Nature-Inspired Search and Optimisation
Advanced Aspects of Nature-Inspired Search and Optimisation (Ext)

Shan He

School for Computational Science
University of Birmingham

Module 06-28209, 28211, 27818 and 27819: Nature-Inspired Search and Optimisation
Advanced Aspects of Nature-Inspired Search and Optimisation (Ext)

Outline of Topics

Randomised algorithms

"**For many problems, a randomised algorithm is the simplest, the fastest or both.**" - Prabhakar Raghavan, Vice President of Engineering at Google.

Matching Nuts and Bolts

- The everyday problem: Given one bolt and a collection of n nuts of different sizes, find a nut match the bolt
- Algorithmic form: essentially an unsorted array search problem, i.e., given an array of n elements (nuts), find the first element of which the value equals to x (bolt)
- How to solve it using algorithms? How to solve it efficiently?

Categories of Algorithms by by design paradigm

- Divide and conquer algorithms, e.g., **quicksort algorithm**
- Dynamic programming algorithms
- Mathematical programming algorithms, e.g., linear programming
- Search and enumeration algorithms
  - Brute force algorithms, enumerating all possible candidate solutions and check
  - Improved brute force algorithms, e.g., branch and bound algorithms
  - **Heuristic algorithms**
    - Local search, e.g., **greedy search**
    - **Randomised algorithms**, which include Evolutionary Computation, etc.

Heuristic algorithms

- Definition of **heuristic**: **Experience-based** techniques for problem solving, learning, and discovery to find a good enough solution but **not guaranteed to be optimal**"
  - Heuristic: ""find" or 'discover"
  - Experience-based: experimental, less theoretical and systematic
  - (Usually) Non-optimal but satisfactory
  - Faster: alternative to brute force (exhaustive) search
  - The most fundamental heuristic: trial and error

Heuristic algorithms

- CS Definition of **heuristic**: a (usually simple) algorithm produce a **good enough** solution for a problem in a **reasonable time frame**
- Trade of optimality, completeness, accuracy, or precision for speed.
- Usually to solve problems that are difficult for other algorithms, e.g., brute force algorithms
- The only viable option for NP-Hard (Non-deterministic Polynomial-time hard) problems
- One crucial ingredient: randomness

