# ECE353 Lectures

Hanhee Lee

January 28, 2025

# Contents

# 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

**Definition**: 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
- $\mathrm{rwd}_i(s, a, s')$ is agent $i$'s reward for the transition, $(s, a, s')$
- a **path** is any sequence of transitions of the form

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

- 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**:
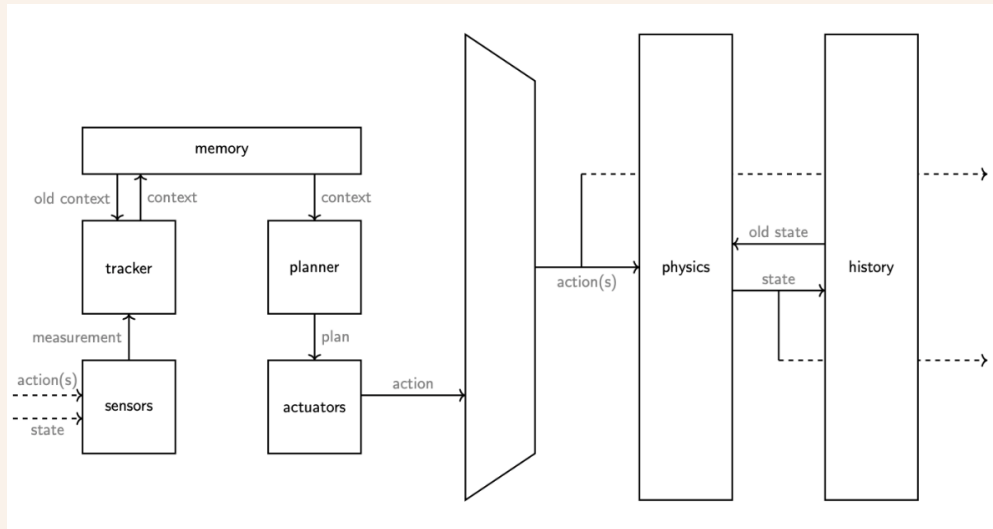


Figure 1: Components of a Robotic System (Words)
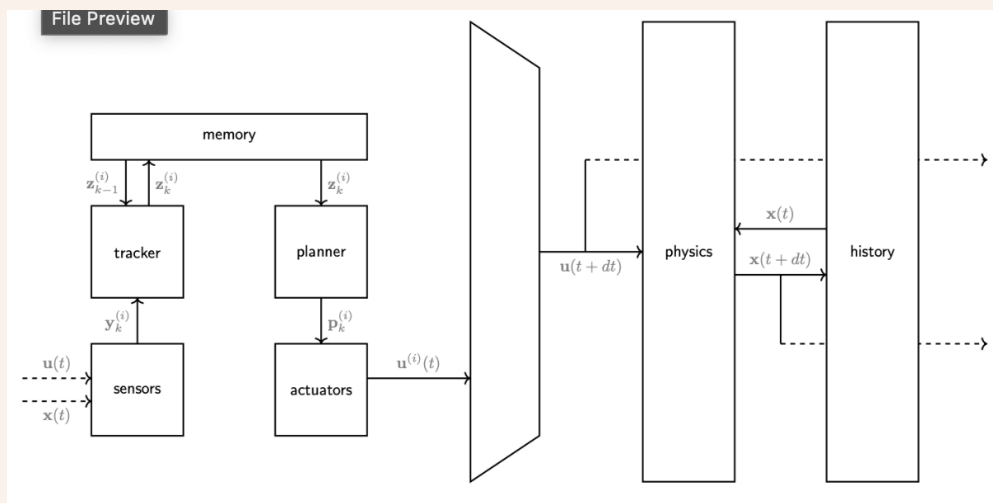


Figure 2: Components of a Robotic System (Math)

### 1.2.1   Overview (Robots, the Environment)

**Definition**:
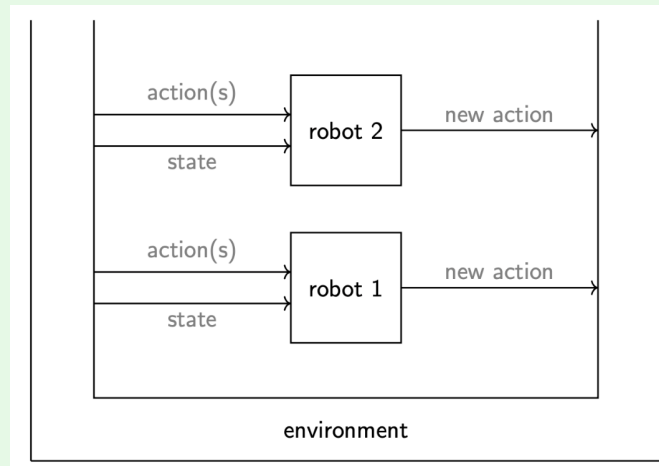


Figure 3: Overview (Robots, the Environment)

**Notes**:
- Environment → previous actions + current state → robot → new action → environment

### 1.2.2   Robot (Sensors, Actuators, the Brain)

**Definition**:
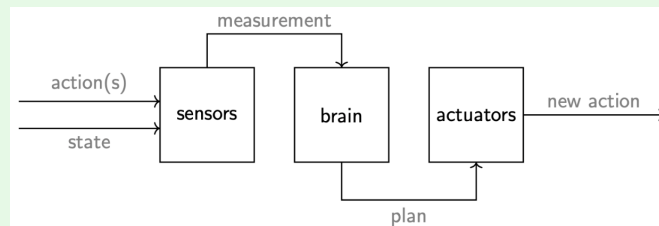


Figure 4: Robot (Sensors, Actuators, the Brain)

**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**:
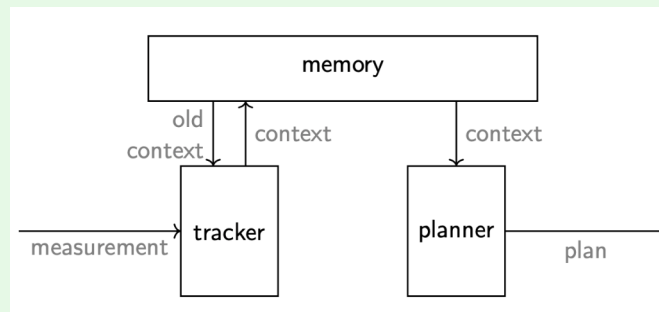


Figure 5: Brain (Tracker, Planner, Memory)

