Introduction to Exponential Time Algorithms

séminaire AlGco

Serge Gaspers[1]

[1]LIRMM – Université Montpellier 2, CNRS

January 22, 2009

Outline

Outline

NP-hard problems

- no known polynomial time algorithm for any NP-hard problem
- belief: $P \neq NP$
- ETH: 3-Sat cannot be solved in subexponential time
- (thus many other problems cannot be solved in subexponential time either)

Dealing with NP-hard problems

- Approaches to attack NP-hard problems
  - approximation algorithms
  - randomized algorithms
  - fixed parameter algorithms
  - exact exponential time algorithms
  - heuristics
  - restricting the inputs

Exponential Time Algorithms

- natural question in Algorithms:
  design faster (worst-case analysis) algorithms for problems
- might lead to practical algorithms
  - for small instances
  - subroutines for
    - (sub)exponential time approximation algorithms
    - randomized algorithms with expected polynomial run time
- interesting combinatorics

Solve a NP hard problem

- exhaustive search
  - trivial method
  - try all possible solutions for a ground set on n elements
  - running times for problems in NP
    - SUBSET PROBLEMS: $\mathcal{O}^*(2^n)$ [1]
    - PERMUTATION PROBLEMS: $\mathcal{O}^*(n!)$
    - PARTITION PROBLEMS: $\mathcal{O}^*(c^{n \log n})$
- faster exact algorithms
  - for some problems, it is possible to obtain provably faster algorithms
  - running times $\mathcal{O}(1.0892^n), \mathcal{O}(1.5086^n), \mathcal{O}(1.9977^n)$

[1] $\mathcal{O}^*(f(n)) \equiv \mathcal{O}(f(n) \cdot \text{poly}(n))$

Exponential Time Algorithms in Practice

- How large are the instances one can solve in practice?

Available time	1 s	1 min	1 hour	3 days	6 months
nb. of operations	2^{30}	2^{36}	2^{42}	2^{48}	2^{54}
n^5	64	145	329	774	1756
n^{10}	8	12	18	27	41
1.05^n	426	510	594	681	765
1.1^n	218	261	304	348	391
1.5^n	51	61	71	82	92
2^n	30	36	42	48	54
5^n	12	15	18	20	23
$n!$	12	14	15	17	18

Technology vs. Algorithms

- Suppose a 2^n algo enables us to solve instances up to size x
- Faster processors
  - processor speed doubles after 18–24 months (according to Moore's law)
  - can solve instances up to size $x+1$
- Faster algorithm
  - design a $2^{n/2} = 1.4143^n$ time algorithm
  - can solve instances up to size $2 \cdot x$

Outline

1. Introduction
   - Exponential Time Algorithms
   - Problem Definitions

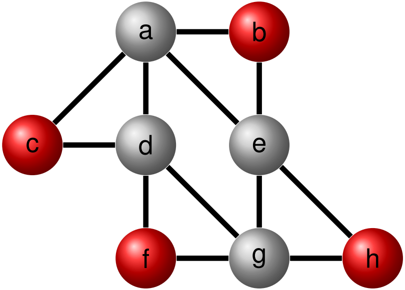
2. Algorithm Design Techniques
   - Dynamic Programming across Subsets
   - Branch & Reduce
   - Memorization
   - Treewidth
   - Treewidth combined with Branch & Reduce
   - Iterative Compression
   - Inclusion-Exclusion

3. Conclusion

Subset Problem: Maximum Independent Set

Maximum Independent Set (MIS)

- Input: A graph $G = (V, E)$.
- Output: An independent set of G of maximum cardinality.
- $I \subseteq V$ is an independent set if the vertices in I are pairwise non-adjacent.



