ROB311 Quiz 1

Hanhee Lee

February 6, 2025

Contents

1 Prologue

Summary:
- 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)
- **Move:** Any tuple, (s, a), where $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
- **Transition:** 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
- **Reward:** $\mathrm{rwd}_i(s, a, s')$ is agent i's reward for the transition, (s, a, s')
- **Path:** Any sequence of transitions of the form.

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

- **Objective:** 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:

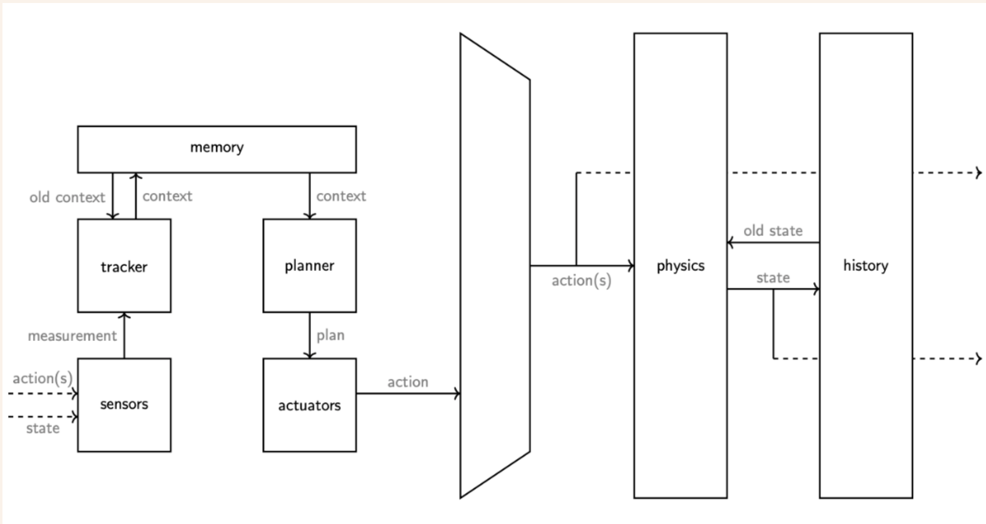


Figure 1: Components of a Robotic System (Words)

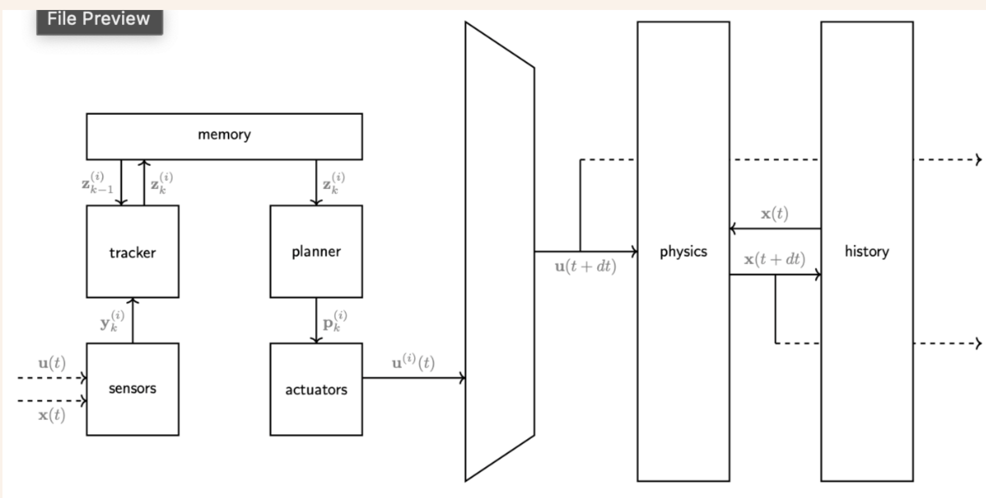


Figure 2: Components of a Robotic System (Math)

1.2.1 Overview (Robots, the Environment)

Definition:

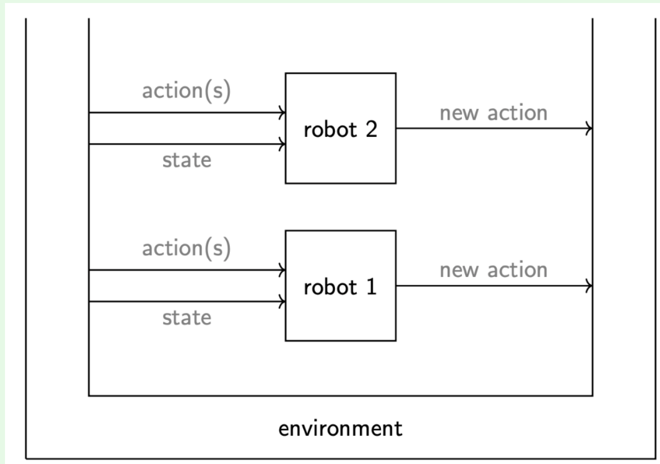


Figure 3: Overview (Robots, the Environment)

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

1.2.2 Robot (Sensors, Actuators, the Brain)

Definition:

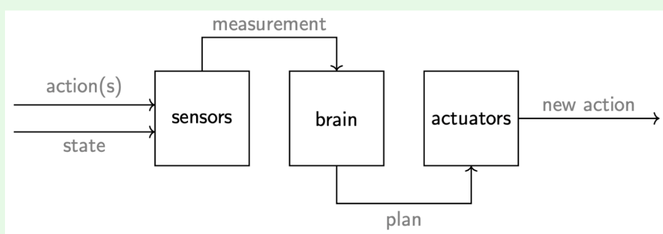


Figure 4: Robot (Sensors, Actuators, the Brain)

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:

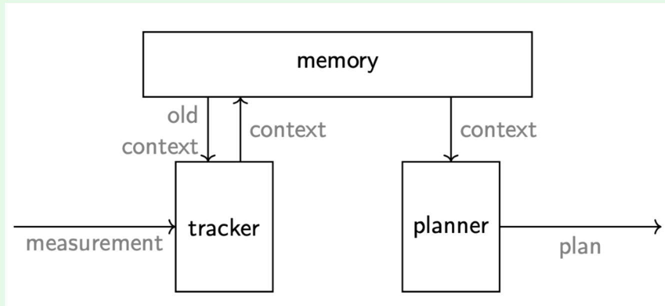


Figure 5: Brain (Tracker, Planner, Memory)

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:

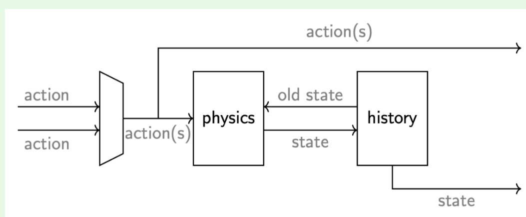


Figure 6: Environment (Physics, State)

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 $T^{(i)}$:

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

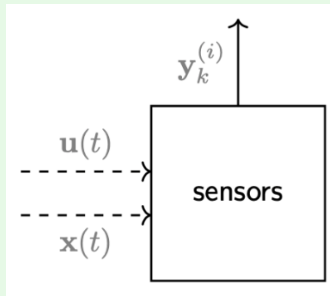


Figure 7: Sensing

1.3.2 Tracking

Definition: Track (update) the context:

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

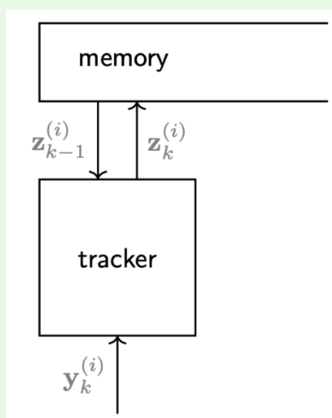


Figure 8: Tracking

1.3.3 Planning

Definition: Make a plan:

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

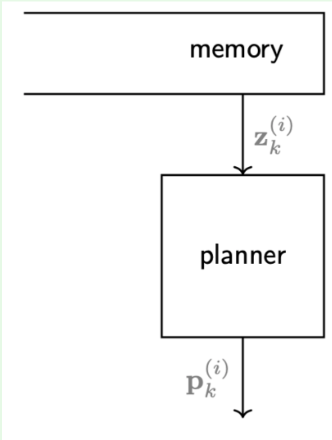


Figure 9: Planning

1.3.4 Acting

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

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

Execute the plan:

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

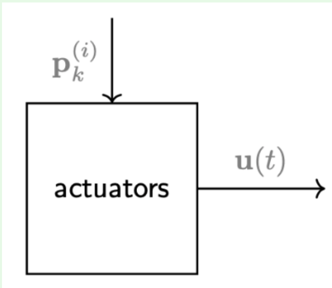


Figure 10: Acting

1.3.5 Simulating

Definition: Simulate the environment's response:

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

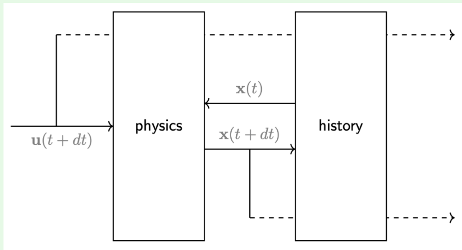


Figure 11: Simulating

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, ϵ: 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, $\text{ecst}(p) = \text{hur}(p)$						
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, $\text{ecst}(p) = \text{cst}(p) + \text{hur}(p)$						
IIA*	always	always	always	always	b^{l^*}	bl^*
• Same as A* but with iterative inflating on ecst.						
WA*	-	-	-	-	-	-
• Same as A* but $\text{ecst}(s) = w\text{cst}(s) + (1-w)\text{hur}(s)$ 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`.

Warning: A NULL solution is not the same as $p = \langle\rangle$ (an empty solution w/ $s^{(0)} \in \mathcal{G}$).

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

Algorithm: All search algorithms follow the template below:

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

- $\langle\rangle$: Empty path, 0: Cost of empty path.

