Iterative Algorithms

Song Liu (song.liu@bristol.ac.uk)
GA 18, Fry Building,
Microsoft Teams (search "song liu").

Search Problem

$$\{i^*\} = \arg\min_{i \in \mathcal{F}} f(i)$$

- where $\mathcal{F}$ defines a **search space**,
- f defines the **search criterion**.

Greedy Algorithm

Greedy algorithm is the name of **a set of search algorithms**.

1. Greedy algorithm finds the best solution(s) to a smaller and more manageable **subproblem**.
2. From the solution, it **revises** the subproblem and solves a new subproblem.
3. Until a certain stopping criterion is satisfied.

Hill Climbing

$$x_{t+1}^* = \arg \max_{x \in [x_t^* - \epsilon, x_t^* + \epsilon]} f(x),$$

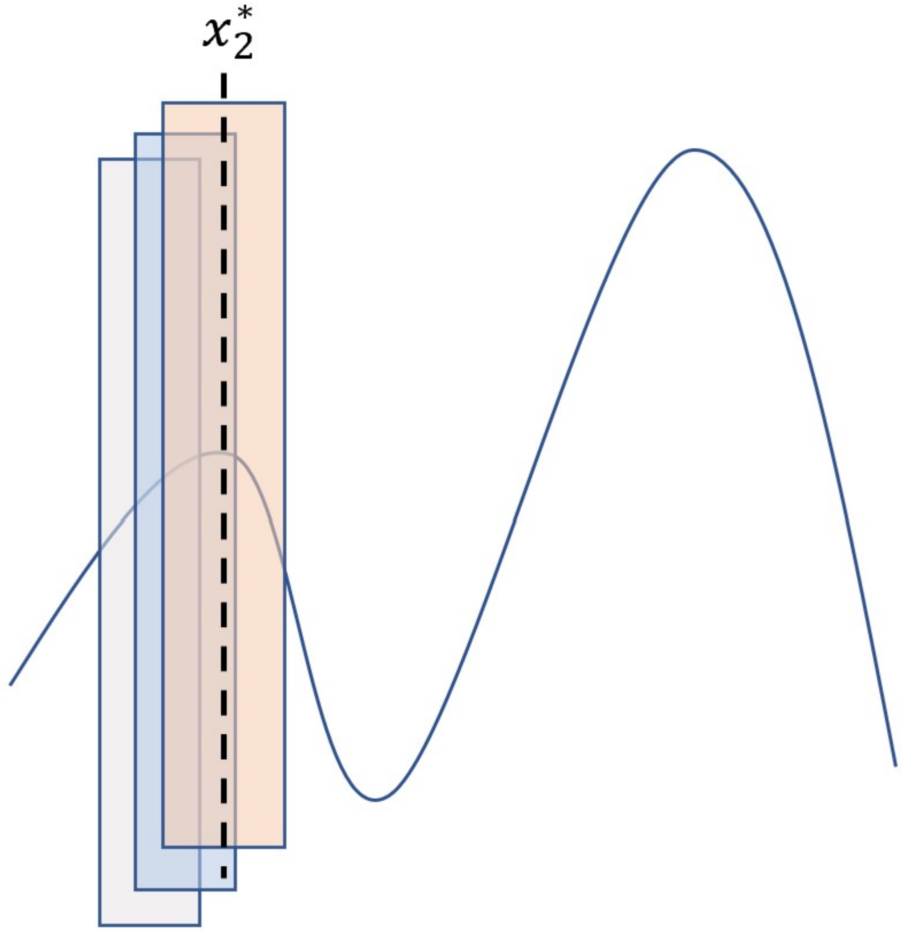
- Show Demo.

Limitations of Greedy Algorithm

Greedy Algorithm is a myopic algorithm: **It may not lead to the global optimal solution.**

- Hill-climbing does not find the global maximum if the search space of each subproblem is too small.
- TicTacToe does not anticipate opponent moves leading to a suboptimal move.

Local Optimum



Algorithm stuck at x_2^*!

A Suboptimal Move

```
X O *
X X O
* * *
AI plays...
X O O
X X O
* * *
```

It does not anticipate your opponent's move (X checkmate!)

