CS 224: Advanced Algorithms	Fall 2014
Lecture 24 – Nov. 20, 2014	
Prof. Jelani Nelson	*Scribe: Xiaoyu He*

Overview

Today we will move to a new topic: another way to deal with NP-hard problems. We have already seen approximation algorithms for NP-Hard algorithms. This lecture we will bite the bullet and solve NP-hard problems in the "fastest" possible exponential time. Here are the techniques we will cover and the problems we will apply them to:

1. Exponential Divide and Conquer $\implies$ Travelling Salesman.
2. Pruned brute force $\implies$ 3-SAT.
3. Randomization $\implies$ 3-SAT.
4. Inclusion-Exclusion/Fast Möbius Transform $\implies$ k-coloring.

Our goal is to find the best time/space complexity for exponential time solutions to these problems. Sometimes we can reduce to polynomial space even when the natural implementation of some algorithm requires exponential space. Throughout we drop polynomial factors: the notation $O^*(f(n))$ means $O(\text{poly}(n)f(n))$, i.e. up to polynomial factors.

For each problem, we first describe naive brute-force solutions and then show how to improve the exponential part of the time or space complexity.

1 Exponential Divide and Conquer $\implies$ Travelling Salesman

Here is our formulation of travelling salesman. The input is

- Set P of points.
- Metric distance function $d : P \times P \to \mathbb{R}_{\geq 0}$ (i.e. makes P into a finite metric space).
- Two identified points $s, t \in P$.

Goal: Visit every vertex starting at s, ending at t, minimizing total tour length.

Remark: it is an open problem to find an algorithm for travelling salesman in $O^*(2^n)$ time with $\text{poly}(n)$ space. Can get one or the other combining the following algorithms.

1.1 Brute Force

If we try all $(n-2)!$ paths, each path taking n time to check, runtime is $O^*(n!)$, and space is poly(n).

1.2 Dynamic Programming

With dynamic programming we can improve to $O^*(2^n)$ time and at the cost of also using $O^*(2^n)$ space.

Define $f(x, S)$ to be the length of the shortest $x - t$ path visiting every node in S exactly once. Dynamically solve for all $f(x, S)$ for all pairs x, S, using

$$f(x, S) = \min_{y \in S}\{\text{dist}(x, y) + f(y, S - \{x\})\}.$$

The minimal path is $f(s, P - \{s, t\})$. Dynamic programming needs to store $O(n \cdot 2^n)$ states, making $O(n)$ steps from each step, so the time and space are both $O^*(2^n)$.

1.3 Divide and Conquer

If the exponential space is prohibitive, we can get $O((4+\varepsilon)^n)$ time, poly(n) space, for any $\varepsilon > 0$.

Let $OPT(U, s, t)$ be the length of the cheapest s-t tour touching each point in $U \subseteq P$ once. Divide and Conquer by splitting the path into two halves in all possible ways:

$$OPT(U, s, t) = \min_{S,T,m} OPT(S, s, m) + OPT(T, m, t)$$

We take the minimum over all $O^*(\binom{|U|}{|U|/2})$ ways of partitioning U into subsets S, T and an intermediate point $m \neq s, t$ satisfying $|S| = \lfloor \frac{|U|}{2} \rfloor + 1$, $T \cup S = U$, $T \cap S = m$. Compute $OPT(P, s, t)$ recursively; the base case is $|U| \leq 2$. Space is poly(n). Time satisfies

$$T(n) = (n-2)\binom{n-2}{\lceil \frac{n-2}{2} \rceil} \cdot 2 \cdot T(n/2),$$

which solves to $T(n) = O(4^n \cdot n^{C \log n}) = O^*((4+\varepsilon)^n)$ for all $\varepsilon > 0$, where $T(n)$ is the time for n nodes.

2 Pruned Brute Force and Randomization $\implies$ 3-SAT

The input for 3-SAT involves:

- Logical expression $\phi = C_1 \wedge C_2 \wedge \ldots \wedge C_m$.
- Each clause C_j is a $\vee$ of three literals: $x_{i_1} \vee x_{i_2} \vee x_{i_3}$ (possibly negated)

- Variables x_i, $i = 1, \ldots, n$.

Goal: Decide whether there exists an assignment $x = a$ of truth values which makes ϕ evaluate to true.

Remark: There are at most $O(n^3)$ distinct possible clauses C_i so we may assume $m = O(\text{poly}(n))$.

The current records for 3-SAT are randomized $O^*(1.30704^n)$ [2] and deterministic $O^*(1.3303^n)$ [4].

2.1 Brute Force

Try all 2^n assignments: time $O^*(2^n)$, space poly(n).

2.2 Pruned Brute Force

The following example of pruned brute force is from [1, Section 4] (further speedups along these lines are also discussed there).

We introduce the following conditional notation: $\phi|x_1\bar{x_2}$ is the new expression obtained from ϕ by setting x_1 true, x_2 false, and simplifying.

