Lecture 21: Dynamic Programming IV: Piano Fingering, Structural DP (Trees), Vertex Cover, Dominating Set, Treewidth

Lecture Overview

• Piano Fingering
• Structural DP (trees)
• Vertex Cover & Dominating Set in trees
• Beyond trees: treewidth

Readings

CLRS 15

Review:

5 easy steps for DP

1. subproblems (define & count)
2. guessing (what & count)
3. relation (the true test of successful subproblem definition)
4. DP (put pieces together)
5. recover solution to original problem

* 2 kinds of guessing:

in 3: guess which other subproblems to use (used by every DP except Fibonacci)

in 1: add more structure to the subproblem definition, if the obvious subproblems do not result in a useful recursion (used by knapsack DP)

• effectively keep track of many solutions to the subproblem; e.g., in the knapsack problem we kept track of the best knapsacks of various sizes using items $i, \ldots, n$;
• informs parent subproblem about features of the solution.
Piano fingering:

[Parncutt, Sloboda, Clarke, Rackallio, Desain, 1997]
[Hart, Bosch, Tsai 2000]
[Al Kasimi, Nichols, Raphael 2007] etc.

- given musical piece to play, say sequence of (single) notes with right hand
- metric \( d(f,p,g,q) \) of difficulty going from note \( p \) with finger \( f \) to note \( q \) with finger \( g \)
  
  e.g., \( 1 < f < g \) & \( p > q \) \( \implies \) uncomfortable
  stretch rule: \( p \ll q \) \( \implies \) uncomfortable
  legato (smooth) \( \implies \) \( \infty \) if \( f = g \)
  weak-finger rule: prefer to avoid \( g \in \{4,5\} \)
  \( 3 \rightarrow 4 \) & \( 4 \rightarrow 3 \) annoying \( \sim \) etc.

First Attempt:
1. subproblem = min difficulty for suffix notes\([i:]\)
2. guessing = finger \( f \) for first note\([i]\)
3. \( \text{DP}[i] = \min \left( \text{DP}[i+1] + d(\text{note}[i], f, \text{note}[i+1], g) \right) \)
   \( \rightarrow \) not enough information

Second Attempt, enriching the subproblems we keep track of:
1. subproblem \( \text{DP}[i, f] = \min \) difficulty for suffix note\([i:]\) given finger \( f \) on note\([i]\)
2. guessing = finger \( g \) for next note\([i+1]\)
3. \( \text{DP}[i, f] = \min_{g \in \text{range}(F)} (\text{DP}[i+1, g] + d(\text{note}[i], f, \text{note}[i+1], g)) \)
   \( \leftarrow \#\) fingers = 5 for humans
   \( \text{DP}[n, f] = 0 \)
4. \( F^n \) subproblems, \( F \) choices per subproblem \( \implies O(F^2n) \) time
5. recovering original solution: \( \min_{f \in \text{range}(F)} (\text{DP}[\emptyset, f]) \)
Structural DP:

Follow combinatorial structure of the problem, usually richer than mere subsequences.
* for DP on trees, useful subproblem is subtree rooted at vertex $v$, for all $v$

Figure 1: DP on Trees.

Vertex Cover:

Find minimum set of vertices (cover) such that every edge is covered on at least 1 end

Figure 2: Vertex Cover of Petersen’s Graph.

- NP-complete\(^1\) in general graphs
- polynomial time for trees:
  1. subproblem = min. cover for subtree rooted at $v$
     $\implies n$ subproblems
  2. guessing = is $v$ in cover?
     - $\implies 2$ choices

\(^1\)Recall from last lecture that NP-complete problems is a family of problems which are all equivalent to each other and for which no polynomial-time algorithm is known. Most computer scientists believe that no efficient algorithm exists for these problems, but showing this is beyond the power of current techniques. More on NP-completeness later in the term.
Figure 3: Vertex Cover.

- YES \implies children edges already covered :-)
  \implies left with children subtrees
- NO \implies all children must be in cover otherwise the edges adjacent to \( v \) will not be covered
  \implies left with grandchildren subtrees

3. \( DP[v] = \min(1 + \sum(DP[c] \text{ for } c \in \text{children}[v]), \text{YES CASE}) \)
   \#children of \( v \) + \sum(DP[g] \text{ for } g \in \text{grandchildren}(v)) \text{ NO CASE}

4. time = \( O(\sum \text{deg}(v)) = O(E) = O(n) \) (because the edges going out of node \( v \) are “explored” at most twice by the recursion: once for computing \( DP[v] \), and once for computing \( DP[\text{father}[v]] \))

5. solution to the original problem: \( DP[\text{root}] \)

**Dominating set:**

Find minimum set of vertices such that every vertex is in or adjacent to set
- again NP-complete in general graphs, polynomial time on trees.

Figure 4: Dominating Set in Petersen’s Graph.
1. subproblem = min. dom. for subtree rooted at \( v \)

2. guessing = is \( v \) in dom. set?
   - YES \( \implies \) children become already dominated! :-)
   - NO \( \implies \) must put at least one child in dom. set, otherwise \( v \) won’t be dominated
     \( \implies \) on the positive side, the chosen child’s children will be automatically dominated

3. Let us try to write down a structural DP recursion for this problem:

   In the YES CASE: we have to pay 1 for choosing \( v \) and then
   - sum up the solutions for all the children — too pessimistic because the children are already dominated...
   - sum up the solutions for all the grandchildren — too pessimistic, because it is possible that the best solution for the subtrees rooted at the children of \( v \) involves choosing some of the children even though the children do not need to be dominated...

   This discussion leads us to enrich the set of subproblems as follows:
   \[
   \text{DP}(v) = \text{min. dom. set for subtree rooted at } v
   \]
   \[
   \text{DP}'(v) = \text{min. dom. set for subtree rooted at } v \text{ with no requirement of dominating } v
   \implies 2n \text{ subproblems total}
   \]

4. Now we can write down the following recursive formula for DP (see Figure 5).

   \[
   \text{DP}[v] = \min(1 + \sum(\text{DP}'[c] \text{ for } c \text{ in children}[v]), \text{YES CASE})
   \]
   \[
   \min_{d \in \text{children}[v]} \{1 + \sum(\text{DP}'(g) \text{ for } g \text{ in children}[d]) + \sum(\text{DP}[c] \text{ for } c \neq d \text{ in children}[v])\} \text{ \ NO CASE}
   \]

---

Figure 5: Structure of the Dynamic Programming Solution for Dominating Set in Trees.
5. Recursion for DP:\':
\[DP'[v] = \min(1 + \sum (DP'[c] \text{ for } c \text{ in children}[v]), \text{ YES CASE} \sum (DP[c] \text{ for } c \text{ in children}[v]), \text{ NO CASE})\]

6. time = \(O(\sum \text{deg}(v)) = O(E) = O(n)\) (justification similar to that in the vertex cover problem)

7. solution to original problem DP[root]

**Beyond Trees** — You are not responsible for this material.

**Treewidth:**

Many graphs are “thick trees” with reasonable “thickness” (~ 7 e.g.).

- Most problems that are NP-complete in general can be solved in such graphs via DP

![Figure 6: A graph (top) and its tree decomposition (bottom). The treewidth is 3.](image-url)