2. Dynamic Programming

COMP6741: Parameterized and Exact Computation

Serge Gaspers

Semester 2, 2015

Contents

1 Dynamic Programming Across Subsets

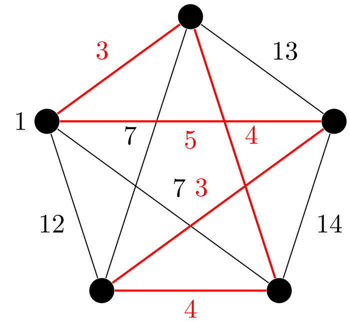
Dynamic Programming across Subsets

- very general technique
- uses solutions of subproblems
- typically stored in a table of exponential size

1.1 Traveling Salesman Problem

TRAVELING SALESMAN PROBLEM (TSP)

Input:	a set of n cities, the distance $d(i,j) \in \mathbb{N}$ between every two cities i and j, integer k
Question:	Is there a permutation of the cities (a *tour*) such that the total distance when traveling from city to city in the specified order, and returning back to the origin, is at most k?



Brute-force: Try all permutations of cities; $O^*(n!)$

Dynamic Programming for TSP
For a non-empty subset of cities $S \subseteq \{2, 3, ..., n\}$ and city $i \in S$:

- $\textsc{Opt}[S; i] \equiv$ length of the shortest path starting in city 1, visits all cities in $S \setminus \{i\}$ and ends in i.

Then,

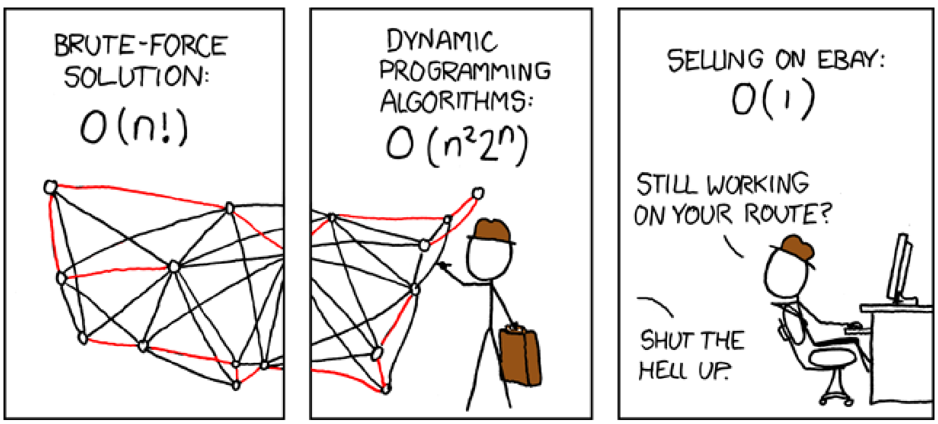
$$\textsc{Opt}[\{i\}; i] = d(1, i)$$
$$\textsc{Opt}[S; i] = \min\{\textsc{Opt}[S \setminus \{i\}; j] + d(j, i) : j \in S \setminus \{i\}\}$$

- For each subset S in order of increasing cardinality, compute $\textsc{Opt}[S; i]$ for each i.
- Final solution:

$$\min_{2 \le j \le n} \{\textsc{Opt}[\{2, 3, ..., n\}; j] + d(j, 1)\}$$

Theorem 1 (Held & Karp '62)**.** *TSP can be solved in time $O(2^n n^2) = O^*(2^n)$.*

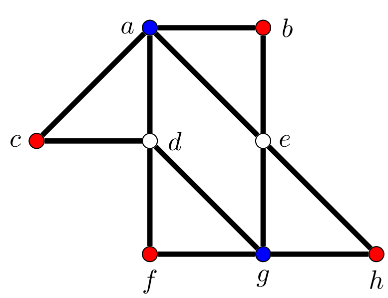
- best known algo for TSP



1.2 Coloring

A *k-coloring* of a graph $G = (V, E)$ is a function $f : V \to \{1, 2, ..., k\}$ assigning colors to V such that no two adjacent vertices receive the same color.

COLORING	
Input:	Graph G, integer k
Question:	Does G have a k-coloring?

