Sublinear Graph Approximation Algorithms

Krzysztof Onak
MIT

Motivation

- Want to learn a combinatorial parameter of a graph:
  - the maximum matching size
  - the independence number $\alpha(G)$,
  - the minimum vertex cover size,
  - the minimum dominating set size

Motivation

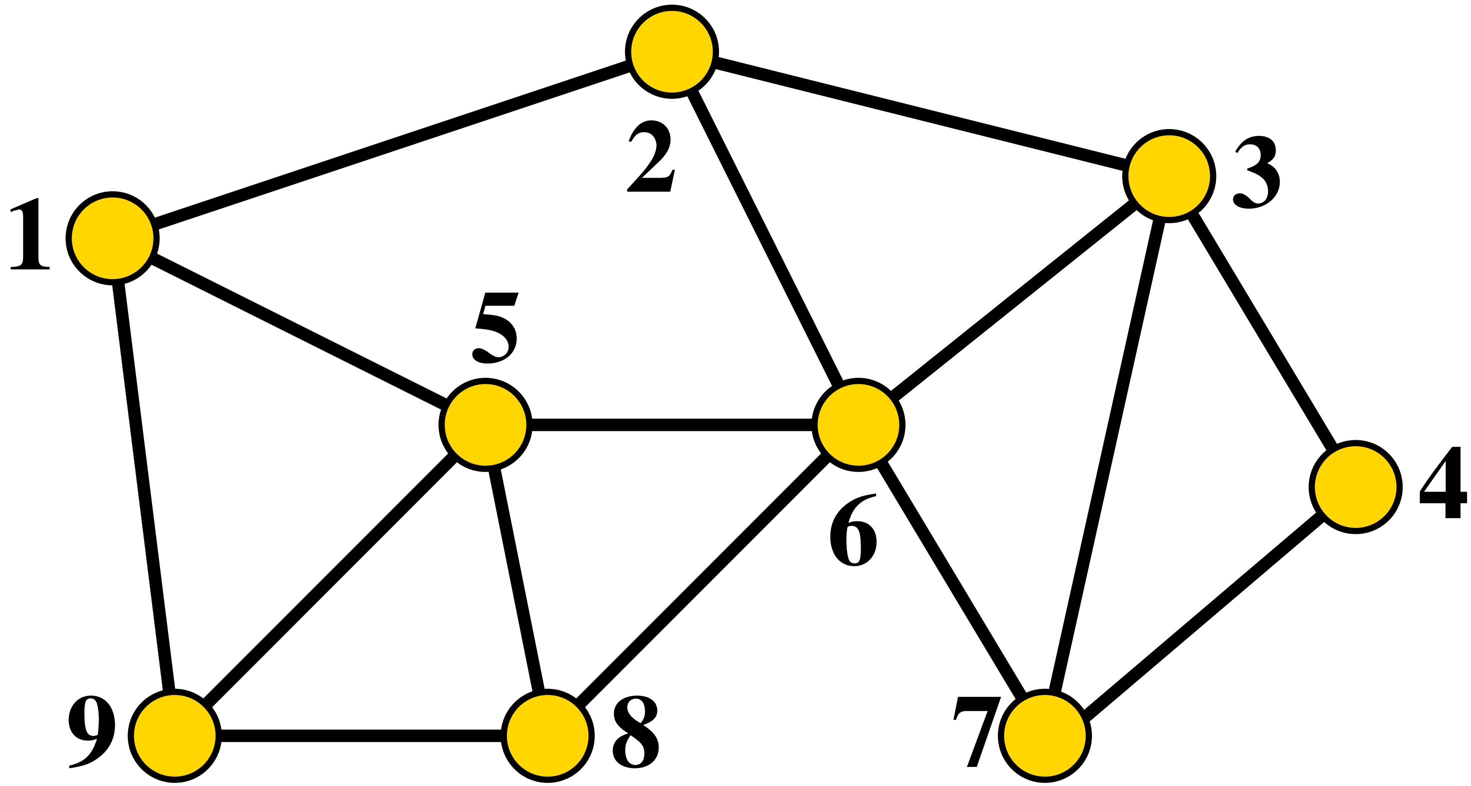
- Want to learn a combinatorial parameter of a graph:
  - the maximum matching size
  - the independence number $\alpha(G)$,
  - the minimum vertex cover size,
  - the minimum dominating set size
- Is there a way to compute/approximate it without finding:
  - large matching,
  - large independent set,
  - small vertex cover,
  - small dominating set?

Motivation

- Want to learn a combinatorial parameter of a graph:
  - the maximum matching size
  - the independence number $\alpha(G)$,
  - the minimum vertex cover size,
  - the minimum dominating set size
- Is there a way to compute/approximate it without finding:
  - large matching,
  - large independent set,
  - small vertex cover,
  - small dominating set?
- The answer is YES in many cases

The Model

Bounded-degree graph G:



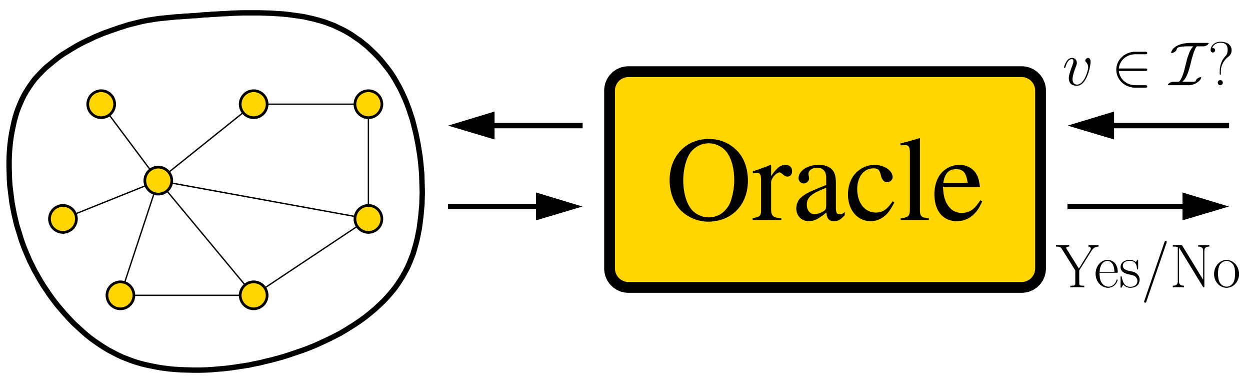
Query access to adjacency list of each node

Finding a Maximal Independent Set Locally

Oracle for Maximal Independent Set

Construct oracle $\mathcal{O}$:

- $\mathcal{O}$ has query access to $G = (V, E)$
- $\mathcal{O}$ provides query access to maximal independent set $\mathcal{I} \subseteq V$
- $\mathcal{I}$ is not a function of queries
  it is a function of G and random bits

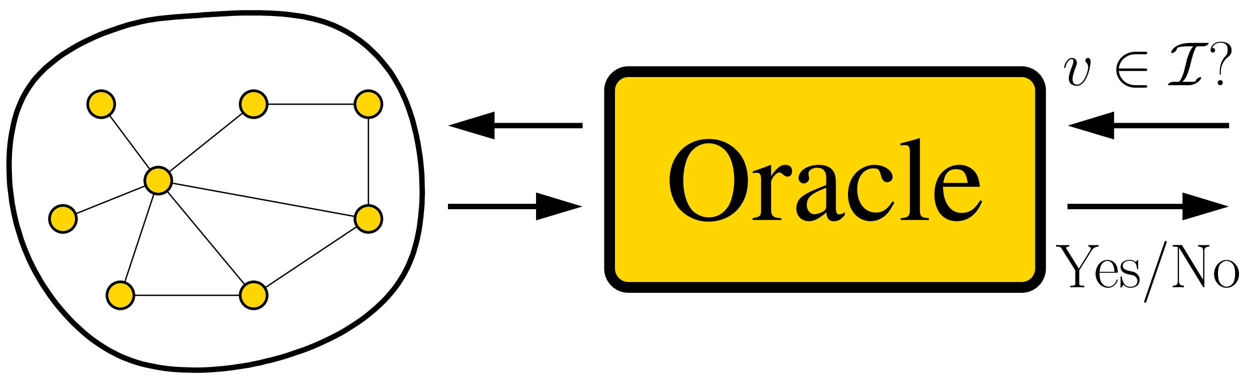


Goal: Minimize the query processing time

Oracle for Maximal Independent Set

Construct oracle $\mathcal{O}$:

- $\mathcal{O}$ has query access to $G = (V, E)$
- $\mathcal{O}$ provides query access to maximal independent set $\mathcal{I} \subseteq V$
- $\mathcal{I}$ is not a function of queries
  it is a function of G and random bits



Goal: Minimize the query processing time

One solution: Luby's maximal independent set algorithm (1986)
simulated locally [Marko, Ron 2007]

Here: a method better for sublinear algorithms

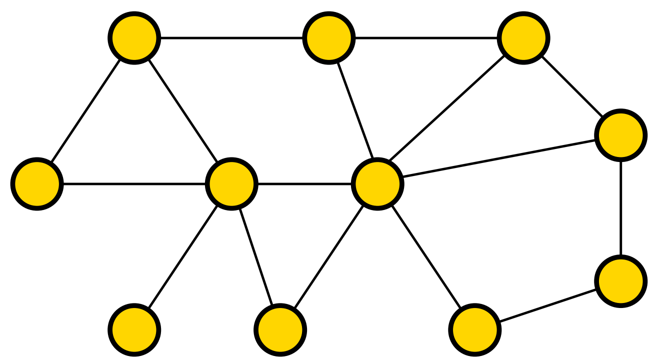
Our Method

Nguyen, O. (2008)

Main idea:

- select maximal independent set greedily
- consider vertices in random order

Random order $\equiv$ random numbers $r(v)$ assigned to each vertex



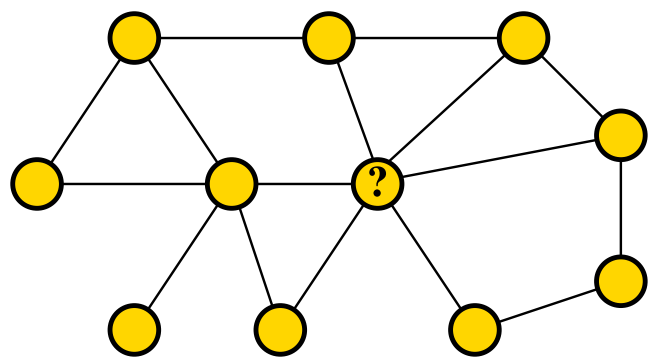
Our Method

Nguyen, O. (2008)

Main idea:

- select maximal independent set greedily
- consider vertices in random order

Random order $\equiv$ random numbers $r(v)$ assigned to each vertex



To check if $v \in \mathcal{I}$

- recursively check if neighbors w s.t. $r(w) < r(v)$ are in $\mathcal{I}$
- $v \in \mathcal{I} \iff$ none of them in $\mathcal{I}$

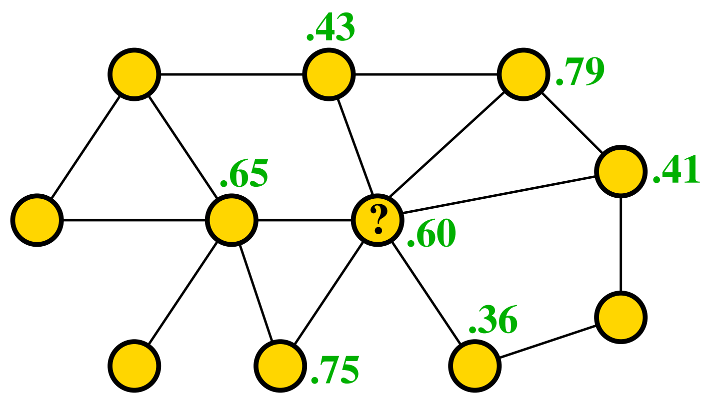
Our Method

Nguyen, O. (2008)

Main idea:

- select maximal independent set greedily
- consider vertices in random order

Random order $\equiv$ random numbers $r(v)$ assigned to each vertex



To check if $v \in \mathcal{I}$

- recursively check if neighbors w s.t. $r(w) < r(v)$ are in $\mathcal{I}$
- $v \in \mathcal{I} \iff$ none of them in $\mathcal{I}$

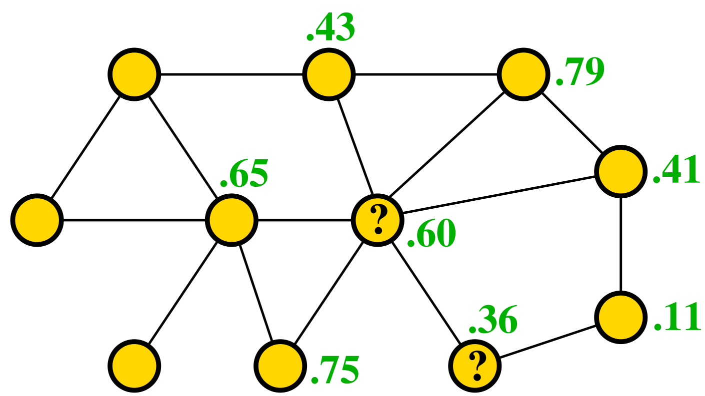
Our Method

Nguyen, O. (2008)

Main idea:

- select maximal independent set greedily
- consider vertices in random order

Random order $\equiv$ random numbers $r(v)$ assigned to each vertex



To check if $v \in \mathcal{I}$

- recursively check if neighbors w s.t. $r(w) < r(v)$ are in $\mathcal{I}$
- $v \in \mathcal{I} \iff$ none of them in $\mathcal{I}$

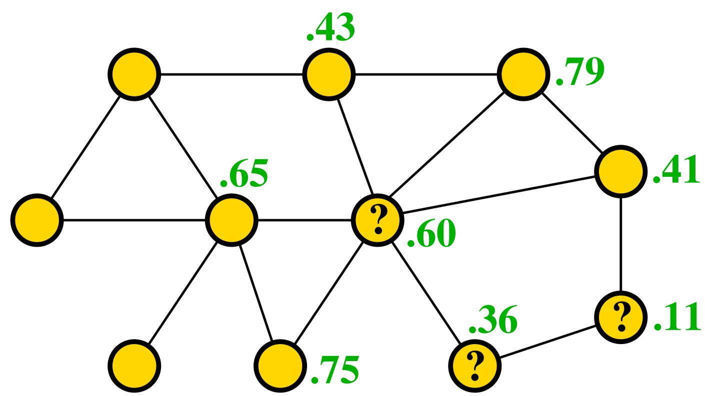
Our Method

Nguyen, O. (2008)

Main idea:

- select maximal independent set greedily
- consider vertices in random order

Random order $\equiv$ random numbers $r(v)$ assigned to each vertex



To check if $v \in \mathcal{I}$

- recursively check if neighbors w s.t. $r(w) < r(v)$ are in $\mathcal{I}$
- $v \in \mathcal{I} \iff$ none of them in $\mathcal{I}$

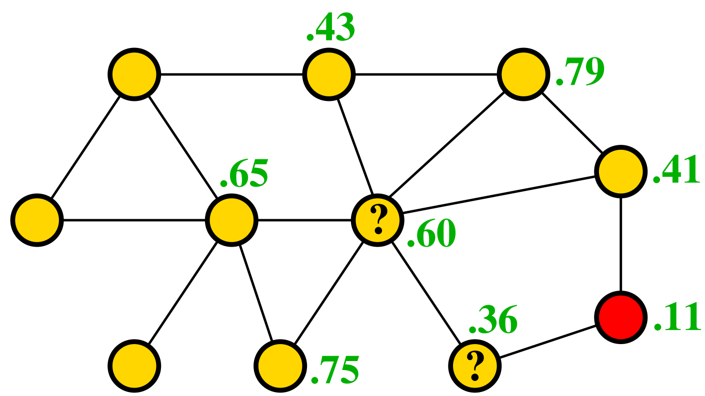
Our Method

Nguyen, O. (2008)