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

2.6 Modifications to Search Algorithms

2.6.1 Depth-Limiting

Algorithm:

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

2.6.2 Iterative Deepening

Algorithm:

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

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

2.6.3 Cost-Limiting

Algorithm:

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

2.6.4 Iterative-Inflating

Algorithm:

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

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

2.6.5 Intra-Path Cycle Checking

Algorithm:

```
procedure SEARCH(𝒪):
    if 𝒪 = ∅ then
        return NULL
    n ← REMOVE(𝒪)
    if dst(n) ∈ 𝒢 then
        return n
    for n′ ∈ chl(n) do                         ▷ "expand" n and "export" its children
        if not CYCLIC(n′) then                     ▷ unless the child is cyclic
            𝒪 ← 𝒪 ∪ {n′}
    SEARCH(𝒪)
```

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

2.6.6 Inter-Path Cycle Checking

Algorithm:

```
procedure SEARCH(𝒪, 𝒞):
    if 𝒪 = ∅ then
        return NULL
    n ← REMOVE(𝒪)
    𝒞 ← 𝒞 ∪ {n}                                 ▷ add n to the closed set
    if dst(n) ∈ 𝒢 then
        return n
    for n′ ∈ chl(n) do                         ▷ "expand" n and "export" its children
        if n′ ∉ 𝒞 then                    ▷ unless the child's destination is closed
            𝒪 ← 𝒪 ∪ {n′}
    SEARCH(𝒪, 𝒞)
```

and then call the algorithm as follows:

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

2.7 Informed Search Algorithms

2.7.1 Estimated Cost

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

2.7.2 Admissible

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

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

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

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

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

Theorem: If a heuristic, $\text{hur}(\cdot)$, is consistent, then it is also admissible.

2.7.4 Domination

Definition: If hur_1 and hur_2 are admissible, then:
- hur_1 **strongly dominates** hur_2 if for all $s \in \mathcal{S} \setminus \mathcal{G}$:

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

- hur_1 **weakly dominates** 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 Designing Heuristics via Problem Relaxation

Definition: Let $\text{hur}^*_{\text{ori}}$ be the perfect heuristic for a search problem, and $\text{cst}^*_{\text{rel}}$ be the optimal cost for a relaxed version of the problem. Then

$$\text{cst}^*_{\text{rel}}(s) \leq \text{hur}^*_{\text{ori}}(s) \text{ for all } s \in \mathcal{S}.$$

2.7.6 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 hur_k is consistent:

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

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

2.7.7 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

2.8.1 How to setup a search problem?

Process:

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 $\mathrm{tr}(\cdot,\cdot)$)
     - $\mathrm{tr}(\cdot,\cdot)$: transition function
   - $\mathrm{cst}(\cdot,\cdot,\cdot)$: cost function (defined by edge weights)

Example:

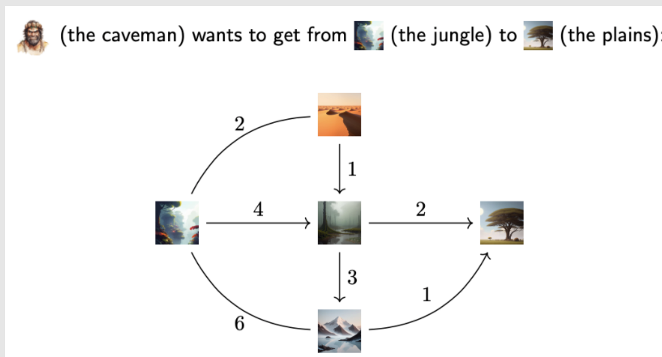


Figure 12

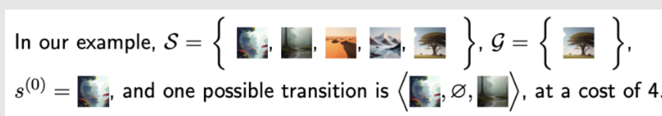


Figure 13

Example:

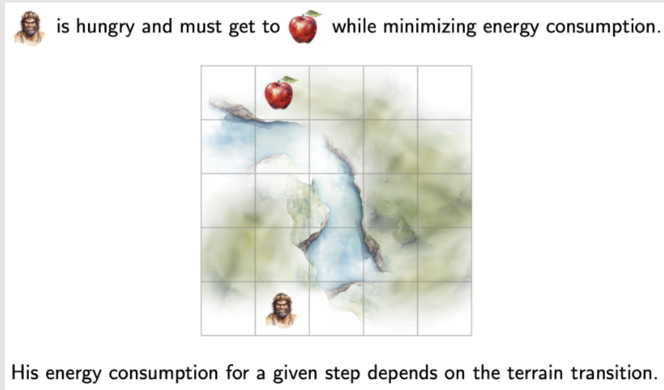


Figure 14

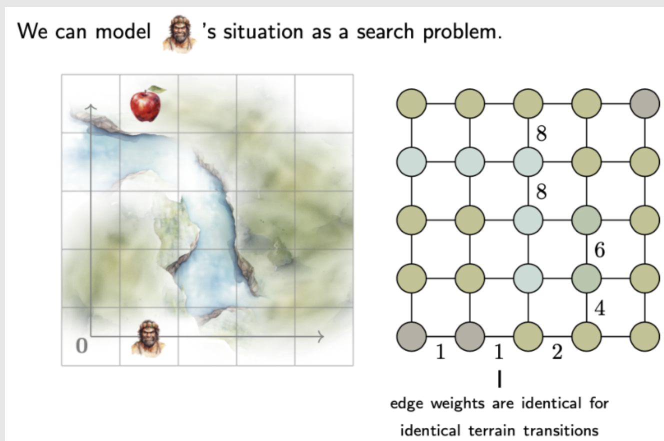


Figure 15

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

2.8.2 How to setup a path tree?

Process:
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:

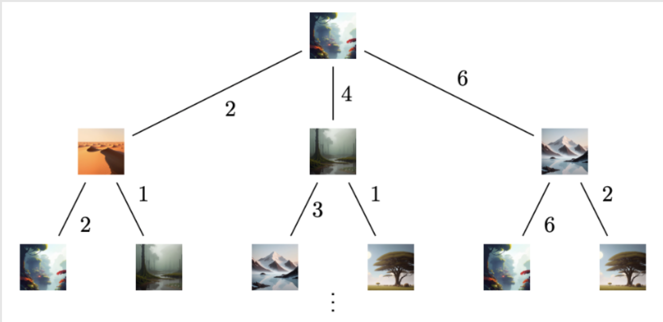


Figure 16

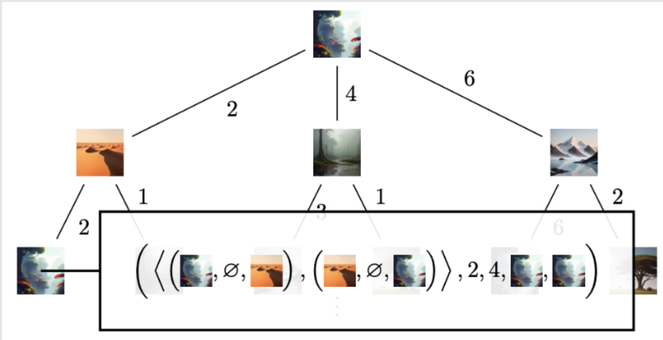


Figure 17

2.8.3 When to use each algorithm?

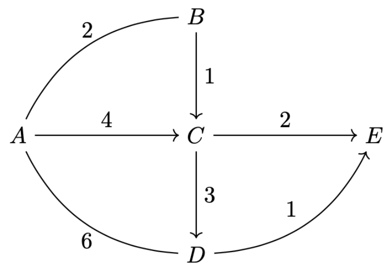
Process:

1. Do we have a heuristic?
    - **Yes:** Use informed search algorithms.
    - **No:** Use uninformed search algorithms.
2. Are path costs non-uniform?
    - **Yes:** Eliminate BFS.
    - **No:** Eliminate CFS, $A*$
3.
4. Is the search space finite or infinite?
    - **Finite:** Use any algorithm.
    - **Infinite:** Use BFS, IDDFS, CFS, or A*.
5. Do we need to guarantee finding a solution (completeness)?
    - **Yes:** Use BFS, IDDFS, IIA*, CFS (if $\epsilon > 0$).
    - **No:** Use DFS, HFS, WA*
6. Find properties needed for the problem and match them to the characteristics of the algorithm.
7. Choose the algorithm that best matches the properties.
    - **BFS:** Need shortest path in an unweighted graph.
    - **DFS:** Explore a deep path quickly, and completeness is not needed.
    - **IDDFS:** Want completeness of BFS but with the complexity of DFS.
    - **CFS:** Need the least-cost path in a weighted graph.
    - **HFS:**
    - **A*:**
    - **IIA*:**
    - **WA*:**

Example:

2.8.4 Tracing Search Algorithms

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 as **current node**
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 add it to the path.
4. Choose the least-recently expanded node from the open set as the **current node**.
5. Repeat steps 2 and 4 until the goal state is reached or the open set is empty.

