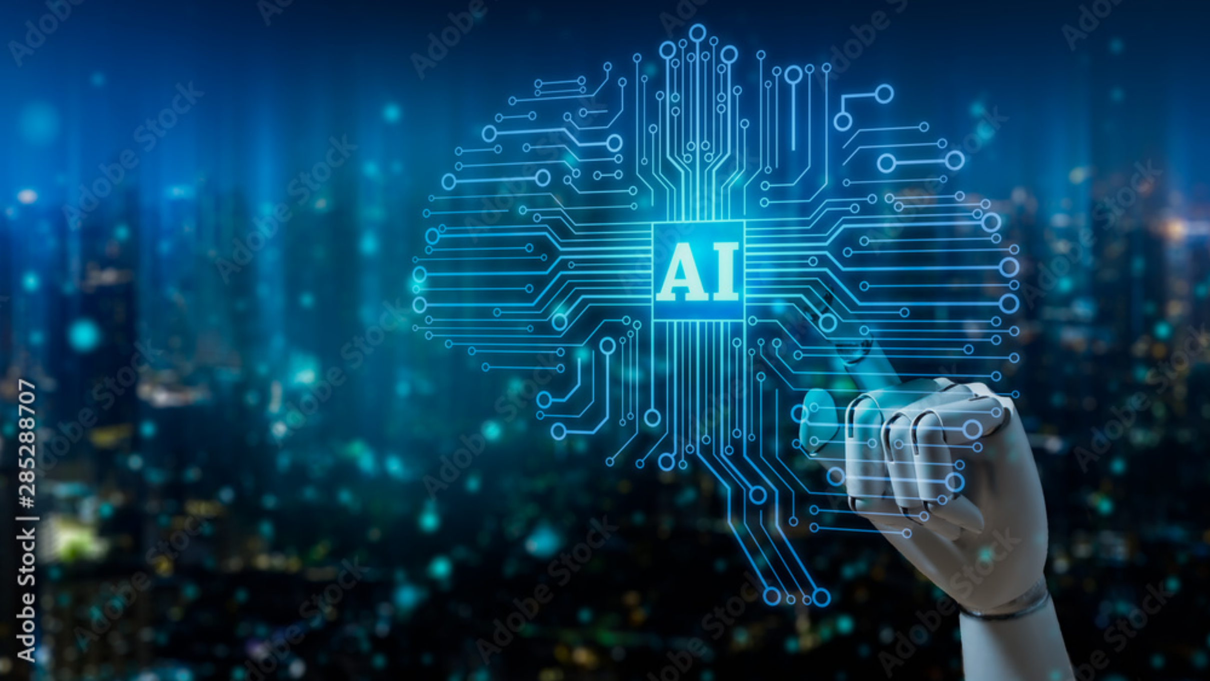

AI

Department of M.A.C.S
NIT-K

ARTIFICIAL INTELLIGENCE

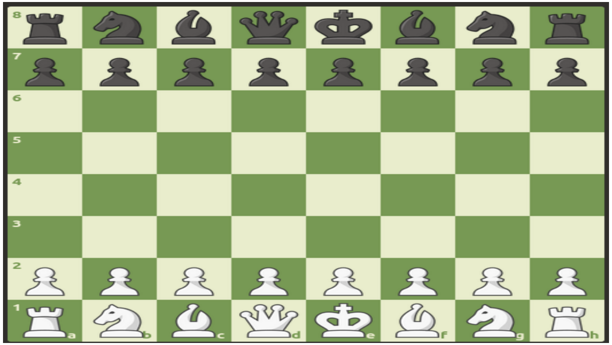
Lecture 3 by Nandagopal S A
Assistant Lecturer
Department of M.A.C.S
NIT-K

August 9, 2024

ARTIFICIAL INTELLIGENCE

INTRODUCTION

In order to play chess,
Think about an AI program!



CHESS PROGRAM STRUCTURE

An easy and a basic way is to

- Mark and label the board
- Mark and label the pieces
- Navigate the program to move according to labels and rules of chess
- And consider each different position as states [Each state is a legal position on board]

CHESS PROGRAM STRUCTURE

An easy and a basic way is to

- Mark and label the board
- Mark and label the pieces
- Navigate the program to move according to labels and rules of chess
- And consider each different position as states [Each state is a legal position on board]

CHESS PROGRAM STRUCTURE

An easy and a basic way is to

- Mark and label the board
- Mark and label the pieces
- Navigate the program to move according to labels and rules of chess
- And consider each different position as states [Each state is a legal position on board]

CHESS PROGRAM STRUCTURE

AI program to play chess

An easy and a basic way is to

- Mark and label the board
- Mark and label the pieces
- Navigate the program to move according to labels and rules of chess
- And consider each different position as states [Each state is a legal position on board]

A PROBLEM

A water jug problem

You are given 2 jugs a 3G and a 4G one, neither has any measurings on it. There is a pump that can fill jugs. How can you get 2G of water in 4G jug.

How to solve this? Think about an AI program to solve this!

A PROBLEM

A water jug problem

You are given 2 jugs a 3G and a 4G one, neither has any measurings on it. There is a pump that can fill jugs. How can you get 2G of water in 4G jug.

