Introduction to Algorithms

What is an Algorithm?

T. C. Lu

1. What's the difference between the analytical and numerical methods to the problems?
   Find the root of $f(x) = x - 3$:
   (1) An analytical way:
       $x - 3 = 0$, add $+3$ to both sides to get the answer $x = 3$.
   (2) A numerical way:
       Guess $x = 1$, we have $f(1) = -2 < 0$;
       guess $x = 7$, we get $f(7) = 4 > 0$.
       Let's try $x = \frac{1+7}{2}$, we get $f(4) = 1 > 0$. So the answer must be between 1 and 4 …etc.
2. Analytical methods give exact solutions, whereas numerical methods give approximate solutions with allowable tolerance.
   (1) Most real-world problems do not have analytical or explicit forms of the function at all.
   (2) Try to solve $x^7 - 3x^5 + \cos(x) - 5 + e^{2x} - \frac{1}{x} = 0$.
3. An algorithm A uses a step-by-step iterative procedure to generate a new and better solution x_{t+1} from the current solution x_t, that is $x_{t+1} = A(x_t)$.
   (1) An algorithm does not generate an exact solution directly in one step. It starts from a guess solution and improves it gradually toward the true solution.
4. Example:
   A simple algorithm of finding $\sqrt{k}$, where $k > 0$:

$$x_{t+1} = \frac{1}{2}\left(x_t + \frac{k}{x_t}\right).$$

   (1) To find $\sqrt{7}$, starting from a guess solution $x_0 = 1$, then we have

$$x_1 = \frac{1}{2}\left(x_0 + \frac{7}{x_0}\right) = \frac{1}{2}\left(1 + \frac{7}{1}\right) = 4,$$

$$x_2 = \frac{1}{2}\left(x_1 + \frac{7}{x_1}\right) = \frac{1}{2}\left(4 + \frac{7}{4}\right) = 2.875,$$

   $x_3 \approx 2.654891304$, $x_4 \approx 2.645767044$, $x_5 \approx 2.6457513111$.
   (the true value of $\sqrt{7} = 2.64575131106459...$)

(2) A good choice of the initial value x_0 will speed up the convergence. A wrong choice of x_0 could make the iteration failx. This is a very common feature and disadvantage of deterministic procedures or algorithms.

Newton's Method

T. C. Lu

- Newton's method is a widely used method for finding the zeros of a nonlinear single-variable function $f(x)$.
- Newton's formula:

$$x_{t+1} = x_t - \frac{f(x_t)}{f'(x_t)}.$$

  - The iteration procedure starts from an initial guess x_0 and continues until a certain criterion is met.
  - A good initial guess will use fewer number of steps. But if the initial value is too far away from the true solution, the iteration process may fail.
  - For nonlinear equations, there are often multiple roots, and the choice of initial guess may affect the root into which the iterative procedure could converge.
- Example:

$f(x) = x^x - e^x = 0$ has two roots: 0 and $e \cong 2.718281828459$.

  - $f'(x) = x^x(\ln x + 1) - e^x$.
  - Starting from $x_0 = 5$:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = 5 - \frac{5^5 - e^5}{5^5(\ln 5 + 1) - e^5} = 4.6282092$$

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} = 5.2543539$$

$$\vdots$$

$$x_{10} = x_9 - \frac{f(x_9)}{f'(x_9)} = 2.7182818$$

$\Rightarrow$The solution x_{10} is very close to the true solution e.

  - Starting from $x_0 = 10$:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = 5 - \frac{5^5 - e^5}{5^5(\ln 5 + 1) - e^5} = 9.6972073$$

$$\vdots$$

$$x_{25} = x_{24} - \frac{f(x_{24})}{f'(x_{24})} = 2.7182819$$

➔The convergence is very slow.

- Starting from $x_0 = 1$:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = 1 - \frac{1^1 - e^1}{1^1(\ln 1 + 1) - e^1} = 0$$

➔It is the exact solution for the other root $x^* = 0$.

