Random Search

Shin Yoo
CS454, Autumn 2018, School of Computing, KAIST

Random Search

- The polar opposite to the deterministic, examine-everything, search.

- Within the given budget, repeatedly generate a random solution, compare its fitness to the known best, and keep the best one.

Pros and Cons

- VERY easy to implement, inherently automatable, no bias at all.

- Depending on the problem, it may be extremely effective.

- No guidance at all: depending on the problem, it may take forever to obtain a meaningful solution.

Usage of Random Search

- The lack of any guidance provides two useful scenarios.
- First, random search should always be the default **sanity check** against your own search methodology: if it does not no better than random search, you are doing something wrong.

Usage of Random Search

- Somewhat ironically, random search is effective when the underlying problem does not give any guidance to begin with. For example:
  - “Search for the input to program A that will result in program crash”
  - In general, given an arbitrary program, you cannot measure the distance between the current program state and a crash!

Fuzz Testing

- Infinite Monkey Theorem: “Thousand monkeys at a thousand typewriters will eventually type out the entire works of Shakespeare"
- Basic idea: provide a stream of random input to the program, until it crashes (=our Shakespeare).
  - Either a stream of really random bits (naive), or
  - Well-formed input randomly mutated (more effective)

Local Search

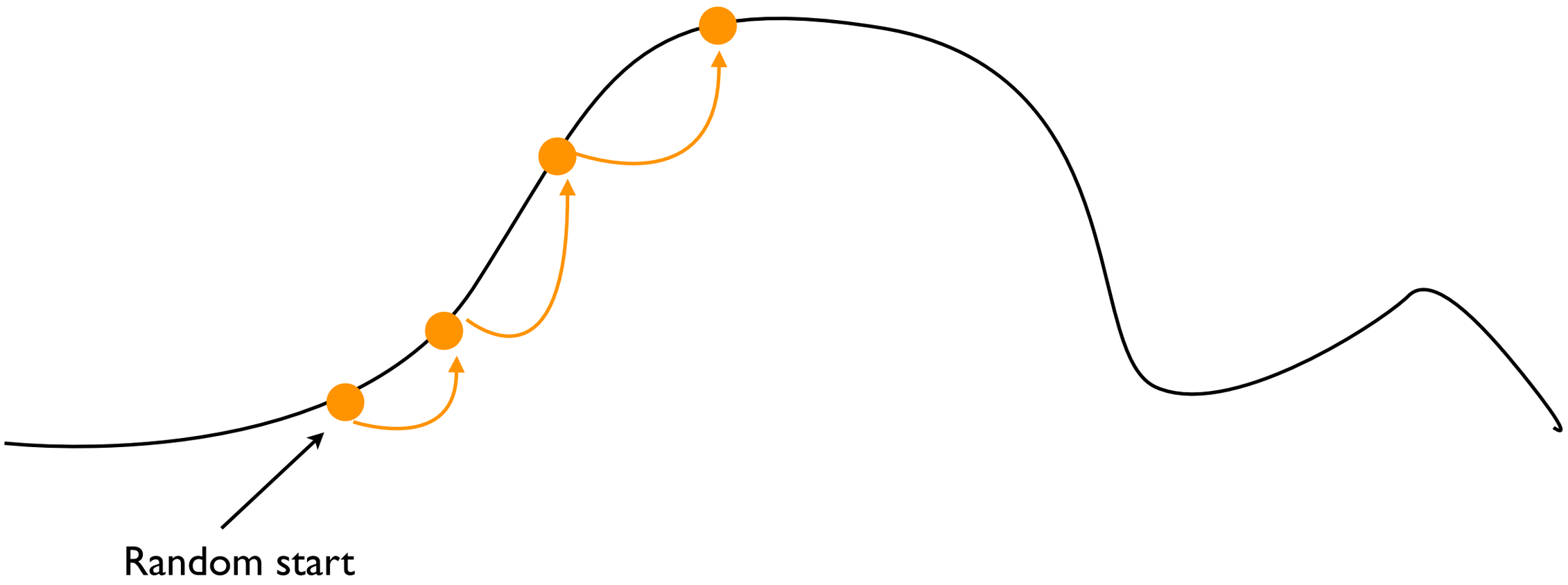
Shin Yoo
CS492D, Fall 2015, School of Computing, KAIST

If your problem forms a fitness **landscape**, what is optimisation?