Let $C_1 = x \vee y \vee z$ (possibly with negations). Try all three possibilities recur:

```
def SAT(φ):

    if SAT(φ|x) return True
    else if SAT(φ|y) return True
    else if SAT(φ|z) return True
    else return False
```

The runtime is $O^*(3^n)$, since each recursive step reduces the number of variables remaining by at least one. Thus if $T(n)$ is the maximum amount of time to solve an instance ϕ with n variables, we have $T(n) \leq 3T(n-1) + O(1)$.

This is worse than regular brute force. However, we can offer the following improvement: if we reach the second if, then the first one failed and we may assume $\bar{x}$! Thus:

```
def SAT(φ):

    if SAT(φ|x) return True
    else if SAT(φ|x̄y) return True
    else if SAT(φ|x̄ȳz) return True
    else return false
```

Our improved runtime satisfies $T(n) \leq T(n-1) + T(n-2) + T(n-3) + O(1)$, which solves to $O^*(1.8393^n)$, the constant coming from the largest real root of $x^3 - x^2 - x - 1 = 0$. This pruning algorithm can be further improved, but we will show something even better.

2.3 Another Brute Force

Suppose a^* satisfies ϕ. WLOG, we know an assignment a s.t. $dist(a, a^*) \leq n/2$ (run the following algorithm twice, one assuming $a = 0$ all zeroes, one assuming $a = 1$ all ones), where the distance here is the Hamming distance.

Start with a. While there exists an unsatisfied C, try all three of the variables in C and flip their assignments in a, recurse for each of them. Limit the recursion depth to $n/2$ iterations. Given our assumption that a^* is within $n/2$ Hamming distance away from a, this algorithm must find a^*. The runtime is $O^*(3^{n/2}) = O^*(1.7321^n)$. This is already better than the previous solution, and we will see how to further improve it via randomization.

2.4 Randomization

In this section we present Schöning's randomized version [6] of the algorithm in the previous section, achieving runtime $O^*((4/3)^n)$, and space poly(n).

```
Algorithm S:
    Pick a uniformly at random.
    Repeat at most t times:
        (i) Let C be the first clause not satisfied by a.
        (ii) Pick a random variable in C and flip its assignment in a.
```

Schöning's algorithm is to repeat S until we succeed. The parameter t we will tune later.

Remark: Similar algorithm for 2-SAT [5] needs to run for $O(n^2)$ steps to find SAT assignment with probability $\geq 2/3$.

2.4.1 Analysis of Schöning's algorithm

Recall the definition of a Markov chain: a directed graph, where every edge e has a assigned probability $p(e)$, such that for all vertices v,

$$\sum_{e \text{ leaving} v} p_e = 1.$$

Let G be a path on $n+1$ vertices, with edges in both directions between adjacent vertices. Vertex i represents all assignments a satisfying $dist(a, a^*) = i$ (fix a solution a^* if there are multiple). At every step in Algorithm S, we are moving left or right on the graph one step, i.e. this is a random walk on $0, \ldots, n$.

At an intermediate vertex $0 < d < n$, there are 3 variables in the clause C we pick, at least one of which a disagrees with a^* on, so the probability of going left towards 0 is at least $1/3$.

Thus, if we calculate the expected time of hitting 0 by going left with probability $1/3$ at every point on an infinite ray $0, 1, \ldots$ this is an upper bound for the amount of time to get to 0 in algorithm S. We simplify even more and only compute the probability of hitting 0 at exactly the t-th step.

Starting off at distance k from 0, what is the probability that we land at 0 at exactly time $t = 3k$?

We will show that this probability is at least $(3/4)^n$:

$$P[k \text{ steps to the right}, 2k \text{ to the left}] \geq \binom{3k}{k}\Big(\frac{1}{3}\Big)^{2k} \cdot \Big(\frac{2}{3}\Big)^{k}.$$

By Stirling's formula, and some calculation, we get that the probability above is

$$\Theta\Big(\frac{1}{\sqrt{k}} \cdot \frac{3^{3k}}{2^{2k}} \cdot \Big(\frac{1}{3}\Big)^{2k} \cdot \Big(\frac{2}{3}\Big)^{k}\Big) = \Theta\Big(\frac{1}{\sqrt{k}} \cdot \frac{1}{2^k}\Big).$$

Thus averaging over all possibilities for k, weighted by their probabilities:

$$\begin{aligned} P[\text{success}] &\geq \sum_{k=0}^{n} P[\text{dist}(a, a^*) = k] \cdot \Theta(\frac{1}{\sqrt{k}} \cdot \frac{1}{2^k}) \\ &\geq \frac{1}{\sqrt{n}} \sum_{k=1}^{n} \binom{n}{k} \frac{1}{2^{n+k}} \\ &= \frac{1}{\sqrt{n}} \frac{1}{2^n} \Big(1 + \frac{1}{2}\Big)^n \\ &= \frac{1}{\sqrt{n}} \frac{3^n}{4^n}, \end{aligned}$$

which is the order $P = n^{-1/2}(3/4)^n$ desired. Thus if we run algorithm S at least $T = \ln(1/\delta)/P$ times with $t = 3n$, the probability of not finding a satisfying assignment any any of the iterations is at most $(1 - P)^T \leq e^{-T/P} \leq \delta$. Thus for constant success probability, the running time is $O^*((4/3)^n)$.