Permutation Problem: TRAVELING SALESMAN

TRAVELING SALESMAN PROBLEM (TSP)

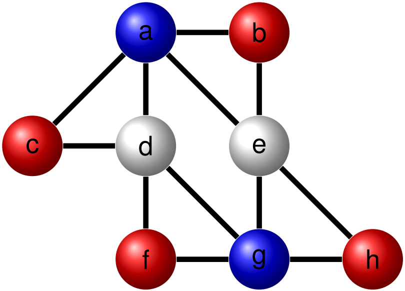
- Input: a set of n cities, the distance $d(i,j)$ between every two cities i and j.
- Output: A tour visiting all cities with minimum total distance.
- A tour is a permutation of the cities, starting and ending in city 1.

- Trivial algorithm checks all the permutations of the cities
- Running time $\mathcal{O}(n!)$

Partition Problem: COLORING

COLORING (COL)

- Input: A graph $G = (V, E)$.
- Output: A coloring of V with the smallest number of colors.
- A coloring $f : V \to \{1, 2, ..., k\}$ is a function assigning colors to V such that 2 adjacent vertices never receive the same color.



Outline

Dynamic Programming across Subsets

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

Dynamic Programming for TSP

Traveling Salesman Problem (TSP)

- Input: a set of n cities $\{1, 2, ..., n\}$, the distance $d(i, j)$ between every two cities i and j.
- Output: A tour visiting all cities with minimum total distance.
- A tour is a permutation of the cities, starting and ending in city 1.

Dynamic Programming for TSP (2)

- city i, non-empty subset of cities $S \subseteq \{2, 3, ..., n\}$
- $\textsc{Opt}[S; i] \equiv$ length of the shortest path starting in city 1, visits all cities in $S \setminus \{i\}$ and ends in i.
- Then,

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

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

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

Dynamic Programming for TSP (3)

Theorem 1 (Held & Karp '62)

TSP can be solved in time $\mathcal{O}(2^n n^2) = O^*(2^n)$.

- best known algo for TSP

Outline

Branch & Reduce

Branch & Reduce Algorithm

- Select a local configuration of the instance
- Determine all possible values this part can take
- Recursively solve smaller subproblems based on these values
- Return the best of these solutions

- 1 possible value: Reduction Rule (polynomial)
- >1 possible value: Branching Rule (exponential)

Branch & Reduce for MIS

MIS(G)

- If there is a vertex v of degree at most 1, return $\{v\} \cup \mathbf{MIS}(G - N[v])$
- Else if G contains $k > 1$ connected components $G_1, ..., G_k$, return $\bigcup_{i=1}^{k} \mathbf{MIS}(G_i)$
- Else if the maximum degree of G is ≤ 2, solve the problem in polynomial time
- Else Select a vertex v of maximum degree
  Return the largest set among
  - $\{\mathbf{MIS}(G - v),$
  - $\{v\} \cup \mathbf{MIS}(G - N[v])\}$

Standard Running time analysis

- The branching rule selects a vertex v of degree ≥ 3
- It considers the subproblems $\{\mathbf{MIS}(G - v), \{v\} \cup \mathbf{MIS}(G - N[v])\}$
- In the 1st branch, 1 vertex is deleted, in the 2nd branch ≥ 4
- $T(n)$ is the running time of the algo for a graph on n vertices
- $T(n) \leq T(n-1) + T(n-4)$
- $x^n \leq x^{n-1} + x^{n-4}$
- $x^4 - x^3 - 1 = 0$
- $x \approx 1.380277$
- Running time: $\mathcal{O}(1.3803^n)$

Measure & Conquer

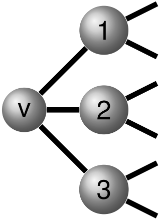
- Measure & Conquer: Technique to better analyze Branch & Reduce algorithms
- same algo, better running-time analysis
- instead of using n as a measure, use sth. more clever
- let's use Measure & Conquer to analyze our algorithm for MIS
- we consider an instance with many vertices of small degree as "easier"
- $\Rightarrow$ assign weights to the vertices according to their degree

Measure & Conquer (2)

