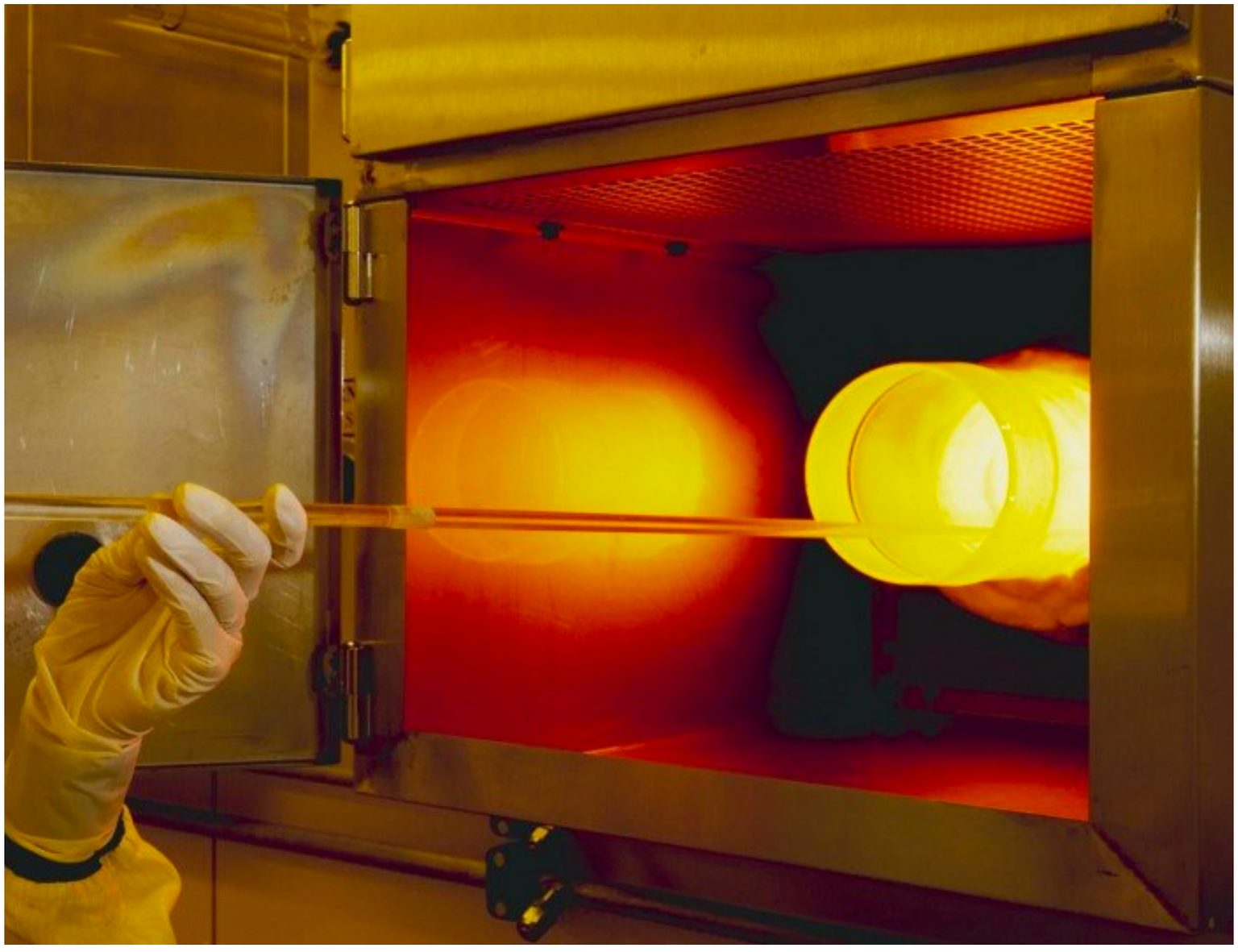


Simulated Annealing

Leandro L. Minku

Overview

- Final remarks on Hill Climbing
- Motivation for Simulated Annealing
- Simulated Annealing

Hill Climbing

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)

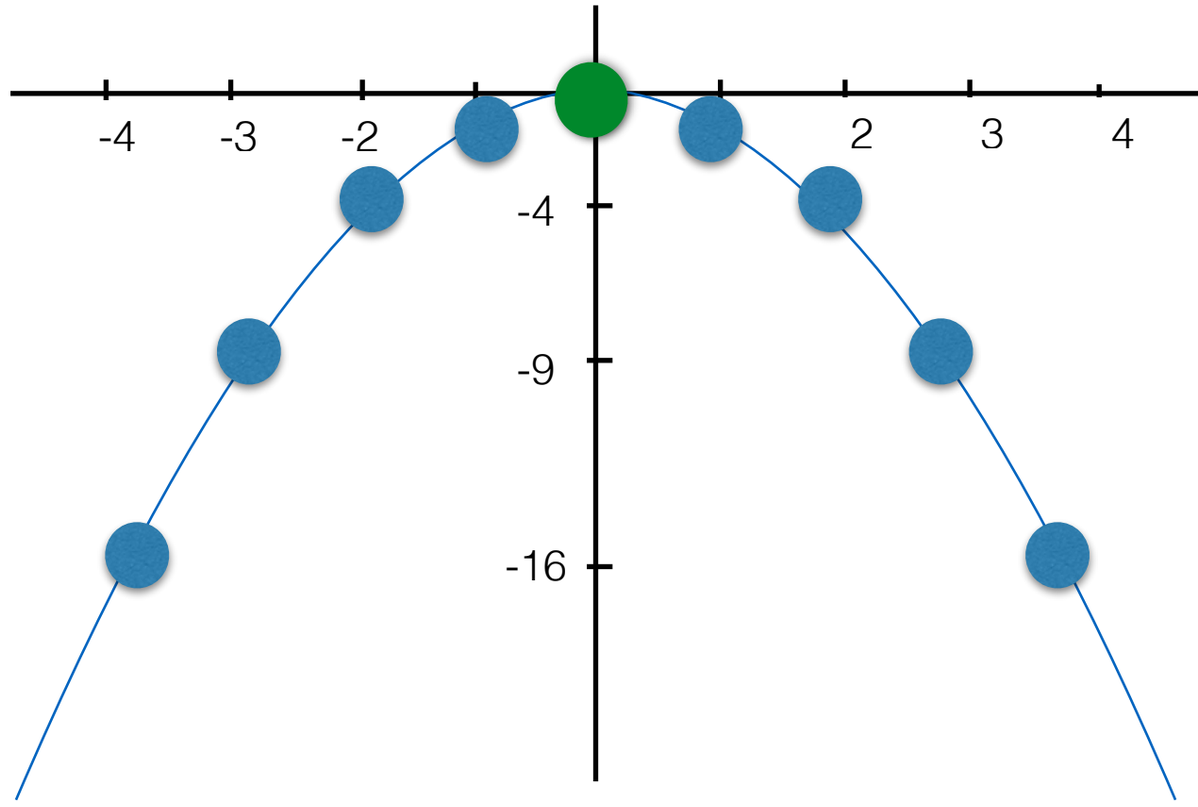
    2.2 best_neighbour = get highest quality neighbour of current_solution

