Photo from: http://cdn.motocross.transworld.net/files/2009/11/img-7532.jpg

Hill Climbing

Leandro L. Minku

Hill-Climbing

Hill-Climbing (assuming maximisation)

1. current_solution = generate initial candidate solution randomly

2. Repeat:

    2.1 generate neighbour solutions (differ from current solution by a single element)

    2.2 best_neighbour = get highest quality neighbour of current_solution

    2.3 If quality(best_neighbour) <= quality(current_solution)

        2.3.1 Return current_solution

    2.4 current_solution = best_neighbour

Designing Representation, Initialisation and Neighbourhood Operators

Hill-Climbing (assuming maximisation)

1. current_solution = generate initial solution randomly

2. Repeat:

2.1 generate neighbour solutions (differ from current solution by a single element)

2.2 best_neighbour = get highest quality neighbour of current_solution

2.3 If quality(best_neighbour) <= quality(current_solution)

2.3.1 Return current_solution

2.4 current_solution = best_neighbour

- Representation:
  - How to store the design variable.
  - E.g., boolean, integer or float variable or array.
- Initialisation procedure:
  - Usually involve randomness.
- Neighbourhood operator:
  - How to generate neighbour solutions.

Illustrative Example

- Design variables represent a candidate solution.
  - $x \in Z$
  - Our search space are all integer numbers.

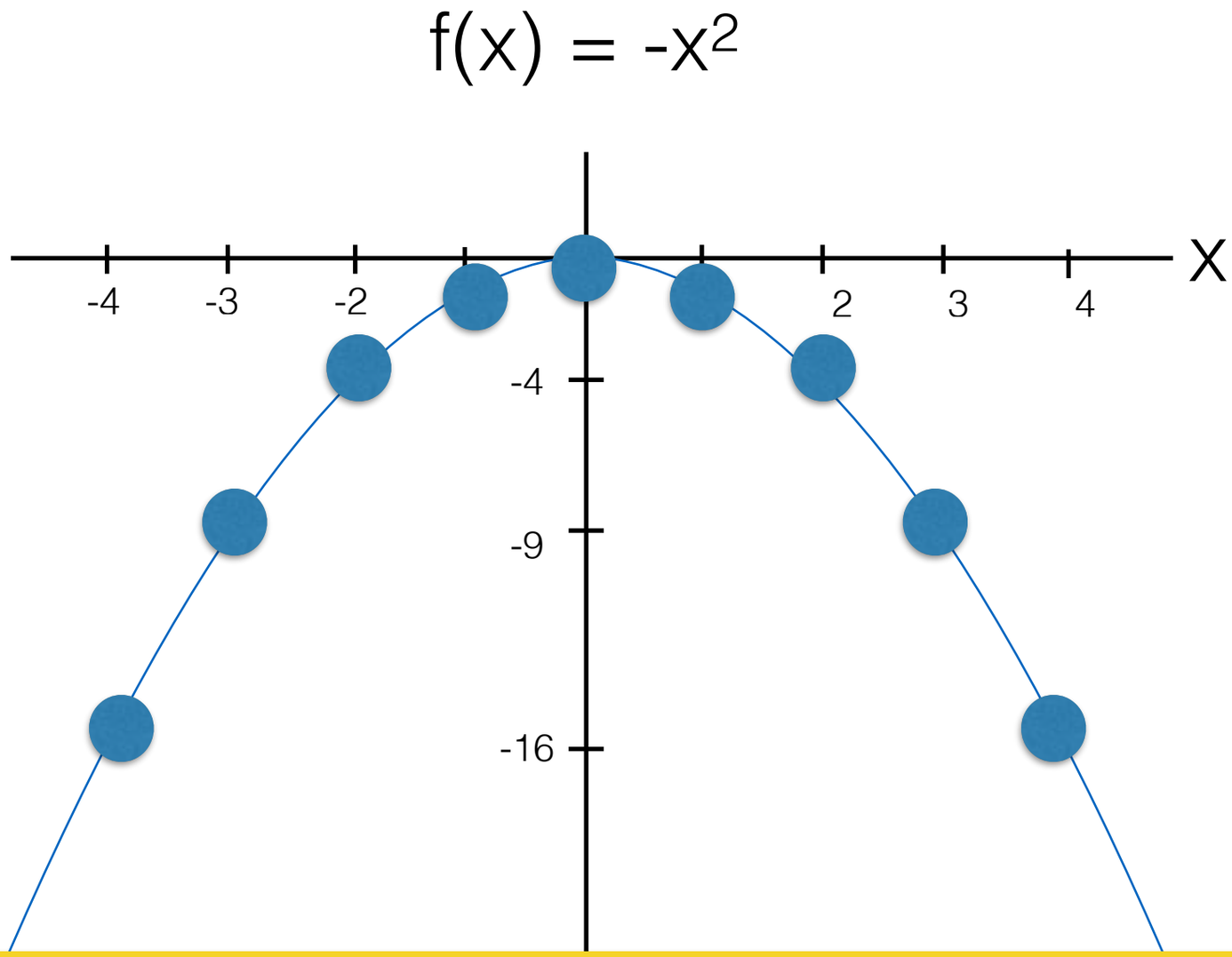
- [Optional] Solutions must satisfy certain constraints, which define solution feasibility.
  - None

- Objective function defines the quality (or cost) of a solution.
  - $f(x) = -x^2$, to be maximised