- Measure: $\mu(G) = w_2 n_2 + w_3 n_3 + w_4 n_{\geq 4}$
- n_x is the number of vertices of degree x
- advantage when the degree of a vertex decreases
- $\Rightarrow w_2 \leq w_3 \leq w_4$
- We want $\mu(G) \leq n \Rightarrow w_4 = 1$
- To simplify the analysis, suppose $w_4 - w_3 \leq w_3 - w_2 \leq w_2$.
- I.e. (i) is more advantageous to (i+1)
  1. delete a vertex (of degree ≥ 2)
  2. decrease the degree of a vertex from 3 to 2
  3. decrease the degree of a vertex from 4 to 3

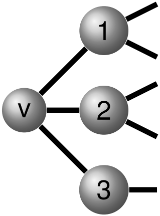
Measure & Conquer (3)

- Branch on a vertex of degree 3 with 3 neighbors of degree 3



$$T(\mu) \leq T(\mu - 4w_3) + T(\mu + 3w_2 - 4w_3)$$

- Branch on a vertex of degree 3 with 2 neighbors of degree 3

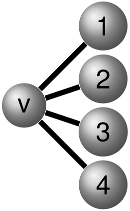


$$T(\mu) \leq T(\mu - w_2 - 3w_3) + T(\mu - 3w_3)$$

- ...

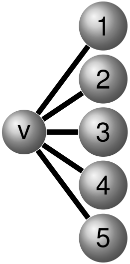
Measure & Conquer (4)

- Branch on a vertex of degree 4



$$T(\mu) \leq T(\mu - 4w_2 - w_4) + T(\mu + 4w_3 - 5w_4)$$

- Branch on a vertex of degree ≥ 5



$$T(\mu) \leq T(\mu - 5w_2 - w_4) + T(\mu - w_4)$$

Measure & Conquer (5)

- System of recurrences

$$T(\mu) \leq max \begin{cases} T(\mu - 4w_3) + T(\mu + 3w_2 - 4w_3) \\ T(\mu - w_2 - 3w_3) + T(\mu - 3w_3) \\ T(\mu - 2w_2 - 2w_3) + T(\mu - 3w_2 - 2w_3) \\ T(\mu - 3w_2 - w_3) + T(\mu - 6w_2 - w_3) \\ T(\mu - 4w_2 - w_4) + T(\mu + 4w_3 - 5w_4) \\ T(\mu - 5w_2 - w_4) + T(\mu - 4w_4) \end{cases}$$

- optimal values for w_2, w_3 found by local search or quasiconvex programming [Eppstein '04]
- $\Rightarrow w_2 = 0.7533, w_3 = 0.9262, w_4 = 1$
- Final running time: $\mathcal{O}(1.3360^n)$

Best Algorithms for MIS

- $\mathcal{O}(1.1889^n)$ [Robson '01] very complicated, computer-generated algorithm, exponential space
- $\mathcal{O}(1.2210^n)$ [Fomin, Grandoni, Kratsch '06] very simple algorithm, Measure & Conquer analysis, polynomial space

Outline

Speed-up by memorization

Memorization

For each subgraph of size $\leq \alpha n$, compute an optimal solution and store it in a DB
Add the following rule to the algorithm:

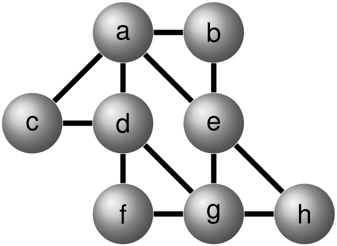
- If $|V| \leq \alpha n$, retrieve the solution from the DB

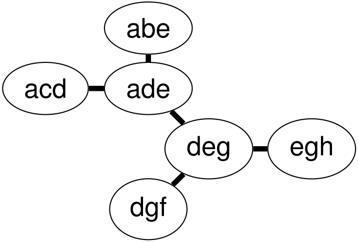
- Compute the optimal solution for small subgraphs takes time $\binom{n}{\alpha n}$ (using dynamic programming)
- The new rule ensures that branching does not occur if the graph has $\leq \alpha n$ vertices
- Running time: $\min_\alpha \max\{1.3803^{n-\alpha n}, \binom{n}{\alpha n}\} = 1.3424^n$ for $\alpha = 0.0865$

