Suggestions from the Project 2.2 (Local Search) Consultations

CS3243 AY24/25 Semester 1

Overview

- Hill Climbing formulation
- Optimizing Random Restart
- Others

Disclaimer:

Besides the requirement that you have to implement a local search (hill-climbing in this case) to solve this problem, everything else suggested in the subsequent slides is just one way among many to optimize, it is not meant to be the only way, I believe you can do even better :)

Problem modelling

State representation:

- [subset_1, subset_2, subset_3] ?

Is the above formulation efficient enough? Can we do any better?

Efficient state representation:

- How about: [[subset_1, sum_of_subset_1], [subset_2, sum_of_subset_2], [subset_3, sum_of_subset_3], …]
- It calculates the sum of each subset in preprocessing step, and later when we need to update the subset, the updated sum can be obtained by: current_sum + (value_added – value_removed), so that we do not have to iterate through the entire updated subset again to find the sum again.

Problem modelling

Locating an initial state: Can we always locate an initial position that leads to the *global optima*? (probably can find those positions in a brute force way, but that is inefficient and it deviates from the point of this assignment…)

Hint: Random restart! But how to do it efficiently? (next slide)

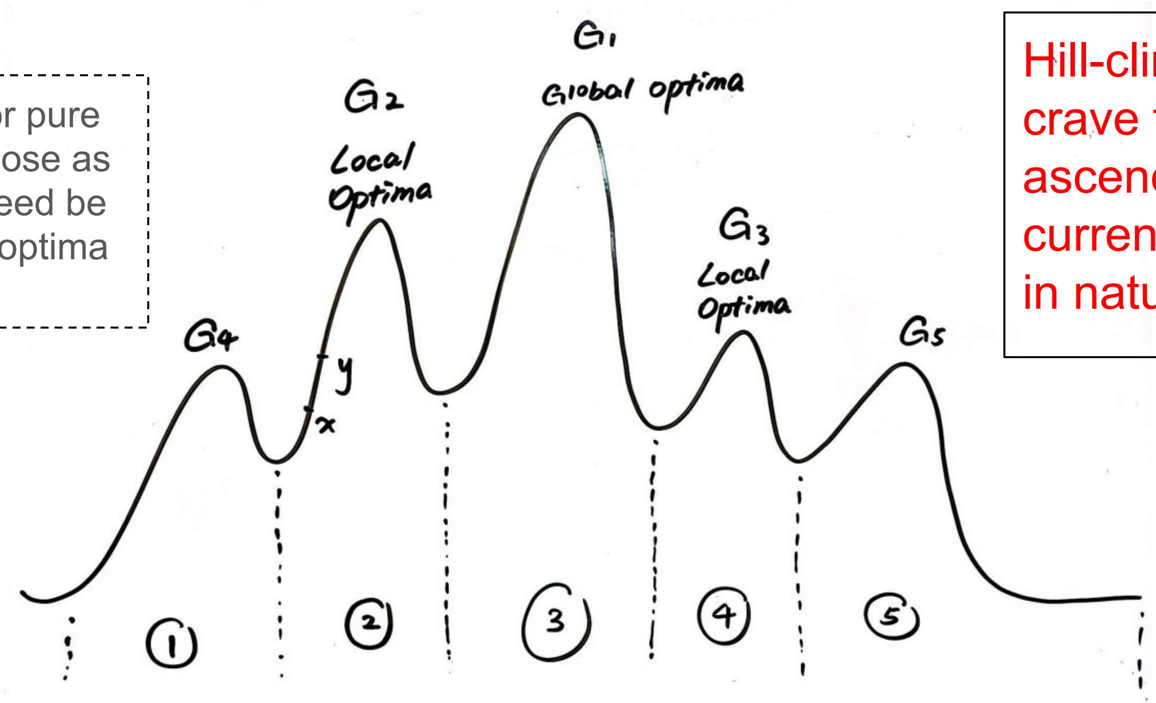
Finding neighbors: Perform a swap between two subsets. (Formally, given S1 = {a1, a2, a3, …} and S2 = {b1, b2, b3, …}, a swapping refers to the action that we remove ai from S1, insert it into S2; remove bj from S2 and insert it into S1)

Hint: How to store a swapping action efficiently? (a tuple of length 4)

Heuristic function: It should generally be a function that measures the deviation of subset sum from the average sum. (this function can be in O(n))

Why random restart works?

