CS3243 Tutorial 4

Eric Han

Sep 12, 2022

Annoucements

1. Mark your attendance, by checking in on telegram
2. Assignment 2 scores and comments are out on Luminus.
   - If you have any queries, please come forward.
   - There should be comments given for marks deducted, if you are having problems seeing the comments please come forward.
3. Congrats on making it to Week 6 - next week is Reading week, but I will be avaliable for consultations on Tuesdays.
   - Tuesday 1-4pm, F2F, Telegram/Email me before coming!

Previously from T03, Q3

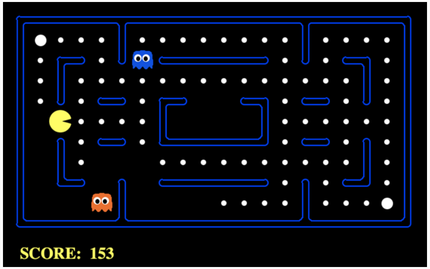


Figure 1: Pac-Man Example

- **State representation**: Position of Pac-Man and the positions of the uneaten pellets
- **Initial State**: Filled grid entirely with pellets
- **Goal State**: No pellets left in the grid
- **Action**: Moving up/down/left/right
- **Transition Model**: Updating the position of Pac-Man and eating pellet (if applicable)
- **Cost function**: 1 for each action taken

Determine the admissibility of the heuristics:

- h_1 : Number of pellets left at any point in time.
- h_2 : Number of pellets left + the minimum among all Manhattan distances from each remaining pellet to the current position of Pac-Man.
- h_3 : The Maximum among all Manhattan distances from each remaining pellet to the current position of Pac-Man.
- h_4 : The average over all Euclidean distances from each remaining pellet to the current position of Pac-Man.

Determine the admissibility of the heuristics:

- h_1 : Number of pellets left at any point in time.
- h_2 : Number of pellets left + the minimum among all Manhattan distances from each remaining pellet to the current position of Pac-Man.
- h_3 : The Maximum among all Manhattan distances from each remaining pellet to the current position of Pac-Man.
- h_4 : The average over all Euclidean distances from each remaining pellet to the current position of Pac-Man.

Answer
h^*: Optimal number of actions for Pac-Man to consume all pellets.

- h_1 - Admissible, each pellet requries at least one move.
- h_2 - Inadmissible, 1 pellet left and 1 step away; $h_2 = 2 > h^*$
- h_3 - Admissible, $\max_{p \in P}$ path over all pellets ensure that $\leq h^*$. ie. furthest
- h_4 - Admissible, $h_4 \leq h_3 \leq h^*$

Question 1

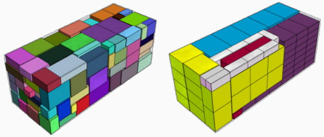


Figure 2: Packing items in 3D

Pack $\{a_1, \cdots a_n\}$ items with sizes $s(a_i) > 0$ into $\{b_1, \cdots b_m\}$ boxes with sizes $c(b_i) > 0$, where $\sum_{i=1}^{m} c(b_i) > \sum_{j=1}^{m} s(a_j)$. Goal is to pack into as few boxes as possible.

Recap

- What is this problem called in the literature? What's its complexity?
- In which industry this is actually useful?
- What are the items needed for local search problem formulation?

Question 1

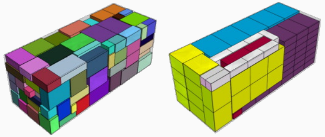


Figure 2: Packing items in 3D

Pack $\{a_1, \cdots a_n\}$ items with sizes $s(a_i) > 0$ into $\{b_1, \cdots b_m\}$ boxes with sizes $c(b_i) > 0$, where $\sum_{i=1}^{m} c(b_i) > \sum_{j=1}^{m} s(a_j)$. Goal is to pack into as few boxes as possible.

Recap

- What is this problem called in the literature? What's its complexity?
- In which industry this is actually useful?
- What are the items needed for local search problem formulation?

Bin packing is NP-Complete, and is a open problem in logistics.

Answer

The value of each state is $\text{val}(s)$:

1. Hyde et al as seen in solutions
2. Sum of used boxes: $\sum_{b \in B_{\text{used}}} c(b)$

Assume boxes sorted in non-increasing order.

- **Inital state**: Pack a random sequence of items into boxes, via *First Fit*.
  - For every item, we find the first box that can accommodate it.
- **Next state**: Generate the next state s' by
  - For any of the filled i-th boxes in the current state, empty the box and fit its items via *First Fit*.
- **Stopping criteria**: Optima is found, ie. $\text{val}(s)$ is better than next states $\text{val}(s')$.

We can use hill-climbing with Random Restarts.

Bonus Qn: Why do we want to use such a complicated cost function?

Question 2

Travelling Salesman Problem: 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?

It can be solved with the minimum spanning tree (MST) heuristic, which estimates the cost of completing a tour through all the cities, given that a partial tour has already been constructed.

The MST cost of a set of cities is the sum of the distance between two cities of any minimum spanning tree that connects all the cities.

Recap

- What is Minimum Spanning Tree?
- What does MST finds?

Question 2a

Show how this heuristic can be derived from a relaxed version of the TSP.

Question 2a

Show how this heuristic can be derived from a relaxed version of the TSP.

Answer
Relax assumption that each city have to visited exactly once, need not be a closed loop.