How to solve this? Think about an AI program to solve this!

A PROBLEM

A water jug problem

You are given 2 jugs a 3G and a 4G one, neither has any measurings on it. There is a pump that can fill jugs. How can you get 2G of water in 4G jug.

How to solve this? Think about an AI program to solve this!

HOW TO SOLVE?

An easy and a basic way is to

- Form a state space and search through to find final state.
- The state space for this problem can be described as the set of ordered pairs of integers (x, y), such that x = 0. 1, 2. 3, or 4 and y = 0, 1, 2, or 3; x represents the number of gallons of water in the 4-gallon jug, and y represents the quantity of water in the 3-gallon jug.
- The start state is (0, 0). The goal state is (2, n) for any value of n (since the problem does not specify how many gallons need to be in the 3-gallon jug).

HOW TO SOLVE?

An easy and a basic way is to

- Form a state space and search through to find final state.
- The state space for this problem can be described as the set of ordered pairs of integers (x, y), such that x = 0. 1, 2. 3, or 4 and y = 0, 1, 2, or 3; x represents the number of gallons of water in the 4-gallon jug, and y represents the quantity of water in the 3-gallon jug.
- The start state is (0, 0). The goal state is (2, n) for any value of n (since the problem does not specify how many gallons need to be in the 3-gallon jug).

HOW TO SOLVE?

An easy and a basic way is to

- Form a state space and search through to find final state.
- The state space for this problem can be described as the set of ordered pairs of integers (x, y), such that x = 0. 1, 2. 3, or 4 and y = 0, 1, 2, or 3; x represents the number of gallons of water in the 4-gallon jug, and y represents the quantity of water in the 3-gallon jug.
- The start state is (0, 0). The goal state is (2, n) for any value of n (since the problem does not specify how many gallons need to be in the 3-gallon jug).

ADDITIONAL STRUCTURE

- To solve the water jug problem, all we need, in addition to the problem description given above, is a control structure that loops through a simple cycle in which some rule whose left side matches the current state is chosen, the appropriate change to the state is made as described in the corresponding right side, and the resulting state is checked to see if it corresponds to a goal state. As long as it does not, the cycle continues.
- Clearly the speed with which the problem gets solved depends on the mechanism that is used to select the next operation to be performed.

Note

The problem can then be solved by using the rules, in combination with an appropriate control strategy, to move through the problem space until a path from an initial state to a goal state is found. Thus the process of search is fundamental to the problem-solving process.

ADDITIONAL STRUCTURE

- To solve the water jug problem, all we need, in addition to the problem description given above, is a control structure that loops through a simple cycle in which some rule whose left side matches the current state is chosen, the appropriate change to the state is made as described in the corresponding right side, and the resulting state is checked to see if it corresponds to a goal state. As long as it does not, the cycle continues.
- Clearly the speed with which the problem gets solved depends on the mechanism that is used to select the next operation to be performed.

Note

The problem can then be solved by using the rules, in combination with an appropriate control strategy, to move through the problem space until a path from an initial state to a goal state is found. Thus the process of search is fundamental to the problem-solving process.

ADDITIONAL STRUCTURE

- To solve the water jug problem, all we need, in addition to the problem description given above, is a control structure that loops through a simple cycle in which some rule whose left side matches the current state is chosen, the appropriate change to the state is made as described in the corresponding right side, and the resulting state is checked to see if it corresponds to a goal state. As long as it does not, the cycle continues.
- Clearly the speed with which the problem gets solved depends on the mechanism that is used to select the next operation to be performed.

Note

The problem can then be solved by using the rules, in combination with an appropriate control strategy, to move through the problem space until a path from an initial state to a goal state is found. Thus the process of search is fundamental to the problem-solving process.

ADDITIONAL STRUCTURE

- To solve the water jug problem, all we need, in addition to the problem description given above, is a control structure that loops through a simple cycle in which some rule whose left side matches the current state is chosen, the appropriate change to the state is made as described in the corresponding right side, and the resulting state is checked to see if it corresponds to a goal state. As long as it does not, the cycle continues.
- Clearly the speed with which the problem gets solved depends on the mechanism that is used to select the next operation to be performed.

