

ECE353 Lectures

Hanhee Lee

January 20, 2025

Contents

1	Prologue	3
1.1	Setup of Planning Problems	3
1.2	Components of a Robotic System	4
1.2.1	Overview (Robots, the Environment)	5
1.2.2	Robot (Sensors, Actuators, the Brain)	5
1.2.3	Brain (Tracker, Planner, Memory)	6
1.2.4	Environment (Physics, State)	6
1.3	Equations of a Robotic System	7
1.3.1	Sensing	7
1.3.2	Tracking	7
1.3.3	Planning	8
1.3.4	Acting	8
1.3.5	Simulating	9
2	Uninformed Search Algorithms	9
2.1	Setup	9
2.2	Search Graphs	9
2.3	Path Trees	9
2.4	Search Algorithms	10
2.4.1	Characteristics of a Search Algorithm	10
2.4.2	Breadth First Search (BFS)	10
2.4.3	Depth First Search (DFS)	11
2.4.4	Iterative Deepening DFS (IDDFS)	11
2.4.5	Cheapest-First Search (CFS)	11
2.5	Modifications to Search Algorithms	11
2.5.1	Depth-Limiting	11
2.5.2	Iterative Deepening	12
2.5.3	Cost-Limiting	12
2.5.4	Iterative-Inflating	12
2.5.5	Intra-Path Cycle Checking	13
2.5.6	Inter-Path Cycle Checking	13
2.6	Informed Search Algorithms	13
2.6.1	Estimated Cost	13
2.7	Characteristics of an Informed Search Algorithm	13
2.7.1	Heuristics	14
2.7.2	Heuristic-First Search (HFS)	14
2.7.3	A-Star Search (A*)	14
2.7.4	Iterative Inflating A-Star Search (IIA*)	14
2.7.5	Designing Heuristics via Problem Relaxation	14
2.7.6	Combining Heuristics	14
2.8	Anytime Search Algorithms	14
2.9	Formulating a Search Problem	14
2.10	Canonical Examples	15
3	Informed Search Algorithms	23

3.1	Admissible and Consistent	23
4	Probability Review	28
5	Probabilistic Inference Problems	28

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.

1.1 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 $\mathcal{A}(s) := \times_i \mathcal{A}_i(s)$ (joint action set)
- 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
- $\text{rwd}_i(s, a, s')$ 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 .

1.2 Components of a Robotic System

Summary:

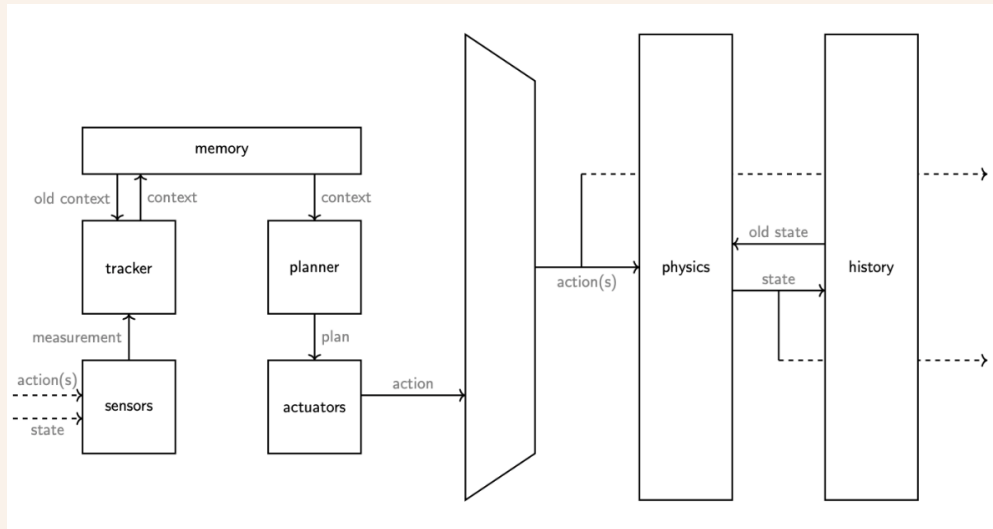


Figure 1: Components of a Robotic System (Words)

