwise, **move** BESTNODE **to** CLOSED (marking it as visited).
4. **Generate the successors (child nodes) of** BESTNODE.

Step 3: Handling Successors

For each SUCCESSOR of BESTNODE:

Compute its cost:

- g(SUCCESSOR)=g(BESTNODE)+cost from BESTNODE to SUCCESSOR
- f(SUCCESSOR)=g(SUCCESSOR)+h(SUCCESSOR)

Check if SUCCESSOR **is already in** OPEN **or** CLOSED:

- If SUCCESSOR is already in OPEN:
    - Compare the **new g-value** with the existing g-value.
    - If the new g-value is lower, update it and **change the parent** of SUCCESSOR to BESTNODE.

- If SUCCESSOR is already in CLOSED:
    - If the new path to SUCCESSOR is better (lower cost), move it **back to** OPEN and update the parent.
- If SUCCESSOR is **not in either list**, add it to OPEN.

Step 4: Repeat Until Goal is Found

- Keep expanding nodes with the lowest **f-value**.
- Stop when the **goal node is removed from** OPEN.
- The solution is the **path from the start node to the goal node**.

Key Observations about A*

*1. A is Optimal (It Always Finds the Best Solution)**

- The algorithm **expands nodes in increasing order of total cost (f-value)**.
- If the **heuristic function h(n) is admissible** (it never overestimates the actual cost), A* will **always** find the optimal (shortest) path.

*2. A is Efficient**

- A* only expands the **minimum number of nodes necessary** to guarantee an optimal solution.
- The efficiency depends on the quality of the heuristic h(n).
  - If h(n) = 0, A* behaves like **Uniform Cost Search** (slow but guarantees optimality).
  - If h(n) is **perfect**, A* follows the optimal path **directly**.

3. Relationship Between Heuristic h(n) and Performance

- **Underestimating h(n)** ensures optimality but may explore more nodes.
- **Overestimating h(n)** can speed up search but may lead to a **non-optimal path**.

Illustrative Example of A*

- Consider **Figure 3.4**, where A* is applied to a graph.
- Each node has:

  - **g(n)**: Cost from the start node to n.
  - **h(n)**: Estimated cost from n to the goal.
  - **f(n) = g(n) + h(n)**.

- The algorithm **always picks the node with the lowest f-value**.

Example Cases

Case 1: Underestimating h(n)

- If h(n) is too low, A* **expands unnecessary nodes** but still finds the optimal solution.

Case 2: Overestimating h(n)

•If h(n) is too high, A* may **miss the shortest path** and become a Greedy Search.

Here’s a detailed explanation of the two uploaded images discussing **Agendas and Agenda- Driven Search**:

Agendas in Search Algorithms

- In search problems, multiple paths may lead to the same node independently, meaning different routes can result in the same final state.
- The idea of **an agenda** is to prioritize certain tasks based on their importance rather than exploring everything equally.
- An example is given where a researcher (AM) is provided with facts about number theory and must determine which ones are worth exploring further.
- Instead of selecting at random, AM needs **heuristics** to decide which ideas are more interesting and likely to lead to new useful discoveries.
- This approach involves maintaining a list (**the agenda**) of promising directions, ranking them, and choosing the best ones to pursue.
- Tasks with a higher probability of leading to interesting results are given priority.
- The **agenda-driven system** helps guide research and decision-making in artificial intelligence.

Algorithm for Agenda-Driven Search

- The algorithm structures how tasks are selected and executed:

1. **Pick a task** from the agenda.
2. **Execute the task**, considering resource constraints.
3. **If the task is already in the agenda, discard duplicates** (ensuring efficiency).
4. **Compute the justification** for adding the task and adjust its ranking in the agenda.
5. **Tasks are weighted by multiple justifications** and combined to give an overall rating.