**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**:
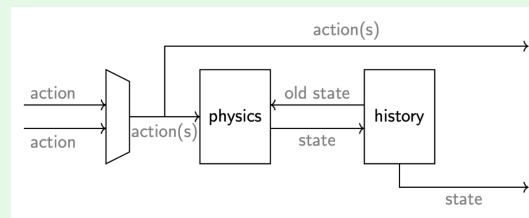


Figure 6: Environment (Physics, State)

## 1.3  Equations of a Robotic System

### 1.3.1  Sensing

**Definition**: Take a measurement:
$$\mathbf{y}^{(i)}(t) = \text{sns}^{(i)}\big(\mathbf{x}(t), \mathbf{u}(t), t\big)$$

Convert the measurement into a discrete-time signal using a sampling period of $T^{(i)}$:
$$\mathbf{y}_k^{(i)} = \text{dt}\big(\mathbf{y}^{(i)}(t), t, T^{(i)}\big)$$



Figure 7: Sensing

### 1.3.2  Tracking

**Definition**: Track (update) the context:
$$\mathbf{z}_k^{(i)} = \text{trk}^{(i)}\big(\mathbf{z}_{k-1}^{(i)}, \mathbf{y}_k^{(i)}, k\big)$$



Figure 8: Tracking

### 1.3.3 Planning

**Definition**: Make a plan:

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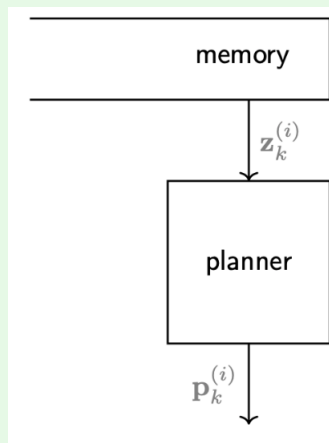


Figure 9: Planning

### 1.3.4 Acting

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

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

Execute the plan:

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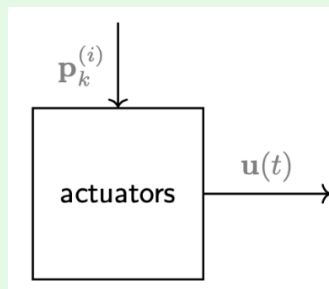


Figure 10: Acting

### 1.3.5 Simulating

**Definition**: Simulate the environment's response:

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



Figure 11: Simulating

# 2   Search Algorithms

**Summary**:

| Alg. | Halting | Sound | Complete | Optimal | Time | Space |
|---|---|---|---|---|---|---|

Choose `REMOVE(·)` so algo. exhibits the characteristics:
- **Halting:** Terminates after finitely many nodes explored | **Sound:** Returned (possibly NULL) soln. is correct
- **Complete:** Halting & sound when a non-NULL soln. exists | **Opt.:** Returns an opt. soln. when mult. exist
- **Time:** Minimizes nodes **explored**/expanded/exported | **Space:** Minimizes nodes simultaneously open

Choose `REMOVE(·)` so algo. exhibits the characteristics for as many path trees as possible.
- $b$ ($b < \infty$): Branching factor (the maximum number of children a node can have)
- $d$: Depth (the length of the longest path), $l^*$: Length of the shortest solution
- $c^*$: Cost of the cheapest solution, $\epsilon$: Cost of the cheapest edge

| **Uninformed Search Algorithms** | | | | | | |
|---|---|---|---|---|---|---|
| **BFS** | $d < \infty$, non-`NULL` | always | always | constant cst | $b^{l^*}$ | $b^{l^*+1}$ |

- Explores the least-recently expanded open node first.

| **DFS** | $d < \infty$ | always | $d < \infty$ | never | $b^d$ | $bd$ |
|---|---|---|---|---|---|---|

- Explores the most-recently expanded open node first.

| **IDDFS** | always | always | always | constant cst | $b^{l^*}$ | $bl^*$ |
|---|---|---|---|---|---|---|

- Same as DFS but with iterative deepening.

| **CFS** | $d < \infty$, non-`NULL` | yes | $\epsilon > 0$ | $\epsilon > 0$ | $b^{c^*/\epsilon}$ | $b^{c^*/\epsilon+1}$ |
|---|---|---|---|---|---|---|

- Explores the cheapest open node first.

| **Informed Search Algorithms** | | | | | | |
|---|---|---|---|---|---|---|
| **HFS** | $d < \infty$ | never | never | never | - | - |

- Explores the node with the smallest hur-value first, $\mathrm{ecst}(p) = \mathrm{hur}(p)$

| **A**$^*$ | hur admissible, $\epsilon > 0$ | always | hur admissible, $\epsilon > 0$ | hur admissible, $\epsilon > 0$ | $O\left(b^{c^*/\epsilon}\right)$ | $O\left(b^{c^*/\epsilon+1}\right)$ |
|---|---|---|---|---|---|---|

- Explores the node with the smallest ecst-value first, $\mathrm{ecst}(p) = \mathrm{cst}(p) + \mathrm{hur}(p)$

| **IIA**$^*$ | always | always | always | always | $b^{l^*}$ | $bl^*$ |
|---|---|---|---|---|---|---|

- Same as A$^*$ but with iterative inflating on ecst.

| **WA**$^*$ | - | - | - | - | - | - |
|---|---|---|---|---|---|---|

- Same as A$^*$ but $\mathrm{ecst}(s) = \mathrm{wcst}(s) + (1 - w)\mathrm{hur}(s)$ w/ $w \in [0, 1]$
- $w = 0$: HFS, $w = 0.5$: A$^*$, $w = 1$: CFS, iteratively increasing $w$ from 0 to 1: anytime version of $WA^*$

## 2.1    Modifications to Search Algorithms:

**Summary**:

---

**Modifications**

**Depth-Limiting**

- Enforce a depth limit, $d_{\max}$, to any search algorithm.

---

**Iterative-Deepening**

- Iteratively increase the depth-limit to any search algorithm w/ depth-limiting.

---

**Cost-Limiting**

- Enforce a cost limit of $c_{\max}$ to any search algorithm.

---

**Iterative Inflating**

- Iteratively increase the cost limit, $c_{\max}$, to any search algorithm w/ cost-limiting.

---

**Intra-Path Cycle Checking**

- Do not expand a path if it is cyclic.

---

**Inter-Path Cycle Checking**

- Do not expand a path if its destination is that of an explored path.

