Exponential time algorithms

Algorithms and networks

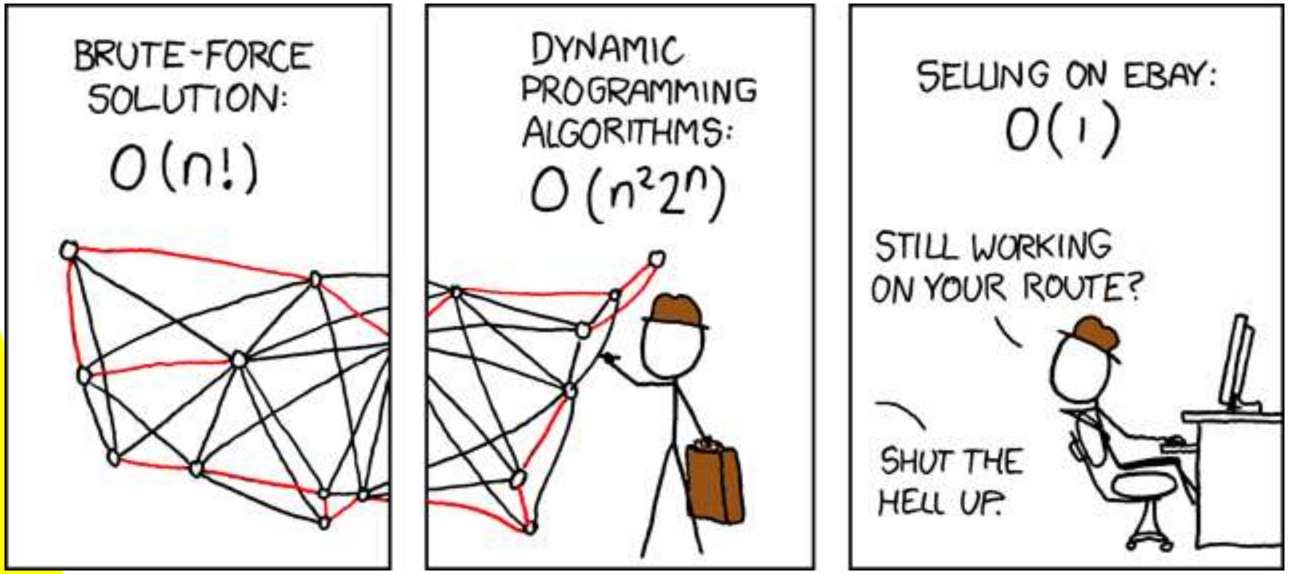
Universiteit Utrecht

Today

- Exponential time algorithms: introduction
- Techniques
- 3-coloring
- 4-coloring
- Coloring
- Maximum Independent Set
- TSP

What to do if a problem is NP-complete?

- Solve on special cases
- Heuristics and approximations
- Algorithms that are fast on average
- Good exponential time algorithms
- …


BRUTE-FORCE SOLUTION:
$O(n!)$
DYNAMIC PROGRAMMING ALGORITHMS:
$O(n^2 2^n)$
SELLING ON EBAY:
$O(1)$
STILL WORKING ON YOUR ROUTE?
SHUT THE HELL UP.

Good exponential time algorithms

- Algorithms with a running time of $c^n.p(n)$
  - c a constant
  - p() a polynomial
  - Notation: $O^*(c^n)$
- Smaller c helps a lot!

Important techniques

- Dynamic programming
- Branch and reduce
  - Measure and conquer (and design)
- Divide and conquer
- Clever enumeration
- Local search
- Inclusion-exclusion

Held-Karp algorithm for TSP

- O($n^2 2^n$) algorithm for TSP
- Uses Dynamic programming
- Take some starting vertex *s*
- For set of vertices R ($s \in R$), vertex $w \in$ R, let
  - B(R,*w*) = minimum length of a path, that
    - Starts in *s*
    - Visits all vertices in R (and no other vertices)
    - Ends in *w*

TSP: Recursive formulation

- $B(\{s\},s) = 0$
- If $|X| > 1$, then
  - **$B(X,w) = \min_{v \in X - \{w\}} B(X-\{w\}, v) + w(v,w)$**
- If we have all $B(V,v)$ then we can solve TSP.
- Gives requested algorithm using DP-techniques.

Notation

- O*(f(n)): hides polynomial factors, i.e.,
- O*(f(n)) = O(p(n)*f(n)) for some polynomial p

ETH

- Exponential Time Hypothesis (ETH): Satisability of n-variable 3-CNF formulas (3-SAT) cannot be decided in subexponential worst case time, e.g., it cannot be done in $O^*(2^{o(n)})$ time.