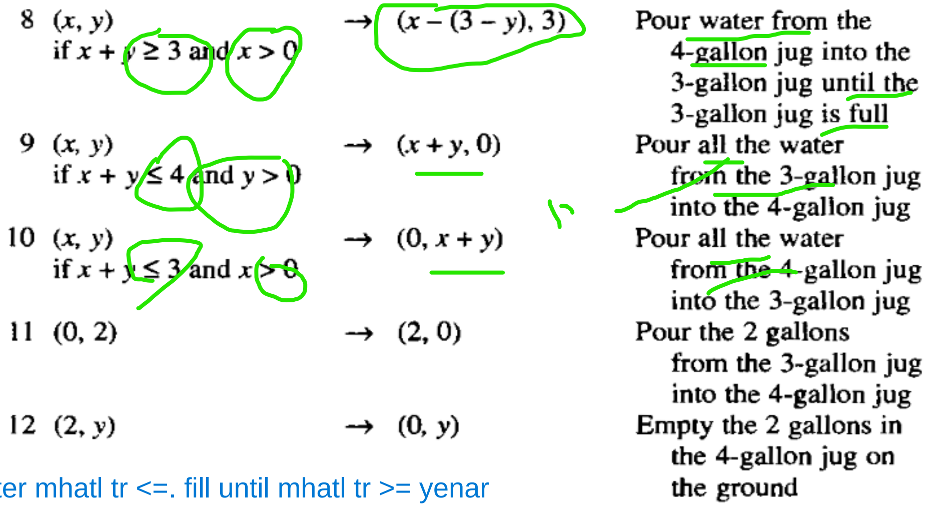
Note

The problem can then be solved by using the rules, in combination with an appropriate control strategy, to move through the problem space until a path from an initial state to a goal state is found. Thus the process of search is fundamental to the problem-solving process.

1	(x, y) if $x < 4$	$\rightarrow$	$(4, y)$	Fill the 4-gallon jug
2	(x, y) if $y < 3$	$\rightarrow$	$(x, 3)$	Fill the 3-gallon jug
3	(x, y) if $x > 0$	$\rightarrow$	$(x - d, y)$	Pour some water out of the 4-gallon jug
4	(x, y) if $y > 0$	$\rightarrow$	$(x, y - d)$	Pour some water out of the 3-gallon jug
5	$(x, y\}$ if $x > 0$	$\rightarrow$	$(0, y)$	Empty the 4-gallon jug on the ground
6	(x, y) if $y > 0$	$\rightarrow$	$(x, 0)$	Empty the 3-gallon jug on the ground
7	(x, y) if $x + y \geq 4$ and $y > 0$	$\rightarrow$	$(4, y - (4 - x))$	Pour water from the 3-gallon jug into the 4-gallon jug until the 4-gallon jug is full

T F

FROM y mhatl tr y>0 (y not empty) same for x. jachat TO mhatl x+y pude te yenar digit

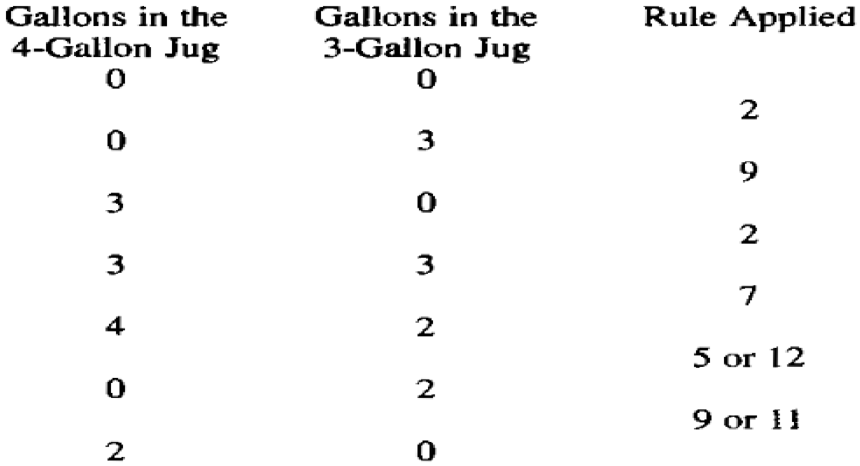


8	(x, y) if $x + y \geq 3$ and $x > 0$	$\rightarrow$	$(x - (3 - y), 3)$	Pour water from the 4-gallon jug into the 3-gallon jug until the 3-gallon jug is full
9	(x, y) if $x + y \leq 4$ and $y > 0$	$\rightarrow$	$(x + y, 0)$	Pour all the water from the 3-gallon jug into the 4-gallon jug
10	(x, y) if $x + y \leq 3$ and $x > 0$	$\rightarrow$	$(0, x + y)$	Pour all the water from the 4-gallon jug into the 3-gallon jug
11	$(0, 2)$	$\rightarrow$	$(2, 0)$	Pour the 2 gallons from the 3-gallon jug into the 4-gallon jug
12	$(2, y)$	$\rightarrow$	$(0, y)$	Empty the 2 gallons in the 4-gallon jug on the ground

all water mhatl tr <=. fill until mhatl tr >= yenar

Fig. 2.3 *Production Rules for the Water Jug Problem*

apla hat



Gallons in the 4-Gallon Jug	Gallons in the 3-Gallon Jug	Rule Applied
0	0	
		2
0	3	
		9
3	0	
		2
3	3	
		7
4	2	
		5 or 12
0	2	
		9 or 11
2	0	

Fig. 2.4 *One Solution to the Water Jug Problem*

OBSERVATIONS

- Compare tic -tac-toe problem with this one.
- Observe each of these rules are maximally specific, in t-t-t problem, i.e it applies to a single board configuration and as a result no search is required when such rules are used.
- From the discussion of these 2 problems what is the first step toward design of a problem?
- The creation of a formal and manipulable description of the problem itself.
- Creating program that can themselves produce formal descriptions from informal ones is known as ***operationalization***.