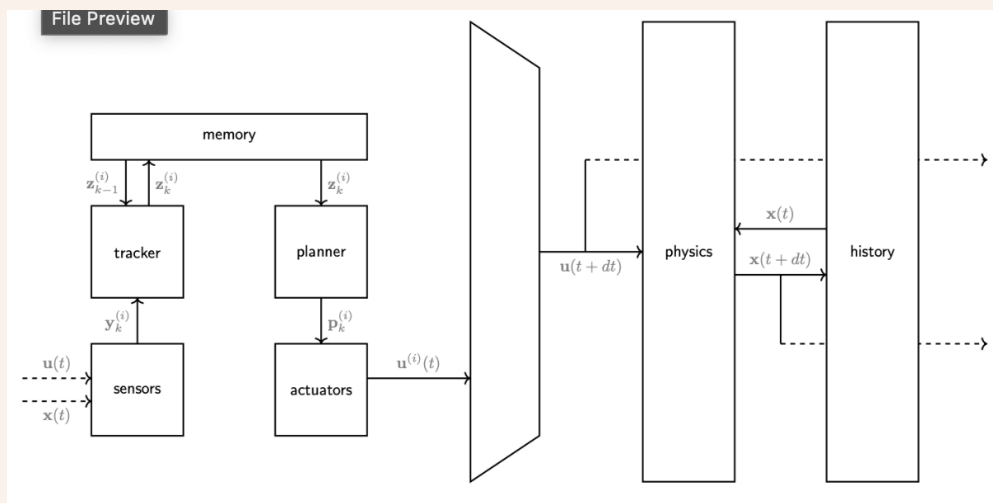


Figure 2: Components of a Robotic System (Math)

1.2.1 Overview (Robots, the Environment)

Definition:

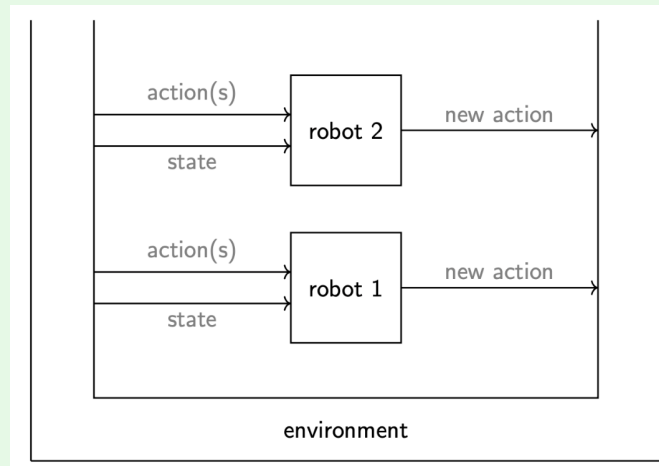


Figure 3: Overview (Robots, the Environment)

Notes:

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

1.2.2 Robot (Sensors, Actuators, the Brain)

Definition:

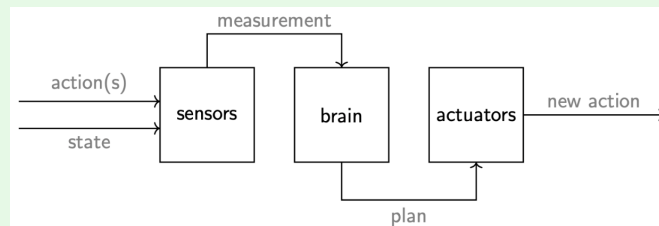


Figure 4: Robot (Sensors, Actuators, the Brain)

Notes:

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

1.2.3 Brain (Tracker, Planner, Memory)

Definition:

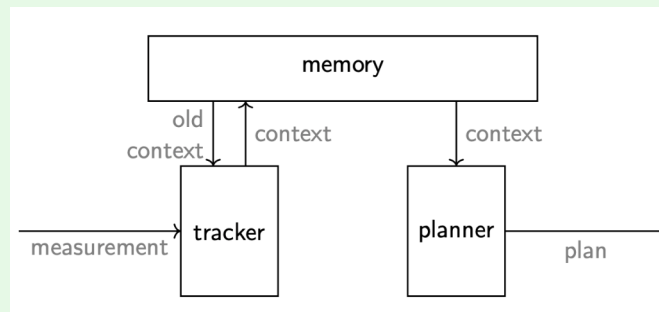


Figure 5: Brain (Tracker, Planner, Memory)

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.