Managing the Agenda Efficiently

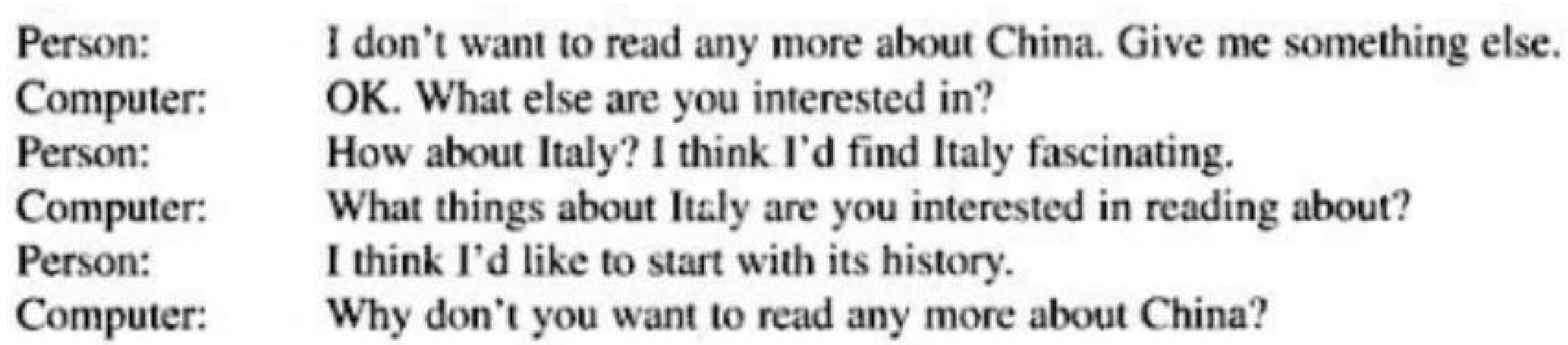
- **Choosing the best task** is crucial—tasks should not be picked randomly but rather based on justifications.
- When a task is justified multiple times, **it moves higher in priority**.
- If an already existing justification is strengthened, it is compared to existing ones before reordering.
- **Negative justifications** may also lower a task’s priority.

Example of Human-Like Decision Making

- The text presents a **conversation between a person and a computer**, demonstrating **how agenda-driven reasoning works**.
- The computer asks about a person’s interest in reading a book and attempts to determine the reason behind that interest.
- This mirrors how an AI system can **track justifications, re-rank options, and refine its knowledge dynamically**.
- The AI must balance **too many justifications** (overwhelming options) and **too few justifications** (lack of meaningful decision-making).

5. Challenges of Agenda-Driven Systems

- **Efficiency concerns**: Sorting and ranking tasks require computational effort.
- **Uncertainty in justification**: Some justifications might be weak, affecting prioritization accuracy.
- **Application in AI**: Despite challenges, agenda-driven search is useful in AI for **intelligent decision-making, prioritization, and knowledge discovery**.



Person:	I don't want to read any more about China. Give me something else.
Computer:	OK. What else are you interested in?
Person:	How about Italy? I think I'd find Italy fascinating.
Computer:	What things about Italy are you interested in reading about?
Person:	I think I'd like to start with its history.
Computer:	Why don't you want to read any more about China?

Figure 3.6: A Simple AND-OR Graph

This figure presents a basic AND-OR graph, demonstrating how a goal can be achieved by either:

1. **Stealing a test**
2. **Earning money and then buying the test**

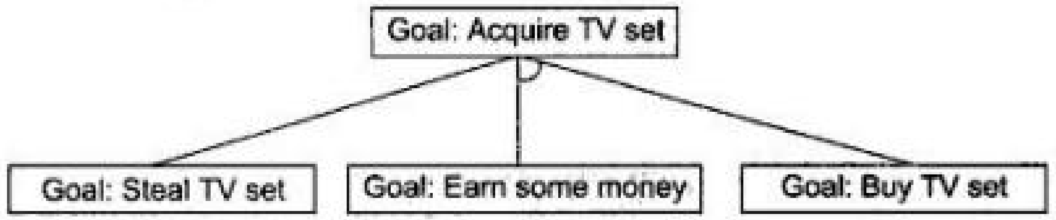


Fig. 3.6 *A Simple AND-OR Graph*

Each of these strategies is represented as branches in the graph:

- **OR branch:** Either "Steal Goal" or "Buy Goal" can be followed.
- **AND branch:** "Earn Money" and "Buy Test" must both be achieved together.

Explanation: To find a solution, an algorithm must consider the constraints imposed by AND branches. Unlike best-first search, the algorithm used for AND-OR graphs needs to:

Algorithm for Problem Reduction

This section introduces an algorithm specifically designed for problem reduction within AND-OR graphs.

