

ECE353 Lectures

Hanhee Lee

January 10, 2025

Contents

1	Prologue	2
1.1	Components of a Robotic System	2
1.1.1	Overview (Robots, the Environment)	3
1.1.2	Robot (Sensors, Actuators, the Brain)	3
1.1.3	Brain (Tracker, Planner, Memory)	4
1.1.4	Environment (Physics, State)	4
1.2	Equations of a Robotic System	4
1.2.1	Sensing	4
1.2.2	Tracking	5
1.2.3	Planning	5
1.2.4	Acting	6
1.2.5	Simulating	6
1.3	Setup of Planning Problems	7
2	Search Problems	7
2.1	Setup	7
2.2	Search Graphs	7
2.3	Path Trees	7
2.4	Search Algorithms	8
2.4.1	Characteristics of a Search Algorithm	8
2.4.2	Breadth First Search (BFS)	8
2.4.3	Depth First Search (DFS)	9
2.4.4	Iterative Deepening DFS (IDDFS)	9
2.4.5	Cheapest-First Search (CFS)	9
2.5	Modifications to Search Algorithms	9
2.5.1	Depth-Limiting	9
2.5.2	Iterative Deepening	10
2.5.3	Cost-Limiting	10
2.5.4	Iterative-Inflating	10
2.5.5	Intra-Path Cycle Checking	11
2.5.6	Inter-Path Cycle Checking	11
2.6	Informed Search Algorithms	11
2.6.1	Estimated Cost	11
2.6.2	Heuristics	12
2.6.3	Heuristic-First Search (HFS)	12
2.6.4	A-Star Search (A*)	12
2.6.5	Iterative Inflating A-Star Search (IIA*)	12
2.6.6	Designing Heuristics via Problem Relaxation	12
2.6.7	Combining Heuristics	12
2.7	Characteristics of an Informed Search Algorithm	12
2.8	Anytime Search Algorithms	12
2.9	Formulating a Search Problem	12

1 Prologue

Summary:

- This course will focus on planning
- Variables:
 - State: $\mathbf{x}(t)$
 - Action(s): $\mathbf{u}(t)$
 - Measurement: $\mathbf{y}_k^{(i)}$
 - Context: $\mathbf{z}_k^{(i)}$
 - Old Context: $\mathbf{z}_{k-1}^{(i)}$
 - Plan: $\mathbf{p}_k^{(i)}$
 - (i): Ith agent
- Conversion to DT is necessary because robots are digitalized system and then converted back to CT for execution.

FAQ:

- What does the environment do?
- What is the joint action set?

1.1 Components of a Robotic System

Summary:

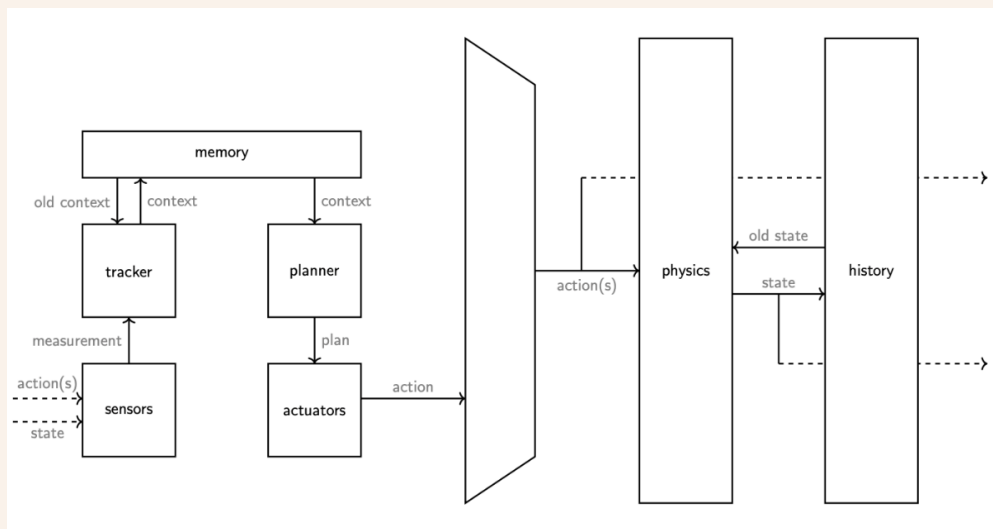


Figure 1: Components of a Robotic System (Words)

