ECE353 Lectures

Hanhee Lee

January 25, 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

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

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

- 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:

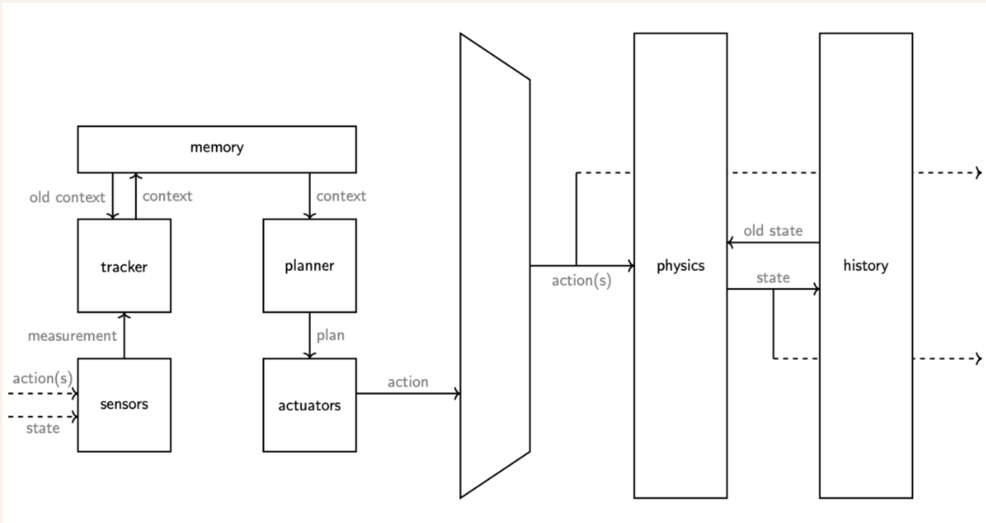


Figure 1: Components of a Robotic System (Words)

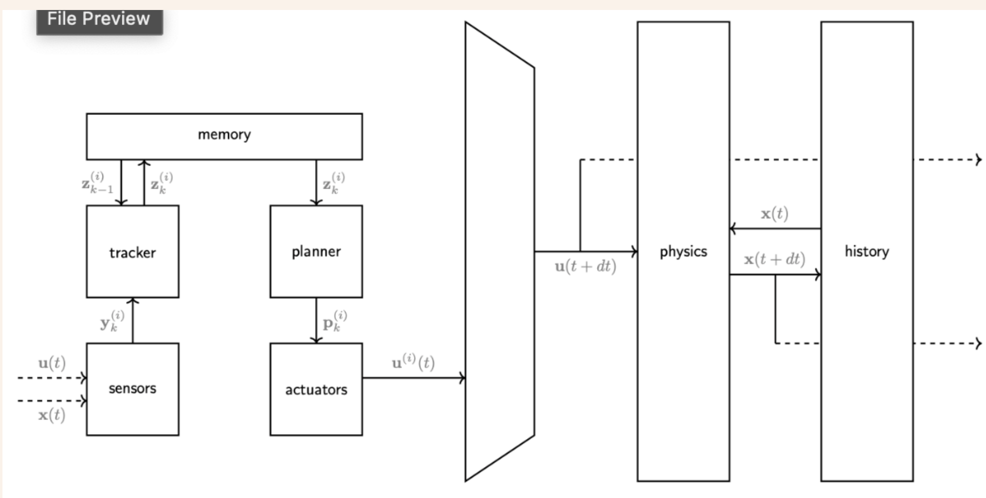


Figure 2: Components of a Robotic System (Math)

1.2.1 Overview (Robots, the Environment)

Definition:

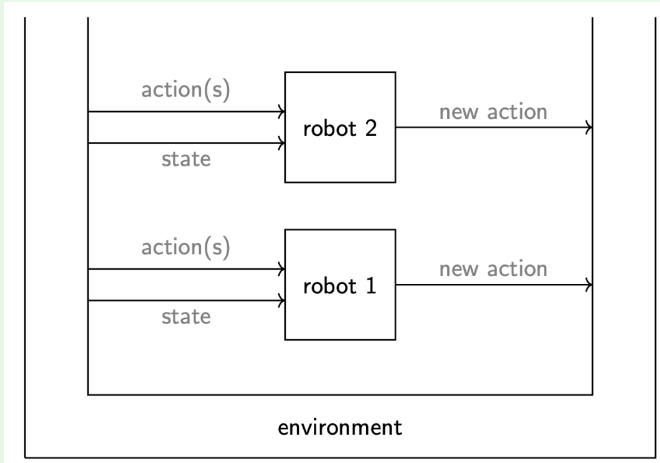


Figure 3: Overview (Robots, the Environment)

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

1.2.2 Robot (Sensors, Actuators, the Brain)

Definition:

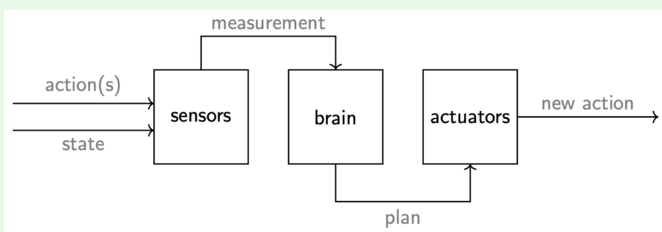


Figure 4: Robot (Sensors, Actuators, the Brain)

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:

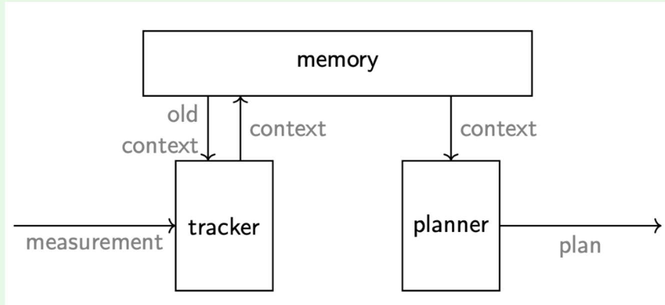


Figure 5: Brain (Tracker, Planner, Memory)

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:

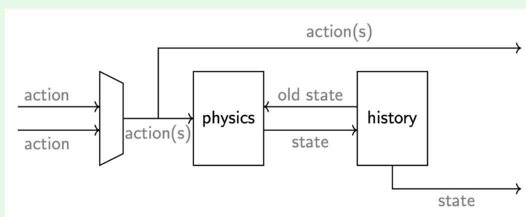


Figure 6: Environment (Physics, State)

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 $T^{(i)}$:

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

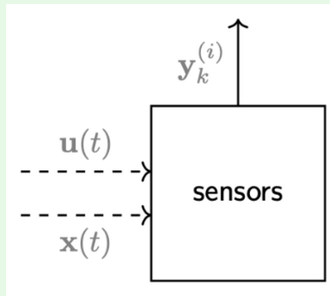


Figure 7: Sensing

1.3.2 Tracking

Definition: Track (update) the context:

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

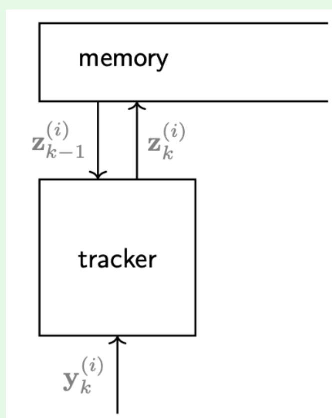


Figure 8: Tracking

1.3.3 Planning

Definition: Make a plan:

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

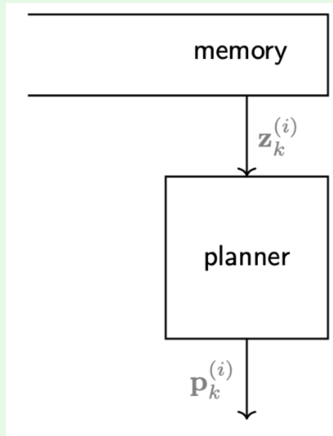


Figure 9: Planning

1.3.4 Acting

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

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

Execute the plan:

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

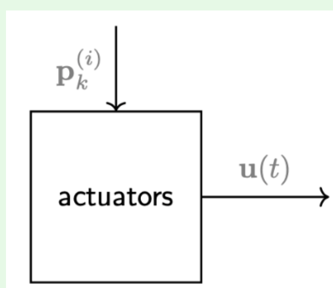


Figure 10: Acting

1.3.5 Simulating

Definition: Simulate the environment's response:

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

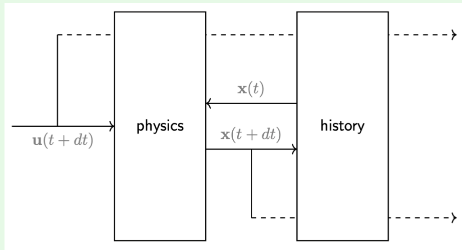


Figure 11: Simulating

2 Uninformed/Informed Search Algorithms

Summary:

Alg.	Halting	Sound	Complete	Optimal	Time	Space
			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.						
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.						
IIA*	always	always	always	always	b^{l^*}	bl^*
• Same as A* but with iterative inflating on ecst.						

2.1 Setup

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

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

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

2.2 Search Graphs

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

2.3 Path Trees

Definition: A search algorithm explores a tree of possible paths.
- In such a tree, each node represents the path from the root to itself.
  - The node may also include other info (such as the path's origiin, cost, etc).

2.4 Search Algorithms

Definition: All search algorithms follow the template below:

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

- $\langle\rangle$ is the empty path, and 0 is the cost of the 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
        𝒪 ← 𝒪 ∪ {n′}                            ▷ "expand" n and "export" its children
    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 $\textsc{Remove}(\cdot)$ removes nodes.

2.4.1 Characteristics of a Search Algorithm

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

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

- Will be using explored for time efficiency.

The characteristics of the algorithm also depend on several properties of the path tree over which it searches. These properties include:
- Branching factor: b ($b < \infty$), the maximum number of children a node can have.
- Depth: d, the length of the longest path.
- Length of the shortest solution: l^*

- Cost of the cheapest solution: c^*
- Cost of the cheapest edge: ϵ

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

2.5 Modifications to Search Algorithms

2.5.1 Depth-Limiting

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

```
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)
```

2.5.2 Iterative Deepening

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

```
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.

2.5.3 Cost-Limiting

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

```
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)
```

2.5.4 Iterative-Inflating

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

```
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.

2.5.5 Intra-Path Cycle Checking

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

```
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 $\epsilon > 0$.

2.5.6 Inter-Path Cycle Checking

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