1. **Initialize the graph at the starting node.**
2. **Loop until a solution is found:**

    - Expand the lowest-cost node and track the cumulative cost.
    - Assign **FUTILITY** to nodes that exceed a threshold cost, meaning they are too expensive to pursue.
    - Mark any node as **SOLVED** if all its successors are solved.
    - If a node is marked as **SOLVED**, propagate this status backward.

Figure 3.7: AND-OR Graph Expansion

This figure illustrates an example where nodes are expanded based on cost. The best path to follow depends on whether **all** required subproblems in an AND branch are solved or not.
The algorithm avoids pursuing futile paths by evaluating cost estimates.

The Operation of Problem Reduction

Figure 3.8: Step-by-Step Expansion in an AND-OR Graph

This figure demonstrates the process of expanding nodes:

- **Before Step 1:** The initial tree is evaluated.
- **Before Step 2:** The most promising node is expanded.
- **Before Step 3 & 4:** Additional nodes are expanded while considering costs.

When a Longer Path is Better

Unlike standard OR search, problem reduction sometimes benefits from choosing a longer path if it leads to a guaranteed solution.

Figure 3.9: Why a Longer Path Might Be Preferred

- In traditional search, shorter paths are always preferred.
- In an AND-OR graph, a shorter path may **not** always be the best choice if it leads to an unsolvable state.

Interacting Subgoals

One of the limitations of the algorithm is that it **does not consider interactions between subgoals**. This can lead to inefficient solutions.

Figure 3.10: Interaction Between Subgoals

- Some problems require considering the **relationship** between subgoals.
- The example in the figure demonstrates a case where ignoring interactions leads to suboptimal results.
- More advanced algorithms in **Chapter 13** address these issues by incorporating **subgoal interactions** into problem-solving.

*AO Algorithm Explanation**

The **AO*** (AND-OR Star) algorithm is a refinement of problem reduction techniques used in AI. It is a best-first search algorithm specifically designed for AND-OR graphs, which are useful for solving problems that can be broken down into subproblems.

Unlike the standard **A*** algorithm that maintains **OPEN** and **CLOSED** lists, AO* works with a **GRAPH**, which represents the search space. Each node in the graph is associated with an **h’ value**, representing the estimated cost from that node to the goal.

A key feature of AO* is that it propagates cost estimates **backwards through the graph** to ensure that the most promising path is chosen. The algorithm guarantees an optimal solution while minimizing the number of expanded nodes.

Algorithm AO* (Steps)

Initialize the Graph

1. The graph consists of an **initial node (INIT)** representing the starting point.
2. Compute the **h’ value** for INIT.

Iterate Until Solution is Found

If INIT is labeled **SOLVED** or its **h’ value** exceeds a threshold called **FUTILITY**, stop.

1. Otherwise, expand one of the most promising unexpanded nodes.

Node Expansion Process

1. Select a node from **INIT** and generate its successors.
2. Assign a cost estimate **h’** to each new successor.
3. If a successor is terminal (goal node), mark it **SOLVED**.
4. If it is not a goal node, compute the **minimum cost path**.

Update the Current Best Path

If the **h’ value** of a node changes, propagate this update **backward** to parent nodes.

1. Update **h’ values** iteratively until the best path stabilizes.

Final Solution

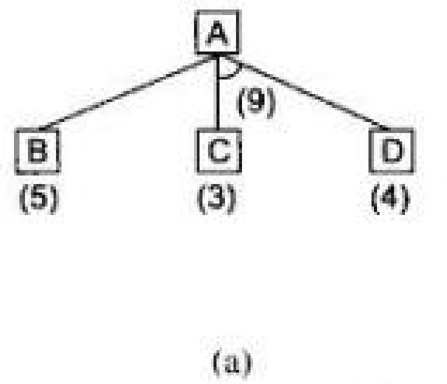
1. Once all nodes along the best path are **SOLVED**, the algorithm terminates.

Figures Explanation

Figure 3.11: The Unnecessary Backward Propagation

This figure illustrates a problem that can occur if **cost estimates are not correctly propagated backward**.

- In the figure, **path C** is clearly better than **path B**.
- However, if the cost at **B** is updated incorrectly without propagating it **back through A**, it can lead to an inefficient path being expanded.
- The cost should **always** be updated along the best path to ensure an optimal solution.