Illustrative Example

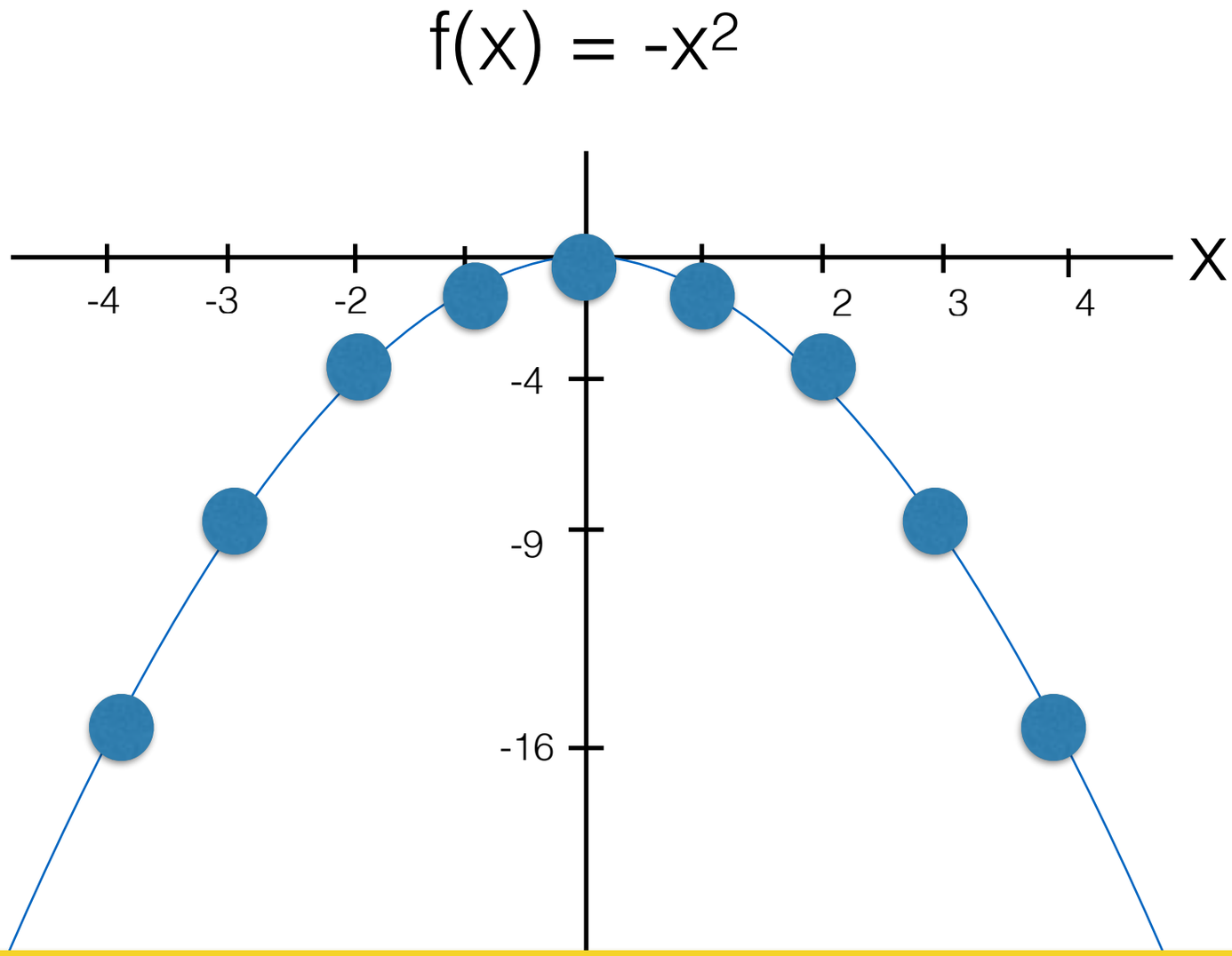
- Representation:
  - Integer variable.
- Initialisation procedure:
  - Initialise with an integer value picked uniformly at random.
- Neighbourhood operator:
  - Add or subtract 1.

Illustrative Example



This is an illustrative example to understand the behaviour of the algorithm. In reality, we are unlikely to know the shape of our function to be optimised beforehand.

Illustrative Example



Other problems may have more dimensions,
and more neighbours for each candidate solution.

Illustrative Example

Hill-Climbing (assuming maximisation)

1. current_solution = generate initial solution randomly

2. Repeat:

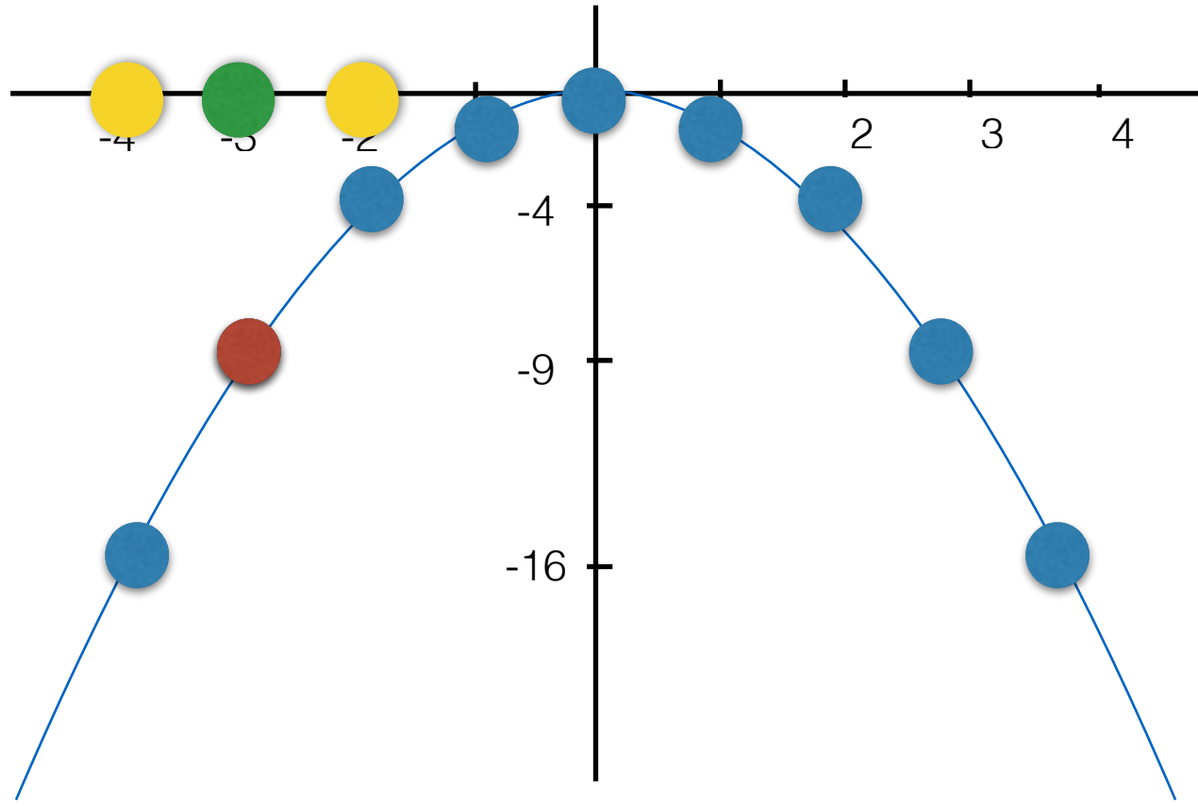
    2.1 generate neighbour solutions (differ from current solution by a single element)

    2.2 best_neighbour = get highest quality neighbour of current_solution

    2.3 If quality(best_neighbour) <= quality(current_solution)

        2.3.1 Return current_solution

    2.4 current_solution = best_neighbour



Illustrative Example

Hill-Climbing (assuming maximisation)