Example: BFS

Path	**Open Set**
	$\{A\}$
A	$\{AB, AC, AD\}$
AB	$\{AC, AD, ABA, ABC\}$
AC	$\{AD, ABA, ABC, ACD, ACE\}$
AD	$\{ABA, ABC, ACD, ACE, ADA, ADE\}$
ABA	$\{ABC, ACD, ACE, ADA, ADE, ABAB, ABAC, ABAD\}$
ABC	$\{ACD, ACE, ADA, ADE, ABAB, ABAC, ABAD, ABCD, ABCE\}$
ACD	$\{ACE, ADA, ADE, ABAB, ABAC, ABAD, ABCD, ABCE, ACDA, ACDE\}$
ACE	$\{ADA, ADE, ABAB, ABAC, ABAD, ABCD, ABCE, ACDA, ACDE\}$

Intra:

Path	**Open Set**
	$\{A\}$
A	$\{AB, AC, AD\}$
AB	$\{AC, AD, ABC\}$
AC	$\{AD, ABC, ACD, ACE\}$
AD	$\{ABC, ACD, ACE, ADE\}$
ABC	$\{ACD, ACE, ADE, ABCD, ABCE\}$
ACD	$\{ACE, ADE, ABCD, ABCE, ACDE\}$
ACE	$\{ADE, ABCD, ABCE, ACDE\}$

Process: DFS

1. Start at s_0 as **current node**
2. Expand all neighboring nodes of the **current node** and add them to the open set (stack).
3. Remove the **current node** from the open set and add it to the path.
4. Choose the most-recently expanded node from the open set as the **current node**.
5. Repeat steps 2 and 4 until the goal state is reached or the open set is empty.

Example: DFS

Path	**Open Set**
	$\{A\}$
A	$\{AB, AC, AD\}$
AD	$\{AB, AC, ADA, ADE\}$
ADE	$\{AB, AC, ADA, ADE\}$

Intra:

Path	**Open Set**
	$\{A\}$
A	$\{AB, AC, AD\}$
AD	$\{AB, AC, ADE\}$
ADE	$\{AB, AC, ADE\}$

Inter:

Process: IDDFS

1. Start with a depth limit of 0.
2. Perform DFS up to the current depth limit.
3. If the goal state is not reached, increment the depth limit based on given fcn 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	$\{AB, AC, AD\}$
1	AD	$\{AB, AC\}$
1	AC	$\{AB\}$
1	AB	$\{\}$
2	AB	$\{ABA, ABC\}$
2	ABC	$\{ABA\}$
2	ABA	$\{\}$
3	ABA	$\{ABAB, ABAC, ABAD\}$
3	$ABAB$	$\{ABAC, ABAD, ABABC\}$

Process: CFS

1. Start at s_0 as **current node**
2. Expand all neighboring nodes of the **current node** and add them to the open set (priority queue).
3. Remove the **current node** from the open set and add it to the path.
4. Choose the cheapest expanded node from the open set as the **current node**.
5. Repeat steps 2 and 4 until the goal state is reached or the open set is empty.

Example: CFS

No Inter-Path Cycle Checking, No Intra-Path Cycle Checking

Path	Open Set
	$\{A \mid 0\}$
A	$\{AB \mid 2,\ AC \mid 4,\ AD \mid 6\}$
AB	$\{AC \mid 4,\ AD \mid 6,\ ABC \mid 3,\ ABA \mid 4\}$
ABC	$\{AC \mid 4,\ AD \mid 6,\ ABA \mid 4,\ ABCE \mid 5,\ ABCD \mid 6\}$
AC	$\{AD \mid 6,\ ABA \mid 4,\ ABCE \mid 5,\ ABCD \mid 6,\ ACD \mid 7,\ ACE \mid 6\}$
ABA	$\{AD \mid 6,\ ABCE \mid 5,\ ABCD \mid 6,\ ACD \mid 7,\ ACE \mid 6,\ ABAB \mid 6,\ ABAC \mid 8,\ ABAD \mid 10\}$
$ABCE$	$\{AC \mid 4,\ ABA \mid 4,\ AD \mid 6,\ ABCD \mid 6\}$

Intra-Path Cycle Checking:

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

- **Inter-Path Cycle Checking:** Doesn't affect the algorithm in this instance.

Warning:

- How to perform shortcut?

Process: HFS

1. Start at s_0 as **current node**
2. Expand all neighboring nodes of the **current node** and add them to the open set (priority queue).
3. Remove the **current node** from the open set and add it to the path.
4. Choose the lowest heuristic value expanded node from the open set as the **current node**.
5. Repeat steps 2 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	$\{AB \mid 2,\ AD \mid 1\}$

- **Inter/Intra-Path Cycle Checking:** Doesn't affect the algorithm in this instance.

Process: A*

1. Start at s_0 as **current node**
2. Expand all neighboring nodes of the **current node** and add them to the open set (priority queue).
3. Remove the **current node** from the open set and add it to the path.
4. Choose the lowest $\text{esct}(p) = \text{cst}(p) + \text{hur}(p)$ expanded node from the open set as the **current node**.
5. Repeat steps 2 and 4 until the goal state is reached or the open set is empty.

Example: A*

No Inter-Path Cycle Checking, No Intra-Path Cycle Checking

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

Intra-Path Cycle Checking:

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	$\{AD \mid 7,\ ABCD \mid 7,\ ABCE \mid 5,\ ACD \mid 7+1,\ ACE \mid 6+0\}$
$ABCE$	$\{AD \mid 7,\ ABCD \mid 7,\ ACD \mid 8,\ ACE \mid 6\}$

- **Inter-Path Cycle Checking:** Doesn't affect the algorithm in this instance.

Process: IIA*

1. Start with a cost limit of 0.
2. Perform A* up to the current cost limit.
3. If the goal state is not reached, increment the cost limit based on given fcn and repeat step 2.
4. Continue until the goal state is found or all nodes are explored.

Example: IIA*
Inter-Path Cycle Checking, Intra-Path Cycle Checking

Cost	Path	Open Set
0	$\langle\rangle$	$\{A \mid 0+3\}$
1	$\langle\rangle$	$\{A \mid 3\}$
2	$\langle\rangle$	$\{A \mid 3\}$
3	$\langle\rangle$	$\{A \mid 3\}$
3	A	$\{AB \mid 2+2,\ AC \mid 4+1,\ AD \mid 6+1\}$
4	AB	$\{AC \mid 5,\ AD \mid 7,\ ABC \mid 3+1\}$
4	ABC	$\{AC \mid 5,\ AD \mid 7,\ ABCD \mid 6+1,\ ABCE \mid 5+0\}$
5	AC	$\{AD \mid 7,\ ABCD \mid 7,\ ABCE \mid 5,\ ACD \mid 7+1,\ ACE \mid 6+0\}$
5	$ABCE$	$\{AD \mid 7,\ ABCD \mid 7,\ ACD \mid 8,\ ACE \mid 6\}$

Process: WA*

1. Start at s_0 as **current node**
2. Expand all neighboring nodes of the **current node** and add them to the open set (priority queue).
3. Remove the **current node** from the open set and add it to the path.
4. Choose the lowest $\text{esct}(p) = w \cdot \text{cst}(p) + (1-w) \cdot \text{hur}(p)$ expanded node from the open set as the **current node**.
5. Repeat steps 2 and 4 until the goal state is reached or the open set is empty.

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

Admissible:

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

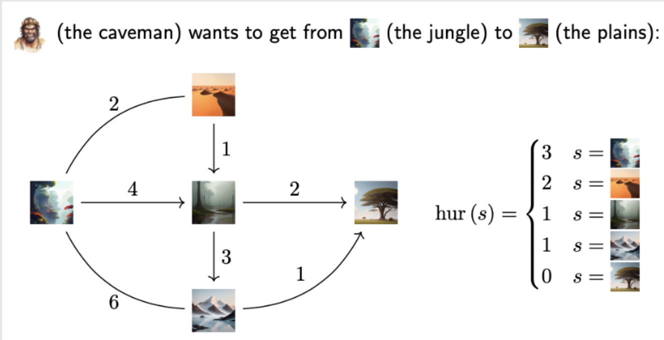


Figure 19: Jungle ($s^{(0)}$), Desert, Swamp, Mountain, Plains (Goal)

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:

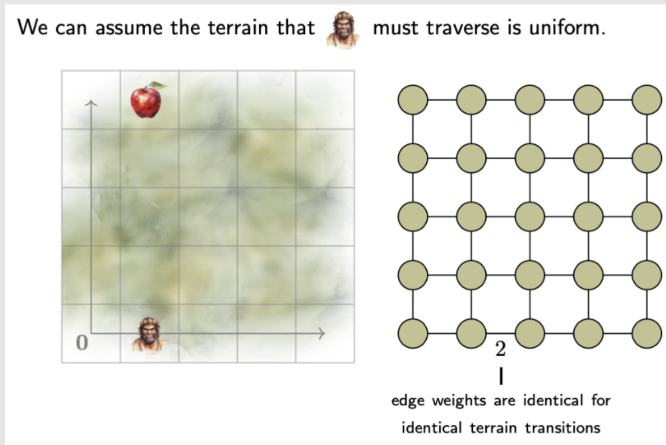


Figure 20

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 $\text{scp}(C) \subseteq \tilde{\mathcal{V}}$.

3.3.3 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^{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.

3.3.4 Edge/Arc Consistent

Definition: 2-consistent.

3.4 Constraint Satisfaction Algorithms

3.4.1 Main

Algorithm:

3.4.2 Satisfy

Algorithm:

3.4.3 Enforce: Enforcing k-Consistency

Algorithm:

3.4.4 EnforceVar: Enforcing k-Consistency

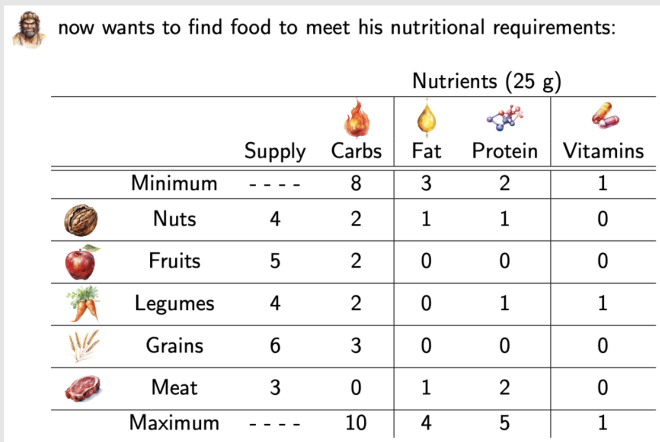
Algorithm:

3.5 Setup of CSP

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.