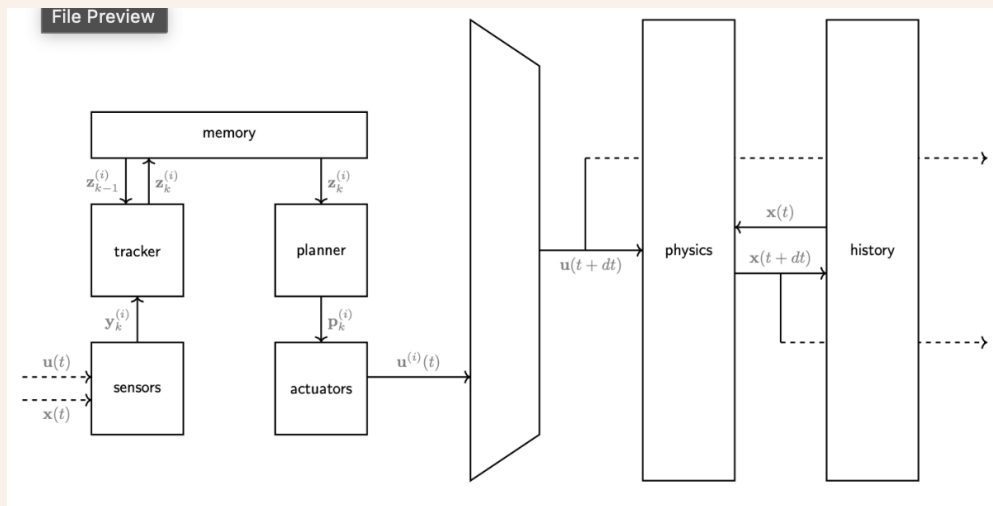


Figure 2: Components of a Robotic System (Math)

1.1.1 Overview (Robots, the Environment)

Definition:

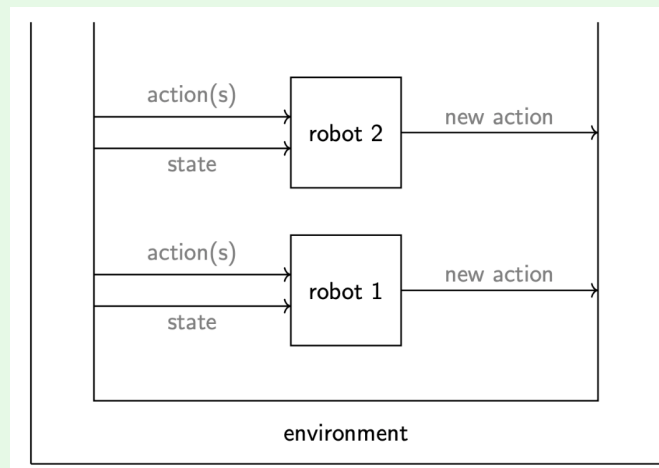


Figure 3: Overview (Robots, the Environment)

Notes:

- Environment \rightarrow previous actions + current state \rightarrow robot \rightarrow new action \rightarrow environment

1.1.2 Robot (Sensors, Actuators, the Brain)

Definition:

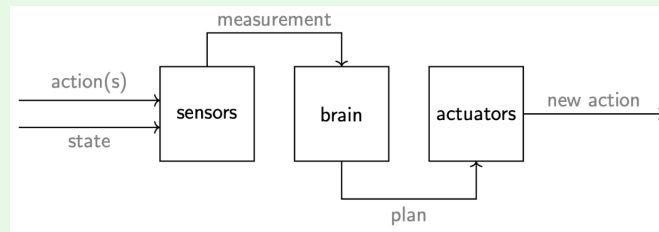


Figure 4: Robot (Sensors, Actuators, the Brain)

Notes:

- Measurements can be noisy and inaccurate if not a perfect sensor.
- Measurements go into the brain which can create a plan.