1. current_solution = generate initial solution randomly

2. Repeat:

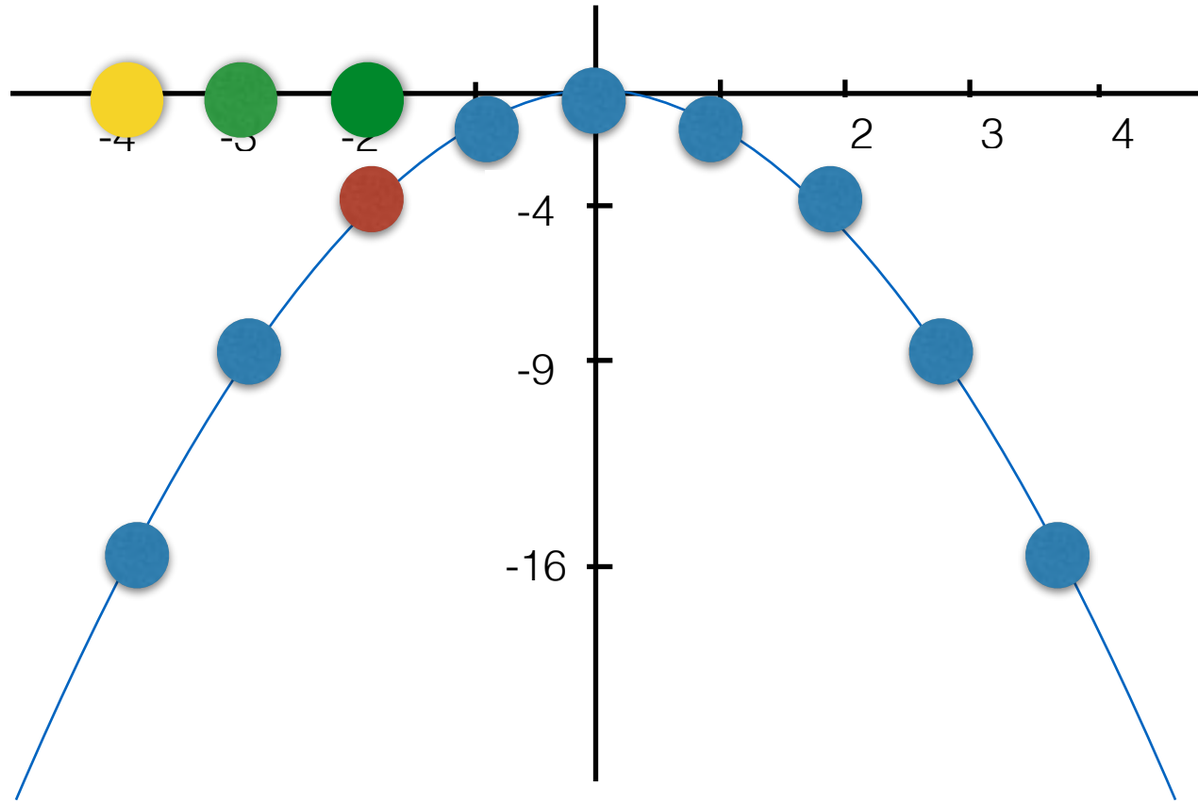
    2.1 generate neighbour solutions (differ from current solution by a single element)

    2.2 best_neighbour = get highest quality neighbour of current_solution

    2.3 If quality(best_neighbour) <= quality(current_solution)

        2.3.1 Return current_solution

    2.4 current_solution = best_neighbour



Illustrative Example

Hill-Climbing (assuming maximisation)

1. current_solution = generate initial solution randomly

2. Repeat:

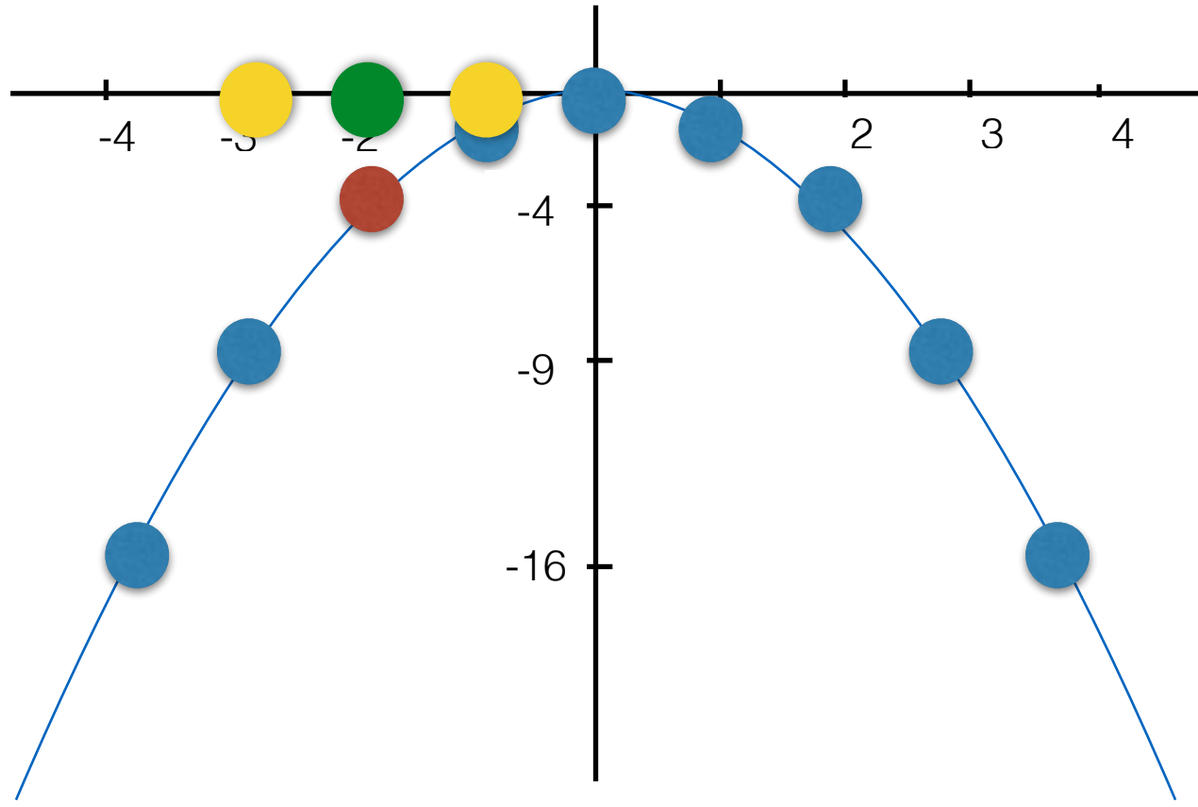
    2.1 generate neighbour solutions (differ from current solution by a single element)

    2.2 best_neighbour = get highest quality neighbour of current_solution

    2.3 If quality(best_neighbour) <= quality(current_solution)

        2.3.1 Return current_solution

    2.4 current_solution = best_neighbour



Illustrative Example

Hill-Climbing (assuming maximisation)