OBSERVATIONS

- Compare tic -tac-toe problem with this one.
- Observe each of these rules are maximally specific, in t-t-t problem, i.e it applies to a single board configuration and as a result no search is required when such rules are used.
- From the discussion of these 2 problems what is the first step toward design of a problem?
- The creation of a formal and manipulable description of the problem itself.
- Creating program that can themselves produce formal descriptions from informal ones is known as ***operationalization***.

OBSERVATIONS

- Compare tic -tac-toe problem with this one.
- Observe each of these rules are maximally specific, in t-t-t problem, i.e it applies to a single board configuration and as a result no search is required when such rules are used.
- From the discussion of these 2 problems what is the first step toward design of a problem?
- The creation of a formal and manipulable description of the problem itself.
- Creating program that can themselves produce formal descriptions from informal ones is known as ***operationalization***.

OBSERVATIONS

- Compare tic -tac-toe problem with this one.
- Observe each of these rules are maximally specific, in t-t-t problem, i.e it applies to a single board configuration and as a result no search is required when such rules are used.
- From the discussion of these 2 problems what is the first step toward design of a problem?
- The creation of a formal and manipulable description of the problem itself.
- Creating program that can themselves produce formal descriptions from informal ones is known as ***operationalization***.

OBSERVATIONS

- Compare tic -tac-toe problem with this one.
- Observe each of these rules are maximally specific, in t-t-t problem, i.e it applies to a single board configuration and as a result no search is required when such rules are used.
- From the discussion of these 2 problems what is the first step toward design of a problem?
- The creation of a formal and manipulable description of the problem itself.
- Creating program that can themselves produce formal descriptions from informal ones is known as ***operationalization***.

PRODUCTION SYSTEMS

Since search forms the core of many intelligent processes, it is useful to structure AI programs in a way that facilitates describing and performing the search process. Production systems provide such structures.

PRODUCTION SYSTEMS

Production systems, also known as production rule systems or production rule-based systems, are a fundamental concept in artificial intelligence (AI) and expert systems. They are used to represent knowledge in the form of rules and make decisions based on these rules. Here's an overview of production systems in AI:

- Rule-based Knowledge Representation: In production systems, knowledge is typically represented in the form of "if-then" rules. Each rule consists of an antecedent (condition) and a consequent (action).
- Working Memory: The system maintains a working memory, which is a set of facts or information about the current state of the world. These facts are used to evaluate the conditions of rules.
- Inference Engine: The inference engine is responsible for applying the rules to the working memory. It matches the antecedent of rules with the facts in the working memory to determine which rules are applicable.

PRODUCTION SYSTEMS

Production systems, also known as production rule systems or production rule-based systems, are a fundamental concept in artificial intelligence (AI) and expert systems. They are used to represent knowledge in the form of rules and make decisions based on these rules. Here's an overview of production systems in AI:

- Rule-based Knowledge Representation: In production systems, knowledge is typically represented in the form of "if-then" rules. Each rule consists of an antecedent (condition) and a consequent (action).
- Working Memory: The system maintains a working memory, which is a set of facts or information about the current state of the world. These facts are used to evaluate the conditions of rules.
- Inference Engine: The inference engine is responsible for applying the rules to the working memory. It matches the antecedent of rules with the facts in the working memory to determine which rules are applicable.

PRODUCTION SYSTEMS

Production systems, also known as production rule systems or production rule-based systems, are a fundamental concept in artificial intelligence (AI) and expert systems. They are used to represent knowledge in the form of rules and make decisions based on these rules. Here's an overview of production systems in AI:

- Rule-based Knowledge Representation: In production systems, knowledge is typically represented in the form of "if-then" rules. Each rule consists of an antecedent (condition) and a consequent (action).
- Working Memory: The system maintains a working memory, which is a set of facts or information about the current state of the world. These facts are used to evaluate the conditions of rules.
- Inference Engine: The inference engine is responsible for applying the rules to the working memory. It matches the antecedent of rules with the facts in the working memory to determine which rules are applicable.

PRODUCTION SYSTEMS

Production systems, also known as production rule systems or production rule-based systems, are a fundamental concept in artificial intelligence (AI) and expert systems. They are used to represent knowledge in the form of rules and make decisions based on these rules. Here's an overview of production systems in AI:

- Rule-based Knowledge Representation: In production systems, knowledge is typically represented in the form of "if-then" rules. Each rule consists of an antecedent (condition) and a consequent (action).
- Working Memory: The system maintains a working memory, which is a set of facts or information about the current state of the world. These facts are used to evaluate the conditions of rules.
- Inference Engine: The inference engine is responsible for applying the rules to the working memory. It matches the antecedent of rules with the facts in the working memory to determine which rules are applicable.

PRODUCTION SYSTEMS

Production systems, also known as production rule systems or production rule-based systems, are a fundamental concept in artificial intelligence (AI) and expert systems. They are used to represent knowledge in the form of rules and make decisions based on these rules. Here's an overview of production systems in AI:

- Rule-based Knowledge Representation: In production systems, knowledge is typically represented in the form of "if-then" rules. Each rule consists of an antecedent (condition) and a consequent (action).
- Working Memory: The system maintains a working memory, which is a set of facts or information about the current state of the world. These facts are used to evaluate the conditions of rules.
- Inference Engine: The inference engine is responsible for applying the rules to the working memory. It matches the antecedent of rules with the facts in the working memory to determine which rules are applicable.