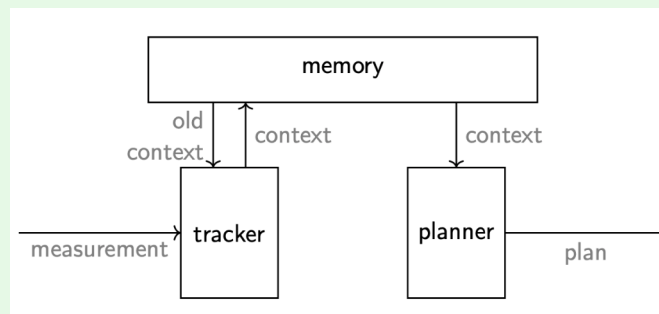
1.1.3 Brain (Tracker, Planner, Memory)**Definition:**

Figure 5: Brain (Tracker, Planner, Memory)

Notes:

- The tracker takes in the measurements and old context and updates the context.
- The planner takes in the context and creates a plan.
- The memory stores the context.

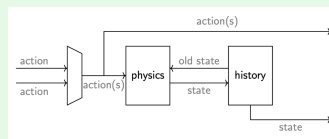
1.1.4 Environment (Physics, State)**Definition:**

Figure 6: Environment (Physics, State)

1.2 Equations of a Robotic System**1.2.1 Sensing**

Definition: Take a measurement:

$$\mathbf{y}^{(i)}(t) = \text{sns}^{(i)}(\mathbf{x}(t), \mathbf{u}(t), t)$$

Convert the measurement into a discrete-time signal using a sampling period of $T^{(i)}$:

$$\mathbf{y}_k^{(i)} = \text{dt}(\mathbf{y}^{(i)}(t), t, T^{(i)})$$

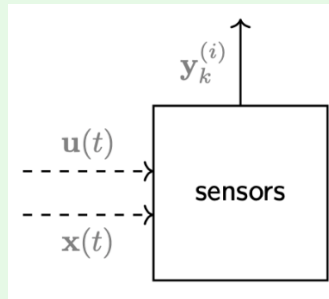


Figure 7: Sensing

1.2.2 Tracking

Definition: Track (update) the context:

$$\mathbf{z}_k^{(i)} = \text{trk}^{(i)}(\mathbf{z}_{k-1}^{(i)}, \mathbf{y}_k^{(i)}, k)$$

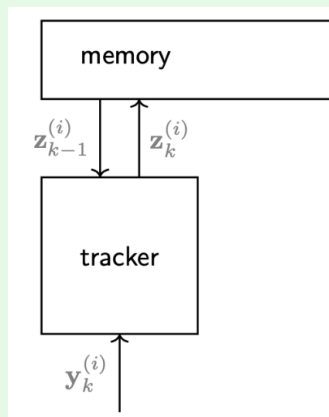


Figure 8: Tracking

1.2.3 Planning

Definition: Make a plan:

$$\mathbf{p}_k^{(i)} = \text{pln}^{(i)}(\mathbf{z}_k^{(i)}, k)$$

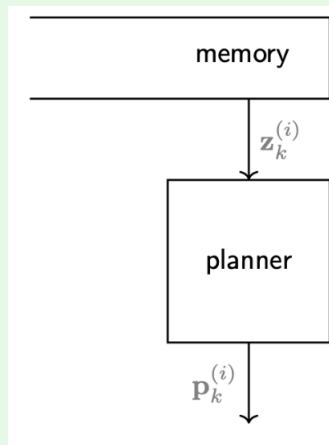


Figure 9: Planning

1.2.4 Acting

Definition: Convert the plan into a continuous-time signal using a sampling period of $T^{(i)}$:

$$\mathbf{p}(t) = \text{ct}(\mathbf{p}_k^{(i)}, t, T^{(i)})$$

Execute the plan:

$$\mathbf{u}^{(i)}(t) = \text{act}^{(i)}(\mathbf{p}^{(i)}(t), t)$$

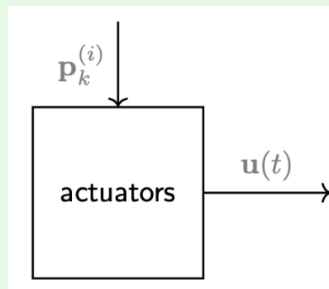


Figure 10: Acting

1.2.5 Simulating

Definition: Simulate the environment's response:

$$\dot{\mathbf{x}}(t) = \text{phy}(\mathbf{x}(t), \mathbf{u}(t), t)$$

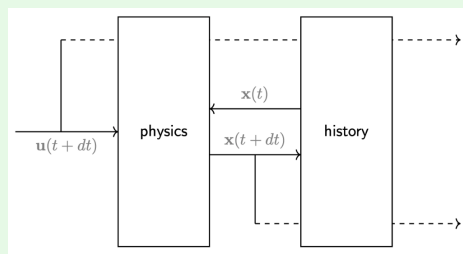


Figure 11: Simulating

1.3 Setup of Planning Problems

Summary: In a planning problem, it is assumed that:

- the environment is representable using a discrete set of states, \mathcal{S}
- for each state, $s \in \mathcal{S}$, each agent, i , has a discrete set of actions, $\mathcal{A}_i(s)$, with $\mathcal{A}(s) := \times_i \mathcal{A}_i(s)$
- a **move** is any tuple, (s, a) , where $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
- a **transition** is any 3-tuple, (s, a, s') , where $s, s' \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
- the transition resulting from a move may be deterministic/stochastic
- $rd_i(s, a, s')$ is agent i 's reward for the transition, (s, a, s')
- a **path** is any sequence of transitions of the form

$$p = \langle (s^{(0)}, a^{(1)}, s^{(1)}), (s^{(1)}, a^{(2)}, s^{(2)}), \dots \rangle$$

- each agent wants to realize a path that maximizes its own reward

Warning: $\mathcal{A}(s)$ is the joint action set of all agents at state s .

2 Search Problems

Summary:

- Not responsible for proofs, but know when to use each algorithm.

2.1 Setup

Definition: In a search problem, it is assumed that:

- There is only one agent (us).
- For each state, $s \in \mathcal{S}$, we have a discrete set of actions, $\mathcal{A}(s)$.
- The transition resulting from a move, (s, a) , is deterministic; the resulting state is $tr(s, a)$.
- $cst(s, a, tr(s, a))$ is our cost for the transition, $(s, a, tr(s, a))$.
- We want to realize a path that minimizes our cost.

A search problem may have no solutions, in which case, we define the solution as NULL.

2.2 Search Graphs

Definition: In a search graph (a graph representing a search problem):

- \mathcal{S} is defined by the vertices.
- \mathcal{G} is a subset of the vertices.
- $s^{(0)}$ is some vertex.
- $tr(\cdot, \cdot)$ and \mathcal{T} are defined by the edges.
- $cst(\cdot, \cdot, \cdot)$ is defined by the edge weights.

2.3 Path Trees

Definition: A search algorithm explores a tree of possible paths.

- In such a tree, each node represents the path from the root to itself.
 - The node may also include other info (such as the path's origin, cost, etc).

2.4 Search Algorithms

Definition: All search algorithms follow the template below:

```

1  $\mathcal{O} \leftarrow \{(\langle \rangle, 0)\}$  ▷ initialize a set of open nodes
2 SEARCH( $\mathcal{O}$ )

```

- $\langle \rangle$ is the empty path, and 0 is the cost of the empty path.

```

1 procedure SEARCH( $\mathcal{O}$ )
2   if  $\mathcal{O} = \emptyset$  then
3     return NULL ▷ the search algorithm failed to find a path to a goal
4    $n \leftarrow \text{REMOVE}(\mathcal{O})$  ▷ "explore" a node  $n$ 
5   if  $\text{DST}(n) \in \mathcal{G}$  then
6     return  $n$  ▷ the search algorithm found a path to a goal
7   for  $n' \in \text{CHL}(n)$  do
8      $\mathcal{O} \leftarrow \mathcal{O} \cup \{n'\}$  ▷ "expand"  $n$  and "export" its children
9   SEARCH( $\mathcal{O}$ )

```

- Explore: Remove a node from the open set.
- Expand: Generate the children of the node.
- Export: Add the children to the open set.

Warning: The key difference is in the order that REMOVE(\cdot) removes nodes.

2.4.1 Characteristics of a Search Algorithm

Definition: We want to choose REMOVE(\cdot) so that the algorithm exhibits the following characteristics:

Characteristic	Description
Halting	Terminates after finitely many nodes explored
Sound	Returned (possibly NULL) solution is correct
Complete	Halting and sound when a non-NULL solution exists
Optimal	Returns an optimal solution when multiple exist
Time Efficient	Minimizes the nodes explored /expanded/exported
Space Efficient	Minimizes the nodes simultaneously open

- Will be using explored for time efficiency.