1. current_solution = generate initial solution randomly

2. Repeat:

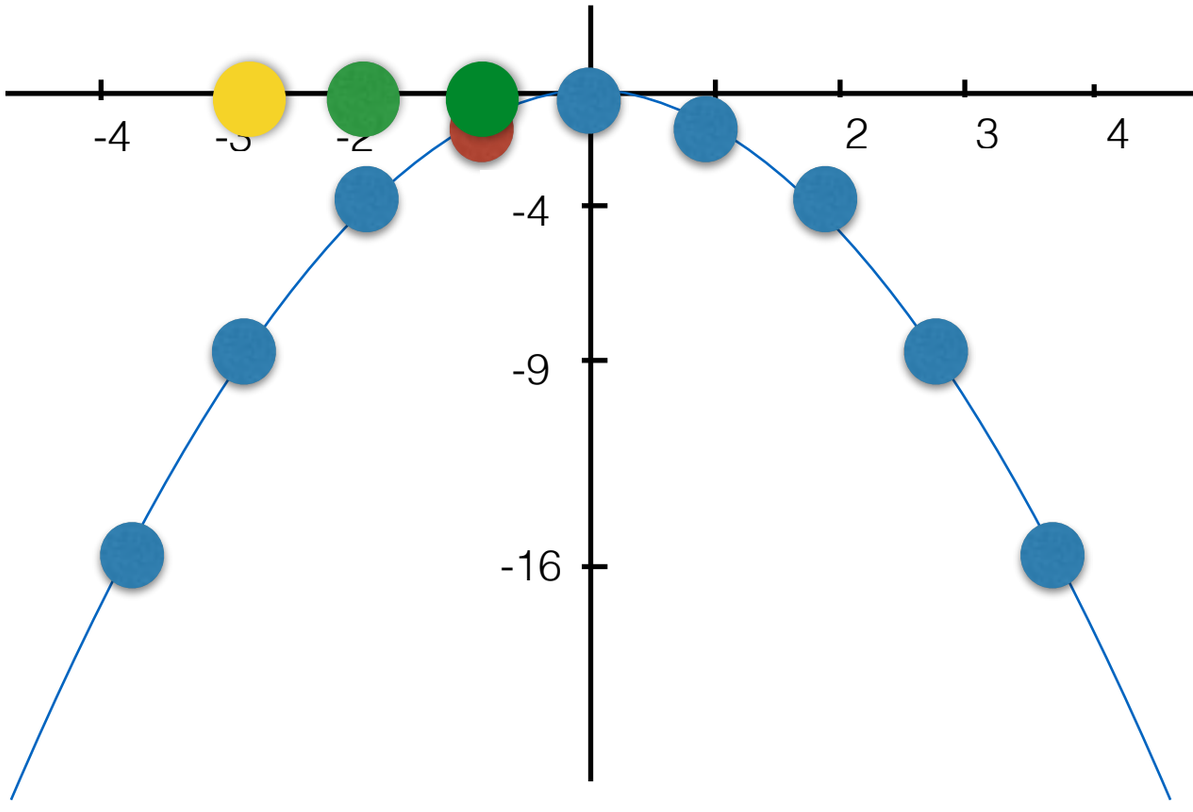
2.1 generate neighbour solutions (differ from current solution by a single element)

2.2 best_neighbour = get highest quality neighbour of current_solution

2.3 If quality(best_neighbour) <= quality(current_solution)

2.3.1 Return current_solution

2.4 current_solution = best_neighbour



Illustrative Example

Hill-Climbing (assuming maximisation)

1. current_solution = generate initial solution randomly

2. Repeat:

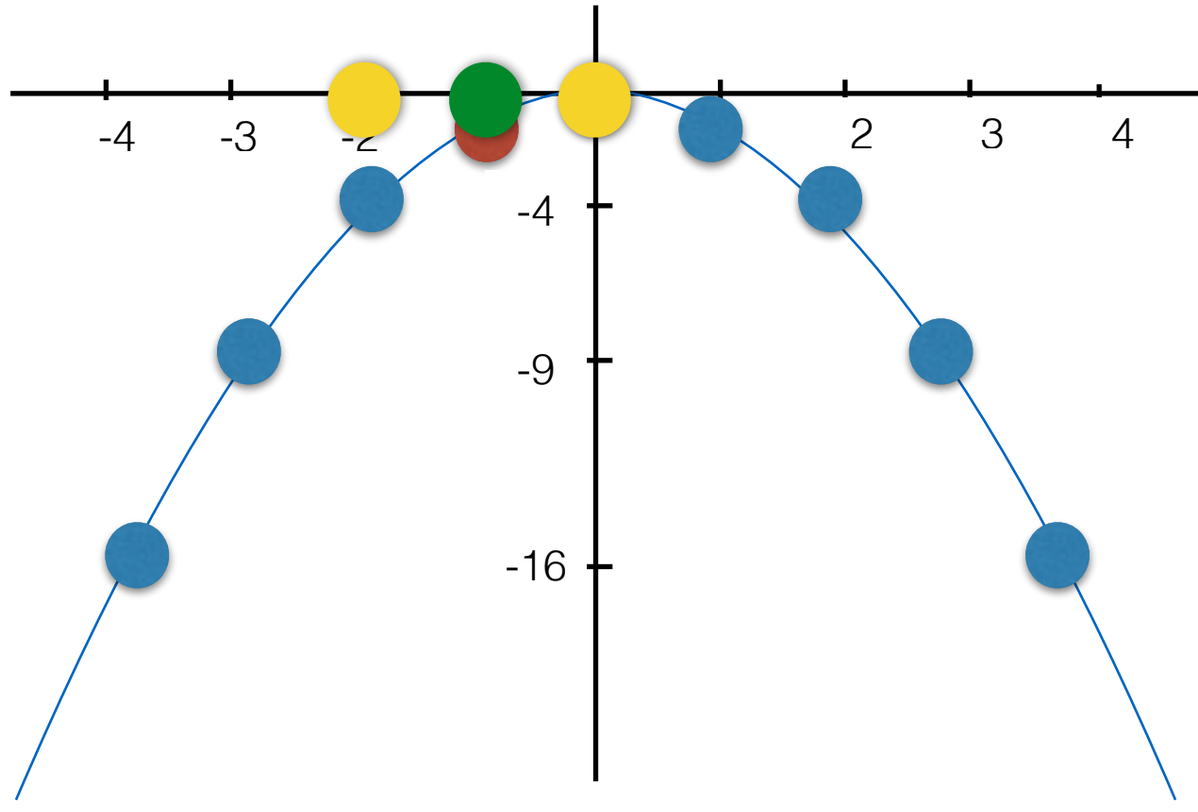
2.1 generate neighbour solutions (differ from current solution by a single element)

2.2 best_neighbour = get highest quality neighbour of current_solution

2.3 If quality(best_neighbour) <= quality(current_solution)

2.3.1 Return current_solution

2.4 current_solution = best_neighbour



Illustrative Example

Hill-Climbing (assuming maximisation)