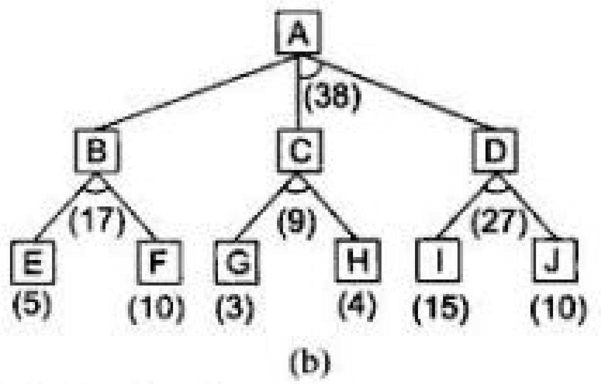


Fig. 3.7 *AND-OR Graphs*

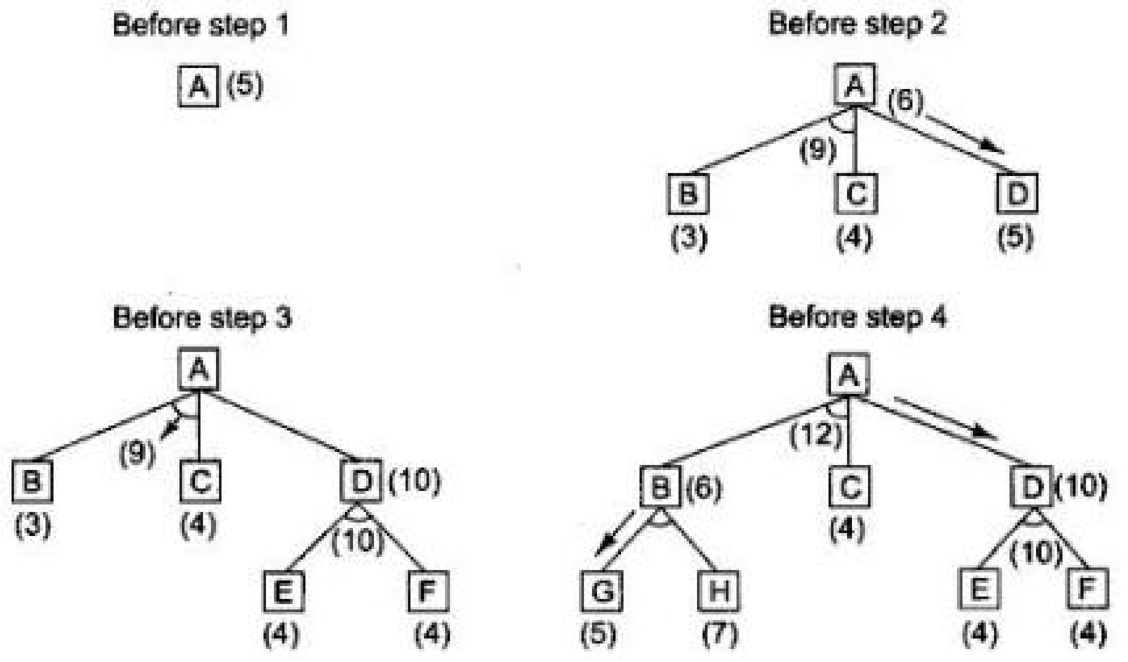


Fig. 3.8 *The Operation of Problem Reduction*

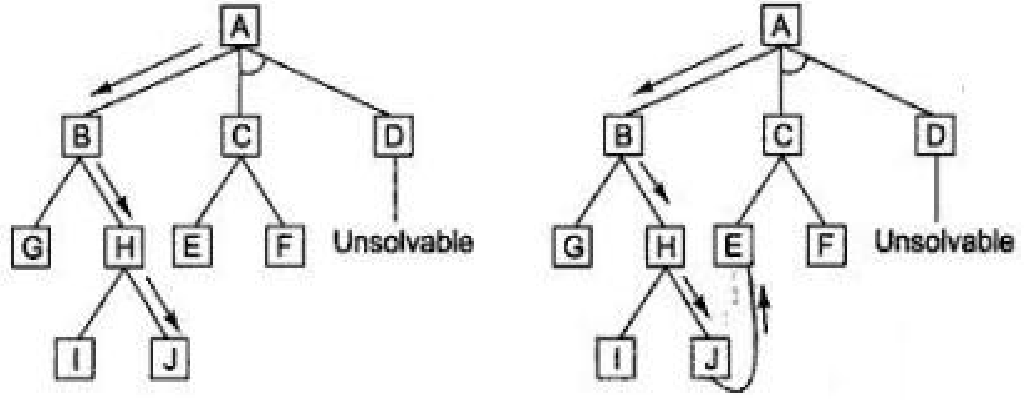


Fig. 3.9 *A Longer Path May Be Better*

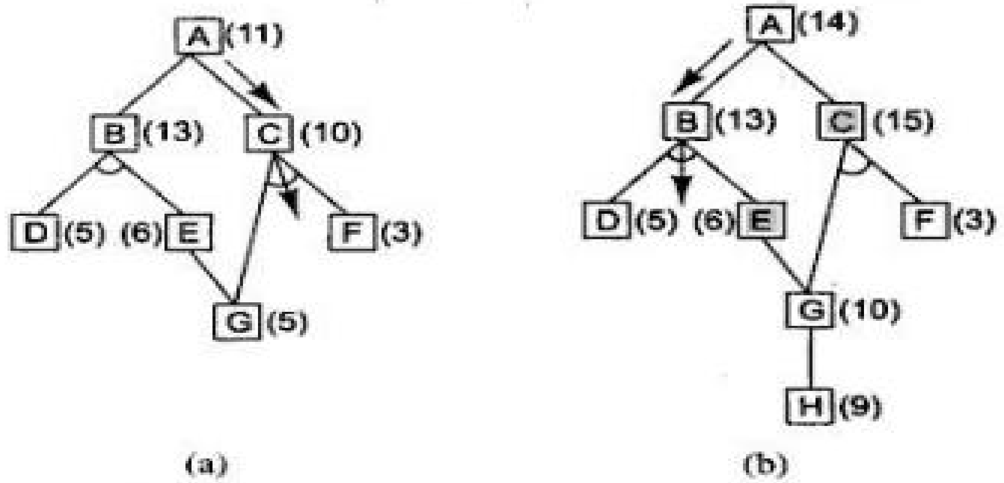


Fig. 3.12 *A Necessary Backward Propagation*

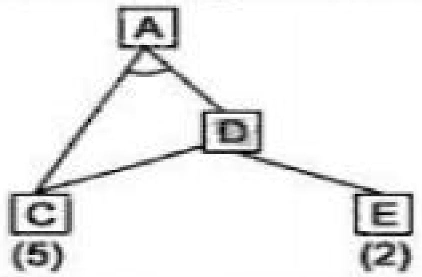


Fig. 3.10 *Interacting Subgoals*

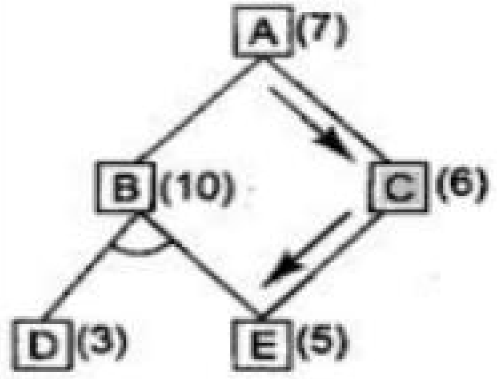


Fig. 3.11 *An Unnecessary Backward Propagation*

Figure 3.12: A Necessary Backward Propagation

This figure highlights why **backward cost propagation** is essential.

- When a **node's cost estimate is updated**, it should be **propagated up the tree** to reflect the most accurate path.
- Without this, the algorithm might **continue exploring suboptimal paths** instead of adjusting the search strategy to focus on the best path.

Constraint Satisfaction:

Definition of Constraint Satisfaction Problems (CSPs)

A **Constraint Satisfaction Problem (CSP)** is a type of AI problem where the goal is to **assign values** to variables while ensuring that all given **constraints** are met.

Key Components of CSPs

1. **Variables**: Elements that need values (e.g., digits in a cryptarithmetic problem).
2. **Domains**: Possible values each variable can take.
3. **Constraints**: Rules that limit how values can be assigned.

Examples of CSPs

Cryptarithmetic problems (e.g., SEND + MORE = MONEY).
- **Scheduling problems** (e.g., assigning meetings to rooms).
- **Graph coloring problems** (ensuring adjacent regions have different colors).
- **Design constraints** (like circuit board layouts).