Main idea:

- select maximal independent set greedily
- consider vertices in random order

Random order $\equiv$ random numbers $r(v)$ assigned to each vertex



To check if $v \in \mathcal{I}$

- recursively check if neighbors w s.t. $r(w) < r(v)$ are in $\mathcal{I}$
- $v \in \mathcal{I} \iff$ none of them in $\mathcal{I}$

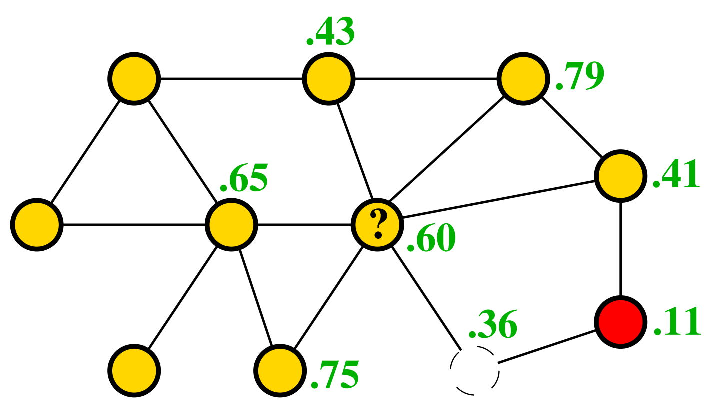
Our Method

Nguyen, O. (2008)

Main idea:

- select maximal independent set greedily
- consider vertices in random order

Random order $\equiv$ random numbers $r(v)$ assigned to each vertex



To check if $v \in \mathcal{I}$

- recursively check if neighbors w s.t. $r(w) < r(v)$ are in $\mathcal{I}$
- $v \in \mathcal{I} \iff$ none of them in $\mathcal{I}$

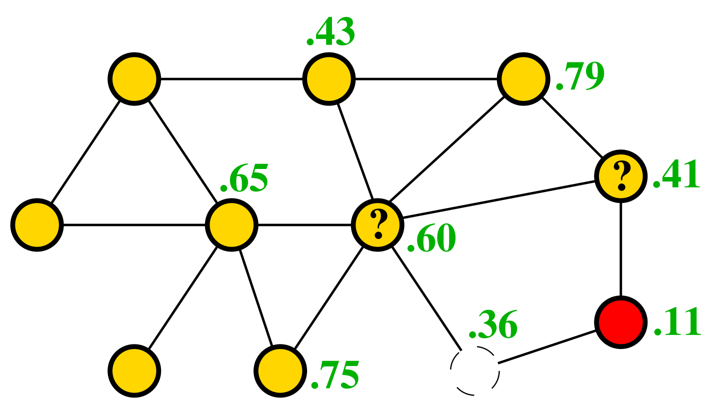
Our Method

Nguyen, O. (2008)

Main idea:

- select maximal independent set greedily
- consider vertices in random order

Random order $\equiv$ random numbers $r(v)$ assigned to each vertex



To check if $v \in \mathcal{I}$

- recursively check if neighbors w s.t. $r(w) < r(v)$ are in $\mathcal{I}$
- $v \in \mathcal{I} \iff$ none of them in $\mathcal{I}$

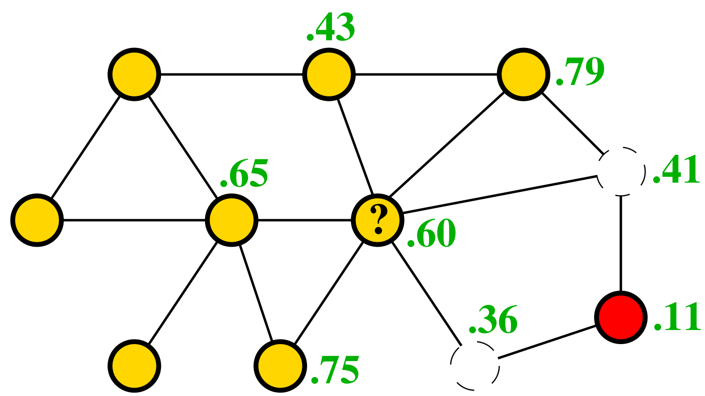
Our Method

Nguyen, O. (2008)

Main idea:

- select maximal independent set greedily
- consider vertices in random order

Random order $\equiv$ random numbers $r(v)$ assigned to each vertex



To check if $v \in \mathcal{I}$

- recursively check if neighbors w s.t. $r(w) < r(v)$ are in $\mathcal{I}$
- $v \in \mathcal{I} \iff$ none of them in $\mathcal{I}$

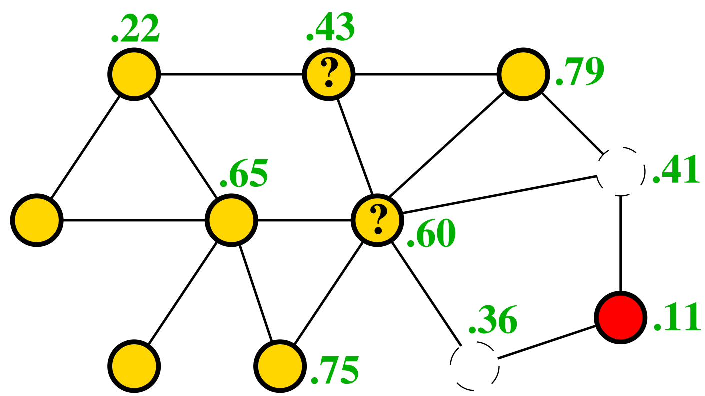
Our Method

Nguyen, O. (2008)

Main idea:

- select maximal independent set greedily
- consider vertices in random order

Random order $\equiv$ random numbers $r(v)$ assigned to each vertex



To check if $v \in \mathcal{I}$

- recursively check if neighbors w s.t. $r(w) < r(v)$ are in $\mathcal{I}$
- $v \in \mathcal{I} \iff$ none of them in $\mathcal{I}$

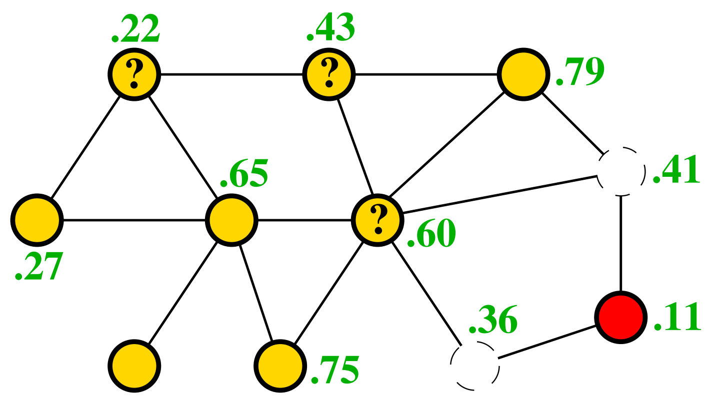
Our Method

Nguyen, O. (2008)

Main idea:

- select maximal independent set greedily
- consider vertices in random order

Random order $\equiv$ random numbers $r(v)$ assigned to each vertex



To check if $v \in \mathcal{I}$

- recursively check if neighbors w s.t. $r(w) < r(v)$ are in $\mathcal{I}$
- $v \in \mathcal{I} \iff$ none of them in $\mathcal{I}$

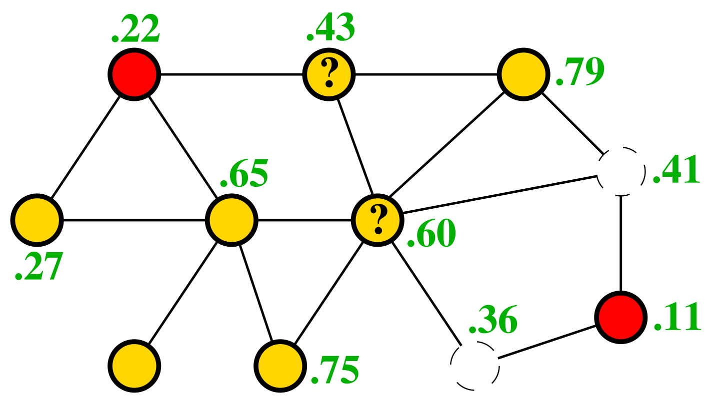
Our Method

Nguyen, O. (2008)

Main idea:

- select maximal independent set greedily
- consider vertices in random order

Random order $\equiv$ random numbers $r(v)$ assigned to each vertex



To check if $v \in \mathcal{I}$

- recursively check if neighbors w s.t. $r(w) < r(v)$ are in $\mathcal{I}$
- $v \in \mathcal{I} \iff$ none of them in $\mathcal{I}$

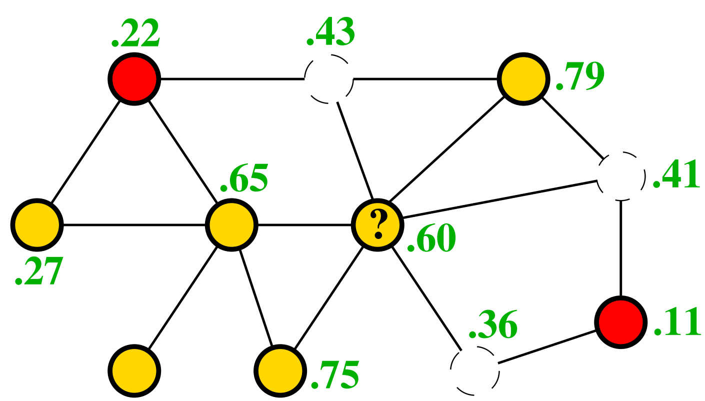
Our Method

Nguyen, O. (2008)

Main idea:

- select maximal independent set greedily
- consider vertices in random order

Random order $\equiv$ random numbers $r(v)$ assigned to each vertex



To check if $v \in \mathcal{I}$

- recursively check if neighbors w s.t. $r(w) < r(v)$ are in $\mathcal{I}$
- $v \in \mathcal{I} \iff$ none of them in $\mathcal{I}$

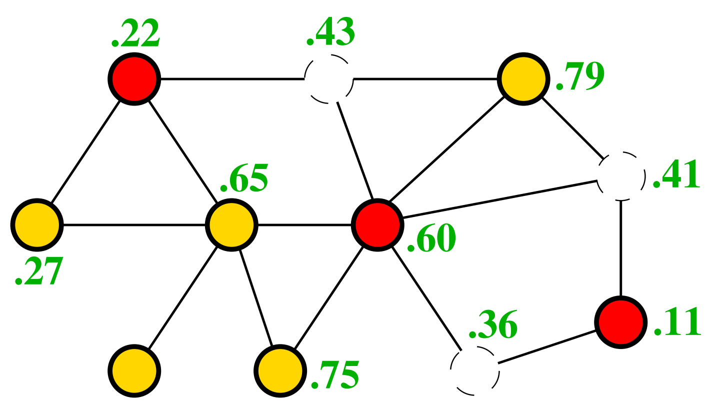
Our Method

Nguyen, O. (2008)

Main idea:

- select maximal independent set greedily
- consider vertices in random order

Random order $\equiv$ random numbers $r(v)$ assigned to each vertex



To check if $v \in \mathcal{I}$

- recursively check if neighbors w s.t. $r(w) < r(v)$ are in $\mathcal{I}$
- $v \in \mathcal{I} \iff$ none of them in $\mathcal{I}$

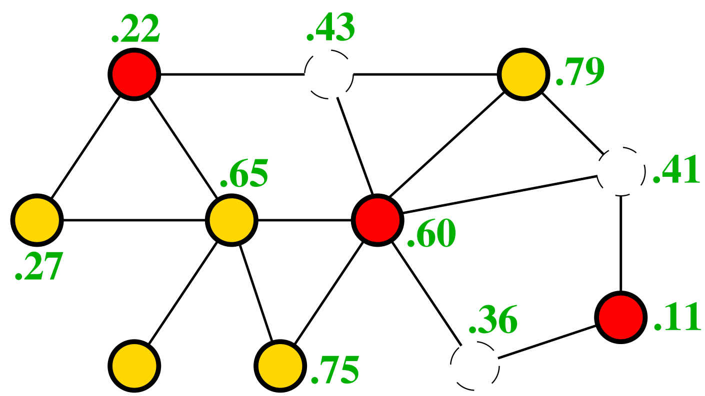
Our Method

Nguyen, O. (2008)

Main idea:

- select maximal independent set greedily
- consider vertices in random order

Random order $\equiv$ random numbers $r(v)$ assigned to each vertex



To check if $v \in \mathcal{I}$

- recursively check if neighbors w s.t. $r(w) < r(v)$ are in $\mathcal{I}$
- $v \in \mathcal{I} \iff$ none of them in $\mathcal{I}$

$\mathrm{E}[$#visited vertices$]$ and query complexity of order $2^{O(d)}$

Improvement for Random Query

Yoshida, Yamamoto, Ito (STOC 2009)

Heuristic:

- Consider neighbors w of v in ascending order of $r(w)$
- Once you find $w \in \mathcal{I}$, $v \notin \mathcal{I}$
  (i.e., don't check other neighbors)

Improvement for Random Query

Yoshida, Yamamoto, Ito (STOC 2009)

Heuristic:

- Consider neighbors w of v in ascending order of $r(w)$
- Once you find $w \in \mathcal{I}$, $v \notin \mathcal{I}$
  (i.e., don't check other neighbors)

They show:

$$\mathrm{E}_{\text{permutations, start vertex}}[\text{\#recursive calls}] \leq 1 + \frac{m}{n}$$

Improvement for Random Query

Yoshida, Yamamoto, Ito (STOC 2009)

Heuristic:

- Consider neighbors w of v in ascending order of $r(w)$
- Once you find $w \in \mathcal{I}$, $v \notin \mathcal{I}$
  (i.e., don't check other neighbors)

They show:

$$\mathrm{E}_{\text{permutations, start vertex}}[\text{\#recursive calls}] \leq 1 + \frac{m}{n}$$

Which gives:

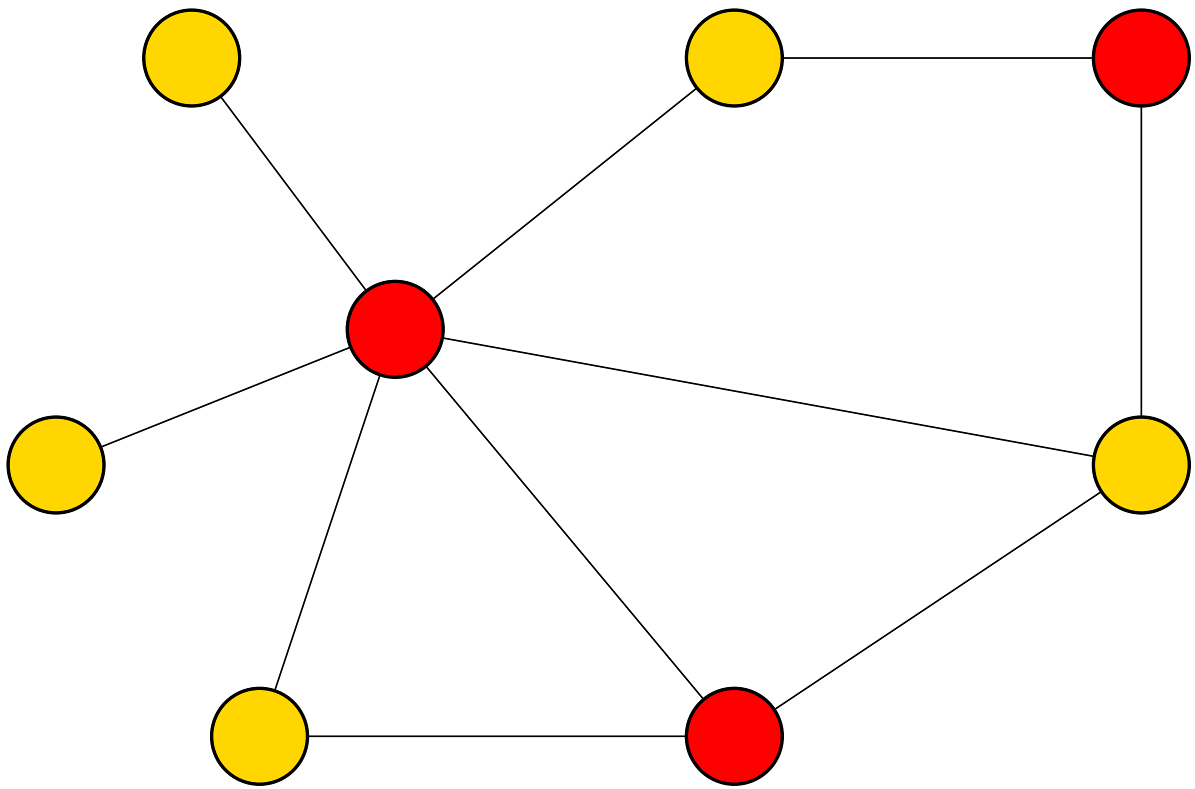
expected query complexity for random vertex = $O(d^2)$

Simplest Application: Vertex Cover

Vertex Cover

Graph $G = (V, E)$

Goal: find smallest set S of nodes such that each edge has endpoint in S



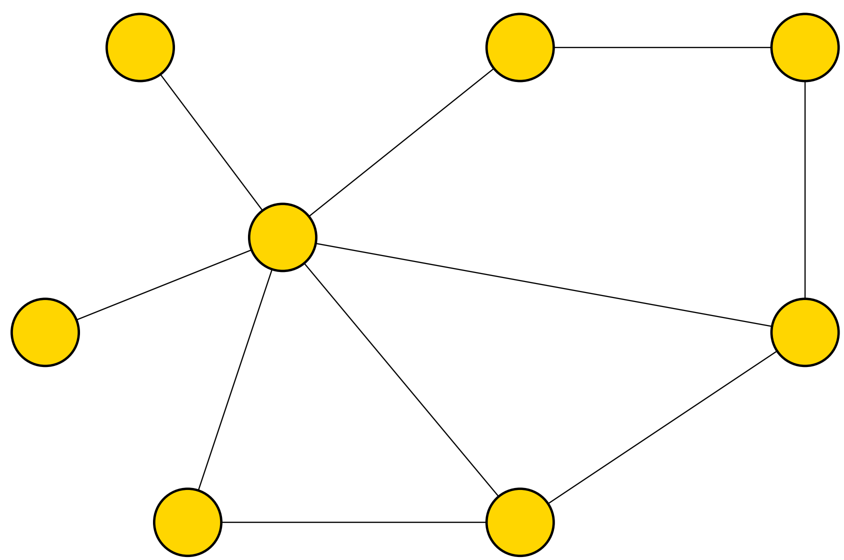
Vertex Cover

Graph $G = (V, E)$

Goal: find smallest set S of nodes such that each edge has endpoint in S

Classical 2-approximation algorithm [Gavril]:

- Greedily find a maximal matching M
- Output the set of nodes matched in M



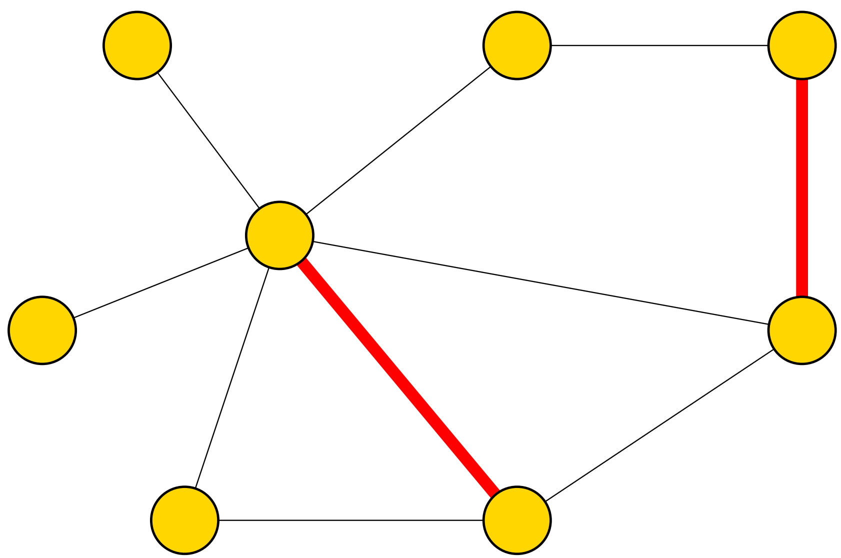
Vertex Cover

Graph $G = (V, E)$

Goal: find smallest set S of nodes such that each edge has endpoint in S

Classical 2-approximation algorithm [Gavril]:

- Greedily find a maximal matching M
- Output the set of nodes matched in M



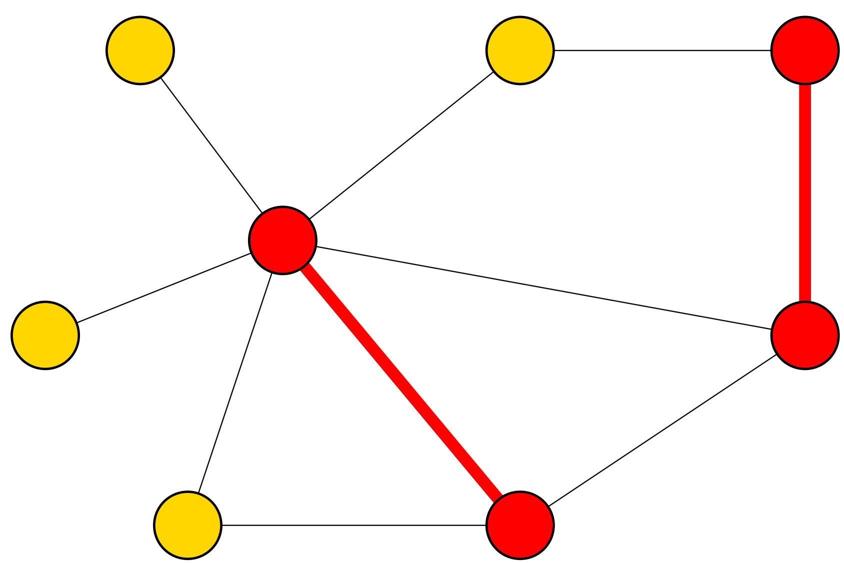
Vertex Cover

Graph $G = (V, E)$

Goal: find smallest set S of nodes such that each edge has endpoint in S

Classical 2-approximation algorithm [Gavril]:

- Greedily find a maximal matching M
- Output the set of nodes matched in M



Sublinear-Time Algorithm

Idea of Parnas and Ron (2007):

- construct oracle $\mathcal{O}$ that answers queries: Is $e \in E$ in M? for a fixed maximal matching M

Sublinear-Time Algorithm

Idea of Parnas and Ron (2007):

- construct oracle $\mathcal{O}$ that answers queries: Is $e \in E$ in M? for a fixed maximal matching M
- approximate the number of vertices matched in M up to $\pm\epsilon n$ by checking for $O(1/\epsilon^2)$ vertices if they are matched

$$\text{\#queries to } \mathcal{O} = (\text{\#tested nodes}) \cdot (\text{max-degree}) = O(d/\epsilon^2)$$

Sublinear-Time Algorithm

Idea of Parnas and Ron (2007):

- construct oracle $\mathcal{O}$ that answers queries: Is $e \in E$ in M? for a fixed maximal matching M
- approximate the number of vertices matched in M up to $\pm\epsilon n$ by checking for $O(1/\epsilon^2)$ vertices if they are matched

$$\text{\#queries to } \mathcal{O} = (\text{\#tested nodes}) \cdot (\text{max-degree}) = O(d/\epsilon^2)$$

Approximation notion:

Y is an (α, β)-approximation to X if $X \leq Y \leq \alpha \cdot X + \beta$

Sublinear-Time Algorithm

Idea of Parnas and Ron (2007):

- construct oracle $\mathcal{O}$ that answers queries: Is $e \in E$ in M? for a fixed maximal matching M
- approximate the number of vertices matched in M up to $\pm\epsilon n$ by checking for $O(1/\epsilon^2)$ vertices if they are matched

$$\#\text{queries to } \mathcal{O} = (\#\text{tested nodes}) \cdot (\text{max-degree}) = O(d/\epsilon^2)$$

Approximation notion:
Y is an (α, β)-approximation to X if $X \le Y \le \alpha \cdot X + \beta$

$(2, \epsilon n)$-approximation:
Simulate $\mathcal{O}$ using our method

Query Complexity

Parnas, Ron (2007):

- oracles via simulation of local distributed algorithms
- used Kuhn, Moscibroda, Wattenhofer (2006)
- $\forall c > 2$, $(c, \epsilon n)$-approximation with $d^{O(\log(d))}/\epsilon^2$ queries
- $(2, \epsilon n)$-approximation with $d^{O(\log(d)/\epsilon^3)}$ queries

t communication rounds $\Rightarrow d^{O(t)}$ queries

Query Complexity

Parnas, Ron (2007):

- oracles via simulation of local distributed algorithms
- used Kuhn, Moscibroda, Wattenhofer (2006)
- $\forall c > 2$, $(c, \epsilon n)$-approximation with $d^{O(\log(d))}/\epsilon^2$ queries
- $(2, \epsilon n)$-approximation with $d^{O(\log(d)/\epsilon^3)}$ queries

Marko, Ron (2007) using Luby's algorithm:

- $(2, \epsilon n)$-approximation with $d^{O(\log(d/\epsilon))}$ queries

Query Complexity

Parnas, Ron (2007):

- oracles via simulation of local distributed algorithms
- used Kuhn, Moscibroda, Wattenhofer (2006)
- $\forall c > 2$, $(c, \epsilon n)$-approximation with $d^{O(\log(d))}/\epsilon^2$ queries
- $(2, \epsilon n)$-approximation with $d^{O(\log(d)/\epsilon^3)}$ queries

Marko, Ron (2007) using Luby's algorithm:

- $(2, \epsilon n)$-approximation with $d^{O(\log(d/\epsilon))}$ queries

Nguyen, O. (2008):

- $(2, \epsilon n)$-approximation with $2^{O(d)}/\epsilon^2$ queries

Query Complexity

Parnas, Ron (2007):

- oracles via simulation of local distributed algorithms
- used Kuhn, Moscibroda, Wattenhofer (2006)
- $\forall c > 2$, $(c, \epsilon n)$-approximation with $d^{O(\log(d))}/\epsilon^2$ queries
- $(2, \epsilon n)$-approximation with $d^{O(\log(d)/\epsilon^3)}$ queries

Marko, Ron (2007) using Luby's algorithm:

- $(2, \epsilon n)$-approximation with $d^{O(\log(d/\epsilon))}$ queries

Nguyen, O. (2008):

- $(2, \epsilon n)$-approximation with $2^{O(d)}/\epsilon^2$ queries

Yoshida, Yamamoto, Ito (2009) using our suggestion:

- $(2, \epsilon n)$-approximation with $O(d^3/\epsilon^2)$ queries

Lower Bounds

Trevisan 2007:

- $(c, \epsilon n)$-approximation requires $\Omega(\sqrt{n})$ queries for $c < 2$

Lower Bounds

Trevisan 2007:

- $(c, \epsilon n)$-approximation requires $\Omega(\sqrt{n})$ queries for $c < 2$

Parnas, Ron 2007:

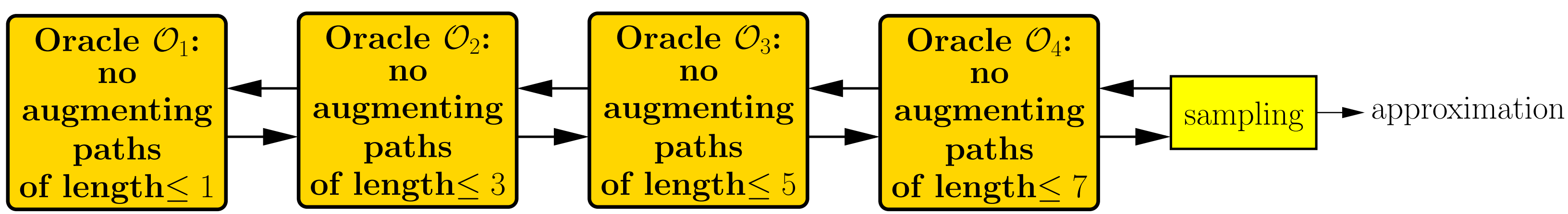
- $(O(1), \epsilon n)$-approximation requires $\Omega(d)$ queries

Other Problems

Maximum Matching Size

$(1, \epsilon n)$-approximation for maximum matching size

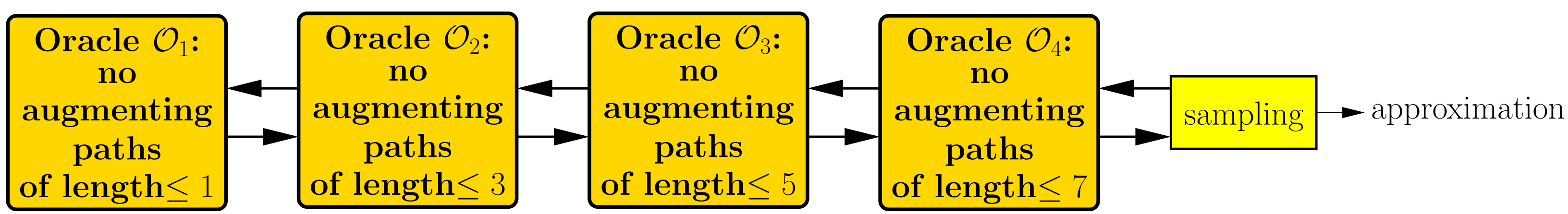
- Construct an oracle for a matching with no augmenting paths of length $\Theta(1/\epsilon)$
- Can be achieved by a sequence of oracles, where each oracle improves the matching from the previous oracle
- Each improvement corresponds to a maximal set of augmenting paths



Maximum Matching Size

$(1, \epsilon n)$-approximation for maximum matching size

- Construct an oracle for a matching with no augmenting paths of length $\Theta(1/\epsilon)$
- Can be achieved by a sequence of oracles, where each oracle improves the matching from the previous oracle
- Each improvement corresponds to a maximal set of augmenting paths



Query complexity and running time:

- Our analysis: $2^{d^{O(1/\epsilon)}}$
- Yoshida, Yamamoto, Ito (2009): $d^{O(1/\epsilon^2)}$

Set Cover and Dominating Set

Set Cover:

- Assumption:
  - each element in at most $t = O(1)$ of n sets
  - each set has at most $s = O(1)$ elements
- Guarantee: $(1 + \ln s, \epsilon n)$-approximation
- How: use classical greedy algorithm
- Complexity: function of s, t, and ϵ

