CS3243: Introduction to Artificial Intelligence
Lecture Notes 4: Heuristics and Games

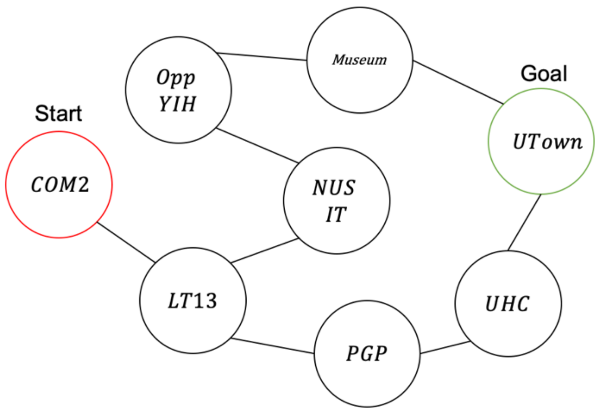
First, we are presented with the following theorems.

- **Theorem.** If h is consistent, then A* with Graph Search is optimal.
- **Theorem.** The Tree Search variant of A* is optimal under an admissible heuristic.

1. Designing Heuristic Functions

Recall that a heuristic h is **admissible** if $h(S_i) \leq OPT(S_i)$, where $OPT(S_i)$ is the optimal path cost from state S_i to the goal.

To make such a heuristic, we should think of a way to relax some of the restrictions that are inherent in the problem, to an extent that $h(S_i)$ is both computable and informative.



For example, refer to the graph above where each state corresponds to a certain location in NUS. Examples of admissible heuristics that are relevant to the graph include:

- **Straight-Line Distance (SLD)**: $h(S_i) = d(S_i, G)$, where $d(S_i, G)$ refers to the Euclidean distance between state S_i and the goal state.

The Straight-Line Distance heuristic is also consistent.

Proof. Based on the triangle inequality, $d(S_i, G) \leq d(S_i, S_{i+1}) + d(S_{i+1}, G)$.
Because SLD is an admissible heuristic, $d(S_i, S_{i+1}) \leq c(S_i, S_{i+1})$.
Therefore, $h(S_i) \leq c(S_i, S_{i+1}) + h(S_{i+1})$, which means that SLD is consistent.

Take note that while consistency implies admissibility, admissibility does not necessarily imply consistency.

- **Zero Heuristic:** $h(S_i) = 0$ for all S_i.

- **Manhattan Distance:** A heuristic that is much better than the SLD heuristic.

2. Greedy Best-First Search

The **Greedy Best First Search** algorithm makes use of only $h(u)$ to decide on which node to pop next from the frontier.

In contrast, the Uniform-Cost Search (UCS) algorithm makes use of $\hat{g}(u)$, and A* search uses $\hat{g}(u) + h(u)$.

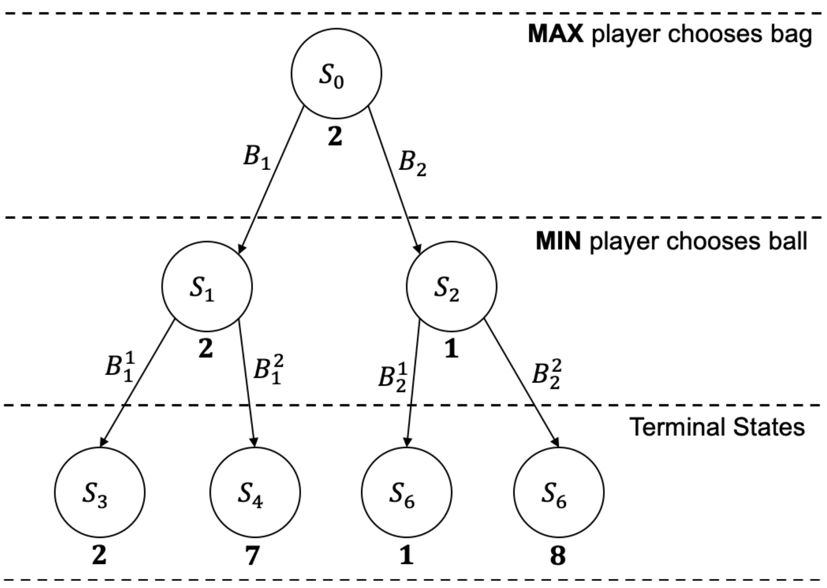
3. Multiple Agents in an Adversarial Situation

Imagine a game played between you and Professor Kuldeep. You have two bags, B_1 and B_2. Each bag contains two balls each.