Running example

- Graph coloring
  - Several applications: scheduling, frequency assignment (usually more complex variants)
  - Different algorithms for small fixed number of colors (3-coloring, 4-coloring, …) and arbitrary number of colors
  - 2-coloring is easy in $O(n+m)$ time

3-coloring

- O*(3^n) is trivial
- Can we do this faster?

3-coloring in O*(2^n) time

- G is 3-colorable, if and only if there is a set of vertices S with
  - S is independent
  - G[V-S] is 2-colorable
- Algorithm: enumerate all sets, and test these properties (2^n tests of O(n+m) time each)

3-coloring

- Lawler, 1976:
  - We may assume S is a *maximal independent set*
  - Enumerating all maximal independent sets in O*($3^{n/3}$) = O*(1.4423^n) time
    - There are O*($3^{n/3}$) maximal independent sets (will be proved later.)
  - Thus O*(1.4423^n) time algorithm for 3-coloring
- Schiermeyer, 1994; O*(1.398^n) time
- Beigel, Eppstein, 1995: O*(1.3446^n) time

4-coloring in O*(2^n) time

- Lawler, 1976
- G is 4-colorable, if and only if we can partition the vertices in two sets X and Y such that G[X] and G[Y] are both 2-colorable
- Enumerate all partitions
  – For each, check both halves in O(n+m) time

4-coloring

- Using 3-coloring
  - Enumerate all maximal independent sets S
  - For each, check 3-colorability of G[V-S]
  - $1.4423^n * 1.3446^n = 1.939^n$
- Better: there is always a color with at least $n/4$ vertices
  - Enumerate all m.i.s. S with at least $n/4$ vertices
  - For each, check 3-colorability of G[V-S]
  - $1.4423^n * 1.3446^{3n/4} = 1.8009^n$
- Byskov, 2004: $O^*(1.7504^n)$ time

Coloring

- Next: coloring when the number of colors is some arbitrary number (not necessarily small)
- First: a dynamic program

Coloring with dynamic programming

- Lawler, 1976: using DP for solving graph coloring.
- $X(G) = \min_{S \text{ is m.i.s. in } G} 1 + X(G[V-S])$
- Tabulate chromatic number of G[W] over all subsets W
  - In increasing size
  - Using formula above
  - $2^n * 1.4423^n = 2.8868^n$

Coloring

- Lawler 1976: 2.4423^n (improved analysis)
- Eppstein, 2003: 2.4151^n
- Byskov, 2004: 2.4023^n
  - All using $O^*(2^n)$ memory
  - Improvements on DP method
- Björklund, Husfeld, 2005: 2.3236^n
- 2006: Inclusion/Exclusion

Inclusion-exclusion

- Björklund and Husfeld, 2006, and independently Koivisto, 2006
- O*(2^n) time algorithm for coloring
- Expression: number of ways to cover all vertices with *k* independent sets

First formula

- Let $c_k(G)$ be the number of ways we can cover all vertices in G with *k* independent sets, where the stable sets may be overlapping, or even the same
  - Sequences $(V_1,\ldots,V_k)$ with the union of the V_i's = V, and each V_i independent
- Lemma: G is *k*-colorable, if and only if $c_k(G) > 0$

Counting independent sets

- Let s(X) be the number of independent sets that do not intersect X, i.e., the number of independent sets in G(V-X).
- We can compute all values s(X) in O*(2^n) time.
  - s(X) = s(X ∪ {v}) + s(X ∪ {v} ∪ N(v)) for $v \notin X$
    - Count IS's with v and IS's without v
  - Now use DP and store all values
  - Polynomial space slower algorithm also possible, by computing s(X) each time again

Expressing c$_k$ in *s*

$$c_k(G) = \sum_{X \subseteq V} (-1)^{|X|} s(X)^k$$

- s(X)^k counts the number of ways to pick *k* independent sets from V-X
- If a pick covers all vertices, it is counted in s(∅)
- If a pick does not cover all vertices, suppose it covers all vertices in V-Y, then it is counted in all X that are a subset in Y
  - With a +1 if X is even, and a -1 if X is odd
  - Y has equally many even as odd subsets: total contribution is 0