Set Cover and Dominating Set

Set Cover:

- Assumption:
  - each element in at most $t = O(1)$ of n sets
  - each set has at most $s = O(1)$ elements
- Guarantee: $(1 + \ln s, \epsilon n)$-approximation
- How: use classical greedy algorithm
- Complexity: function of s, t, and ϵ

Dominating Set:

- $(1 + \ln(d+1), \epsilon n)$-approximation in time $\mathrm{function}(d, \epsilon)$

Set Cover and Dominating Set

Set Cover:

- Assumption:
  - each element in at most $t = O(1)$ of n sets
  - each set has at most $s = O(1)$ elements
- Guarantee: $(1 + \ln s, \epsilon n)$-approximation
- How: use classical greedy algorithm
- Complexity: function of s, t, and ϵ

Dominating Set:

- $(1 + \ln(d + 1), \epsilon n)$-approximation in time $\mathrm{function}(d, \epsilon)$
- $(O(\log d), \epsilon n)$-approximation known before via Parnas, Ron (2007) + Kuhn, Moscibroda, Wattenhofer (2006)

Set Cover and Dominating Set

Set Cover:

- Assumption:
  - each element in at most $t = O(1)$ of n sets
  - each set has at most $s = O(1)$ elements
- Guarantee: $(1 + \ln s, \epsilon n)$-approximation
- How: use classical greedy algorithm
- Complexity: function of s, t, and ϵ

Dominating Set:

- $(1 + \ln(d+1), \epsilon n)$-approximation in time $\mathrm{function}(d, \epsilon)$
- $(O(\log d), \epsilon n)$-approximation known before via Parnas, Ron (2007) + Kuhn, Moscibroda, Wattenhofer (2006)
- Alon: $\Omega(\log n)$ queries to $(o(\log d), \epsilon n)$-approximate

Maximum Matching

- Maximum Weight Matching:
  - Assumption: degree d and all weights in [0,1]
  - Guarantee: $(1, \epsilon n)$-approximation
  - How: use Pettie and Sanders (2004)
  - Complexity: function of d and ϵ

Maximum Matching

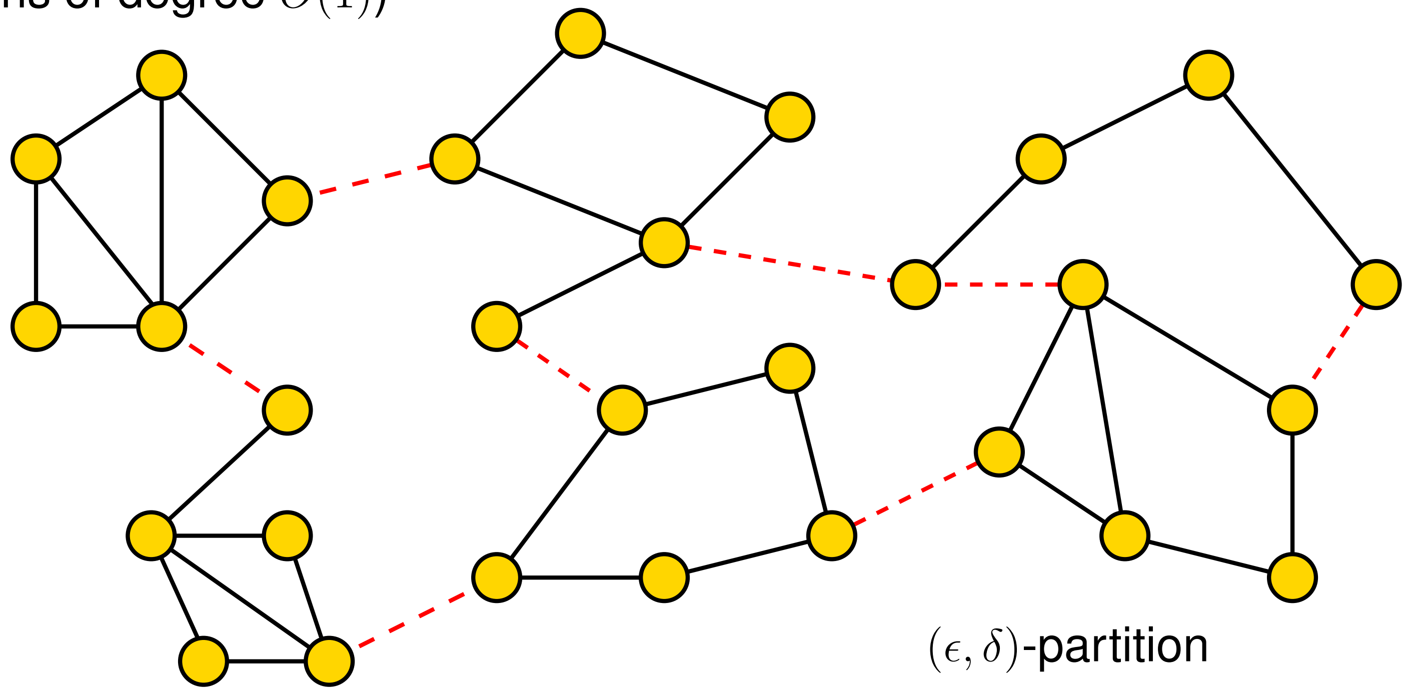
- Maximum Weight Matching:
  - Assumption: degree d and all weights in [0,1]
  - Guarantee: $(1, \epsilon n)$-approximation
  - How: use Pettie and Sanders (2004)
  - Complexity: function of d and ϵ
- Maximum Independent Set (Alon):
  - Upper bound:
    $\left(O\left(\frac{d \cdot \log\log d}{\log d}\right), \epsilon n\right)$-approximation in time $\mathrm{function}(d, \epsilon)$
  - Lower bound:
    $\Omega(\log n)$ queries to $\left(o\left(\frac{d}{\log d}\right), \epsilon n\right)$-approximate

Local Graph Partitions

[Hassidim, Kelner, Nguyen, O. 2009]

Hyperfinite Graphs

(All graphs of degree $O(1)$)

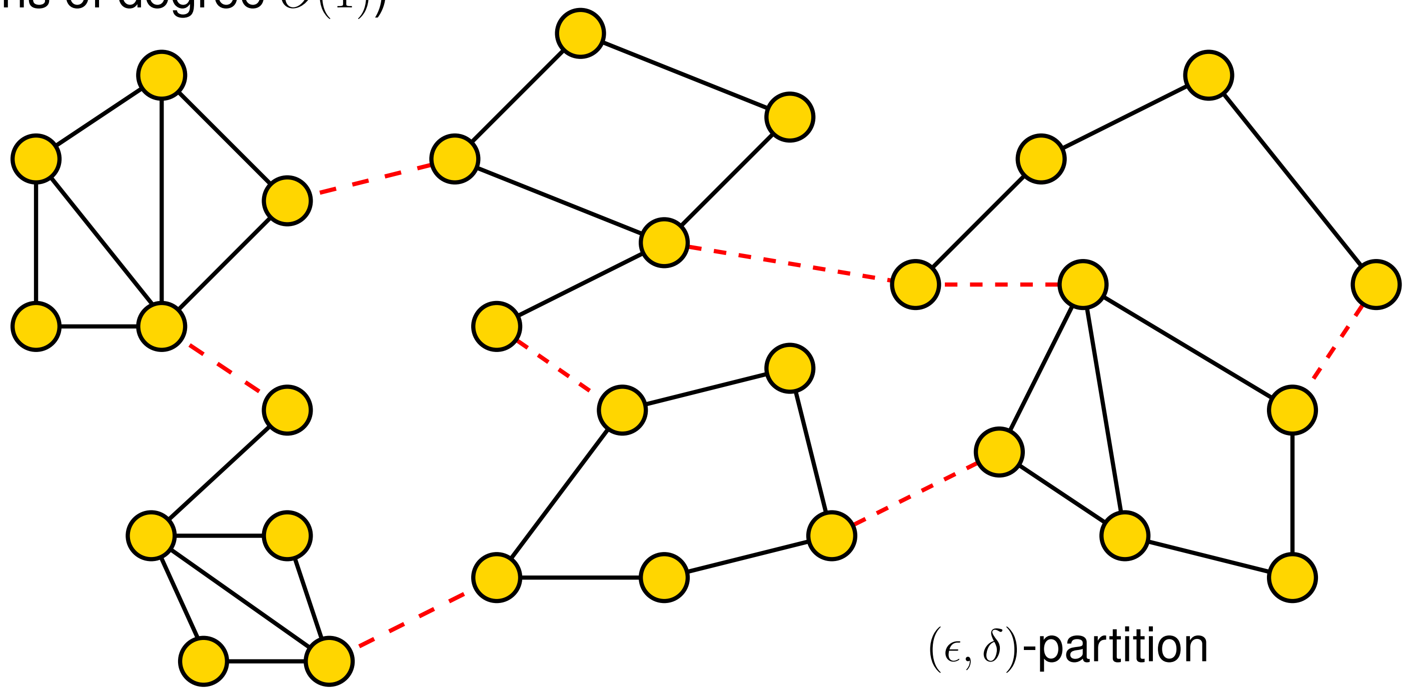


(ϵ, δ)-partition

- (ϵ, δ)-hyperfinite graphs: can remove $\epsilon|V|$ edges and get components of size at most δ

Hyperfinite Graphs

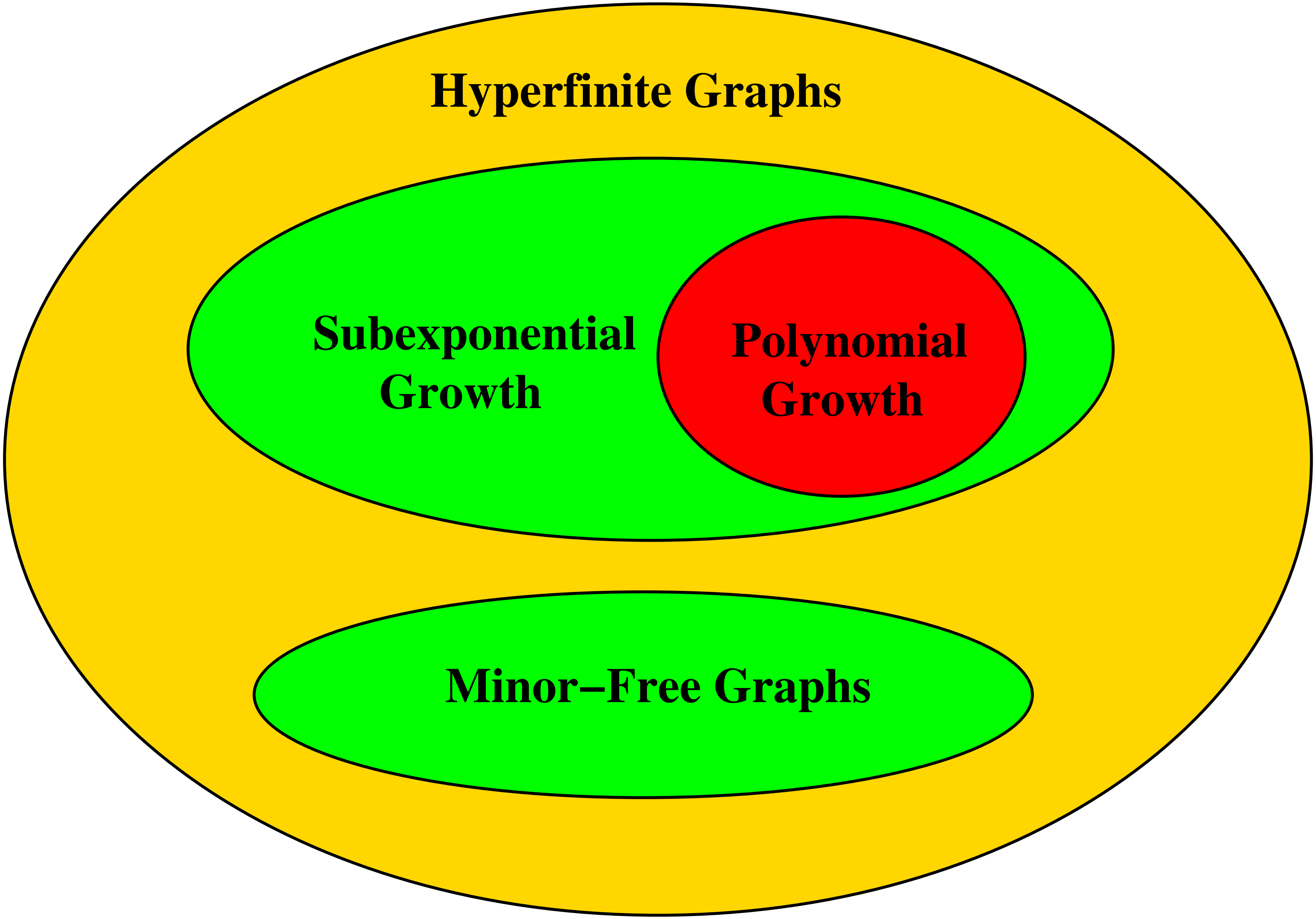
(All graphs of degree $O(1)$)



(ϵ, δ)-partition

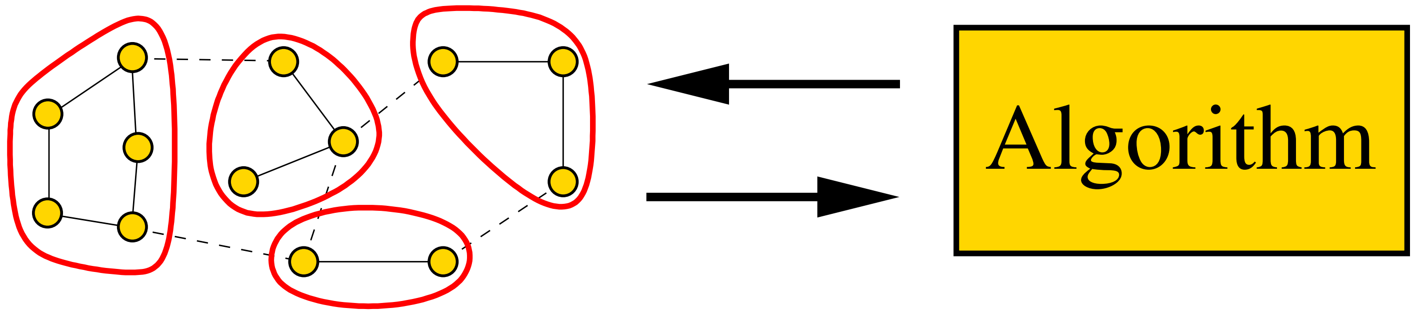
- (ϵ, δ)-hyperfinite graphs: can remove $\epsilon|V|$ edges and get components of size at most δ
- hyperfinite family of graphs: there is ρ such that all graphs are $(\epsilon, \rho(\epsilon))$-hyperfinite for all $\epsilon > 0$