## 2.2 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`.

## 2.3 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.

## 2.4 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 origiin, cost, etc).

## 2.5 Search Algorithms

**Definition**: All search algorithms follow the template below:

```
1  O ← {(⟨⟩, 0)}                                      ▷ initialize a set of open nodes
2  SEARCH(O)
```

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

```
1  procedure SEARCH(O)
2      if O = ∅ then
3          return NULL                      ▷ the search algorithm failed to find a path to a goal
4      n ← REMOVE(O)                                             ▷ "explore" a node n
5      if DST(n) ∈ G then
6          return n                            ▷ the search algorithm found a path to a goal
7      for n' ∈ CHL(n) do
8          O ← O ∪ {n'}                                  ▷ "expand" n and "export" its children
9      SEARCH(O)
```

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

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

## 2.6    Modifications to Search Algorithms

### 2.6.1    Depth-Limiting

**Definition**:

```
1  procedure SEARCHDL(𝒪, d_max):
2      if 𝒪 = ∅ then
3          return NULL                              ▷ the search algorithm failed to find a path to a goal
4      n ← REMOVE(𝒪)                                                            ▷ "explore" a node, n
5      if dst(n) ∈ 𝒢 then
6          return n                                         ▷ the search algorithm found a path to a goal
7      for n' ∈ chl(n) do                                       ▷ "expand" n and "export" its children
8          if len(n') ≤ d_max then                                       ▷ unless the child is too long
9              𝒪 ← 𝒪 ∪ {n'}
10     SEARCHDL(𝒪, d_max)
```

### 2.6.2    Iterative Deepening

**Definition**:

```
1  procedure SEARCHID():
2      n ← NULL
3      d_max ← 0
4      ▷ while a solution has not been found, reset the open set, run the search algorithm, then increase the
       depth-limit
5      while n = NULL do
6          𝒪 ← {(⟨⟩, 0)}
7          n ← SEARCHDL(𝒪, d_max)
8          d_max ← d_max + 1
9      return n
```

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

### 2.6.3    Cost-Limiting

**Definition**:

```
1  procedure SEARCHCL(𝒪, c_max):
2      if 𝒪 = ∅ then
3          return NULL                              ▷ the search algorithm failed to find a path to a goal
4      n ← REMOVE(𝒪)                                                            ▷ "explore" a node, n
5      if dst(n) ∈ 𝒢 then
6          return n                                         ▷ the search algorithm found a path to a goal
7      for n' ∈ chl(n) do                                       ▷ "expand" n and "export" its children
8          if cst(n') ≤ c_max then                                  ▷ unless the child is too expensive
9              𝒪 ← 𝒪 ∪ {n'}
10     SEARCHCL(𝒪, c_max)
```

### 2.6.4 Iterative-Inflating

**Definition**:

```
 1  procedure SEARCHII():
 2      n ← NULL
 3      c_max ← 0
 4      ▷ while a solution has not been found, reset the open set, run the search algorithm, then increase the
        cost-limit
 5      while n = NULL do
 6          O ← {(⟨⟩, 0)}
 7          n ← SEARCHCL(O, c_max)
 8          c_max ← c_max + ε
 9      return n
```

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

### 2.6.5 Intra-Path Cycle Checking

**Definition**:

```
 1  procedure SEARCH(O):
 2      if O = ∅ then
 3          return NULL
 4      n ← REMOVE(O)
 5      if dst(n) ∈ G then
 6          return n
 7      for n' ∈ chl(n) do                        ▷ "expand" n and "export" its children
 8          if not CYCLIC(n') then                       ▷ unless the child is cyclic
 9              O ← O ∪ {n'}
10      SEARCH(O)
```

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

### 2.6.6 Inter-Path Cycle Checking

**Definition**:

```
 1  procedure SEARCH(O, C):
 2      if O = ∅ then
 3          return NULL
 4      n ← REMOVE(O)
 5      C ← C ∪ {n}                                    ▷ add n to the closed set
 6      if dst(n) ∈ G then
 7          return n
 8      for n' ∈ chl(n) do                        ▷ "expand" n and "export" its children
 9          if n' ∉ C then                ▷ unless the child's destination is closed
10              O ← O ∪ {n'}
11      SEARCH(O, C)
```

and then call the algorithm as follows:

```
 1  O ← {(⟨⟩, 0)}
 2  C ← ∅                                      ▷ initialize a set of closed vertices
 3  SEARCH(O, C)
```

## 2.7   Informed Search Algorithms

### 2.7.1   Estimated Cost

**Definition**: ecst($\cdot$): estimate of the total cost to a goal given a path, $p$, based on:
- cst($p$): Cost of path $p$
- hur $: S \to \mathbb{R}_+$: Estimate of the extra cost needed to get to a goal from dst($p$)
  - hur($s$) estimates the cost to get to $\mathcal{G}$ from $s$ and hur($p$) means hur(dst($p$)).

### 2.7.2   Admissible

**Definition**: A heuristic, hur($\cdot$), is said to be **admissible** if

$$\text{hur}(s) \leq \text{hur}^*(s)$$

for all $s \in \mathcal{S}$ and

$$\text{hur}(s) = 0$$

for all $s \in \mathcal{G}$.

### 2.7.3   Consistent

**Definition**: A heuristic, hur($\cdot$), is said to be **consistent** if

$$\underbrace{\text{hur}(s) - \text{hur}(\text{tr}(s,a))}_{\text{estimated cost of the transition } (s,a,\text{tr}(s,a))} \quad \leq \quad \underbrace{\text{cst}(s,a,\text{tr}(s,a))}_{\text{true cost of the transition, } (s,a,\text{tr}(s,a))}$$

for all $s \in \mathcal{S}$, and $a \in \mathcal{A}(s)$, and

$$\text{hur}(s) = 0$$

for all $s \in \mathcal{G}$.

### 2.7.4   Domination

**Definition**: If $\text{hur}_1$ and $\text{hur}_2$ are admissible, then:
- $\text{hur}_1$ **strongly dominates** $\text{hur}_2$ if for all $s \in \mathcal{S} \setminus \mathcal{G}$:

$$\text{hur}_1(s) > \text{hur}_2(s)$$

- $\text{hur}_1$ **weakly dominates** $\text{hur}_2$ if for all $s \in \mathcal{S}$:

$$\text{hur}_1(s) \geq \text{hur}_2(s)$$

and for some $s \in \mathcal{S}$:

$$\text{hur}_1(s) > \text{hur}_2(s)$$

### 2.7.5   Combining Heuristics

**Definition**: If $\{\text{hur}_k(\cdot)\}_k$ are admissible (resp. consistent), then $\max_k\{\text{hur}_k\}(\cdot)$ is also admissible (resp. consistent).