- If we start from 0 or less than 0, this formula does not work because of the singularity in logarithms. In fact, if we start from any value from 0.01 to 0.99, this will not work either; neither does the initial value $x_0 = 2$.
- This highlights the importance of choosing the right initial value.

● The Newton's formula can be extended to find the maximum or minimum of $f(x)$, which is equivalent to finding the roots of $f'(x) = 0$.

$$x_{t+1} = x_t - \frac{f'(x_t)}{f''(x_t)}.$$

Optimization

- $\underset{\boldsymbol{x}\in\mathbb{R}^d}{\text{minimize}}\, f_i(\boldsymbol{x})$, ($i=1, 2, \ldots, M$),

  subject to $h_j(\boldsymbol{x})=0$, ($j=1, 2, \ldots, J$),

  $g_k(\boldsymbol{x})\le 0$, ($k=1, 2, \ldots, K$),

  where $f_i(\boldsymbol{x})$, $h_j(\boldsymbol{x})$ and $g_k(\boldsymbol{x})$ are functions of $\boldsymbol{x}=(x_1, x_2, \ldots, x_d)^T$.

  - x_i is *design variable*, it can be real continuous or discrete.
    $f_i(\boldsymbol{x})$ is *objective function* or *cost function.*

    $h_j(\boldsymbol{x})$ is *equality constraint.*

    $g_k(\boldsymbol{x})$ is *inequality constraint.*
  - The space spanned by the design variables is called the *design space* or *search space* $\mathbb{R}^d$. The space formed by the objective function values is called the *solution space*.
  - According to the number of objectives, optimization problems can be classified into single-objective ($M=1$) and multi-objective ($M>1$).
  - For the equality constraints, it is very difficult to get sampling points that satisfy the equality exactly. Some tolerance or allowance is used in practice.
  - If both the objective functions and the constraints are all linear, it becomes a linear programming problem.
    - ✧ Example:

      maximize $3x_1+2x_2$,

      subject to $2x_1+x_2\le 100$,

      $x_1+x_2\le 80$,

      $x_1\le 40$,

      $x_1, x_2\ge 0$.

Gradient-Based Algorithms

- The *gradient* of $f(\boldsymbol{x})$ is defined as

$$\nabla f(\boldsymbol{x}) = \begin{bmatrix} \dfrac{\partial f(\boldsymbol{x})}{\partial x_1} \\ \vdots \\ \dfrac{\partial f(\boldsymbol{x})}{\partial x_d} \end{bmatrix}.$$

- The *Hessian* of $f(\boldsymbol{x})$ is defined as

$$\nabla^2 f(\boldsymbol{x}) = \begin{bmatrix} \dfrac{\partial^2 f(\boldsymbol{x})}{\partial x_1^2} & \dfrac{\partial^2 f(\boldsymbol{x})}{\partial x_2 \partial x_1} & \cdots & \dfrac{\partial^2 f(\boldsymbol{x})}{\partial x_d \partial x_1} \\ \dfrac{\partial^2 f(\boldsymbol{x})}{\partial x_1 \partial x_2} & \dfrac{\partial^2 f(\boldsymbol{x})}{\partial x_2^2} & \cdots & \dfrac{\partial^2 f(\boldsymbol{x})}{\partial x_d \partial x_2} \\ \vdots & \vdots & \ddots & \vdots \\ \dfrac{\partial^2 f(\boldsymbol{x})}{\partial x_1 \partial x_d} & \dfrac{\partial^2 f(\boldsymbol{x})}{\partial x_2 \partial x_d} & \cdots & \dfrac{\partial^2 f(\boldsymbol{x})}{\partial x_d^2} \end{bmatrix}.$$

- For a multi-variable function $f(\boldsymbol{x})$ to be optimized, Newton's formula can be written as

$$\boldsymbol{x}^{(t+1)} = \boldsymbol{x}^{(t)} - \left(\nabla^2 f(\boldsymbol{x}^{(t)})\right)^{-1} \nabla f(\boldsymbol{x}^{(t)}).$$

- Modified Newton's formula:

$$\boldsymbol{x}^{(t+1)} = \boldsymbol{x}^{(t)} - \alpha\left(\nabla^2 f(\boldsymbol{x}^{(t)})\right)^{-1} \nabla f(\boldsymbol{x}^{(t)})$$

  where $\alpha \in (0,1]$ is step size.