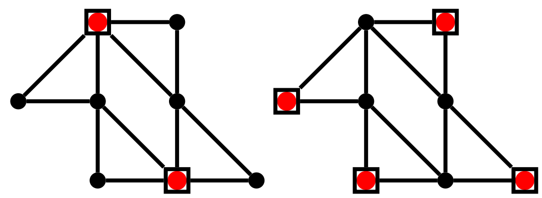


Exercise

Design an $O^*(4^n)$ time algorithm for COLORING.

Maximal Independent Sets

- An independent set is *maximal* if it is not a subset of any other independent set.
- Examples:



Coloring and Maximal Independent Sets

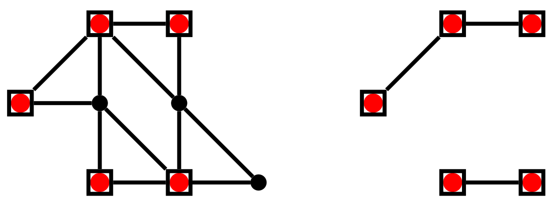
Theorem 2 ([Moon, Moser '65], [Johnson, Yannakakis, Papadimitriou '88])**.** *A graph on n vertices contains at most $3^{n/3} \subseteq O(1.4423^n)$ maximal independent sets. Moreover, they can all be enumerated in time $O^*(3^{n/3})$.*

Lemma 3 ([Lawler '76])**.** *For any graph G, there exists an optimal coloring for G where one color class is a maximal independent set in G.*

Proof. Exercise □

Dynamic Programming for Coloring

- $G[S] \equiv$ subgraph of G induced by the vertices in S



- $\textsc{Opt}[S] \equiv$ minimum k such that $G[S]$ is k-colorable.
- Then,

$$\begin{aligned}\textsc{Opt}[\emptyset] &= 0\\ \textsc{Opt}[S] &= 1 + \min\{\textsc{Opt}[S \setminus I] : I \text{ maximal ind. set in } G[S]\}\end{aligned}$$

- go through the sets S in order of increasing cardinality
- to compute $\textsc{Opt}[S]$, generate all maximal independent sets I of $G[S]$
- this can be done in time $|S|^2 3^{|S|/3}$
- time complexity:

$$\sum_{s=0}^{n} \binom{n}{s} s^2 3^{s/3} \le n^2 \sum_{s=0}^{n} \binom{n}{s} 3^{s/3} = n^2(1+3^{1/3})^n = O(2.4423^n)$$

[Recall the *Binomial Theorem*: $(x+y)^n = \sum_{k=0}^{n} \binom{n}{k} x^{n-k} y^k$.]

Theorem 4 ([Lawler '76])**.** $\textsc{Coloring}$ *can be solved in time $O(2.4423^n)$.*

- was best known algorithm for 25 years (until [Eppstein '01])
- current best: $O^*(2^n)$ [Bjørklund & Husfeldt '06], [Koivisto '06]

k-Coloring for small k

k-COLORING Input: Graph G, integer k Question: Does G have a k-coloring?

- $k \leq 2$: polynomial
- $k > 2$: NP-complete

Algorithm for 3-Coloring

Theorem 5 ([Lawler '76])**.** 3-COLORING *can be decided in time* $O(1.4423^n)$*.*

Proof. For every maximal independent I set of G, check if $G - I$ is 2-colorable. □

current best: $O(1.3289^n)$ [Eppstein '01]

Algorithm for 4-Coloring

Theorem 6. 4-COLORING *can be decided in time* $O(1.7851^n)$*.*

Proof.
- For each maximal independent set I of G of size at least $n/4$, check if $G - I$ is 3-colorable.
- We need to prove that each 4-colorable graph G has a 4-coloring where one color class is a maximal i.s. of size $\geq n/4$?
- Pick a 4-coloring of G.
- ≥ 1 color class contains $\geq n/4$ vertices.
- If this color class is not a maximal i.s., recolor some other vertices such that it becomes a maximal i.s.
- Running time: $O(3^{n/3} 1.3289^{3n/4}) \subseteq O(1.7851^n)$

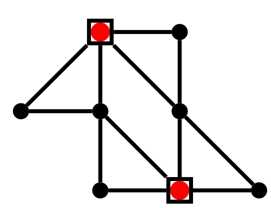
□

current best: $O(1.7272^n)$ [Fomin, Gaspers, Saurabh '07]

1.3 Dominating Set in bipartite graphs

A *dominating set* in a graph $G = (V, E)$ is a subset of vertices $S \subseteq V$ such that each vertex of G is either in S or adjacent to a vertex in S.