- Conflict Resolution: If multiple rules are applicable at a given time, a conflict resolution strategy is used to determine which rule to execute. Common strategies include prioritizing rules based on specificity or using a predefined order.
- Execution Cycle: The execution cycle of a production system typically involves the following steps:
  1. Match: The engine matches rules whose conditions (antecedents) are satisfied by the current state of the working memory.
  2. Select: If multiple rules are matched, a conflict resolution strategy selects one rule for execution.
  3. Execute: The consequent of the selected rule is executed, which may involve modifying the working memory.
  4. Repeat: The cycle continues until no further rules are applicable.

- Conflict Resolution: If multiple rules are applicable at a given time, a conflict resolution strategy is used to determine which rule to execute. Common strategies include prioritizing rules based on specificity or using a predefined order.
- Execution Cycle: The execution cycle of a production system typically involves the following steps:
    1. Match: The engine matches rules whose conditions (antecedents) are satisfied by the current state of the working memory.
    2. Select: If multiple rules are matched, a conflict resolution strategy selects one rule for execution.
    3. Execute: The consequent of the selected rule is executed, which may involve modifying the working memory.
    4. Repeat: The cycle continues until no further rules are applicable.

- Conflict Resolution: If multiple rules are applicable at a given time, a conflict resolution strategy is used to determine which rule to execute. Common strategies include prioritizing rules based on specificity or using a predefined order.
- Execution Cycle: The execution cycle of a production system typically involves the following steps:
    1. Match: The engine matches rules whose conditions (antecedents) are satisfied by the current state of the working memory.
    2. Select: If multiple rules are matched, a conflict resolution strategy selects one rule for execution.
    3. Execute: The consequent of the selected rule is executed, which may involve modifying the working memory.
    4. Repeat: The cycle continues until no further rules are applicable.

- Conflict Resolution: If multiple rules are applicable at a given time, a conflict resolution strategy is used to determine which rule to execute. Common strategies include prioritizing rules based on specificity or using a predefined order.
- Execution Cycle: The execution cycle of a production system typically involves the following steps:
  1. Match: The engine matches rules whose conditions (antecedents) are satisfied by the current state of the working memory.
  2. Select: If multiple rules are matched, a conflict resolution strategy selects one rule for execution.
  3. Execute: The consequent of the selected rule is executed, which may involve modifying the working memory.
  4. Repeat: The cycle continues until no further rules are applicable.

- Conflict Resolution: If multiple rules are applicable at a given time, a conflict resolution strategy is used to determine which rule to execute. Common strategies include prioritizing rules based on specificity or using a predefined order.
- Execution Cycle: The execution cycle of a production system typically involves the following steps:
  1. Match: The engine matches rules whose conditions (antecedents) are satisfied by the current state of the working memory.
  2. Select: If multiple rules are matched, a conflict resolution strategy selects one rule for execution.
  3. Execute: The consequent of the selected rule is executed, which may involve modifying the working memory.
  4. Repeat: The cycle continues until no further rules are applicable.

- Conflict Resolution: If multiple rules are applicable at a given time, a conflict resolution strategy is used to determine which rule to execute. Common strategies include prioritizing rules based on specificity or using a predefined order.
- Execution Cycle: The execution cycle of a production system typically involves the following steps:
    1. Match: The engine matches rules whose conditions (antecedents) are satisfied by the current state of the working memory.
    2. Select: If multiple rules are matched, a conflict resolution strategy selects one rule for execution.
    3. Execute: The consequent of the selected rule is executed, which may involve modifying the working memory.
    4. Repeat: The cycle continues until no further rules are applicable.

- Forward Chaining vs. Backward Chaining: Forward Chaining: Starts with the known facts and applies rules to derive conclusions. It continues until a goal is reached. Backward Chaining: Starts with a goal and works backward, trying to find rules whose consequents match the goal.
- Applications: Production systems have been used in various domains, including expert systems, medical diagnosis, process control, natural language processing, and more.
- Advantages: They provide a transparent and interpretable way to represent and use knowledge. They are suitable for domains with well-defined rules and expert knowledge.
- Disadvantages: Handling uncertainty and complex reasoning can be challenging for simple production systems. They may not be the most efficient approach for certain types of problems.

- Forward Chaining vs. Backward Chaining: Forward Chaining: Starts with the known facts and applies rules to derive conclusions. It continues until a goal is reached. Backward Chaining: Starts with a goal and works backward, trying to find rules whose consequents match the goal.
- Applications: Production systems have been used in various domains, including expert systems, medical diagnosis, process control, natural language processing, and more.
- Advantages: They provide a transparent and interpretable way to represent and use knowledge. They are suitable for domains with well-defined rules and expert knowledge.
- Disadvantages: Handling uncertainty and complex reasoning can be challenging for simple production systems. They may not be the most efficient approach for certain types of problems.