  — A small step size means slow movement toward the minimum, whereas a large step size may overshoot.

  — The step size $\alpha = \alpha^{(t)}$ should be different at each iteration.

- Quasi-Newton method:

$$\boldsymbol{x}^{(t+1)} = \boldsymbol{x}^{(t)} - \alpha^{(t)} \nabla f(\boldsymbol{x}^{(t)}),$$

  which is essentially the steepest descent method.

  — At each iteration, the gradient and step size will be calculated. Again, a good initial guess of both $\boldsymbol{x}^{(0)}$ and $\alpha^{(0)}$ is useful.

- Example:

  Minimize $f(x_1, x_2) = 10x_1^2 + 5x_1x_2 + 10(x_2 - 3)^2$,

  where $x_1 \in [-10, 10]$ and $x_2 \in [-15, 15]$.

— $\nabla f(\boldsymbol{x}) = \begin{bmatrix} 20x_1 + 5x_2 \\ 5x_1 + 20x_2 - 60 \end{bmatrix}$.

— Starting from $\boldsymbol{x}^{(0)} = \begin{bmatrix} 10 \\ 15 \end{bmatrix}$, then we have $\nabla f(\boldsymbol{x}^{(0)}) = \begin{bmatrix} 275 \\ 290 \end{bmatrix}$.

— At the first iteration, $\boldsymbol{x}^{(1)} = \boldsymbol{x}^{(0)} - \alpha^{(0)} \nabla f(\boldsymbol{x}^{(0)}) = \begin{bmatrix} 10 - 275\alpha^{(0)} \\ 15 - 290\alpha^{(0)} \end{bmatrix}$.

— The step size $\alpha^{(0)}$ should be chosen such that $f(\boldsymbol{x}^{(1)})$ is at the minimum, which means that

$$f(\alpha^{(0)}) = 10(10 - 275\alpha^{(0)})^2 + 5(10 - 275\alpha^{(0)})(15 - 290\alpha^{(0)}) + 10(12 - 290\alpha^{(0)})^2$$

should be minimized. This becomes an optimization problem for a single-variable $\alpha^{(0)}$. Newton's method can be used to get $\alpha^{(0)} \approx 0.04001$.

— At the second step, we have

$$\nabla f(\boldsymbol{x}^{(1)}) = \begin{bmatrix} -3.078 \\ 2.919 \end{bmatrix}, \quad \boldsymbol{x}^{(2)} = \boldsymbol{x}^{(1)} - \alpha^{(1)} \nabla f(\boldsymbol{x}^{(1)}) = \begin{bmatrix} -1.00275 + 3.078\alpha^{(1)} \\ 3.3971 - 2.919\alpha^{(1)} \end{bmatrix}.$$

The minimization of $f(\alpha^{(1)})$ gives $\alpha^{(1)} \approx 0.066$, and $\boldsymbol{x}^{(2)} \approx \begin{bmatrix} -0.797 \\ 3.202 \end{bmatrix}$.

— At the third iteration, we have

$$\nabla f(\boldsymbol{x}^{(2)}) = \begin{bmatrix} 0.060 \\ 0.064 \end{bmatrix}, \quad \boldsymbol{x}^{(3)} = \boldsymbol{x}^{(2)} - \alpha^{(2)} \begin{bmatrix} 0.060 \\ 0.064 \end{bmatrix}.$$

The minimization of $f(\alpha^{(2)})$ leads to $\alpha^{(2)} \approx 0.040$, and thus

$$\boldsymbol{x}^{(3)} \approx \begin{bmatrix} -0.8000299 \\ 3.20029 \end{bmatrix}.$$

Then the iterations continue until a prescribed tolerance is met.