Iterative Algorithm

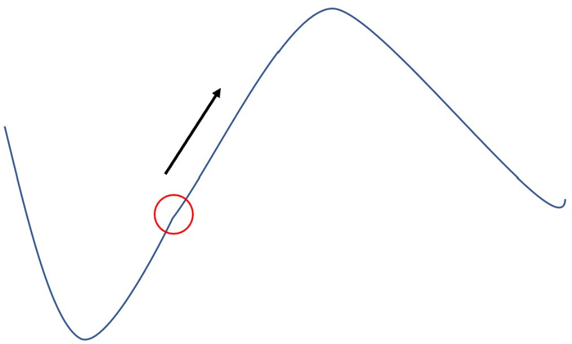
- The hill climbing algorithm is also an iterative algorithm:
  - It starts searching from an initial value x_0.
  - It solves a sequence of search problems:
    - $x^*_{t+1} = \arg \max_{x \in [x^*_t - \epsilon, x^*_t + \epsilon]} f(x)$,
    - where $t+1$-th problem is derived from the t-th
  - It terminates when x^*_{t+1} converges.
- **Iterative algorithms** seek to approximate the solution to a numeric problem by **successive improvements**.
  - At each iteration, it revises the current approximation, so that it is closer to the true solution.
  - The algorithm stops when a stopping criteria is met.

Iterative Algorithm

- Iterative algorithm **does not have to be greedy** when improving your approximation.
- It is fine even if the **approximate** is random at each iteration.
    - We will see an example of that.

The Slope of a Function

- From the previous example, one can see our hill climbing algorithm is quite naive.
  - It does not realize the **slope** of $f(x)$.

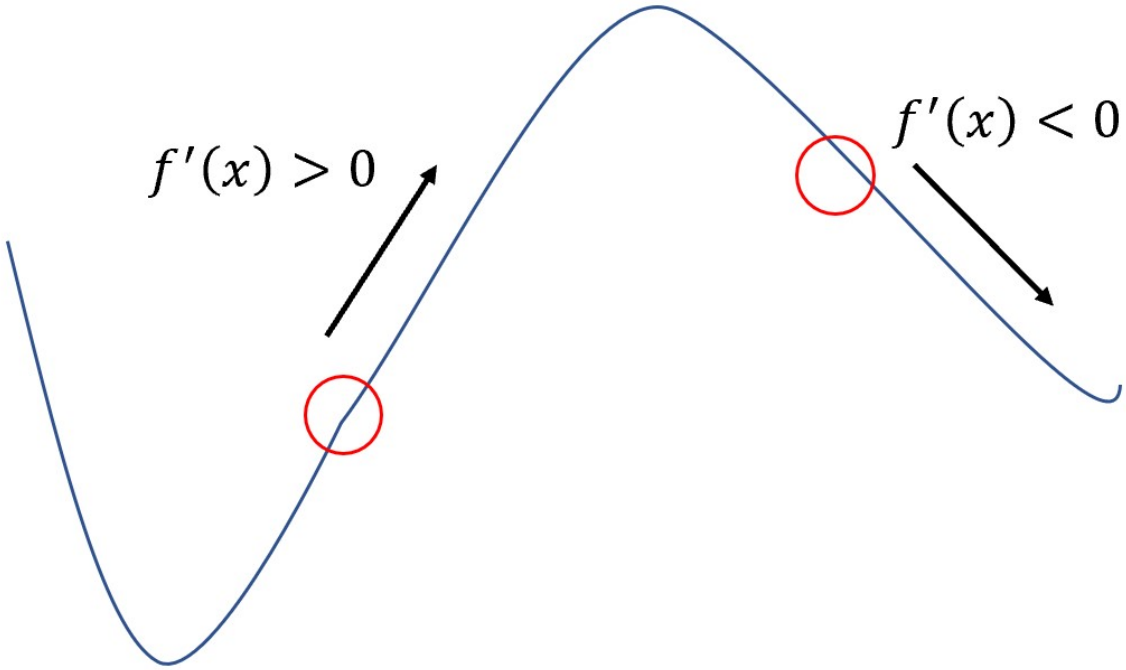


- For **a differentiable function**, its slope indicates the direction you should go to maximize/minimize your function.
  - To maximize, climb up the slope.
  - To minimize, slide down the slope.