Figure 3.13: Shows an example of a **cryptarithmetic problem** SEND + MORE = MONEY.

- Each letter represents a **unique digit**.
- The sum must be **mathematically valid**.

Problem:

```
  SEND
+ MORE
......
 MONEY
```

Initial State:

No two letters have the same value.
The sums of the digits must be as shown in the problem.

Fig. 3.13 *A Cryptarithmetic Problem*

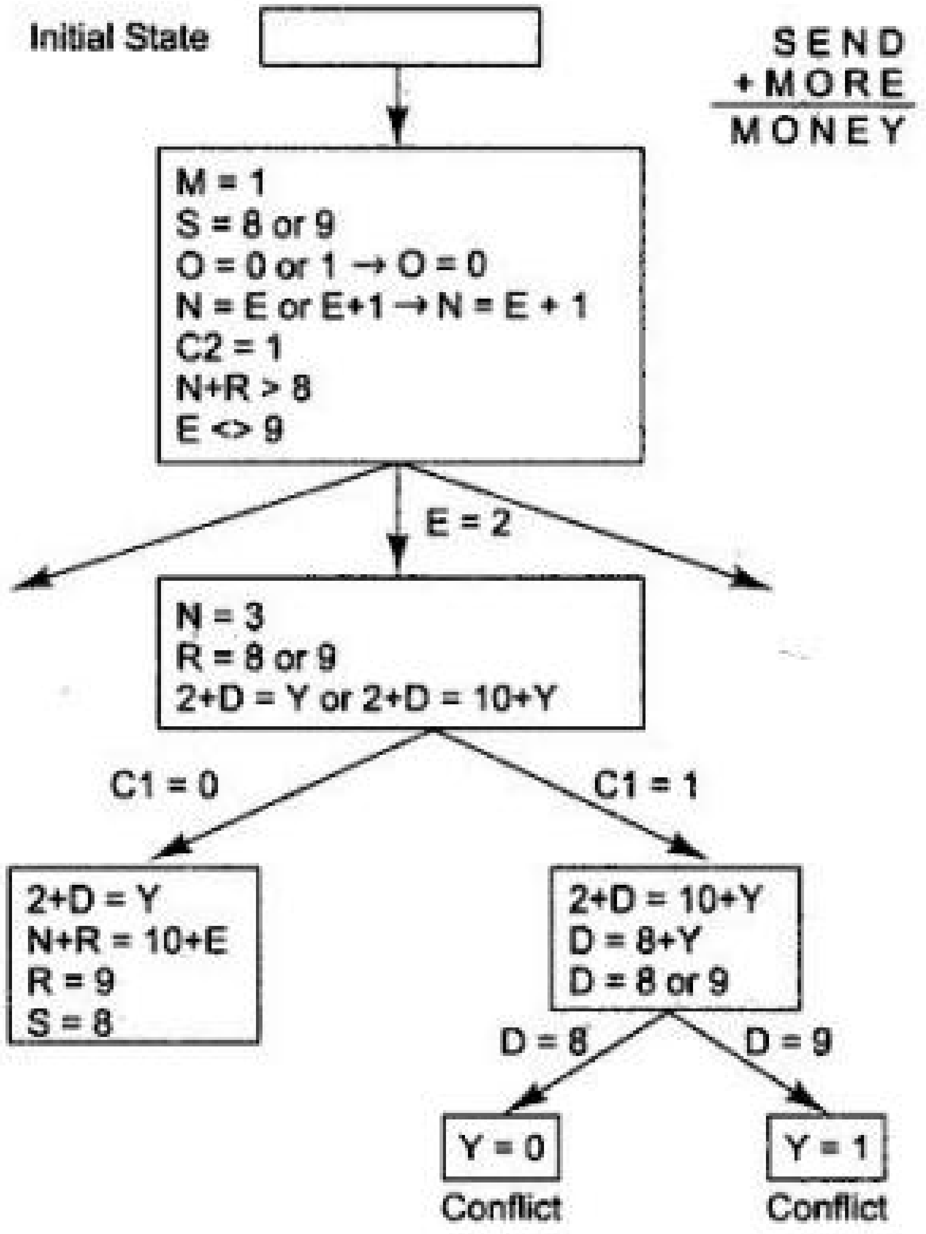


Fig. 3.14 *Solving a Cryptarithmetic Problem*

Constraint Propagation in CSPs

Instead of using **brute-force search**, we use **constraint propagation** to eliminate impossible values early.