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

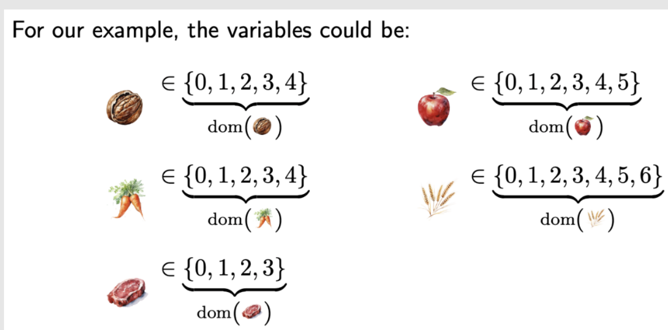


Figure 22

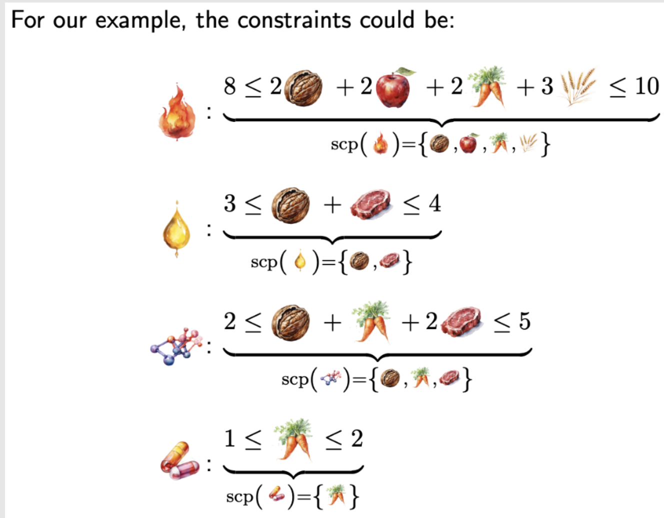


Figure 23

Process: How to build a hyper-graph?

1. Circle the variables that appear in constraint C_i $\forall i$.

Example:

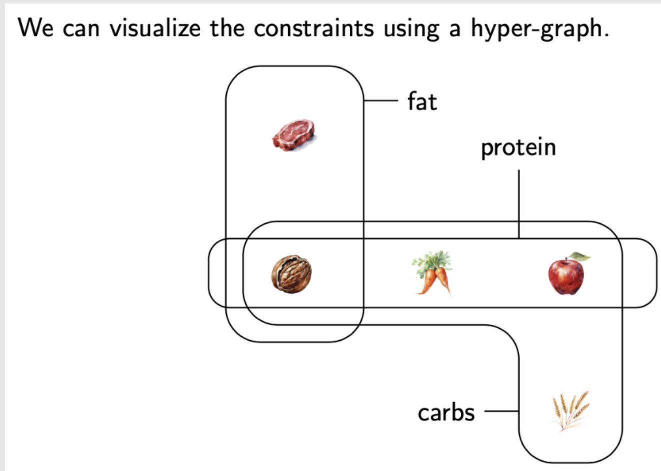


Figure 24

Process: How to build a path tree?

1.

Example:

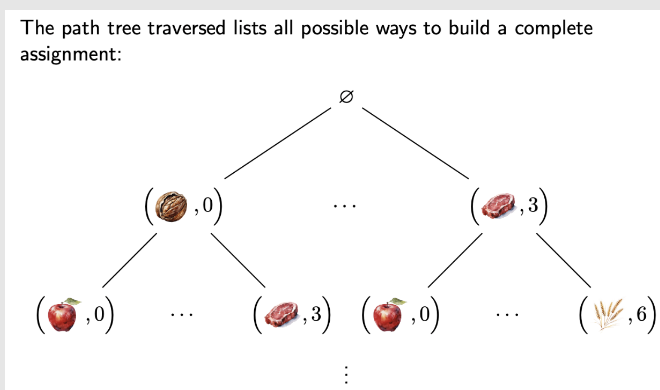


Figure 25

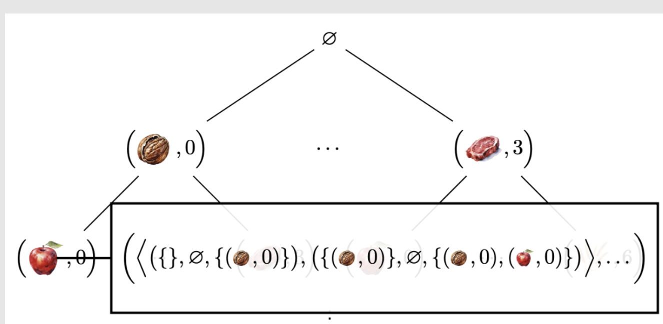


Figure 26

Process: How to determine a solution to a CSP?

1.

Example:

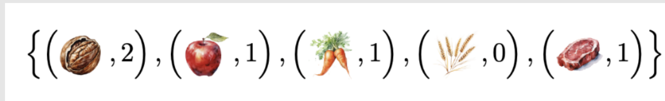


Figure 27

Process: How to check k-Consistency?

1. Given $\mathcal{V}$ w/ $\text{dom}(V) = \{v_1, \dots, v_{|\text{dom}(V)|}\}$ $\forall V \in \mathcal{V}$ and $\mathcal{C}$ w/ $\text{scp}(C) = \{V_1, \dots, V_{|\text{scp}(C)|}\}$ $\forall C \in \mathcal{C}$.
2. Remove all constraints that have $k+1$ or more variables.
3. For each $C \in \mathcal{C}$, do the following:
   (a) For each $V \in \text{scp}(C)$, do the following:
      i. For each $v \in \text{dom}(V)$, do the following:
      - Fix V to v for $k-1$ variables.
      - For the last variable V, find a value in $\text{dom}(V)$ that satisfies all constraints.
      - **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/ $\text{dom}(V) = \{v_1, \dots, v_{|\text{dom}(V)|}\}$ $\forall V \in \mathcal{V}$ and $\mathcal{C}$ w/ $\text{scp}(C) = \{V_1, \dots, V_{|\text{scp}(C)|}\}$ $\forall C \in \mathcal{C}$.
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 $V \in \text{scp}(C)$, do the following:
      i. For each $v \in \text{dom}(V)$, do the following:
      - Fix V to v.
      - For the other $V \in \text{scp}(C)$, 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 $\text{dom}(V)$.
4. If you had to remove any values from $\text{dom}(V)$, then check with the other constraints.
5. **Pruning:** Every constraint is satisfied.

Warning: Can think of checking as picking $k-1$ variables, then choosing any value for the k^{th} variable that satisfies all constraints. While enforcing is fixing a variable to a value, then checking if there is a combination for the other variables that satisfies all constraints.

Warning: Enforcing k-consistency is enforcing $k-1, \dots, 1$-consistency.

Example:

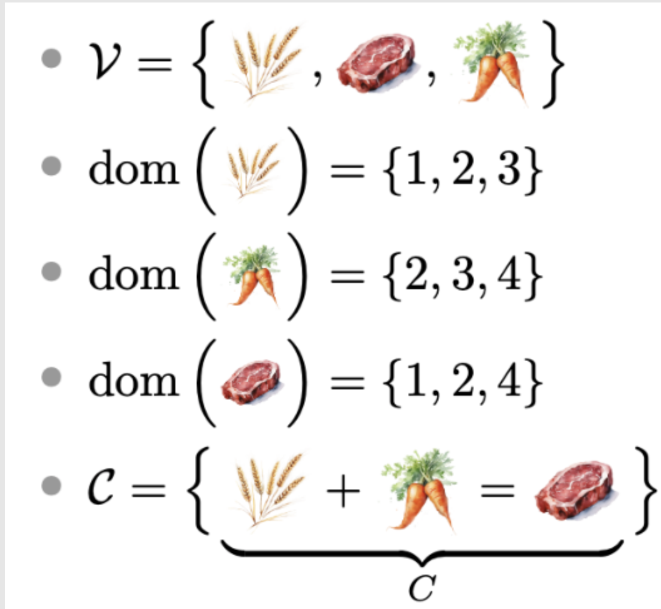


Figure 28

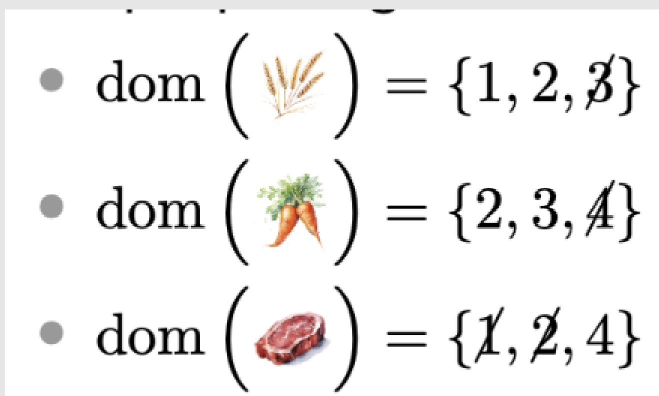


Figure 29: Pre-pruning. Since only one constraint, it is also pruning.

Example:

1. **Given:** Consider a CSP in which $\mathcal{V} = \{A, B, C, D, E\}$, where:

$$\begin{aligned} \text{dom}(A) &= \{0, 1, 2, 3, 4\} \\ \text{dom}(B) &= \{0, 1, 2, 3, 4\} \\ \text{dom}(C) &= \{0, 1, 2, 3\} \\ \text{dom}(D) &= \{0, 1, 2, 3, 4, 5\} \\ \text{dom}(E) &= \{0, 1, 2, 3, 4, 5, 6\} \end{aligned}$$

   and $\mathcal{C} = \{C_1, C_2, C_3, C_4\}$, where:

$$\begin{aligned} &C_1 : 8 \leq 2a + 2b + 2d + 3e \leq 10 \\ &C_2 : 3 \leq a + c \leq 4 \\ &C_3 : 2 \leq a + b + 2c \leq 5 \\ &C_4 : 1 \leq b \leq 2 \end{aligned}$$

2. **Problem:** Solve the following CSP using $k = 4$ consistency. Pre-prune the domains using $k = 4$ consistency. Assign variables in alphabetical order and values in numerical order.