The Slope of a Function

What is the slope of a differentiable function?

- Its derivative!



Gradient Ascent Algorithm

- Gradient Ascent is an iterative algorithm for solving the following search problem:
  $x^* = \arg\max_{x \in \mathcal{X}} f(x),$
  where $\mathcal{X}$ is the search space, e.g., $\mathbb{R}$.
- It starts with a random guess x_0.
- At iteration t, it revise the current guess x_t by using the derivative (gradient) information $f'(x_t)$.
  - The revision does not have to be greedy!
- Repeat until stopping criteria is met.

Gradient Ascent Algorithm

Pseudo Code: Gradient Ascent

- initialize x_0 with a random guess
- For t = 0 to T
  - $x_{t+1} \leftarrow x_t + \epsilon f'(x_t)$.
    - ϵ is the **step size**, a small number, say 0.1.
  - If $|x_{t+1} - x_t| < \eta$, stop the loop.
    - η is an extremely small number, say 1e-5.
- x_t is your approximation to x^*.

Gradient Ascent Algorithm

```
double x0 = -4; //initial guess
int T = 10000; //maximum iteration
double epsilon = .1; // step size
double eta = 1e-5; // stopping threshold

double xt = x0;
for(int t=0; t<T; t++){
    // gradient ascent!
    double xt1 = xt + epsilon*df(xt);
    // do we stop?
    if (fabs(xt1 - xt) < eta){
        xt = xt1;
        break;
    }
    xt = xt1;
}

  //NO need to search a grid!
```

Gradient Asecent

```
f(-4.000000) = -0.368995)
f(-3.999771) = -0.368995)
f(-3.999537) = -0.368994)
...
f(2.539099) = 1.760173)
f(2.539110) = 1.760173)
f(2.539120) = 1.760173)
f(2.539130) = 1.760173)
f(2.539140) = 1.760173)
```

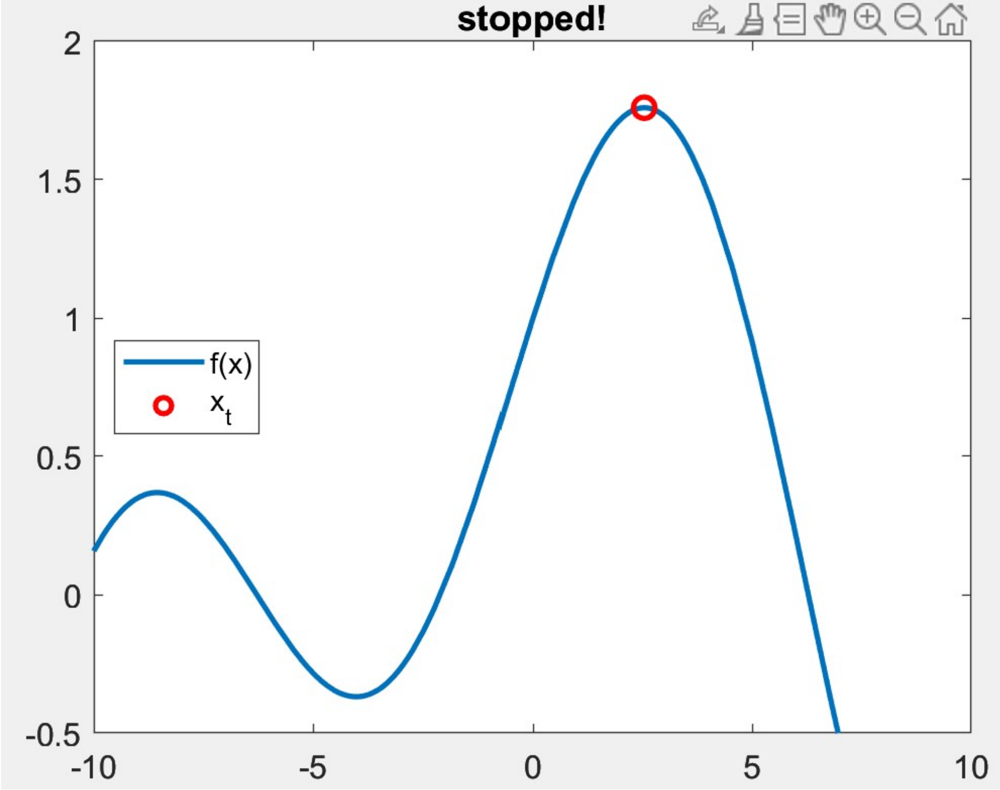
Show demo.

