FEDOR V. FOMIN

Kernelization Algorithms

LNMB conference, Luntheren

15.01.2013

Exact solutions for NP-hard problems

Early interest in exponential algorithms in the late 70's and early 80's (following fundamental results on NP-completeness)

Emerging interest in fast exponential algorithms for NP-hard (and other hard) problems in the last years

Why is it interesting?

(I) *Approaches to attack hard computational problems*

- approximation algorithms
- randomized algorithms
- fixed parameter algorithms
- heuristics
- restricting the inputs

have weak points

- necessity of exact solutions
- difficulty of approximation
- limited power of the method itself

Why is it interesting?

(I) *Approaches to attack hard computational problems*

- approximation algorithms
- randomized algorithms
- fixed parameter algorithms
- heuristics
- restricting the inputs

have weak points

- necessity of exact solutions
- difficulty of approximation
- limited power of the method itself

Why is it interesting?

(II) *Curiosity*

- If we believe that $P \neq NP$, are all NP complete problems on the same level of intractability? Or maybe some of them can be solved faster than the others? If yes, then why?
- Is the exhaustive search (trying all possible solutions) the only alternative for solving NP complete problems?

Why is it interesting?

(III) *Are exponential algorithms really bad?*

- What is better, algorithm with exponential running time $\mathcal{O}(1.01^n)$ or algorithm with polynomial running time $\mathcal{O}(n^4)$?

For $n = 5000$, $1.01^n < n^4$.

Why is it interesting?

(III) *Are exponential algorithms really bad?*

- What is better, algorithm with exponential running time $\mathcal{O}(1.01^n)$ or algorithm with polynomial running time $\mathcal{O}(n^4)$?

For $n = 5000$, $1.01^n < n^4$.

Why is it interesting?

(IV) *Fun*

- Often there is a nice combinatorics

Old exponential algorithms

TSP: $\mathcal{O}^*(2^n)$	Held, Karp (1962)
COLORING: $\mathcal{O}(2.4422^n)$	Lawler (1976)
3-COLORING: $\mathcal{O}(1.4422^n)$	Lawler (1976)
3-SAT: $\mathcal{O}(1.6181^n)$	Speckenmeyer & Monien (1985)
INDEPEND. SET: $\mathcal{O}(1.2599^n)$	Tarjan & Trojanowski (1977)

Some new exponential algorithms

3-SAT: $\mathcal{O}(1.465^n)$	Scheder (2008)
COLORING: $\mathcal{O}^*(2^n)$	Bjorklund & Husfeldt, Koivisto (2006)
TREEWIDTH: $\mathcal{O}(1.7348^n)$	Fomin & Villanger (2010)
3-COLORING: $\mathcal{O}(1.3289^n)$	Beigel & Eppstein (2005)
INDEPEND. SET: $\mathcal{O}(1.1893^n)$	Robson (2001)
BANDWIDTH: $\mathcal{O}^*(4.383^n)$	Cygan & Pilipczuk (2010)
HAMILTONICITY: $O^*(1.657^n)$	Björklund (2010)

Basic question in EA:

Is there an algorithm significantly faster than a trivial (brute-force) one?

Exhaustive search: What is a trivial solution for NP complete problem?

- Subset problems:
  - SAT for CNF with n variables: $\mathcal{O}^*(2^n)$
  - Maximum Independent Set in a graph on n vertices : $\mathcal{O}^*(2^n)$
- Permutation problems:
  - TSP on n cities $\mathcal{O}^*(n!)$
- Partitioning problems:
  - Graph coloring $\mathcal{O}^*(2^{n \log n})$

Exhaustive search: What is a trivial solution for NP complete problem?

- Subset problems:
  - SAT for CNF with n variables: $\mathcal{O}^*(2^n)$
  - Maximum Independent Set in a graph on n vertices : $\mathcal{O}^*(2^n)$
- Permutation problems:
  - TSP on n cities $\mathcal{O}^*(n!)$
- Partitioning problems:
  - Graph coloring $\mathcal{O}^*(2^{n \log n})$

Exhaustive search: What is a trivial solution for NP complete problem?

- Subset problems:
  - SAT for CNF with n variables: $\mathcal{O}^*(2^n)$
  - Maximum Independent Set in a graph on n vertices : $\mathcal{O}^*(2^n)$
- Permutation problems:
  - TSP on n cities $\mathcal{O}^*(n!)$
- Partitioning problems:
  - Graph coloring $\mathcal{O}^*(2^{n \log n})$

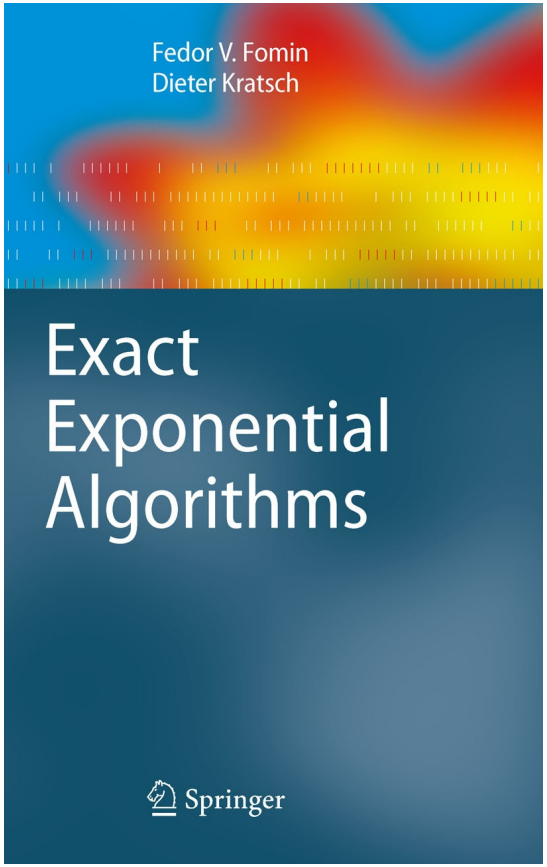
Exhaustive search: What is a trivial solution for NP complete problem?

- Subset problems:
  - SAT for CNF with n variables: $\mathcal{O}^*(2^n)$
  - Maximum Independent Set in a graph on n vertices : $\mathcal{O}^*(2^n)$
- Permutation problems:
  - TSP on n cities $\mathcal{O}^*(n!)$
- Partitioning problems:
  - Graph coloring $\mathcal{O}^*(2^{n \log n})$

Exhaustive search: What is a trivial solution for NP complete problem?

- Subset problems:
  - SAT for CNF with n variables: $\mathcal{O}^*(2^n)$
  - Maximum Independent Set in a graph on n vertices : $\mathcal{O}^*(2^n)$
- Permutation problems:
  - TSP on n cities $\mathcal{O}^*(n!)$
- Partitioning problems:
  - Graph coloring $\mathcal{O}^*(2^{n \log n})$

Beating the brute-force

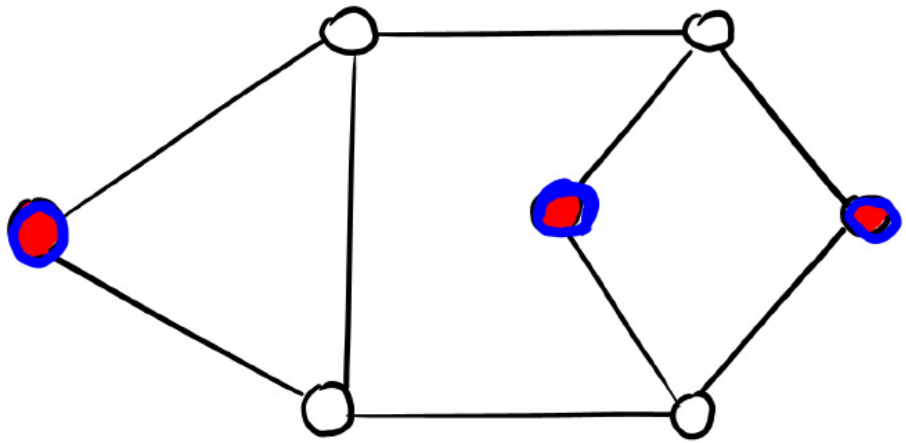


Many different algorithmic techniques and approaches.

Branching

Example:

Maximum Independent Set



Trivial algorithm to find a maximum independent set is: try all subsets, for each try check if it is independent set and count how many vertices are in this set. Output the maximum number. Runs in time $\mathcal{O}^*(2^n)$.

Example: Branching for IS

If all vertices of a graph are of degree 2 or less, the problem is trivial. Let v be a vertex of maximum degree (≥ 3).

If v belongs to an optimal independent set I, then none of its neighbors can be in I. Thus we can branch on two subproblems of smaller size: $G \setminus \{v\}$ and $G \setminus N[v]$.

Example: Branching for IS

The running time of the algorithm (up to a polynomial factor) is proportional to the number of leaves $t(n)$ in the branching tree.

$$t(n) \leq t(n-1) + t(n-4),$$

and $t(n) = 1$.

Better analysis

Number of leaves

$$t(n) = t(n-1) + t(n-4)$$

There is a standard technique for bounding such functions asymptotically.

We prove by induction that $t(n) \leq x^n$ for some $x > 1$ as small as possible.