```
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)
```

2.6 Informed Search Algorithms

2.6.1 Estimated Cost

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

Example: Some common choices for $\text{ecst}(\cdot)$ include:
1. $\text{ecst}(p) = \text{hur}(p)$; called nearest-first search (NFS)
2. $\text{ecst}(p) = \text{cst}(p) + \text{hur}(p)$; called A^* (A-star)

2.7 Characteristics of an Informed Search Algorithm

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

2.7.1 Heuristics

2.7.2 Designing Heuristics via Problem Relaxation

2.7.3 Combining Heuristics

2.8 Anytime Search Algorithms

2.9 Formulating a Search Problem

2.10 Canonical Examples

Process: How to setup a search problem?

1. 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:

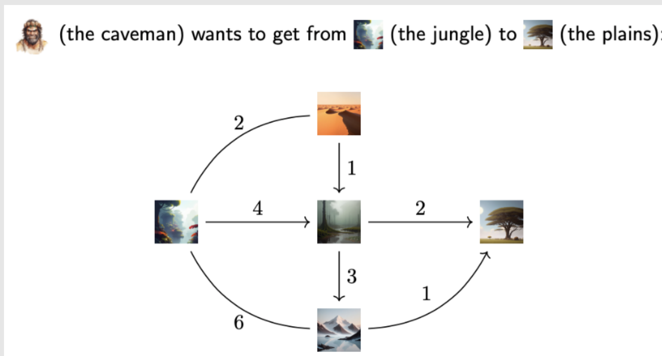


Figure 12

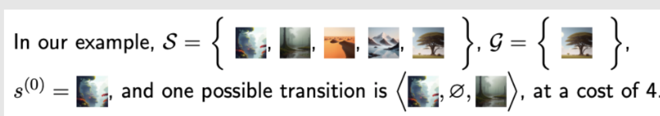


Figure 13

Example:

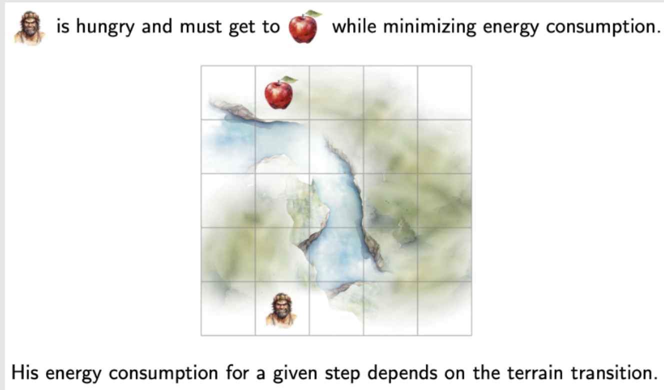


Figure 14

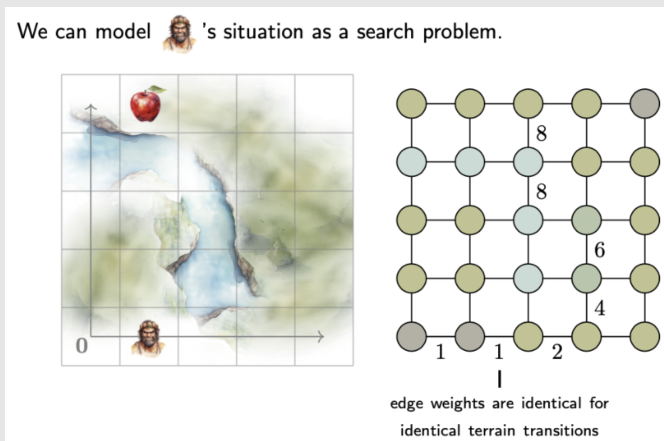


Figure 15

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

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:

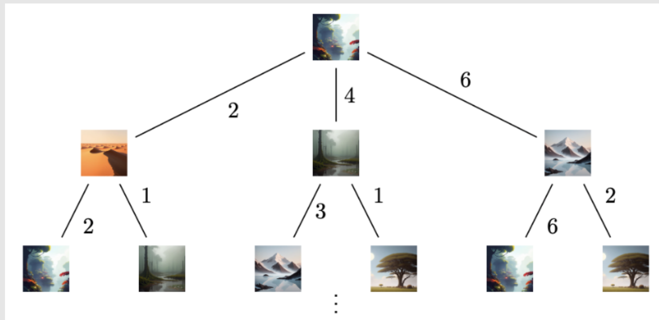


Figure 16

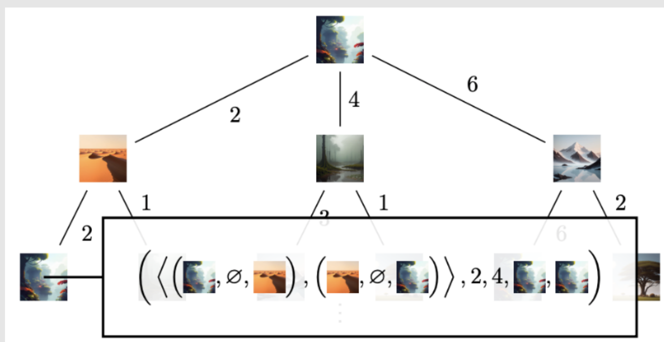


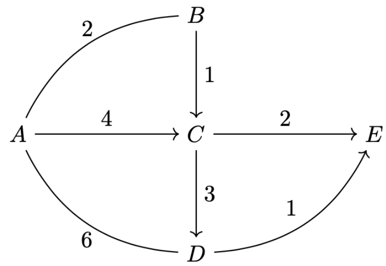
Figure 17

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	$\{AC \mid 4,\ AD \mid 6,\ ABCE \mid 5,\ ABCD \mid 6\}$
$ABCE$	$\{AC \mid 4,\ AD \mid 6,\ ABCD \mid 6\}$

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 $\text{esct}(p) = \text{cst}(p) + \text{hur}(p)$.
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 Figure Out Soln. w/o Performing Search Algorithm?

1.

Example:

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 $\text{hur}(s) \le \text{hur}^*(s)\ \forall s \in \mathcal{S}$. 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 $\le \text{cst}(s, a, \text{tr}(s, a))$ or $\ge \text{rwd}(s, a, \text{tr}(s, a))$. 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:

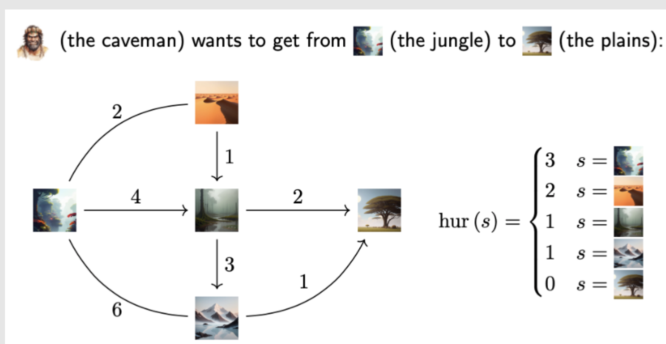


Figure 19: Jungle ($s^{(0)}$), Desert, Swamp, Mountain, Plains (Goal)

Admissible:

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

Consistent:

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

   (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:

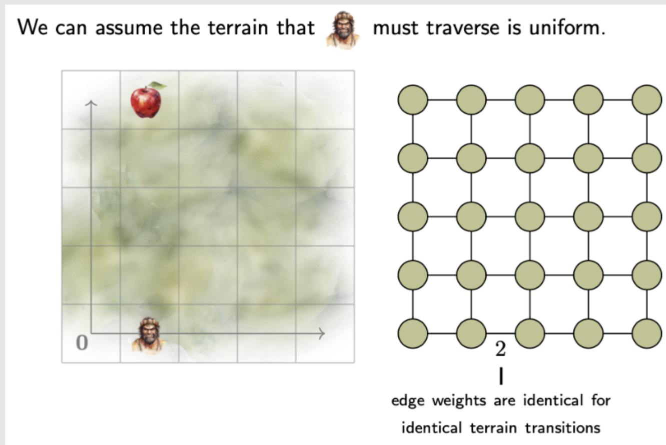


Figure 20

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

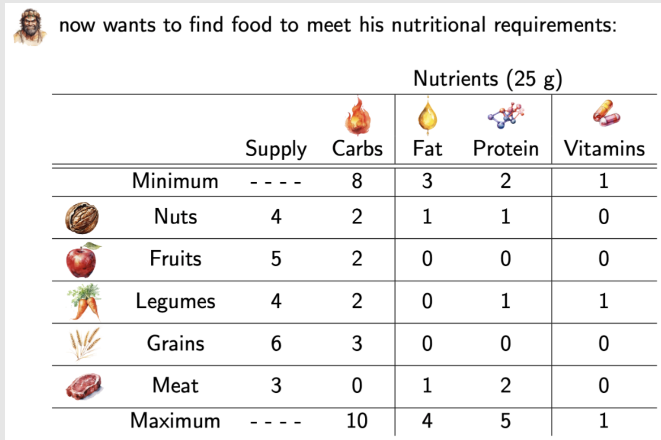
Definition: A CSP is k-**consistent** if for any consistent assignment of $k-1$ variables, $\{(V, v)\}_{V \in \tilde{\mathcal{V}}}$, and any k^{th} variable, V', there is a value, $v' \in \text{dom}(V')$, so the assignment, $\{(V, v)\}_{V \in \tilde{\mathcal{V}}} \cup \{(V', v')\}$ is consistent.

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:

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

Figure 21

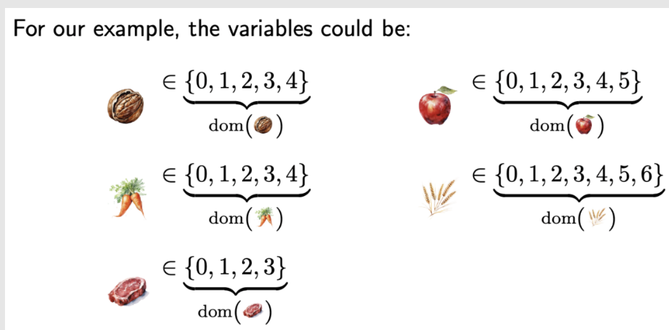


Figure 22

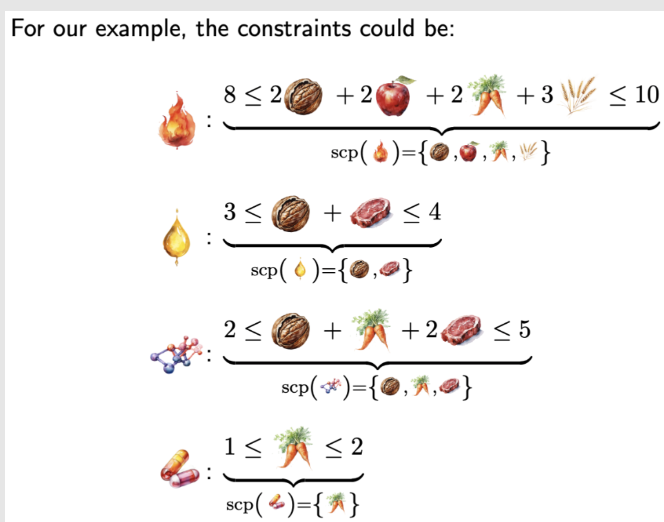


Figure 23

Process: How to build a hyper-graph?

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

Example:

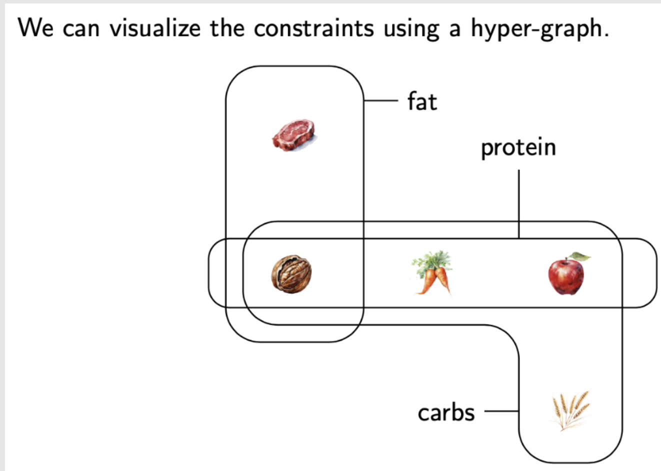


Figure 24

Process: How to build a path tree?

1.

Example:

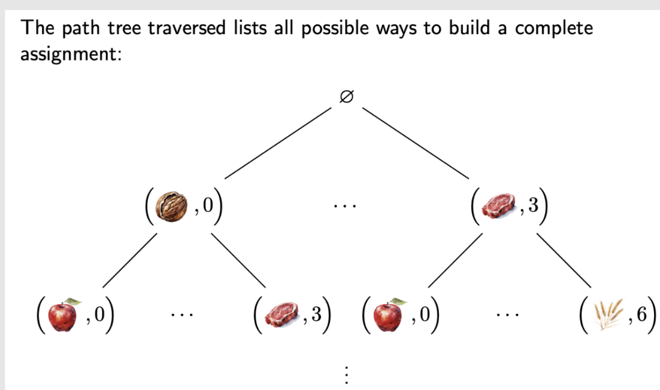


Figure 25

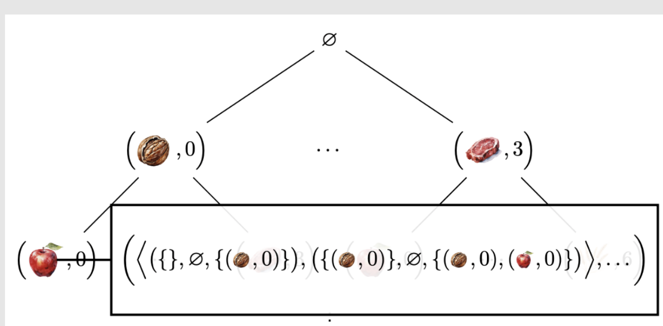


Figure 26

Process: How to determine a solution to a CSP?

1.

Example:

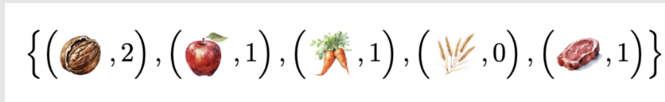


Figure 27