Example: 4-Consistency

Fixed Value	Satisfactory Combination?
	$C_4 : 1 \leq b \leq 2$
$b = 0$, $b = 1$, $b = 2$, $b = 3$, $b = 4$	No, Yes, Yes, No, No

- $\text{dom}(A) = \{0, 1, 2, 3, 4\}$
- $\text{dom}(B) = \{\cancel{0}, 1, 2, \cancel{3}, \cancel{4}\}$
- $\text{dom}(C) = \{0, 1, 2, 3\}$
- $\text{dom}(D) = \{0, 1, 2, 3, 4, 5\}$
- $\text{dom}(E) = \{0, 1, 2, 3, 4, 5, 6\}$

Fixed Value	Satisfactory Combination?
	$C_2 : 3 \leq a + c \leq 4$
$a = 0$, $a = 1$, $a = 2$, $a = 3$, $a = 4$	Yes, Yes, Yes, Yes, Yes
$c = 0$, $c = 1$, $c = 2$, $c = 3$	Yes, Yes, Yes, Yes

- $\text{dom}(A) = \{0, 1, 2, 3, 4\}$
- $\text{dom}(B) = \{\cancel{0}, 1, 2, \cancel{3}, \cancel{4}\}$
- $\text{dom}(C) = \{0, 1, 2, 3\}$
- $\text{dom}(D) = \{0, 1, 2, 3, 4, 5\}$
- $\text{dom}(E) = \{0, 1, 2, 3, 4, 5, 6\}$

Fixed Value	Satisfactory Combination?
	$C_3 : 2 \leq a + b + 2c \leq 5$
$a = 0$, $a = 1$, $a = 2$, $a = 3$, $a = 4$	Yes, Yes, Yes, Yes, Yes
$b = 1$, $b = 2$	Yes, Yes
$c = 0$, $c = 1$, $c = 2$, $c = 3$	Yes, Yes, Yes, No

- $\text{dom}(A) = \{0, 1, 2, 3, 4\}$
- $\text{dom}(B) = \{\cancel{0}, 1, 2, \cancel{3}, \cancel{4}\}$
- $\text{dom}(C) = \{0, 1, 2, \cancel{3}\}$
- $\text{dom}(D) = \{0, 1, 2, 3, 4, 5\}$
- $\text{dom}(E) = \{0, 1, 2, 3, 4, 5, 6\}$

Example:

$$C_2 : 3 \leq a + c \leq 4$$

Fixed Value	**Satisfactory Combination?**
$a = 0$, $a = 1$, $a = 2$, $a = 3$, $a = 4$	No, Yes, Yes, Yes, Yes
$c = 0$, $c = 1$, $c = 2$	Yes, Yes, Yes

- $\text{dom}(A) = \{\cancel{0}, 1, 2, 3, 4\}$
- $\text{dom}(B) = \{\cancel{0}, 1, 2, \cancel{3}, \cancel{4}\}$
- $\text{dom}(C) = \{0, 1, 2, \cancel{3}\}$
- $\text{dom}(D) = \{0, 1, 2, 3, 4, 5\}$
- $\text{dom}(E) = \{0, 1, 2, 3, 4, 5, 6\}$

$$C_3 : 2 \leq a + b + 2c \leq 5$$

Fixed Value	**Satisfactory Combination?**
$a = 1$, $a = 2$, $a = 3$, $a = 4$	Yes, Yes, Yes, Yes
$b = 1$, $b = 2$	Yes, Yes
$c = 0$, $c = 1$, $c = 2$	Yes, Yes, No

- $\text{dom}(A) = \{\cancel{0}, 1, 2, 3, 4\}$
- $\text{dom}(B) = \{\cancel{0}, 1, 2, \cancel{3}, \cancel{4}\}$
- $\text{dom}(C) = \{0, 1, \cancel{2}, \cancel{3}\}$
- $\text{dom}(D) = \{0, 1, 2, 3, 4, 5\}$
- $\text{dom}(E) = \{0, 1, 2, 3, 4, 5, 6\}$

$$C_2 : 3 \leq a + c \leq 4$$

Fixed Value	**Satisfactory Combination?**
$a = 1$, $a = 2$, $a = 3$, $a = 4$	No, Yes, Yes, Yes
$c = 0$, $c = 1$	Yes, Yes

- $\text{dom}(A) = \{\cancel{0}, \cancel{1}, 2, 3, 4\}$
- $\text{dom}(B) = \{\cancel{0}, 1, 2, \cancel{3}, \cancel{4}\}$
- $\text{dom}(C) = \{0, 1, \cancel{2}, \cancel{3}\}$
- $\text{dom}(D) = \{0, 1, 2, 3, 4, 5\}$
- $\text{dom}(E) = \{0, 1, 2, 3, 4, 5, 6\}$

$$C_3 : 2 \leq a + b + 2c \leq 5$$

Fixed Value	**Satisfactory Combination?**
$a = 2$, $a = 3$, $a = 4$	Yes, Yes, Yes
$b = 1$, $b = 2$	Yes, Yes
$c = 0$, $c = 1$	Yes, Yes

- $\text{dom}(A) = \{\cancel{0}, \cancel{1}, 2, 3, 4\}$
- $\text{dom}(B) = \{\cancel{0}, 1, 2, \cancel{3}, \cancel{4}\}$
- $\text{dom}(C) = \{0, 1, \cancel{2}, \cancel{3}\}$
- $\text{dom}(D) = \{0, 1, 2, 3, 4, 5\}$
- $\text{dom}(E) = \{0, 1, 2, 3, 4, 5, 6\}$

Example:

Fixed Value	**Satisfactory Combination?**
$C_1 : 8 \leq 2a + 2b + 2d + 3e \leq 10$	
$a = 2$, $a = 3$, $a = 4$	Yes, Yes, Yes
$b = 1$, $b = 2$	Yes, Yes
$d = 0$, $d = 1$, $d = 2$, $d = 3$, $d = 4$, $d = 5$	Yes, Yes, Yes, No, No, No
$e = 0$, $e = 1$, $e = 2$, $e = 3$, $e = 4$, $e = 5$, $e = 6$	Yes, Yes, No, No, No, No

- $\text{dom}(A) = \{\cancel{0}, \cancel{1}, 2, 3, 4\}$
- $\text{dom}(B) = \{\cancel{0}, 1, 2, \cancel{3}, \cancel{4}\}$
- $\text{dom}(C) = \{0, 1, \cancel{2}, \cancel{3}\}$
- $\text{dom}(D) = \{0, 1, 2, \cancel{3}, \cancel{4}, \cancel{5}\}$
- $\text{dom}(E) = \{0, 1, \cancel{2}, \cancel{3}, \cancel{4}, \cancel{5}, \cancel{6}\}$

Fixed Value	**Satisfactory Combination?**
$C_1 : 8 \leq 2a + 2b + 2d + 3e \leq 10$	
$a = 2$, $a = 3$, $a = 4$	Yes, Yes, Yes
$b = 1$, $b = 2$	Yes, Yes
$d = 0$, $d = 1$, $d = 2$	Yes, Yes, Yes
$e = 0$, $e = 1$	Yes, Yes

- $\text{dom}(A) = \{\cancel{0}, \cancel{1}, 2, 3, 4\}$
- $\text{dom}(B) = \{\cancel{0}, 1, 2, \cancel{3}, \cancel{4}\}$
- $\text{dom}(C) = \{0, 1, \cancel{2}, \cancel{3}\}$
- $\text{dom}(D) = \{0, 1, 2, \cancel{3}, \cancel{4}, \cancel{5}\}$
- $\text{dom}(E) = \{0, 1, \cancel{2}, \cancel{3}, \cancel{4}, \cancel{5}, \cancel{6}\}$

4. **Conclusion:** This is the final pruned domain, which means if I fix a variable to a value, then I can find a satisfactory combination for the rest of the variables in this pruned domain.
    - $\text{dom}(A) = \{2, 3, 4\}$
    - $\text{dom}(B) = \{1, 2\}$
    - $\text{dom}(C) = \{0, 1\}$
    - $\text{dom}(D) = \{0, 1, 2\}$
    - $\text{dom}(E) = \{0, 1\}$

Example: 3-Consistency

Fixed Value	**Satisfactory Combination?**
$C_4 : 1 \leq b \leq 2$	
$b=0$, $b=1$, $b=2$, $b=3$, $b=4$	No, Yes, Yes, No, No

- $\text{dom}(A) = \{0, 1, 2, 3, 4\}$
- $\text{dom}(B) = \{\cancel{0}, 1, 2, \cancel{3}, \cancel{4}\}$
- $\text{dom}(C) = \{0, 1, 2, 3\}$
- $\text{dom}(D) = \{0, 1, 2, 3, 4, 5\}$
- $\text{dom}(E) = \{0, 1, 2, 3, 4, 5, 6\}$

Fixed Value	**Satisfactory Combination?**
$C_2 : 3 \leq a + c \leq 4$	
$a=0$, $a=1$, $a=2$, $a=3$, $a=4$	Yes, Yes, Yes, Yes, Yes
$c=0$, $c=1$, $c=2$, $c=3$	Yes, Yes, Yes, Yes

- $\text{dom}(A) = \{0, 1, 2, 3, 4\}$
- $\text{dom}(B) = \{\cancel{0}, 1, 2, \cancel{3}, \cancel{4}\}$
- $\text{dom}(C) = \{0, 1, 2, 3\}$
- $\text{dom}(D) = \{0, 1, 2, 3, 4, 5\}$
- $\text{dom}(E) = \{0, 1, 2, 3, 4, 5, 6\}$