- Forward Chaining vs. Backward Chaining: Forward Chaining: Starts with the known facts and applies rules to derive conclusions. It continues until a goal is reached. Backward Chaining: Starts with a goal and works backward, trying to find rules whose consequents match the goal.
- Applications: Production systems have been used in various domains, including expert systems, medical diagnosis, process control, natural language processing, and more.
- Advantages: They provide a transparent and interpretable way to represent and use knowledge. They are suitable for domains with well-defined rules and expert knowledge.
- Disadvantages: Handling uncertainty and complex reasoning can be challenging for simple production systems. They may not be the most efficient approach for certain types of problems.

- Forward Chaining vs. Backward Chaining: Forward Chaining: Starts with the known facts and applies rules to derive conclusions. It continues until a goal is reached. Backward Chaining: Starts with a goal and works backward, trying to find rules whose consequents match the goal.
- Applications: Production systems have been used in various domains, including expert systems, medical diagnosis, process control, natural language processing, and more.
- Advantages: They provide a transparent and interpretable way to represent and use knowledge. They are suitable for domains with well-defined rules and expert knowledge.
- Disadvantages: Handling uncertainty and complex reasoning can be challenging for simple production systems. They may not be the most efficient approach for certain types of problems.

The process of solving the problem can usefully be modeled as a production system.
Next we will look at the problem of choosing the appropriate control structure for production system so that search is efficient.

BFS

Breadth-First Search (BFS) is an algorithm used for traversing or searching tree or graph data structures. It starts at the tree root (or some arbitrary node in the case of a graph) and explores the neighbor nodes at the present depth prior to moving on to nodes at the next depth level.

Here's a basic outline of how BFS works:

1. Start at a Node: Begin at the starting node.
2. Explore Neighbors: Visit all of the neighbors of the current node. These are the nodes that are directly connected to the current node.
3. Queue Up Unvisited Neighbors: Add the unvisited neighbors to a queue to be processed in the next steps.
4. Mark as Visited: Mark the current node as visited to avoid revisiting it.
5. Move to Next Node: Take the next node from the queue and repeat steps 2-4.
6. Repeat: Continue this process until all nodes have been visited or until you've found your target node (if you're searching for a specific node).

1. Start at a Node: Begin at the starting node.
2. Explore Neighbors: Visit all of the neighbors of the current node. These are the nodes that are directly connected to the current node.
3. Queue Up Unvisited Neighbors: Add the unvisited neighbors to a queue to be processed in the next steps.
4. Mark as Visited: Mark the current node as visited to avoid revisiting it.
5. Move to Next Node: Take the next node from the queue and repeat steps 2-4.
6. Repeat: Continue this process until all nodes have been visited or until you've found your target node (if you're searching for a specific node).

1. Start at a Node: Begin at the starting node.
2. Explore Neighbors: Visit all of the neighbors of the current node. These are the nodes that are directly connected to the current node.
3. Queue Up Unvisited Neighbors: Add the unvisited neighbors to a queue to be processed in the next steps.
4. Mark as Visited: Mark the current node as visited to avoid revisiting it.
5. Move to Next Node: Take the next node from the queue and repeat steps 2-4.
6. Repeat: Continue this process until all nodes have been visited or until you've found your target node (if you're searching for a specific node).

1. Start at a Node: Begin at the starting node.
2. Explore Neighbors: Visit all of the neighbors of the current node. These are the nodes that are directly connected to the current node.
3. Queue Up Unvisited Neighbors: Add the unvisited neighbors to a queue to be processed in the next steps.
4. Mark as Visited: Mark the current node as visited to avoid revisiting it.
5. Move to Next Node: Take the next node from the queue and repeat steps 2-4.
6. Repeat: Continue this process until all nodes have been visited or until you've found your target node (if you're searching for a specific node).

1. Start at a Node: Begin at the starting node.
2. Explore Neighbors: Visit all of the neighbors of the current node. These are the nodes that are directly connected to the current node.
3. Queue Up Unvisited Neighbors: Add the unvisited neighbors to a queue to be processed in the next steps.
4. Mark as Visited: Mark the current node as visited to avoid revisiting it.
5. Move to Next Node: Take the next node from the queue and repeat steps 2-4.
6. Repeat: Continue this process until all nodes have been visited or until you've found your target node (if you're searching for a specific node).

1. Start at a Node: Begin at the starting node.
2. Explore Neighbors: Visit all of the neighbors of the current node. These are the nodes that are directly connected to the current node.
3. Queue Up Unvisited Neighbors: Add the unvisited neighbors to a queue to be processed in the next steps.
4. Mark as Visited: Mark the current node as visited to avoid revisiting it.
5. Move to Next Node: Take the next node from the queue and repeat steps 2-4.
6. Repeat: Continue this process until all nodes have been visited or until you've found your target node (if you're searching for a specific node).