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

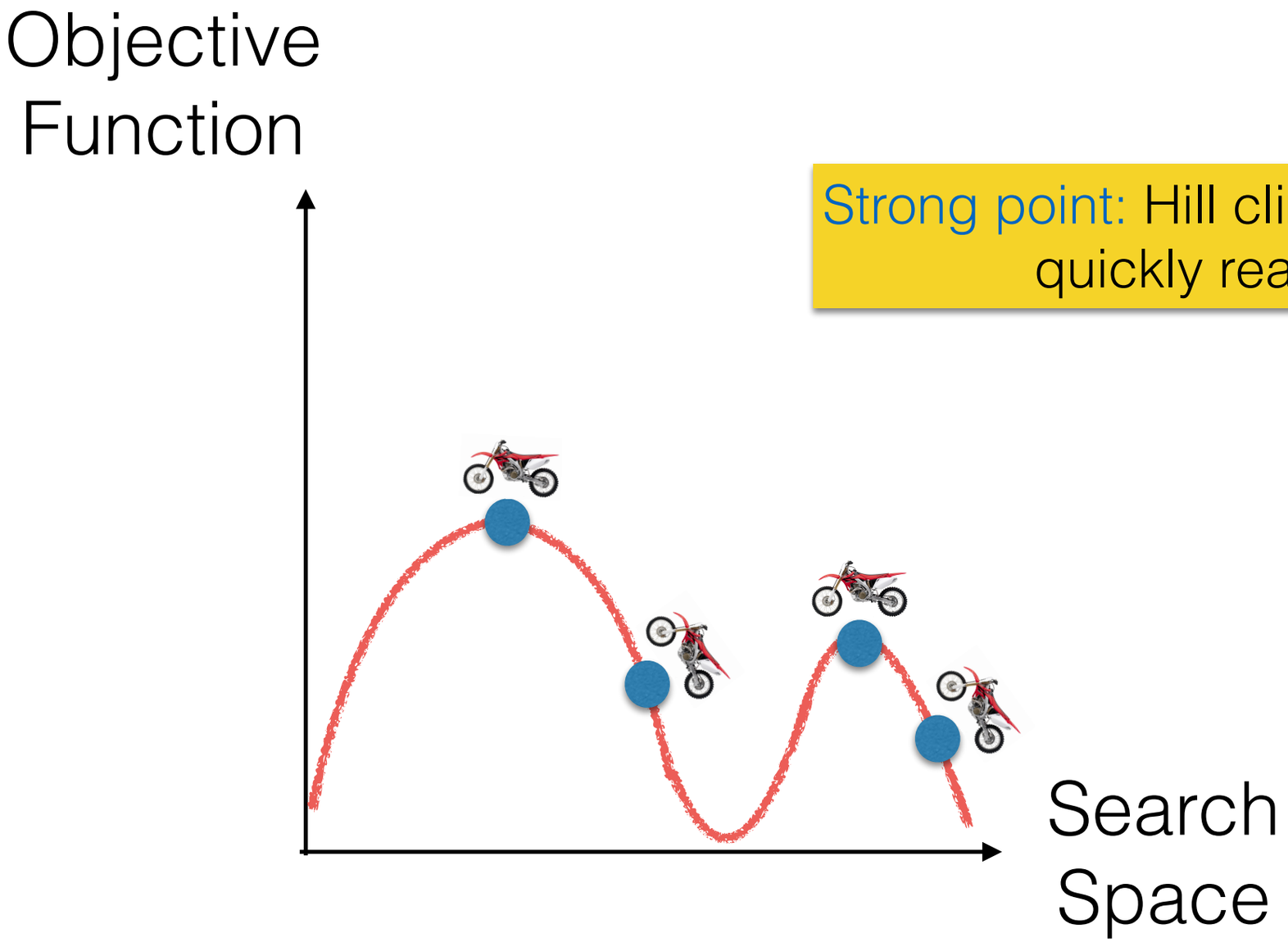
        2.3.1 Return current_solution

    2.4 current_solution = best_neighbour

Until a maximum number of iterations

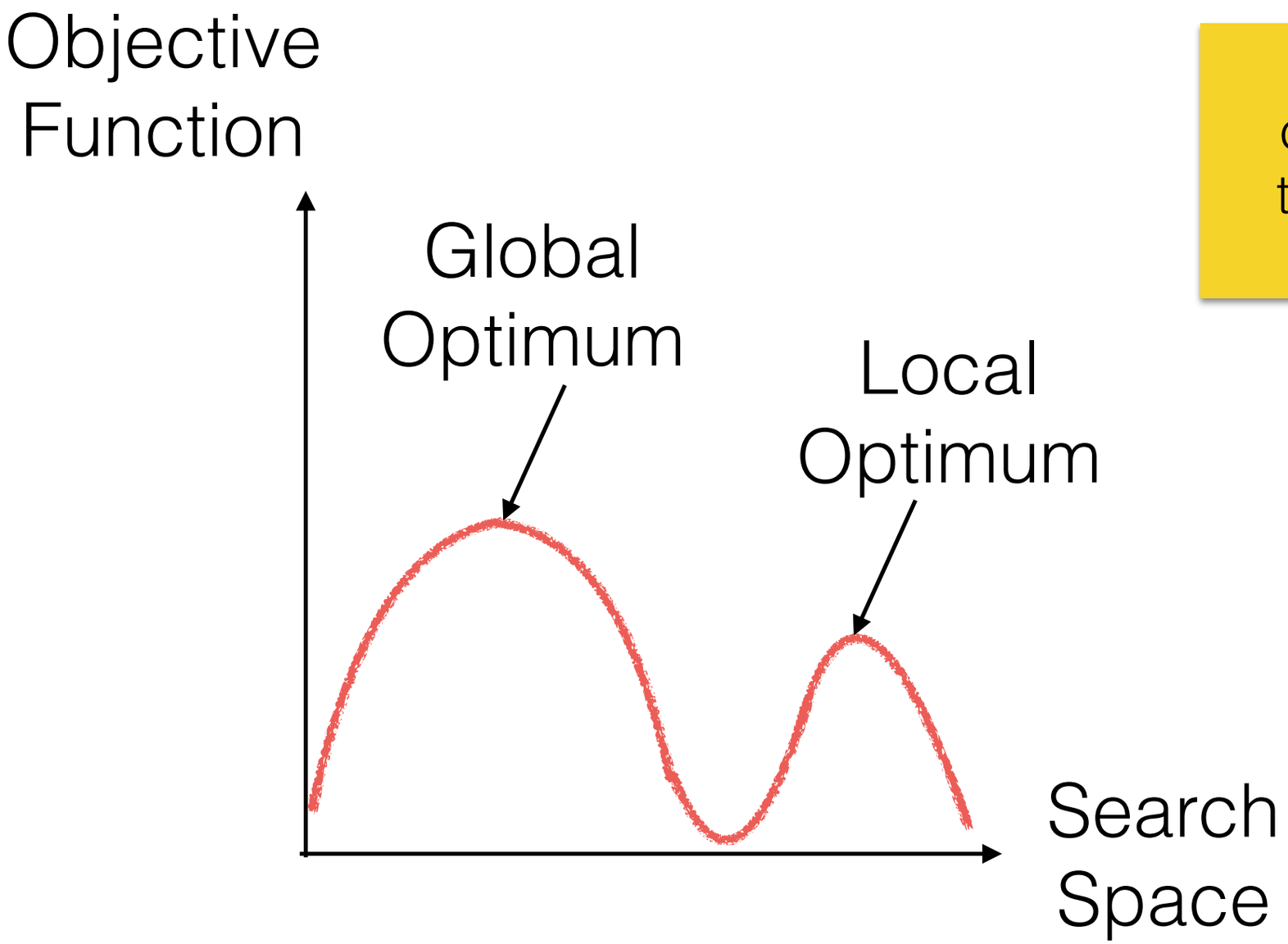


Strengths and Weaknesses of Hill Climbing



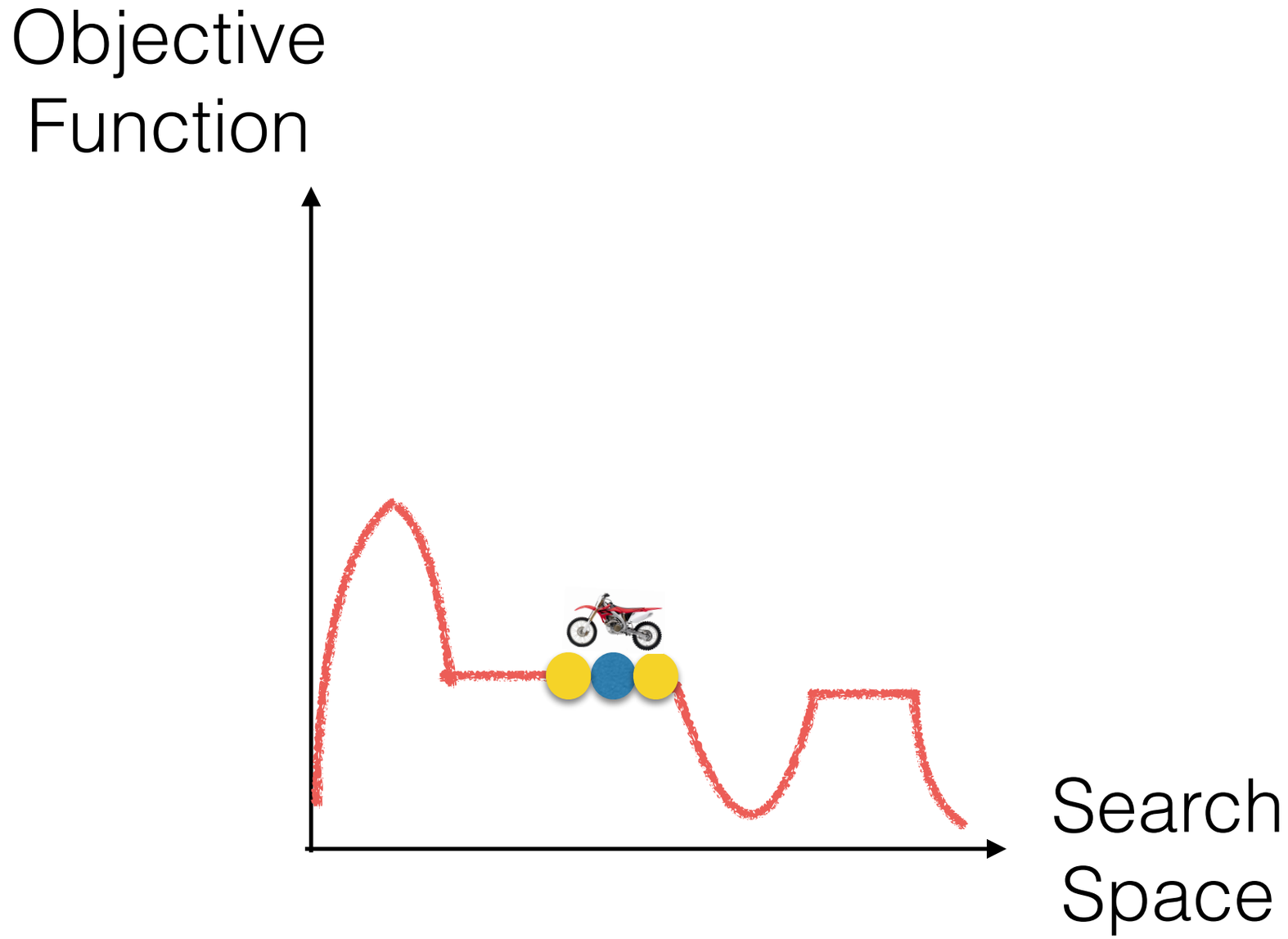
Strong point: Hill climbing allows you to quickly reach the top.