1. current_solution = generate initial solution randomly

2. Repeat:

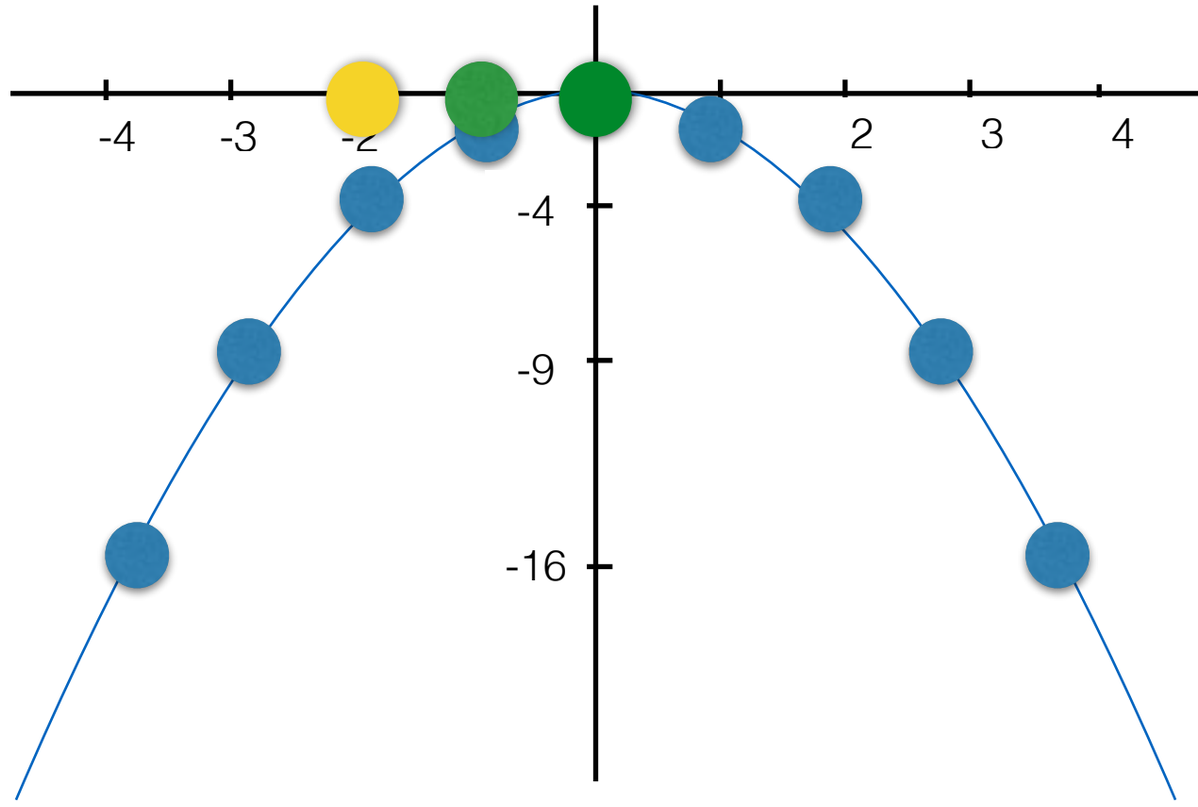
    2.1 generate neighbour solutions (differ from current solution by a single element)

    2.2 best_neighbour = get highest quality neighbour of current_solution

    2.3 If quality(best_neighbour) <= quality(current_solution)

        2.3.1 Return current_solution

    2.4 current_solution = best_neighbour



Illustrative Example

Hill-Climbing (assuming maximisation)

1. current_solution = generate initial solution randomly

2. Repeat:

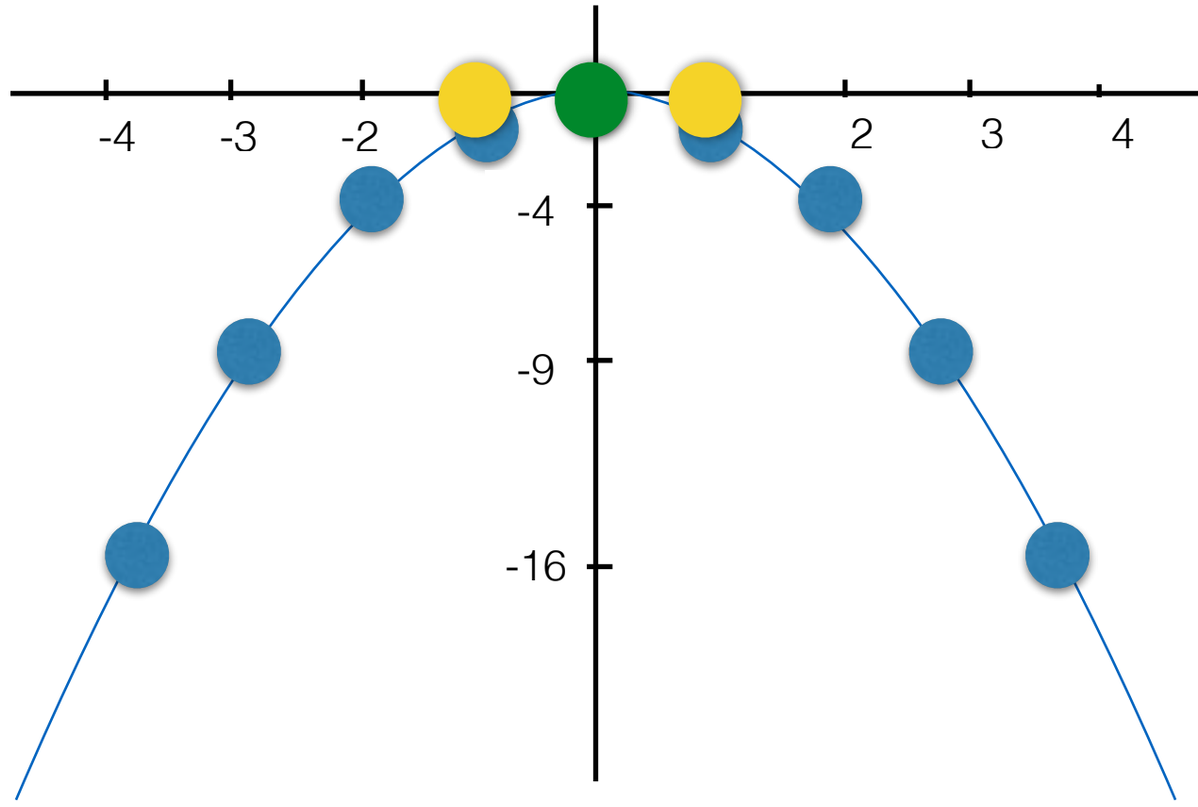
    2.1 generate neighbour solutions (differ from current solution by a single element)