Local Search Loop

- Start with a single, random solution
- Consider the neighbouring solutions
- Move to one of the neighbours if better
- Repeat until no neighbour is better

Local Search



Hill Climbing Algorithm

- This particular variation is also known as the steepest-ascent hill climbing.
- Why? :)
- What other versions are there?

```
HillClimbing()
(1)     climb ← True
(2)     s ← GetRandom()
(3)     while climb
(4)         N ← GetNeighbours(s)
(5)         climb ← False
(6)         foreach n ∈ N
(7)             if Fitness(n) > Fitness(s)
(8)                 climb ← True
(9)                 s ← n
(10)            return s
```

Hill Climbing Algorithm

```
HillClimbing()
(1)     climb ← True
(2)     s ← GetRandom()
(3)     while climb
(4)         N ← GetNeighbours(s)
(5)         climb ← False
(6)         foreach n ∈ N
(7)             if Fitness(n) > Fitness(s)
(8)                 climb ← True
(9)                 s ← n
(10)            return s
```

Steepest Ascent

```
HillClimbing()
(1)     climb ← True
(2)     s ← GetRandom()
(3)     while climb
(4)         N ← GetNeighbours(s)
(5)         climb ← False
(6)         foreach n ∈ N
(7)             if Fitness(n) > Fitness(s)
(8)                 climb ← True
(9)                 s ← n
(10)                break
(11)            return s
```

First Ascent

Hill Climbing Algorithm

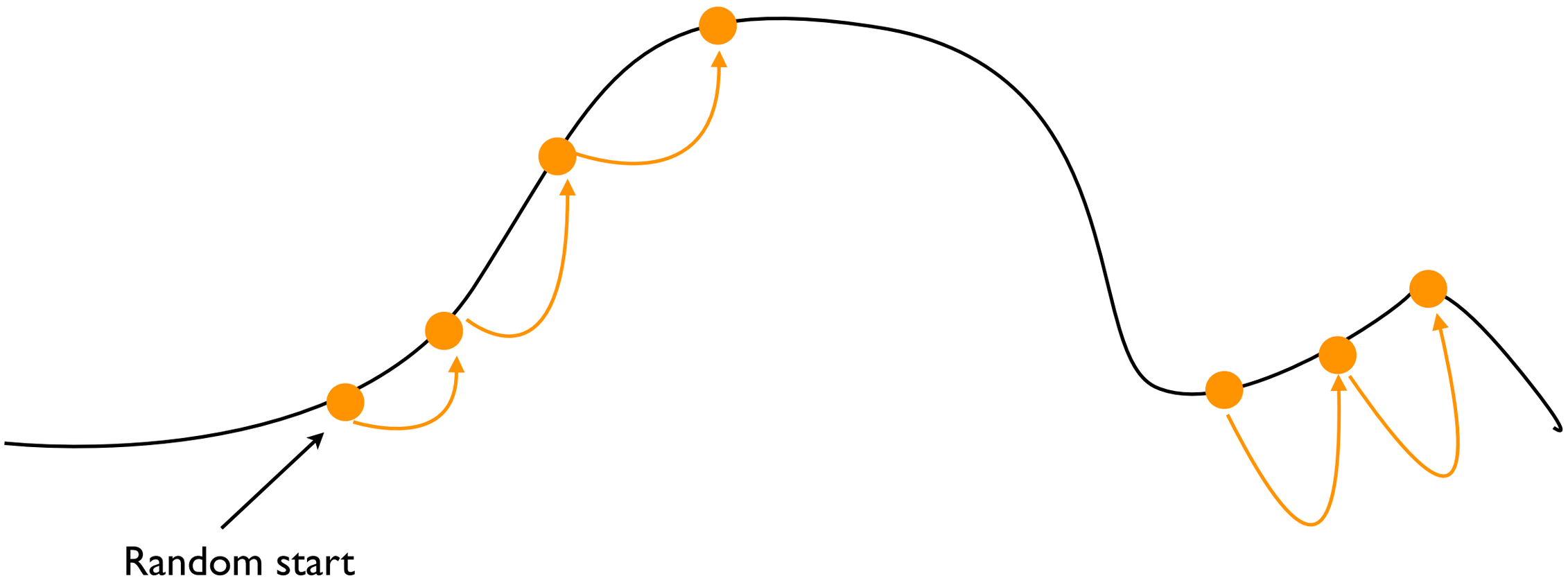
```
HillClimbing()
(1)     climb ← True
(2)     s ← GetRandom()
(3)     while climb
(4)         N ← GetNeighbours(s)
(5)         climb ← False
(6)         foreach n ∈ N
(7)             if Fitness(n) > Fitness(s)
(8)                 climb ← True
(9)                 s ← n
(10)                break
(11)            return s
```