3 Inclusion-Exclusion $\implies$ k-coloring

Given an undirected graph $G = (V, E)$, $|V| = n$, $|E| = m$, a k-coloring of G is a function $c : V \to \{1, \ldots, k\}$ such that $(u, v) \in E \implies c(u) \neq c(v)$, i.e. no monochromatic edges.

Goal: Decide if a k-coloring exists. Equivalently, decide if V can be partitioned into k independent sets (sets of vertices with no edges between them).

3.1 Brute Force

If we simply try all possible colorings, this takes $O^*(k^n)$ time, poly(n) space.

3.2 Dynamic Programming

We compute with dynamic programming $F(t, S)$ = indicator of whether or not $S \subseteq V$ can be partitioned into t independent sets; we want $F(k, V)$.

To compute $F(t, S)$, we use the recurrence:

$$F(t, S) = \bigvee_{T \subseteq S} (T \text{ independent} \wedge F(t-1, S-T))$$

The runtime is at most

$$\sum_{S \subseteq V} 2^{|S|} = O^*(3^n)$$

by the binomial theorem. The space is $O^*(2^n)$ since we will be storing $\text{poly}(n)$ data for each subset $S \subseteq V$.

3.3 Inclusion-Exclusion

Here we follow the presentation of [3] (see that survey for more history and references).

Recall the statement of the Principle of Inclusion/Exclusion:

Theorem 1. *(Inclusion/Exclusion) For any sets $R \subseteq T$,*

$$\sum_{R \subseteq S \subseteq T} (-1)^{|T-S|} = [R = T],$$

i.e. the sum is zero unless $R = T$.

Proof. If $R = T$ then the sum has just one term 1. Otherwise, pick some $t \in T - R$, so that $S \mapsto S \oplus \{t\}$ (i.e. put in t if it is not there, remove it if it is) is a bijection between the terms with $|T - S|$ even and $|T - S|$ odd. Thus the whole sum cancels. □

Define $f(S)$ to be 1 iff S is a nonempty independent set, and define the dual function $\hat{f}(S) = \sum_{R \subseteq S} f(R)$.

Claim 2. *G is k-colorable iff*

$$\sum_{S \subseteq V} (-1)^{n-|S|} \hat{f}(S)^k > 0. \tag{1}$$

Proof. Each $\hat{f}(S)^k$ expands as a sum

$$\hat{f}(S)^k = \sum_{R_i \subseteq S} f(R_1) f(R_2) \cdots f(R_k),$$

over all choices of k subsets R_i of S. These vanish unless all of the R_i are independent. For any fixed choice of k independent sets $R_1, R_2, \ldots, R_k \subseteq V$, the coefficient of $f(R_1) \cdots f(R_k)$ in (1) is just

$$\sum_{R \subseteq S \subseteq V} (-1)^{|V-S|},$$

where $R = \bigcup R_i$. By Inclusion/Exclusion, the only terms that remain are those in which $R = V$, and these are positive. If there is such a term, V can be written as a union of k independent sets, as desired. Although the R_i themselves need not be disjoint, we can remove elements from them until they are disjoint; they will remain a partition of V by independent sets. □

Thus we have reduced the problem to: efficiently compute $\hat{f}(S)$ for all S. We can achieve either $O^*(3^n)$ time and poly(n) space or $O^*(2^n)$ time and space.

References

[1] Jeff Erickson. Algorithms. Download at `http://web.engr.illinois.edu/~jeffe/teaching/algorithms/`.

[2] Timon Hertli. "3-SAT Faster and Simpler—Unique-SAT Bounds for PPSZ Hold in General." *SIAM Journal on Computing*, 43(2), pp. 718–729, 2014.

[3] Thore Husfeldt. Invitation to Algorithmic Uses of Inclusion-Exclusion. *ICALP*, pp. 42–59, 2011.

[4] Kazuhisa Makino, Suguru Tamaki, and Masaki Yamamoto. "Derandomizing the HSSW Algorithm for 3-SAT." *Algorithmica*, 67(2), pp. 112-1124, 2013.

[5] Christos H. Papadimitriou. "On selecting a satisfying truth assignment." In *Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science*, pp. 163-169, 1991.

[6] Uwe Schöning. "A probabilistic algorithm for k-SAT based on limited local search and restart." *Algorithmica*, 32, no. 4 (2002): 615-623.