**Definition**: If $\text{hur}_{\max} \equiv \max\{\text{hur}_1, \text{hur}_2\}$, then if $\text{hur}_k$ is consistent:

$$\text{hur}_k(s) - \text{hur}_k(\text{tr}(s,a)) \leq \text{cst}(s,a,\text{tr}(s,a))$$

$$\text{hur}_{\max}(s) = \text{hur}_{\max}(\text{tr}(s,a)) - \text{cst}^*(s,a,\text{tr}(s,a))$$

### 2.7.6   Anytime Search Algorithms

**Definition**: An **anytime algorithm** finds a solution quickly (even if it is sub-optimal), and then iteratively improves it (if time permits).

## 2.8   Canonical Examples

> **Process**: **How to setup a search problem?**
> 1. Givne 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)
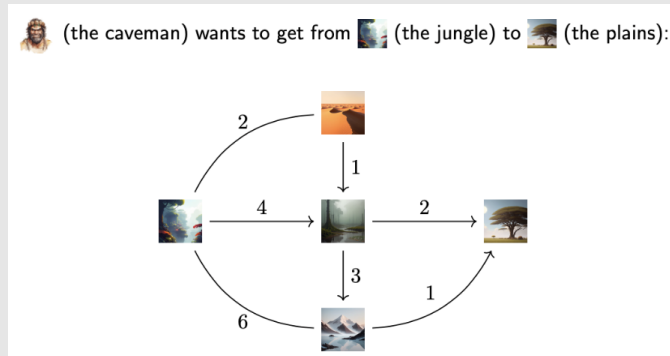
**Example**:



Figure 12
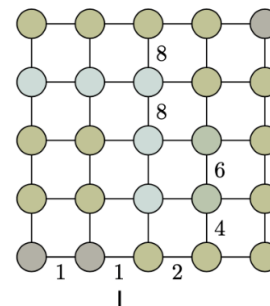


Figure 13

**Example**:



Figure 14



Figure 15

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

**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.
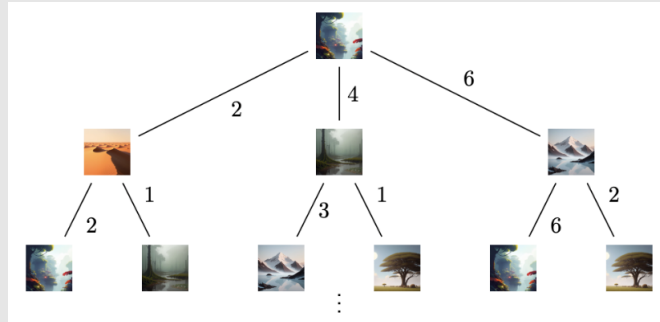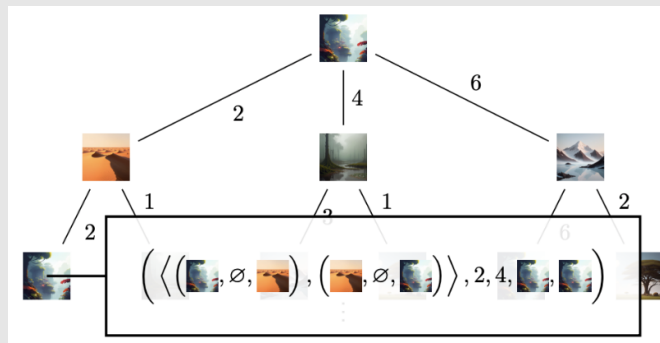    - **BFS:**
    - **DFS:**
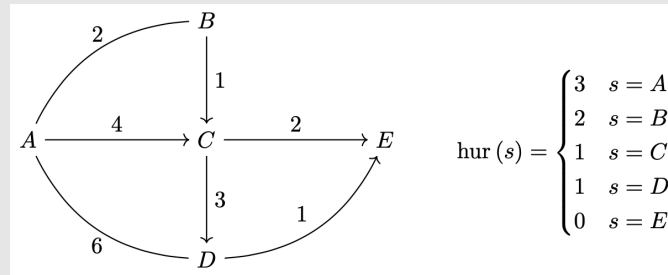    - **IDDFS:**
    - **CFS:**
    - **A*:**

**Example:**

**Example:**



$$\text{hur}(s) = \begin{cases} 3 & s = A \\ 2 & s = B \\ 1 & s = C \\ 1 & s = D \\ 0 & s = E \end{cases}$$

Figure 18

**Process: BFS**

1. Start at $s_0$
2. Expand all neighboring nodes of the current node and add them to the open set (queue).
3. Remove the current node from the open set and mark it as visited.
4. Repeat steps 2 and 3 until the goal state is reached or the open set is empty.

**Example: BFS**

| Path | Open Set |
|------|----------|
|  | $\{A\}$ |
| $A$ | $\{B, C, D\}$ |
| $AB$ | $\{C, D, E\}$ |
| $ABC$ | $\{D, E\}$ |
| $ABCD$ | $\{E\}$ |
| $ABCDE$ | $\{\}$ |

**Process: DFS**

1. Start at $s_0$ (initial state).
2. Push the initial node onto the stack.
3. Pop a node from the stack and expand it.
4. Push all unvisited children of the current node onto the stack.
5. Repeat steps 3 and 4 until the goal state is reached or the stack is empty.

**Example: DFS**

| Path | Open Set |
|------|----------|
|  | $\{A\}$ |
| $A$ | $\{B, C, D\}$ |
| $AD$ | $\{B, C, E\}$ |
| $ADE$ | $\{C, B\}$ |

**Process: IDDFS**

1. Start with a depth limit of 0.
2. Perform Depth-First Search (DFS) up to the current depth limit.
3. If the goal state is not reached and there are unexplored nodes, increment the depth limit and repeat step 2.
4. Continue until the goal state is found or all nodes are explored.

**Example: IDDFS**

| Depth | Path | Open Set |
|-------|------|----------|
| 0 |  | $\{A\}$ |
| 0 | A | $\{\}$ |
| 1 | $A$ | $\{B, C, D\}$ |
| 1 | $AB$ | $\{C, D\}$ |
| 1 | $AC$ | $\{D\}$ |
| 1 | $AD$ | $\{\}$ |
| 2 | $AD$ | $\{E\}$ |
| 2 | $ADE$ | $\{\}$ |

**Process: CFS**

1. Start at $s_0$ (initial state).
2. Initialize the open set (priority queue) with the initial state and its cost.
3. Remove the node with the lowest cost from the open set.
4. Expand the node and add all unvisited neighbors to the open set with their cumulative costs.
5. Repeat steps 3 and 4 until the goal state is reached or the open set is empty.