MST will find a tour for any fully connected graph, and we can use that fully connected graph to find a feasible tour.

Question 2b

Determine whether this heuristic is an admissible heuristic.

Question 2b

Determine whether this heuristic is an admissible heuristic.

Answer
MST is admissible as it finds a tree of minimum total weight. But it is not a closed loop, so the heuristic is always $\leq$ than length of closed loop.

Question 2c

Suggest a hill-climbing algorithm to solve TSP.

Question 2c

Suggest a hill-climbing algorithm to solve TSP.

Answer
val(s) = sum of distances of the closed loop s.

- **Inital state**: Randomly chosen closed loop.
- **Next state**: Generate the next state s' by
    - Swapping 2 cities
- **Stopping criteria**: Depending on the size of the problem,
    - Optima is found, ie. val(s) is better than next states val(s').
    - Iterate the steps above until no improvement is observed for k iterations.

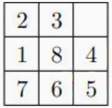
We can use hill-climbing with Random Restarts.

Question 3

Assignment Question; we will go through this question next week.

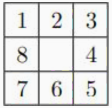
FAQ - questions on mismatched

$f(\text{state})$ = number of mismatched tiles compared to the goal state



2	3	
1	8	4
7	6	5

Figure 3: Q3a inital



1	2	3
8		4
7	6	5

Figure 4: Q3a goal

So, $f(\text{inital}) = 4$

- With a left move, $f(\text{next state}) =$
- With a down move, $f(\text{next state}) =$

Question 4

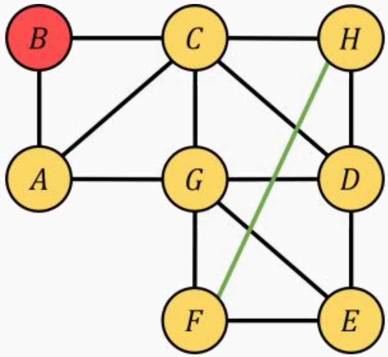


Figure 5: Q4 Example

The goal is to find a colouring of the set of vertices using only the colours:

1. Red
2. Yellow
3. Blue

No 2 adjacent vertices are assigned the same color.

Recap

- What is the graph coloring problem?
- How do we know there are no solutions?

Applications? - Not very much practical applications (ie. outside math/cs): Scheduling (CS), Map coloring, Cryptography (Math)

Question 4a

Give an example of a solution state for the graph G if it exists.

Answer

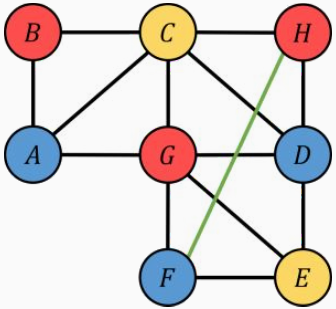


Figure 6: Q4a Solution

Question 4b

Cost function associated with every state:

$$f(s) = \text{number of pairs of adjacent vertices with the same colour}$$

- **Inital state**: As seen in Q4 Example.
- **Next state**: Generate the next state s' by
  - Changing color of a single vertex.
- **Stopping criteria**: Optima is found, ie. $f(s)$ is better than next states $f(s')$.

We use hill-climbing (steepest descent) algorithm.

Note that this formulation can be thought of as 'construction', but it can also be viewed as every s is a possible coloring.

Answer

Intuition: Color the node that will reduce the f maximally.

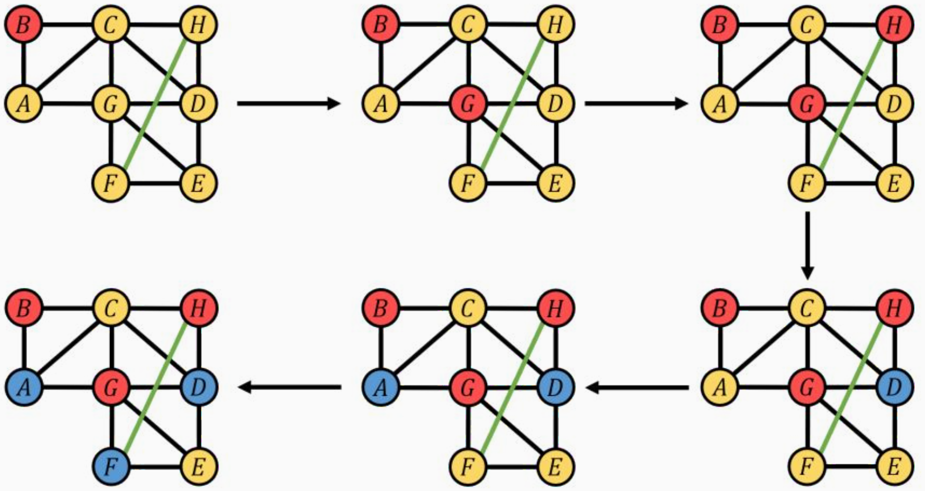


Figure 7: Q4b Solution, also possible - E,C

Bonus Question - Work for Snack

To help you further your understanding, not compulsory.

Tasks

1. Fork the repository https://github.com/eric-vader/CS3243-2223s1-bonus
2. Look for `tutorial4.py`, we will be solving Q3a using code.
3. Some boilerplate code is given, implement everywhere TODO is mentioned.

To claim your snack, show me your forked repository and your code's output.