First Ascent

```
HillClimbing()
(1)     s ← GetRandom()
(2)     while True
(3)         N ← GetNeighbours(s)
(4)         N′ ← {n ∈ N|Fitness(n) > Fitness(s)}
(5)         if |N′| > 0
(6)             s ← RandomPick(N′)
(7)         else
(8)             break
(9)     return s
```

Random Ascent

Local Search



Ascent Strategy

- Not possible to know which one is better.
- IF the current solution is near a local optimum, slowing down the ascent may (or may not) be better.
- Regardless of strategy, hill climbing monotonically climbs, until it reaches local/global optimum; it never goes down.

Pros/Cons

- Pros:cheap (fewer fitness evaluations compared to GAs), easy to implement, repeated applications can give insights into the landscape, suitable for solutions that need to be built through small incremental changes

- Cons: more likely to get stuck in local optima, unable to escape local optima

Simulated Annealing

- Big question: how do we escape local optima, if we are one?
    - Thought 1: we never know whether we are climbing a local or a global optimum!
    - Thought 2: assuming that there are more local than global optima, it makes sense to escape.
    - Thought 3: but not always - when we stop, we want to stop near the top of SOME optimum.

Annealing (풀림)

- Keep metal in a very high temperature for a long time, and then slowly cool down: it then becomes more workable.

- At high temperature, atoms are released from internal stress by the energy; during the cool-down, they form new nucleates without any strain, becoming softer.

Simulated Annealing

- Introduce “temperature” into local search: start with a high temperature, and slowly cool down.
  - When the temperature is high, the solution (like atom) is unstable and can make random moves (i.e. escapes).
  - As the temperature decreases, the energy level gradually gets lower, and escapes become more infrequent.

Simulated Annealing

SimulatedAnnealing()

(1) $s = s_0$
(2) $T \leftarrow T_0$
(3) **for** $k = 0$ **to** n
(4) $s_{new} \leftarrow$ GetRandomNeighbour(s)
(5) **if** $P(\mathrm{F}(s), \mathrm{F}(s_{new}), T) \geq random(0,1)$ **then** $s \leftarrow s_{new}$
(6) $T \leftarrow$ Cool(T)
(7) **return** s

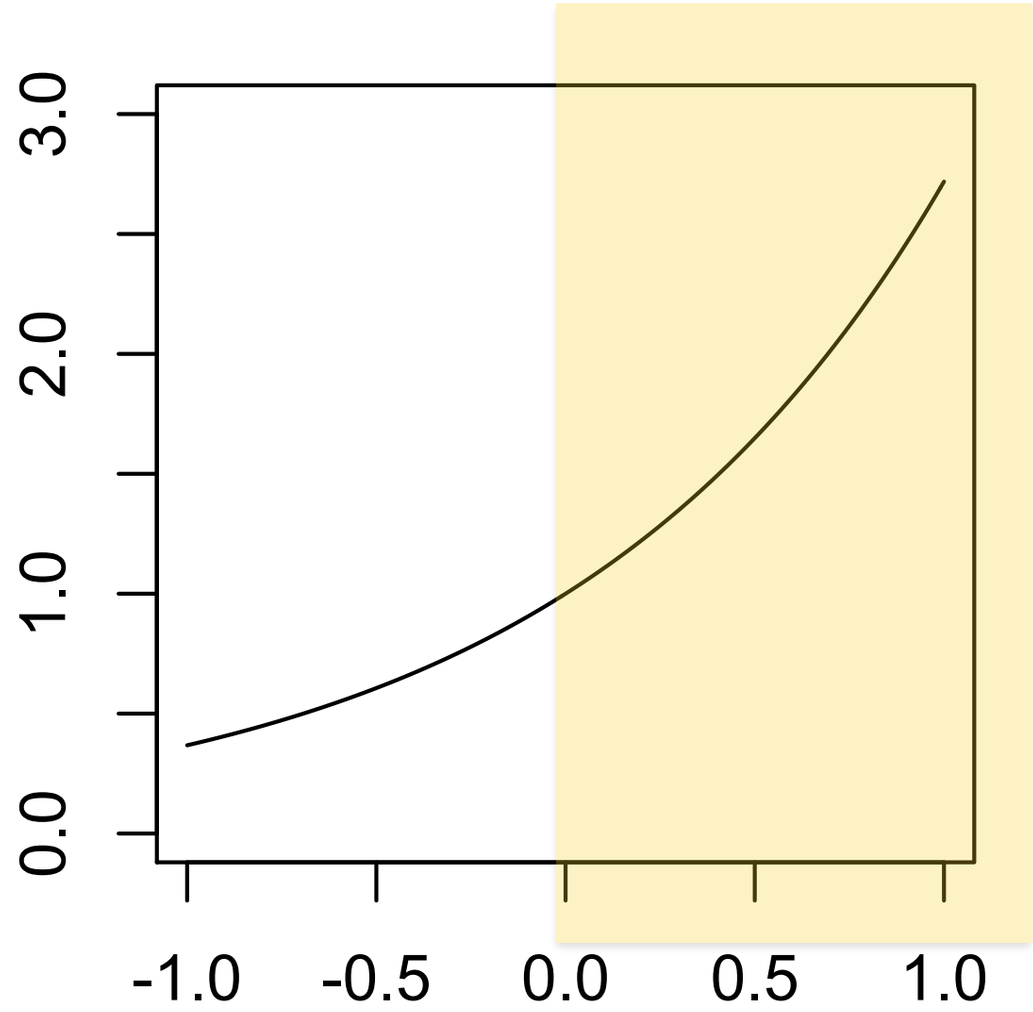
P($F(s)$,$F(s_{new})$, T)