What values of x are good? We need:

$$x^n \geq x^{n-1} + x^{n-4}$$

$$x^4 - x^3 - 1 \geq 0$$

Hence

$$t(n) \leq \alpha^n,$$

where $\alpha < 1.3802$ is the unique positive root of

$$1 = \frac{1}{x} + \frac{1}{x^4}.$$

Note: it is always true that such an equation has a unique positive

Better analysis

Is this bound tight? There are two questions:

- Can the function $t(k)$ be that large?
  Yes (ignoring rounding problems).
- Can the search tree of the Vertex Cover algorithm be that large?
  Difficult question, hard to answer in general.

Branch And Bound—Why Does It Work?

DECEMBER 19, 2012

by Pip

tags: Algorithms, branch-and-bound, chess, complexity, George Nemhauser, lower bounds, Proofs, Rubik's Cube

One of the mysteries of computational theory

source—our congratulations

George Nemhauser is one of the world experts on all things having to do with large-scale optimization problems. He has received countless honors for his brilliant work, including membership in the National Academy of Engineering, the John Von Neumann Theory Prize, the Khachiyan Prize, and the Lanchester Prize. He has been a faculty member at Georgia Tech for many years in the ISyE school, which has been one of the top industrial engineering schools in the world for years—these events are certainly correlated.

Today I wish to talk about a class of algorithms that are used all the time in practice, that George has used in his research for decades, but that have eluded our making any definite statements about their theoretical performance.

Recently at Tech there was a lunch meeting with some of the theory faculty and some of the ISyE faculty, including George and also Arkadi Nemirovski. The discussion was about a joint project that is in progress at Tech, which is a whole other story.

During the conversation I was asked what I thought were some of the biggest open questions in optimization. This is not my area of expertise—but that never stops me from having an opinion. I pointed out several questions that I thought were quite important, and then added that perhaps the biggest mystery to me was why we have such difficulty in proving anything about sequential algorithms like branch-and-bound.

Nemhauser smiled and said,

SUBSCRIBE TO GÖDEL'S LOST LETTER

type and press enter

RECENT POSTS

- Predictions and Principles
- The Year That Was
- Scientific Gifts
- What Would Be Left, If...?
- Branch And Bound—Why Does It Work?
- When Less Is More
- Mounting or Solving Open Problems
- The Amazing Zeta Code
- Barriers to P=NP Proofs
- Thanks For Sharing
- A Wonderful Riff On Rank
- Progress On Progressive Algorithms
- The Ryan Williams Combine
- The Power Of Guessing
- The Election Outcome

TOP POSTS

Independent Set: How to improve?

More cases... (from Tarjan & Trojanowski, SICOMP 1977)

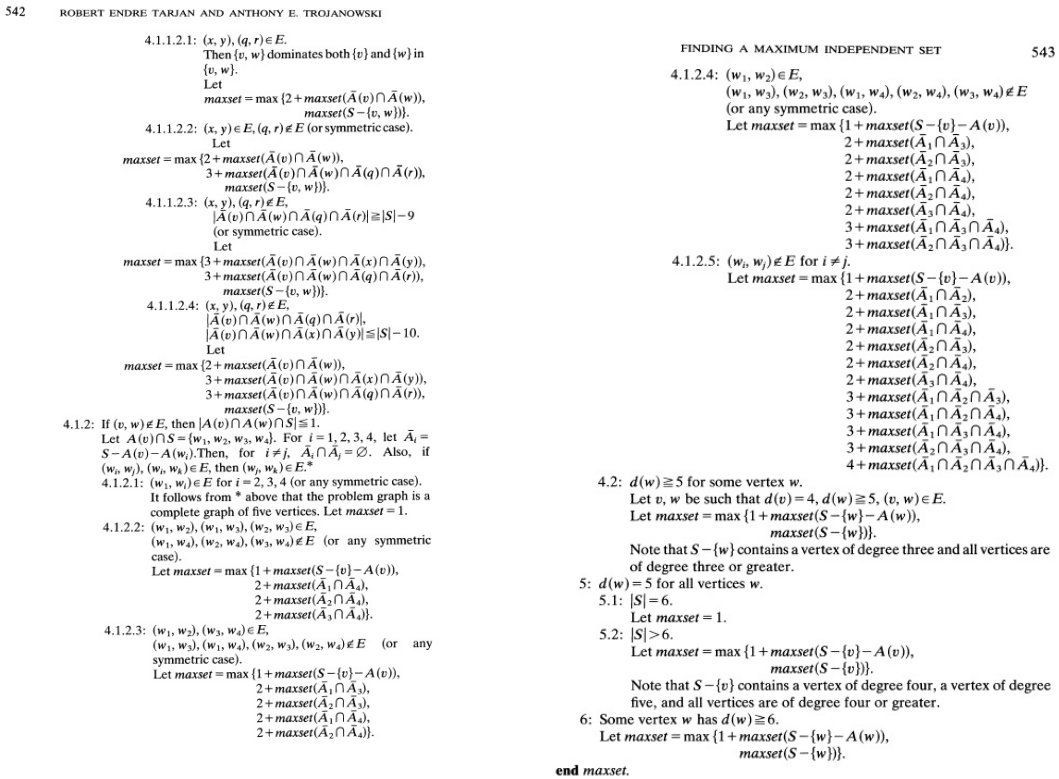
procedure *maxset*(S);
begin
0: S is not connected in $G(S)$.
Let $S_1, S_2, \cdots, S_k$ be the connected components of $G(S)$. Note that every maximum independent set consists of a union of maximum independent sets, one from each connected component. Let *maxset* $= \sum_{i=1}^{k}$ *maxset*(S_i).
not 0: S is connected.
Let v be a vertex of minimum degree in $G(S)$. One of the following six cases applies.
1: $d(v) = 1$.
Let $A(v) \cap S = \{w\}$.
Let *maxset* $= 1 +$ *maxset*$(S - \{v, w\})$.
2: $d(v) = 2$.
2.1: $d(w) = 2$ for all $w \in V$.
Note that the vertices of S form a cycle in $G(S)$.
Let *maxset* $= \lfloor |S|/2 \rfloor$.
2.2: There exist u, w_1 such that $d(u) = 2$, $d(w_1) \geq 3$, and $(u, w_1) \in E$.
Let $A(u) \cap S = \{w_1, w_2\}$.
2.2.1: $(w_1, w_2) \in E$.
Let *maxset* $= 1 +$ *maxset*$(S - \{u, w_1, w_2\})$.
2.2.2: $(w_1, w_2) \notin E$.
Let *maxset* $= \max\{1 +$ *maxset*$(S - \{u, w_1, w_2\}),$
$2 +$ *maxset*$(S - A(w_1) - A(w_2))\}$.
3: $d(v) = 3$.
Let $A(v) \cap S = \{w_1, w_2, w_3\}$.
3.1: $(w_1, w_2), (w_1, w_3), (w_2, w_3) \in E$.
Let *maxset* $= 1 +$ *maxset*$(S - \{v, w_1, w_2, w_3\})$.
3.2: $(w_1, w_2), (w_1, w_3) \in E$ (or any symmetric case).
Let *maxset* $= \max\{1 +$ *maxset*$(S - \{v, w_1, w_2, w_3\}),$
$2 +$ *maxset*$(S - A(w_2) - A(w_3))\}$.
3.3: $(w_1, w_2) \in E$ (or any symmetric case).
For $i = 1, 2, 3$, let $\bar{A}_i = S - \{w_1, w_2, w_3\} - A(w_i)$.