Fixed Value	**Satisfactory Combination?**
$C_3 : 2 \leq a + b + 2c \leq 5$	
$a=0$, $a=1$, $a=2$, $a=3$, $a=4$	Yes, Yes, Yes, Yes, Yes
$b=1$, $b=2$	Yes, Yes
$c=0$, $c=1$, $c=2$, $c=3$	Yes, Yes, Yes, No

- $\text{dom}(A) = \{0, 1, 2, 3, 4\}$
- $\text{dom}(B) = \{\cancel{0}, 1, 2, \cancel{3}, \cancel{4}\}$
- $\text{dom}(C) = \{0, 1, 2, \cancel{3}\}$
- $\text{dom}(D) = \{0, 1, 2, 3, 4, 5\}$
- $\text{dom}(E) = \{0, 1, 2, 3, 4, 5, 6\}$

Example:

Fixed Value	**Satisfactory Combination?**
	$C_2 : 3 \leq a + c \leq 4$
$a = 0$, $a = 1$, $a = 2$, $a = 3$, $a = 4$	No, Yes, Yes, Yes, Yes
$c = 0$, $c = 1$, $c = 2$	Yes, Yes, Yes

- $\text{dom}(A) = \{\cancel{0}, 1, 2, 3, 4\}$
- $\text{dom}(B) = \{\cancel{0}, 1, 2, \cancel{3}, \cancel{4}\}$
- $\text{dom}(C) = \{0, 1, 2, \cancel{3}\}$
- $\text{dom}(D) = \{0, 1, 2, 3, 4, 5\}$
- $\text{dom}(E) = \{0, 1, 2, 3, 4, 5, 6\}$

	$C_3 : 2 \leq a + b + 2c \leq 5$
$a = 1$, $a = 2$, $a = 3$, $a = 4$	Yes, Yes, Yes, Yes
$b = 1$, $b = 2$	Yes, Yes
$c = 0$, $c = 1$, $c = 2$	Yes, Yes, No

- $\text{dom}(A) = \{\cancel{0}, 1, 2, 3, 4\}$
- $\text{dom}(B) = \{\cancel{0}, 1, 2, \cancel{3}, \cancel{4}\}$
- $\text{dom}(C) = \{0, 1, \cancel{2}, \cancel{3}\}$
- $\text{dom}(D) = \{0, 1, 2, 3, 4, 5\}$
- $\text{dom}(E) = \{0, 1, 2, 3, 4, 5, 6\}$

	$C_2 : 3 \leq a + c \leq 4$
$a = 1$, $a = 2$, $a = 3$, $a = 4$	No, Yes, Yes, Yes
$c = 0$, $c = 1$	Yes, Yes

- $\text{dom}(A) = \{\cancel{0}, \cancel{1}, 2, 3, 4\}$
- $\text{dom}(B) = \{\cancel{0}, 1, 2, \cancel{3}, \cancel{4}\}$
- $\text{dom}(C) = \{0, 1, \cancel{2}, \cancel{3}\}$
- $\text{dom}(D) = \{0, 1, 2, 3, 4, 5\}$
- $\text{dom}(E) = \{0, 1, 2, 3, 4, 5, 6\}$

	$C_3 : 2 \leq a + b + 2c \leq 5$
$a = 2$, $a = 3$, $a = 4$	Yes, Yes, Yes
$b = 1$, $b = 2$	Yes, Yes
$c = 0$, $c = 1$	Yes, Yes

- $\text{dom}(A) = \{\cancel{0}, \cancel{1}, 2, 3, 4\}$
- $\text{dom}(B) = \{\cancel{0}, 1, 2, \cancel{3}, \cancel{4}\}$
- $\text{dom}(C) = \{0, 1, \cancel{2}, \cancel{3}\}$
- $\text{dom}(D) = \{0, 1, 2, 3, 4, 5\}$
- $\text{dom}(E) = \{0, 1, 2, 3, 4, 5, 6\}$

4. **Conclusion:** $\text{dom}(A) = \{2, 3, 4\}$, $\text{dom}(B) = \{1, 2\}$, $\text{dom}(C) = \{0, 1\}$, $\text{dom}(D) = \{0, 1, 2, 3, 4, 5\}$, $\text{dom}(E) = \{0, 1, 2, 3, 4, 5, 6\}$

Example: **2-Consistency**

Fixed Value	**Satisfactory Combination?**
$C_4 : 1 \leq b \leq 2$	
$b = 0$, $b = 1$, $b = 2$, $b = 3$, $b = 4$	No, Yes, Yes, No, No

- $\text{dom}(A) = \{0, 1, 2, 3, 4\}$
- $\text{dom}(B) = \{\cancel{0}, 1, 2, \cancel{3}, \cancel{4}\}$
- $\text{dom}(C) = \{0, 1, 2, 3\}$
- $\text{dom}(D) = \{0, 1, 2, 3, 4, 5\}$
- $\text{dom}(E) = \{0, 1, 2, 3, 4, 5, 6\}$

Fixed Value	**Satisfactory Combination?**
$C_2 : 3 \leq a + c \leq 4$	
$a = 0$, $a = 1$, $a = 2$, $a = 3$, $a = 4$	Yes, Yes, Yes, Yes, Yes
$c = 0$, $c = 1$, $c = 2$, $c = 3$	Yes, Yes, Yes, Yes

- $\text{dom}(A) = \{0, 1, 2, 3, 4\}$
- $\text{dom}(B) = \{\cancel{0}, 1, 2, \cancel{3}, \cancel{4}\}$
- $\text{dom}(C) = \{0, 1, 2, 3\}$
- $\text{dom}(D) = \{0, 1, 2, 3, 4, 5\}$
- $\text{dom}(E) = \{0, 1, 2, 3, 4, 5, 6\}$

4. **Conclusion:** $\text{dom}(A) = \{0, 1, 2, 3, 4\}$, $\text{dom}(B) = \{1, 2\}$, $\text{dom}(C) = \{0, 1, 2, 3\}$, $\text{dom}(D) = \{0, 1, 2, 3, 4, 5\}$, $\text{dom}(E) = \{0, 1, 2, 3, 4, 5, 6\}$

Example: **1-Consistency**

Fixed Value	**Satisfactory Combination?**
$C_4 : 1 \leq b \leq 2$	
$b = 0$, $b = 1$, $b = 2$, $b = 3$, $b = 4$	No, Yes, Yes, No, No

- $\text{dom}(A) = \{0, 1, 2, 3, 4\}$
- $\text{dom}(B) = \{\cancel{0}, 1, 2, \cancel{3}, \cancel{4}\}$
- $\text{dom}(C) = \{0, 1, 2, 3\}$
- $\text{dom}(D) = \{0, 1, 2, 3, 4, 5\}$
- $\text{dom}(E) = \{0, 1, 2, 3, 4, 5, 6\}$

4. **Conclusion:** $\text{dom}(A) = \{0, 1, 2, 3, 4\}$, $\text{dom}(B) = \{1, 2\}$, $\text{dom}(C) = \{0, 1, 2, 3\}$, $\text{dom}(D) = \{0, 1, 2, 3, 4, 5\}$, $\text{dom}(E) = \{0, 1, 2, 3, 4, 5, 6\}$

Learning Problems

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:

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

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

3.6 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 $\mathcal{Y} \subseteq \mathbb{R}$

3.7 Feature Spaces

Definition: It is often easier to learn relationships from high-level features (instead of the raw input). Need mapping b/w input space and feature space:

$$\phi : \mathcal{X} \to \mathcal{F}$$

4 PAC Learning

4.1 Probably Approximately Correct (PAC) Estimations

4.1.1 Hoeffding's Inequality

Definition: For any $\epsilon > 0$,

$$\mathbb{P}(|\nu - \mu| \geq \epsilon) \leq 2e^{-2\epsilon^2 N} \tag{1}$$

- μ: Probabillity of an event.
- ν: Relative frequency in a sample size N.
- $\mu \overset{?}{\approx} \nu$: μ is probably approximately equal to ν. As $N \to \infty$: $\nu \to \mu$
- ϵ: Tolerance (i.e. how close we want ν to be to μ).
  - $\epsilon \to 0$: $\nu = \mu$

Warning: Approximate the true distribution with high probability by taking a large enough sample size (i.e. empirical distribution converges to true distribution).

4.2 PAC Learning

4.2.1 Error

Definition:
- **Out-Sample Error:**
$$E_{\text{out}} = \mathbb{P}[f \neq h]$$
- **In-Sample Error:**
$$E_{\text{in}} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}[f(x^{(i)}) \neq h(x^{(i)})]$$

4.2.2 Union Bound Theorem

Theorem:

$$\mathbb{P}\left[E_1 \vee \cdots \vee E_M\right] \leq \sum_{i=1}^{M} \mathbb{P}[E_i]$$

Warning: If the events are highly correlated, then the union bound is not tight.

4.2.3 Generalization of Hoeffding's Inequality

Definition: Assuming that h is chosen from a set of hypotheses $\mathcal{H}$, derive a (loose) upper-bound on $|E_{\text{out}} - E_{\text{in}}|$:

$$\begin{aligned}
\mathbb{P}\left[\bigvee_{h \in \mathcal{H}} (|E_{\text{out}} - E_{\text{in}}(h)| > \varepsilon)\right] &\leq \sum_{h \in \mathcal{H}} \mathbb{P}\left[|E_{\text{out}} - E_{\text{in}}(h)| > \varepsilon\right] \\
&\leq \sum_{h \in \mathcal{H}} 2e^{-2\varepsilon^2 N} \\
&= 2|\mathcal{H}|e^{-2\varepsilon^2 N}
\end{aligned}$$

- Endow $\mathcal{F}$ w/ prob. distribution, $P : \mathcal{X} \to [0, 1]$, then
  - E_{out} is analogous to μ
  - $E_{\text{in}}(h)$ is analogous to ν.