Explanations

- Consider the number of k-tuples (W(1), … , W(k)) with each W(i) an independent set in G
- If we count all these k-tuples, we count all colourings, but also some ***wrong*** k-tuples: those which avoid some vertices
- So, subtract from this number all k-tuples of independent sets that avoid a vertex v, for all v
- However, we now subtract too many, as k-tuples that avoid two or more vertices are subtracted twice
- So, add for all pairs $\{v,w\}$, the number of k-tuples that avoid both v and w
- But, then what happens to k-tuples that avoid 3 vertices???
- Continue, and note that the parity tells if we add or subtract…
- This gives the formula of the previous slide

The algorithm

- Tabulate all s(X)
- Compute values $c_k(G)$ with the formula
- Take the smallest k for which $c_k(G) > 0$

$$O^*(2^n)$$

Maximum independent set

- Branch and reduce algorithm (folklore)
- Uses:
  - Branching rule
  - Reduction rules

Two simple reduction rules

- Reduction rule 1: if v has degree 0, put v in the solution set and recurse on G-v
- Reduction rule 2: if v has degree 1, then put v in the solution set. Suppose v has neighbor w. Recurse on G – $\{v,w\}$.
  - If v has degree 1, then there is always a maximum independent set containing v

Idea for branching rule

- Consider some vertex v with neighbors w_1, w_2, … , w_d. Suppose S is a maximum independent set. One of the following cases must hold:

1. $v \in S$. Then $w_1, w_2, \ldots, w_d$ are not in S.
2. For some i, $1 \leq i \leq d$, $v, w_1, w_2, \ldots, w_{i-1}$ are not in S and $w_i \in S$. Also, no neighbor of w_i is in S.

Branching rule

- Take a vertex v of minimum degree.
- Suppose v has neighbors $w_1, w_2, \ldots, w_d$.
- Set *best* to 1 + what we get when we recurse on G – $\{v, w_1, w_2, \ldots, w_d\}$. (Here we put v in the solution set.)
- For $i = 1$ to d do
  - Recurse on G – $\{v, w_1, w_2, \ldots, w_i\}$ – N(w_i). Say, it gives a solution of value x. (N(w_i) is set of neighbors of w_i. Here we put w_i in S.)
  - Set *best* = max (*best*, x+1).
- Return *best*

Using some bookkeeping gives the corresponding set S

Analysis

- Say T(n) is the number of *leaves* in the search tree when we have a graph with n vertices.
- If v has degree d, then we have T(n) ≤ (d+1) T(n-d-1).
  - Note that each w_i has degree d as v had minimum degree, so we always recurse on a graph with at least d-1 fewer vertices.
- $d > 1$ (because of reduction rules).
- With induction: T(n) ≤ $3^{n/3}$.
- Total time is O*($3^{n/3}$) = O*(1.4423^n).

Number of maximal independent sets

- Suppose M(n) is maximum number of m.i.s.'s in graph with n vertices
- Choose v of minimum degree d.
- If v has degree 0: number of m.i.s.'s is at most 1* M(n)
- If v has degree 1: number of m.i.s.'s is at most 2* M(n-2)
- If v has degree d>1: number of m.i.s's is at most (d+1)* M(n – d – 1)
- M(n) $\leq 3^{n/3}$ with induction

Some remarks

- Can be done without reduction step
- Bound on number of m.i.s.'s sharp: consider a collection of triangles