Note that $|\bar{A}_1|, |\bar{A}_2| \leq |S| - 5$, $|\bar{A}_3| \leq |S| - 6$.
3.3.1: $|\bar{A}_1 \cap \bar{A}_3| \leq |\bar{A}_2 \cap \bar{A}_3| = |S| - 6$ (or the symmetric case).
Note that $\bar{A}_2 \cap \bar{A}_3 = \bar{A}_3$. Thus $\{w_2, w_3\}$ dominates $\{w_1, w_3\}$.
Let *maxset* $= \max\{1 +$ *maxset*$(S - \{v, w_1, w_2, w_3\}),$
$2 +$ *maxset*$(\bar{A}_3)\}$.
3.3.2: $|\bar{A}_1 \cap \bar{A}_3|, |\bar{A}_2 \cap \bar{A}_3| \leq |S| - 7$.
Let *maxset* $= \max\{1 +$ *maxset*$(S - \{v, w_1, w_2, w_3\}),$
$2 +$ *maxset*$(\bar{A}_1 \cap \bar{A}_3),$
$2 +$ *maxset*$(\bar{A}_2 \cap \bar{A}_3)\}$.
3.4: $(w_i, w_j) \notin E$ for $i, j \in \{1, 2, 3\}$.
For $i = 1, 2, 3$, let $\bar{A}_i = S - \{w_1, w_2, w_3\} - A(w_i)$.
Note that $|\bar{A}_i| \leq |S| - 6$ for $i = 1, 2, 3$.
3.4.1: $|\bar{A}_1 \cap \bar{A}_2 \cap \bar{A}_3| \geq |S| - 7$.
Set $\{w_1, w_2, w_3\}$ dominates $\{w_i, w_j\}$ for $i, j \in \{1, 2, 3\}$.
Let *maxset* $= \max\{1 +$ *maxset*$(S - \{v, w_1, w_2, w_3\}),$
$3 +$ *maxset*$(\bar{A}_1 \cap \bar{A}_2 \cap \bar{A}_3)\}$.
3.4.2: $|\bar{A}_1 \cap \bar{A}_2 \cap \bar{A}_3| = |S| - 8$ or $|S| - 9$.
If, for some i, j, $|\bar{A}_i \cap \bar{A}_j| \geq |\bar{A}_1 \cap \bar{A}_2 \cap \bar{A}_3| + 1$, then $\{v_i, w_j\}$ is dominated by $\{w_1, w_2, w_3\}$.
For distinct i, j, k,
$|\bar{A}_i| \geq |\bar{A}_1 \cap \bar{A}_2 \cap \bar{A}_3| + |\bar{A}_j \cap \bar{A}_j - (\bar{A}_1 \cap \bar{A}_2 \cap \bar{A}_3)|$
$+ |\bar{A}_k \cap \bar{A}_k - (\bar{A}_1 \cap \bar{A}_2 \cap \bar{A}_3)|$,
$|\bar{A}_i| \leq |S| - 6$, and $|\bar{A}_1 \cap \bar{A}_2 \cap \bar{A}_3| \geq |S| - 9$.
Thus $|\bar{A}_i \cap \bar{A}_j| \geq |\bar{A}_1 \cap \bar{A}_2 \cap \bar{A}_3| + 2$ for only one possible pair $i \neq j$. Let 1, 2 be the pair (if any).
3.4.2.1: $|\bar{A}_i \cap \bar{A}_j| \leq |\bar{A}_1 \cap \bar{A}_2 \cap \bar{A}_3| + 1$ for all $i \neq j$.
Let *maxset* $= \max\{1 +$ *maxset*$(S - \{v, w_1, w_2, w_3\}),$
$3 +$ *maxset*$(\bar{A}_1 \cap \bar{A}_2 \cap \bar{A}_3)\}$.
3.4.2.2: $|\bar{A}_1 \cap \bar{A}_2| \geq |\bar{A}_1 \cap \bar{A}_2 \cap \bar{A}_3| + 2$ (or any symmetric case).
Let *maxset* $= \max\{1 +$ *maxset*$(S - \{v, w_1, w_2, w_3\}),$
$2 +$ *maxset*$(\bar{A}_1 \cap \bar{A}_2),$
$3 +$ *maxset*$(\bar{A}_1 \cap \bar{A}_2 \cap \bar{A}_3)\}$.
3.4.3: $|\bar{A}_1 \cap \bar{A}_2 \cap \bar{A}_3| \leq |S| - 10$.
3.4.3.1: $|\bar{A}_i \cap \bar{A}_j| \leq |\bar{A}_1 \cap \bar{A}_2 \cap \bar{A}_3| + 1$ for all $i \neq j$.
Same as 3.4.2.1.
3.4.3.2: $|\bar{A}_1 \cap \bar{A}_2| \geq |\bar{A}_1 \cap \bar{A}_2 \cap \bar{A}_3| + 2$ (or any symmetric case).
Same as 3.4.2.2.
3.4.3.3: $|\bar{A}_1 \cap \bar{A}_2|, |\bar{A}_1 \cap \bar{A}_3| \geq |\bar{A}_1 \cap \bar{A}_2 \cap \bar{A}_3| + 2$ (or any symmetric case).
Let *maxset* $= \max\{1 +$ *maxset*$(S - \{v, w_1, w_2, w_3\}),$
$2 +$ *maxset*$(\bar{A}_1 \cap \bar{A}_2),$
$2 +$ *maxset*$(\bar{A}_1 \cap \bar{A}_3),$
$3 +$ *maxset*$(\bar{A}_1 \cap \bar{A}_2 \cap \bar{A}_3)\}$.
3.4.3.4: $|\bar{A}_1 \cap \bar{A}_2|, |\bar{A}_1 \cap \bar{A}_3|, |\bar{A}_2 \cap \bar{A}_3| \geq |\bar{A}_1 \cap \bar{A}_2 \cap \bar{A}_3| + 2$.
For $i = 1, 2, 3$, let $u_{i1}, u_{i2} \in (\bar{A}_i \cap \bar{A}_k) - \bar{A}_i$ $(j, k \neq i)$.

3.4.3.4.1: $|\bar{A}_j \cap \bar{A}_k| = |\bar{A}_1 \cap \bar{A}_2 \cap \bar{A}_3| + 2$ and $(u_{i1}, u_{i2}) \in E$ for some distinct i, j, k.
Then $\{w_1, w_2, w_3\}$ dominates $\{w_i, w_k\}$.
Same as 3.4.3.3.
3.4.3.4.2: $|\bar{A}_j \cap \bar{A}_k| = |\bar{A}_1 \cap \bar{A}_2 \cap \bar{A}_3| + 2$ and $(u_{i1}, u_{i2}) \notin E$ for all distinct i, j, k. Let
maxset $= \max\{1 +$ *maxset*$(S - \{v, w_1, w_2, w_3\}),$
$4 +$ *maxset*$(\bar{A}_1 \cap \bar{A}_2 \cap \bar{A}_3 - A(u_{11}) - A(u_{12})),$
$4 +$ *maxset*$(\bar{A}_1 \cap \bar{A}_2 \cap \bar{A}_3 - A(u_{21}) - A(u_{22})),$
$4 +$ *maxset*$(\bar{A}_1 \cap \bar{A}_2 \cap \bar{A}_3 - A(u_{31}) - A(u_{32})),$
$3 +$ *maxset*$(\bar{A}_1 \cap \bar{A}_2 \cap \bar{A}_3)\}$.
3.4.3.4.3: $|\bar{A}_1 \cap \bar{A}_2|, |\bar{A}_1 \cap \bar{A}_3| = |\bar{A}_1 \cap \bar{A}_2 \cap \bar{A}_3| + 2$ (or any symmetric case).
Let
maxset $= \max\{1 +$ *maxset*$(S - \{v, w_1, w_2, w_3\}),$
$4 +$ *maxset*$(\bar{A}_1 \cap \bar{A}_2 \cap \bar{A}_3 - A(u_{31}) - A(u_{32})),$
$4 +$ *maxset*$(\bar{A}_1 \cap \bar{A}_2 \cap \bar{A}_3 - A(u_{21}) - A(u_{22})),$
$2 +$ *maxset*$(\bar{A}_2 \cap \bar{A}_3),$
$3 +$ *maxset*$(\bar{A}_1 \cap \bar{A}_2 \cap \bar{A}_3)\}$.
3.4.3.4.4: $|\bar{A}_1 \cap \bar{A}_2| = |\bar{A}_1 \cap \bar{A}_2 \cap \bar{A}_3| + 2$ (or any symmetric case).
Let
maxset $= \max\{1 +$ *maxset*$(S - \{v, w_1, w_2, w_3\}),$
$4 +$ *maxset*$(\bar{A}_1 \cap \bar{A}_2 \cap \bar{A}_3 - A(u_{31}) - A(u_{32})),$
$2 +$ *maxset*$(\bar{A}_1 \cap \bar{A}_3),$
$2 +$ *maxset*$(\bar{A}_2 \cap \bar{A}_3),$
$3 +$ *maxset*$(\bar{A}_1 \cap \bar{A}_2 \cap \bar{A}_3)\}$.
3.4.3.4.5: $|\bar{A}_i \cap \bar{A}_j| \geq |\bar{A}_1 \cap \bar{A}_2 \cap \bar{A}_3| + 3$ for $i \neq j$.
Let
maxset $= \max\{1 +$ *maxset*$(S - \{v, w_1, w_2, w_3\}),$
$2 +$ *maxset*$(\bar{A}_1 \cap \bar{A}_2),$
$2 +$ *maxset*$(\bar{A}_1 \cap \bar{A}_3),$
$2 +$ *maxset*$(\bar{A}_2 \cap \bar{A}_3),$
$3 +$ *maxset*$(\bar{A}_1 \cap \bar{A}_2 \cap \bar{A}_3)\}$.
4: $d(v) = 4$.
4.1: $d(w) = 4$ for all vertices w.
4.1.1: There are vertices v, w such that $(v, w) \notin E$ and $|A(v) \cap A(w) \cap S| \geq 2$.
4.1.1.1: $|A(v) \cap A(w) \cap S| \geq 3$.
Then $\{v, w\}$ dominates both $\{v\}$ and $\{w\}$ in $\{v, w\}$.
Let
maxset $= \max\{2 +$ *maxset*$(S - \{v, w\} - A(v) - A(w)),$
maxset$(S - \{v, w\})\}$.
4.1.1.2: $|A(v) \cap A(w) \cap S| = 2$.
Let $x, y \in (A(v) - A(w)) \cap S$, q, $q, r \in (A(w) - A(v)) \cap S$.
Let $\bar{A}(z) = S - \{z\} - A(z)$ for $z \in S$.

Independent Set: How to improve?

More cases (...continued) $\Rightarrow \mathcal{O}(1.2599^n)$



542 ROBERT ENDRE TARJAN AND ANTHONY E. TROJANOWSKI

4.1.1.2.1: $(x, y), (q, r) \in E$.
Then $\{v, w\}$ dominates both $\{v\}$ and $\{w\}$ in $\{v, w\}$.
Let
$maxset = \max\{2 + maxset(\bar{A}(v) \cap \bar{A}(w)), maxset(S - \{v, w\})\}$.

4.1.1.2.2: $(x, y) \in E, (q, r) \notin E$ (or symmetric case).
Let
$maxset = \max\{2 + maxset(\bar{A}(v) \cap \bar{A}(w)), 3 + maxset(\bar{A}(v) \cap \bar{A}(w) \cap \bar{A}(q) \cap \bar{A}(r)), maxset(S - \{v, w\})\}$.