    2.2 best_neighbour = get highest quality neighbour of current_solution

    2.3 If quality(best_neighbour) <= quality(current_solution)

        2.3.1 Return current_solution

    2.4 current_solution = best_neighbour



Illustrative Example

Hill-Climbing (assuming maximisation)

1. current_solution = generate initial solution randomly

2. Repeat:

    2.1 generate neighbour solutions (differ from current solution by a single element)

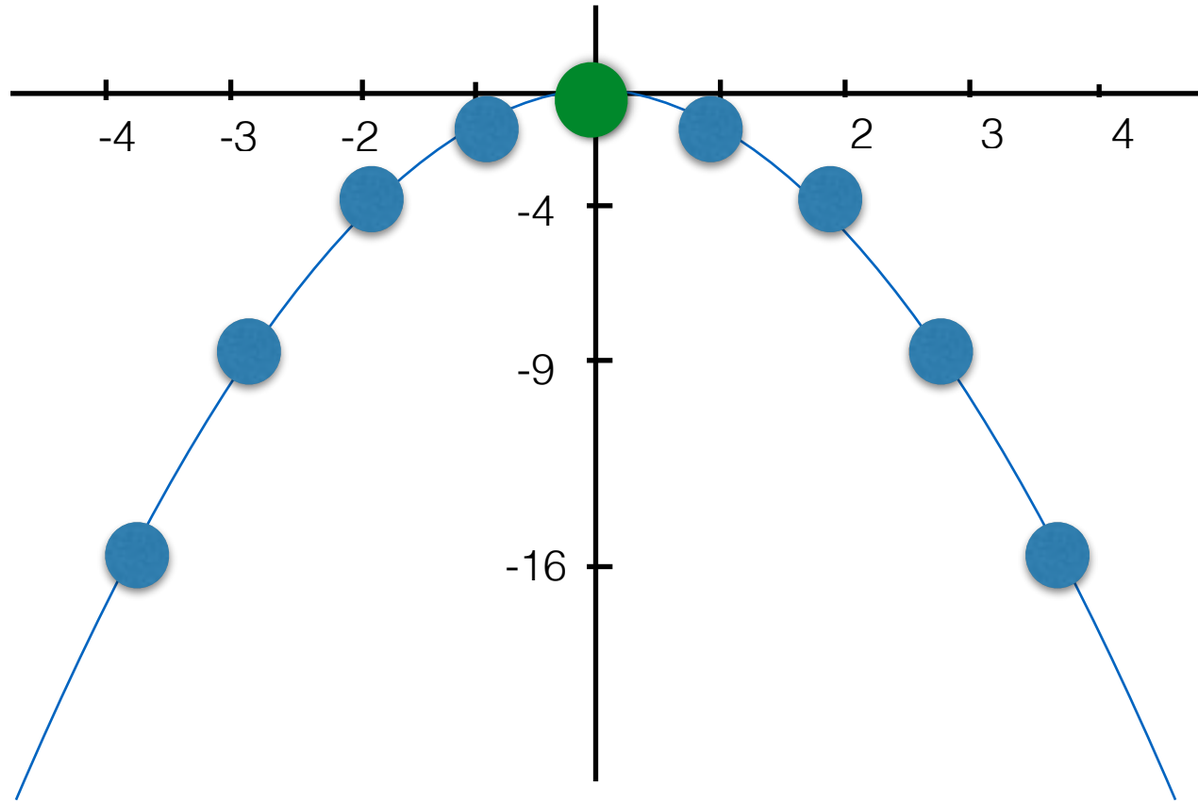
    2.2 best_neighbour = get highest quality neighbour of current_solution