Taxonomy


Hyperfinite Graphs
Subexponential Growth
Polynomial Growth
Minor–Free Graphs

Using a Partition

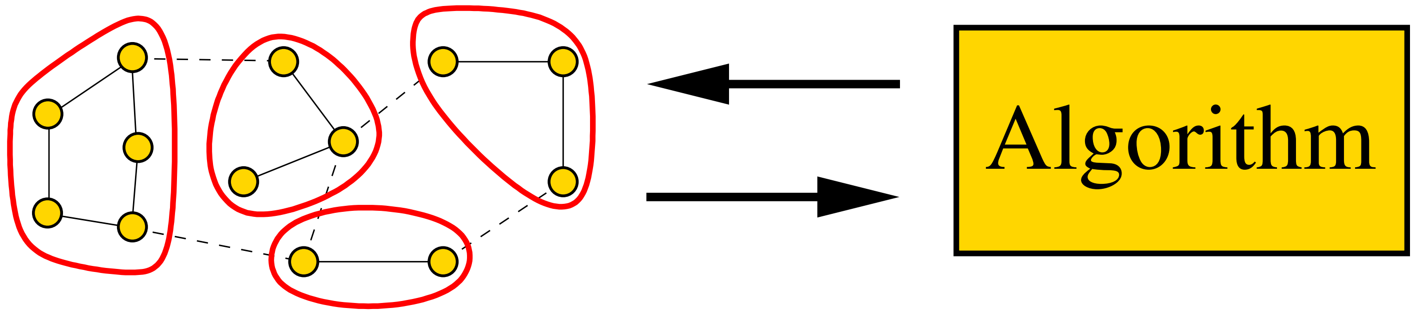
If someone gave us a $(\epsilon/2, \delta)$-partition:



- Sample $O(1/\epsilon^2)$ vertices
- Compute minimum vertex cover for the sampled components
- Return the fraction of the sampled vertices in the covers

Using a Partition

If someone gave us a $(\epsilon/2, \delta)$-partition:



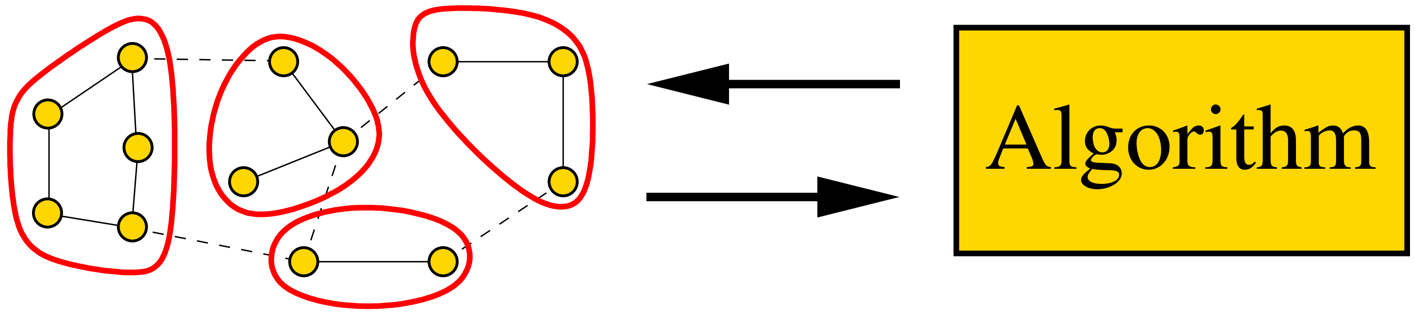
- Sample $O(1/\epsilon^2)$ vertices
- Compute minimum vertex cover for the sampled components
- Return the fraction of the sampled vertices in the covers

This gives $\pm\epsilon$ approximation to $\mathrm{VC}(G)/n$ in constant time:

- Cut edges change $\mathrm{VC}(G)$ by at most $\epsilon n/2$
- Can compute vertex cover separately for each component

Using a Partition

If someone gave us a $(\epsilon/2, \delta)$-partition:

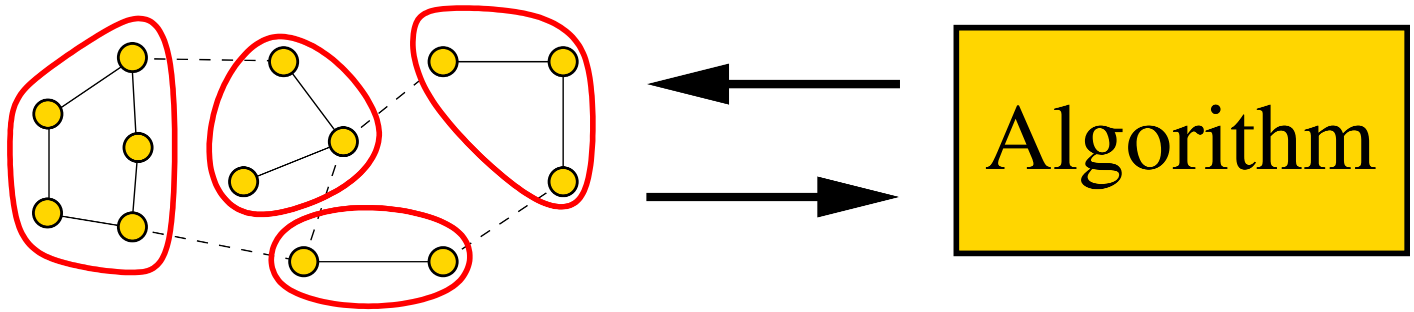


Bad news:

We don't have a partition

Using a Partition

If someone gave us a $(\epsilon/2, \delta)$-partition:



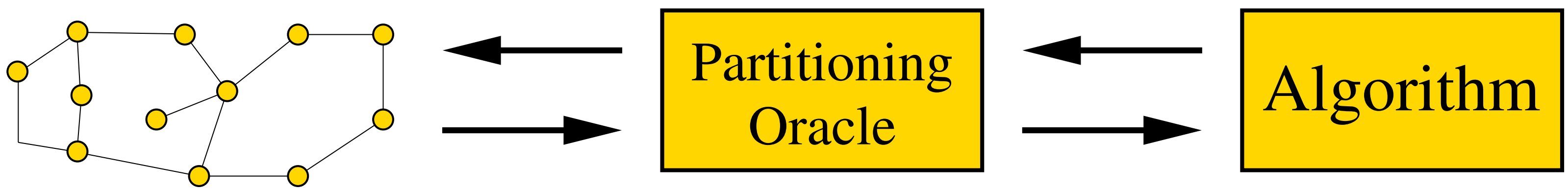
Bad news:

We don't have a partition

Good news:

We can compute it ourselves
without looking at the entire graph

Using a Partition



Bad news:

We don't have a partition

Good news:

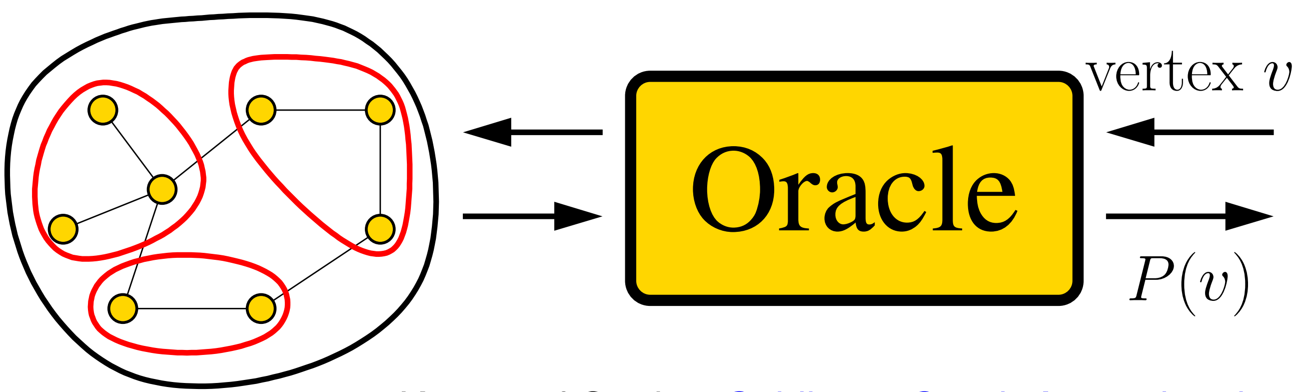
We can compute it ourselves
without looking at the entire graph

New Tool: Partitioning Oracles

Partitioning Oracle

$\mathcal{C} =$ fixed hyperfinite class

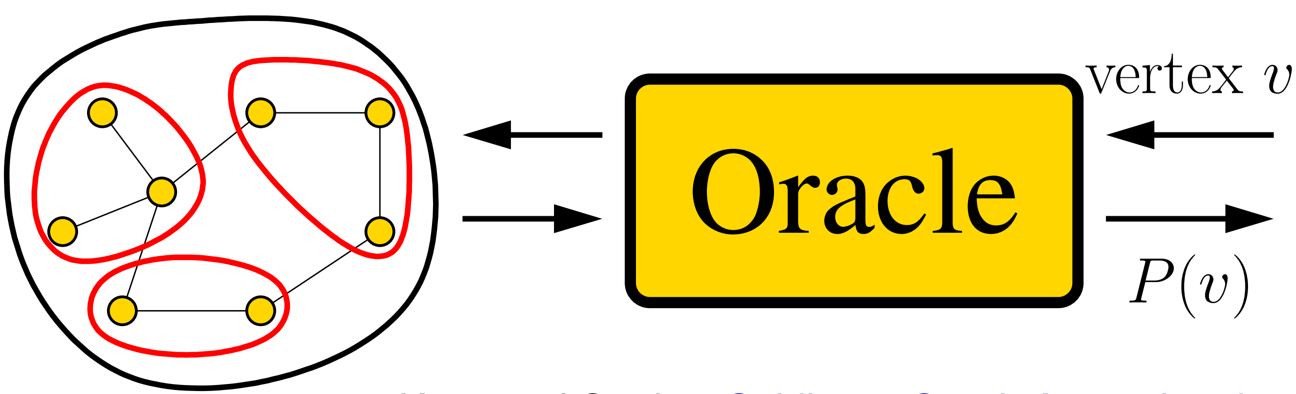
- oracle has query access to $G = (V, E)$
  (G need not be in $\mathcal{C}$)



Partitioning Oracle

$\mathcal{C} =$ fixed hyperfinite class

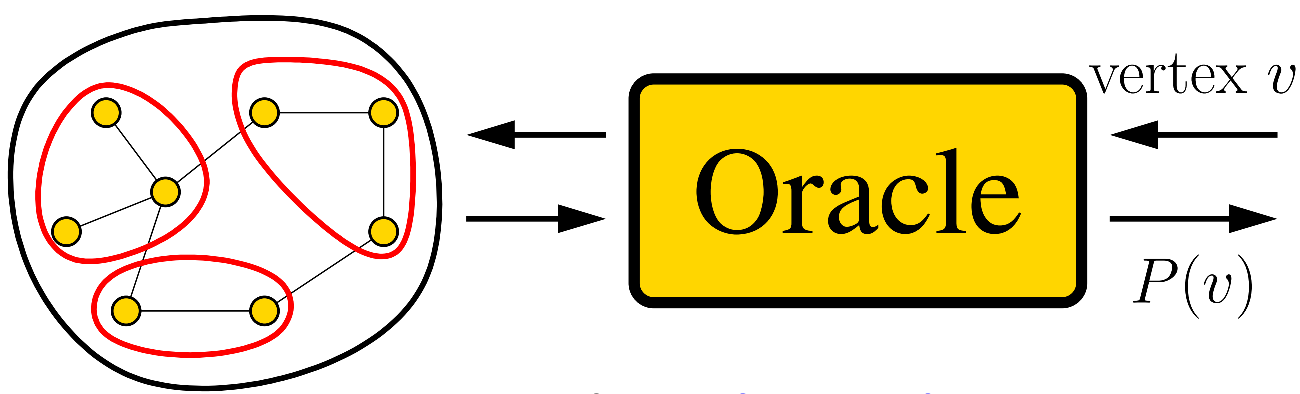
- oracle has query access to $G = (V, E)$
  (G need not be in $\mathcal{C}$)
- oracle provides query access to partition P of V;
  for each v, oracle returns $P(v) \subseteq V$ s.t. $v \in P(v)$



Partitioning Oracle

$\mathcal{C} =$ fixed hyperfinite class

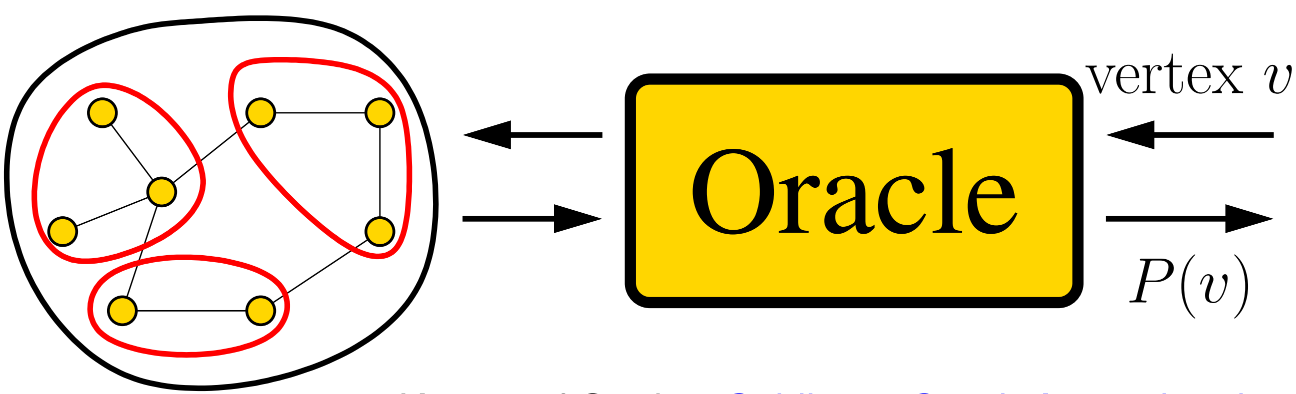
- oracle has query access to $G = (V, E)$
  (G need not be in $\mathcal{C}$)
- oracle provides query access to partition P of V; for each v, oracle returns $P(v) \subseteq V$ s.t. $v \in P(v)$
- Properties of P:
  - each $|P(v)| = O(1)$



Partitioning Oracle

$\mathcal{C} =$ fixed hyperfinite class

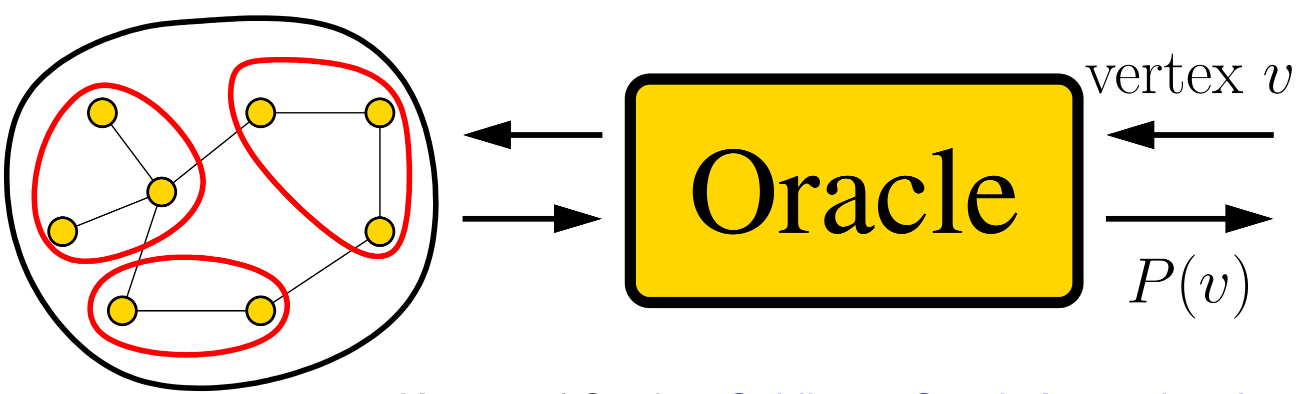
- oracle has query access to $G = (V, E)$
  (G need not be in $\mathcal{C}$)
- oracle provides query access to partition P of V; for each v, oracle returns $P(v) \subseteq V$ s.t. $v \in P(v)$
- Properties of P:
  - each $|P(v)| = O(1)$
  - If $G \in \mathcal{C}$, number of cut edges $\leq \epsilon n$ w.p. $\frac{99}{100}$



