CS580 Algorithm Design, Analysis, and Implementation

Lecture notes, Jan 27, 2022
Wufei Ma

1 Dynamic Programming

Weighted interval scheduling. The dynamic programming algorithms compute the optimal value. We can use post-processing to find the exact solution.

```
Function Find-Solution(j):
   if j == 0 then:
      return null
   else if v[j] + M[p(j)] > M[j-1]:
      print j
      Find-Solution(p(j))
   else:
      Find-Solution(j-1)
   endif
```

The number of recursive calls is less than or equal to n so the complexity is $O(n)$.

We can also implement the dynamic programming algorithm in a bottom-up manner by computing $\text{OPT}(\cdot)$ from 1 to n.

Least squares. Given n points in the plane, $(x_1, y_1), \dots, (x_n, y_n)$. Find a line $y = ax + b$ that minimizes the sum of the squared error.

We can solve this problem with calculus:

$$a = \frac{n \sum_i x_i y_i - (\sum_i x_i)(\sum_i y_i)}{n \sum_i x_i^2 - (\sum_i x_i)^2}, \quad b = \frac{\sum_i y_i - a \sum_i x_i}{n}$$

Segmented least squares. Points lie roughly on a sequence of several line segments. Given n points in the plane $(x_1, y_1), \dots, (x_n, y_n)$ with $x_1 < \cdots < x_n$, find a sequence of lines that minimizes the sum of the sums of the squared errors E in each segment and the number of lines L:

$$E + c \cdot L, \quad c > 0$$

Let $\text{OPT}(j)$ be the minimum cost for points $p_1, \dots, p_j$. Let $e(i, j)$ be the minimum sum of squares for points $p_1, \dots, p_j$. To compute $\text{OPT}(j)$, the last line segment uses points $p_i, \dots, p_j$ for some i and the cost is $e(i, j) + c + \text{OPT}(i - 1)$.

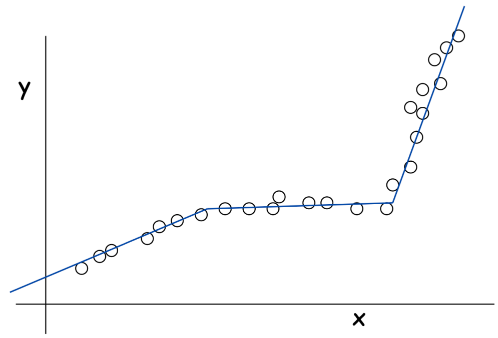

Figure 1: Segmented least squares.

```
Function Segmented-Least-Squares():
    M[0] = 0
    for j = 1 to n:
        for i = 1 to j:
            compute e(i, j)
    for j = 1 to n:
        M[j] = min(e(i,j) + c + M[i-1]) for 1 <= i <= j
    return M[n]
```

Since we compute $e(i, j)$, which takes $O(n)$ time, for $O(n^2)$ pairs, the running time of this algorithm is $O(n^3)$.

How do we find the optimal solution to this problem?

In fact, we can solve this problem in $O(n^2)$ time by pre-computing various statistics.

Knapsack problem. Given n objects and a knapsack. Item i weights $w_i > 0$ kilograms and has value $v_i > 0$. Knapsack has capacity of W kilograms. The goal is to fill knapsack so as to maximize the total value.

Let $\text{OPT}(i, w)$ be the max profit from items $1, \ldots, i$ with weight limit w. We have

$$\text{OPT}(i) = \begin{cases} 0 & \text{if } i = 0 \\ \text{OPT}(i-1, w) & \text{if } w_i > w \\ \max\{\text{OPT}(i-1, w), v_i + \text{OPT}(i-1, w-w_i)\} & \text{otherwise} \end{cases}$$

This algorithm basically fills up an n-by-W matrix, and the running time is $O(nW)$, which is pseudo-polynomial. The decision version of Knapsack is NP-complete.

Knapsack approximation algorithm. There exists a poly-time algorithm that produces a feasible solution that has a value within 0.01% of optimum.