CS C364
Design & Analysis of Algorithms

Algorithms – Design Techniques

Intractable Problems

- Dealing with hard problems

- Approaches and Tradeoffs

INTRACTABLE PROBLEMS

- Problems that *cannot be solved in polynomial time* are referred to as ***intractable*** problems.
  - NP-Completeness is used as the baseline for "**intractable**" problems - i.e.
    - If a problem can be proven to *be NP-Complete* then it is treated as "hard evidence" that the problem is *intractable*.
- NP-Complete problems have been found in numerous real-life application domains
    - What could be done (to solve these problems)?

INTRACTABLE PROBLEMS – BROAD APPROACHES

1. Accept exponential complexity for solutions
2. Find algorithms for special case inputs that may be more common in real-life
3. Accept “inexact” solutions with some guarantees
4. Accept “imperfect” solutions with no guarantees, only expecations.

Accepting exponential solutions

- Accept exact algorithms of exponential time complexity:
- Implication:
  - Can handle only small inputs:
    - A modern computer can execute (approx.) 2^{30} instructions per second
      - i.e. an $O(2^n)$ algorithm can be executed in several days for an input of size 50
- Consider:
  - E.g. Graph Coloring is NP-complete:
    - Given a graph, can its vertices be assigned one of k colors each such that *no two adjacent vertices share a color*?

ACCEPTING EXPONENTIAL SOLUTIONS [2]

- Register Allocation for a program can be modeled as Graph Coloring
  - Each variable is a vertex and
  - an edge captures the relation that
    - *there is an overlap in the **live intervals** of the two variables.*
- Practice:
  - Compilers may use exact Graph Coloring solutions to solve register allocation.
    - But they *perform register allocation for one procedure at a time* and
    - in practice, *the number of variables in a procedure is small*

ALGORITHMS FOR SPECIAL CASE INPUTS

- Find algorithms for special case inputs that may be more common in real-life.
  - For example consider the Graph Coloring problem again
    - *Graph Coloring is $\mathcal{NP}$-complete* and
    - *Graph Coloring for Planar graphs is $\mathcal{NP}$-complete*
  - But there exist polynomial time algorithms for special case inputs.

ALGORITHMS FOR SPECIAL CASE INPUTS – GRAPH COLORING

- There exist polynomial time algorithms for special case inputs.
  - a quadratic algorithm for 4-coloring planar graphs (*see* RSST97)
  - and a linear algorithm for 5-coloring planar graphs (*see* MKB)
- References:

[RSST97]: Robertson, Sanders, Seymour and Thomas. *Efficiently four-coloring planar graphs*.

[MKB]: Mott, Kandel, and Baker. [The text book for the **Discrete Structures** course]

ALGORITHMS FOR SPECIAL CASE INPUTS – GRAPH COLORING [2]

- Consider the decision version of the Coloring problem for planar graph:
  - Can a Planar Graph be colored with k colors?
- This problem exhibits the following interesting structure:
  - The solution for k>=4 is trivial
  - The solution for k=2 is simple
    - Bipartite graphs (and only bipartite graphs) are 2-colorable.
  - There is no known polynomial time solution for k=3:
    - *in fact 3-coloring of planar graphs is an $\mathcal{NP}$-complete problem*

INEXACT SOLUTIONS WITH GUARANTEES

Accept algorithms that generate ***inexact solutions****:*

1. **Probabilistic algorithms** (for decision problems)

2. **Approximation algorithms** (for optimization problems)

3. **Combine both** of the above approaches (for optimization problems)

INEXACT SOLUTIONS - PROBABILISTIC ALGORITHMS

Probabilistic solutions (for decision problems)

Monte-Carlo algorithms that run in polynomial time but may generate false positives, false negatives, or both with low probability

[Note:

We have seen Monte Carlo algorithms for reducing time complexity of problems that are in $\mathbb{P}$

End of Note.]

INEXACT SOLUTIONS – APPROXIMATION ALGORITHMS

Approximation Algorithms (for optimization problems)

Algorithms that run in polynomial time but produce sub-optimal solutions with an approximation guarantee

i.e. that a solution produced by such an algorithm is guaranteed to be

no worse than the optimal solution by an ***approximation factor***

INEXACT SOLUTIONS – PROBABILISTIC APPROXIMATION

Combine the two approaches - *of Monte Carlo techniques and approximation* -

Monte-Carlo algorithms that run in polynomial time

but with a <u>high probability</u> produce solutions with an <u>approximation guarantee</u>

INEXACT SOLUTIONS WITH NO GUARANTEES

Accept "inexact" solutions without any guarantees:

Heuristic solutions that *are likely to run in polynomial time for most input instances* and *are likely to provide "good" solutions*

i.e. good expectations but no guarantees

Definition of a good solution is determined by experimentation or by practice.

Worst case scenarios are ignored