Strengths and Weaknesses of Hill Climbing

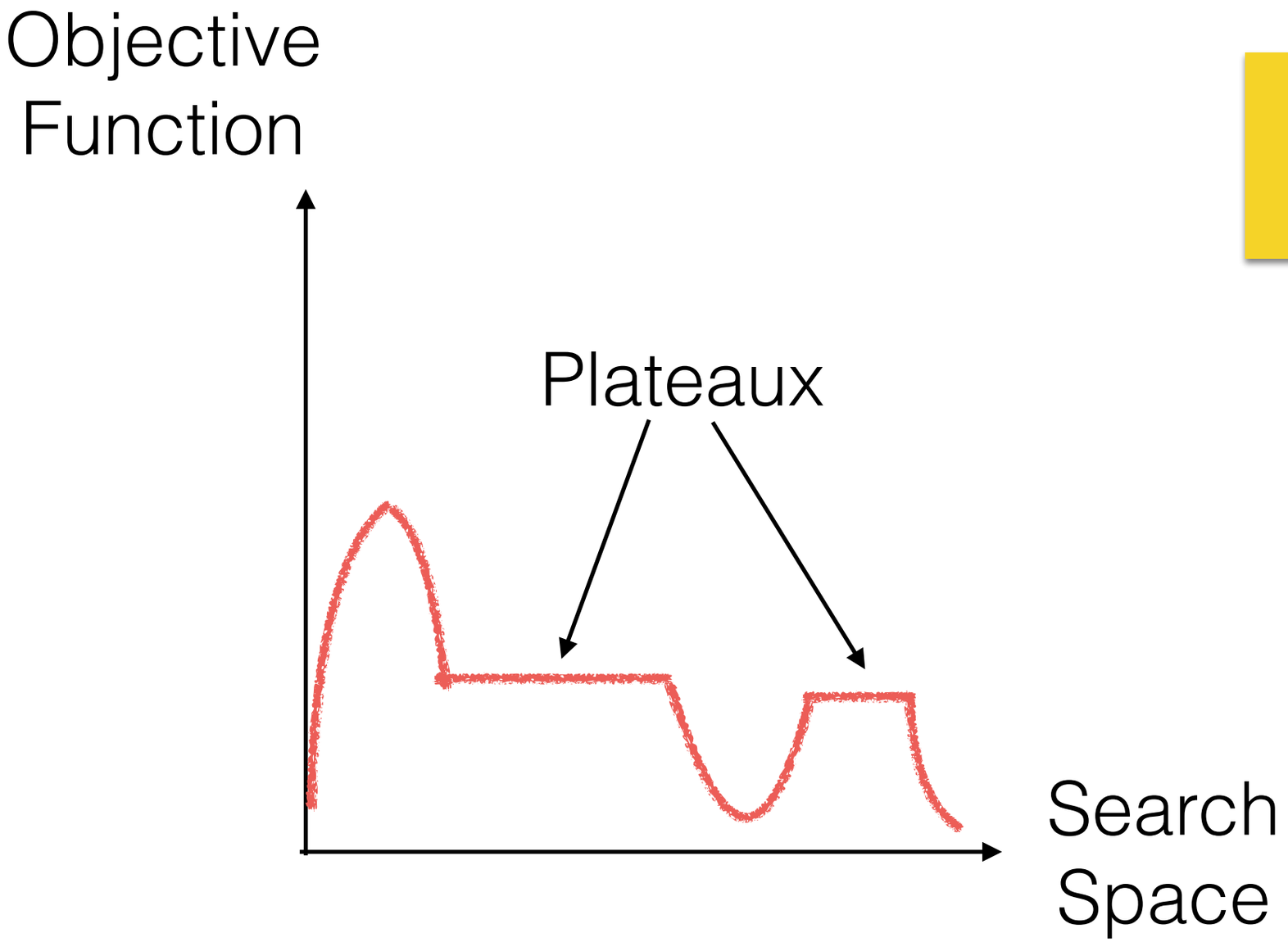


Weakness: Hill-climbing may get trapped in a local optimum.

Strengths and Weaknesses of Hill Climbing

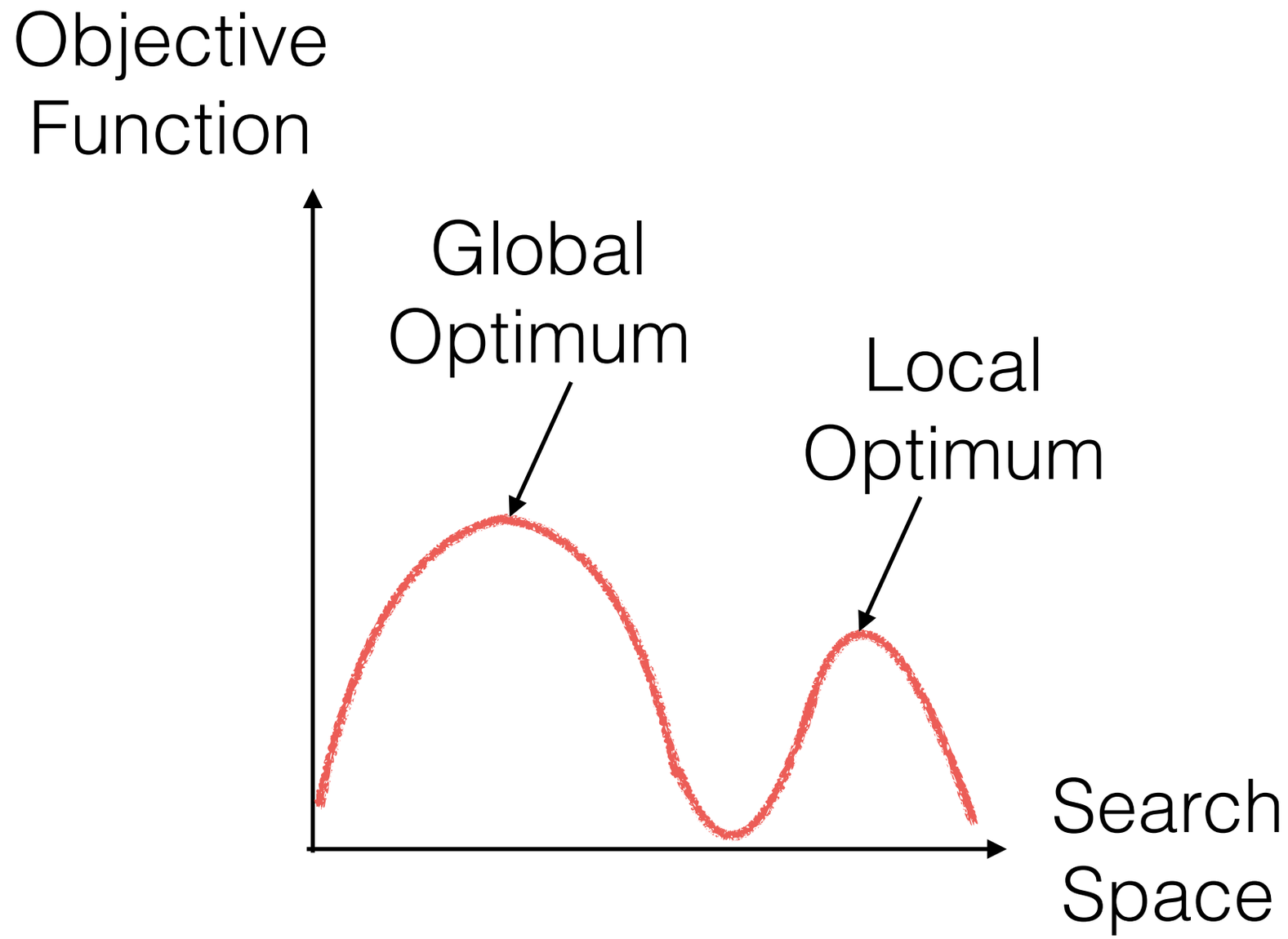


Strengths and Weaknesses of Hill Climbing



Weakness: Hill-climbing may get trapped in plateaux.

Optimality



Hill Climbing is not guaranteed to find optimal solutions.

However, it is a simple algorithm that may be able to quickly find good enough solutions for some problems.

Time Complexity (Worst Case Scenario)

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

- 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).

Time Complexity (Worst Case Scenario)

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

- 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)

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

- Within each iteration, we will generate a maximum number of neighbours *n*.
- Assume that storing the candidate solution is in O(q).
- Assume that the space needed for the process of generating the neighbours is negligible compared to *n* and *q.*
- Space complexity: O(nq+r) = O(nq).
- PS: this can be improved if we merge instructions 2.1 and 2.2.

Space Complexity (Worst Case Scenario)

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)

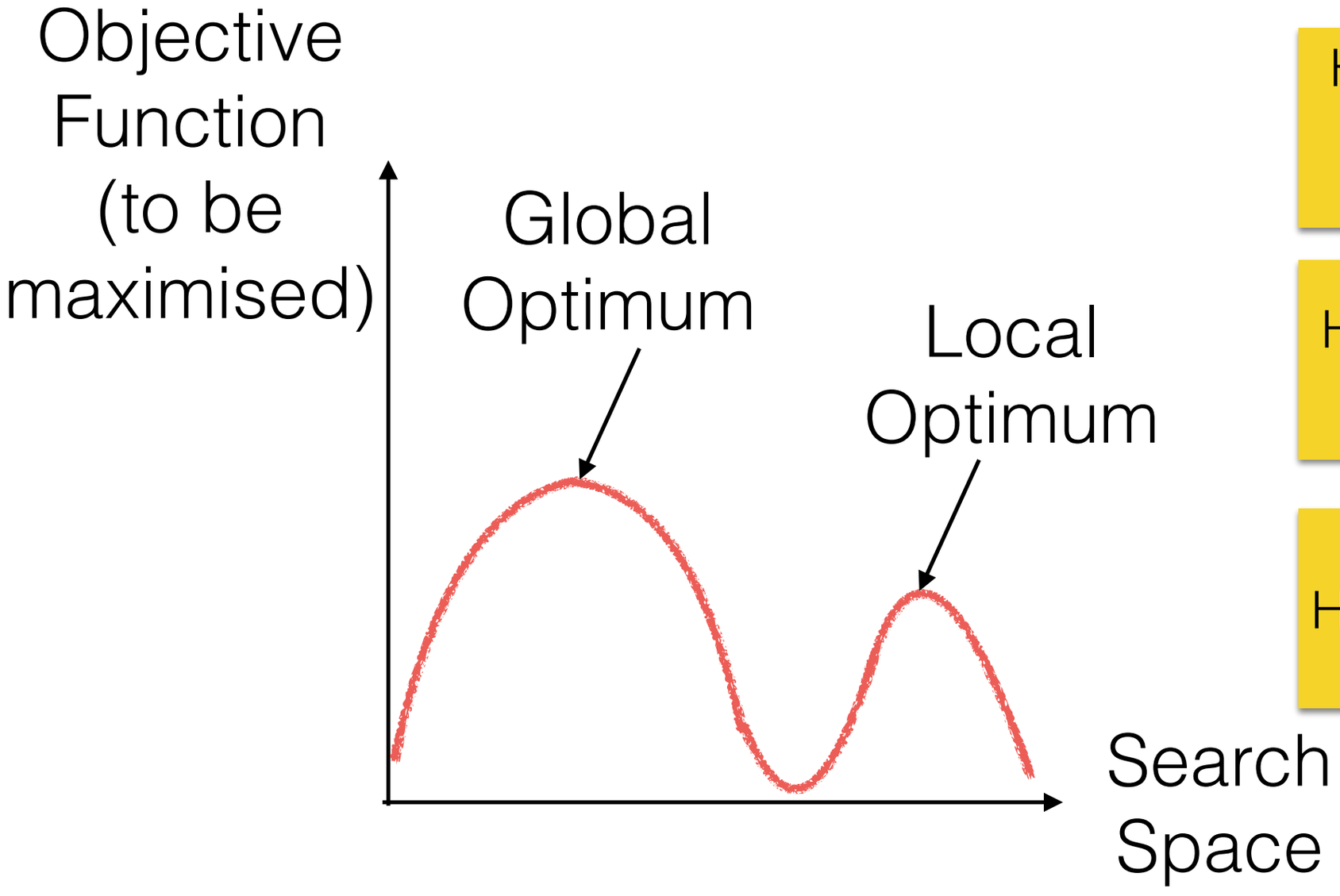
        2.3.1 Return current_solution

    2.4 current_solution = best_neighbour

Until a maximum number of iterations

- Within each iteration, we will generate a maximum number of neighbours *n*.
- Assume that storing the candidate solution is in O(q).
- Assume that the space needed for the process of generating the neighbours is negligible compared to *n* and *q.*
- Space complexity: O(nq).
- PS: this can be improved if we merge instructions 2.1 and 2.2.