Outline

Treewidth, Tree Decomposition

- Treewidth (tw) measures how tree-like a graph is





- This graph has treewidth 2
- Trees have treewidth 1

Treewidth bound

Theorem 2 (Fomin, Gaspers, Saurabh, Stepanov)

For any $\epsilon > 0$, there exists an integer n_ϵ such that for every graph G with $n > n_\epsilon$ vertices,

$$pw(G) \leq \frac{1}{6}n_3 + \frac{1}{3}n_4 + \frac{13}{30}n_5 + \frac{23}{45}n_6 + n_{\geq 7} + \epsilon n$$

where n_x is the number of vertices of degree x in G.
Moreover, a path decomposition of the corresponding width can be constructed in polynomial time.

- $tw(G) \leq pw(G)$ for any graph G because every path decomposition of a graph *is* a tree decomposition

Treewidth Algorithm for MIS

- Given a graph G and a tree decomposition for G of width k,
- MIS can be solved in time $2^k n^{\mathcal{O}(1)}$
- (dynamic programming using the tree decomposition)
- For graphs of maximum degree 3:
  $\mathcal{O}^*(2^{n/6+\epsilon n}) = \mathcal{O}^*(1.1225^n)$
- For graphs of maximum degree 4:
  $\mathcal{O}^*(2^{n/3+\epsilon n}) = \mathcal{O}^*(1.2600^n)$

Outline

Treewidth/Branch & Reduce Algorithm for MIS

MIS(G)

- If there is a vertex v of degree at least 5,
  Return the largest set among
  - $\{\mathbf{MIS}(G - v),$
  - $\{v\} \cup \mathbf{MIS}(G - N[v])\}$
- Else (the maximum degree of G is ≤ 4)
  - compute a tree decomposition of G
  - solve the problem using this tree decomposition

- $T(n) \leq T(n-1) + T(n-6) \Rightarrow \mathcal{O}^*(1.2852^n)$
- Tree decomposition has width $\leq \frac{1}{3}n \Rightarrow \mathcal{O}^*(1.2600^n)$
- Total: $\mathcal{O}^*(1.2852^n)$

Outline

Iterative Compression

Core Idea

Inductive approach: Compute a solution for a problem instance using the information provided by a solution for a smaller instance.

Iterative Compression

- **Compression step:** Given a solution of size $k+1$, compress it to a solution of size k or prove that there is no solution of size k
- **Iteration step:** Incrementally build a solution to the given instance by deriving solutions for larger and larger subinstances

- Seen a lot of success in Parameterized Complexity
- Examples: best known fixed parameter algorithms for (DIRECTED) FEEDBACK VERTEX SET, EDGE BIPARTIZATION, ALMOST 2-SAT, ...

Iterative Compression

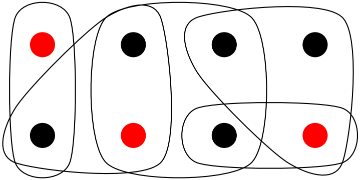
- **Compression step:** Given a solution of size $k+1$, compress it to a solution of size k or prove that there is no solution of size k
- **Iteration step:** Incrementally build a solution to the given instance by deriving solutions for larger and larger subinstances

- Seen a lot of success in Parameterized Complexity
- Examples: best known fixed parameter algorithms for (DIRECTED) FEEDBACK VERTEX SET, EDGE BIPARTIZATION, ALMOST 2-SAT, ...

k-Hitting Set

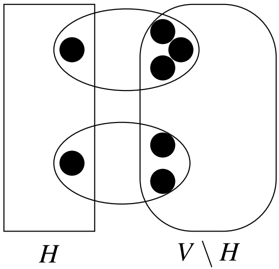
k-Hitting Set (k-HS)

- Input: $(U, \mathcal{S})$ where U is a universe U of n elements and $\mathcal{S}$ is a set of subsets of U such that for each $S \in \mathcal{S}$, $|S| \leq k$.
- Output: A hitting set of $(U, \mathcal{S})$ of minimum size.
- A hitting set of $(U, \mathcal{S})$ is set of elements $H \subseteq U$ such that for each $S \in \mathcal{S}$, $S \cap H \neq \emptyset$.



