CS3243 : Introduction to Artificial Intelligence

Tutorial 4

NUS School of Computing

February 13, 2022

Review

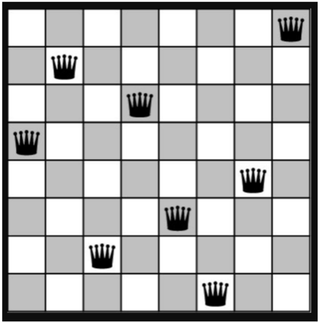
- The problems covered so far dealt with agents having to choose a series of actions to reach the goal state, where the path to the goal also constitutes the solution to the problem.
- In some other problems, the path to the goal is irrelevant as only the end state is required for the solution.
- Local Search

Review

- Board State S : Represents a state of the board with 8 queens at various positions
- Successors of a State : Represents the new board state brought about by an "action" from current state, eg. move a Queen to another position. (different possible "action"s to change the state)
- Value of State $Val(S)$: Should reflect the "quality" of a state, eg. how close it is to the goal state (no queens attack each other)

Review

- For N-Queen problems, $Val(S)$ can be the number of pairs of queens that can attack each other
- Stopping Criteria : Depends on chosen algorithm (eg. Hill Climbing Algorithm w/ steepest ascent, etc)



Review

- The intuition - The "state", "action" and "value/heuristic" representations in local search are different from what we saw in previous search strategies
- Before : We try to build up valid states from an initial empty/partially-filled board until we reach a complete board
- Now : We trial different possible complete boards (since we don't need the path to goal) and check for validity. Use value/heuristic to guide us to the successors that are likely closer to goal
- **Main idea :** Make smart "guesses" of the solution and evaluate

Review

- Variant : Steepest Ascent/Descent
  - Generate an (arbitrary/random) initial state.
  - Continually move to a neighboring state that leads to an increasing (or decreasing) value, until it reaches a local “peak”.
  - Only allows moves to better position $\rightarrow$ May get stuck at local maximum

```
current = initial state
while true :
    neighbor =
    HighestValuedSuccessor(current)

    if (value(neighbor) <= value(current)) :
        return current
    else : current = neighbor
```

Review

- Variant : Steepest Ascent/Descent
  - Generate an (arbitrary/random) initial state.
  - Continually move to a neighboring state that leads to an increasing (or decreasing) value, until it reaches a local “peak”.
  - Only allows moves to better position → May get stuck at local maximum

```
current = initial state
while true :
    neighbor =
    HighestValuedSuccessor(current)

    if (value(neighbor) <= value(current)) :
        return current
    else : current = neighbor
```

- Strategies to mitigate this?

More on Hill Climbing Algorithm

- **Steepest Ascent/Descent** $\rightarrow$ Problem: May get stuck in local maxima, shoulder/plateau, ridges
- How could we escape shoulders? Allow sideways movement $\rightarrow$ **Sideways move**
- How could we try to explore more of different states?
  - Add randomness in selecting next state $\rightarrow$ **Stochastic/First-choice hill-climbing** (First Choice helps with problems with b, branching factor)
  - Repeatedly run Steepest Ascent/Descent with random initial states $\rightarrow$ **Random Restart Hill Climbing**
- What other strategies could we use?
  Local beam search (tracks k best states in a hill-climbing algorithm; better than running k random-restart in parallel due to information sharing)

Tutorial Question 1

- n items $\{a_1, a_2, .., a_n\}$, each with size $s(a_i) > 0$
- m boxes $\{b_1, b_2, .., b_m\}$, each with capacity $c(b_i) > 0$
- Goal : Pack all items into as few boxes as possible

Tutorial Question 1

- Intuition behind local search : How do we “guess” a solution? (State representation)
- How do we “check” its value? (Valuation function)
- How do we generate the successors of each state? (Successor generation strategy)

Tutorial Question 1

- Note: Many ways to formulate a local search problem given different (i) valuation functions and (ii) neighbor generation strategies (state representation is often similar)
- Recall: Our 8-Queens + Hill-climbing example in lecture
- We discuss one possible solution with cost function suggested by **Hyde et al**

  Matthew Hyde, Gabriela Ochoa, T Curtois, and JA Vazquez-Rodrguez. A hyflex module for the one dimensional bin-packing problem. School of Computer Science, University of Nottingham, Tech. Rep, 2009

Tutorial Question 1

- Define notations and functions:
  - $x_{i,j}$: represents the i-th item in the j-th box, $i \in \{1...n\}$ and $j \in \{1...m\}$
  - If the i-th item is packed in j-th box, then $x_{i,j} = 1$. Otherwise, $x_{i,j} = 0$
  - Cost function provided by Hyde et al. (can treat as black box)

$$\text{val}(state) = 1 - \left[\frac{\sum_{j=1}^{m} \left(\frac{\sum_{i=1}^{n} (x_{i,j} \times s(a_i))}{c(b_j)} \right)^2}{m} \right]$$

