CPSC 500: FUNDAMENTALS OF ALGORITHM DESIGN AND ANALYSIS

Introduction to Dynamic Programming

Victor Gan

October 21, 2013

1 INTRODUCTION

This lecture we introduce dynamic programming and three algorithms that exploit it. Dynamic programming is a method for solving problems by breaking them down into subproblems[1].

This is desirable as dynamic programming aims to solve each subproblem only once, compared to a more naive method where many of the subproblems may be calculated many times. Hence, dynamic programming only applies to problems that have overlapping subproblems: if the problem can be solved by combining solutions to non-overlapping subproblems, the strategy is simply "divide and conquer". The problem must also have optimal substructure for dynamic programming to be applicable. Optimal substructure means the optimal solution can be obtained by combining the optimal solutions of substructures. [1]

We introduce three examples of dynamic programming problems: the minimum edit distance, sequence alignment and local sequence alignment problems.

2 MINIMUM EDIT DISTANCE

Imagine you are given two strings, 'howdy' and 'throw'. You want to transform 'howdy' into 'throw', and the only operations possible are inserting a letter and deleting a letter. What is the minimum number of operations needed?

One sequence of operations is 'howdy' $\implies$ 'thowdy' (inserting t) $\implies$ 'throwdy' (inserting r) $\implies$ throwd (deleting y) $\implies$ throw (deleting d). This is four operations, and four is in fact the minimum number of operations needed.

The Minimum Edit Distance problem is defined as follows: Given two strings, $A = a_1 a_2 \ldots a_n$, $B = b_1 b_2 \ldots b_m$, find the minimum number of insert and delete operations to transform A to B.

Note the following observation:

```
1: if $a_n = b_m$ then
2:     the number of operations is the number of operations in $(a_1 \ldots a_{n-1}, b_1 \ldots b_{m-1})$
3: else if $a_n \neq b_n$ then
4:     the number of operations is either the number of operations in
5:     $(a_1 \ldots a_{n-1}, b_1 \ldots b_m) + 1$ or
6:     $(a_1 \ldots a_n, b_1 \ldots b_{m-1}) + 1$
7: end if
```

Line 2 is true because if a_n and b_n are the same, no additional insertions or deletions are needed transform a_n to b_n, and so the problem reduces to a smaller one. Line 5 is the number of operations where a_n is deleted from A, plus an additional operation for deleting a_n. Line 6 is the number of operations where b_m is deleted from B, plus the additional operation.

Motivated by this intuition, let us find a suitable algorithm. Let $F(i, j)$ be the minimum edit distance of $a_1 \ldots a_i, b_1 \ldots b_j$, where $1 \leq i \leq n, 1 \leq j \leq m$. For the general case,

```
function F$(i, j)$
    if $a_i = b_j$ then
        F$(i-1, j-1)$
    else if $a_i \neq b_j$ then
        min( F$(i-1, j)$, F$(i, j-1)$) + 1
    end if
end function
```

And for the base cases,

$F(0,0) = 0$
$F(i,0) = i, 1 \leq i \leq n$
$F(0, j) = j, 1 \leq j \leq m$

Which is intuitively justified: If you wish to transform i letters to an empty set, you require i deletions. If both strings are empty, they are equal and no additional operations are required. As an example of the algorithm, let $A = q$ and $B = r$, where both strings are single letters. As $q \neq r$, $F(1,1) = min(F(0,\ 1), F(1,\ 0)) + 1 = 2$. This is expected as it requires one deletion and one addition to get from q to r.

3 Sequence Alignment

Consider the sequence alignment problem 6.26 of Algorithms by S. Dasgupta, C. H. Papadimitriou, and U. V. Vazirani. Noting that deleting and inserting letters from the previous problem

δ	$-$	A	C	G	T
$-$	$-\infty$	-10	-10	-10	-10
A	-10	4	-6	-6	-6
C	-10	-6	5	-6	-6
G	-10	-6	-6	5	-6
T	-10	-6	-6	-6	4

is analogous to adding a gap in this problem, we can modify the previous algorithm to fit this. First note that the table δ will likely have negative values when a gap "$-$" is present. One example of δ is:
Let $F(i,j)$ be the value of the highest scoring alignment of $X_1 \dots X_i$ and $Y_1 \dots Y_j$, where $1 \le i \le n, 1 \le j \le m$. For the general case,

```
function F$(i,j) = max \begin{cases} F(i-1,j-1)+\delta(X_i,Y_j) \\ F(i-1,j)+\delta(X_i,-) \\ F(i,j-1)+\delta(-,Y_j) \end{cases}$
end function
```

And for the base cases,

$F(0,0) = 0$
$F(i,0) = \sum_{1\le\alpha\le i} \delta(X_\alpha,-), 1 \le i \le n$
$F(0,j) = \sum_{1\le\alpha\le j} \delta(X_\alpha,-), 1 \le j \le m$

4 Local Sequence Alignment

Consider the local sequence alignment problem 6.28 of Algorithms by S. Dasgupta, C. H. Papadimitriou, and U. V. Vazirani. Again let us modify the algorithm for this problem.
For example, let $x = CCCCATAGATTTTT$ and $y = GGGCAAAGGGG$. What is an algorithm that will align local sequences, i.e. align $CATAG$ in X with $CAAAG$ in y?
Again, the table δ will likely have negative values when a gap "$-$" is present.
Let $F(i,j)$ be the value of the highest scoring alignment of a suffix of $X_1 \dots X_i$ and $Y_1 \dots Y_j$, where $1 \le i \le n, 1 \le j \le m$. For the base cases,

```
1: $F(0,0) = 0$
2: $F(i,0) = 0, 1 \le i \le n$
3: $F(0,j) = 0, 1 \le j \le m$
```

Lines 2 and 3 are true because when a string is compared to an empty string, the best local alignment is two empty strings, which has a score of 0. For the general case,

```
function F$(i,j) = max \begin{cases} F(i-1,j-1)+\delta(X_i,Y_j) \\ F(i-1,j)+\delta(X_i,-) \\ F(i,j-1)+\delta(-,Y_j) \\ 0 \end{cases}$
end function
```

5 Conclusion

This lecture walked through three example problems of dynamic programming. In the the minimum edit distance problem, we see that a recursive solution that is found that is a combination of the solutions of overlapping subproblems, hence satisfying the two attributes needed for a dynamic programming to apply. The next problems, sequence alignment and local sequence alignment, showed that this solution structure can be applied with minimal changes for similar problems.

References

[1] Wikipedia, "Dynamic Programming" `http://en.wikipedia.org/wiki/Dynamic_programming`