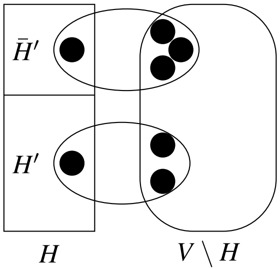
Minimum 4-Hitting Set: Compression Step

COMP-4HS: Given a MINIMUM 4-HITTING SET instance $(V, \mathcal{C})$ and a hitting set $H \subseteq V$ of $\mathcal{C}$ such that every hitting set of $\mathcal{C}$ has size at least $|H| - 1$, find a hitting set H^* of size $|H| - 1$ if one exists.



Minimum 4-Hitting Set: Compression Step

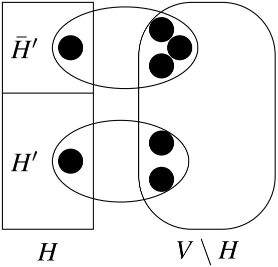
COMP-4HS: Given a MINIMUM 4-HITTING SET instance $(V, \mathcal{C})$ and a hitting set $H \subseteq V$ of $\mathcal{C}$ such that every hitting set of $\mathcal{C}$ has size at least $|H| - 1$, find a hitting set H^* of size $|H| - 1$ if one exists.



Go over all partitions $(H', \bar{H}')$ of H such that $|H'| \geq 2|H| - n - 1$

Minimum 4-Hitting Set: Compression Step

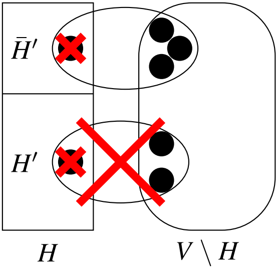
COMP-4HS: Given a MINIMUM 4-HITTING SET instance $(V, \mathcal{C})$ and a hitting set $H \subseteq V$ of $\mathcal{C}$ such that every hitting set of $\mathcal{C}$ has size at least $|H| - 1$, find a hitting set H^* of size $|H| - 1$ if one exists.



Reject a partition if there is a $C_i \in \mathcal{C}$ such that $C_i \subseteq \bar{H}'$

Minimum 4-Hitting Set: Compression Step

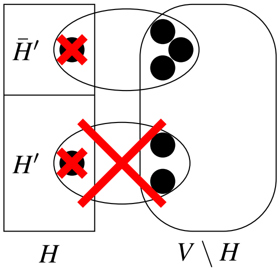
COMP-4HS: Given a MINIMUM 4-HITTING SET instance $(V, \mathcal{C})$ and a hitting set $H \subseteq V$ of $\mathcal{C}$ such that every hitting set of $\mathcal{C}$ has size at least $|H| - 1$, find a hitting set H^* of size $|H| - 1$ if one exists.



Compute a minimum hitting set H'' for $(V', \mathcal{C}')$ where $V' = V \setminus H$ and $\mathcal{C}' = \{C_i \cap V \mid C_i \in \mathcal{C} \wedge C_i \cap H' = \emptyset\}$

Minimum 4-Hitting Set: Compression Step

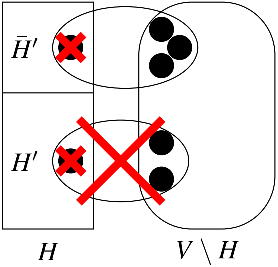
COMP-4HS: Given a MINIMUM 4-HITTING SET instance $(V, \mathcal{C})$ and a hitting set $H \subseteq V$ of $\mathcal{C}$ such that every hitting set of $\mathcal{C}$ has size at least $|H| - 1$, find a hitting set H^* of size $|H| - 1$ if one exists.