— The convergence from the second iteration to the third iteration is slow. This is because near the minimum the gradient is nearly zero. If high accuracy is needed, other local search methods should be used.

- From the basic calculus, we know that first partial derivatives are equal to zero:

$$\begin{cases} \dfrac{\partial f(\boldsymbol{x})}{\partial x_1} = 20x_1 + 5x_2 = 0 \\ \dfrac{\partial f(\boldsymbol{x})}{\partial x_2} = 5x_1 + 20x_2 - 60 = 0 \end{cases}.$$

The minimum occurs exactly at $x^* = \begin{bmatrix} -0.8 \\ 3.2 \end{bmatrix}$.

We see that the steepest descent method gives almost the exact solution after only three iterations.

- If we can use the calculus, why not use the same method to get the minimum point of $f(\boldsymbol{x})$?
  - For complicated functions of multiple variables $f(x_1,...,x_d)$, $f(\alpha^{(t)})$ at any step t is still a single-variable function, and the optimization of such $f(\alpha^{(t)})$ is much simpler compared with the original multi-variable problem.

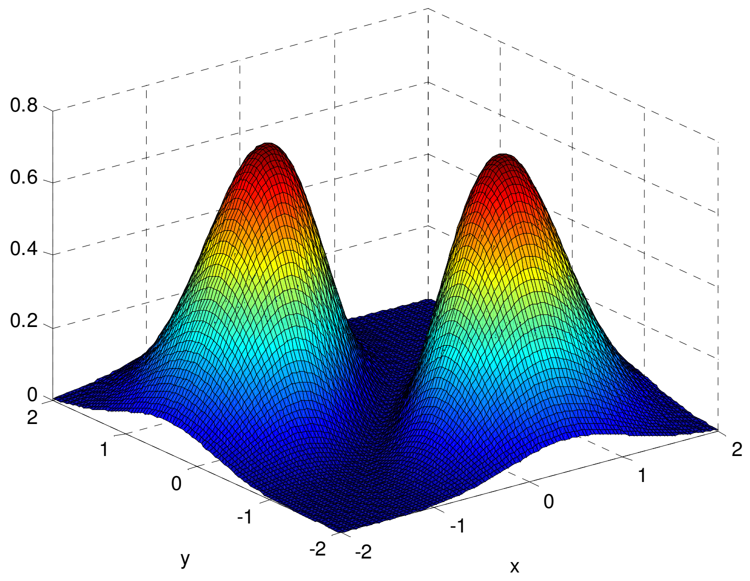
Hill Climbing with Random Restart

- *Multimodal function*:

  A function with multiple peaks or valleys is a multimodal function, and its landscape is multimodal.
- The following function,

  $$f(x,y) = (x-y)^2 e^{(-x^2-y^2)},$$

  has two global maxima at $\left(\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}\right)$ and $\left(-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right)$ with $f_{\max} \approx 0.7357$.



  — If we use the gradient-based methods such as hill climbing, the final results may depend on the initial value $\boldsymbol{x}_0 = (x_0, y_0)$.

  — If we draw a biased sample as the starting point in one region, the other peak may never be reached.

- A common strategy to ensure that all peaks are reachable is to carry out the hill climbing with multiple random restarts. This leads to a so-called *hill climbing with random restart*.
- If a function has k peaks and the hill climbing with random restart is ran n ($n \gg k$) times:

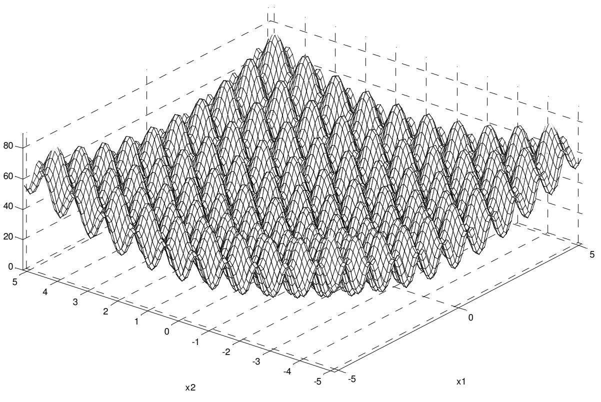
  — We may not know how many peaks and valleys a function has.