    2.3 If quality(best_neighbour) <= quality(current_solution)

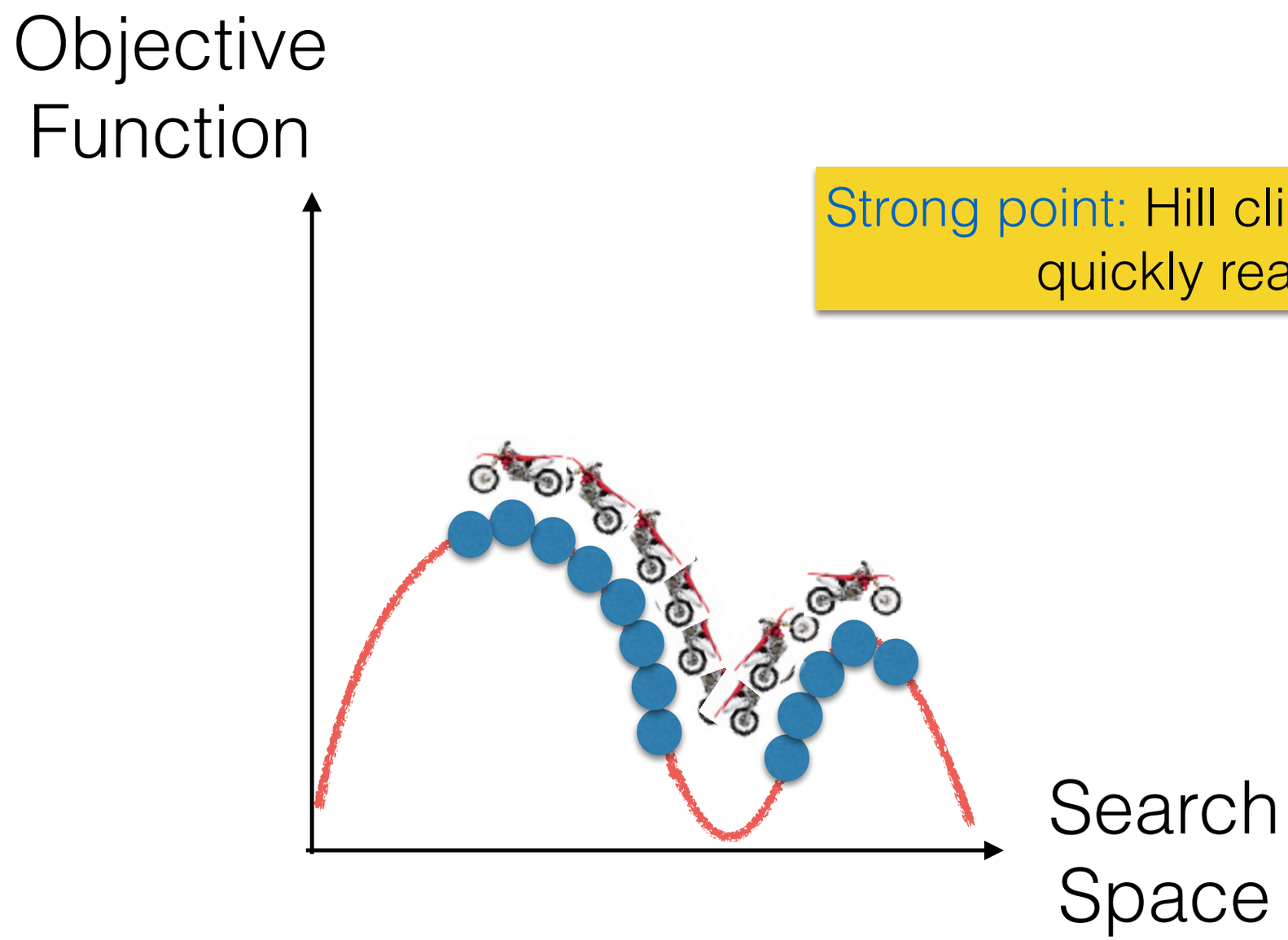
        2.3.1 Return current_solution

    2.4 current_solution = best_neighbour

Until a maximum number of iterations

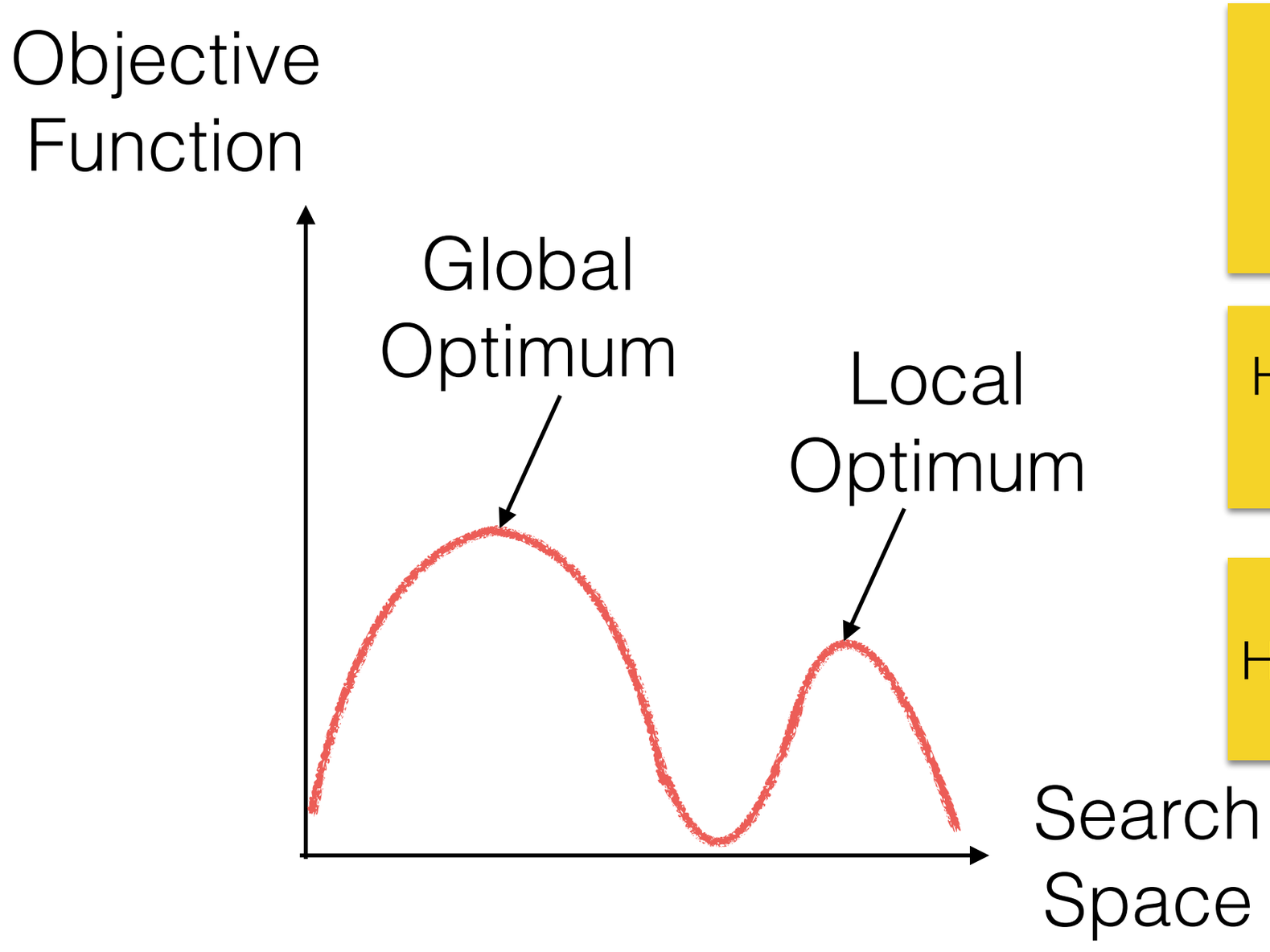


General Idea



Strong point: Hill climbing allows you to quickly reach the top.