$H^* = H' \cup H''$

Minimum 4-Hitting Set: Compression Step

COMP-4HS: Given a MINIMUM 4-HITTING SET instance $(V, \mathcal{C})$ and a hitting set $H \subseteq V$ of $\mathcal{C}$ such that every hitting set of $\mathcal{C}$ has size at least $|H| - 1$, find a hitting set H^* of size $|H| - 1$ if one exists.



If $|H^*| \leq |H| - 1$ then return H^*

Minimum 4-Hitting Set: Compression Step (2)

- Algo considers only partitions into $(H', \bar{H}')$ such that $|H'| \geq 2|H| - n - 1$.
  Nb. of partitions $\leq$

$$\mathcal{O}\left(\max\left\{2^{2n/3}, \max_{2n/3 \leq j \leq n} \binom{j}{2j-n}\right\}\right) = \mathcal{O}\left(\max_{2n/3 \leq j \leq n} \binom{j}{2j-n}\right)$$

- The subinstances $(V', \mathcal{C}')$ where $V' = V \setminus H$ and $\mathcal{C}' = \{C_i \cap V \mid C_i \in \mathcal{C} \wedge C_i \cap H' = \emptyset\}$ are instances of MINIMUM 3-HITTING SET and we use a $\mathcal{O}(1.6278^n)$ algorithm [Wahlström '07] to solve them
- Total running time:[2]

$$\mathcal{O}\left(\max_{2n/3 \leq j \leq n} \binom{j}{2j-n} 1.6278^{n-j}\right) = \mathcal{O}(1.8704^n)$$

[2] maximum obtained for $j \approx 0.6824 \cdot n$

Minimum 4-Hitting Set: Compression Step (2)

- Algo considers only partitions into $(H', \bar{H}')$ such that $|H'| \geq 2|H| - n - 1$.
  Nb. of partitions $\leq$

$$\mathcal{O}\left(\max\left\{2^{2n/3}, \max_{2n/3 \leq j \leq n} \binom{j}{2j-n}\right\}\right) = \mathcal{O}\left(\max_{2n/3 \leq j \leq n} \binom{j}{2j-n}\right)$$

- The subinstances $(V', \mathcal{C}')$ where $V' = V \setminus H$ and $\mathcal{C}' = \{C_i \cap V \mid C_i \in \mathcal{C} \wedge C_i \cap H' = \emptyset\}$ are instances of MINIMUM 3-HITTING SET and we use a $\mathcal{O}(1.6278^n)$ algorithm [Wahlström '07] to solve them
- Total running time:[2]

$$\mathcal{O}\left(\max_{2n/3 \leq j \leq n} \binom{j}{2j-n} 1.6278^{n-j}\right) = \mathcal{O}(1.8704^n)$$

[2] maximum obtained for $j \approx 0.6824 \cdot n$

Minimum 4-Hitting Set: Compression Step (2)

- Algo considers only partitions into $(H', \bar{H}')$ such that $|H'| \geq 2|H| - n - 1$.
  Nb. of partitions $\leq$

$$\mathcal{O}\left(\max\left\{2^{2n/3}, \max_{2n/3 \leq j \leq n} \binom{j}{2j-n}\right\}\right) = \mathcal{O}\left(\max_{2n/3 \leq j \leq n} \binom{j}{2j-n}\right)$$

- The subinstances $(V', \mathcal{C}')$ where $V' = V \setminus H$ and $\mathcal{C}' = \{C_i \cap V \mid C_i \in \mathcal{C} \wedge C_i \cap H' = \emptyset\}$ are instances of MINIMUM 3-HITTING SET and we use a $\mathcal{O}(1.6278^n)$ algorithm [Wahlström '07] to solve them
- Total running time:[2]

$$\mathcal{O}\left(\max_{2n/3 \leq j \leq n} \binom{j}{2j-n} 1.6278^{n-j}\right) = \mathcal{O}(1.8704^n)$$