The image is for pure illustration purpose as there could indeed be multiple global optima in this project

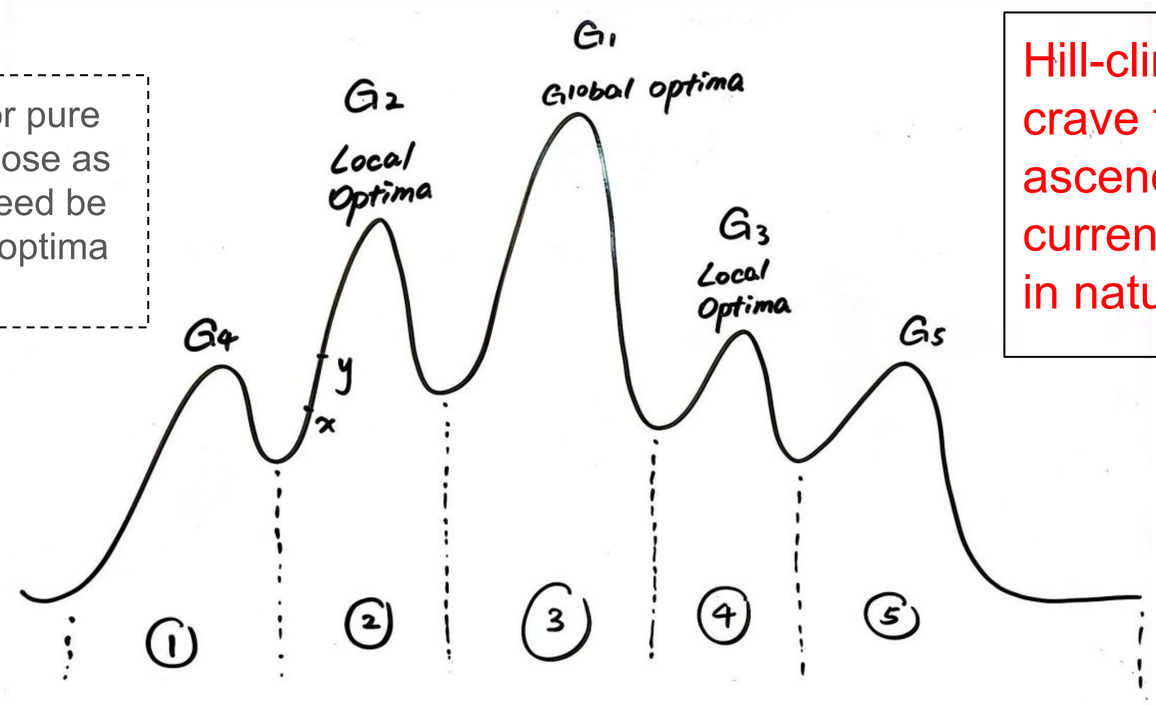


Hill-climbing always crave for the steepest ascend given the current locality (greedy in nature)

Consider one local search formulation as above. **Randomly selecting a point is equivalent to randomly selecting a region.** For example, if you start at x as indicated in region 2, since it always find the steepest ascend, it will deterministically always terminate at G2, it will not somehow terminate at G1, G3, G4, G5. (assuming the ascend is not stochastic)

How does random restart work?

The image is for pure illustration purpose as there could indeed be multiple global optima in this question

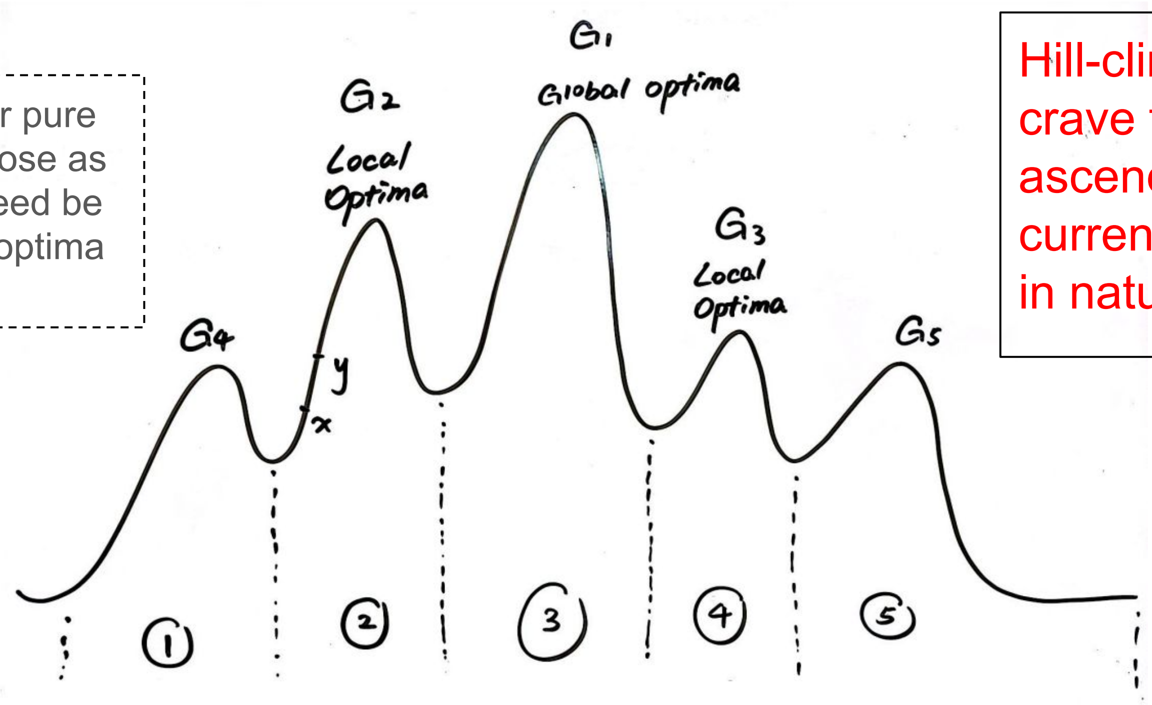


Hill-climbing always crave for the steepest ascend given the current locality (greedy in nature)