(1) **if** $F(s_{new}) > F(s)$ **then return** 1.0
(2) **else return** $e^{\frac{F(s_{new})-F(s)}{T}}$

Acceptance Probability

$$P = e^{\frac{F(s_{new}) - F(s)}{T}}$$

- Borrowed from metallurgy
- When new solution is better ($F(s_{new}) > F(s)$), always accept ($P > 1$)
- When new solution is equally good, accept
- When new solution is worse:
  - more likely to accept small downhill movement
  - gets smaller as temperature drops



Cooling Schedule

Temperature at time step t as a function of t:

- Linear: $T(t) = T_0 - \alpha t$
- Exponential: $T(t) = T_0 \alpha^t (0 < \alpha < 1)$
- Logarithmic $T(t) = \frac{c}{\log(t+d)}$

  - With large c, this can be very slow cooling
  - There is an existence proof that says logarithmic will find the global optimum in infinite time… huh?
  - It becomes essentially a random search
  - Theoretically interesting, but practically not so much.

Tabu Search

- Another attempt to escape local optima
- Two exceptions to local search:
  - It is possible to accept a worse move
  - Remember “visited” solutions and avoid coming back

Tabu Search

TabuSearch()
(1) $s \leftarrow s_0$
(2) $s_{best} \leftarrow s$
(3) $T \leftarrow []$ // tabu list
(4) **while** not stoppingCondition()
(5) $c_{best} \leftarrow null$
(6) **foreach** $c \in$ GetNeighbours(s)
(7) **if** $(c \notin T) \wedge (F(c) > F(c_{best}))$ **then** $c_{best} \leftarrow c$
(8) $s \leftarrow c_{best}$
(9) **if** $F(c_{best}) > F(s_{best})$ **then** $s_{best} \leftarrow c_{best}$
(10) append(T, c_{best})
(11) **if** $|T| > maxTabuSize$ **then** removeAt$(T, 0)$
(12) **return** sBest

Tabu list is a FIFO queue: with the **`maxTabuSize`** we can control the memory span of the search.

Random Restart

- Search budget is usually given in limited time ("terminate after 5 minutes") or in number of fitness evaluation ("terminate after 5000 fitness evaluations")

- If a local search reaches optima and budget remains? Start again from another random solution and keep the best answer across multiple runs.

Search Radius

- For local search algorithms to be effective: the search space may be large, but the search radius should be reasonably small

- Search radius: the number of moves required to go **across** the search space

Search Radius: TSP

- Travelling Salesman Problem: what is the shortest path that visits all N cities?

- Search Space: N! (e.g 2,432,902,008,176,640,000 when N = 20)

- Search Radius: at most N(N-1)/2 swaps to change any permutation of cities to any other (e.g. 190 when N=20)

Summary

- Local search: direct use of fitness landscape concept, with various mechanism to escape local optima.

- Easy to implement, easy to understand what is going on; good for insights into landscape

- Design of search space (especially discrete one) affects the performance of search