A faster algorithm

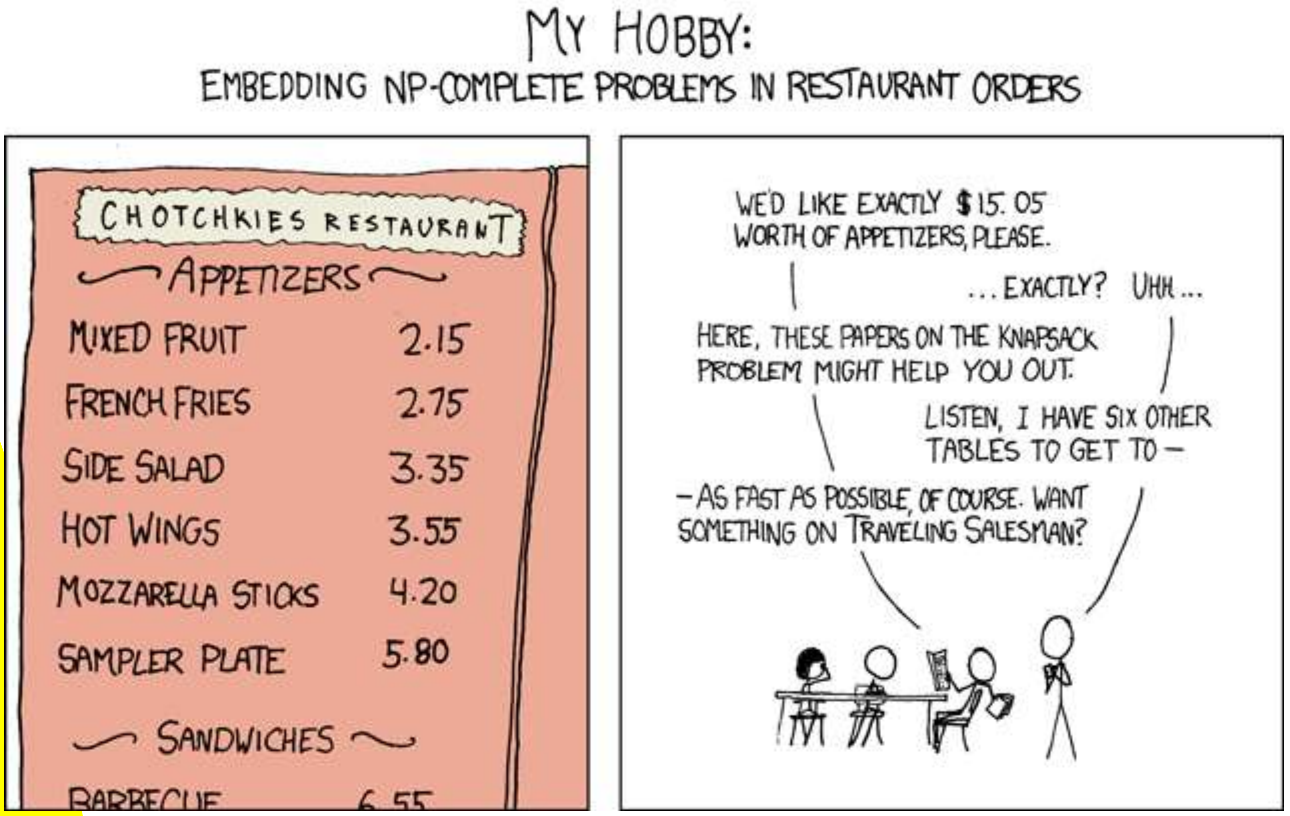
- Reduction rule 3: if all vertices of G have degree at most two, solve problem directly. (Easy in O(n+m) time.)
- New branching rule:
  - Take vertex v of **maximum** degree
  - Take best of two recursive steps:
    - v not in solution: recurse of G – {v}
    - v in solution: recurse on G – {v} – N(v); add 1.

Analysis

- Time on graph with n vertices $T(n)$.
- We have $T(n) \leq T(n-1) + T(n-4) + O(n+m)$
  - As v has degree at least 3, we loose in the second case at least 4 vertices
- Induction: $T(n) = O^*(1.3803^n)$
  - Solve (with e.g., Maple or Mathematica, SAGEmath (http://www.sagemath.org) or Solver from Excel)
    - $x^4 = x^3 + 1$

Maximum Independent Set Final remarks

- More detailed analysis gives better bounds
- Current best known: $O(1.1844^n)$ (Robson, 2001)
  - Extensive, computer generated case analysis!
  - Includes memorization (DP)
- 2005: Fomin, Grandoni, Kratsch: the *measure and conquer* technique for better analysis of *branch and reduce* algorithms
  - Much simpler and only slightly slower compared to Robson


MY HOBBY:
EMBEDDING NP-COMPLETE PROBLEMS IN RESTAURANT ORDERS
CHOTCHKIES RESTAURANT
APPETIZERS
MIXED FRUIT 2.15
FRENCH FRIES 2.75
SIDE SALAD 3.35
HOT WINGS 3.55
MOZZARELLA STICKS 4.20
SAMPLER PLATE 5.80
SANDWICHES
BARBECUE 6.55
WE'D LIKE EXACTLY $15.05 WORTH OF APPETIZERS, PLEASE.
...EXACTLY? UHH...
HERE, THESE PAPERS ON THE KNAPSACK PROBLEM MIGHT HELP YOU OUT.
LISTEN, I HAVE SIX OTHER TABLES TO GET TO –
– AS FAST AS POSSIBLE, OF COURSE. WANT SOMETHING ON TRAVELING SALESMAN?

Final remarks

- Techniques for designing exponential time algorithms
- Other techniques, e.g., local search
- Combination of techniques
- Several interesting open problems