[2] maximum obtained for $j \approx 0.6824 \cdot n$

Minimum 4-Hitting Set: Iteration Step

- $(V, \mathcal{C})$ instance of MINIMUM 4-HITTING SET with $V = \{v_1, v_2, \ldots, v_n\}$
- $V_i = \{v_1, v_2, \ldots, v_i\}$ for $i = 1$ to n
- $\mathcal{C}_i = \{C_j \in \mathcal{C} \mid C_j \subseteq V_i\}$
- Note that $|H_{i-1}| \leq |H_i| \leq |H_{i-1}| + 1$ where H_j is a minimum hitting set of instance $(V_i, \mathcal{C}_i)$

Minimum 4-Hitting Set: Iteration Step

- $(V, \mathcal{C})$ instance of MINIMUM 4-HITTING SET with $V = \{v_1, v_2, \dots, v_n\}$
- $V_i = \{v_1, v_2, \dots, v_i\}$ for $i = 1$ to n
- $\mathcal{C}_i = \{C_j \in \mathcal{C} \mid C_j \subseteq V_i\}$
- Note that $|H_{i-1}| \leq |H_i| \leq |H_{i-1}| + 1$ where H_j is a minimum hitting set of instance $(V_i, \mathcal{C}_i)$

Minimum 4-Hitting Set

Theorem 3

MINIMUM 4-HITTING SET *can be solved in time* $\mathcal{O}(1.8704^n)$.

- Can be generalized to the counting version of MINIMUM k-HITTING SET for any fixed k

Minimum 4-Hitting Set

Theorem 3

MINIMUM 4-HITTING SET *can be solved in time* $\mathcal{O}(1.8704^n)$.

- Can be generalized to the counting version of MINIMUM k-HITTING SET for any fixed k

Outline

The Principle of Inclusion-Exclusion

- Let $V_1, V_2, \ldots, V_m$ be finite sets
- Then,

$$\left|\bigcup_{i=1}^{m} V_i\right| = \sum_{i=1}^{m} |V_i| - \sum_{1 \le i < j \le m} |V_i \cap V_j| + \sum_{1 \le i < j < k \le m} |V_i \cap V_j \cap V_k| - \ldots$$

- Such a formula together with dynamic programming: best algorithm for COLORING

Inclusion-Exclusion for COLORING

Lemma 4 (Bjørkund, Husfeldt '06)

A graph $G = (V, E)$ is k-colorable iff

$$c_k(G) = \sum_{X \subseteq V} (-1)^{|X|} s(X)^k > 0$$

where $s(X)$ = number of independent sets not intersecting X.

Proof.

- $c_k(G)$ = nb. of ways to cover V with k i.s. (possibly overlapping)
- $s(X)^k$ = nb. of ways to choose k i.s. not intersecting X
- a set of k i.s. covering V is counted only in $s(\emptyset)$
- a set of k i.s. not covering V avoids some vertices U
  - hence counted once in every $s(W)$ for every $W \subseteq U$
  - every non-empty set has as many even- as odd-sized subsets

$\square$

Inclusion-Exclusion for COLORING (2)

- Dynamic programming to compute $s(X)$ (number of independent sets not intersecting X)
- $s(X) = s(X \cup \{v\}) + s(X \cup N[v]) + 1, v \in V \setminus X$
- all $s(X)$ computed in time $\mathcal{O}^*(2^n)$
- now, $c_k(G) = \sum_{X \subseteq V} (-1)^{|X|} s(X)^k$ can easily be computed
- to obtain the least k for which $c_k(G) > 0$, use binary search

Theorem 5 (Bjørkund, Husfeldt '06 & Koivisto '06)

COLORING *can be solved in time* $\mathcal{O}^*(2^n)$.

Conclusion

- We have seen some of the most important techniques in the design and analysis of exponential time algorithms
- Other techniques: Preprocessing Data, Local Search, Problem-Reduction, Combination of Techniques, Combination of Measures
- Also useful: Lower Bounds (especially for Branch & Reduce Algorithms)
- Classification among problems
- Properties of problems
- Q: big-Oh appropriate?
- Q: exponential space practical?

Thank you!

Questions?

Comments?