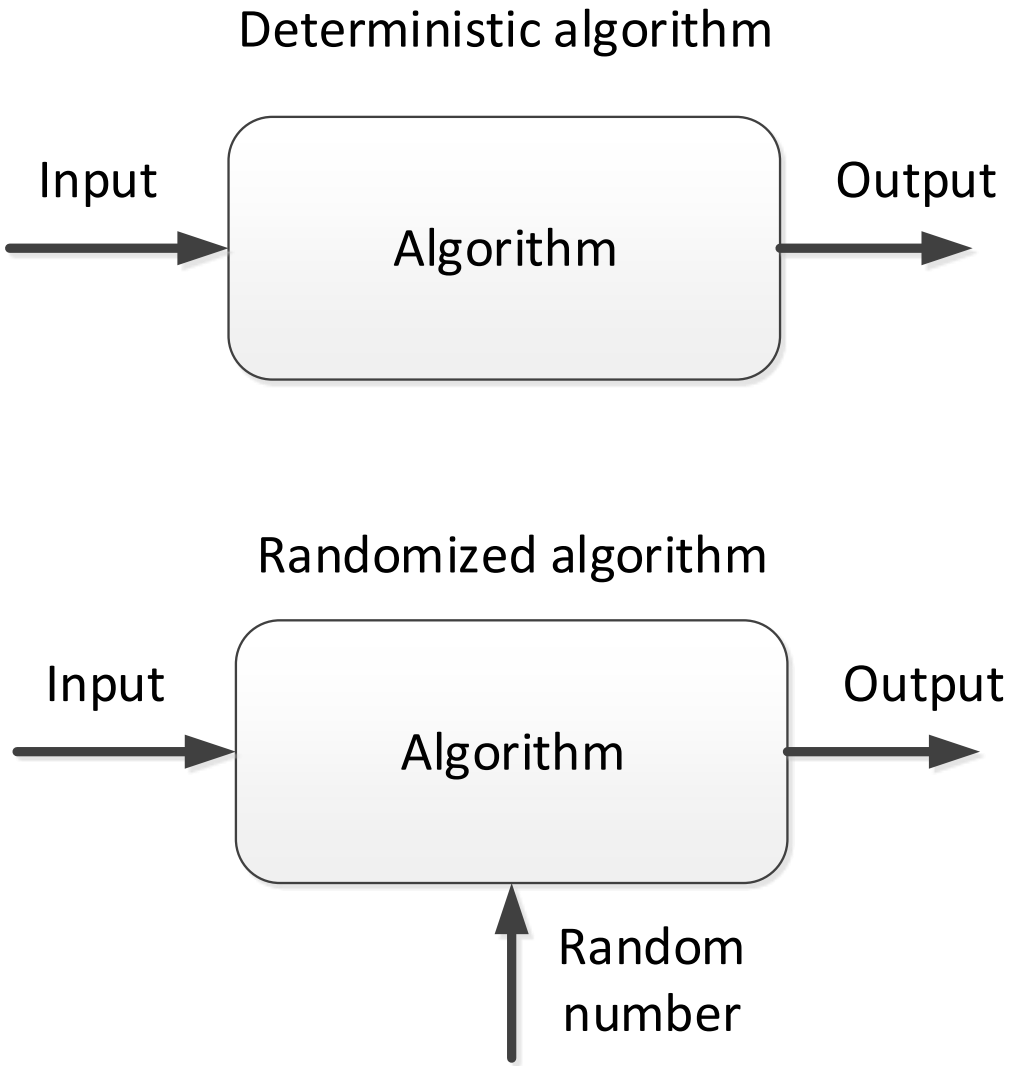
Determistic algorithms

- **Definition**: an algorithm that will always produce the same output given a particular input, can be seen as:
  - **A state machine**: at a particular instant in time, there exists a state describes what the machine is doing
    - State machines pass in a discrete manner from one state to another
    - When the machine receive input, it is in its initial (start) state
    - From this point onwards, its current state determines what its next state will be
    - Its course through the set of states is predetermined.
- **Goal**: prove that the algorithm solves the problem always correctly and quickly
- **Difficulties**: form some problems it is too slow or the exact solutions cannot be found in reasonable time

Randomised algorithms

- **Definition**: An algorithm that **makes random choices** during execution to produce a result
  - Takes a source of random numbers to make random choices
  - Behaviour can vary even on a fixed input
- **Goal**: design an algorithm and analyse it to show that its behaviour is likely to be good, on every input

Randomised algorithms vs deterministic algorithms



Randomised algorithms

- Two categories of randomised algorithms:
  - Using random numbers to **find** a solution to a problem
  - Using random numbers to **improve** a solution to a problem
- For the first category, two representative simple algorithms:
  - Las Vegas algorithm
  - Monte Carlo algorithm

Randomise algorithms: a movtivating example

- **Motivating example:** given one bolt and a collection of n nuts of various sizes, find a nut match the bolt
- **The problem:** given an unsorted array of n elements, find the first element of which the value equals to x
- Solution 1:

Las Vegas algorithm:

begin

repeat

Randomly select one element a out of n elements.

until $a == x$

end

Solving the movtivating example using randomise algorithms

- **Motivating example:** Given one bolt and a collection of n nuts of various sizes, find a nut match the bolt
- **The problem:** given an unsorted array of n elements, find the first element of which the value equals to x
- Solution 2:

Monte Carlo algorithm:

begin

$i := 0$

repeat

Randomly select one element a out of n elements.

$i := i + 1$

until $(a == x) \,||\, i = k$

end

Monte Carlo algorithm vs Las Vegas algorithm

- Las Vegas algorithm: a randomised algorithm that always gives correct results, the only variation from one run to another is the running time
- Monte Carlo algorithm: a randomised algorithm whose running time is deterministic, but whose results may be incorrect with a certain (typically small) probability.
- Differences:
  - Monte Carlo algorithm runs for a fixed number of steps
  - Las Vegas algorithm runs in an infinite loop until the correct results are found
  - Las Vegas algorithm can be converted into Monte Carlo algorithm using early termination

Why randomised algorithm?