BFS

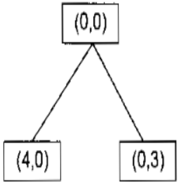


Fig. 2.5 *One Level of a Breadth-First Search Tree*

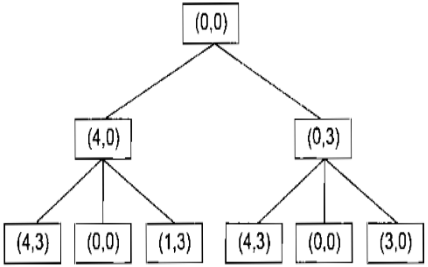


Fig. 2.6 *Two Levels of a Breadth-First Search Tree*

Rule 3, 4, 11, and 12 have been ignored in constructing the search tree.

DFS

Depth-First Search (DFS) is another fundamental graph traversal algorithm. Like BFS, it's used to visit and explore all the nodes in a graph. However, the approach is different.
Here's how DFS works:

1. Start at a Node: Begin at the starting node.
2. Explore as Far as Possible: Move as far as possible along each branch before backtracking. This means visiting a neighbor of the current node and repeating this step recursively.
3. Backtrack: When you reach a node with no unvisited neighbors, backtrack to the previous node and continue the process.
4. Repeat: Continue this process until all nodes have been visited.

DFS

Depth-First Search (DFS) is another fundamental graph traversal algorithm. Like BFS, it's used to visit and explore all the nodes in a graph. However, the approach is different.

Here's how DFS works:

1. Start at a Node: Begin at the starting node.
2. Explore as Far as Possible: Move as far as possible along each branch before backtracking. This means visiting a neighbor of the current node and repeating this step recursively.
3. Backtrack: When you reach a node with no unvisited neighbors, backtrack to the previous node and continue the process.
4. Repeat: Continue this process until all nodes have been visited.

DFS

Depth-First Search (DFS) is another fundamental graph traversal algorithm. Like BFS, it's used to visit and explore all the nodes in a graph. However, the approach is different.
Here's how DFS works:

1. Start at a Node: Begin at the starting node.
2. Explore as Far as Possible: Move as far as possible along each branch before backtracking. This means visiting a neighbor of the current node and repeating this step recursively.
3. Backtrack: When you reach a node with no unvisited neighbors, backtrack to the previous node and continue the process.
4. Repeat: Continue this process until all nodes have been visited.

DFS

Depth-First Search (DFS) is another fundamental graph traversal algorithm. Like BFS, it's used to visit and explore all the nodes in a graph. However, the approach is different.
Here's how DFS works:

1. Start at a Node: Begin at the starting node.
2. Explore as Far as Possible: Move as far as possible along each branch before backtracking. This means visiting a neighbor of the current node and repeating this step recursively.
3. Backtrack: When you reach a node with no unvisited neighbors, backtrack to the previous node and continue the process.
4. Repeat: Continue this process until all nodes have been visited.

DFS

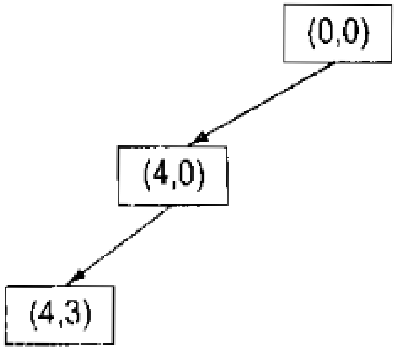


Fig. 2.7 *A Depth-First Search Tree*

APPLICATIONS

- BFS is used in various applications, including shortest path finding in unweighted graphs, level-order traversal in trees, finding connected components, and solving puzzles with a state space.
- DFS is used in a variety of applications, including path-finding algorithms, topological sorting, cycle detection, and solving puzzles with a state space (like Sudoku and maze solving).

APPLICATIONS

- BFS is used in various applications, including shortest path finding in unweighted graphs, level-order traversal in trees, finding connected components, and solving puzzles with a state space.
- DFS is used in a variety of applications, including path-finding algorithms, topological sorting, cycle detection, and solving puzzles with a state space (like Sudoku and maze solving).

TSP

The Traveling Salesman Problem (TSP) is a well-known combinatorial optimization problem in computer science and mathematics. It is often used as an example of a problem that is both easy to state and very difficult to solve efficiently.

Problem statement

Given a list of cities and the distances between each pair of cities, the task is to find the shortest possible tour that visits each city exactly once and returns to the original city (the salesman's home city).

TSP

The Traveling Salesman Problem (TSP) is a well-known combinatorial optimization problem in computer science and mathematics. It is often used as an example of a problem that is both easy to state and very difficult to solve efficiently.

Problem statement

Given a list of cities and the distances between each pair of cities, the task is to find the shortest possible tour that visits each city exactly once and returns to the original city (the salesman's home city).