1.2.4 Environment (Physics, State)

Definition:

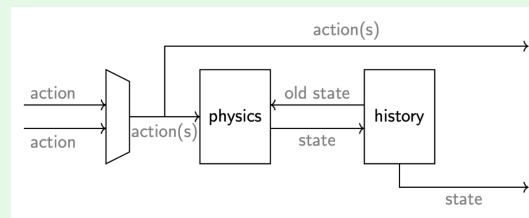


Figure 6: Environment (Physics, State)

1.3 Equations of a Robotic System

1.3.1 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

1.3.2 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

1.3.3 Planning

Definition: Make a plan:

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

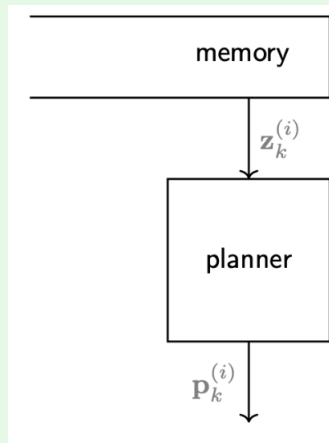


Figure 9: Planning

1.3.4 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:

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

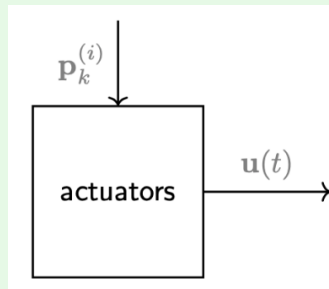


Figure 10: Acting

1.3.5 Simulating

Definition: Simulate the environment's response:

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

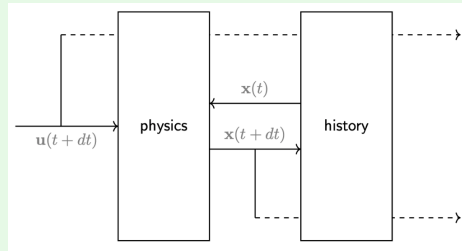


Figure 11: Simulating

2 Uninformed Search Algorithms

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, $s \in S$, 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):

- 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
       $\triangleright$  the search algorithm failed to find a path to a goal
       $\triangleright$  "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 $\text{SEARCH}(\cdot)$ 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 $\text{SEARCH}(\cdot)$ 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

2.6.1 Estimated Cost

Definition: $\text{ecst}(\cdot)$, 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 $\text{ecst}(\cdot)$ 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)

2.7 Characteristics of an Informed Search Algorithm

Definition:

1. Heuristic: $\text{hur}(\cdot)$
2. Cost estimation: $\text{ecst}(\cdot)$

2.7.1 Heuristics**2.7.2 Heuristic-First Search (HFS)****2.7.3 A-Star Search (A*)****2.7.4 Iterative Inflating A-Star Search (IIA*)****2.7.5 Designing Heuristics via Problem Relaxation****2.7.6 Combining Heuristics****2.8 Anytime Search Algorithms****2.9 Formulating a Search Problem**

2.10 Canonical Examples

Process: How to setup a search problem?