4.1.1.2.3: $(x, y), (q, r) \notin E$, $|\bar{A}(v) \cap \bar{A}(w) \cap \bar{A}(q) \cap \bar{A}(r)| \geqq |S| - 9$ (or symmetric case).
Let
$maxset = \max\{3 + maxset(\bar{A}(v) \cap \bar{A}(w) \cap \bar{A}(x) \cap \bar{A}(y)), 3 + maxset(\bar{A}(v) \cap \bar{A}(w) \cap \bar{A}(q) \cap \bar{A}(r)), maxset(S - \{v, w\})\}$.

4.1.1.2.4: $(x, y), (q, r) \notin E$, $|\bar{A}(v) \cap \bar{A}(w) \cap \bar{A}(q) \cap \bar{A}(r)|$, $|\bar{A}(v) \cap \bar{A}(w) \cap \bar{A}(x) \cap \bar{A}(y)| \leqq |S| - 10$.
Let
$maxset = \max\{2 + maxset(\bar{A}(v) \cap \bar{A}(w)), 3 + maxset(\bar{A}(v) \cap \bar{A}(w) \cap \bar{A}(x) \cap \bar{A}(y)), 3 + maxset(\bar{A}(v) \cap \bar{A}(w) \cap \bar{A}(q) \cap \bar{A}(r)), maxset(S - \{v, w\})\}$.

4.1.2: If $(v, w) \notin E$, then $|A(v) \cap A(w) \cap S| \leqq 1$.
Let $A(v) \cap S = \{w_1, w_2, w_3, w_4\}$. For $i = 1, 2, 3, 4$, let $\bar{A}_i = S - A(v) - A(w_i)$. Then, for $i \neq j$, $\bar{A}_i \cap \bar{A}_j = \emptyset$. Also, if $(w_i, w_j), (w_i, w_k) \in E$, then $(w_j, w_k) \in E$.*

4.1.2.1: $(w_1, w_i) \in E$ for $i = 2, 3, 4$ (or any symmetric case).
It follows from * above that the problem graph is a complete graph of five vertices. Let $maxset = 1$.

4.1.2.2: $(w_1, w_2), (w_1, w_3), (w_2, w_3) \in E$, $(w_1, w_4), (w_2, w_4), (w_3, w_4) \notin E$ (or any symmetric case).
Let $maxset = \max\{1 + maxset(S - \{v\} - A(v)), 2 + maxset(\bar{A}_1 \cap \bar{A}_4), 2 + maxset(\bar{A}_2 \cap \bar{A}_4), 2 + maxset(\bar{A}_3 \cap \bar{A}_4)\}$.

4.1.2.3: $(w_1, w_2), (w_3, w_4) \in E$, $(w_1, w_3), (w_1, w_4), (w_2, w_3), (w_2, w_4) \notin E$ (or any symmetric case).
Let $maxset = \max\{1 + maxset(S - \{v\} - A(v)), 2 + maxset(\bar{A}_1 \cap \bar{A}_3), 2 + maxset(\bar{A}_2 \cap \bar{A}_3), 2 + maxset(\bar{A}_1 \cap \bar{A}_4), 2 + maxset(\bar{A}_2 \cap \bar{A}_4)\}$.

FINDING A MAXIMUM INDEPENDENT SET 543

4.1.2.4: $(w_1, w_2) \in E$, $(w_1, w_3), (w_2, w_3), (w_1, w_4), (w_2, w_4), (w_3, w_4) \notin E$ (or any symmetric case).
Let $maxset = \max\{1 + maxset(S - \{v\} - A(v)), 2 + maxset(\bar{A}_1 \cap \bar{A}_3), 2 + maxset(\bar{A}_2 \cap \bar{A}_3), 2 + maxset(\bar{A}_1 \cap \bar{A}_4), 2 + maxset(\bar{A}_2 \cap \bar{A}_4), 2 + maxset(\bar{A}_3 \cap \bar{A}_4), 3 + maxset(\bar{A}_1 \cap \bar{A}_3 \cap \bar{A}_4), 3 + maxset(\bar{A}_2 \cap \bar{A}_3 \cap \bar{A}_4)\}$.

4.1.2.5: $(w_i, w_j) \notin E$ for $i \neq j$.
Let $maxset = \max\{1 + maxset(S - \{v\} - A(v)), 2 + maxset(\bar{A}_1 \cap \bar{A}_2), 2 + maxset(\bar{A}_1 \cap \bar{A}_3), 2 + maxset(\bar{A}_1 \cap \bar{A}_4), 2 + maxset(\bar{A}_2 \cap \bar{A}_3), 2 + maxset(\bar{A}_2 \cap \bar{A}_4), 2 + maxset(\bar{A}_3 \cap \bar{A}_4), 3 + maxset(\bar{A}_1 \cap \bar{A}_2 \cap \bar{A}_3), 3 + maxset(\bar{A}_1 \cap \bar{A}_2 \cap \bar{A}_4), 3 + maxset(\bar{A}_1 \cap \bar{A}_3 \cap \bar{A}_4), 3 + maxset(\bar{A}_2 \cap \bar{A}_3 \cap \bar{A}_4), 4 + maxset(\bar{A}_1 \cap \bar{A}_2 \cap \bar{A}_3 \cap \bar{A}_4)\}$.

4.2: $d(w) \geqq 5$ for some vertex w.
Let v, w be such that $d(v) = 4$, $d(w) \geqq 5$, $(v, w) \in E$.
Let $maxset = \max\{1 + maxset(S - \{w\} - A(w)), maxset(S - \{v\})\}$.
Note that $S - \{w\}$ contains a vertex of degree three and all vertices are of degree three or greater.

5: $d(w) = 5$ for all vertices w.
5.1: $|S| = 6$.
Let $maxset = 1$.

5.2: $|S| > 6$.
Let $maxset = \max\{1 + maxset(S - \{v\} - A(v)), maxset(S - \{v\})\}$.
Note that $S - \{v\}$ contains a vertex of degree four, a vertex of degree five, and all vertices are of degree four or greater.

6: Some vertex w has $d(w) \geqq 6$.
Let $maxset = \max\{1 + maxset(S - \{w\} - A(w)), maxset(S - \{w\})\}$.

end *maxset*.

The description of Robson's algorithms (2001) takes 18 pages resulting into claimed running time $\mathcal{O}(1.1893^n)$.

Difference between polynomial and exponential algorithms

Polynomial time algorithms: (usually) exact time analysis.

Exponential time algorithms: different story

Playing with measure μ

We measure the progress in the number of vertices, $\mu = n$.

It can be the number of edges, $\mu = n$, or some function $\mu = f(m, n)$.

Or a function $\mu = f(m, n_1, n_2, \ldots, n_k)$, where n_i is the number of vertices of degree i.

Playing with measure

Then recursions are of type

$$T(\mu) = \mathcal{O}^*(\sum_{1 \leq i \leq k} T(\mu - \varepsilon_i))$$

The running time of the algorithm is $\mathcal{O}(c^{\mu})$, where c is the unique positive root of

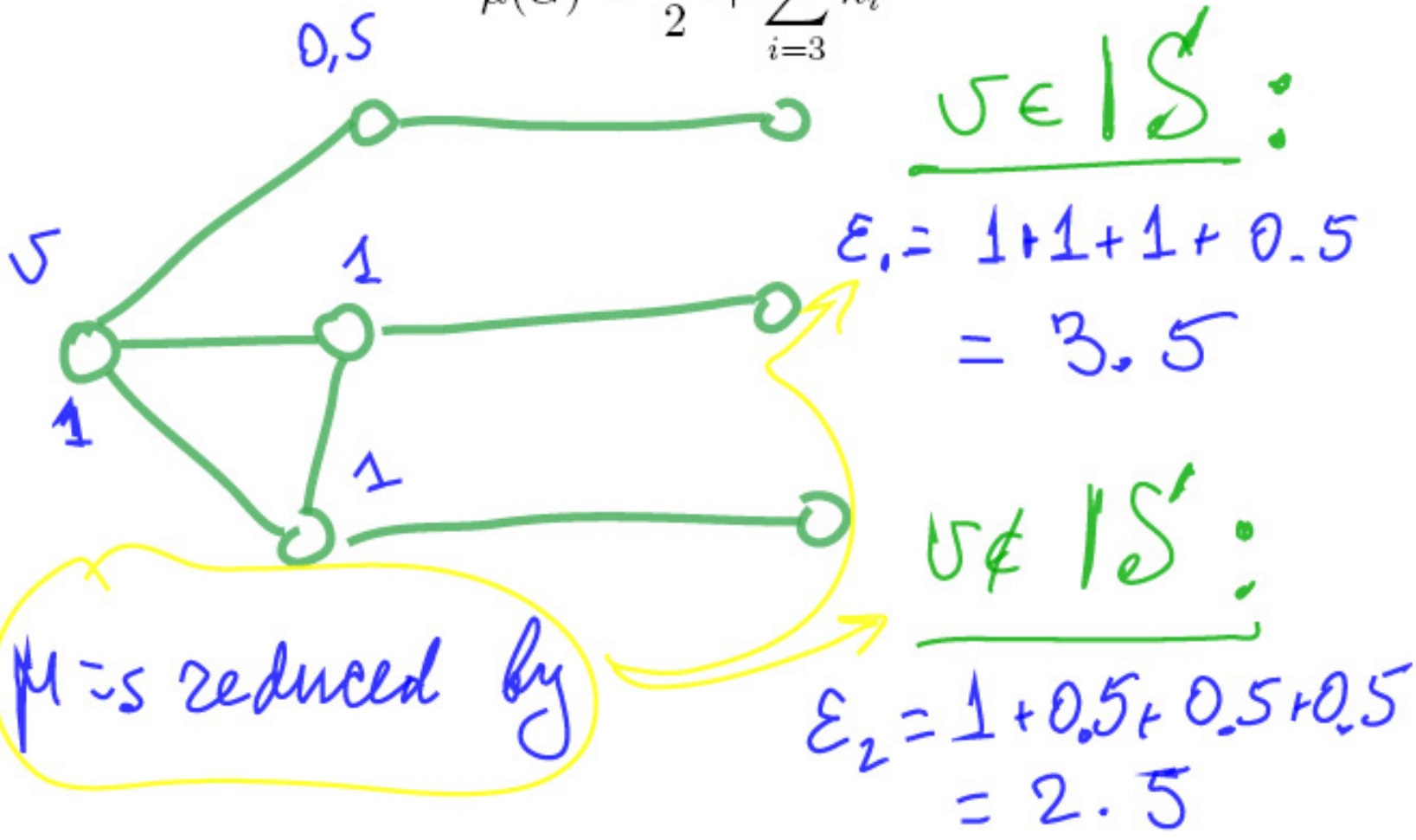
$$1 = \sum_{1 \leq i \leq k} \frac{1}{x^{\varepsilon_i}}$$

