Types of Algorithms

Sadoon Azizi

s.azizi@uok.ac.ir

Department of Computer Engineering and IT

Spring 2019

Introduction

- In this lecture we will discuss different ways to categorize classes of algorithms.
- There is no one “correct” classification.
- One should regard the task of categorizing algorithms more as giving them certain attributes.

Deterministic vs. Randomized

- **Deterministic** algorithms produce on a given input the same results following the same computation steps.

- **Randomized** algorithms throw coins during execution.
  - Hence either the order of execution or the result of the algorithm might be different for each run on the same input.

- What are randomized algorithms good for?
  - Randomized algorithms usually have the effect of **perturbing the input**. Or put it differently, the input looks random, which makes bad cases **very seldom**.
  - Randomized algorithms are often conceptually **very easy to implement**. At the same time they are in **run time** often superior to their deterministic counterparts.

Offline vs. Online

- **Offline** algorithms know their input beforehand.
- Whereas, **Online** algorithms do not know their input at the beginning. It is given to them online.
  - Online algorithms are usually analyzed by using the concept of **competitiveness**, that is the worst case factor they take longer compared to the best algorithm with complete information.

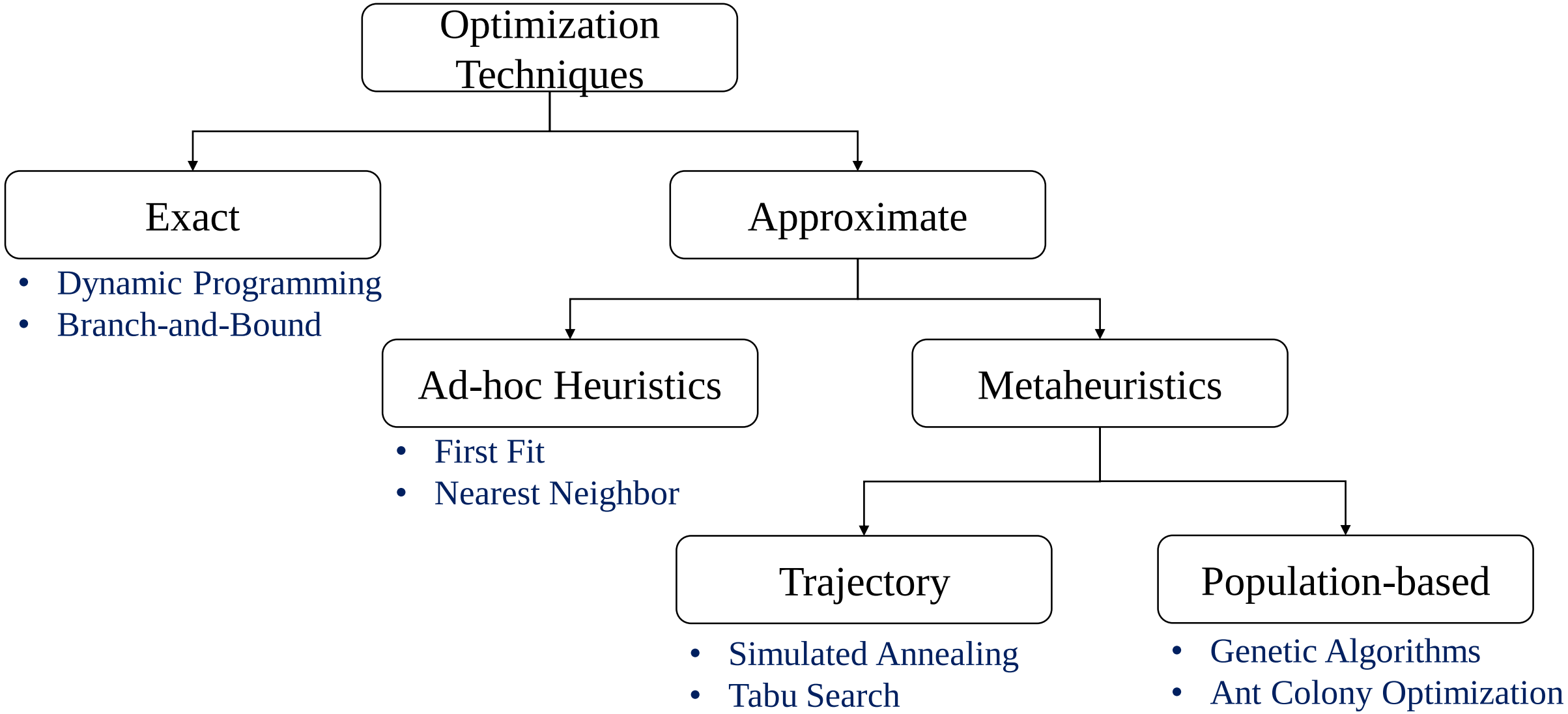
Exact vs Approximate vs. Heuristic vs. Metaheuristic