Techniques for Solving CSPs

Backtracking Search

- Assign values **one by one**.
- If a conflict is found, **"backtrack"** and try another value.
- **Slow** if constraints are not applied early.

Constraint Propagation

- Constraints are **applied early** to prune impossible values.
- Uses **Arc Consistency (AC-3)** to systematically eliminate invalid choices.

Figure 3.14: Illustrates how constraint propagation works in the **cryptarithmetic puzzle**.

- As we assign values, **new constraints are generated** to reduce possibilities.
- If an assignment leads to an invalid state, **backtracking** is used.

Algorithm: Constraint Satisfaction

Steps to Solve a CSP

1. **Start with an initial state** where all possible values are assigned.
2. **Apply constraints** to remove impossible values.
3. **Use backtracking if necessary.**
4. **Repeat until a valid solution is found.**

Applying to Cryptarithmetic (SEND + MORE = MONEY)

1.

Assign initial constraints:

- No two letters can have the same value.
- The sum of digits must match the given total.

Constraint Propagation Example:

- If **S + M ≥ 10**, a carry-over occurs (C1 = 1).
- This restricts possible values for **S** and **M**.

Narrow Down Choices:

- If **S = 9**, then no other letter can be **9**.

Continue Eliminating Values:

- Use constraint propagation to refine the solution.

Figure 3.14 (continued) shows the **step-by-step elimination of values** based on constraints.

- This helps find the correct values **without brute-force guessing**.

The Concept of Means-Ends Analysis

- MEA focuses on analyzing **differences** between the **current state** and the **goal state**.
- It selects **operators** that help **reduce these differences**, gradually reaching the solution.

Example

Imagine solving a **puzzle** where you must **move objects** from one location to another.

- The **main challenge** is deciding which moves to **prioritize** to reach the goal efficiently.
- MEA helps in structuring the solution by selecting **appropriate operators** that progressively **eliminate differences**.

Historical Background and AI Application

- MEA was a key component of **early AI systems**, especially in the **General Problem Solver (GPS)** developed by **Newell and Simon (1961)**.
- The **GPS program** used MEA to **break down complex problems into smaller steps**, solving them incrementally.
- While effective, MEA **struggled with real-world unpredictability**, as it required well-defined **operators** and **clear goals**.

Operators and the Difference Table (Figures 3.15 & 3.16)

Operators Used in MEA

The system uses various **operators** to transform the **current state** into the **goal state**.

Figure 3.15: The Robot's Operators

- **PUSH (object, loc):** Moves an object.
- **CARRY (object, loc):** Moves an object while carrying it.
- **WALK (loc):** Moves the robot to a new location.
- **PICKUP (object):** Lifts an object.
- **PLACE (object, loc):** Places an object at a specific location.

Operator	*Preconditions*	*Results*
PUSH(obj, loc)	at(robot, obj)^ large(obj)^ clear(obj)^ armempty	at(obj, loc)^ at(robot, loc)
CARRY(obj, loc)	at(robot, obj)^ small(obj)	at(obj, loc)^ at(robot, loc)
WALK(loc)	none	at(robot, loc)
PICKUP(obj)	at(robot, obj)	holding(obj)
PUTDOWN(obj)	holding(obj)	¬holding(obj)
PLACE(obj1, obj2)	at(robot, obj2)^ holding(obj1)	on(obj1, obj2)

Fig. 3.15 *The Robot's Operators*

	Push	Carry	Walk	Pickup	Putdown	Place
Move object	*	*				
Move robot			*			
Clear object				*		
Get object on object						*
Get arm empty					*	*
Be holding object				*		