1. Given a search graph, we need to define the following:
 - \mathcal{S} : set of vertices
 - \mathcal{G} : goal states (subset of \mathcal{S})
 - $s^{(0)}$: initial state
 - \mathcal{T} : set of edges (defined by $\text{tr}(\cdot, \cdot)$)
 - $\text{tr}(\cdot, \cdot)$: transition function
 - $\text{cst}(\cdot, \cdot, \cdot)$: cost function (defined by edge weights)

Example:

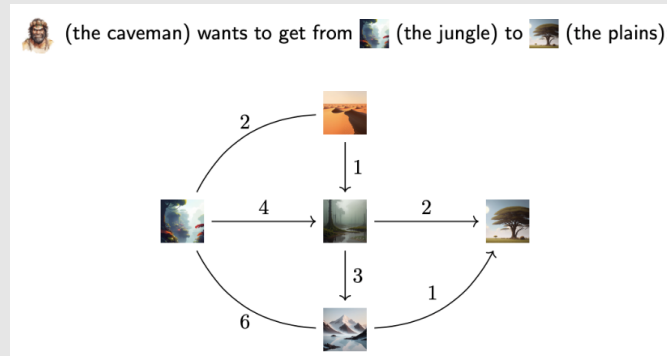


Figure 12

In our example, $\mathcal{S} = \left\{ \text{jungle}, \text{savanna}, \text{desert}, \text{mountains}, \text{plains} \right\}$, $\mathcal{G} = \left\{ \text{plains} \right\}$,
 $s^{(0)} = \text{jungle}$, and one possible transition is $\langle \text{jungle}, \emptyset, \text{savanna} \rangle$, at a cost of 4.

Figure 13

Example:

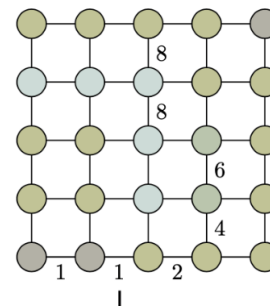
👤 is hungry and must get to 🍏 while minimizing energy consumption.



His energy consumption for a given step depends on the terrain transition.

Figure 14

We can model 👤's situation as a search problem.



edge weights are identical for identical terrain transitions

Figure 15

- $\mathcal{S} = \{0, \dots, 4\}^2$
- $\mathcal{G} = \left\{ \begin{bmatrix} 1 \\ 4 \end{bmatrix} \right\}$
- $s^{(0)} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

Process: How to setup a path tree?

1. Start at $s^{(0)}$
2. Choose a path until you reach a goal state.
3. Repeat until you have found all paths (probably infinite).

Example:

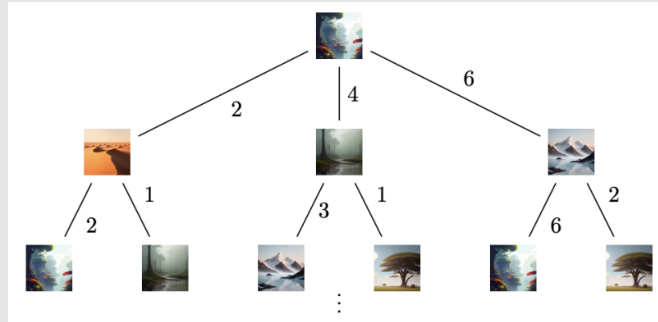


Figure 16

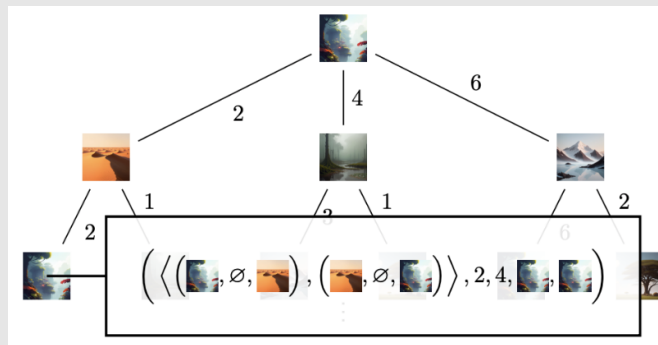


Figure 17

Process: When to use each algorithm?

1. Find properties needed for the problem and match them to the characteristics of the algorithm.
2. Choose the algorithm that best matches the properties.
 - **BFS:**
 - **DFS:**
 - **IDDFS:**
 - **CFS:**
 - **A*:**

Example:

Process: How to Trace Through a Search Algorithm

1.