- **Exact** algorithms aim at computing the optimal solution given a goal.
  - Often this is quite expensive in terms of run time or memory and hence not possible for large input.
- **Approximation** algorithms aim at computing a solution which is for example only a certain, guaranteed factor worse than the optimal solution, that means an algorithm yields a ***c-approximation***, if it can guarantee that its solution is never worse than a factor c compared to the optimal solution.
- **Heuristic** algorithms are specific algorithms for specific problems
- **Metaheuristics** are not problem-specific. Metaheuristics are higher level strategies that *guide* the search process. The goal is to efficiently explore the search space in order to find (*quasi*-) optimal solutions.

Heuristic vs. Metaheuristic

- In general, special-purpose heuristic algorithms are more effective than general-purpose meta-heuristics.
  - In fact, designing one such method that actually produces high quality solutions is a nontrivial task, since it mainly depends on the problem, and requires thorough understanding of it.
- On the other hand, meta-heuristics are more easily applicable to a wide variety of different problems.
  - general ideas for almost any problem

Exact vs Approximate vs. Heuristic vs. Metaheuristic



Categorization according to main concept

- Backtracking algorithms
- Divide-and-conquer algorithms
- Dynamic programming algorithms
- Greedy algorithms
- Branch-and-bound algorithms
- Brute force algorithms
- and others....

Backtracking algorithms

- A backtracking algorithm is based on a depth-first recursive search.
- It tests to see if a solution has been found, and if so, returns it; otherwise
- For each choice that can be made at this point,
  - Make that choice
  - Recur
  - If the recursion returns a solution, return it
- If no choices remain, return failure
- **Example:** Graph Coloring

Divide-and-conquer algorithms

A divide-and-conquer algorithm consists of two parts.

- Divide the problem into smaller subproblems of the same type and solve these subproblems recursively
- Combine the solutions to the subproblems into a solution to the original problem
- **Example:** Merge Sort

Dynamic programming algorithms

- A dynamic programming algorithm remembers past results and uses them to find new results.
- Dynamic programming is generally used for optimization problems in which:
  - Multiple solutions exist, need to find the best one
  - Requires optimal substructure and overlapping subproblem
  - Optimal substructure: Optimal solution contains optimal solutions to subproblems
  - Overlapping subproblems: Solutions to subproblems can be stored and reused in a bottom-up fashion
- **Example:** Longest Common Sequence (LCS)

Greedy algorithms

- A greedy algorithm sometimes works well for optimization problems.
- A greedy algorithm works in phases. At each phase:
  - You take the best you can get right now, without regard for future consequences
  - You hope that by choosing a local optimum at each step, you will end up at a global optimum
- This strategy actually often works quite well and for some class of problems it always yields an optimal solution.
- **Example:** Activity Selection Problem

Branch-and-bound algorithms

- Branch-and-bound algorithms are generally used for optimization problems.
- As the algorithm progresses, a tree of subproblems is formed.
- A method is used to construct an upper and lower bound for a given problem.
- At each node, apply the bounding methods.
  - If the bounds match, it is deemed a feasible solution to that particular subproblem.
  - If bounds do not match, partition the problem represented by that node, and make the two subproblems into children nodes.
- Continue, using the best known feasible solution to trim sections of the tree, until all nodes have been solved or trimmed.
- **Example:** Travelling salesman problem (TSP)

Brute force algorithms

- A brute force algorithm simply tries all possibilities until a satisfactory solution is found.
- Such an algorithm can be:
- **Optimizing:** Find the best solution. This may require finding all solutions, or if a value for the best solution is known, it may stop when any best solution is found (**Example:** Finding the best path for a travelling salesman)
- **Satisficing:** Stop as soon as a solution is found that is good enough (**Example:** Finding a travelling salesman path that is within 10% of optimal)