TDDD95 Algorithmic Problem Solving

Le 4 – Problem Solving Paradigms

Fredrik Heintz

with some slides from Fredrik Präntare

Dept of Computer and Information Science

Linköping University

Overview

- Problem solving paradigms:
  - Simulation / Ad-hoc
  - Greedy and Dynamic Programming (again!)
  - Divide and Conquer
  - Complete Search and Branch-and-Bound
  - Probabilistic Search

Simulation/Ad hoc

- Simulation/Ad hoc.
  - Do what is stated in the problem.
  - Example: Simulate a robot.
  - Many such problems last week:
    - The SetStack Computer
    - Introspective Caching
    - Chopping Wood
      (reverse simulation to some degree)

Complete Search

- When a problem is small or (almost) all possibilities must be tried, *complete search* is a candidate approach.
- To determine the feasibility of complete search: estimate the number of calculations that must be made (usually interested in the worst case).

Complete Search

- *Iterative complete search* uses nested loops to *generate* every possible complete solution and *filter* out the valid ones.
  - Iterating over all permutations using std::next_permutation.
  - Iterating over all subsets using bit set technique.
- *Recursive complete search* extends a partial solution with one element until a complete and valid solution is found.
  - This approach is often called *recursive backtracking*.
  - *Pruning* can sometimes be used to significantly improve the efficiency by removing partial solutions that can not lead to a solution. In the best case, only valid solutions are generated.

Branch-and-Bound

- If you can find a search space representation that we can partition, it might be possible to use **branch-and-bound** to reduce the number of solutions you have to test/investigate/look at.
  - Calculate bounds (lower & upper) for regions/branches of the search space, and continuously remove regions/branches (subspaces) that cannot contain the solution that you are looking for (e.g. the optimal).
  - Extensively used in optimization: CPLEX, Gurobi, …
  - Specific problems may make it possible to use more advanced pruning techniques, or approximative solutions (if you are not looking for the optimal solution), for example by taking advantage of expert knowledge (e.g. in chess).

Greedy

- An algorithm is said to be greedy if it makes a locally optimal choice in each step towards the globally optimal solution.
- For a greedy algorithm to give a globally optimal result, a problem must have two properties:
  - It has optimal sub-structures, i.e. an optimal solution contains the optimal solutions to sub problems.
  - It has the greedy choice property, i.e. if we extend a partial solution by making a locally optimal choice, we will get the optimal complete solution without reconsidering previous choices.

Greedy

- Classical examples: Coin change in some currencies, interval coverage, approximation algorithms (e.g. for good-enough suboptimal solutions).
- Greedy algorithms can be useful as heuristics.
- Can be very useful to calculate bounds for branch-and-bound techniques.

Divide and Conquer

- Divide and conquer is very common and powerful technique which divides a problem into smaller parts, solves each part recursively and then puts together the answer from the pieces.
- Many well-known algorithms are based on divide and conquer such as quick sort, merge sort and binary search.

Dynamic Programming

- Dynamic Programming is a problem-solving approach which computes the answer for every possible *state* exactly once.

Dynamic Programming

- **Top-down (memorization):** no need to consider the order of computations, only compute states actually used, natural transition from complete search,
- **Bottom-up (tabulation):** no recursion, computes every state, table size can be reduced if only the previous row of states is used then only two rows are required.

Dynamic Programming

- Displaying the optimal solution:
  Store the previous state for each solution.
- Often used to solve NP-hard problems.

Monte Carlo / Stochastic / Random Search

- Sometimes it is possible to use probabilistic techniques for intractable/difficult problems.
- A high number of random samples from the search space can sometimes make it possible (e.g. by being combined) to guarantee a high-enough probability for a solution to be correct (or good enough).
  - Used extensively in ray tracing/rendering (Hyperion), physics simulations, scheduling tasks, and game-playing systems (MCTS & AlphaGo).
- ***Typically*** not applicable to this course, but there are some exercises that can be solved with AC using this type of technique.

Summary

- Problem solving paradigms:
  - Simulation / Ad-hoc
  - Complete Search and Branch-and-Bound
  - Greedy Search
  - Divide and Conquer
  - Dynamic Programming
  - Probabilistic Search