The characteristics of the algorithm also depend on several properties of the path tree over which it searches. These properties include:

- Branching factor: b ($b < \infty$), the maximum number of children a node can have.
- Depth: d , the length of the longest path.
- Length of the shortest solution: l^*
- Cost of the cheapest solution: c^*
- Cost of the cheapest edge: ϵ

We want to choose REMOVE(\cdot) so that the algorithm exhibits the aforementioned characteristics for as many path trees as possible.

2.4.2 Breadth First Search (BFS)

Definition: Explores the least-recently expanded open node first.

Property	Description
Halting	$d < \infty$ non-NULL
Sound	always
Complete	always
Optimal	constant cst
Time	b^{l^*}
Space	b^{l^*+1}

2.4.3 Depth First Search (DFS)

Definition: Explores the most-recently expanded open node first.

Property	Description
Halting	$d < \infty$
Sound	always
Complete	$d < \infty$
Optimal	never
Time	b^d
Space	bd

2.4.4 Iterative Deepening DFS (IDDFS)

Definition: Same as DFS but with iterative deepening.

Property	Description
Halting	always
Sound	always
Complete	always
Optimal	constant cst
Time	b^{l^*}
Space	bl^*

2.4.5 Cheapest-First Search (CFS)

Definition: Explores the cheapest open node first.

Property	Description
Halting	$d < \infty$ non-NULL
Sound	yes
Complete	$\epsilon > 0$
Optimal	$\epsilon > 0$
Time	$b^{c^*/\epsilon}$
Space	$b^{c^*/\epsilon+1}$

2.5 Modifications to Search Algorithms

2.5.1 Depth-Limiting

Definition: Depth limit of d_{\max} to any search algorithm by modifying SEARCH(\cdot) as follows:

```

1 procedure SEARCHDL( $\mathcal{O}$ ,  $d_{\max}$ ):
2   if  $\mathcal{O} = \emptyset$  then
3     return NULL
4    $n \leftarrow \text{REMOVE}(\mathcal{O})$ 
5   if  $\text{dst}(n) \in \mathcal{G}$  then

```

▷ the search algorithm failed to find a path to a goal
▷ "explore" a node, n

```

6       return n
7   for  $n' \in \text{chl}(n)$  do
8       if  $\text{len}(n') \leq d_{\max}$  then
9            $\mathcal{O} \leftarrow \mathcal{O} \cup \{n'\}$ 
10      SEARCHDL( $\mathcal{O}$ ,  $d_{\max}$ )

```

\triangleright the search algorithm found a path to a goal
 \triangleright "expand" n and "export" its children
 \triangleright unless the child is too long

2.5.2 Iterative Deepening

Definition: Iteratively increase the depth-limit, d_{\max} , to any search algorithm w/ depth-limiting, by placing SEARCHDL(\cdot) in a wrapper, SEARCHID(\cdot):

```

1 procedure SEARCHID():
2      $n \leftarrow \text{NULL}$ 
3      $d_{\max} \leftarrow 0$ 
4      $\triangleright$  while a solution has not been found, reset the open set, run the search algorithm, then increase the
       depth-limit
5     while  $n = \text{NULL}$  do
6          $\mathcal{O} \leftarrow \{(\langle \rangle, 0)\}$ 
7          $n \leftarrow \text{SEARCHDL}(\mathcal{O}, d_{\max})$ 
8          $d_{\max} \leftarrow d_{\max} + 1$ 
9     return  $n$ 

```

Warning: Increasing d_{\max} can be done in different ways.

2.5.3 Cost-Limiting

Definition: Cost limit of c_{\max} to any search algorithm by modifying SEARCH(\cdot) as follows:

```

1 procedure SEARCHCL( $\mathcal{O}$ ,  $c_{\max}$ ):
2     if  $\mathcal{O} = \emptyset$  then
3         return NULL
4      $n \leftarrow \text{REMOVE}(\mathcal{O})$ 
5     if  $\text{dst}(n) \in \mathcal{G}$  then
6         return  $n$ 
7     for  $n' \in \text{chl}(n)$  do
8         if  $\text{cst}(n') \leq c_{\max}$  then
9              $\mathcal{O} \leftarrow \mathcal{O} \cup \{n'\}$ 
10    SEARCHCL( $\mathcal{O}$ ,  $c_{\max}$ )

```

\triangleright the search algorithm failed to find a path to a goal
 \triangleright "explore" a node, n
 \triangleright the search algorithm found a path to a goal
 \triangleright "expand" n and "export" its children
 \triangleright unless the child is too expensive