FORMAL DEF

- Let N be a set of n cities. Let $d(i,j)$ represent the distance between city i and city j.
- The goal is to find a permutation π of the cities such that the total distance traveled, denoted $L(\pi)$, is minimized:

$$L(\pi) = \sum_{i=1}^{n-1} d(\pi(i), \pi(i+1)) + d(\pi(n), \pi(1))$$

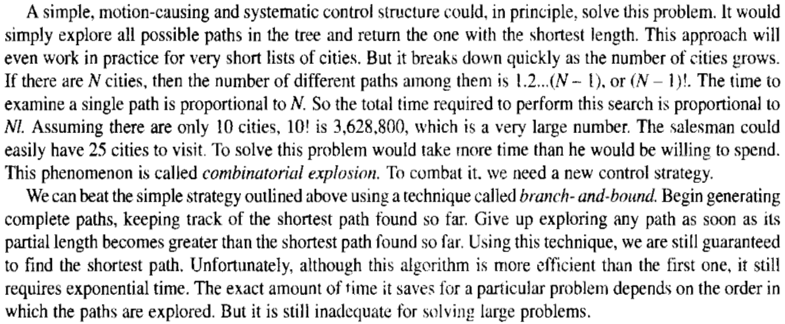
FORMAL DEF

aapla hat

- Let N be a set of n cities. Let $d(i,j)$ represent the distance between city i and city j.
- The goal is to find a permutation π of the cities such that the total distance traveled, denoted $L(\pi)$, is minimized:

$$L(\pi) = \sum_{i=1}^{n-1} d(\pi(i), \pi(i+1)) + d(\pi(n), \pi(1))$$

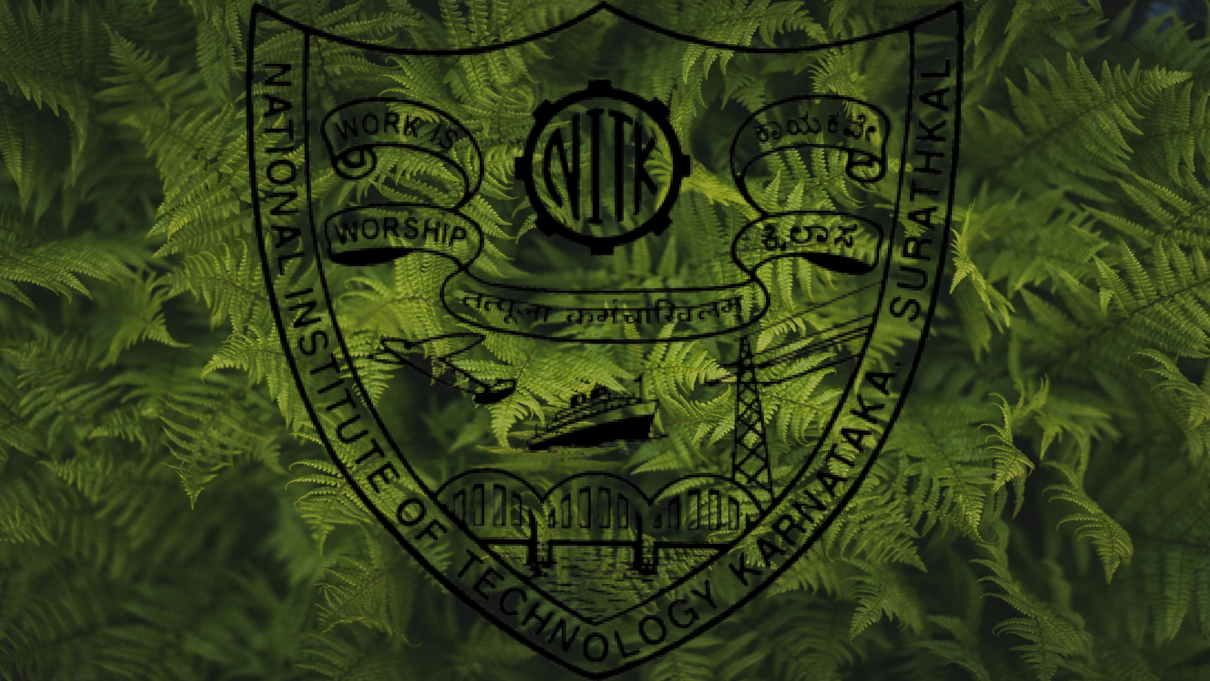
TSP



A simple, motion-causing and systematic control structure could, in principle, solve this problem. It would simply explore all possible paths in the tree and return the one with the shortest length. This approach will even work in practice for very short lists of cities. But it breaks down quickly as the number of cities grows. If there are N cities, then the number of different paths among them is $1.2...(N-1)$, or $(N-1)!$. The time to examine a single path is proportional to N. So the total time required to perform this search is proportional to $N!$. Assuming there are only 10 cities, 10! is 3,628,800, which is a very large number. The salesman could easily have 25 cities to visit. To solve this problem would take more time than he would be willing to spend. This phenomenon is called *combinatorial explosion*. To combat it, we need a new control strategy.

We can beat the simple strategy outlined above using a technique called *branch- and-bound*. Begin generating complete paths, keeping track of the shortest path found so far. Give up exploring any path as soon as its partial length becomes greater than the shortest path found so far. Using this technique, we are still guaranteed to find the shortest path. Unfortunately, although this algorithm is more efficient than the first one, it still requires exponential time. The exact amount of time it saves for a particular problem depends on the order in which the paths are explored. But it is still inadequate for solving large problems.

ARTIFICIAL INTELLIGENCE


NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA, SURATHKAL
NITK
WORK IS
WORSHIP
ಕಾಯಕವೇ
ಕೈಲಾಸ