Why Hill Climbing Gets Trapped in Local Optima?

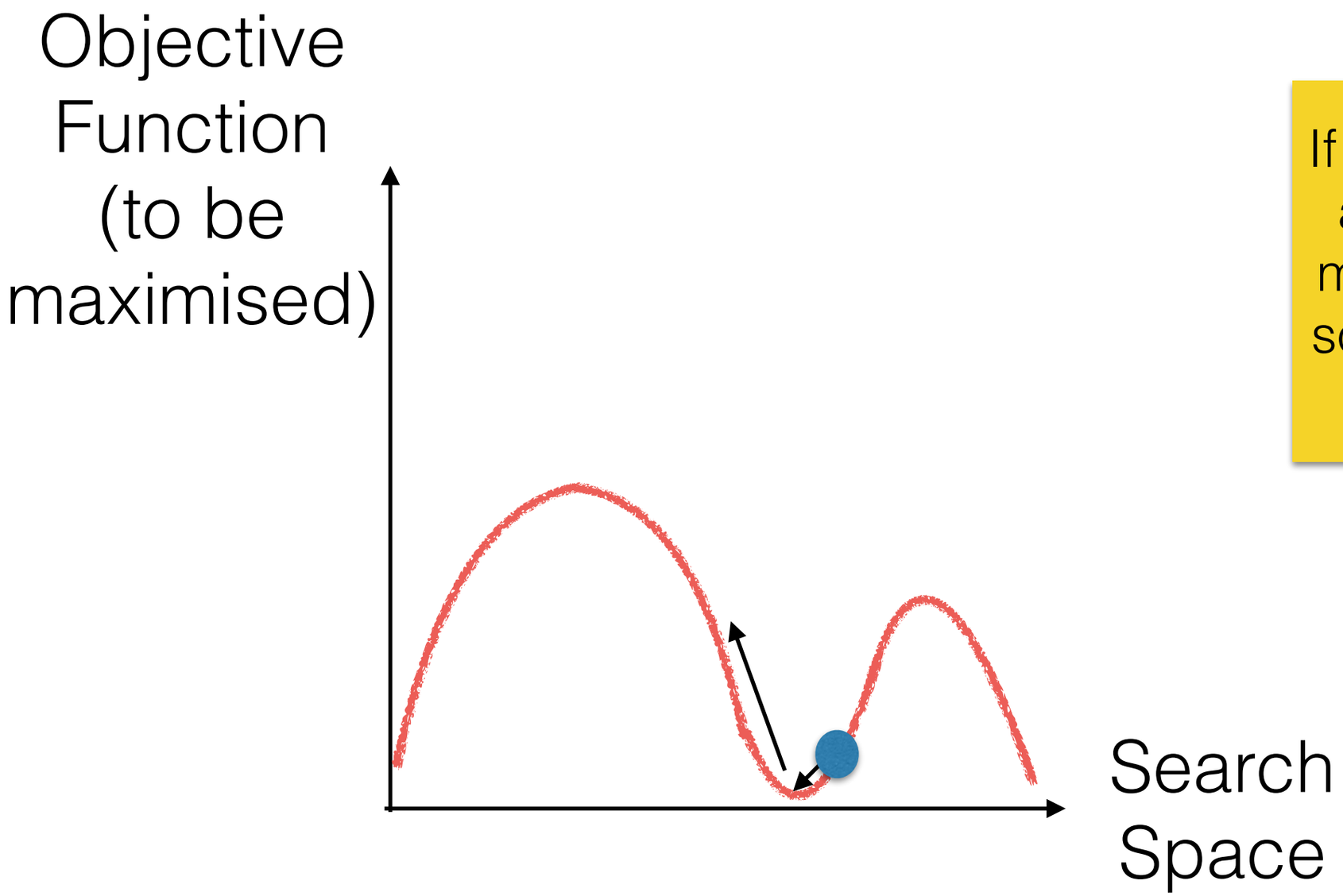


Hill-climbing may get trapped in a local optimum.

Hill-climbing is a local search method.

Hill-climbing is greedy.

Motivation for Simulated Annealing



If we could sometimes accept a downward move, we would have some chance to move to another hill.

Hill-Climbing

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

In simulated annealing, instead of taking the best neighbour, we pick a random neighbour.

Hill-Climbing

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

Simulated annealing will give some chance to accept a bad neighbour.

Simulated Annealing

Simulated Annealing (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 rand_neighbour = get random neighbour of current_solution

2.3 If quality(rand_neighbour) <= quality(current_solution) {

2.3.1 With some probability,
current_solution = rand_neighbour

} Else current_solution = rand_neighbour

Simulated Annealing

Simulated Annealing (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 rand_neighbour = get random neighbour of current_solution

2.3 If quality(rand_neighbour) <= quality(current_solution) {

2.3.1 With some probability,
current_solution = rand_neighbour

} Else current_solution = rand_neighbour

Simulated Annealing

Simulated Annealing (assuming maximisation)

1. current_solution = generate initial solution randomly

2. Repeat:

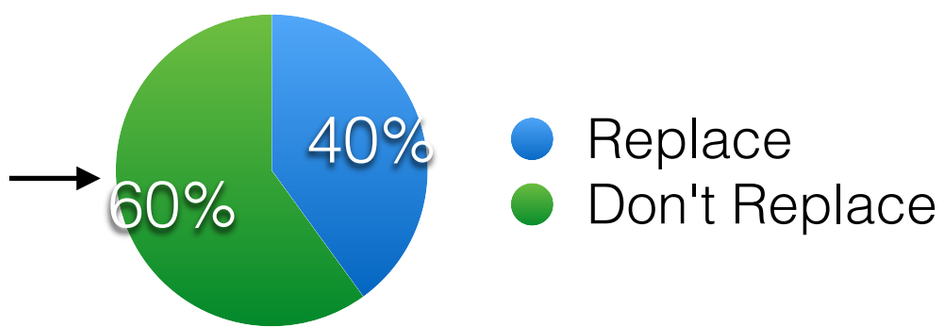
~~2.1 generate neighbour solutions (differ from current solution by a single element)~~

2.2 rand_neighbour = get random neighbour of current_solution

2.3 If quality(rand_neighbour) <= quality(current_solution) {

2.3.1 With some probability,
current_solution = rand_neighbour

} Else current_solution = rand_neighbour



How Should the Probability be Set?

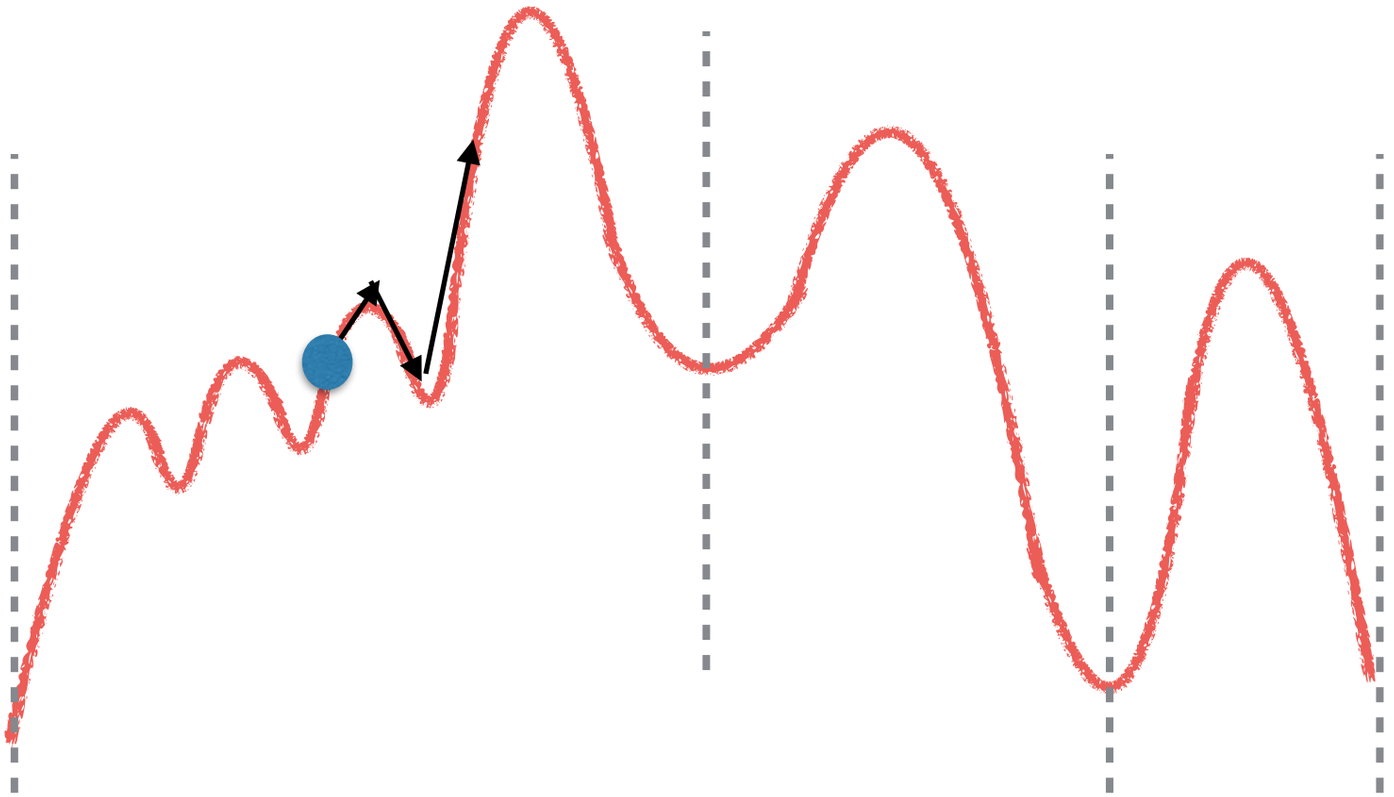
- Probability to accept solutions with much worse quality should be lower.
  - We don’t want to be dislodged from the optimum.

- High probability in the beginning.
  - More similar effect to random search.
  - Allows us to explore the search space.

- Lower probability as time goes by.
  - More similar effect to hill-climbing.
  - Allows us to exploit a hill.

How to Decrease the Probability?