Finally, to estimate the progress in n, we find a function g s.t. $\mu(G) \leq g(n)$. Then the running time is $\mathcal{O}(c^{g(n)})$.

Why measure helps?

Let us put weight on vertices of degree two $\omega = 0.5$.

$$\mu(G) = \frac{n_2}{2} + \sum_{i=3}^{n} n_i$$



Why measure helps?

The running time

$$t(\mu) \leq \min \begin{cases} t(\mu-1)+t(\mu-5) \\ t(\mu-1-\omega)+t(\mu-4-\omega) \\ t(\mu-1-2\omega)+t(\mu-3-2\omega) \\ t(\mu-1-3\omega)+t(\mu-2-3\omega) \\ \dots \\ t(\mu-1-3\omega)+t(\mu-1-3(1-\omega)) \end{cases}$$

brings to

$$t(\mu) = \mathcal{O}^*(1.320^{\mu}) = \mathcal{O}^*(1.320^{n}),$$

Algorithm is the same, only the analyzes has been changed!

Measure & Conquer

- How to find the right measure to analyze branching algorithms?
- Minimum Dominating Set [FF, Grandoni, & Kratsch, ICALP 2005]
- Maximum Independent Set [FF, Grandoni, & Kratsch, SODA 2006, J ACM 2009]
- Surprise: Simple algorithms (which are easier to analyze) provide better running times.
- Importance of giving lower bounds for exponential algorithms

Measure & Conquer

- How to find the right measure to analyze branching algorithms?
- Minimum Dominating Set [FF, Grandoni, & Kratsch, ICALP 2005]
- Maximum Independent Set [FF, Grandoni, & Kratsch, SODA 2006, J ACM 2009]
- Surprise: Simple algorithms (which are easier to analyze) provide better running times.
- Importance of giving lower bounds for exponential algorithms

Maximal Independent Sets

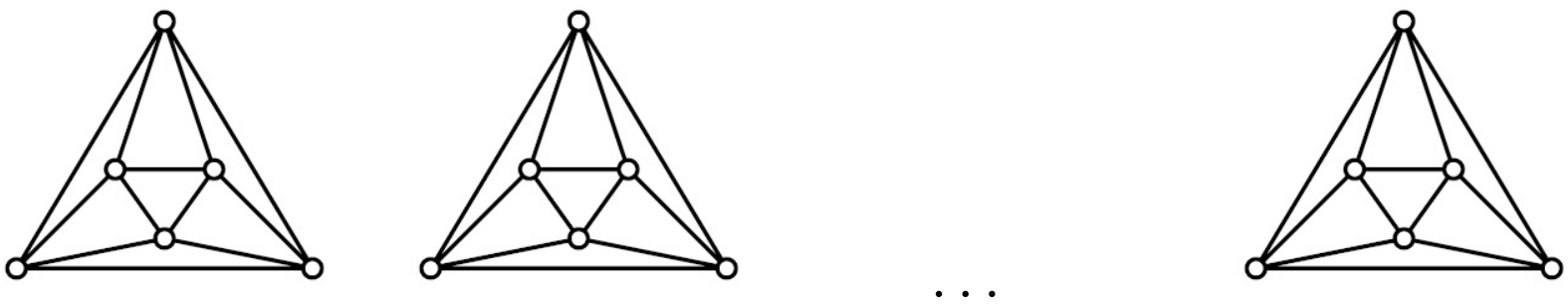
Moon and Moser 1965. The number of maximal independent sets in a graph on n vertices is at most $3^{n/3}$. The bound is tight: there are graphs containing $3^{n/3}$ maximal independent sets.

Combinatorial questions: How many?

Minimal Dominating Sets

[FF, Grandoni, Pyatkin, Stepanov, TALG 2008.] Every n-vertex graph has at most 1.7159^n minimal dominating sets

Lower bound $15^{n/6} = 1.5704^n$ $n/6$ copies of octahedron.

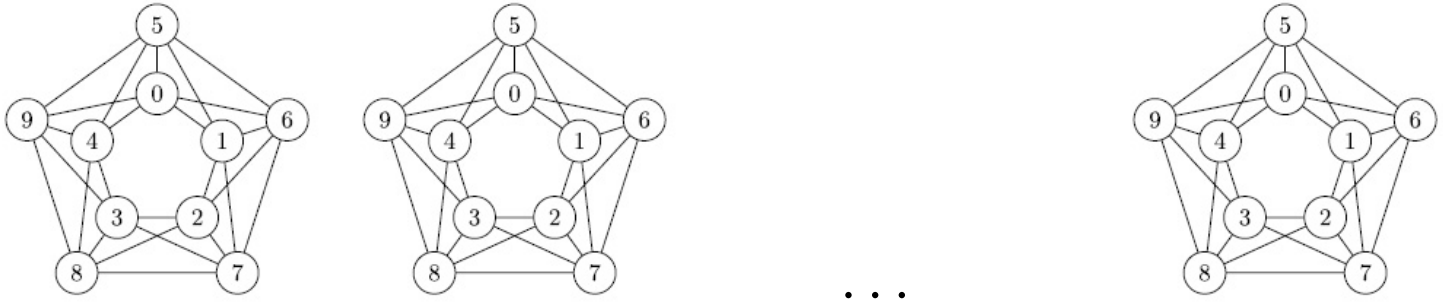


Combinatorial questions: How many?

Maximum Induced Forests (Minimum Vertex Feedback Sets)

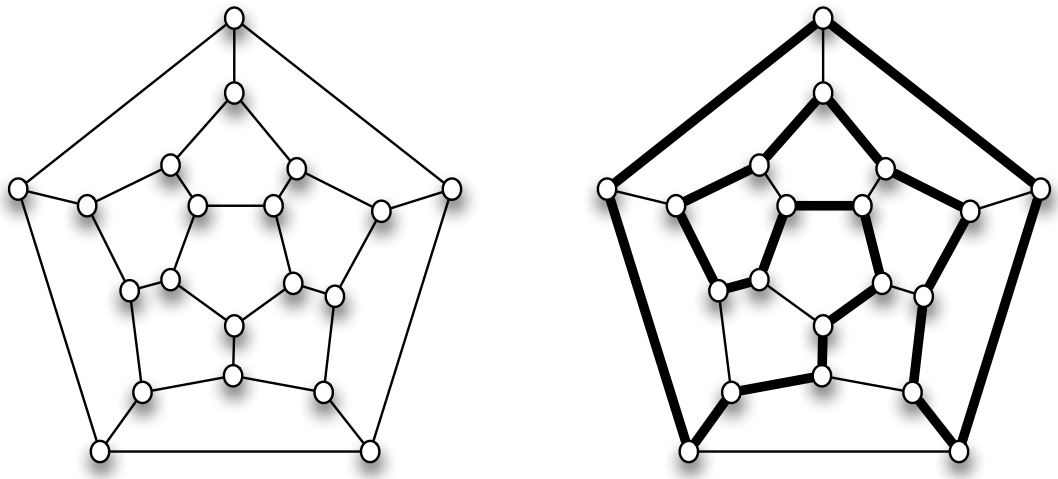
[FF, Gaspers, Pyatkin, Razgon, Algorithmica, 2008] Every n-vertex graph contains at most 1.8638^n maximum induced forests.