Example: BFS

Example: DFS

Example: IDDFS

Example: CFS

Example: HFS

Example: A*

Example: IIA*

Example: WA*

Process: How to Figure Out Soln. w/o Performing Search Algorithm?

1.

Example:

Process: How to Prove Consistent/Admissible Given a Search Graph?**Admissible:**

1. Given $hur(s)$ and search graph with $cst(s, a, tr(s, a))$ or $rwd(s, a, tr(s, a))$. If consistent, then it is admissible.
2. Check $\forall s \in \mathcal{G}, hur(s) = 0$. If not, then it is not admissible.
3. For each $s \in \mathcal{S}$, calculate $hur^*(s)$ (i.e. actual cost of optimal soln.) using the search graph.
 - (a) Start at s and choose path that gives the lowest cost or highest reward to $s \in \mathcal{G}$.
4. Check if $hur(s) \leq hur^*(s) \forall s \in \mathcal{S}$. If not, then it is not admissible.
5. Repeat $\forall s \in \mathcal{S}$.
6. If all are true, then it is admissible.

Consistent:

1. Given $hur(s)$ and search graph with $cst(s, a, tr(s, a))$ or $rwd(s, a, tr(s, a))$.
2. Check $\forall s \in \mathcal{G}, hur(s) = 0$. If not, then it is not consistent.
3. For each $s \in \mathcal{S}$, calculate $hur(s) - hur(tr(s, a))$.
 - (a) check if it is $\leq cst(s, a, tr(s, a))$ or $\geq rwd(s, a, tr(s, a))$. If not, then it is not consistent.
 - (b) Repeat $\forall a \in \mathcal{A}(s)$
4. Repeat $\forall s \in \mathcal{S}$.
5. If all are true, then it is consistent.

Warning: Be careful of bidirectional edges bc for consistency you need compute the cost of the heuristic edge in both directions.

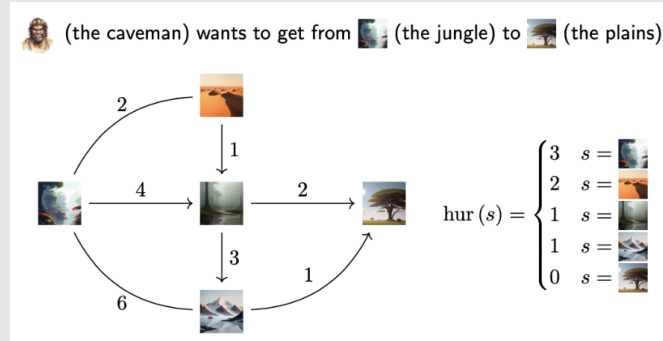
Example:

Figure 18: Jungle ($s^{(0)}$), Desert, Swamp, Mountain, Plains (Goal)

Admissible:

1. $s = \text{Plains}$: $hur(\text{Plains}) = 0$
2. $s = \text{Jungle}$: $hur(\text{Jungle}) = 3 \leq hur^*(\text{Jungle}) = 2 + 1 + 2 = 5$
3. $s = \text{Desert}$: $hur(\text{Desert}) = 2 \leq hur^*(\text{Desert}) = 1 + 2$
4. $s = \text{Swamp}$: $hur(\text{Swamp}) = 1 \leq hur^*(\text{Swamp}) = 2$
5. $s = \text{Mountain}$: $hur(\text{Mountain}) = 1 \leq hur^*(\text{Mountain}) = 1$
6. Therefore, it is admissible.