Partitioning Oracle

$\mathcal{C} =$ fixed hyperfinite class

- oracle has query access to $G = (V, E)$
  (G need not be in $\mathcal{C}$)
- oracle provides query access to partition P of V;
  for each v, oracle returns $P(v) \subseteq V$ s.t. $v \in P(v)$
- Properties of P:
  - each $|P(v)| = O(1)$
  - If $G \in \mathcal{C}$, number of cut edges $\leq \epsilon n$ w.p. $\frac{99}{100}$
  - partition $P(\cdot)$ is not a function of queries,
    it is a function of graph structure and random bits



Our Oracles

- Generic oracle for any hyperfinite class of graphs
  - Query complexity: $2^{d^{O(\rho(\epsilon^3/54000))}}$
  - Via local simulation of a greedy partitioning procedure (uses [Nguyen, O. 2008])

Our Oracles

- Generic oracle for any hyperfinite class of graphs
  - Query complexity: $2^{d^{O(\rho(\epsilon^3/54000))}}$
- For minor-free graphs:
  - Query complexity: $d^{\mathrm{poly}(1/\epsilon)}$
  - Via techniques from distributed algorithms [Czygrinow, Hańćkowiak, Wawrzyniak 2008]

Our Oracles

- Generic oracle for any hyperfinite class of graphs
  - Query complexity: $2^{d^{O(\rho(\epsilon^3/54000))}}$
- For minor-free graphs:
  - Query complexity: $d^{\mathrm{poly}(1/\epsilon)}$
- For $\rho(\epsilon) \leq \mathrm{poly}(1/\epsilon)$:
  - Query complexity: $2^{\mathrm{poly}(d/\epsilon)}$
  - Via methods from distributed algorithms and partitioning methods of Andersen and Peres (2009)

Our Oracles

- Generic oracle for any hyperfinite class of graphs
  - Query complexity: $2^{d^{O(\rho(\epsilon^3/54000))}}$
- For minor-free graphs:
  - Query complexity: $d^{\mathrm{poly}(1/\epsilon)}$
- For $\rho(\epsilon) \leq \mathrm{poly}(1/\epsilon)$:
  - Query complexity: $2^{\mathrm{poly}(d/\epsilon)}$
- Time complexity?
  - $Q =$ query complexity
  - $k =$ number of queries
  - Running time $= O(kQ \cdot \log(kQ))$

Three Applications

Three Applications

1. Approximation of graph parameters
   in hyperfinite graphs

Three Applications

1. Approximation of graph parameters
   in hyperfinite graphs

2. Testing minor-closed properties
   - Simpler proof of the result of Benjamini, Schramm, and Shapira (2008)

Three Applications

1. Approximation of graph parameters
   in hyperfinite graphs

2. Testing minor-closed properties
   - Simpler proof of the result of Benjamini, Schramm, and Shapira (2008)

3. Approximating distance to hereditary properties
   in hyperfinite graphs
   - Earlier only known to be testable [Czumaj, Shapira, Sohler 2009]

Application 1: Approximation

- For hyperfinite graphs, can get $\pm\epsilon n$ approximation to:
  - minimum vertex cover size
    (that is also the independence number)
  - minimum dominating set size

  in time independent of the graph size

Application 1: Approximation

- For hyperfinite graphs, can get $\pm\epsilon n$ approximation to:
  - minimum vertex cover size
    (that is also the independence number)
  - minimum dominating set size

  in time independent of the graph size
- Earlier/independent proofs of the same results
  - Elek 2009: for graphs with subexponential growth

Application 1: Approximation

- For hyperfinite graphs, can get $\pm\epsilon n$ approximation to:
  - minimum vertex cover size
    (that is also the independence number)
  - minimum dominating set size

  in time independent of the graph size
- Earlier/independent proofs of the same results
  - Elek 2009: for graphs with subexponential growth
  - Czygrinow, Hańćkowiak, Wawrzyniak (2008)
    + Parnas, Ron (2007): for minor-free graphs

Application 2: Testing

Testing H-minor-freeness in the sparse graph model of Goldreich and Ron (1997)

- Input: query access to constant degree graph G
  & parameter $\epsilon > 0$
- Goal: w.p. $2/3$
  - accept H-minor-free graphs
  - reject graphs far from H-minor-freeness: $\geq \epsilon n$ edges must be removed to achieve H-minor-freeness

Application 2: Testing

Testing H-minor-freeness in the sparse graph model of Goldreich and Ron (1997)

- Input: query access to constant degree graph G
  & parameter $\epsilon > 0$
- Goal: w.p. $2/3$
  - accept H-minor-free graphs
  - reject graphs far from H-minor-freeness: $\geq \epsilon n$ edges must be removed to achieve H-minor-freeness

Time and query complexity:

- Goldreich, Ron (1997): cycle-freeness in $\mathrm{poly}(1/\epsilon)$ time
- Benjamini, Schramm, Shapira (2008): any minor in $2^{2^{2^{\mathrm{poly}(1/\epsilon)}}}$ time

Application 2: Testing

Testing H-minor-freeness in the sparse graph model of Goldreich and Ron (1997)

- Input: query access to constant degree graph G
  & parameter $\epsilon > 0$
- Goal: w.p. $2/3$
  - accept H-minor-free graphs
  - reject graphs far from H-minor-freeness: $\geq \epsilon n$ edges must be removed to achieve H-minor-freeness

Time and query complexity:

- Goldreich, Ron (1997): cycle-freeness in $\mathrm{poly}(1/\epsilon)$ time
- Benjamini, Schramm, Shapira (2008): any minor in $2^{2^{2^{\mathrm{poly}(1/\epsilon)}}}$ time
- This work: $2^{\mathrm{poly}(1/\epsilon)}$ and simpler proof

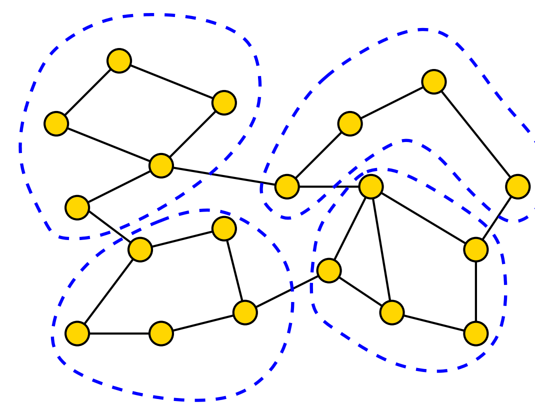
Application 2: Testing

Example: Testing planarity
(i.e., K_5- and $K_{3,3}$-minor-freeness)

Application 2: Testing

Example: Testing planarity
(i.e., K_5- and $K_{3,3}$-minor-freeness)

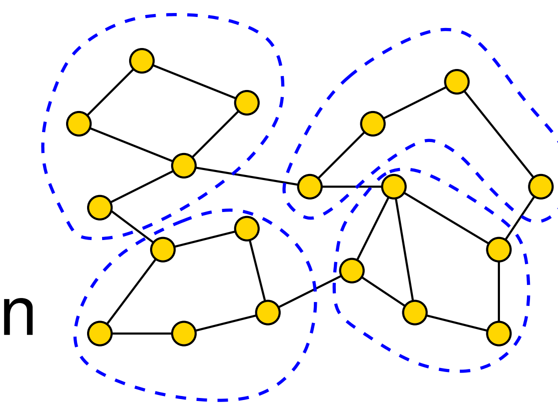
- Algorithm (given partitioning oracle for planar graphs that usually cuts $\leq \epsilon n/2$ edges):
  - Estimate the number of cut edges by sampling
  - If greater than $\epsilon n/2$, reject
  - Check a few random components if planar
  - If any non-planar found, reject
    otherwise, accept



Application 2: Testing

Example: Testing planarity
(i.e., K_5- and $K_{3,3}$-minor-freeness)

- Algorithm (given partitioning oracle for planar graphs that usually cuts $\leq \epsilon n/2$ edges):
  - Estimate the number of cut edges by sampling
  - If greater than $\epsilon n/2$, reject
  - Check a few random components if planar
  - If any non-planar found, reject
    otherwise, accept

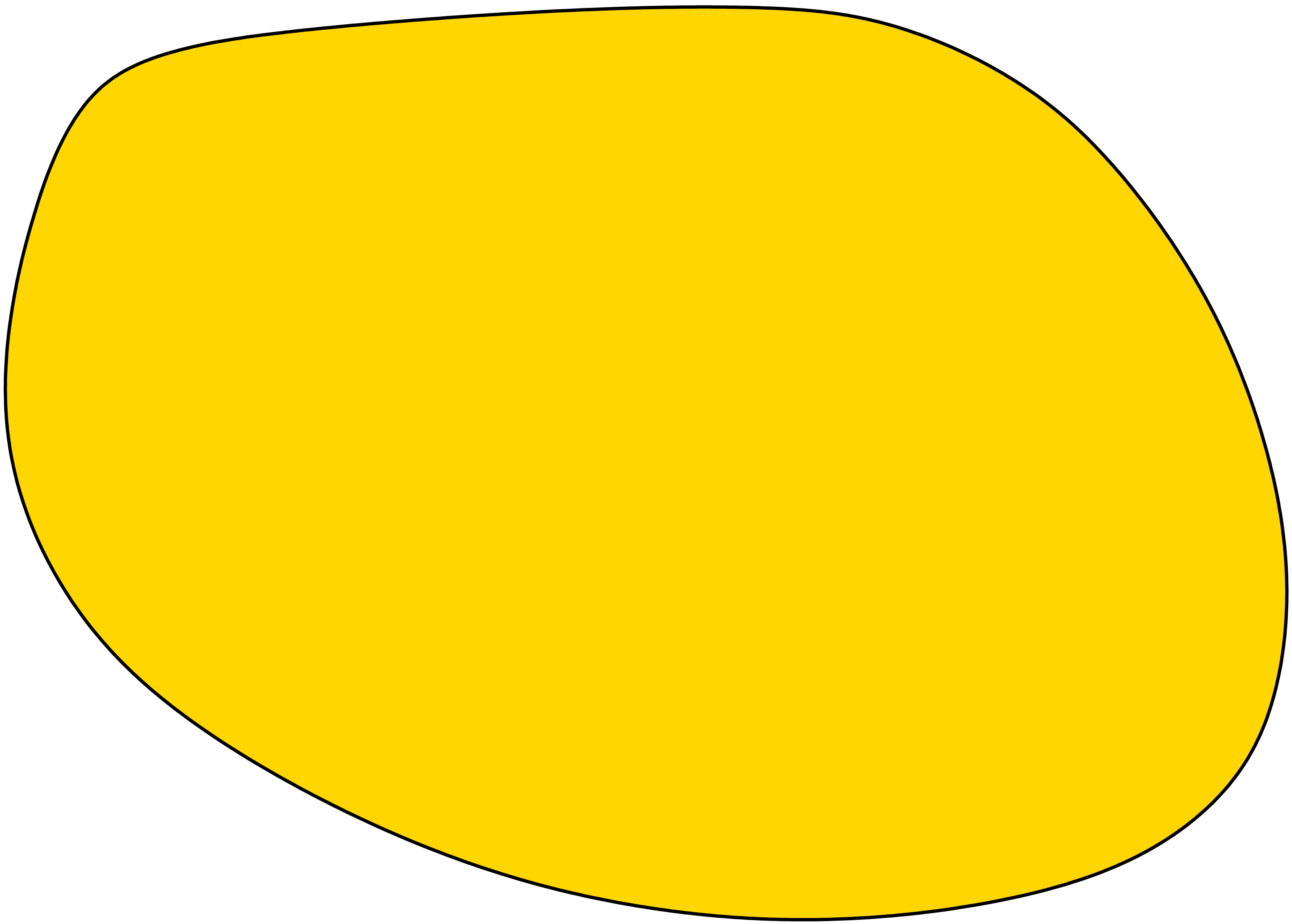


- Why it works:
  - planar: few edges cut in the partition
  - ϵ-far: either many edges cut
    or many copies of $K_{3,3}$ or K_5

Simplest Oracle

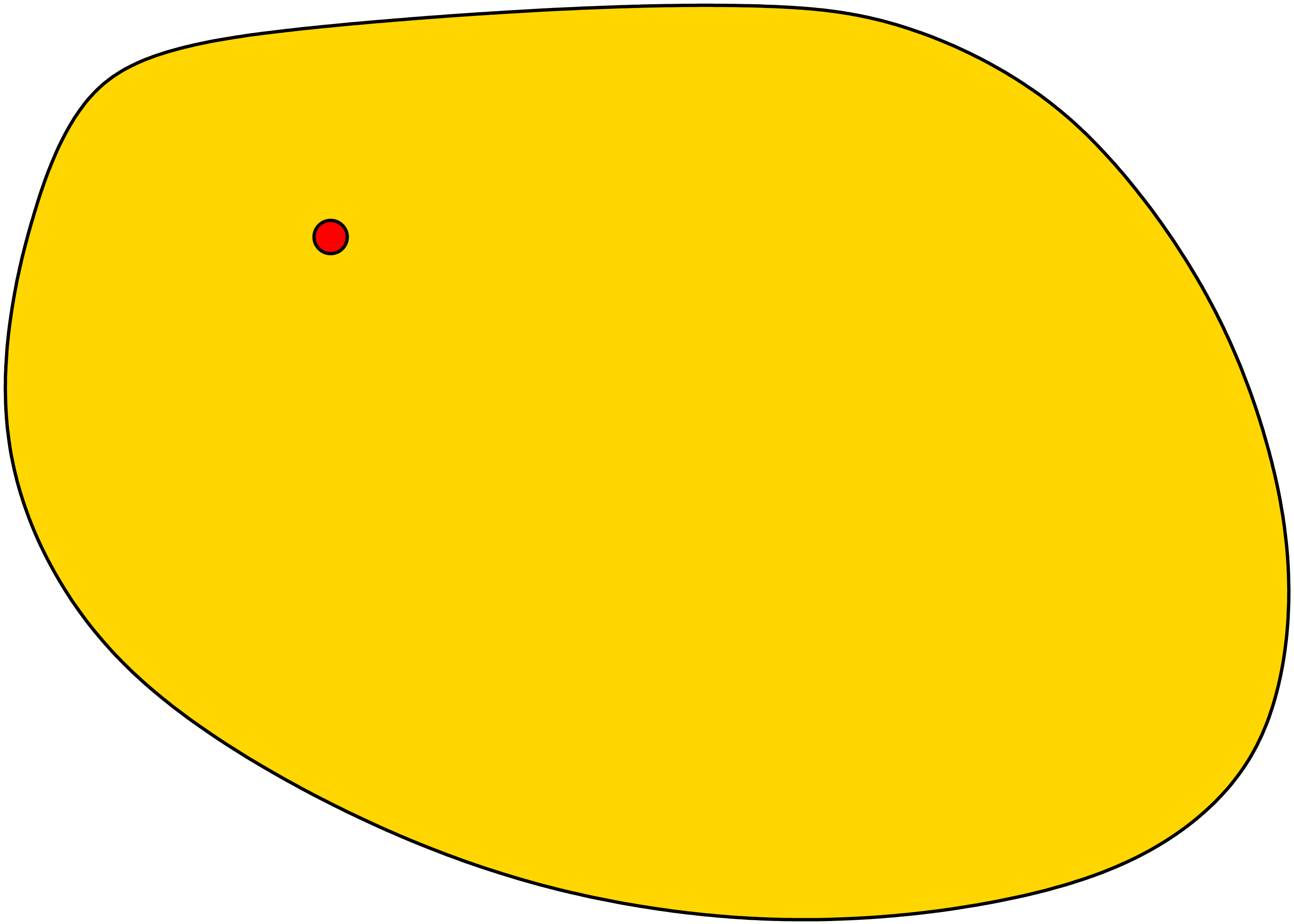
Iterative Procedure

Global procedure:



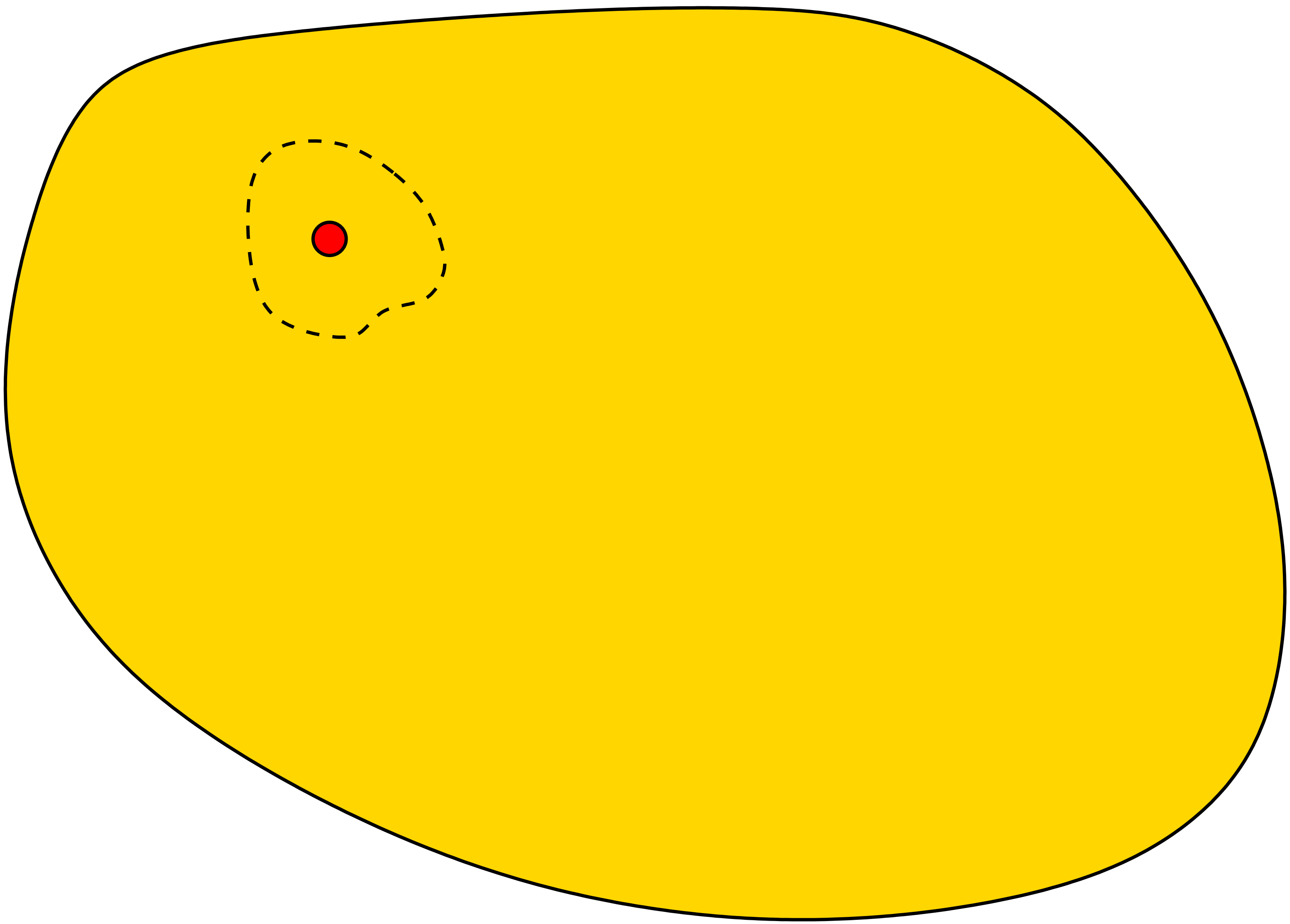
Iterative Procedure

Global procedure:



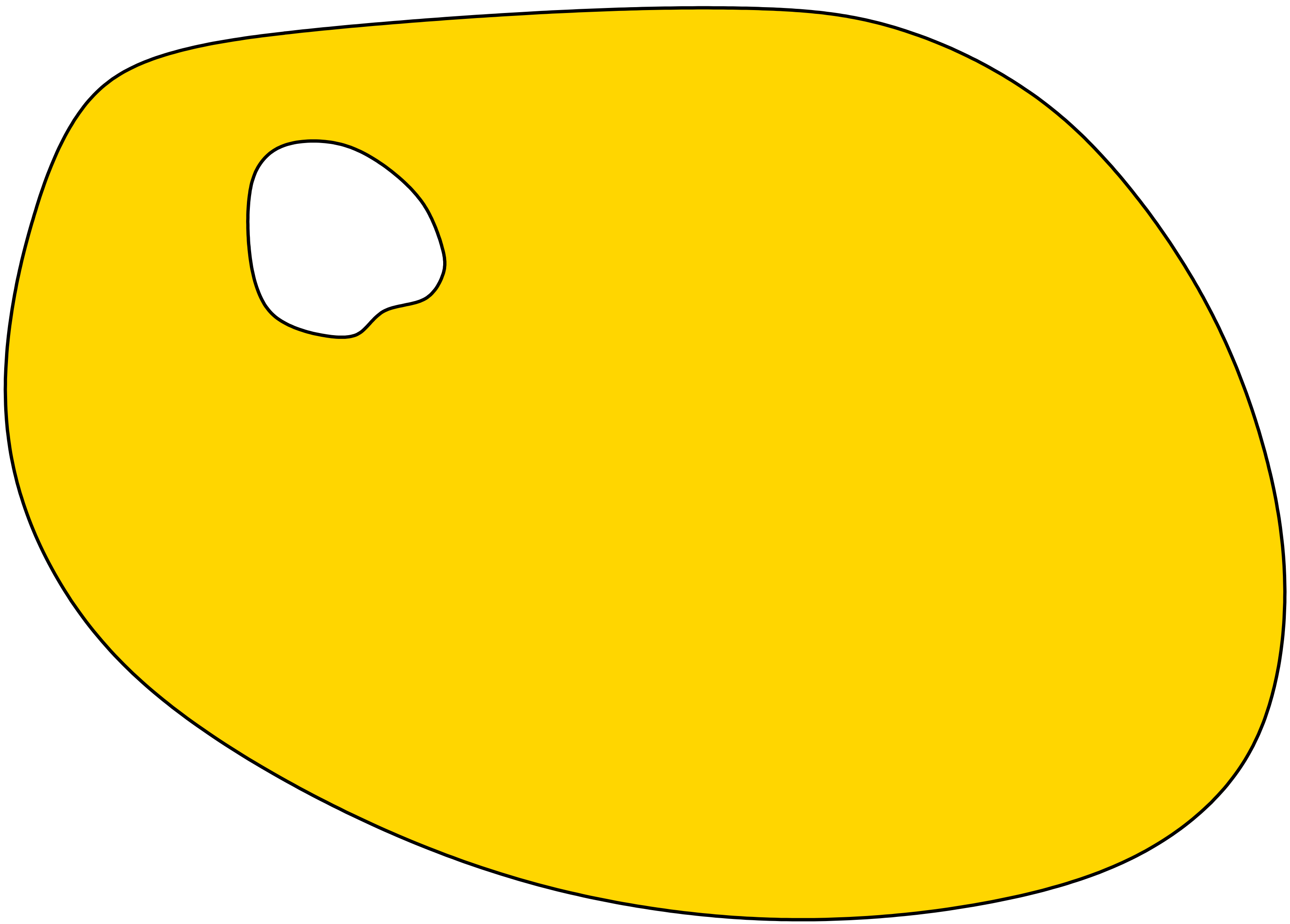
Iterative Procedure

Global procedure:



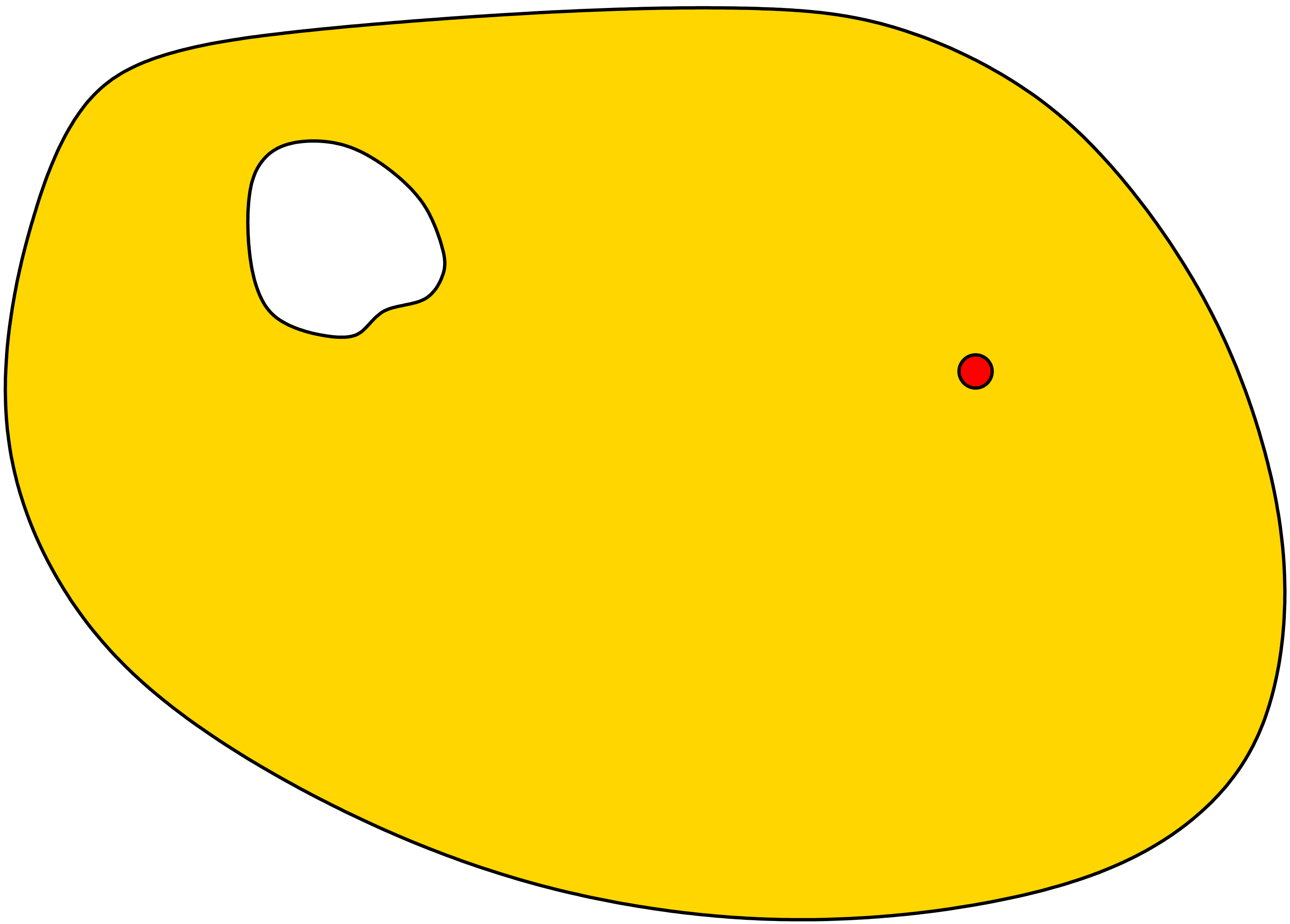
Iterative Procedure

Global procedure:



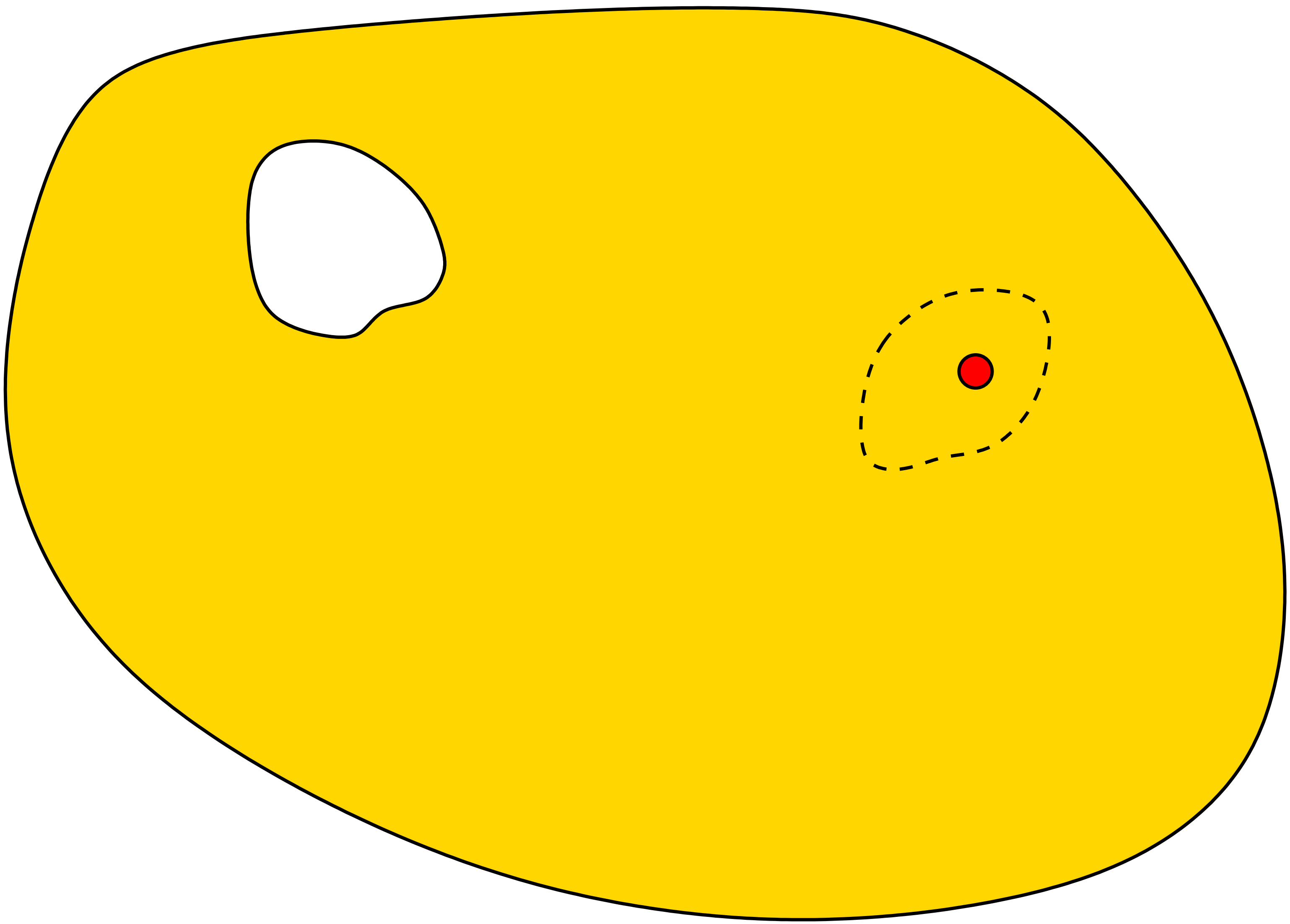
Iterative Procedure

Global procedure:



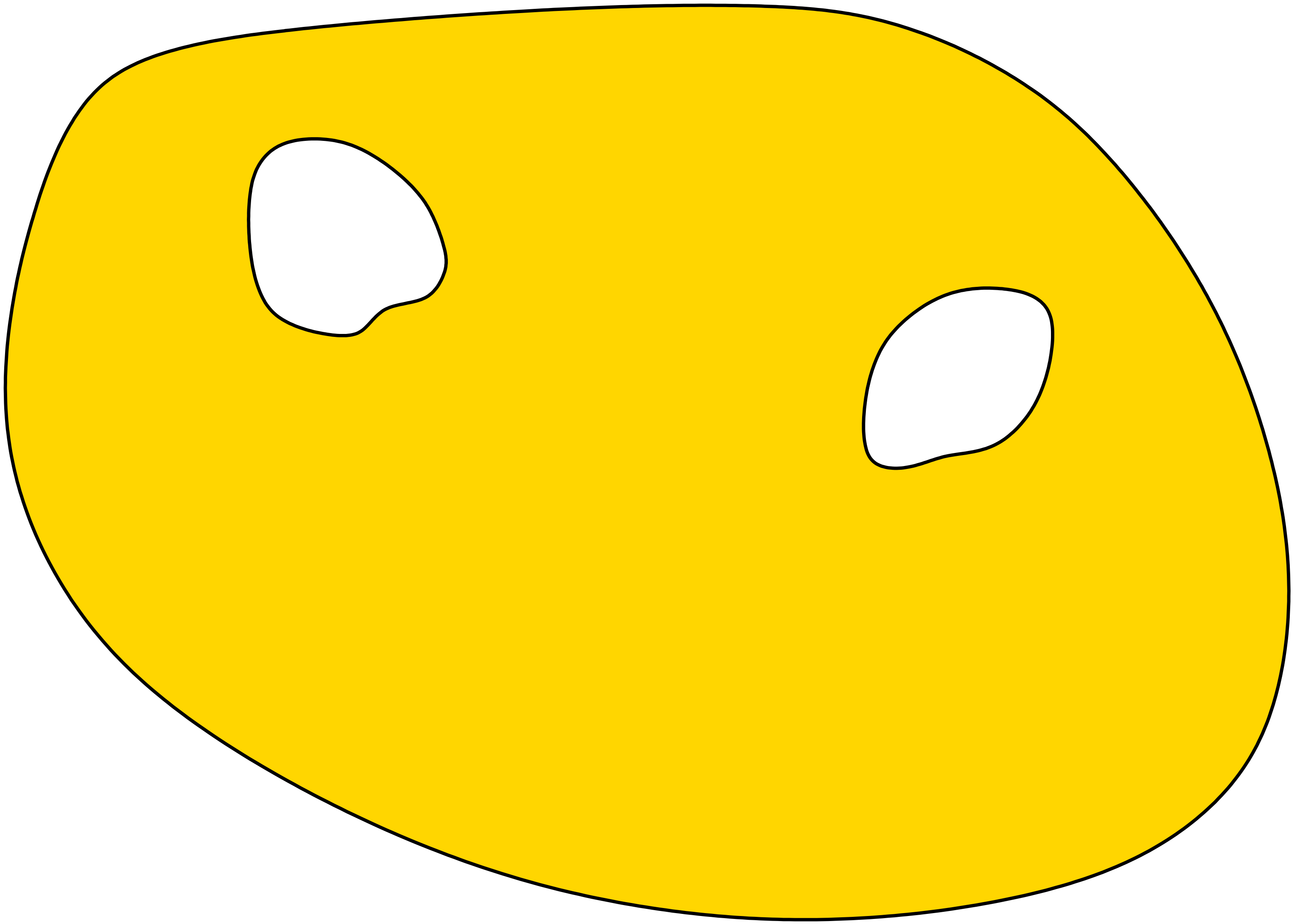
Iterative Procedure

Global procedure:



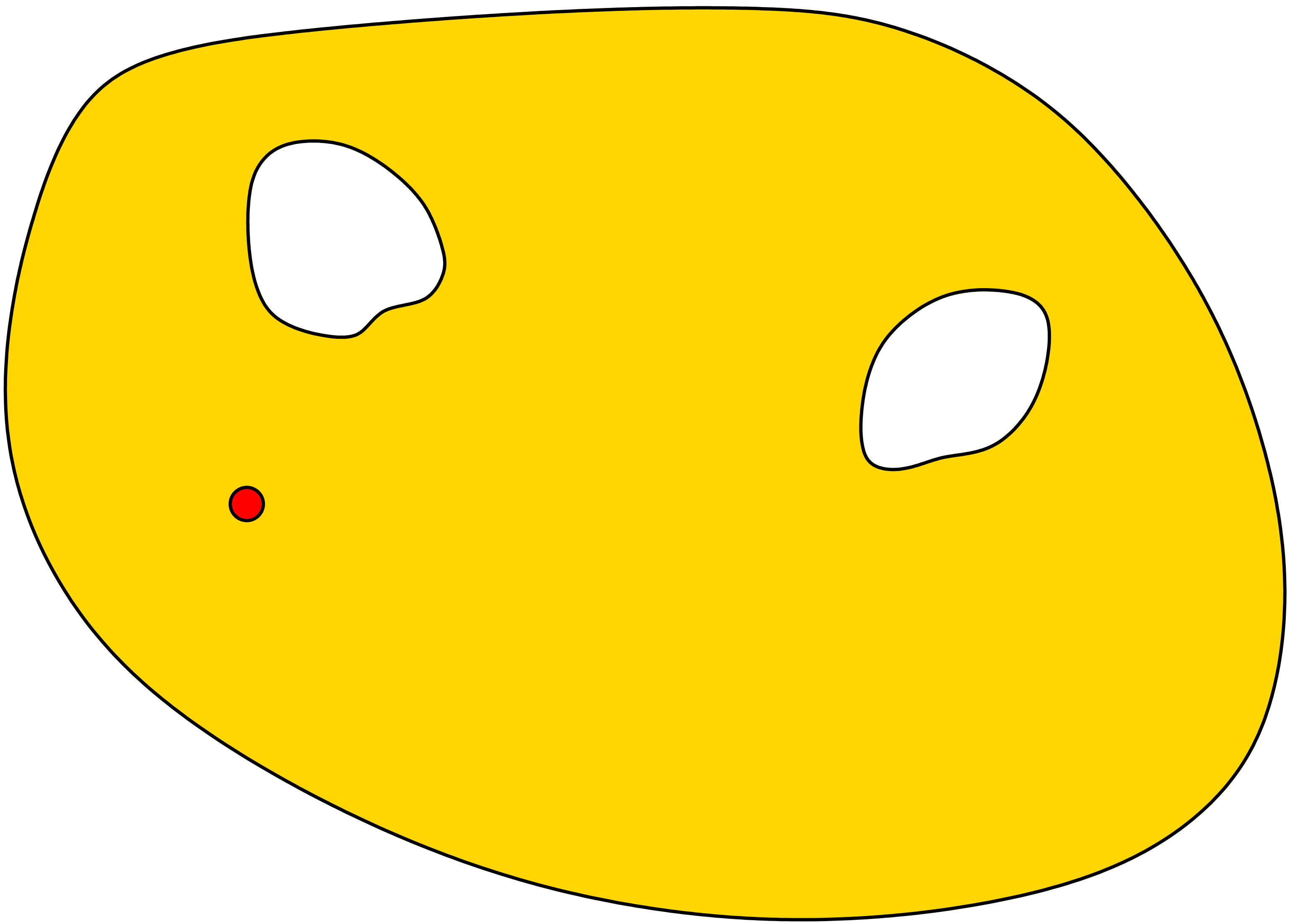
Iterative Procedure

Global procedure:



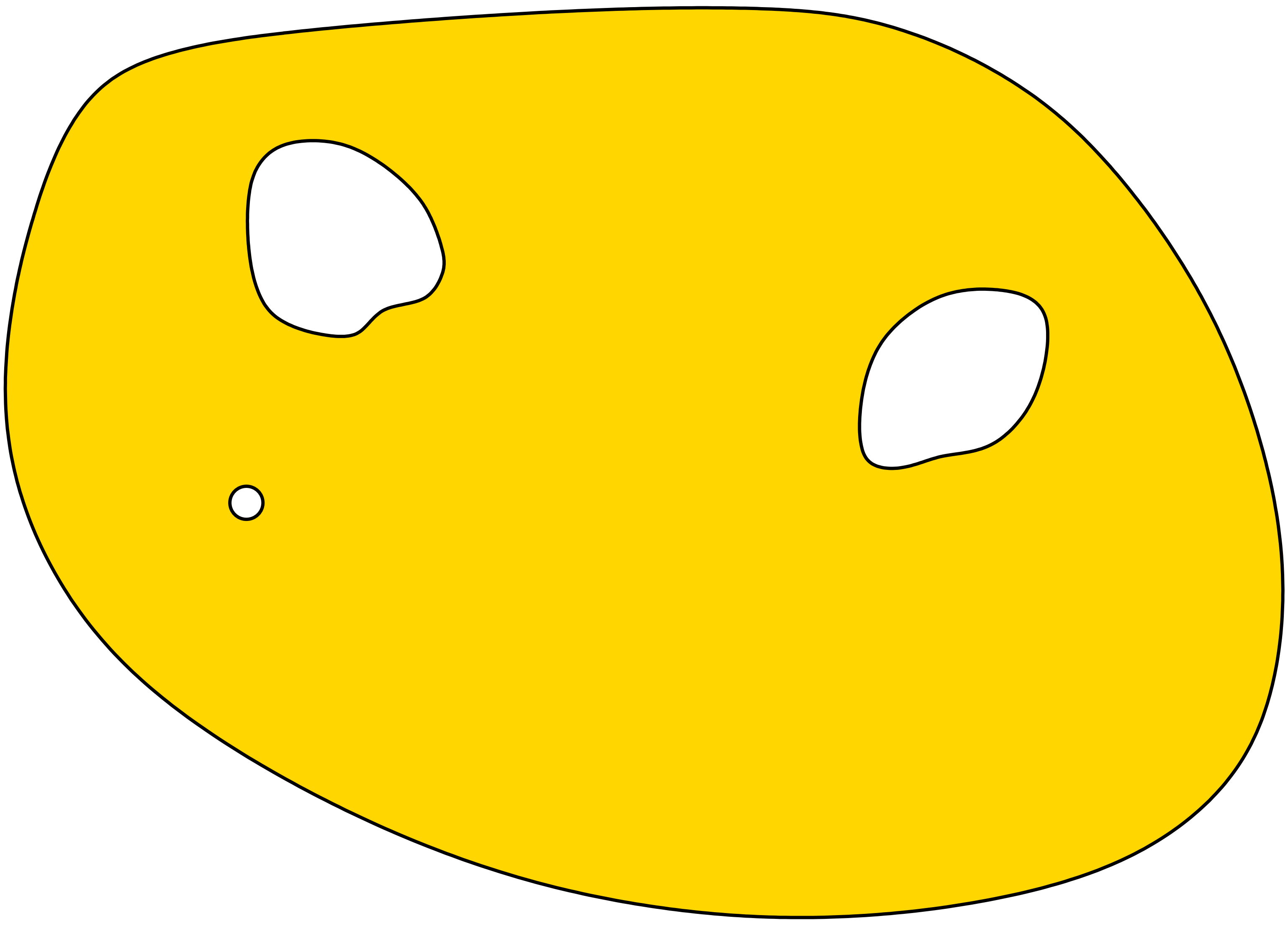
Iterative Procedure

Global procedure:

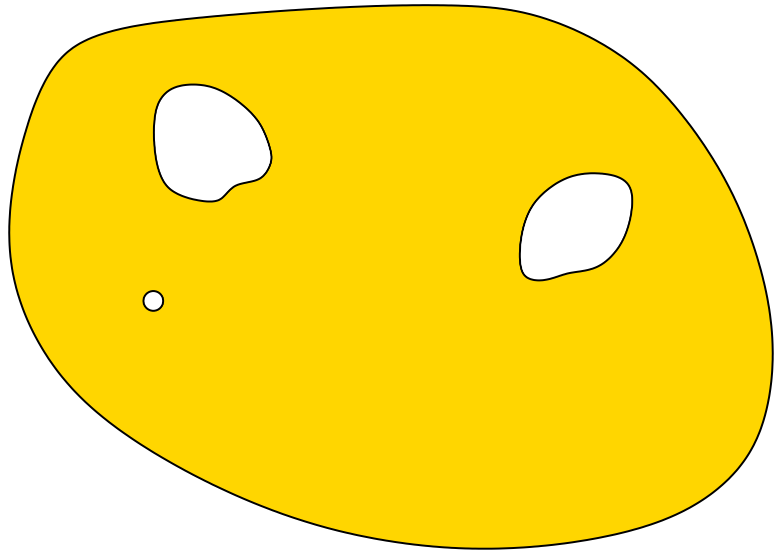


Iterative Procedure

Global procedure:



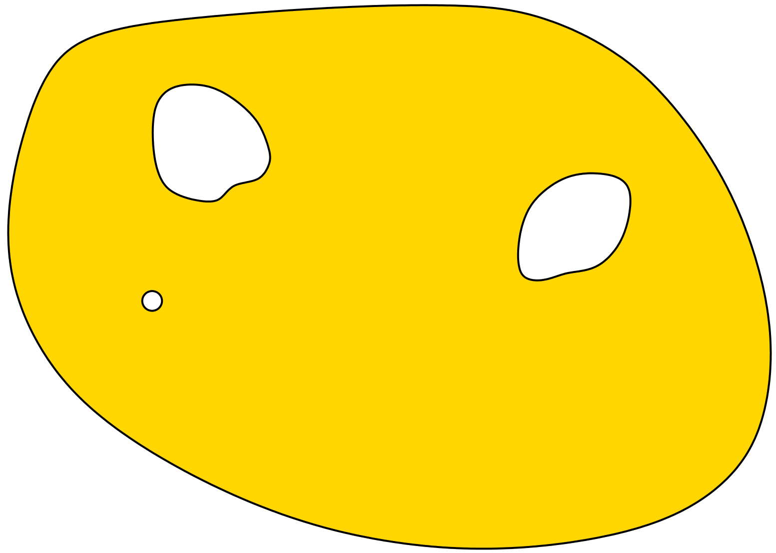
Local simulation



Use technique of Nguyen and O. (2008):

- Random numbers assigned to vertices generate a random permutation

Local simulation



Use technique of Nguyen and O. (2008):

- Random numbers assigned to vertices generate a random permutation
- To find a component of v:
  - recursively check what happened for close vertices with lower numbers
  - if v still in graph, try to construct a component

Open Problems

- Tight bounds for vertex cover and maximum matching

Open Problems

- Tight bounds for vertex cover and maximum matching
- Is there a $\mathrm{poly}(1/\epsilon)$-time/query partitioning oracle for minor-free graphs?
  - This would give a polynomial time/query tester for minor-freeness, and resolve an open question of Benjamini, Schramm, Shapira (2008)

Open Problems

- Tight bounds for vertex cover and maximum matching
- Is there a $\mathrm{poly}(1/\epsilon)$-time/query partitioning oracle for minor-free graphs?
  - This would give a polynomial time/query tester for minor-freeness, and resolve an open question of Benjamini, Schramm, Shapira (2008)
- Good approximation algorithms for other popular classes of graphs