Fig. 3.16 *A Difference Table*

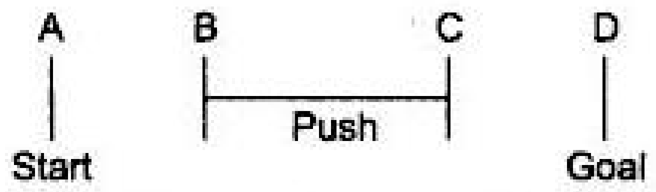


Fig. 3.17 *The Progress of the Means-Ends Analysis Method*

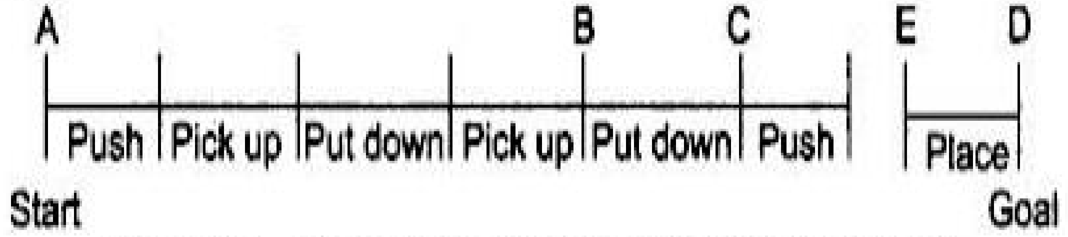


Fig. 3.18 *More Progress of the Means-Ends Method*

These operators help the **robot navigate** and **manipulate objects** in a structured way.

Difference Table for Problem Solving

Figure 3.16: A Difference Table
The **Difference Table** shows **which operators** are applicable based on **differences** between the **current state** and the **goal state**.

- **Example Entries:**
  - **"Move object"** → Can be done using **PUSH or CARRY**.
  - **"Get object"** → Requires **PICKUP**.
  - **"Get to object"** → Needs **WALK**.

This table helps the **robot decide which action to take next**.

Example Walkthrough of Means-Ends Analysis (Figures 3.17 & 3.18)

Now, let's apply **MEA to a real-world scenario**.

Step-by-Step Execution

Figure 3.17: The Problem Setup

- The **robot needs to move an object** to a goal location.
- The **Difference Table (Fig. 3.16)** helps select **appropriate operators**.
- The **robot checks available actions** and **chooses the best one**.

Dead-End Situations and Backtracking

In some cases, an action **leads to a dead-end** (e.g., **pushing an object without being close enough**).

•The system must then **backtrack** and **choose another operator**.

Figure 3.18: More Progress with Means-Ends Analysis

- Shows how **applying different operators** helps the robot gradually **achieve the goal**.
- **Example:**

WALK to the object.

1. **PICKUP** the object.
2. **CARRY** it to the destination.

4. **PLACE** it at the goal location.

Each **operator execution reduces the difference** between the current state and the goal state.

5. Algorithm: Means-Ends Analysis

The **MEA Algorithm** works as follows:

Compare the **current state** to the **goal state**.

If **no differences exist**, the problem is solved!

oIf differences **remain**, select an operator to **reduce the difference**.

Choose an Operator

o Pick an operator that **best eliminates the biggest difference**.

Apply the Operator and Update the State

o Modify the **current state** based on the chosen action.

Continue Until the Goal is Reached

Repeat the process until there are **no differences** left.

1. Explain different characteristics of the AI problem used to analyze the most appropriate method.
2. A water jug problem states "you are provided with two jugs, first one with 4-gallon capacity and the second one with 3-gallon capacity. Neither have any measuring markers on it." How can you get exactly 2 gallons of water into 4-gallon jug?
    a. Write down the production rules for the above problem.
    b. Write any one solution to the above problem.
3. Explain how AND-OR graphs are used in problem reduction.
4. Define artificial intelligence and list the task domains of artificial intelligence.
5. Explain production system and its four types.
6. List and explain the Constraint Satisfaction.
7. Explain constraint satisfaction and solve the cryptarithmetic problem: CROSS + ROADS = DANGER.
8. Discuss the production rules for solving the water-jug problem.
9. Solve the following cryptarithmetic problem DONALD + GERALD = ROBERT.
10. Develop AO* algorithm for AI applications.
11. Write an algorithm for simple Hill Climbing.
12. Describe Briefly the various Key Dimensions of Problem Analysis.
13. Describe the process of simulated annealing with an example.
14. Discuss A* and the various observations about algorithm briefly.
15. Explain in detail about the means–end analysis procedure with example.
16. What is an Artificial Intelligence technique? Explain.
17. Write a note on Production System.
18. Crypt arithmetic problem: SEND + MORE = MONEY. Initial state: No two letters have same value. Sum of digits must be shown.
19. List and explain perspectives and issues in perspective learning.