- $\boldsymbol{B_1^1}$: $Ball\ 1\ in\ Bag\ 1$, with a label '2'
- $\boldsymbol{B_1^2}$: $Ball\ 2\ in\ Bag\ 1$, with a label '7'
- $\boldsymbol{B_2^1}$: $Ball\ 1\ in\ Bag\ 2$, with a label '1'
- $\boldsymbol{B_2^2}$: $Ball\ 2\ in\ Bag\ 2$, with a label '8'

Prof. Kuldeep will proceed to pick a bag, and then you will pick a ball from that chosen bag. Prof. Kuldeep's goal is such that you get a number as big as possible, making him the **MAX** player. Your goal is to get a number as small as possible, making you the **MIN** player.

We can model this game as transition model as shown below.



This model has a utility function $Utility: Terminal\ States \rightarrow \mathbb{R}$ that takes in a state and returns a real number representing the utility at that state.

The following is how we can infer the value of the utility of each state:

- For each of the **terminal states** (S_3, S_4, S_5 and S_6), the utility of each terminal state is equal to the label found on the ball chosen.

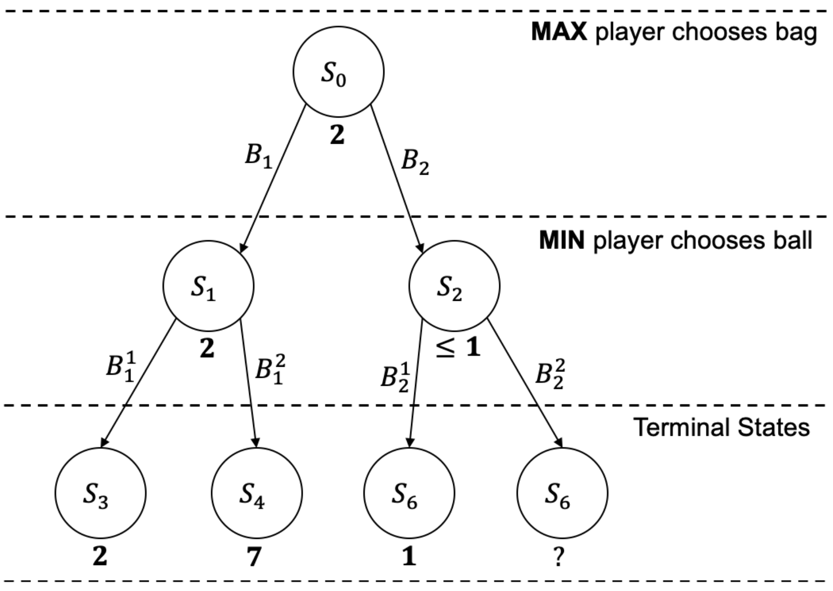
- At state S_1, the utility-minimizing **MIN** player would surely choose to subsequently go to state S_3 which has the lowest utility (2) among the children of S_1. Hence, S_1 will have a utility of 2.
- At state S_2, the utility-minimizing **MIN** player would surely choose to subsequently go to state S_5 which has the lowest utility (1) among the children of S_2. Hence, S_2 will have a utility of 1.

- At the start state S_0, the utility-maximizing **MAX** player will surely choose to go to state S_1 which has the highest utility (2) among the children of S_0. Hence, S_0 will have a utility of 2.

We can hence define the **minimax** value of a state s:

$$MINIMAX(s) = \begin{cases} Utility(s), & if\ Terminal(s) = 1 \\ \max\limits_{a \in Actions} MINIMAX(Result(s,a)), & if\ Player = \boldsymbol{MAX} \\ \min\limits_{a \in Actions} MINIMAX(Result(s,a)), & if\ Player = \boldsymbol{MIN} \end{cases}$$

4. Alpha-Beta Algorithm



Finding the minimax value for every state by examining every single terminal state can be very slow if the graph has a large search space. The number of nodes found at depth d in a graph is equal to bd. A game like Chess has a depth $d = 100$, and a branching factor $b \approx 35$.

We will make use of the **Alpha-Beta Algorithm** instead. We know that the terminal state S_6 has a utility of 1. This means that at its parent state S_2, the **MIN** player will surely choose to go to a state which has a utility of ≤ 1. This means that state S_2 surely has a utility of ≤ 1, which is lower than that of S_1.

In other words, we can **avoid evaluating** the branch taken when choosing ball B_2^2, and yet still be able to compute which bag the **MAX** player would choose from state S_0.