Suppose if we start from 1/n portion of the points, it will lead to the global optima. (In this case, it corresponds to any point inside region 3), then the chance of selecting a position that ended up with local optima is (1 - 1/n).

How does random restart work?

The image is for pure illustration purpose as there could indeed be multiple global optima in this question



Hill-climbing always crave for the steepest ascend given the current locality (greedy in nature)

So the chance of doing m restarts to end up with the global optima, (first (m-1) time restarts all failed, how sad): (1 - 1/n)^m, as m gets moderately large, the chance converges to 0. (since 1 - 1/n is strictly smaller than 1)

But can we speed up our random restart?

Time for the algorithm to terminate at the global optima = event_A * event_B

- Event_A: Find an optima (be it either a local or global)
- Event_B: The number of restarts needed on average. (As discussed in the previous slides, event_B is not something we can control about, as we have no further domain knowledge to determine which positions correspond to the slope of global optima)

Time for Event_A can be further decomposed into Event_A1 * Event_A2:

- Event_A1: Find the best neighbour among all neighbors.
- Event_A2: The number of iterations needed to reach the optima of the current region.

Event_A2 is something that we can optimize!

More Problem Modelling

Optimizing event_A2 in this question:

- **Minimize the number of iterations needed to reach the optima**: Find the best swapping that minimizes the deviation from the average sum ⇒ which maximizes the jump size ⇒ which minimizes the num of iterations. (See next slide for further elaborations)
- **Construct a "good" initial position**: We can try to start from a position that is already not too far away from the optima. *However, we still need to ensure that this position is random enough such that it does not end up always in the same slope*. (The functions random.choice(), random.shuffle() from the random library could be potentially helpful)

How to find the best swap?

- Does the two values to be swapped have to come from the biggest and the smallest subset? **Hint**: No, generally it does not have to, though it depends on which heuristic function that has been implemented.
- What would be a necessary condition for a swap to reduce the deviation? **Hint**: The two values have to come from two subsets such that one subset's current sum is bigger than the average sum, while the other subset's sum is smaller than the average sum.
- It can be achieved in **O(M^2 * N^2)** time (though we can always do better), where **M** is the number of subsets and **N** is the size of the subset.

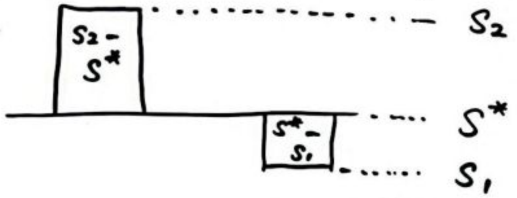
$S^* :=$ targeted average sum.

$S_1 \rightarrow S_1' = S_1 - a_i + b_j$ (remove a_i from S_1, add b_j to S_1)

$S_2 \rightarrow S_2' = S_2 + a_i - b_j$. (remove b_j ..., add a_i ...)

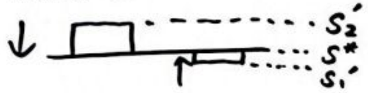
WLOG. assuming $|S_2 - S^*| \geq |S_1 - S^*|$ and $S_2 > S^* > S_1$

it corresponds to the diagram that

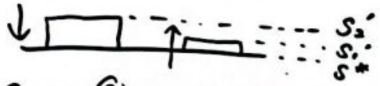


$\Longrightarrow$ 4 cases

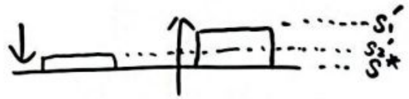
Case ①



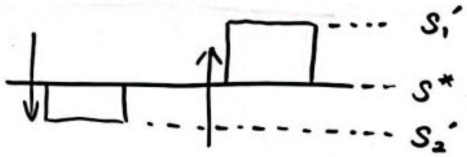
Case ②:



Case ③:



Case ④:



The above shows four potential cases when performing a swapping. However, do we have to iterate through the entire set of subsets to compute the new heuristic value?

We can directly update the current heuristic function by considering the change in heuristic value, so that we do not have to iterate through all subsets to update the heuristic function.

Other potentially useful hints

Some of the ideas might only be needed for bonus test cases, you need not worry about those otherwise.

- Sort() function can be used in initialization to further reduce the time needed to find the best pair to swap
- Besides initial condition, could we also incorporate stochasticity into the tie breaker condition?
- K beam? Will it actually be better?

All the best!