Notes:

- $E_{\text{in}}(h) \overset{?}{\approx} E_{\text{out}}$ requires small $|\mathcal{H}|$ (generalization)
- $E_{\text{in}}(h) \approx 0$ requires large $|\mathcal{H}|$ (discrimination)

Example:

1. **Given:** An opaque box containing red and blue balls. Take N IID samples.
   - μ: Probability of drawing a ball (unknown).
   - ν: Relative frequency of balls in the sample (known).
2. **Problem 1:** What is ν in this case? 8 balls total, 5 are blue.
3. **Solution 1:** $\nu = \frac{5}{8}$
4. **Problem 2:** How to partition $\mathcal{F}$ into regions where $f = h$ and $f \neq h$?
5. **Solution 2:**

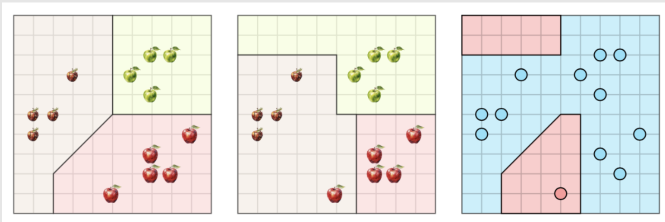


Figure 30: LS h, MS f

6. **Problem 3:** What is the out-sample error?
7. **Solution 3:** In words, the probability of the hypothesis being wrong.

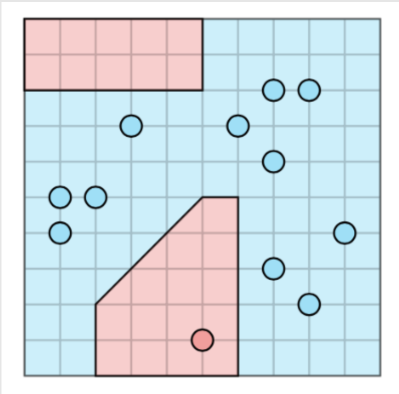


Figure 31

8. **Problem 4:** What is the in-sample error given this sample of 11 balls s.t. $f = h$, 1 ball s.t. $f \neq h$?
9. **Solution 4:** $E_{\text{in}} = \frac{1}{12}$

5 Decision Trees

5.1 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.

Notes: We want to find the minimum # of condition vertices (or questions) needed to "sufficiently discriminate" (identify the class of every point in $\mathcal{D}$).
- More condition vertices improve discrimination.
- Less condition vertices improve generalization.

5.2 Building a Decision Tree

Definition: Consider determining the calss of a randomly chosen target point.
- If we ask a K-ary question abt. the pts. in $\mathcal{D}$, we can form K subsets, $\mathcal{D}^{(1)}, \ldots, \mathcal{D}^{(K)}$, using the answers s.t.
  - $|\mathcal{D}^{(k)}| \in \{0, \ldots, |\mathcal{D}|\}$
  - $|\mathcal{D}| = \sum_{k=1}^{K} |\mathcal{D}^{(k)}|$

5.2.1 Special Case

Notes: Suppose each pt. belongs to a unique class (i.e. the # of classes is $|\mathcal{D}|$).

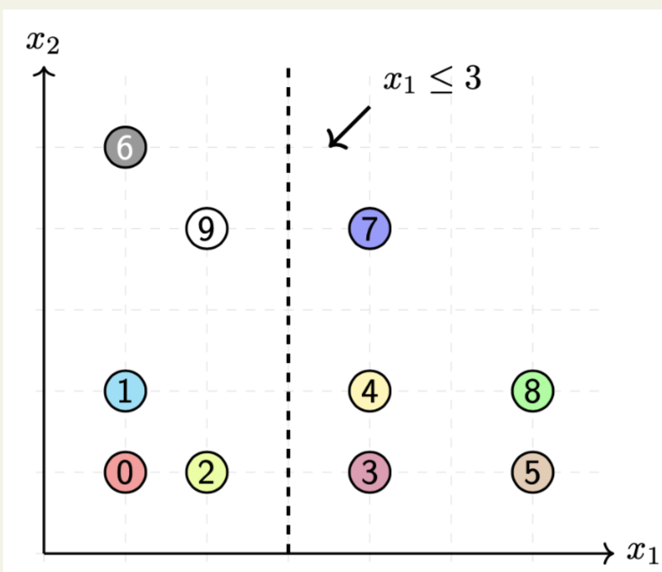


Figure 32

1. Before asking the question: $|\mathcal{D}|$ possible guesses for the target point's class.
2. After asking the question: Either
   - $|\mathcal{D}^{(1)}|, \ldots, |\mathcal{D}^{(K-1)}|$ or
   - $|\mathcal{D}^{(K)}|$

   guesses, depending on the answer for the target point.
3. Goal: Minimize the # of guesses needed in the worst-case, which would be

$$\max\{|\mathcal{D}^{(1)}|, \ldots, |\mathcal{D}^{(K)}|\}.$$

4. Given the constraints on $|\mathcal{D}^{(1)}|, \ldots, |\mathcal{D}^{(K)}|$, we can show that $\max\{|\mathcal{D}^{(1)}|, \ldots, |\mathcal{D}^{(K)}|\}$ is minimized when

$$|\mathcal{D}^{(K)}| \in \left\{ \left\lfloor \frac{|\mathcal{D}|}{K} \right\rfloor, \left\lceil \frac{|\mathcal{D}|}{K} \right\rceil \right\}.$$

   Basically, the best question splits $\mathcal{D}$ into K sets of (roughly) the same size.

Warning: Roughly due to floor/ceil.

Theorem: Given a classification data-set, $\mathcal{D}$, in which the class of each point is unique (i.e., $|\text{out}(\mathcal{D})| = |\mathcal{D}|$), the class of a randomly chosen target point can be determined within

$$\lceil \log_K(|\mathcal{D}|) \rceil$$

K-ary questions.

5.2.2 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).

5.2.3 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 ∞ 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:

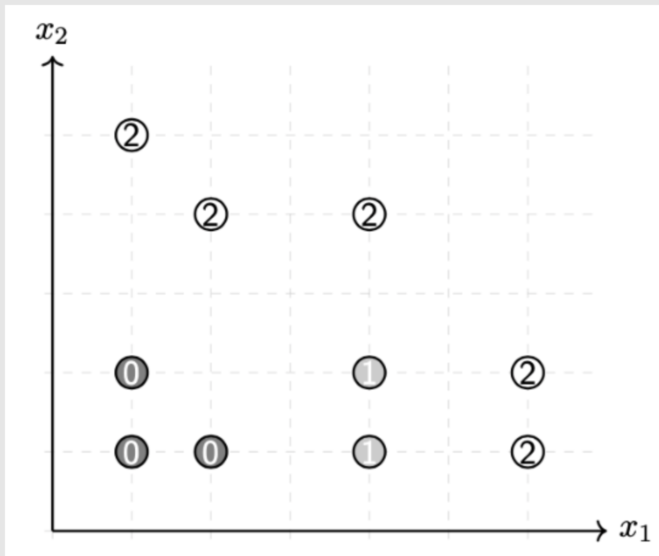


Figure 33

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: $H(X) = \frac{3}{10}\log_2\left(\frac{10}{3}\right) + \frac{2}{10}\log_2\left(\frac{10}{2}\right) + \frac{5}{10}\log_2\left(\frac{10}{5}\right)$
   (b) Entropy after the split:
      i. $H(X \mid x_1 \leq 3) = \frac{3}{5}\log_2\left(\frac{5}{3}\right) + \frac{2}{5}\log_2\left(\frac{5}{2}\right)$
      ii. $H(X \mid x_1 > 3) = \frac{2}{5}\log_2\left(\frac{5}{2}\right) + \frac{3}{5}\log_2\left(\frac{5}{3}\right)$.
      iii. Weighted Avg. Entropy: $H(X|Y) = \frac{5}{10}H(X \mid x_1 \leq 3) + \frac{5}{10}H(X \mid x_1 > 3)$
   (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: $H(X) = \frac{3}{10}\log_2\left(\frac{10}{3}\right) + \frac{2}{10}\log_2\left(\frac{10}{2}\right) + \frac{5}{10}\log_2\left(\frac{10}{5}\right)$
   (b) Entropy after the split:
      i. $H(X \mid x_2 > 3) = \frac{3}{3}\log_2\left(\frac{3}{3}\right)$
      ii. $H(X \mid x_2 \leq 3) = \frac{3}{5}\log_2\left(\frac{5}{3}\right) + \frac{2}{5}\log_2\left(\frac{5}{2}\right) + \frac{2}{5}\log_2\left(\frac{5}{2}\right)$.
      iii. Weighted Avg. Entropy: $H(X|Y) = \frac{3}{10}H(X \mid x_2 > 3) + \frac{7}{10}H(X \mid x_2 \leq 3)$
   (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. $H(X \mid x_1 \leq 3 \text{ and } x_2 \leq 3) = \frac{3}{3}\log_2\left(\frac{3}{3}\right)$
      ii. $H(X \mid x_1 \leq 3 \text{ and } x_2 > 3) = \frac{2}{2}\log_2\left(\frac{2}{2}\right)$
      iii. $H(X \mid x_1 > 3) = \frac{2}{5}\log_2\left(\frac{5}{2}\right) + \frac{3}{5}\log_2\left(\frac{5}{3}\right)$
      iv. $H(X|Y) = \frac{3}{10}H(X \mid x_1 \leq 3 \text{ and } x_2 \leq 3) + \frac{2}{10}H(X \mid x_1 \leq 3 \text{ and } x_2 > 3) + \frac{5}{10}H(X \mid x_1 > 3)$
   (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 (Level 1):**
   (a) Entropy before the split: $H(X) = \frac{3}{10}\log_2\left(\frac{10}{3}\right) + \frac{2}{10}\log_2\left(\frac{10}{2}\right) + \frac{5}{10}\log_2\left(\frac{10}{5}\right) = 1.485[\text{bits}]$
   (b) Entropy after the split and information gain (everything in base 2 since 2-ary).