**Example: CFS**

| Path | Open Set |
|------|----------|
|  | $\{A \mid 0\}$ |
| $A$ | $\{AB \mid 2,\ AC \mid 4,\ AD \mid 6\}$ |
| $AB$ | $\{AC \mid 4,\ AD \mid 6,\ ABC \mid 3\}$ |
| $ABC$ | $\{AC \mid 4,\ AD \mid 6,\ ABCE \mid 5,\ ABCD \mid 6\}$ |
| $ABCE$ | $\{AC \mid 4,\ AD \mid 6,\ ABCD \mid 6\}$ |

**Warning:**

- How to perform shortcut?

**Process: HFS**

1. Start at $s_0$ (initial state).
2. Initialize the open set with the initial state and its heuristic value.
3. Remove the node with the lowest heuristic value from the open set.
4. Expand the node and add all unvisited neighbors to the open set with their heuristic values.
5. Repeat steps 3 and 4 until the goal state is reached or the open set is empty.

**Example: HFS**

| Path | Open Set |
|------|----------|
|  | $\{A \mid 3\}$ |
| $A$ | $\{AB \mid 2,\ AC \mid 1,\ AD \mid 1\}$ |
| $AC$ | $\{AB \mid 2,\ AD \mid 1,\ ACE \mid 0\}$ |
| $ACE$ | $\{AB \mid 2,\ AD \mid 1\}$ |

**Process: $A^*$**

1. Start at $s_0$
2. Choose path in open set that gives lowest $\mathrm{esct}(p) = \mathrm{cst}(p) + \mathrm{hur}(p)$.
3. Expand and export children onto open set.
4. Repeat until goal state is reached.

**Example: A**$^*$

| Path | Open Set |
|------|----------|
|      | $\{A \mid 3\}$ |
| $A$ | $\{AB \mid 2+2,\ AC \mid 4+1,\ AD \mid 6+1\}$ |
| $AB$ | $\{AC \mid 5,\ AD \mid 7,\ ABC \mid 3+1,\ \cancel{ABA}\ \ \text{intra}\}$ |
| $ABC$ | $\{AC \mid 5,\ AD \mid 7,\ ABCD \mid 6+1,\ ABCE \mid 5+0\}$ |
| $AC$ | $\{AD \mid 7,\ ABCD \mid 7,\ ABCE \mid 5,\ ACD \mid 7+1,\ ACE \mid 6+0\}$ |
| $ABCE$ | $\{AD \mid 7,\ ABCD \mid 7,\ ACD \mid 8,\ ACE \mid 6\}$ |

**Example: IIA\***

**Example: WA\***

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

**Admissible:**
1. Given $\text{hur}(s)$ and search graph with $\text{cst}(s, a, \text{tr}(s, a))$ or $\text{rwd}(s, a, \text{tr}(s, a))$. If consistent, then it is admissible.
2. Check $\forall s \in \mathcal{G}$, $\text{hur}(s) = 0$. If not, then it is not admissible.
3. For each $s \in \mathcal{S}$, calculate $\text{hur}^*(s)$ (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 $\text{hur}(s) \leq \text{hur}^*(s)$ $\forall s \in \mathcal{S}$. 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 $\text{hur}(s)$ and search graph with $\text{cst}(s, a, \text{tr}(s, a))$ or $\text{rwd}(s, a, \text{tr}(s, a))$.
2. Check $\forall s \in \mathcal{G}$, $\text{hur}(s) = 0$. If not, then it is not consistent.
3. For each $s \in \mathcal{S}$, calculate $\text{hur}(s) - \text{hur}(\text{tr}(s, a))$.
   (a) check if it is $\leq \text{cst}(s, a, \text{tr}(s, a))$ or $\geq \text{rwd}(s, a, \text{tr}(s, a))$. 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.
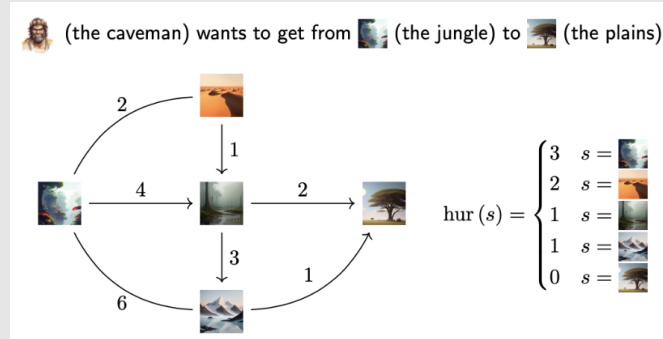
**Example:**



Figure 19: Jungle ($s^{(0)}$), Desert, Swamp, Mountain, Plains (Goal)

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

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

    (b) $\text{hur}(\text{Swamp}) - \text{hur}(\text{Plains}) = 1 - 0 = 1 \leq \text{cst}(\text{Swamp}, \cdot, \text{Plains}) = 2$

5. $s =$**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, $\text{cst}_{\text{rel}}(s)$.
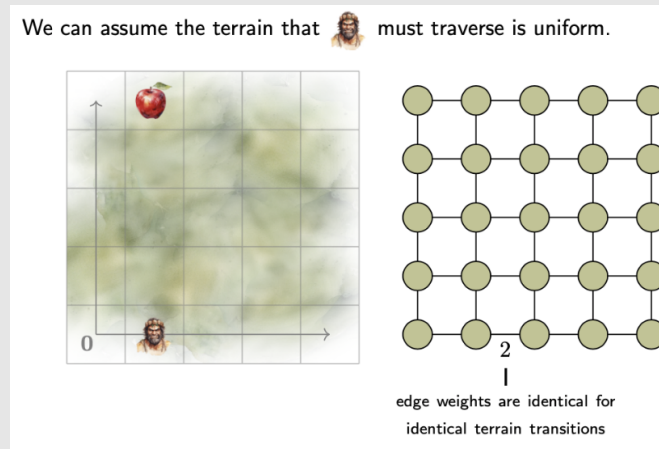3. HOW TO FIND THE COST OF THE OPTIMAL SOLUTION?

