Dynamic Programming Speedups

Dynamic Programming is a classic bottom-up optimization technique. It usually requires filling in a table $T[i]$ indexed by some $i \in \mathcal{I}$. The running time of the algorithm will be $\sum_{i \in \mathcal{I}} w(i)$, the total amount of time required to fill in all of the table entries; $w(i)$ is the work (time) needed to calculate the value of $T[i]$.

A **Dynamic Programming Speedup** is a way of improving the run-time of the DP algorithm by noting that it is possible to fill in the table entries quicker than expected by taking advantage of dependencies between the different table entries.

The two examples we will see are

(i) The *Quadrangle-Inequality* speedup, illustrated by constructing Optimal Binary Search Trees and

(ii) The *Monotone-Matrix* speedup, illustrated by the placement of web proxies on a line.

The Optimal Binary Search Tree Recurrence:

For $0 \le i \le j \le n$, we are given constants $w(i,j)$ and define table $c[\cdot,\cdot]$ by $c[i,i] = 0$ and

$$c[i,j] = w(i,j) + \min_{i<k\le j} \left(c[i,k-1] + c[k,j] \right).$$

Note that filling in the table seems to require $\Theta(n^3)$ time. We will see that if $w(i,j)$ satisfies the quadrangle-inequality then the table can be filled-in using only $\Theta(n^2)$ time.

Totally Monotone Matrices:

For $0 \le i \le j \le n$, we are (implicitly) given $a(i,j)$ and also $b(i)$. Define table $E[\cdot]$ by

$$E[j] = \min_{1\le i\le j} \left(b(i) + a(i,j) \right)$$

Note that filling in the table seems to require $\Theta(n^2)$ time. We will see that if $b(i)+a(i,j)$ defines a Totally Monotone Matrix then the table can be filled-in using only $\Theta(n)$ time.

Selected References

1. Donald E. Knuth, "Optimum Binary Search Trees," *Acta Informatica* 1, pp. 14-25 (1971). (QI)

2. F. F. Yao, "Efficient Dynamic Programming Using Quadrangle Inequalities," *Proceedings of the 12 Annual ACM Symposium on Theory of Computing (STOC'80),* pp. 429-43, (1980). (QI)

3. A.Aggarwal, M.M.Klawe, S.Moran, P.Shor, R.Wilber. "Geometric applications of a matrix-searching algorithm.," *Algorithmica* (2) pp. 195-208 (1987). (MM)

4. G. Woeginger, "Monge strikes again: optimal placement of web proxies in the internet," *Operations Research Letters,* 27(3), pp. 93-96 (2000). (MM)

5. Amotz Bar-Noy and Richard E. Ladner, "Efficient Algorithms for Optimal Stream Merging for Media-on-Demand, *SIAM Journal on Computing,* 33(5), pp. 1011-1034 (2004). (QI)

Optimal Binary Search Trees: The Problem

We are given $2n+1$ probabilities, $p_1, \ldots, p_n$ and $q_0, \ldots, q_n$; p_i is the probability that a search is for key_i; such a search is called successful. q_i is probability that the search argument is unsuccessful and is for an argument between key_i and key_{i+1} (where we set $key_0 = -\infty$ and $key_{n+1} = \infty$.

Our problem is to find an *optimal binary search tree (BST)* with n internal nodes corresponding to successful searches and $n+1$ leaves corresponding to unsuccssful searches that minimizes the average search time. Let $d(p_i)$ be the depth of internal node corresponding to p_i and $d(q_i)$ the depth of leaf corresponding to q_i. Then we want to find a tree that minimizes

$$\sum_{1 \le j \le n} p_j(1 + d(p_j)) + \sum_{0 \le k \le n} q_k\, d(q_k).$$

Given $p_1, \ldots, p_n$ and $q_0, \ldots, q_n$ our problem is to find a BST that minimizes.