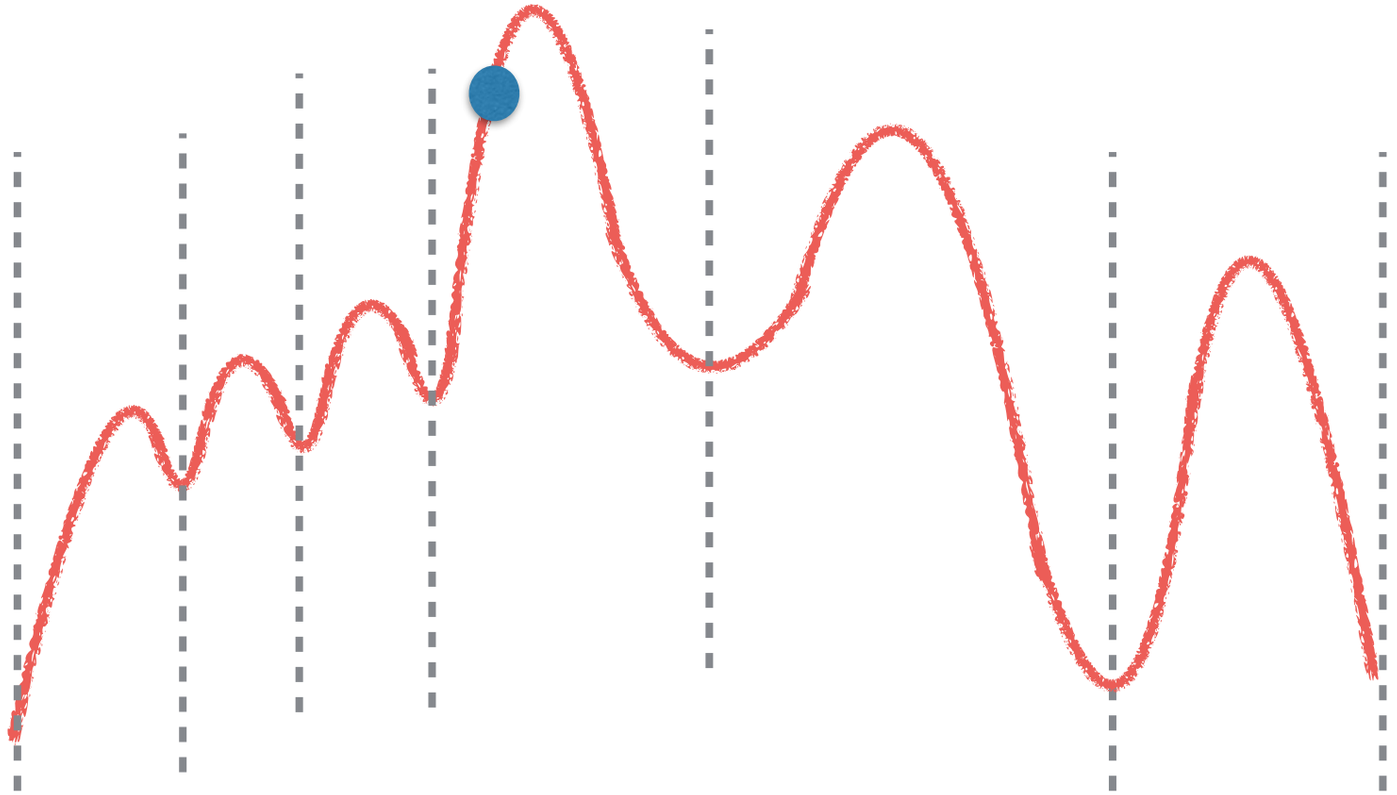
- We would like to decrease the probability slowly.



If you decrease the probability slowly, you start to form basis of attraction, but you can still walk over small hills initially.

How to Decrease the Probability?

- We would like to decrease the probability slowly.



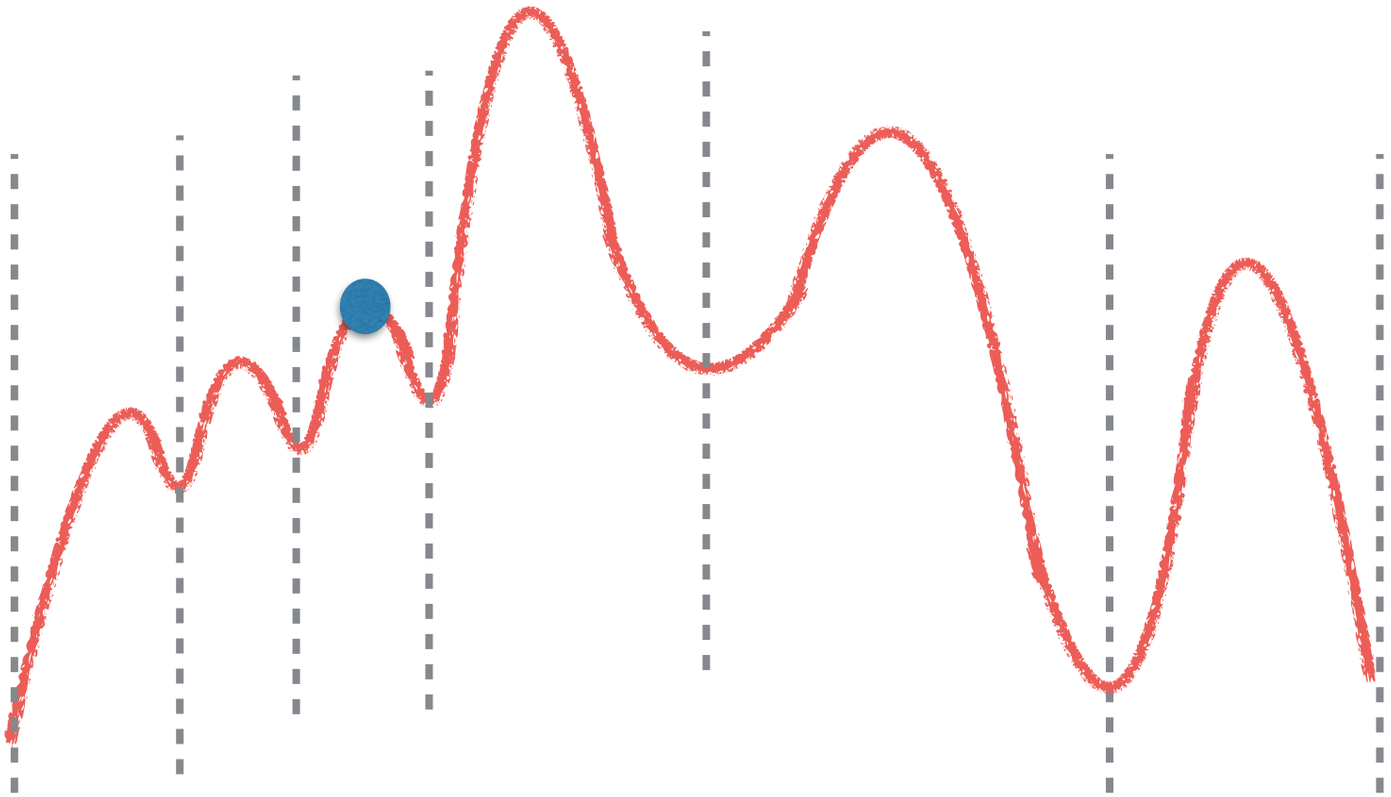
As the probability decreases further, the small hills start to form basis of attraction too.

But if you do so slowly enough, you give time to wander to the higher value hills before starting to exploit.

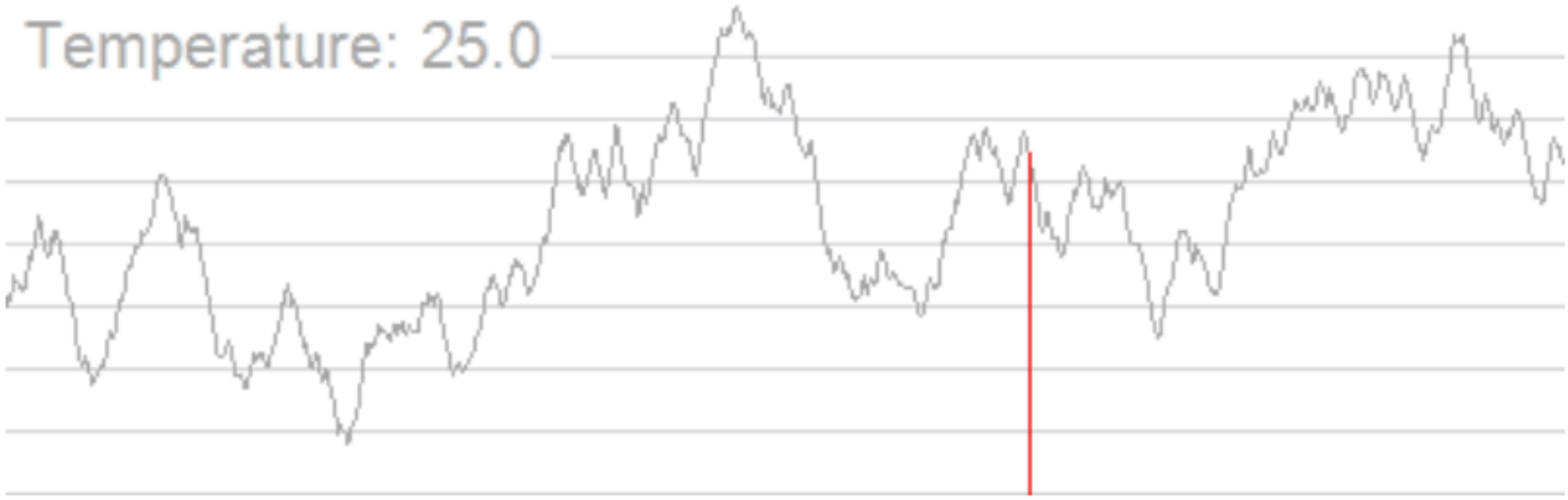
So, you can hopefully find the global optimum!

How to Decrease the Probability?

- We would like to decrease the probability slowly.



If you decrease too quickly, you can get trapped in local optima.