Lower bound $105^{n/10} > 1.5926^n$



Inclusion-Exclusion

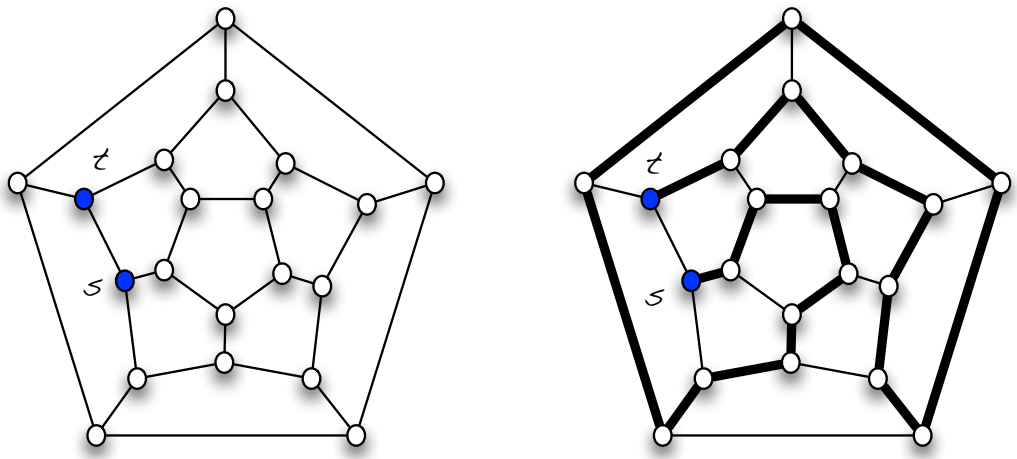
Hamiltonian cycle problem: we are given a graph on n vertices, the task is to decide whether the graph has a Hamiltonian cycle, which is a cycle visiting every vertex of the graph exactly once.



A special case of the famous *Travelling Salesman Problem*.

Hamiltonian path

Stronger version, *Hamiltonian path* problem: the first vertex s and the last vertex t, and asks to decide whether the graph has a path that starts at s, ends at t, and visits all the vertices exactly once.



Hamiltonian path

- Brute-force algorithm — try all possible permutations starting from s and ending by t.
- On n-vertex graphs it takes time $\mathcal{O}((n-2)!n) = \mathcal{O}^*(n!)$.
- Bellman (1962) and Held and Karp (1962) used dynamic programming to solve the problem in time $\mathcal{O}(2^n n^2)$ (and exponential space)
- We discuss: How to use inclusion-exclusion

Hamiltonian path

- Brute-force algorithm — try all possible permutations starting from s and ending by t.
- On n-vertex graphs it takes time $\mathcal{O}((n-2)!n) = \mathcal{O}^*(n!)$.
- Bellman (1962) and Held and Karp (1962) used dynamic programming to solve the problem in time $\mathcal{O}(2^n n^2)$ (and exponential space)
- We discuss: How to use inclusion-exclusion

Hamiltonian path

- Brute-force algorithm — try all possible permutations starting from s and ending by t.
- On n-vertex graphs it takes time $\mathcal{O}((n-2)!n) = \mathcal{O}^*(n!)$.
- Bellman (1962) and Held and Karp (1962) used dynamic programming to solve the problem in time $\mathcal{O}(2^n n^2)$ (and exponential space)
- We discuss: How to use inclusion-exclusion

A DP Approach to Hamiltonian Path Problem

Dmitriy Nuriyev

(Submitted on 14 Jan 2013)

A Dynamic Programming based polynomial worst case time algorithm is described for computing Hamiltonian Path of a directed graph. Complexity constructive proofs along with a tested C++ implementation are provided as well. The result is obtained via the use of original colored hypergraph structures in order to maintain and update the necessary DP states.

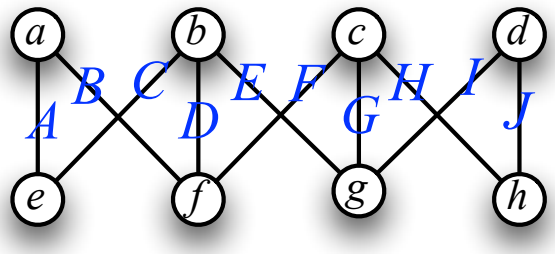
Subjects: **Data Structures and Algorithms (cs.DS)**
Cite as: arXiv:1301.3093 **[cs.DS]**
(or arXiv:1301.3093v1 **[cs.DS]** for this version)

Submission history

From: Dmitriy Nuriyev [view email]
[v1] Mon, 14 Jan 2013 18:46:00 GMT (410kb,D)

Which authors of this paper are endorsers?

Walks in graphs



We assume that $s = a$ and $t = h$. A *walk* of length $n - 1$ that starts from s and ends at t can be viewed as a string of length $2n - 1$ with alternating and possibly repeating vertices and edges, such as

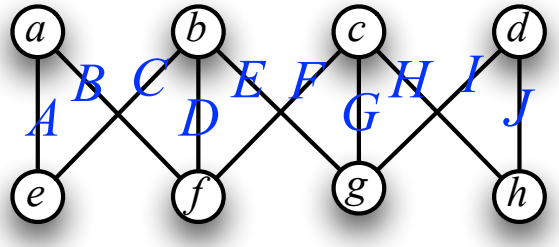
$$aAeCbDfFcGgIdJh$$

or

$$aBfDbEgGcFfFcHh\,.$$

Walks in graphs

Each such walk makes exactly n visits to vertices and contains, possibly with repetitions, $n-1$ edges.
The walk is a Hamiltonian path if and only if the walk visits n distinct vertices



$$aAeCbDfFcGgIdJh$$

is a path, and

$$aBfDbEgGcFfFcHh\,.$$

is a non-path.

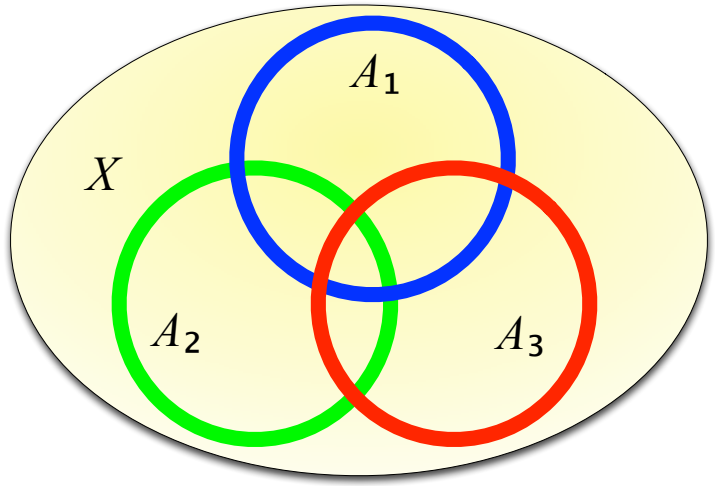
Walks in graphs

Counting Hamiltonian paths is hard but counting walks is easy.

- To compute the number of walks of length k between s and t just look at the s,t-entry of A^k matrix. (A adjacency matrix of the input graph.)
- Alternative way: do dynamic programming

Inclusion-Exclusion

Consider a finite set X and three subsets A_1, A_2, and A_3.



To obtain $|A_1 \cup A_2 \cup A_3|$, we can use the following formula

$$\begin{aligned} |A_1 \cup A_2 \cup A_3| = {} & |A_1| + |A_2| + |A_3| \\ & - |A_1 \cap A_2| - |A_1 \cap A_3| - |A_2 \cap A_3| \\ & + |A_1 \cap A_2 \cap A_3| \,, \end{aligned}$$

or, equivalently,

$$\begin{aligned} |X \setminus (A_1 \cup A_2 \cup A_3)| = {} & |X| - |A_1| - |A_2| - |A_3| \\ & + |A_1 \cap A_2| + |A_1 \cap A_3| + |A_2 \cap A_3| \\ & - |A_1 \cap A_2 \cap A_3| \,. \end{aligned}$$

Inclusion-Exclusion

The case when there are q subsets $A_1, A_2, \ldots, A_q$ of X

$$\left| X \setminus \bigcup_{i=1}^{q} A_i \right| = \sum_{J \subseteq \{1,2,\ldots,q\}} (-1)^{|J|} \left| \bigcap_{j \in J} A_j \right|.$$

What it has to do with walks and cycles?

- Take $q = n - 2$ and suppose that the vertices other than s and t are labeled with integers $1, 2, \ldots, n - 2$.
- Let X be the set of all walks of length $n - 1$ from s to t and, for each $i = 1, 2, \ldots, n - 2$, let A_i be the set of walks in X that avoid the vertex i.
- Then $X \setminus \bigcup_{i=1}^{q} A_i$ is the set of Hamiltonian paths, and we can use I-E to count their number.

Running time

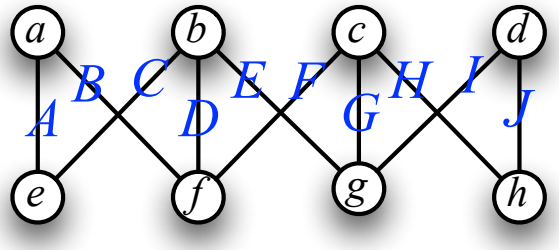
$$\left|X \setminus \bigcup_{i=1}^{q} A_i\right| = \sum_{J \subseteq \{1,2,\ldots,q\}} (-1)^{|J|} \left|\bigcap_{j \in J} A_j\right|.$$

- For each fixed $J \subseteq \{1, 2, \ldots, q\}$, the right-hand side can be computed in time polynomial in n by counting the number of walks of length $n-1$ from s to t in the graph with the vertices in J deleted.
- Running time $\mathcal{O}(2^n n)$.