$$\sum_{1 \le j \le n} p_j (1 + d(p_j)) + \sum_{0 \le k \le n} q_k \, d(q_k).$$

Let $c[i, j]$ be the minimum cost subtree for the weights $p_{i+1}, \ldots, p_j$ and $q_i, \ldots, q_j$. Our problem is to calculate $c[0, n]$ (and associated BST). Since both left and right subtrees of a min-cost tree are also min-cost (optimal) we find that we need to solve:

$$\boxed{\begin{array}{rcl} c[i,i] & = & 0 \quad \text{and,} \quad \text{for } 0 \le i < j \le n, \\ c[i,j] & = & w(i,j) + \min\limits_{i<k\le j} (c[i,k-1] + c[k,j]) \end{array}}$$

where

$$w(i, j) = p_{i+1} + \cdots p_j + q_i + \cdots q_j.$$

Let $w(i,j) = p_{i+1} + \cdots p_j + q_i + \cdots q_j$.
Our dynamic programming problem is to find $c[0,n]$ where the DP table is

$$\boxed{\begin{array}{rcl} c[i,i] & = & 0 \quad \text{and,} \quad \text{for } 0 \le i < j \le n, \\ c[i,j] & = & w(i,j) + \min\limits_{i<k\le j} \left(c[i,k-1] + c[k,j]\right) \end{array}}$$

We will assume that we can calculate $w(i,j)$ in $O(1)$ time (this can be done using $O(n)$ preprocessing time and $O(n)$ space. How?).

Straightforwardly filling in $c[i,j]$ requires $\Theta(j-i)$ time, leading to an $\sum_{i,j}\Theta(j-i) = \Theta(n^3)$ time algorithm.

We will now see how to fill in the DP table using only $\Theta(n^2)$ time.

Definition: $w(i,j)$ satisfies the quadrangle inequality (QI) if

$$\forall i \le i' \le j \le j', \quad w(i,j)+w(i',j') \le w(i',j)+w(i,j')$$

Definition: $w(i,j)$ is monotone on the lattice of intervals (MLI) (ordered by inclusion) when

$$\forall [i,j] \subseteq [i',j'], \quad w(i,j) \le w(i',j')$$

In our problem

$$w(i,j) = p_{i+1} + \cdots p_j + q_i + \cdots q_j.$$

It is obvious that this $w(i,j)$ is MLI. To see that it satisfies the QI note $w(i,j) = w(0,j) - w(0,i-1)$. So,

$$\begin{aligned} w(i,j) + w(i',j') &= (w(0,j) - w(0,i-1)) + \big(w(0,j') - w(0,i'-1)\big) \\ &= \big(w(0,j) - w(0,i'-1)\big) + \big(w(0,j') - w(0,i-1)\big) \\ &= w(i',j) + w(i,j'). \end{aligned}$$

$$\begin{array}{rcl} c[i,i] & = & 0 \quad \text{and,} \quad \text{for } 0 \le i < j \le n \\ c[i,j] & = & w(i,j) + \min_{i<k\le j} \left(c[i,k-1] + c[k,j] \right) \end{array}$$

Speedup Theorem: (F.F. Yao) If $w(i,j)$ satisfies the QI and is MLI then the DP table above can be filled in using only $\Theta(n^2)$ time.

This was proved in two steps. The first was

Lemma 1: If $w(i,j)$ satisfies the QI and is MLI then $c[i,j]$ also satisfies the QI.

This lemma implies that $c[i,j]$ satisfies the QI.
The second step was

Lemma 2: Let $c_k(i,j) = w(i,j)+c[i,k-1]+c[k,j]$. Let $K_c(i,j) = \max\{k : c_k(i,j) = c(i,j)\}$ be the largest index k at which minimum occurs in the DP (we set $K_c(i,i) = i$). Then, if $c[i,j]$ satisfies the QI,

$$K_c(i,j) \le K_c(i,j+1) \le K_c(i+1,j+1).$$

$$
\boxed{\begin{aligned} c[i,i] &= 0 \quad \text{and,} \quad \text{for } 0 \le i < j \le n \\ c[i,j] &= w(i,j) + \min_{i<k\le j} \left(c[i,k-1] + c[k,j]\right) \end{aligned}}
$$

Assume that conditions of Lemma 2 hold so that

$$K_c(i,j) \le K_c(i,j+1) \le K_c(i+1,j+1).$$

If we had already calculated $c[i,j], K_c(i,j)$ and $c[i+1,j+1], K_c(i+1,j+1)$, then we could calculate $c[i,j]$ in

$$1 + K_c(i+1,j+1) - K_c(i,j)$$

time.