**Example:**



Figure 20

# 3 Constraint Satisfaction Problems

## 3.1 Setup of CSP

**Definition**: A **constraint satisfaction problem (CSP)** consists of:
- a set of **variables**, $\mathcal{V}$, where the domain of $V \in \mathcal{V}$ is $\text{dom}(V)$
- a set of **constraints**, $\mathcal{C}$, where the scope of $C \in \mathcal{C}$ is $\text{scp}(C) \subseteq \mathcal{V}$

## 3.2 Assignment

**Definition**: An **assignment** is a set of pairs, $\{(V, v)\}_{V \in \tilde{\mathcal{V}}}$, where $v \in \text{dom}(V)$, and $\tilde{\mathcal{V}} \subseteq \mathcal{V}$. It is **complete** if $\tilde{\mathcal{V}} = \mathcal{V}$, and **partial** otherwise.

## 3.3 Consistent

### 3.3.1 Complete Assignment

**Definition**: A complete assignment, $A$, is **consistent** if it satisfies every constraint $\mathcal{C}$ with $\text{scp}(\mathcal{C}) \subseteq \tilde{\mathcal{V}}$.

**Warning**: A solution to a CSP is any complete and consistent assignment.

### 3.3.2 Partial Assignment

**Definition**: A (possibly partial) assignment, $\{(V, v)\}_{V \in \tilde{\mathcal{V}}}$, is **consistent** if it satisfies every constraint, $C \in \mathcal{C}$ such that $\text{scp}(C) \subseteq \tilde{\mathcal{V}}$.

## 3.4 k-Consistent

**Definition**: A CSP is $k$-**consistent** if for any consistent assignment of $k - 1$ variables, $\{(V, v)\}_{V \in \tilde{\mathcal{V}}}$, and any $k^{\text{th}}$ variable, $V'$, there is a value, $v' \in \text{dom}(V')$, so the assignment, $\{(V, v)\}_{V \in \tilde{\mathcal{V}}} \cup \{(V', v')\}$ is consistent.

### 3.4.1 Edge/Arc Consistent

**Definition**: 2-consistent.

## 3.5   Setup of CSP

**Example**:

now wants to find food to meet his nutritional requirements:

|  |  | Supply | Carbs | Fat | Protein | Vitamins |
|---|---|---|---|---|---|---|
|  |  |  | **Nutrients (25 g)** | | | |
|  | 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 21

For our example, the variables could be:

$$\bullet \in \underbrace{\{0, 1, 2, 3, 4\}}_{\text{dom}(\bullet)} \qquad \bullet \in \underbrace{\{0, 1, 2, 3, 4, 5\}}_{\text{dom}(\bullet)}$$

$$\bullet \in \underbrace{\{0, 1, 2, 3, 4\}}_{\text{dom}(\bullet)} \qquad \bullet \in \underbrace{\{0, 1, 2, 3, 4, 5, 6\}}_{\text{dom}(\bullet)}$$

$$\bullet \in \underbrace{\{0, 1, 2, 3\}}_{\text{dom}(\bullet)}$$

Figure 22

For our example, the constraints could be:

$$\bullet : \underbrace{8 \leq 2\,\bullet + 2\,\bullet + 2\,\bullet + 3\,\bullet \leq 10}_{\text{scp}(\bullet) = \{\bullet, \bullet, \bullet, \bullet\}}$$

$$\bullet : \underbrace{3 \leq \bullet + \bullet \leq 4}_{\text{scp}(\bullet) = \{\bullet, \bullet\}}$$

$$\bullet : \underbrace{2 \leq \bullet + \bullet + 2\,\bullet \leq 5}_{\text{scp}(\bullet) = \{\bullet, \bullet, \bullet\}}$$

$$\bullet : \underbrace{1 \leq \bullet \leq 2}_{\text{scp}(\bullet) = \{\bullet\}}$$

Figure 23

> **Process: How to build a hyper-graph?**
> 1. Circle the variables that appear in constraint $C_i$ $\forall i$.

**Example:**

We can visualize the constraints using a hyper-graph.
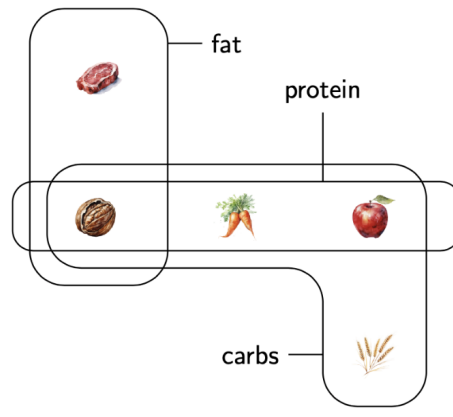


Figure 24

> **Process: How to build a path tree?**
> 1.

**Example:**

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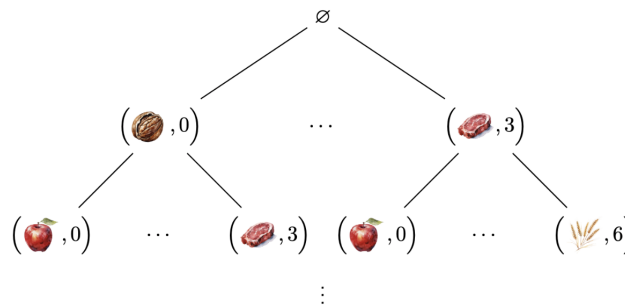


Figure 25
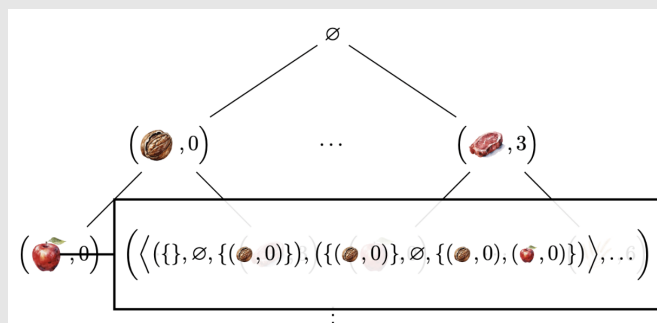


Figure 26

**Process:** **How to determine a solution to a CSP?**
　　1.

**Example**:

$$\left\{\left(\vcenter{\hbox{🥜}},2\right),\left(\vcenter{\hbox{🍎}},1\right),\left(\vcenter{\hbox{🥕}},1\right),\left(\vcenter{\hbox{🌾}},0\right),\left(\vcenter{\hbox{🥩}},1\right)\right\}$$