Tutorial Question 1

- Basic idea: Models the proportion of filled capacity across all boxes (sum all item sizes within each box and divide by capacity)
- Boxes that completely filled or nearly so would give (much) lower values
- Value of the state is a value between 0 and 1, where lower value is better. A set of completely full boxes would return val(state) = 0

Tutorial Question 1

- Assumption : Boxes are sorted in non-increasing (ie. equal or decreasing) order of capacity. Note : This assumption is given in the formalization of the Variable Size Bin Packing Problem (reference), and is needed for this solution.

Note : The cost function proposed by Hyde et al. needs to be paired with the use of First-Fit heuristic in this solution (reference).

The basic idea of the problem formulation:

Initial state:

→ Generate random sequence of items.

→ Pack each item into boxes using *First Fit* heuristic.

- For each item, attempt to pack in the first (sorted) box that can fit.

→ Evaluate `val(initial_state)` using the cost function defined.

Finding next state:

→ Randomly choose an occupied box from *current_state*.

→ *next_state* ← Remove items from the box and repack them into existing boxes using *First Fit* heuristic.

→ Evaluate `val(next_state)` using the cost function defined.

Stopping criteria:

→ If `val(next_state)` ≤ `val(current_state)`: Set *current_state* = *next_state* and repeat.

- Recall: Lower val is better

→ If `val(next_state)` > `val(current_state)`: Trial other next_states. If all next_states exhausted, terminate process and return *current_state* as solution.

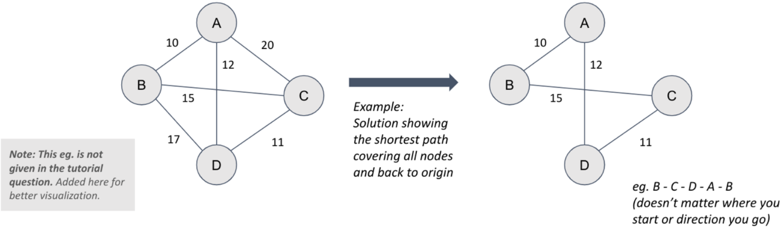
Tutorial Question 2

- Traveling Salesman Problem (TSP)
- Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once, and returns to the origin city?

Tutorial Question 2

- Traveling Salesman Problem (TSP)
- Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once, and returns to the origin city?
- Now, this is a hard problem. We haven't been able to find a polynomial time algorithm for it. But, we have heuristics to the rescue.

Example TSP Input and Output



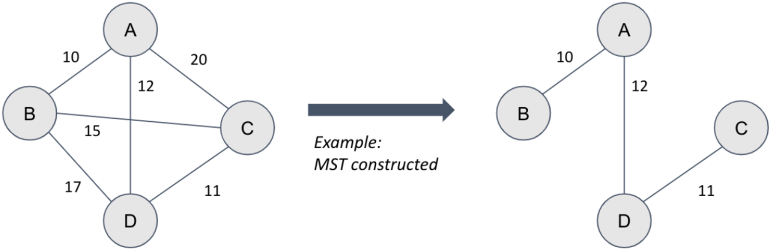
Tutorial Question 2(a)

- How can MST heuristic be derived from a relaxed version of TSP?

Tutorial Question 2(a)

- How can MST heuristic be derived from a relaxed version of TSP?
- The TSP problem is to find a minimal length path through the cities, which forms a closed loop.
- MST is a relaxed version of the TSP problem because it only seeks for a minimal length graph that need not be a closed loop – it can be any fully-connected graph.

Example MST



Tutorial Question 2(b)

- Is the MST heuristic admissible?

Tutorial Question 2(b)

- Is the MST heuristic admissible?
- Yes, because it is always shorter than or equal to the length of a closed loop through the cities.

Tutorial Question 2(c)

- How to start?
- Think about how we can represent state, generate successors and value(s)
- State : Recall that Hill-Climbing iterates over possible solutions (until it reaches a “peak” over its successors)
- What could possible solutions look like? (“complete states”)
- What could value(s) be? – Total distance of each route
- How can we generate successors from a state? – Many ways

Tutorial Question 2(c)

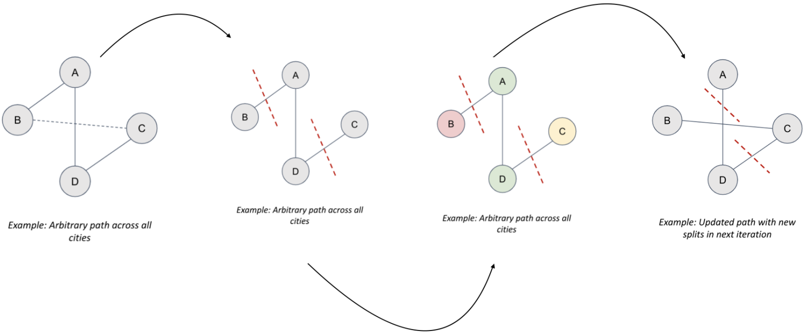
- Hill Climbing Approach to solve TSP?