If $j - i = t + 1$, we can calculate $c[i, i+t+1]$ in

$$1 + K_c(i+1, i+1+t) - K_c(i, i+t)$$

time.

$$\begin{array}{rcl} c[i,i] & = & 0 \quad \text{and,} \quad \text{for } 0 \le i < j \le n \\ c[i,j] & = & w(i,j) + \min\limits_{i<k\le j} \left(c[i,k-1] + c[k,j]\right) \end{array}$$

Let $t = j - i$, $t = 0, 1, \ldots n.$ We will fill in the DP table $c[i,j]$ in increasing order of t. Assume that we have already calculated all of the entries for $j - i \le t$. Then the **total** amount of time to fill in **all** of the $c[i,j]$ entries with $j - i = t + 1$ is

$$\begin{array}{rcl} \sum\limits_{i=0}^{n-t-1} (1 & + & K_c(i+1, i+1+t) - K_c(i,t)) \\ & \le & n - t + K_c(n-t, n) \\ & \le & 2n \end{array}$$

Thus, the total amount of time to fill in the DP table is $O(n \cdot 2n) = O(n^2)$.

We have just seen that Yao's Theorem follows from her two lemmas and we therefore only have to prove the two lemmas.

Also, in the **optimal binary search tree problem** the $w(i,j)$ satisfy Yao's conditions, so we can solve that problem in $O(n^2)$ time.

Lemma 1: If $w(i, j)$ satisfies the QI and is MLI then $c[i, j]$ also satisfies the QI.

Proof: This is a straightforward case by case analysis. See Yao's original paper for details.

Lemma 2: Let $c_k(i,j) = w(i,j)+c[i,k-1]+c[k,j]$. Let $K_c(i,j) = \max\{k : c_k(i,j) = c(i,j)$ be the largest index k at which minimum occurs in the DP (we set $K_c(i,i) = i$). Then, if $c[i,j]$ satisfies the QI,

$$K_c(i,j) \le K_c(i,j+1) \le K_c(i+1,j+1).$$

Proof: We will assume $i < j$ since the lemma is obviously true when $i = j$.

We will prove $K_c(i,j) \le K_c(i,j+1)$; to do this it suffices to prove that, if $i < k \le k' \le j$ then

$$c_{k'}[i,j] \le c_k[i,j] \quad \Rightarrow c_{k'}[i,j+1] \le c_k[i,j+!] \quad (1)$$

The QI of $c[i,j]$ says

$$c[k,j] + c[k',j+1] \le c[k',j] + c[k,j+1].$$

Addding $w(i,j)+w(i,j+1)+c[i,k-1]+c[i,k'-1]$ to both sides gives gives

$$c_k(i,j) + c_{k'}(i,j+1) \le c_{k'}(i,j) + c_k(i,j+1),$$

yielding (1) and therefore $K_c(i,j) \le K_c(i,j+1)$.

The proof of $K_c(i,j+1) \le K_c(i+1,j+1)$ is similar.

Review

We just saw Yao's proof that, if $w(i,j)$ satisfies the QI and is MLI then the DP table

$$\begin{array}{rcl} c[i,i] & = & 0 \quad \text{and,} \quad \text{for } 0 \le i < j \le n \\ c[i,j] & = & w(i,j) + \min\limits_{i<k\le j} \left(c[i,k-1] + c[k,j]\right) \end{array}$$

can be filled in using only $\Theta(n^2)$ time. This DP was originally formulated for the Optimum Binary Search Tree problem, with a $\Theta(n^3)$ solution, by Gilbert and Moore in 1959. The $\Theta(n^2)$ improvement was originally proven by Knuth in 1971 using a very problem specific analysis.

The importance of Yao's result (1980) is that she gave very simple conditions on $w(i,j)$ that, if satisfied, guarantee that the same speedup works. While the conditions might seem rather artificial, they do arise quite often in practice. See, e.g., the paper on Optimal Stream Merging by Bar-Noy and Ladner in the references.