Local Optimum

- Like Hill Climbing algorithm, gradient descent is not guaranteed to return the global maximum solution.
  - Like Hill Climbing algorithm, the gradient ascent is also not aware of the global structure of $f(x)$.
- It may be stuck at a **local optimum**.

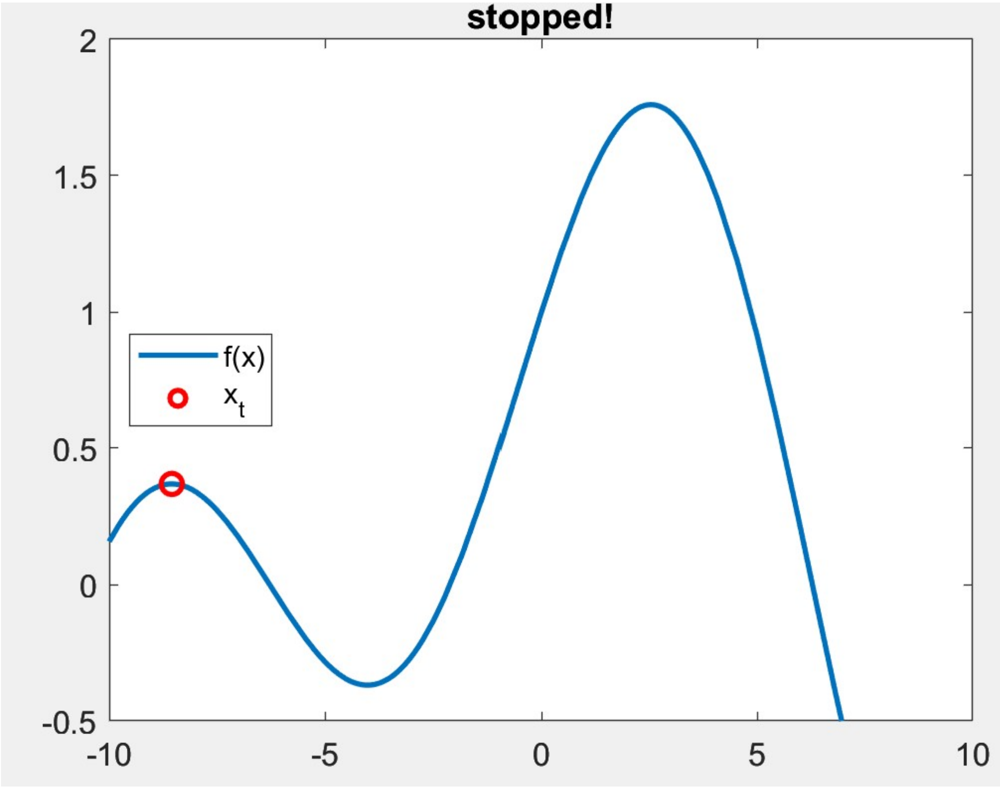
Local Optimum

$x_0 = -3$:



Local Optimum

$x_0 = -5$:



Gradient Ascent vs. Hill Climbing

Similarities:

- They both search for the maximum of $f(x)$.
- They are both iterative algorithms.
- They both use local structure of function $f(x)$.
  - They both may be stuck at a local optimum.

Dissimilarities:

- Hill climbing evaluates $f(x), x \in [x_t - \epsilon, x_t + \epsilon]$ while gradient ascent evaluates $f'(x_t)$.
  - **Gradient ascent cannot be applied to non-differentiable functions** (e.g. $f(x) = |x|$), but Hill climbing can.
- Gradient ascent does not seek for "the best" solution of a subproblem. It only makes an improvement.