DOMINATING SET Input: Graph G, integer k Question: Does G have a dominating set of size k?



A graph $G = (V, E)$ is *bipartite* if its vertex set can be partitioned into two independent sets.

DOMINATING SET IN BIPARTITE GRAPHS Input: Bipartite graph G, integer k Question: Does G have a dominating set of size k?

Note: DOMINATING SET IN BIPARTITE GRAPHS is NP-complete.

Algorithm for Dominating Set in Bipartite Graphs

Partition V into independent sets A and B, with $|B| \geq |A|$.

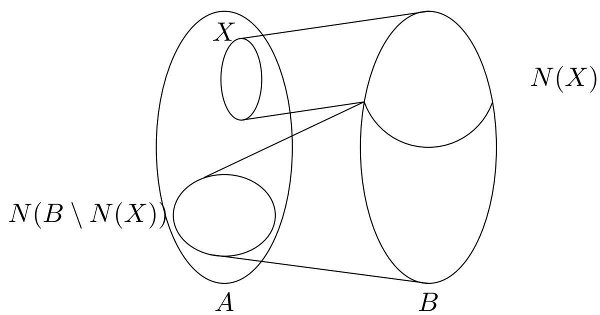
The algorithm has 2 phases:

- **Preprocessing phase**: compute for each $X \subseteq A$ a subset $\mathsf{Opt}[X]$ which is a smallest subset of B that dominates X.
- **Main phase**: for each subset $X \subseteq A$, compute a dominating set D of G of minimum size such that $D \cap A = X$.

Main phase. For a vertex subset $X \subseteq A$, a dominating set D of G of minimum size such that $D \cap A = X$ is obtained by setting

$$D := X \cup (B \setminus N(X)) \cup \mathsf{Opt}[A \setminus (X \cup N(B \setminus N(X)))]$$

if $A \setminus X$ contains no degree-0 vertex. (If $A \setminus X$ contains a degree-0 vertex, we skip this set X, because there is no dominating set D of G with $D \cap A = X$.)



Preprocessing phase. Let $B = \{b_1, \ldots, b_{|B|}\}$. We compute for each $X \subseteq A$ and integer k, $0 \leq k \leq |B|$, a subset $\mathsf{Opt}[X, k] \subseteq \{b_1, \ldots, b_k\}$ which is defined as

- a smallest subset of $\{b_1, \ldots, b_k\}$ that dominates X if $X \subseteq N(\{b_1, \ldots, b_k\})$, and
- B if $X \not\subseteq N(\{b_1, \ldots, b_k\})$.

Note: $\mathsf{Opt}[X, |B|] = \mathsf{Opt}[X]$.

Base cases

$$\begin{aligned} \mathsf{Opt}[\emptyset, k] &= \emptyset & \forall k \in \{0, \ldots, |B|\}, \\ \mathsf{Opt}[X, 0] &= B & \forall X,\ \emptyset \subsetneq X \subseteq A. \end{aligned}$$

Dynamic Programming recurrence

$$\mathsf{Opt}[X, k] = \begin{cases} \mathsf{Opt}[X, k-1] & \text{if } |\mathsf{Opt}[X, k-1]| < 1 + |\mathsf{Opt}[X \setminus N(b_k), k-1]| \\ \{b_k\} \cup \mathsf{Opt}[X \setminus N(b_k), k-1] & \text{otherwise} \end{cases}$$

for each X, $\emptyset \subsetneq X \subseteq A$ and $k \in \{1, \ldots, |B|\}$.

Theorem 7 ([Liedloff '08]). Dominating Set in Bipartite Graphs *can be solved in* $O^*(2^{n/2})$ *time, where* n *is the number of vertices of the input graph.*

2 Further Reading

- Chapter 3, *Dynamic Programming* in Fedor V. Fomin and Dieter Kratsch. Exact Exponential Algorithms. Springer, 2010.