Historical comments

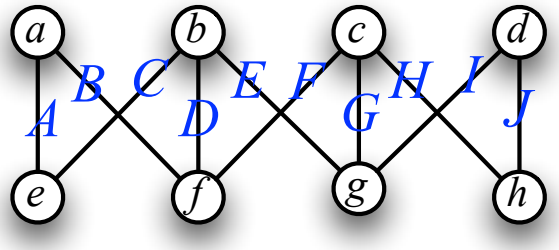
- Dynamic programming $\mathcal{O}^*(2^n)$, Bellman and Held-Karp (1962)
- I-E for Hamiltonian path and TSP rediscovered several times
- In 1969, Kohn, Gottlieb, and Kohn
- In 1982, Karp
- In 1993, Bax

More on Hamiltonian cycles



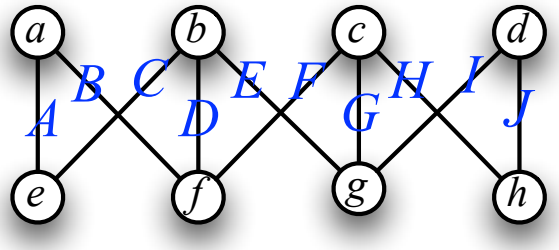
- Can we do better on bipartite graphs?
- Naive approach: do inclusion-exclusion on each part of the graph, time $O(2^{n/2})$.
- Naive approach does not work, why?

More on Hamiltonian cycles



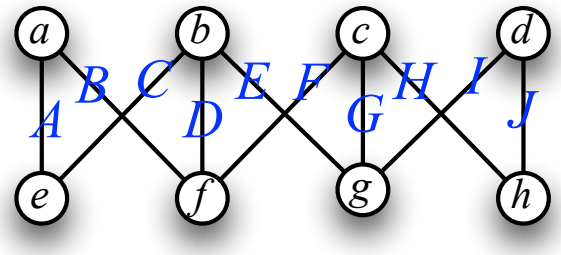
- Can we do better on bipartite graphs?
- Naive approach: do inclusion-exclusion on each part of the graph, time $O(2^{n/2})$.
- Naive approach does not work, why?

More on Hamiltonian cycles



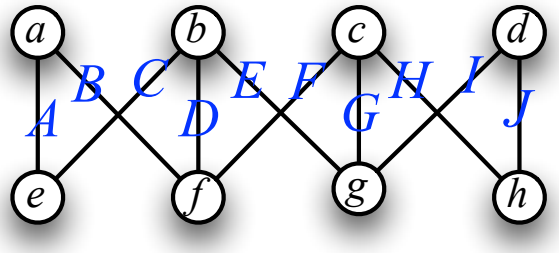
- Can we do better on bipartite graphs?
- Naive approach: do inclusion-exclusion on each part of the graph, time $O(2^{n/2})$.
- Naive approach does not work, why?

More on Hamiltonian cycles



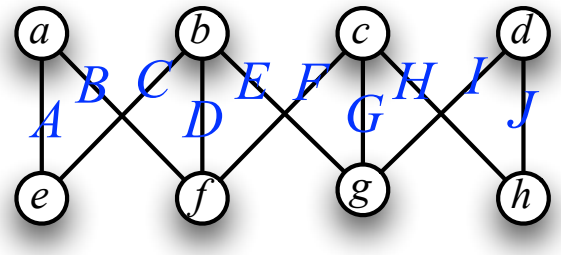
- IDEA:
- Do inclusion-exclusion on the vertices of the upper part, to exclude all walks not covering it.
- "Expand" signatures of walks to ensure that walks not covering the lower part are counted even number of times.

More on Hamiltonian cycles



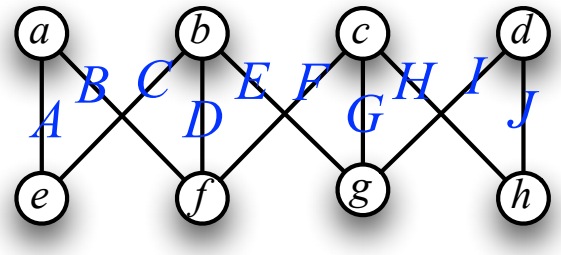
- IDEA:
- Do inclusion-exclusion on the vertices of the upper part, to exclude all walks not covering it.
- "Expand" signatures of walks to ensure that walks not covering the lower part are counted even number of times.

More on Hamiltonian cycles



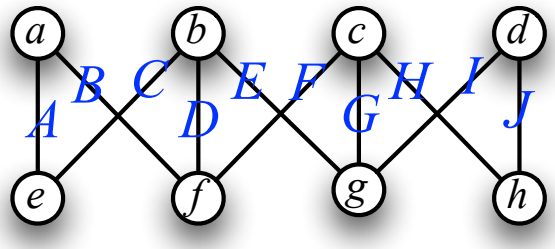
- IDEA:
- Do inclusion-exclusion on the vertices of the upper part, to exclude all walks not covering it.
- "Expand" signatures of walks to ensure that walks not covering the lower part are counted even number of times.

More on Hamiltonian cycles



- IDEA:
- Do inclusion-exclusion on the vertices of the upper part, to exclude all walks not covering it.
- "Expand" signatures of walks to ensure that walks not covering the lower part are counted even number of times.

More on Hamiltonian cycles



- Every walk of length $n-1$ makes n visits to vertices, where exactly $n/2$ visits are to vertices in V_1.
- *Label* each of the $n/2$ visits to V_1 by an integer from $L = \{1, 2, \ldots, n/2\}$. Each walk has $(n/2)^{n/2}$ possible labelings, exactly $(n/2)!$ of which are *bijective*; that is, each label is used exactly once.

For example,

$$\underset{1}{a}AeC\underset{3}{b}Df\underset{4}{B}a\underset{2}{B}fF\underset{}{c}Hh\,.$$

is a bijectively labeled non-path.

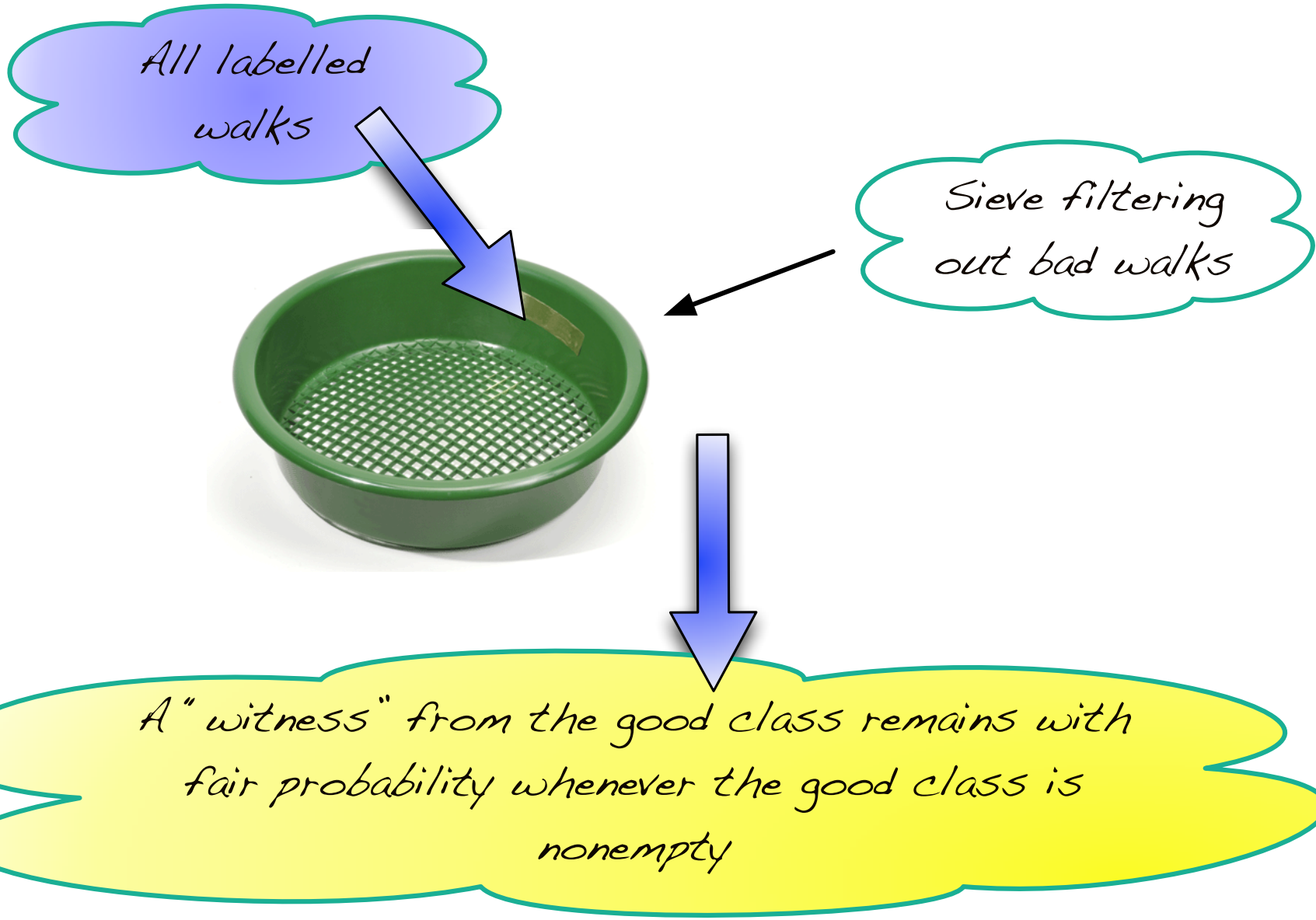
More on Hamiltonian cycle

We partition the set of all labeled walks into two disjoint classes, the "good" class and the "bad" class.