Tutorial Question 2(c)

- Hill Climbing Approach to solve TSP?
- Algorithm:
  - Choose a random path that connects all the cities
  - Along that path, pick two cities at random
  - Partition the whole path across these two cities, which leaves us with 3 paths
  - Try all 6 possible permutations of arranging the partitions to obtain the least cost loop
  - Upon getting the best cost, repeat the steps to further improve
  - Halt when done with a certain k number of iterations

Tutorial Question 2(c)

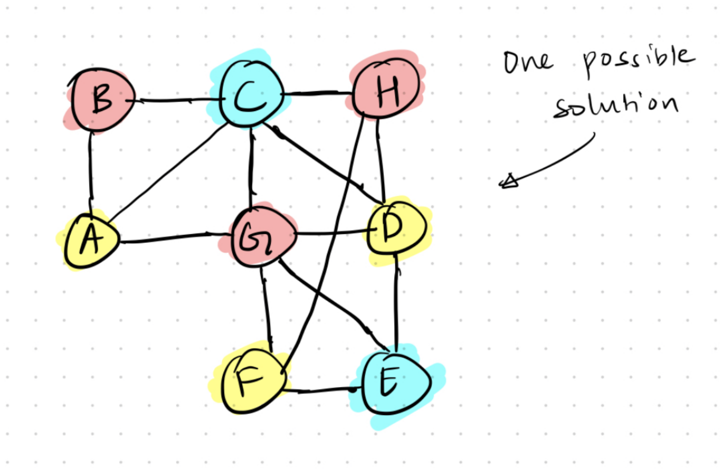
- Pictorial Trace



Tutorial Question 4

- Given : A graph G
- Goal is to find a colouring of the set of vertices using only the colours RED, YELLOW, and BLUE, so that no two adjacent vertices are assigned the same colour

Tutorial Question 4(a)



Tutorial Question 4(b)

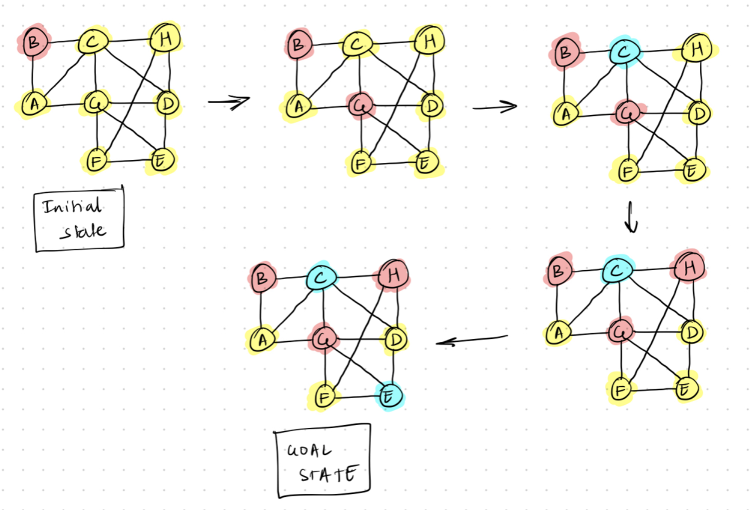
- State representation?
  A complete assignment of colors to vertices, eg. given starting state
- Evaluation function?
  Given in question
- Successor generation strategy?
  Given in question

Tutorial Question 4(b)

- Algorithm
  - Generate all possible successors of start state given action defined
  - Compute $f(state)$ of each successor
  - Pick the successor w/ the lowest cost
  - Repeat until stopping criteria met

Tutorial Question 4(b)

- Pictorial Trace



Tutorial Question 4(b)

- Value Trace

Steps :

Actions :	$C_G = \text{Red}$	$C_C = \text{Blue}$	$C_H = \text{Red}$	$\boxed{C_E = \text{Blue}}$
f(state) :	7	4	2	$\boxed{0}$

$\downarrow$ goal state achieved

As the algorithm proceeds, f(state) decreases.

Thank you!

If you have any questions, please don't hesitate. Feel free to ask!
We are here to learn together!