Figure 27

**Process: How to check $k$-Consistency?**
1. Given $\mathcal{V}$ w/ $\mathrm{dom}(V) = \{v_1, \ldots, v_{|\mathrm{dom}(V)|}\}$ $\forall V \in \mathcal{V}$ and $\mathcal{C}$ w/ $\mathrm{scp}(C) = \{V_1, \ldots, V_{|\mathrm{scp}(C)|}\}$ $\forall C \in \mathcal{C}$.
2. Remove all constraints that have $k+1$ or more variables.
3. For each $C \in \mathcal{C}$, do the following:
   (a) For each $V \in \mathrm{scp}(C)$, do the following:
       i. For each $v \in \mathrm{dom}(V)$, do the following:
          - Fix $V$ to $v$.
          - For the other $V \in \mathrm{scp}(C)$, check if the constraint is satisfied by trying all combinations (need only one).
          - **Key:** If there is one combination that doesn't satisfy the constraint, then the CSP is not $k$-consistent.
4. If all constraints are satisfied, then the CSP is $k$-consistent.

**Process: How to Enforce $k$-Consistency?**
1. Given $\mathcal{V}$ w/ $\mathrm{dom}(V) = \{v_1, \ldots, v_{|\mathrm{dom}(V)|}\}$ $\forall V \in \mathcal{V}$ and $\mathcal{C}$ w/ $\mathrm{scp}(C) = \{V_1, \ldots, V_{|\mathrm{scp}(C)|}\}$ $\forall C \in \mathcal{C}$.
2. Remove all constraints that have $k+1$ or more variables.
3. **Pre-pruning:** For each remaining $C \in \mathcal{C}$, do the following:
   (a) For each $V \in \mathrm{scp}(C)$, do the following:
       i. For each $v \in \mathrm{dom}(V)$, do the following:
          - Fix $V$ to $v$.
          - For the other $V \in \mathrm{scp}(C)$, check if the constraint is satisfied by trying all combinations (need only one).
          - **Key:** If the constraint is not satisfied, then remove the value from $\mathrm{dom}(V)$.
4. If you had to remove any values from $\mathrm{dom}(V)$, then check with the other constraints.
5. **Pruning:** Every constraint is satisfied.

**Example**:



Figure 28



Figure 29: Pre-pruning. Since only one constraint, it is also pruning.

**Example**: Different ways to formulate the CSP problem.
- How can you formualte 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 contans a path.
    - How does that match the example on slide 10. It does.
- Why is formulalting 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.

# Learning Problems

# 4　PAC Learning

**Definition**: Assume that there is some (unknown) relationship,

$$f : \mathcal{X} \to \mathcal{Y} \text{ s.t. } x \mapsto_f y$$

- $\mathcal{X}$: Input Space
- $\mathcal{Y}$: Output Space (i.e. Information we desire about input)

Find $h : \mathcal{X} \to \mathcal{Y}$ (hypothesis) s.t. $h \approx f$, given some data about $f$:

$$\mathcal{D} = \left\{ \left( x^{(i)}, y_i \right), x^{(i)} \in \mathcal{X}, y_i = f\left( x^{(i)} \right) \in \mathcal{Y}, i = 1 \dots N \right\}$$

- $\text{in}(\mathcal{D}) = \{ x \text{ s.t. } (x, y) \in \mathcal{D} \}$
- $\text{out}(\mathcal{D}) = \{ y \text{ s.t. } (x, y) \in \mathcal{D} \}$

## 4.1　Classification vs. Regression Problems

**Definition**:
- **Classification Problems:** $\mathcal{X} \subseteq \mathbb{R}^M$ and $\mathcal{Y} \subseteq \mathbb{N}$
- **Regression Problems:** $\mathcal{X} \subseteq \mathbb{R}^M$ and $\mathcal{Y} \subseteq \mathbb{R}$

## 4.2　Feature Spaces

**Definition**: It is often easier to learn relationships from high-level features (instead of the raw input).

## 4.3　Feasibility of Learning

**Motivation**: More than one function (hypothesis) may be consistent with the data.

**Notes**: So it may appear that finding the correct one should be impossible.

### 4.3.1　Probably Approximately Correct (PAC) Estimations

**Example**: Take $N$ i.i.d. samples (i.e. take out a ball from an urn, record its color, and put it back in).
- $\nu \to \mu$ (empirical distribution $\to$ true distribution) as $N \to \infty$

### 4.3.2　Hoeffding's Inequality

**Definition**: Let $\mu$ denote the probability of an event, and $\nu$ denote its relative frequency in a sample size $N$. Then, for any $\epsilon > 0$,

$$P(|\nu - \mu| > \epsilon) \le 2e^{-2\epsilon^2 N} \tag{1}$$

- $\nu$: Relative frequency in the sample (known)
- $\mu$: Probabillity of drawing a blue ball (unknown)
- $N \to \infty$: $\nu \to \mu$
- $\epsilon$: How close we want $\nu$ to be to $\mu$
- $\epsilon \to 0$: Probability will be 1
- $\epsilon \to \infty$: $\nu \to \mu$
- $\mu \overset{?}{\approx} \nu$: $\mu$ is probably approximately equal to $nu$.

**Warning**: We can approximate the true distribution with high probability by taking a large enough sample size, NOT guaranteeing that we can find the true distribution.
- Don't need to know where this theorem comes from.

Consider determining the class of a randomly chosen target point. If we ask a K-ary question about the points in $\mathcal{D}$

### 4.3.3  PAC Learning

# 5   Decision Trees

## 5.1   Decision Trees

**Motivation**: A simple model used for classification problems.

We want to find the minimum # of condition vertices (or questions) needed to "sufficiently discriminate" (identify the class of every point in $\mathcal{D}$).

### 5.1.1   Structure

**Definition**: Each vertex in a decision tree is either:
1. A **condition vertex**: a vertex that sorts points based on a question.
2. A **decision vertex**: a vertex that assigns all points a specific class.

**Warning**:
- More condition vertices improve discrimination.
- Less condition vertices improve generalization.

## 5.2   Building a Decision Tree: The General Case

**Motivation**: Suppose points do not necessarily belong to a unique class.

In the context of decision trees:
- $X$ is the class of a randomly chosen target point.
- $Y$ is the answer to a $K$-ary question for $X$.

Maximize $IG(X|Y)$ (i.e. choose the question to maximize the information gained).