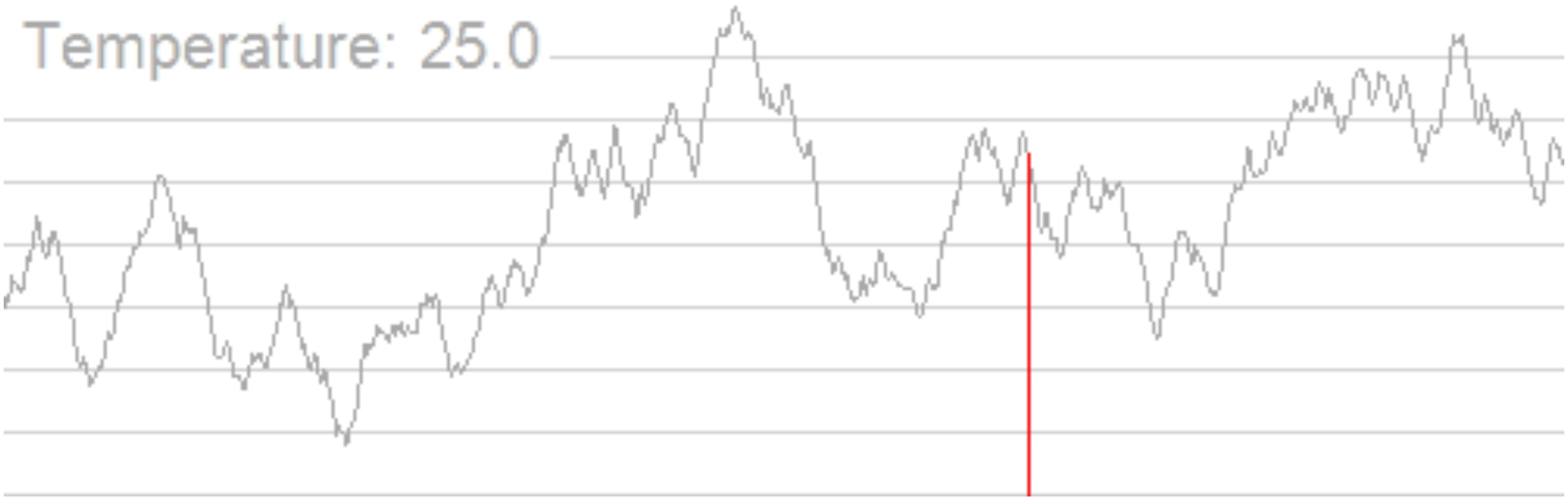


[By Kingpin13 - Own work, CC0, https://commons.wikimedia.org/w/index.php?curid=25010763]

Note 1: it may be very difficult to visualise the quality function in real world problems.

Note 2: in this example we have only 2 neighbours for each solution, but in real world problems we may have many neighbours for each solution.

Note 3: real world problems may have much more dimensions.



[By Kingpin13 - Own work, CC0, https://commons.wikimedia.org/w/index.php?curid=25010763]

Note 4: the algorithm itself does not know the shape of the function beforehand.

Note 5: the line is not jumping to non-neighbouring positions, it's just moving very fast!

Simulated Annealing

Simulated Annealing (assuming maximisation)

1. current_solution = generate initial solution randomly
2. Repeat:

2.1 rand_neighbour = get random neighbour of current_solution

2.2 If quality(rand_neighbour) <= quality(current_solution) {

2.2.1 With some probability,
current_solution = rand_neighbour

} Else current_solution = rand_neighbour

2.3 Reduce probability

Simulated Annealing

Simulated Annealing (assuming maximisation)

1. current_solution = generate initial solution randomly
2. Repeat:

2.1 rand_neighbour = get random neighbour of current_solution

2.2 If quality(rand_neighbour) <= quality(current_solution) {

2.2.1 With some probability,
current_solution = rand_neighbour

} Else current_solution = rand_neighbour

2.3 Reduce probability

Metallurgy Annealing

- A blacksmith heats the metal to a very high temperature.
- When heated, the steel's atoms can move fast and randomly.

Image from: http://www.stormthecastle.com/indeximages/sting-steel-thumb.jpg

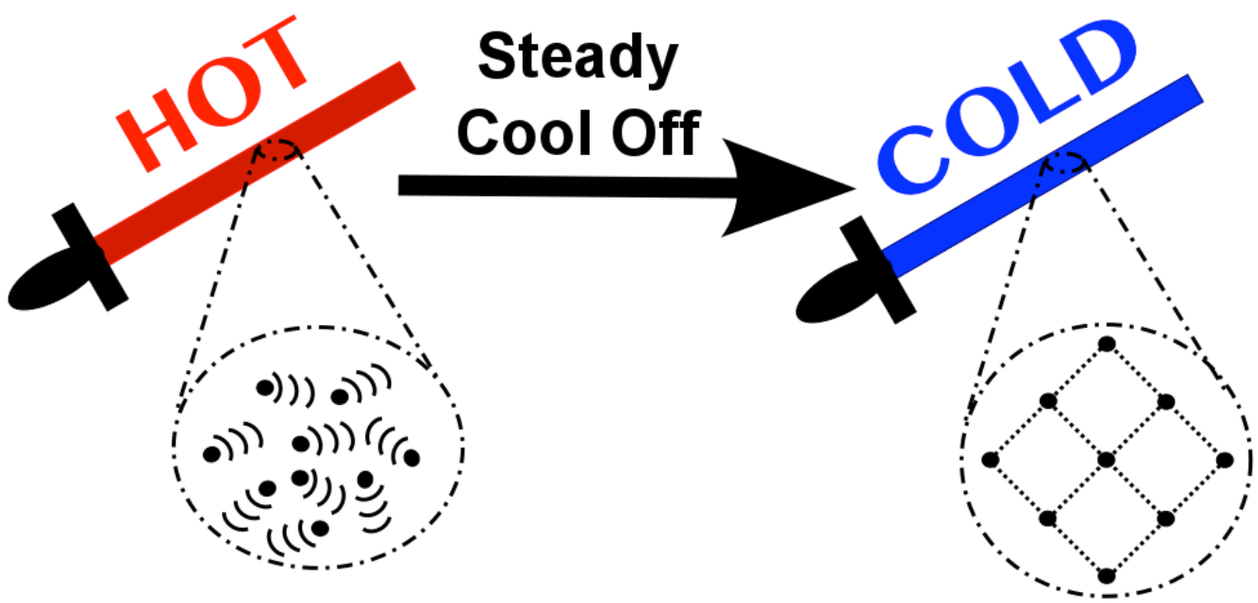


Image from: http://2.bp.blogspot.com/--kOlrodykkg/UbfVZ0_I5HI/AAAAAAAAAJ4/0rQ98g6tDDA/s1600/annealingAtoms.png

- The blacksmith then lets it cool down slowly.
- If cooled down at the right speed, the atoms will settle in nicely.
- This makes the sword stronger than the untreated steel.

Probability Function

Probability of accepting a solution of equal or worse quality, inspired by thermodynamics:

$$e^{\Delta E/T}$$

ΔE = quality(rand_neighbour) - quality(current_solution)
(<=0)

Assuming maximisation…

T = temperature
(>0)

e = 2.71828…

Exponential Function

$e^{\Delta E/T}$

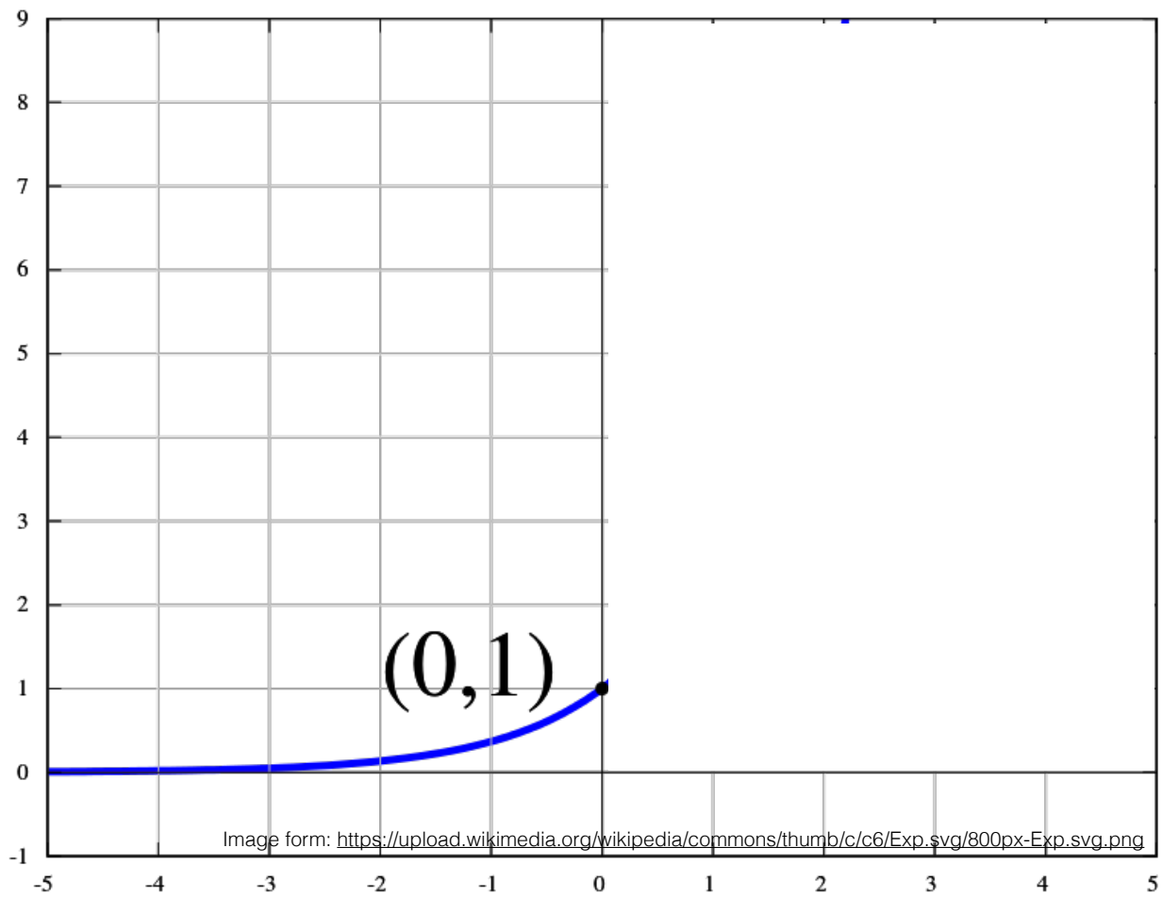


Image form: https://upload.wikimedia.org/wikipedia/commons/thumb/c/c6/Exp.svg/800px-Exp.svg.png