Consistent:

1. $s = \text{Plains}$: $hur(\text{Plains}) = 0$
2. $s = \text{Jungle}$:
 - (a) $hur(\text{Jungle}) - hur(\text{Desert}) = 3 - 2 = 1 \leq cst(\text{Jungle}, \cdot, \text{Desert}) = 2$
 - (b) $hur(\text{Jungle}) - hur(\text{Swamp}) = 3 - 1 = 2 \leq cst(\text{Jungle}, \cdot, \text{Swamp}) = 4$
 - (c) $hur(\text{Jungle}) - hur(\text{Mountain}) = 3 - 1 = 2 \leq cst(\text{Jungle}, \cdot, \text{Mountain}) = 6$
3. $s = \text{Desert}$:
 - (a) $hur(\text{Desert}) - hur(\text{Jungle}) = 2 - 3 = -1 \leq cst(\text{Desert}, \cdot, \text{Jungle}) = 2$
 - (b) $hur(\text{Desert}) - hur(\text{Swamp}) = 2 - 1 = 1 \leq cst(\text{Desert}, \cdot, \text{Swamp}) = 1$
4. $s = \text{Swamp}$:
 - (a) $hur(\text{Swamp}) - hur(\text{Mountain}) = 1 - 1 = 0 \leq cst(\text{Swamp}, \cdot, \text{Mountain}) = 3$

- (b) $\text{hur}(\text{Swamp}) - \text{hur}(\text{Plains}) = 1 - 0 = 1 \leq \text{cst}(\text{Swamp}, \cdot, \text{Plains}) = 2$
- 5. $s = \mathbf{Mountain}$:
 - (a) $\text{hur}(\text{Mountain}) - \text{hur}(\text{Jungle}) = 1 - 3 = -2 \leq \text{cst}(\text{Mountain}, \cdot, \text{Desert}) = 6$
 - (b) $\text{hur}(\text{Mountain}) - \text{hur}(\text{Plains}) = 1 - 0 = 1 \leq \text{cst}(\text{Mountain}, \cdot, \text{Plains}) = 1$
- 6. Therefore, it is consistent.

Process: How to Design Heuristic via Problem Relaxation?

1. Make an assumption to simplify the problem as a relaxed problem.
2. Find the cost of the optimal solution of the relaxed problem, $\text{cst}_{\text{rel}}(s)$.
3. HOW TO FIND THE COST OF THE OPTIMAL SOLUTION?

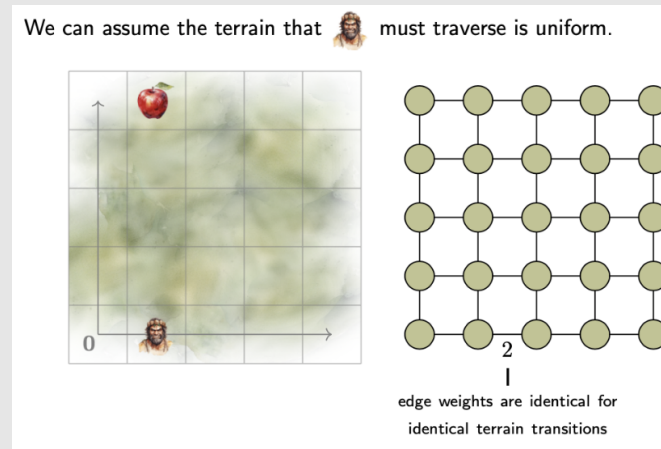
Example:

Figure 19

3 Informed Search Algorithms

Summary:**Example:** Different ways to formulate the CSP problem.

- How can you formulate the CSP problem in a different way? Can I get a specific example?
 - The domain could be set to everything, then set the constraints later.
- What is the constraint graph showing? Grouping the variables
- How do you check consistency in a CSP?
- Why can you use any search algorithm when you formulate this as a search problem?
- What does a node contain? A node contains a path.
 - How does that match the example on slide 10. It does.
- Why is formulating a CSP problem as a search problem a bad idea? B/c you have to search through all possible combinations, but if you find a constraint then you can prune the search space.
 - A lot easier to see if there is a solution or not. But in a search problem, you see if there's a solution and how to get to it.


3.1 Admissible and Consistent

Summary: Want a way to learn heuristics.

Process: How to Setup a CSP?

1.