### 5.2.1   Entropy, Conditional Entropy, and Information Gain

**Definition**: The **entropy** of a random variable $X$ (in $K$-its) is defined as

$$H(X) = -\sum_{\forall x \in X} p_X(x) \log_K(p_X(x)).$$

The **conditional entropy** of a random variable, $X$, given a random variable $Y$, is

$$H(X|Y) = -\sum_{\forall y \in Y} \sum_{\forall x \in X} p_{X|Y}(x|y) \log_K(p_{X|Y}(x|y)).$$

The **information gain** from $Y$ is:
$$IG(X|Y) = H(X) - H(X|Y).$$

**Warning**:
- There are $\infty$ many potential questions, but there are only finite many ways to split the dataset.

**Process:**
1. Calculate $H(X)$ (i.e. entropy before the split).
2. Calculate $H(X|Y)$ (i.e. entropy after the split).
    (a) Calculate entropy for each subset of $X$ based on the question, $Y$.
    (b) Calculate the weighted average of the entropies.
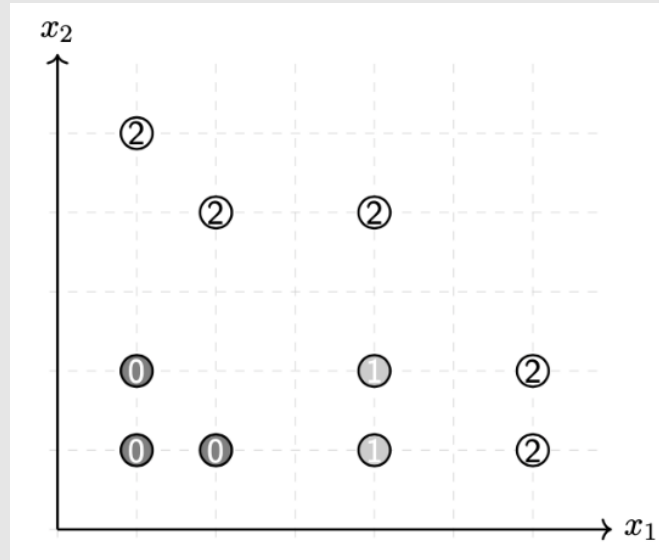3. Calculate $IG(X|Y) = H(X) - H(X|Y)$.

**Example:**



Figure 30

**Example: 2-Ary Question**
1. **Given:** $X = \{0, 1, 2\}$, $Y = \begin{cases} 1, & \text{if } x_1 \leq 3 \quad \text{(Yes)} \\ 0, & \text{if } x_1 > 3 \quad \text{(No)} \end{cases}$,
2. **Problem:** $IG(X|Y) =$?
3. **Solution:**
    (a) Entropy before the split: $H(X) = \dfrac{3}{10} \log_2\left(\dfrac{10}{3}\right) + \dfrac{2}{10} \log_2\left(\dfrac{10}{2}\right) + \dfrac{5}{10} \log_2\left(\dfrac{10}{5}\right)$
    (b) Entropy after the split:
        i. $H(X_{\text{left}}) = \dfrac{3}{5} \log_2\left(\dfrac{5}{3}\right) + \dfrac{2}{5} \log_2\left(\dfrac{5}{2}\right)$
        ii. $H(X_{\text{right}}) = \dfrac{2}{5} \log_2\left(\dfrac{5}{2}\right) + \dfrac{3}{5} \log_2\left(\dfrac{5}{3}\right)$.
        iii. Weighted Avg. Entropy: $H(X|Y) = \dfrac{5}{10} H(X_{\text{left}}) + \dfrac{5}{10} H(X_{\text{right}})$
    (c) $IG(X|Y) = H(X) - H(X|Y)$

**Example: 2-Ary Question**
1. **Given:** $X = \{0, 1, 2\}$, $Y = \begin{cases} 1, & \text{if } x_2 \leq 3 \quad \text{(Yes)} \\ 0, & \text{if } x_2 > 3 \quad \text{(No)} \end{cases}$,
2. **Problem:** $IG(X|Y) =$?
3. **Solution:**
    (a) Entropy before the split: $H(X) = \dfrac{3}{10} \log_2\left(\dfrac{10}{3}\right) + \dfrac{2}{10} \log_2\left(\dfrac{10}{2}\right) + \dfrac{5}{10} \log_2\left(\dfrac{10}{5}\right)$
    (b) Entropy after the split:

i. $H(X_{\text{top}}) = \frac{3}{3} \log_2\left(\frac{3}{3}\right)$

ii. $H(X_{\text{bottom}}) = \frac{3}{5} \log_2\left(\frac{5}{3}\right) + \frac{2}{5} \log_2\left(\frac{5}{2}\right) + \frac{2}{5} \log_2\left(\frac{5}{2}\right).$

iii. Weighted Avg. Entropy: $H(X|Y) = \frac{3}{10} H(X_{\text{top}}) + \frac{7}{10} H(X_{\text{bottom}})$

(c) $IG(X|Y) = H(X) - H(X|Y)$

**Example: 3-Ary Question**

1. **Given:** $X = \{0, 1, 2\}$, $Y = \begin{cases} 1, & \text{if } x_1 \leq 3 \text{ and } x_2 \leq 3 \\ 2, & \text{if } x_1 \leq 3 \text{ and } x_2 > 3 \\ 3, & \text{if } x_1 > 3 \end{cases}$

2. **Problem:** $IG(X|Y) = ?$

3. **Solution:**

(a) Entropy before the split: $H(X) = \frac{3}{10} \log_2\left(\frac{10}{3}\right) + \frac{2}{10} \log_2\left(\frac{10}{2}\right) + \frac{5}{10} \log_2\left(\frac{10}{5}\right)$

(b) Entropy after the split:

i. $H(X_1) = \frac{3}{3} \log_2\left(\frac{3}{3}\right)$

ii. $H(X_2) = \frac{2}{2} \log_2\left(\frac{2}{2}\right)$

iii. $H(X_3) = \frac{2}{5} \log_2\left(\frac{5}{2}\right) + \frac{3}{5} \log_2\left(\frac{5}{3}\right)$

iv. $H(X|Y) = \frac{3}{10} H(X_1) + \frac{2}{10} H(X_2) + \frac{5}{10} H(X_3)$

(c) $IG(X|Y) = H(X) - H(X|Y)$

**Example: Decision Tree**

1. **Given:** $X = \{0, 1, 2\}$

2. **Problem:** Draw a decision tree using binary conditions of the form, $x_i \leq k$, where $i \in \{1, 2\}$ and $k \in \mathbb{Z}$, that maximizes the information gained at each level.

3. **Solution:**

(a)