$\Delta E/T$
(<=0)

e = 2.71828…

Exponential Function

$e^{\Delta E/T}$

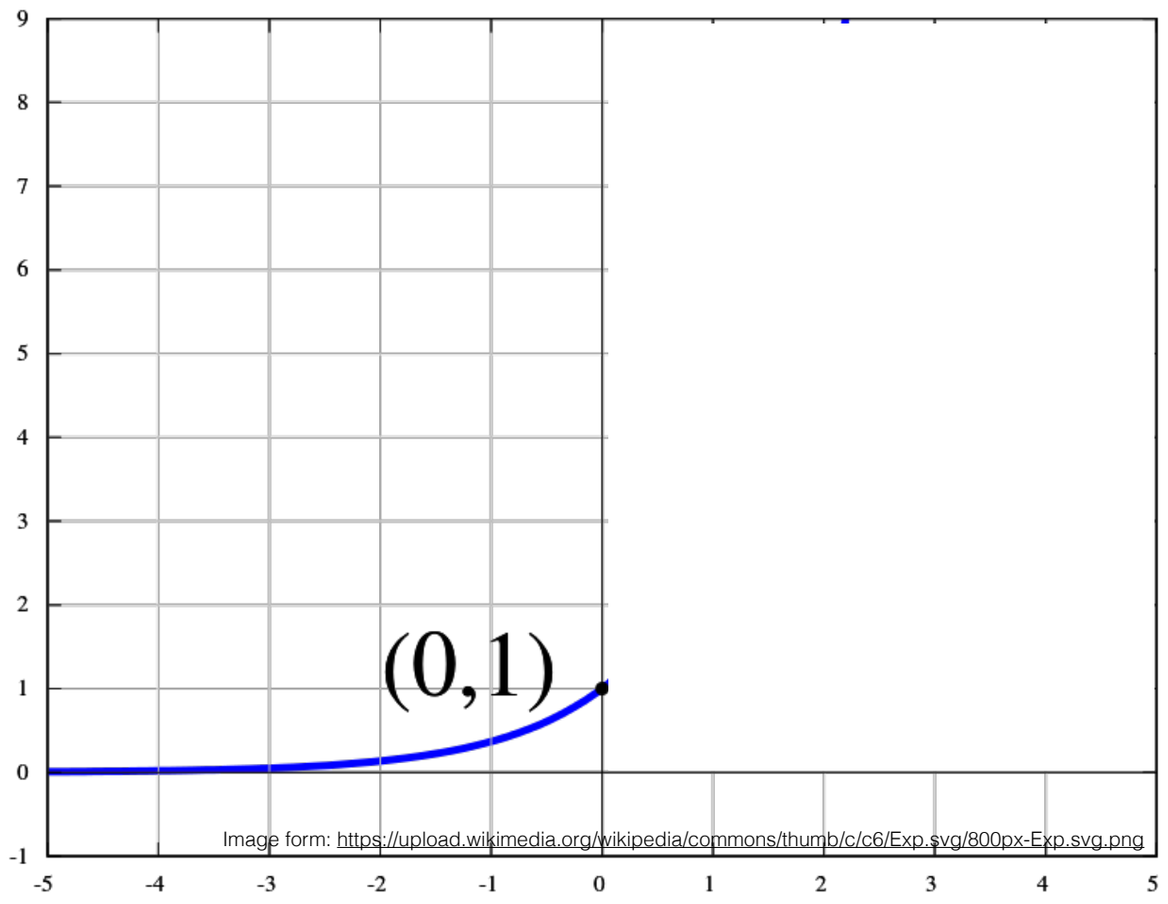


Image form: https://upload.wikimedia.org/wikipedia/commons/thumb/c/c6/Exp.svg/800px-Exp.svg.png

$\Delta E/T$

Exponential Function

$e^{\Delta E/T}$

$e^{\Delta E/T}$

But never reaches zero

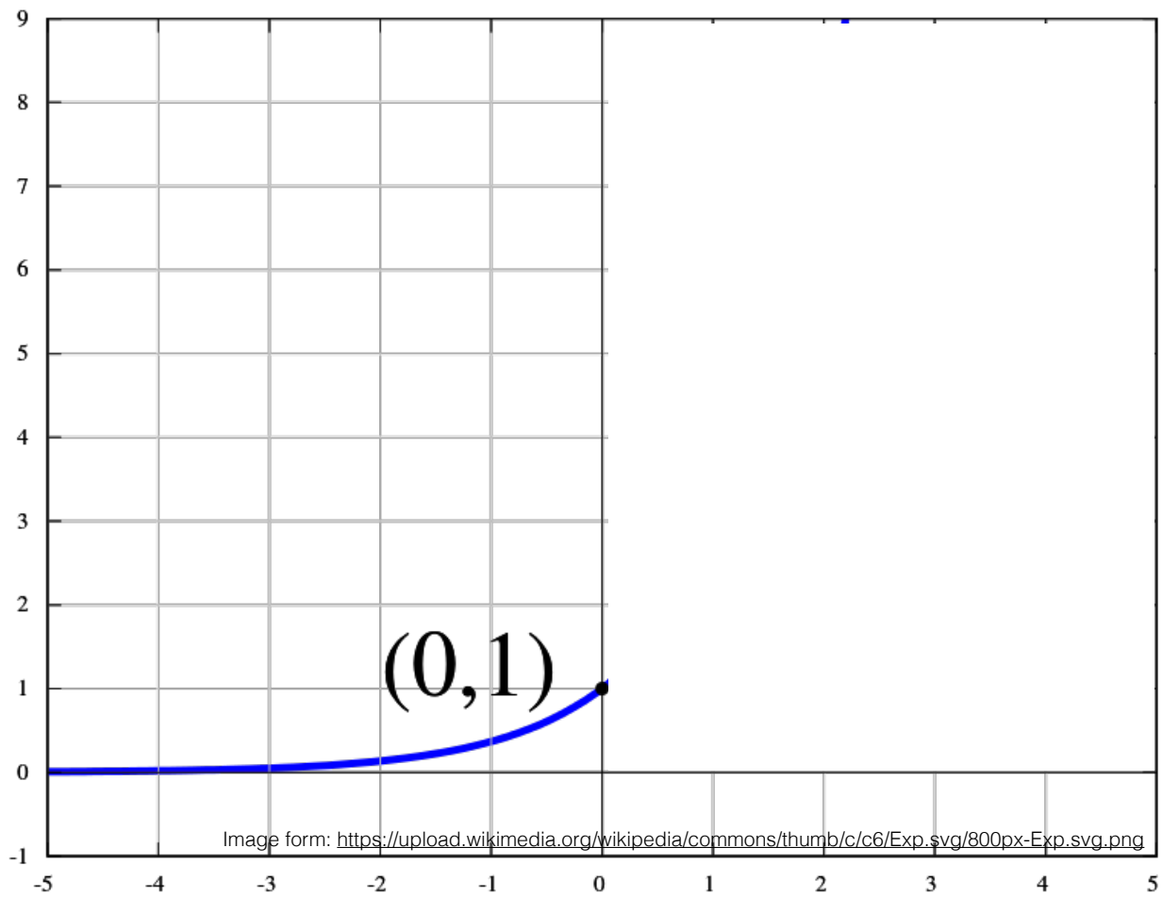


Image form: https://upload.wikimedia.org/wikipedia/commons/thumb/c/c6/Exp.svg/800px-Exp.svg.png

$\Delta E/T$

How Does ΔE Affect the Probability?

Probability of accepting a solution of equal or worse quality:

$$e^{\Delta E/T}$$

$$\Delta E = \text{quality(rand\_neighbour)} - \text{quality(current\_solution)}$$

(ΔE <=0)

Assuming maximisation…

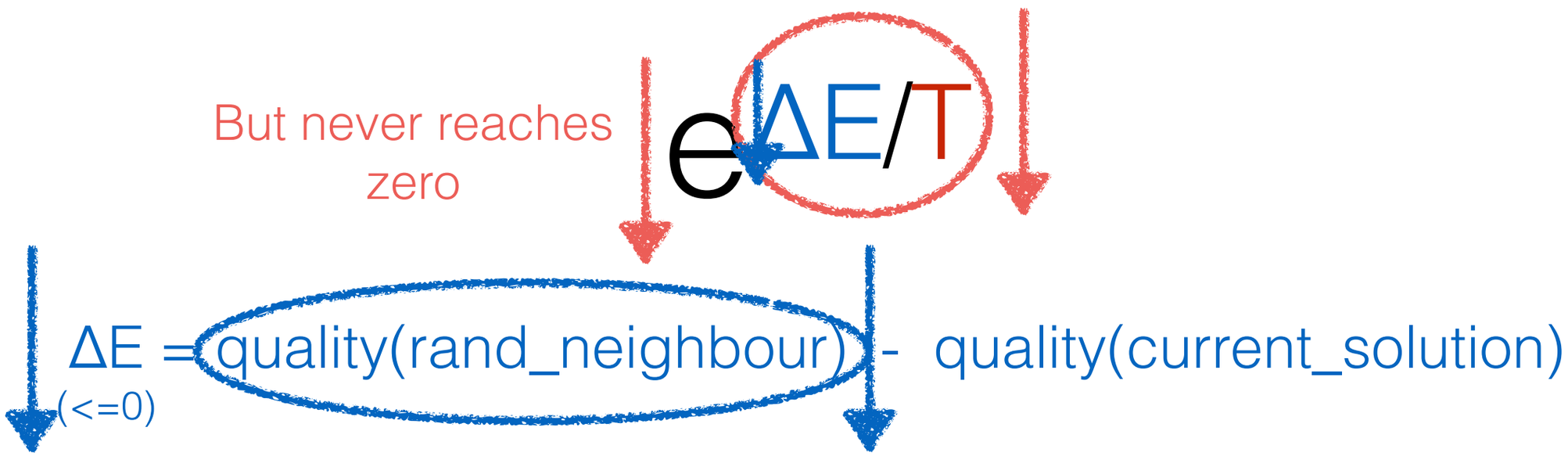
T = temperature

(>0)

The worse the neighbour is in comparison to the current solution, the less likely to accept it.

How Does ΔE Affect the Probability?