- A labeled walk is *good* if the labeling is bijective and the walk is a path.
- Otherwise a labeled walk is *bad*.

Important: the good class is nonempty if and only if the graph has a Hamiltonian path from s to t.

Randomized algorithm



Sieve construction

'hash' labelled walks so that every bad hash value appeared even number of times

Each good hash value appears once

Hash

The *hash* of a labeled walk is the multiset that consists of all the elements visited by a walk, together with their labels (if any).

$$\underset{1}{a}AeC\underset{3}{b}Df\,B\underset{4}{a}Bf\,F\underset{2}{c}Hh\,.$$

Hash:

$$\{A, B, B, C, D, F, H, \underset{1}{a}, \underset{4}{a}, \underset{3}{b}, \underset{2}{c}, e, f, f, h\}\,.$$

- We cannot reconstruct a labeled walk from its hash value.
- However, every bijectively labeled path—that is, every good labeled walk—can be reconstructed from its hash value.

Each good labeled walk has a unique hash value!

Sieve construction

'hash' labelled walks so that every bad hash value appeared even number of times

bijectively labeled non-paths

non-bijectively labeled walks

Each good hash value appears once

Sieve construction

Case A. Counting bijectively labeled non-paths. Let W be a bijectively labeled non-path.

- We want to map it with a b.l.n-p. W' of the same hash value.
- Take the first minimal closed subwalk in W.

For example, for

$$\underset{1}{a}Ae\underset{3}{C}\underset{3}{b}D\underset{4}{f}B\underset{4}{a}B\underset{2}{f}F\underset{2}{c}Hh\,.$$

we take

$$\underset{1}{a}Ae\underset{3}{C}\underset{3}{b}D\underset{4}{f}B\underset{4}{a}$$

Sieve construction

Case A1. If the repeated vertex is in V_1, then swap labels

$$\underset{1}{a}AeC\underset{3}{b}D\underset{4}{B}\underset{2}{F}cHh \;\to\; \underset{4}{a}AeC\underset{3}{b}D\underset{1}{B}\underset{2}{F}cHh$$

Case A2. If the repeated vertex is in V_2, then reverse the subwalk

$$\underset{1}{a}B\underset{3}{D}\underset{4}{E}g\underset{2}{G}cF\underset{2}{F}cHh \to \underset{1}{a}B\underset{4}{F}\underset{3}{G}g\underset{2}{E}bDFcHh\,.$$

- W and W' have the same hash values.

Sieve construction

- Is $W \neq W'$?

Case A1. Yes

Case A2. If the repeated vertex is in V_2, then reverse the subwalk

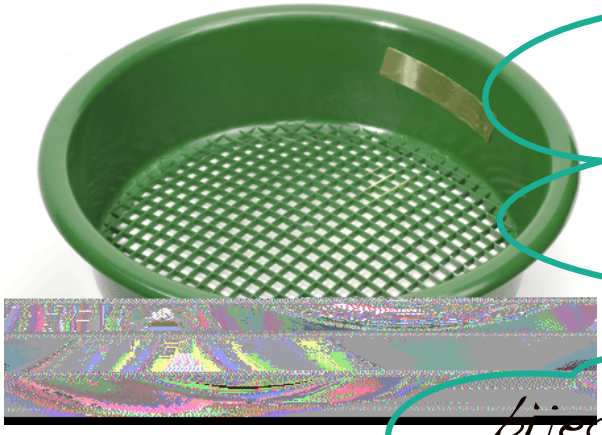
$$\underset{1}{a}B\underset{3}{f}D\underset{4}{b}E\underset{2}{g}GcFfFcHh \to \underset{1}{a}B\underset{4}{f}F\underset{3}{c}G\underset{2}{g}EbDfFcHh\,.$$

In general, reversing the first closed subwalk will not result in a different labeled walk – it may be a palindrome

$$\underset{1}{a}Ae\underset{2}{C}b\underset{3}{C}eA\underset{4}{a}BfFcHh\,.$$

Fortunately, because of bijective labeling, the only possible pitfall is a palindrome of length 5 that starts at V_2, visits a vertex in V_1, and returns to the same vertex in V_2!!!

Sieve construction



'hash' labelled walks so that every bad hash value appeared even number of times

bijectively labeled non-paths

non-bijectively labeled walks

Each good hash value appears [illegible]

For every such walk W (with small exception), we found a matching walk W'

Sieve construction

Case B. Counting non-bijectively labeled walks.

- Each non-bijectively labeled walk W *avoids at least one label* from the set of all labels L. In particular, if W avoids exactly a labels, there are exactly 2^a sets $A \subseteq L$ such that W avoids every label in A (and possibly some other labels outside A).
- For each subset $A \subseteq L$, we insert into the sieve the hash value of each labeled walk that avoids every label in A.
- After all subsets A have been considered, a hash value occurs with odd multiplicity in the sieve if and only if it originates from a good labeled walk.

Sieve construction

'hash' labelled walks so that every bad hash value appeared even number of times

bijectively-labeled non-paths

non-bijectively labeled walks

For every such walk w (with small exception), we found a matching walk w'

Inclusion-exclusion over subsets of labels

Second key idea

- There are too many hash values, so instead of sieving hash values explicitly, we sieve only their weights.
- Assign an integer weight in the interval $1, 2, \ldots, n(n+1)$ independently and uniformly at random to each of the $n/2 + n/2 \cdot n/2 + n/2 \cdot n/2 = (n+1)n/2$ elements that may occur in a hash value.
- The *weight* of a hash value is the sum of the weights of its elements.

Second key idea

- When running the sieve, instead of tracking the (partial) walks and their (partial) hash values by dynamic programming, we only track the number of hash values of each weight.
- This enables us to process each fixed $A \subseteq L$ in time polynomial in n.
- The number of all sets $A \subseteq L$ is $2^{|L|} \leq 2^{n/2} < 1.42^n$. Thus the total running time of the above procedure is $\mathcal{O}(1.42^n)$.

Sieve construction

What we have:

- Each bad hash value gets inserted into the sieve an even number of times, and in particular contributes an even increment to the counter corresponding to the weight of the hash value.
- Thus, an odd counter can arise only if a good hash value was inserted into the sieve; that is, the graph has a Hamiltonian path.

The presence of an odd counter implies the existence of a Hamiltonian path!!!

Algorithm

- Assign an integer weight in the interval $1, 2, \ldots, n(n+1)$ independently and uniformly at random to each of the $(n+1)n/2$ elements that may occur in a hash value.
- Implement sieve
- When the sieve terminates, we assert that the input graph has a Hamiltonian path if the counter for the number of hash values of at least one weight is odd; otherwise we assert that the graph has no Hamiltonian path.

Isolation Lemma

- What is the probability of failure?

Lemma (Isolation Lemma, Mulmuley, Vazirani, and Vazirani 1987)

For any set family over a base set of m elements, if we assign a weight independently and uniformly at random from $1, 2, \ldots, r$ to each element of the base set, there will be a unique set of the minimum weight in the family with probability at least $1 - m/r$.

Isolation Lemma

Lemma (Isolation Lemma, Mulmuley, Vazirani, and Vazirani 1987)

For any set family over a base set of m elements, if we assign a weight independently and uniformly at random from $1, 2, \ldots, r$ to each element of the base set, there will be a unique set of the minimum weight in the family with probability at least $1 - m/r$.

If we consider the set family of good hash values—indeed, each good hash value is a set—there is a unique such hash value of the minimum weight—and hence an odd counter in the sieve—with probability at least $1/2$.

Conclusion

Our randomized algorithm is

- detecting Hamiltonian paths in bipartite graphs in time $\mathcal{O}(1.42^n)$,
- gives no false positives, and gives a false negative with probability at most $1/2$.

The algorithm could be extended to graphs that are not bipartite with running time $\mathcal{O}(1.66^n)$ by partitioning the vertices randomly into V_1 and V_2 and employing a bijective labeling also for the edges with both ends in V_2.

Historic notes

- The breakthrough for Hamiltonian cycle by [Andreas Björklund, FOCS 2010].
- We followed the line of the proof used by [Cygan, Nederlof, Pilipczuk, Pilipczuk, van Rooij, and Wojtaszczyk, FOCS 2011] for a related problem.

Other techniques

- Local search for k-SAT [Dantsin, Goerdt, Hirsch, Kannan, Kleinberg, Papadimitriou, Raghavan, & Schöning, TCS 2002],
- Split & List for MAX-2-SAT or MAX-CUT [Williams, TCS 2005]
- More, more, more...

Concrete open problems

- All techniques fail on SAT so far. Is it possible to solve SAT in time $\mathcal{O}((2-\varepsilon)^n + m)$ for some $\varepsilon > 0$? (n -number of variables, m - number of clauses.)
- TSP in time $\mathcal{O}((2-\varepsilon)^n)$?
- Is it possible to derandomize Bjorklund's algorithm?
- Graph Coloring in time $\mathcal{O}((2-\varepsilon)^n)$?
- Chromatic index, subgraph isomorphism, hundreds of scheduling problems, ...

Challenge

- Complexity theory for exponential algorithms

Conclusion

The area of moderately exponential time algorithms is still in nascent stage and there is a lot to discover.

Choose your favorite problem hard problem and try to get an exact algorithm for it. Enjoy!

Conclusion

The area of moderately exponential time algorithms is still in nascent stage and there is a lot to discover.

Choose your favorite problem hard problem and try to get an exact algorithm for it. Enjoy!