Process: How to check k-Consistency?

1. Given $\mathcal{V}$ w/ $\text{dom}(V) = \{v_1, \ldots, v_{|\text{dom}(V)|}\}\ \forall V \in \mathcal{V}$ and $\mathcal{C}$ w/ $\text{scp}(C) = \{V_1, \ldots, V_{|\text{scp}(C)|}\}\ \forall C \in \mathcal{C}$.
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 the other $V \in \text{scp}(C)$, 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/ $\text{dom}(V) = \{v_1, \ldots, v_{|\text{dom}(V)|}\}\ \forall V \in \mathcal{V}$ and $\mathcal{C}$ w/ $\text{scp}(C) = \{V_1, \ldots, V_{|\text{scp}(C)|}\}\ \forall C \in \mathcal{C}$.
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.

Example:

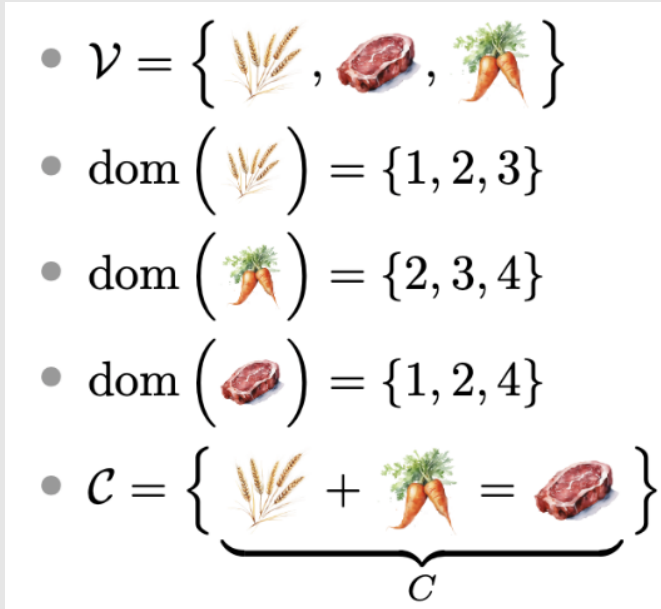


Figure 28

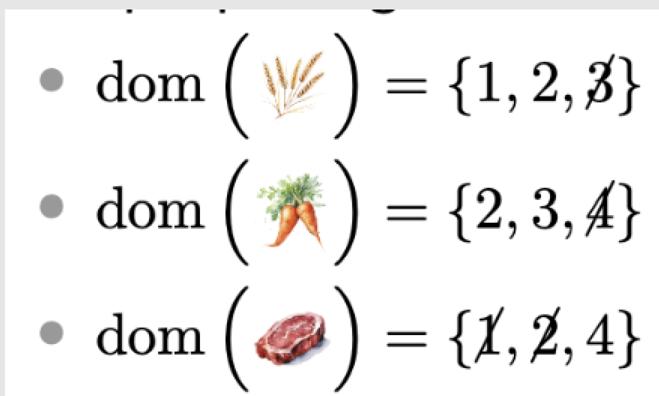


Figure 29: Pre-pruning. Since only one constraint, it is also pruning.

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:

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

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

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 μ denote the probability of an event, and ν denote its relative frequency in a sample size N. Then, for any $\epsilon > 0$,

