Divide and Conquer methods (and Branch and Bound) in a nutshell

UCLA NanoCAD

August 3, 2011

Introduction

Divide and conquer works by:

1. Breaking it into subproblems, that are themselves smaller instances of the same type of problem
2. Recursively solving these subproblems
3. Appropriately combining their answers

(from Dasgupta, Papadimitriou and Vazirani)

Analyzing Divide and Conquer Methods

- Analyzing Divide and Conquer methods relies on analyzing recurrence relations, e.g.

$$T(n) = 3T(n/2) + O(n) \tag{1}$$

- The resulting complexity can be described using the following theorem:

Theorem

If $T(n) = aT(\lceil n/b \rceil) + O(n^d)$ *for some constants* $a > 0$, $b > 1$, *and* $d \geq 0$, *then*

$$T(n) = \begin{cases} O(n^d) & \text{if } d > \log_b a \\ O(n^d \log n) & \text{if } d = \log_b a \\ O(n^{\log_b a}) & \text{if } d < \log_b a \end{cases} \tag{2}$$

Merge Sort

- Merge sort works by
  1. Diving the list in two
  2. Calling merge sort on each half
  3. Merging the sorted lists
- This has the recursion relation:

$$T(n) = 2T(\lceil n/2 \rceil) + O(n) \tag{3}$$

  because there are two calls to merge-sort ,$2T(\lceil n/2 \rceil)$, and the merge process takes $O(n)$
- From the Theorem, this gives a complexity of $O(n \log(n)$ which is very good.

Matrix Multiplication

- Matrix multiplication was generally believed to be $O(n^3)$! For $Z = XY$

$$Z_{ik} = \sum_j X_{ij} Y_{jk} \tag{4}$$

- Strassen discovered a more efficient divide and conquer approach. Instead of:

$$XY = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix} \tag{5}$$

- Which has recurrence relation $T(n) = 8T(n/2) + O(n)^2$ and complexity $O(n^3)$

Matrix Multiplication, cont'd

- The multiplication was expressed as:

$$XY = \begin{bmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{bmatrix} \tag{6}$$

where:

$$\begin{array}{lll} P_1 = A(F-H) & P_2 = (A+B)H & P_3 = (C+D)E \\ P_4 = D(G-E) & P_5 = (A+D)(E+H) & P_6 = (B-D)(G+H) \\ & & P_7 = (A-C)(E+F) \end{array} \tag{7}$$

- The complexity is $T(n) = 7T(n/2) + O(n^2)$, which is

$$O(n^{\log_2 7}) \approx O(n^{2.81}) \tag{8}$$

Other examples

- Fast fourier transform
- Nearest Neighbor search

Branch and Bound

- Method to solve integer programming problems
- Simple example:

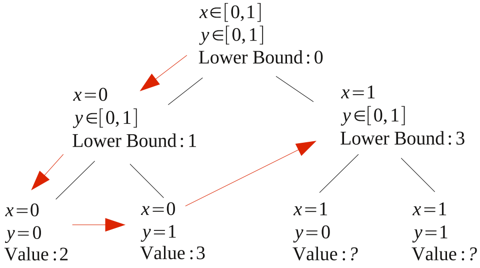
$$\begin{aligned} \text{minimize} \quad & f(x,y) \\ \text{subject to} \quad & g(x,y) = 0 \\ & x \in \{0,1\} \\ & y \in \{0,1\} \end{aligned} \tag{9}$$

- Note that the optimal solution to

$$\begin{aligned} \text{minimize} \quad & f(x,y) \\ \text{subject to} \quad & g(x,y) = 0 \\ & x \in [0,1] \\ & y \in \{0,1\} \end{aligned} \tag{10}$$

is a lower bound on the original (x has more choices).

Branch and Bound, cont'd



- The lower bounds can be compared with a given feasible solution.
- This can reduce the number of solutions that need to be checked.