Gradient Ascent vs. Hill Climbing

- Time Complexity:
- Suppose
  - Evaluating $f(x)$ and $f'(x)$ takes the unit time.
  - Both algorithm run for T steps before stop.
  - There are total G grid points in the hill climbing search interval $[x_t - \epsilon, x_t + \epsilon]$.
- Hill Climbing: $O(GT)$
- Gradient Ascent: $O(T)$

Gradient Ascent for Multivariate Functions

- The idea of gradient ascent easily extends to multivariate functions.
  $x^* = \arg\max_{x \in \mathcal{X}} f(x_1, x_2),$
  where $\mathcal{X} := \mathbb{R}^2$.
- The gradient of a function is defined as
  $\nabla f(x_1, x_2) := \begin{bmatrix} \partial_{x_1} f(x_1, x_2) \\ \partial_{x_2} f(x_1, x_2) \end{bmatrix}$
- The gradient of a multivariate function is a **vector** that points to the direction where f increases the fastest.

Gradient Ascent Algorithm

Pseudo Code: Gradient Ascent (bivariate)

- initialize x_0 with a random guess
- For t = 0 to T
  - $x_{t+1} \leftarrow x_t + \epsilon \nabla f(x_1, x_2)\big|_{(x_1,x_2)=x_t}$.
    - ϵ is the **step size**, a small number, say 0.1.
  - If $\|x_{t+1} - x_t\| < \eta$, stop the loop.
    - ϵ is a extremely small number, say 1e-5.
- x_t is your approximation to x^*.

Curse of Dimensionality

- Hill climbing algorithm can also be generalized to multivariate function.
- However, the number of grid points G grows exponentially with dimensionality of your search space.
  - Number of unit length intervals in $[0, 10]$: 10.
  - Number of unit squares in $[0, 10]^2$: 100.
  - Number of unit cubes in $[0, 10]^3$: 1000.
- Time complexity of Hill Climbing grows exponentially with dimensionality of your search space!
- Gradient Ascent does not have such a problem, assuming the gradient of f can be easily evaluated.

Conclusion

- **Iterative algorithms** seek to approximate the solution to a numeric problem by **successive improvements**.
- Gradient ascent solves the following problem:
  $x^* = \arg\max_{x \in \mathcal{X}} f(x)$
- Gradient ascent revises the current guess x_t by using the derivative (gradient) information $f'(x_t)$.
- Gradient ascent may also be stuck at the local optimum.
- Gradient ascent easily generalizes to multivariate functions without suffering from the curse of dimensionality.

Homework 1.

1. Open `ab1234.cpp`
2. Implement the greedy algorithm according to the skeleton code.

Homework 2.

3. Verify your implementation with the example output provided in the slides.
4. By changing `x0`, could you get your greedy algorithm stuck on one of the local optimums?
   - By changing `epsilon` (and keeping `x0`), could the algorithm recover the global optimal solution again?

Homework 3 (optional)

5. Change the function `f` from

```
double f(double x) // f(x),
{
    return sin(x / 2) + cos(x / 4);
}
```

to

```
virtual double f(double x) // adding "virtual"
{
    return sin(x / 2) + cos(x / 4);
}
```

Homework 3 (optional)

6. Now, write a new class:

```
class NewProblem: public Problem
{
  // we rewrite f(x) for the new problem
  double f(double x){
    return -x*x;
  }
public:
  // constructor for the new class
  NewProblem(double epsilon, double initial_guess)
    : Problem(epsilon, initial_guess){}
}
```

and add the following two lines of code in `main`

```
NewProblem p2(.5, -4);
p2.solve(); // what do we maximize?
```

Homework 3 (optional)

7. What do you see? Does the `NewProblem` class behave as you expected?
    - Once you declare the function to be `virtual` in a parent class, you are allowed to **override** its behavior in a child class.
    - In the child class, your new `f` will replace the parent `f` in all methods (including inherited methods).
    - This is another way, OOP allows you to reuse your code.
8. If you are interested, see Virtual Function.

Homework 4 (submit)

9. Write a new public method `void GAsolve()` in `Problem` class:
    - It finds the maximizer of $f(x)$ using gradient ascent algorithm.
10. Call `GAsolve()` in main (use the same `Problem` object).
11. Change the initial guess, observe gradient ascent produces a locally optimal solution.

Homework 4 (submit)

12. Using the same `x_0` and `epsilon`, do `GAsolve` and `solve` produce the exact same solution?
    - If not, how different are they?
    - Which one produces a better solution?