Example:

 now wants to find food to meet his nutritional requirements:










		Nutrients (25 g)			
	Supply	 Carbs	 Fat	 Protein	 Vitamins
Minimum	----	8	3	2	1
 Nuts	4	2	1	1	0
 Fruits	5	2	0	0	0
 Legumes	4	2	0	1	1
 Grains	6	3	0	0	0
 Meat	3	0	1	2	0
Maximum	----	10	4	5	1

Figure 20

For our example, the variables could be:

$$\begin{array}{ll}
 \begin{array}{c} \text{Nut icon} \\ \in \{0, 1, 2, 3, 4\} \\ \text{dom}(\text{Nut icon}) \end{array} & \begin{array}{c} \text{Apple icon} \\ \in \{0, 1, 2, 3, 4, 5\} \\ \text{dom}(\text{Apple icon}) \end{array} \\
 \begin{array}{c} \text{Carrot icon} \\ \in \{0, 1, 2, 3, 4\} \\ \text{dom}(\text{Carrot icon}) \end{array} & \begin{array}{c} \text{Grain icon} \\ \in \{0, 1, 2, 3, 4, 5, 6\} \\ \text{dom}(\text{Grain icon}) \end{array} \\
 \begin{array}{c} \text{Meat icon} \\ \in \{0, 1, 2, 3\} \\ \text{dom}(\text{Meat icon}) \end{array} &
 \end{array}$$

Figure 21

For our example, the constraints could be:

$$\begin{array}{l}
 \begin{array}{c} \text{Fire icon} \\ : \end{array} \underbrace{8 \leq 2 \text{Nut icon} + 2 \text{Apple icon} + 2 \text{Carrot icon} + 3 \text{Grain icon} \leq 10}_{\text{scp}(\text{Fire icon}) = \{\text{Nut icon}, \text{Apple icon}, \text{Carrot icon}, \text{Grain icon}\}} \\
 \begin{array}{c} \text{Drop icon} \\ : \end{array} \underbrace{3 \leq \text{Nut icon} + \text{Meat icon} \leq 4}_{\text{scp}(\text{Drop icon}) = \{\text{Nut icon}, \text{Meat icon}\}} \\
 \begin{array}{c} \text{Molecule icon} \\ : \end{array} \underbrace{2 \leq \text{Nut icon} + \text{Carrot icon} + 2 \text{Meat icon} \leq 5}_{\text{scp}(\text{Molecule icon}) = \{\text{Nut icon}, \text{Carrot icon}, \text{Meat icon}\}} \\
 \begin{array}{c} \text{Pill icon} \\ : \end{array} \underbrace{1 \leq \text{Carrot icon} \leq 2}_{\text{scp}(\text{Pill icon}) = \{\text{Carrot icon}\}}
 \end{array}$$

Figure 22

Process: How to build a hyper-graph?

1. Circle the variables that appear in constraint C_i .

Example:

We can visualize the constraints using a hyper-graph.

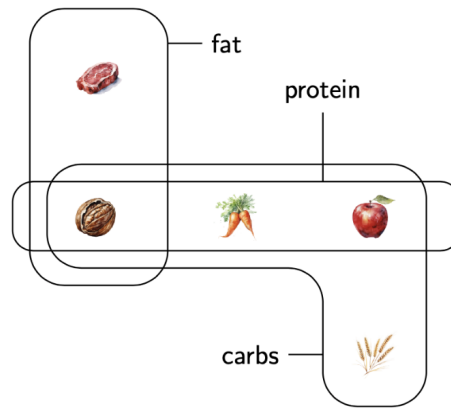


Figure 23

Process: How to build a path tree?

- 1.

Example:

The path tree traversed lists all possible ways to build a complete assignment:

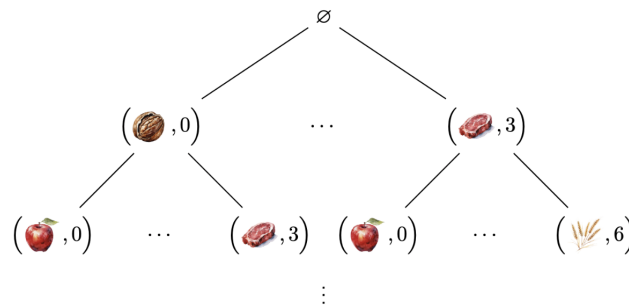


Figure 24

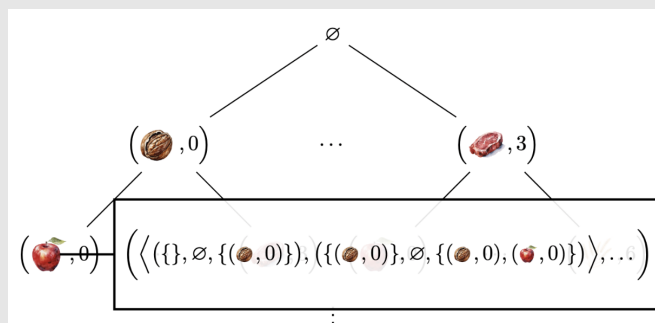


Figure 25

Process: How to determine a solution to a CSP?

1.

Example:

$$\left\{ \left(\text{🍌}, 2 \right), \left(\text{🍏}, 1 \right), \left(\text{🥕}, 1 \right), \left(\text{🌾}, 0 \right), \left(\text{🥩}, 1 \right) \right\}$$

Figure 26

Process: How to check k -Consistency? FIS

1. Given a set of variables \mathcal{V} w/ $\text{dom}(V) = \{v_1, \dots, v_{\text{len}(V)}\} \forall V \in \mathcal{V}$ and a set of constraints \mathcal{C} w/ $\text{scp}(C) = \{V_1, \dots, V_{\text{len}(C)}\} \forall C \in \mathcal{C}$, check k -consistency.
2. For each $C \in \mathcal{C}$, do the following:
 - (a) For $V \in \text{scp}(C)$, fix V to a value in $\text{dom}(V)$.
 - i. For the other $V \in \text{scp}(C)$, check if the constraint is satisfied by trying all combinations.
 - (b) If there is one combination that doesn't satisfy the constraint, then the CSP is not k -consistent.
 - (c) Repeat $\forall V \in \text{scp}(C)$.
3. Repeat $\forall C \in \mathcal{C}$.
4. If all constraints are satisfied, then the CSP is k -consistent.

Process: How to Enforce k -Consistency? FIX

1. Given a set of variables \mathcal{V} w/ $\text{dom}(V) = \{v_1, \dots, v_{\text{len}(V)}\} \forall V \in \mathcal{V}$ and a set of constraints \mathcal{C} w/ $\text{scp}(C) = \{V_1, \dots, V_{\text{len}(C)}\} \forall C \in \mathcal{C}$, enforce k -consistency.
2. For each $C \in \mathcal{C}$, do the following:
 - (a) For $V \in \text{scp}(C)$, fix V to a value in $\text{dom}(V)$.
 - i. For the other $V \in \text{scp}(C)$, check if the constraint is satisfied by trying all combinations. If the constraint is not satisfied, then remove the value from $\text{dom}(V)$.
 - (b) Repeat $\forall V \in \text{scp}(C)$.
3. Check the resulting $\text{dom}(V) = \{v_1, \dots, v_{\text{len}(V)}\} \forall V \in \mathcal{V}$ w/ the other constraints.
4. Repeat $\forall C \in \mathcal{C}$.

Example:

- $\mathcal{V} = \left\{ \text{wheat}, \text{beef}, \text{carrots} \right\}$
- $\text{dom} \left(\text{wheat} \right) = \{1, 2, 3\}$
- $\text{dom} \left(\text{carrots} \right) = \{2, 3, 4\}$
- $\text{dom} \left(\text{beef} \right) = \{1, 2, 4\}$
- $\mathcal{C} = \left\{ \underbrace{\text{wheat} + \text{carrots}}_C = \text{beef} \right\}$

Figure 27

- $\text{dom} \left(\text{wheat} \right) = \{1, 2, \cancel{3}\}$
- $\text{dom} \left(\text{carrots} \right) = \{2, 3, \cancel{4}\}$
- $\text{dom} \left(\text{beef} \right) = \{\cancel{1}, \cancel{2}, 4\}$

Figure 28

4 Probability Review

5 Probabilistic Inference Problems