- For the unsorted array search problem:
  - For deterministic algorithm,
    - Average run time: $O(n)$ (n is the size of the array)
    - The worst run time complexity is $O(n)$
  - For Monte Carlo algorithm, both the average and worst run time complexity are fixed: $O(1)$, i.e., the run time complexity is constant.

Quicksort algorithm

- The sorting problem: given an array A of n numbers, sort the numbers in ascending order

Quicksort algorithm:

```
less, equal, greater := three empty arrays
if length(array) > 1
        pivot := select an element of array
        for each x in array
                if x < pivot then add x to less
                if x = pivot then add x to equal
                if x > pivot then add x to greater
quicksort(less)
quicksort(greater)
array := concatenate(less, equal, greater)
```

The problem with quicksort algorithm

- Deterministically select an element as pivot, usually the last element
- Deterministic quicksort algorithm time complexity: $O(n \log n)$ on average for a random permutation array
- The worst case: $O(n^2)$, e.g., for a sorted array

Randomised quicksort algorithm

- The sorting problem: given an array A of n numbers, sort the numbers in ascending order

Randomised quicksort algorithm:

```
less, equal, greater := three empty arrays
if length(array) > 1
        pivot := randomly select an element of array
        for each x in array
                if x < pivot then add x to less
                if x = pivot then add x to equal
                if x > pivot then add x to greater
quicksort(less)
quicksort(greater)
array := concatenate(less, equal, greater)
```

Randomised quicksort algorithm

- Randomised quicksort algorithm: random partitioning, i.e., selecting a pivot randomly
  - Average run time complexity: $O(n \log n)$
  - The worst case: $O(2n \log n)$
  - No too bad compared with the theoretical information lower bound $O(n \log_2 n)$
- An example of the second category of randomised algorithms: using random numbers to improve a solution to a problem

Applications of randomised algorithms

- Mathematics:
  - Number theory, e.g., primality test
  - Computational Geometry: graph algorithms, e.g., minimum cut
  - Linear algebra: matrix computations
- Computer Science:
  - Data analysis: PageRank
  - Parallel computing: Deadlock avoidance
  - Optimisation: evolutionary computation
- Computational biology: DNA read alignment

Advantages/disavantages of randomised algorithms

- Advantages:
  - Simplicity: usually very easy to implement
  - Performance: for many problems, they usually produce (near-) optimum solutions with high probability
- Disadvantages:
  - Getting incorrect answer with a finite probability (but for some problems, the probability is very large).
    - Solution: Repeat the algorithm
  - Difficult to analyse the running time and probability of getting an incorrect solution
  - Getting truly random numbers is impossible

Conclusion

- Randomness is a resource to improve simplicity and efficiency of algorithms.
- In fact, randomness is also the important 'resource' in other fields, such as biology, physics, psychology and even theology.
- Randomised algorithms, e.g., random projection are finding wider applications in the era of big data
- However, accessing to truly unbiased and independent sequence of random bits is expensive.
- We expect quantum computer will change all these

Further readings

- Motwani, Rajeev; Raghavan, Prabhakar (1995). Randomized Algorithms. New York: Cambridge University Press. ISBN 0-521-47465-5.
- Richard M. Karp (1991), An introduction to randomized algorithms, 34, Discrete Mathematics.
- Randomized algorithms for matrices and data

Assignment: Test equality of long strings

- The problem: suppose Dan Brown is having holiday on Mars, he discovers a digital manuscript (a long string) of The Da Vinci Code in his laptop. He wants to compare this manuscript with his final version in his PC (also a long string) at home on Earth to see whether they are the same. He can access his manuscript in his PC at home, but the communication is very expensive. How should he do to compare these two versions via this expensive communication channel.
- Hint: Read "An introduction to randomized algorithms" 175-176, implement Algorithm 5.2