2.5.4 Iterative-Inflating

Definition: Iteratively increase the cost limit, c_{\max} , to any search algorithm with cost-limiting, by placing SEARCHCL(\cdot) in a wrapper, SEARCHII(\cdot):

```

1 procedure SEARCHII():
2      $n \leftarrow \text{NULL}$ 
3      $c_{\max} \leftarrow 0$ 
4      $\triangleright$  while a solution has not been found, reset the open set, run the search algorithm, then increase the
       cost-limit
5     while  $n = \text{NULL}$  do
6          $\mathcal{O} \leftarrow \{(\langle \rangle, 0)\}$ 
7          $n \leftarrow \text{SEARCHCL}(\mathcal{O}, c_{\max})$ 
8          $c_{\max} \leftarrow c_{\max} + \epsilon$ 
9     return  $n$ 

```

Warning: Increasing c_{\max} can be done in different ways.

2.5.5 Intra-Path Cycle Checking

Definition: Do not expand a path if it is cyclic. Modify `SEARCH(·)` as follows:

```

1 procedure SEARCH( $\mathcal{O}$ ):
2   if  $\mathcal{O} = \emptyset$  then
3     return NULL
4    $n \leftarrow \text{REMOVE}(\mathcal{O})$ 
5   if  $\text{dst}(n) \in \mathcal{G}$  then
6     return  $n$ 
7   for  $n' \in \text{chl}(n)$  do
8     if not CYCLIC( $n'$ ) then
9        $\mathcal{O} \leftarrow \mathcal{O} \cup \{n'\}$ 
10  SEARCH( $\mathcal{O}$ )

```

▷ "expand" n and "export" its children
▷ unless the child is cyclic

- Optimality of an algorithm is preserved provided $\epsilon > 0$.

2.5.6 Inter-Path Cycle Checking

Definition: We modify `SEARCH(·)` as follows:

```

1 procedure SEARCH( $\mathcal{O}, \mathcal{C}$ ):
2   if  $\mathcal{O} = \emptyset$  then
3     return NULL
4    $n \leftarrow \text{REMOVE}(\mathcal{O})$ 
5    $\mathcal{C} \leftarrow \mathcal{C} \cup \{n\}$ 
6   if  $\text{dst}(n) \in \mathcal{G}$  then
7     return  $n$ 
8   for  $n' \in \text{chl}(n)$  do
9     if  $n' \notin \mathcal{C}$  then
10       $\mathcal{O} \leftarrow \mathcal{O} \cup \{n'\}$ 
11  SEARCH( $\mathcal{O}, \mathcal{C}$ )

```

▷ add n to the closed set
▷ "expand" n and "export" its children
▷ unless the child's destination is closed

and then call the algorithm as follows:

```

1  $\mathcal{O} \leftarrow \{(\langle \rangle, 0)\}$ 
2  $\mathcal{C} \leftarrow \emptyset$ 
3 SEARCH( $\mathcal{O}, \mathcal{C}$ )

```

▷ initialize a set of closed vertices

2.6 Informed Search Algorithms

Motivation: We want to somehow explore more "promising" paths first.

2.6.1 Estimated Cost

Definition: `ecst(·)`, to estimate the total cost to a goal given a path, p , based on the following:

- Cost of path p : $\text{cst}(p)$
- Estimate of the extra cost needed to get to a goal from $\text{dst}(p)$: $\text{hur} : S \rightarrow \mathbb{R}_+$
 - $\text{hur}(s)$ estimates the cost to get to \mathcal{G} from s and $\text{hur}(p)$ means $\text{hur}(\text{dst}(p))$.

Example: Some common choices for `ecst(·)` include:

1. $\text{ecst}(p) = \text{hur}(p)$; called nearest-first search (NFS)
2. $\text{ecst}(p) = \text{cst}(p) + \text{hur}(p)$; called A* (A-star)

2.6.2 Heuristics**2.6.3 Heuristic-First Search (HFS)****2.6.4 A-Star Search (A*)****2.6.5 Iterative Inflating A-Star Search (IIA*)****2.6.6 Designing Heuristics via Problem Relaxation****2.6.7 Combining Heuristics****2.7 Characteristics of an Informed Search Algorithm****2.8 Anytime Search Algorithms****2.9 Formulating a Search Problem**