Probability of accepting a solution of equal or worse quality:



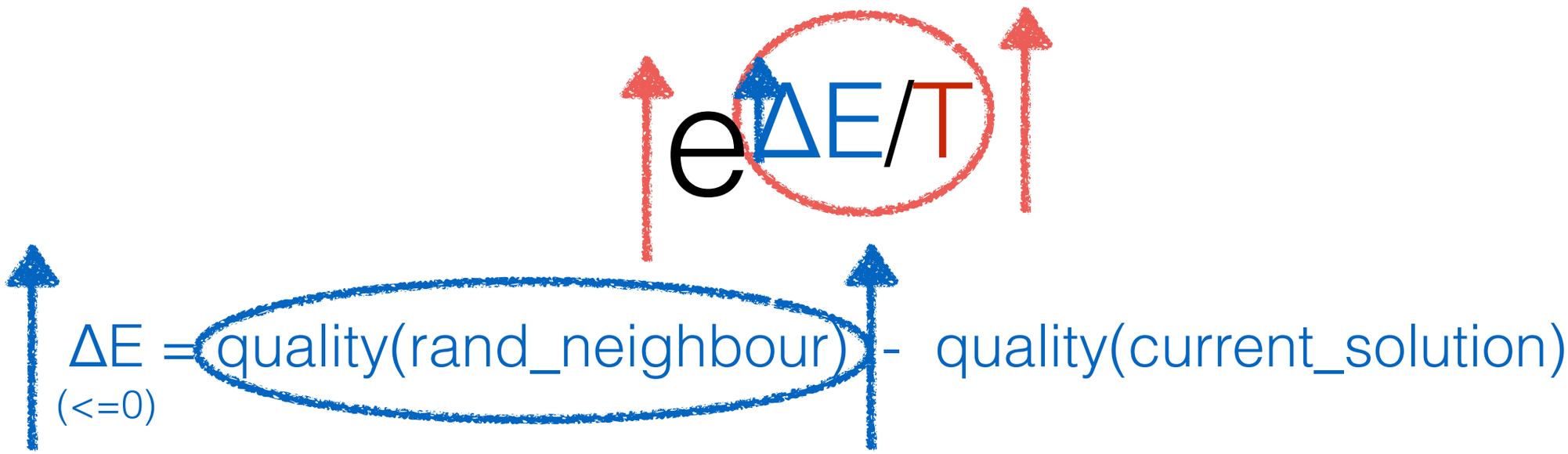
Assuming maximisation…

T = temperature
(>0)

We always have some probability to accept a bad neighbour, no matter how bad it is.

How Does ΔE Affect the Probability?

Probability of accepting a solution of equal or worse quality:



Assuming maximisation…

T = temperature
(>0)

The better the neighbour is, the more likely to accept it.

How Should the Probability be Set?

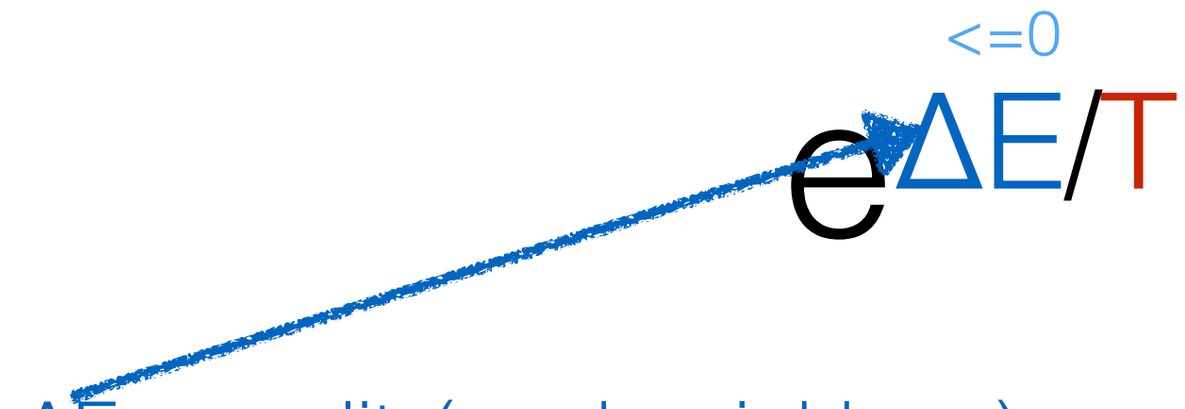
- **Probability to accept solutions with much worse quality should be lower.**
  - **We don't want to be dislodged from the optimum.**

- High probability in the beginning.
  - More similar effect to random search.
  - Allows us to explore the search space.

- Lower probability as time goes by.
  - More similar effect to hill-climbing.
  - Allows us to exploit a hill.

How Does T Affect the Probability?

Probability of accepting a solution of **equal or worse quality**:



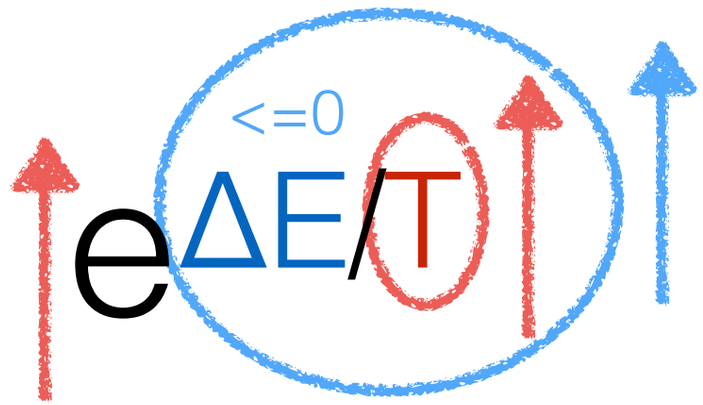
ΔE = quality(rand_neighbour) - quality(current_solution)
(<=0)

Assuming maximisation…

T = temperature
(>0)

How Does T Affect the Probability?

Probability of accepting a solution of equal or worse quality:



ΔE = -10

ΔE/2 = -5

ΔE/5 = -2

ΔE = quality(rand_neighbour) - quality(current_solution)
(<=0)

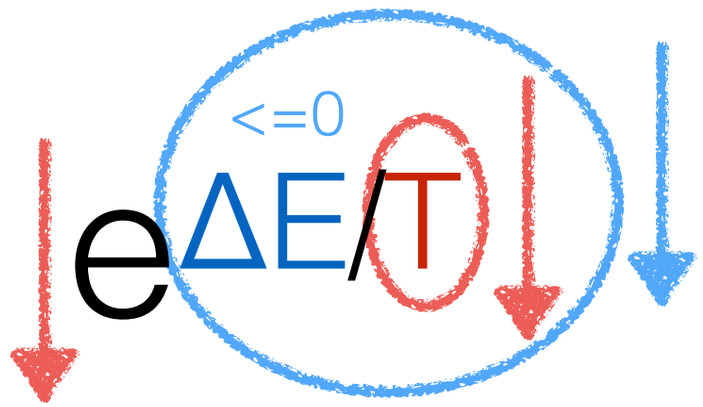
Assuming maximisation…

T = temperature
(>0)

If T is higher, the probability of accepting the bad neighbour is higher.

How Does T Affect the Probability?

Probability of accepting a solution of equal or worse quality:



ΔE = quality(rand_neighbour) - quality(current_solution)
(<=0)

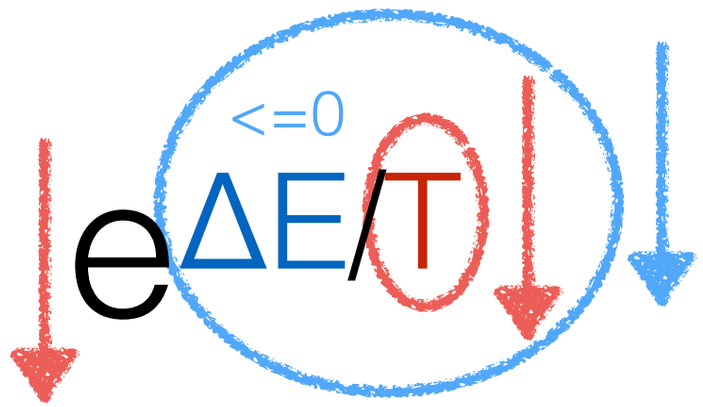
Assuming maximisation…

T = temperature
(>0)

If T is lower, the probability of accepting the bad neighbour is lower.

How Does T Affect the Probability?

Probability of accepting a solution of equal or worse quality:



ΔE = quality(rand_neighbour) - quality(current_solution)
(<=0)

Assuming maximisation…

T = temperature
(>0)

So, reducing the temperature over time would reduce the probability of accepting a bad neighbour.

How Should the Temperature be Set?

- High probability in the beginning.
  - More similar effect to random search.
  - Allows us to explore the search space.

T should start high.

- Lower probability as time goes by.
  - More similar effect to hill-climbing.
  - Allows us to exploit a hill.

T should reduce slowly over time.

How to Set and Reduce T?

- T starts with an initially high pre-defined value (parameter of the algorithm).
- There are different update rules (schedules)…
  - Update rule:
    - $T = \alpha T$,
      α is close to, but smaller than, 1
      e.g., $\alpha = 0.95$

Simulated Annealing

Simulated Annealing (assuming maximisation)

Input: initial temperature Ti

1. current_solution = generate initial solution randomly
2. **T = Ti**
3. Repeat:
   - 3.1 rand_neighbour = generate random neighbour of current_solution
   - 3.2 If quality(rand_neighbour) <= quality(current_solution) {
     - **3.2.1 With probability $e^{\Delta E/T}$,**
       - current_solution = rand_neighbour
     - } Else current_solution = rand_neighbour
   - 3.3 **T = schedule(T)**

   Until a maximum number of iterations

Simulated Annealing

Simulated Annealing (assuming maximisation)

Input: initial temperature Ti, minimum temperature Tf

1. current_solution = generate initial solution randomly
2. **T = Ti**
3. Repeat:
   - 3.1 rand_neighbour = generate random neighbour of current_solution
   - 3.2 If quality(rand_neighbour) <= quality(current_solution) {
     - **3.2.1 With probability $e^{\Delta E/T}$,**
       - current_solution = rand_neighbour
     - } Else current_solution = rand_neighbour
   - 3.3 **T = schedule(T)**

until a minimum temperature Tf is reached or
until the current solution "stops changing"

Optimality

Is simulated annealing guaranteed to find the optimum?

- Simulated annealing is not guaranteed to find the optimum in a reasonable amount of time.
- Whether or not it will find the optimum depends on the termination criteria and the schedule.
- If we leave simulated annealing to run indefinitely, it is guaranteed to find an optimal solution, depending on the schedule used.
- However the time required for that can be prohibitive — even more than the time to enumerate all possible solutions using brute force.
- Therefore, the advantage of simulated annealing is that it can frequently obtain good (near optimal) solutions, by escaping from several poor local optima in a reasonable amount of time.

Time and Space Complexity

- Time complexity:
  - We will run more or less iterations depending on the schedule and minimum temperature / termination criterion.
  - It is possible to compute the time complexity to reach the optimal solution, but it varies depending on the problem and may be even worse than the brute force time complexity, as mentioned in the previous slide.
- Space complexity:
  - Depends on how the design variable is represented in the algorithm.

Summary

- Hill climbing is a greedy local search-based optimisation algorithm.
  - It can reach the top of a hill quickly, but can get study in local optima and plateaux.
- Simulated annealing is a local search-based optimisation algorithm inspired by thermodynamics.
  - It contains mechanisms to try to avoid getting stuck in local optima and plateaux.

Next

- Dealing with constraints.