Split	**Entropy**
$x_1 \leq 1$	$H(X\|Y) = \frac{3}{10}\left[\frac{2}{3}\log\left(\frac{3}{2}\right) + \frac{1}{3}\log\left(\frac{3}{1}\right)\right] + \frac{7}{10}\left[\frac{1}{7}\log\left(\frac{7}{1}\right) + \frac{2}{7}\log\left(\frac{7}{2}\right) + \frac{4}{7}\log\left(\frac{7}{4}\right)\right] = 1.241[\text{bits}]$ • $IG(X\|Y) = 1.485 - 1.241 = 0.244[\text{bits}]$
$x_1 \leq 2, 3$	$H(X\|Y) = \frac{5}{10}\left[\frac{3}{5}\log\left(\frac{5}{3}\right) + \frac{2}{5}\log\left(\frac{5}{2}\right)\right] + \frac{5}{10}\left[\frac{2}{5}\log\left(\frac{5}{2}\right) + \frac{3}{5}\log\left(\frac{5}{3}\right)\right] = 0.971[\text{bits}]$ • $\boxed{IG(X\|Y) = 1.485 - 0.971 = 0.514[\text{bits}]}$
$x_1 \leq 4, 5$	$H(X\|Y) = \frac{8}{10}\left[\frac{3}{8}\log\left(\frac{8}{3}\right) + \frac{2}{8}\log\left(\frac{8}{2}\right) + \frac{3}{8}\log\left(\frac{8}{3}\right)\right] + \frac{2}{10}\left[\frac{2}{2}\log\left(\frac{2}{2}\right)\right] = 1.249[\text{bits}]$ • $IG(X\|Y) = 1.485 - 1.249 = 0.236[\text{bits}]$
$x_1 \leq 6$	$H(X\|Y) = \frac{10}{10}\left[\frac{3}{10}\log\left(\frac{10}{3}\right) + \frac{2}{10}\log\left(\frac{10}{2}\right) + \frac{5}{10}\log\left(\frac{10}{5}\right)\right] = 1.485[\text{bits}]$ • $IG(X\|Y) = 1.485 - 1.485 = 0[\text{bits}]$
$x_2 \leq 1$	$H(X\|Y) = \frac{4}{10}\left[\frac{2}{4}\log\left(\frac{4}{2}\right) + \frac{1}{4}\log\left(\frac{4}{1}\right) + \frac{1}{4}\log\left(\frac{4}{1}\right)\right] + \frac{6}{10}\left[2 \cdot \frac{1}{6}\log\left(\frac{6}{1}\right) + \frac{4}{6}\log\left(\frac{6}{4}\right)\right] = 1.351[\text{bits}]$ • $IG(X\|Y) = 1.485 - 1.351 = 0.134[\text{bits}]$
$x_2 \leq 2, 3$	$H(X\|Y) = \frac{7}{10}\left[\frac{3}{7}\log\left(\frac{7}{3}\right) + \frac{2}{7}\log\left(\frac{7}{2}\right) + \frac{2}{7}\log\left(\frac{7}{2}\right)\right] + \frac{3}{10}\left[\frac{3}{3}\log\left(\frac{3}{3}\right)\right] = 1.090[\text{bits}]$ • $IG(X\|Y) = 1.485 - 1.090 = 0.395[\text{bits}]$
$x_2 \leq 4$	$H(X\|Y) = \frac{9}{10}\left[\frac{3}{9}\log\left(\frac{9}{3}\right) + \frac{2}{9}\log\left(\frac{9}{2}\right) + \frac{4}{9}\log\left(\frac{9}{4}\right)\right] + \frac{1}{10}\left[\frac{1}{1}\log\left(\frac{1}{1}\right)\right] = 1.377[\text{bits}]$ • $IG(X\|Y) = 1.485 - 1.377 = 0.108[\text{bits}]$
$x_2 \leq 5$	$H(X\|Y) = \frac{10}{10}\left[\frac{3}{10}\log\left(\frac{10}{3}\right) + \frac{2}{10}\log\left(\frac{10}{2}\right) + \frac{5}{10}\log\left(\frac{10}{5}\right)\right] = 1.485[\text{bits}]$ • $IG(X\|Y) = 1.485 - 1.485 = 0[\text{bits}]$

Example: Decision Tree Continued:

4. **Solution (Level 2):** $x_1 \leq 2, 3$ has the highest information gain. For clarity, choose $x_1 \leq 3$ as the question.
   (a) Entropy before the split (treat as 2 indep. problems)
      i. $H(X_L) = \frac{3}{5}\log\left(\frac{5}{3}\right) + \frac{2}{5}\log\left(\frac{5}{2}\right) = 0.971$
      ii. $H(X_R) = \frac{2}{5}\log\left(\frac{5}{2}\right) + \frac{3}{5}\log\left(\frac{5}{3}\right) = 0.971$
   (b) Entropy after the split and information gain (everything in base 2 since 2-ary).

Split	Entropy
Left Split	
$x_1 \leq 1$	$H(X_L\|Y) = \frac{3}{5}\left[\frac{2}{3}\log\left(\frac{3}{2}\right) + \frac{1}{3}\log\left(\frac{3}{1}\right)\right] + \frac{2}{5}\left[\frac{1}{2}\log\left(\frac{1}{2}\right) + \frac{1}{2}\log\left(\frac{1}{2}\right)\right] = 0.151[\text{bits}]$
	• $IG(X\|Y) = 0.971 - 0.151 = 0.820[\text{bits}]$
$x_2 \leq 1$	$H(X_L\|Y) = \frac{2}{5}\left[\frac{2}{2}\log\left(\frac{2}{2}\right)\right] + \frac{3}{5}\left[\frac{1}{3}\log\left(\frac{3}{1}\right) + \frac{2}{3}\log\left(\frac{3}{2}\right)\right] = 0.551[\text{bits}]$
	• $IG(X\|Y) = 0.971 - 0.551 = 0.420[\text{bits}]$
$x_2 \leq 2, 3$	$H(X_L\|Y) = \frac{3}{5}\left[\frac{3}{3}\log\left(\frac{3}{3}\right)\right] + \frac{2}{5}\left[\frac{2}{2}\log\left(\frac{2}{2}\right)\right] = 0[\text{bits}]$
	• $\boxed{IG(X_L\|Y) = 0.971 - 0 = 0.971[\text{bits}]}$
Right Split	
$x_1 \leq 4, 5$	$H(X_R\|Y) = \frac{3}{5}\left[\frac{2}{3}\log\left(\frac{3}{2}\right) + \frac{1}{3}\log\left(\frac{3}{1}\right)\right] + \frac{2}{5}\left[\frac{2}{2}\log\left(\frac{2}{2}\right)\right] = 0.551[\text{bits}]$
	• $\boxed{IG(X_L\|Y) = 0.971 - 0.551 = 0.420[\text{bits}]}$
$x_2 \leq 1$	$H(X_R\|Y) = \frac{2}{5}\left[\frac{1}{2}\log\left(\frac{2}{1}\right) + \frac{1}{2}\log\left(\frac{2}{1}\right)\right] + \frac{3}{5}\left[\frac{2}{3}\log\left(\frac{3}{2}\right) + \frac{1}{3}\log\left(\frac{3}{1}\right)\right] = 0.951[\text{bits}]$
	• $IG(X_L\|Y) = 0.971 - 0.951 = 0.020[\text{bits}]$
$x_2 \leq 2, 3$	$H(X_R\|Y) = \frac{4}{5}\left[\frac{2}{4}\log\left(\frac{4}{2}\right) + \frac{2}{4}\log\left(\frac{4}{2}\right)\right] + \frac{1}{5}\left[\frac{1}{1}\log\left(\frac{1}{1}\right)\right] = 0.8[\text{bits}]$
	• $IG(X_L\|Y) = 0.971 - 0.8 = 0.171[\text{bits}]$

Example: Decision Tree Continued:

5. **Solution (Level 3):** $x_2 \leq 2, 3$ and $x_1 \leq 4, 5$ has the highest information gain. For clarity, choose $x_2 \leq 3$ as the question for the left split and choose $x_1 \leq 5$ as the question for the right split.
   (a) Since 3 are pure splits already, therefore, look at right-left side only.
   (b) Entropy before the split for the right-left side
      i. $H(X_{RL}) = \frac{2}{3}\log\left(\frac{3}{2}\right) + \frac{1}{3}\log\left(\frac{3}{1}\right) = 0.918[\text{bits}]$
   (c) Entropy after the split and information gain (everything in base 2 since 2-ary).

Split	**Entropy**
$x_2 \leq 1$	$H(X_{RL}\|Y) = \frac{1}{3}\left[\frac{1}{1}\log\left(\frac{1}{1}\right)\right] + \frac{2}{3}\left[\frac{1}{2}\log\left(\frac{2}{1}\right) + \frac{1}{2}\log\left(\frac{2}{1}\right)\right] = 0.667[\text{bits}]$ • $IG(X\|Y) = 0.971 - 0.667 = 0.304[\text{bits}]$
$x_2 \leq 2, 3$	$H(X_{RL}\|Y) = \frac{1}{3}\left[\frac{1}{1}\log\left(\frac{1}{1}\right)\right] + \frac{2}{3}\left[\frac{2}{2}\log\left(\frac{2}{2}\right)\right] = 0[\text{bits}]$ • $\boxed{IG(X\|Y) = 0.971 - 0 = 0.971[\text{bits}]}$

6. Now all regions in our graph contain a pure set (one class). Note this took more questions than needed, but IG is a heuristic so its not perfect.