  — We can not ensure the samples are drawn from various search regions.
- In addition, many functions are discrete or non-differential. For example,

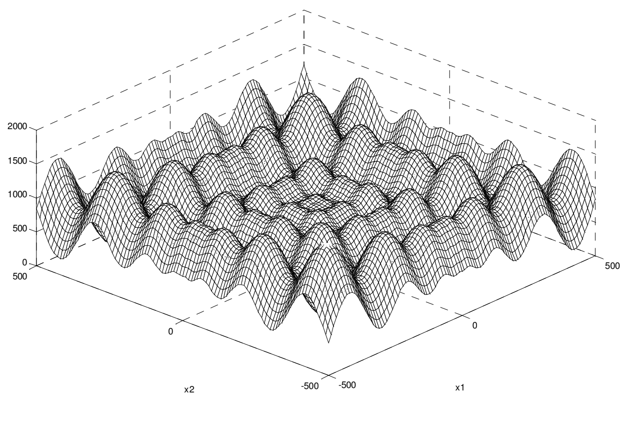
$$g(x,y) = (|x| + |y|)\,\mathrm{e}^{(-x^2-y^2)}$$

has a global minimum $f_{\min} = 0$ at (0,0) but the derivative at (0,0) does not exist (due to the absolute functions). In this case, all the gradient-based methods will not work.

- Quasi-Newton method works extremely well for smooth unimodal problems. However, if there is some discontinuity in the objective function, it does not work well.
- In reality, optimization problems even under various complex constraints are far more complicated, and the calculation of derivatives may be either impossible or too computationally expensive. Therefore, gradient-free methods are preferred.
- In fact, modern nature-inspire algorithms are almost all gradient-free optimization methods.
- $f_1(x) = 837.9658 - x_1 \sin\left(\sqrt{|x_1|}\right) - x_2 \sin\left(\sqrt{|x_2|}\right),\ (x_1, x_2) \in [-500, 500]$



$f_2(x) = 20 + x_1^2 + x_2^2 - 10\cos(2\pi x_1) - 10\cos(2\pi x_2),\ (x_1, x_2) \in [-5.12, 5.12]$



Search for Optimality

- Searching for the optimal solution is like treasure hunting.
  - We can do the treasure hunting alone. Simulated annealing (SA) is such a kind of search.
  - Alternatively, we can ask a group of people to do the hunting and share the information. Genetic algorithms (GA), Particle swarm optimization (PSO) the firefly algorithm (FA), and cuckoo search (CS) are *population-based* because they use lots of individuals.
  - Because some hunters are better than others, we can only keep the better hunters and recruit new ones.
- *Deterministic algorithm*:
  - For the same starting point, the deterministic algorithm will follow the same path whether you run the program today or tomorrow. In other words, the path and values are repeatable.
  - Most conventional algorithms are deterministic.
    - ✧ Newton's formula is a deterministic algorithm.
- *Stochastic algorithm*:
  - Stochastic algorithm uses some pseudo-random numbers.
  - The paths of each individual are not exactly repeatable, but the final results may be no big difference.
    - ✧ Genetic algorithms are a good example.
- There is a third type of algorithm that is a mixture or hybrid of deterministic and stochastic algorithms.
  - Hill climbing with random restart is a good example.
  - Since there is a random component in this hybrid algorithm, we often classify it as a type of stochastic algorithm in the optimization literature.
- Some stochastic algorithms use certain tradeoffs of *exploration* and *exploitation*.
  - *Exploration* means to generate diverse solutions so as to explore the search space on a global scale. It provides a good way to move away from local search to search on a global scale.
  - *Exploitation* means to focus on the search in a local region by exploiting the information that a current good solution is found in this region.