Greedy Local Search

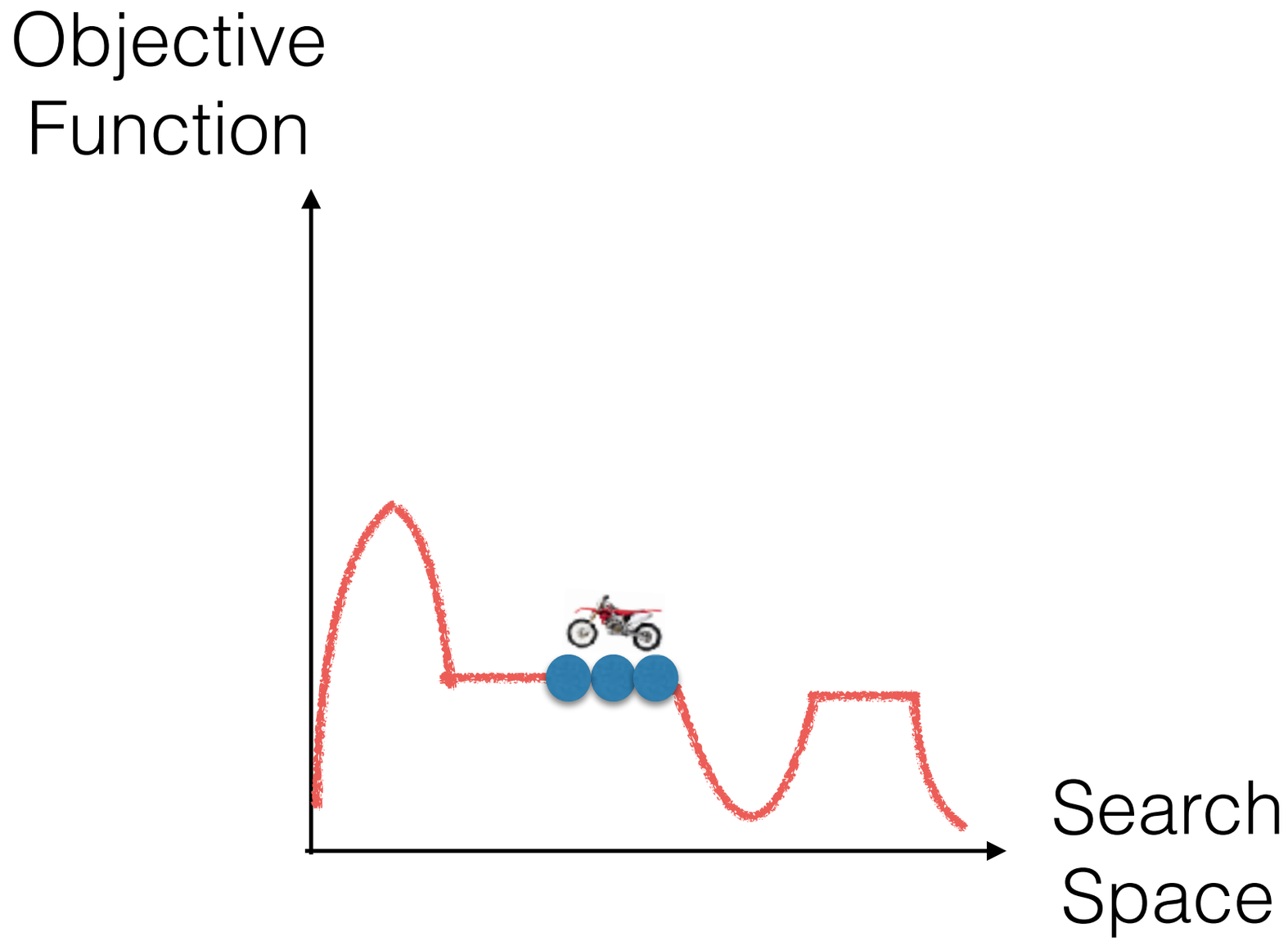


Weakness: Hill-climbing may get trapped in a local optimum.

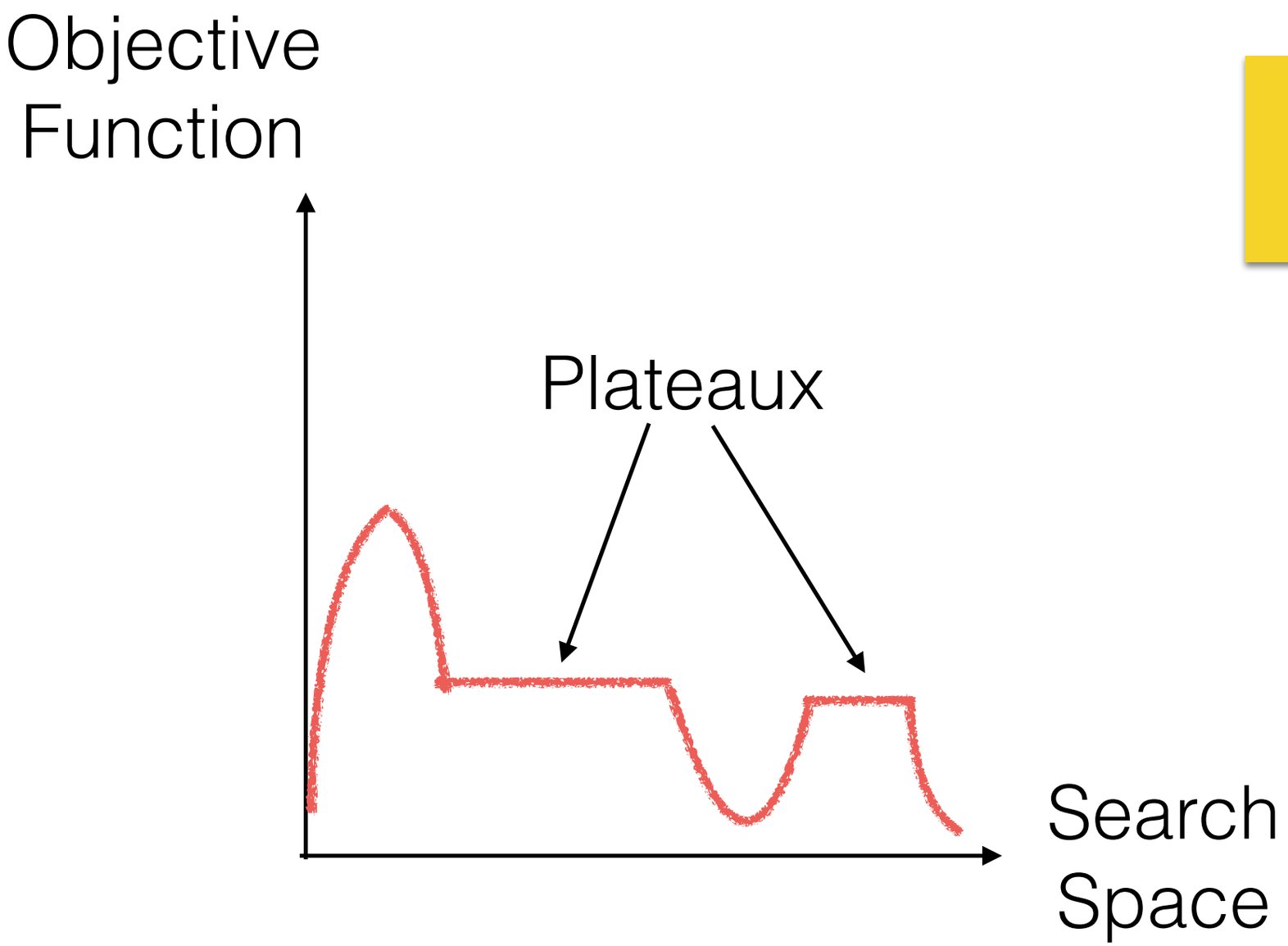
Hill-climbing is a local search method.

Hill-climbing is greedy.

Greedy Local Search



Greedy Local Search



Weakness: Hill-climbing may get trapped in plateaux.

Optimality, Time and Space Complexity

- Optimality:
  - Hill Climbing is not guaranteed to find optimal solutions.
- Time complexity (worst case scenario):
  - We will run until the maximum number of iterations *m* is reached.
  - Within each iteration, we will generate a maximum number of neighbours *n*, each of which may take O(p) each to generate.
  - Worst case scenario: O(mnp).
- Space complexity (worst case scenario):
  - Assume that the design variable is represented by O(q).
  - Within each iteration, we will generate a maximum number of neighbours *n.*
  - Space complexity: O(nq)

Summary

- Hill-climbing is an example of greedy local search optimisation algorithm.
- It can quickly find a local optimum, but may not find optimal solutions.
- Its success depends on the shape of the objective function. As it is a relatively simple algorithm, it can be attempted before more complex algorithms are investigated.

Next

- How to avoid getting stuck in local optima and plateaux?