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

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

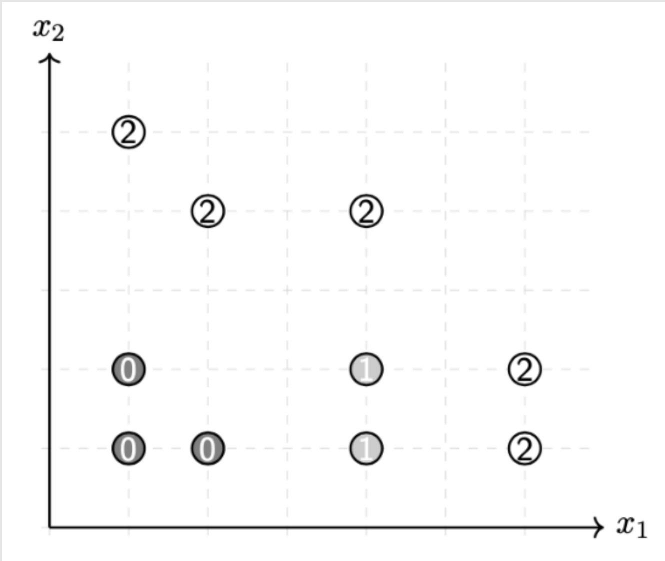


Figure 30

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: $H(X) = \frac{3}{10}\log_2\left(\frac{10}{3}\right) + \frac{2}{10}\log_2\left(\frac{10}{2}\right) + \frac{5}{10}\log_2\left(\frac{10}{5}\right)$
   (b) Entropy after the split:
      i. $H(X_{\text{left}}) = \frac{3}{5}\log_2\left(\frac{5}{3}\right) + \frac{2}{5}\log_2\left(\frac{5}{2}\right)$
      ii. $H(X_{\text{right}}) = \frac{2}{5}\log_2\left(\frac{5}{2}\right) + \frac{3}{5}\log_2\left(\frac{5}{3}\right)$.
      iii. Weighted Avg. Entropy: $H(X|Y) = \frac{5}{10}H(X_{\text{left}}) + \frac{5}{10}H(X_{\text{right}})$
   (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: $H(X) = \frac{3}{10}\log_2\left(\frac{10}{3}\right) + \frac{2}{10}\log_2\left(\frac{10}{2}\right) + \frac{5}{10}\log_2\left(\frac{10}{5}\right)$
   (b) Entropy after the split:

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

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)