On a Connection Between Distributed Algorithms and Sublinear-Time Algorithms

Krzysztof Onak
MIT

In this talk…

- Problems
  - Optimization problems on graphs
  - Examples: vertex cover, maximum matching, dominating set, …
  - Maximum (or average) degree bounded by $d = O(1)$

In this talk…

- Problems
  - Optimization problems on graphs
  - Examples: vertex cover, maximum matching,
    dominating set, …
  - Maximum (or average) degree bounded by $d = O(1)$
- Local Distributed Algorithms
  - compute a solution in a constant number of communication rounds

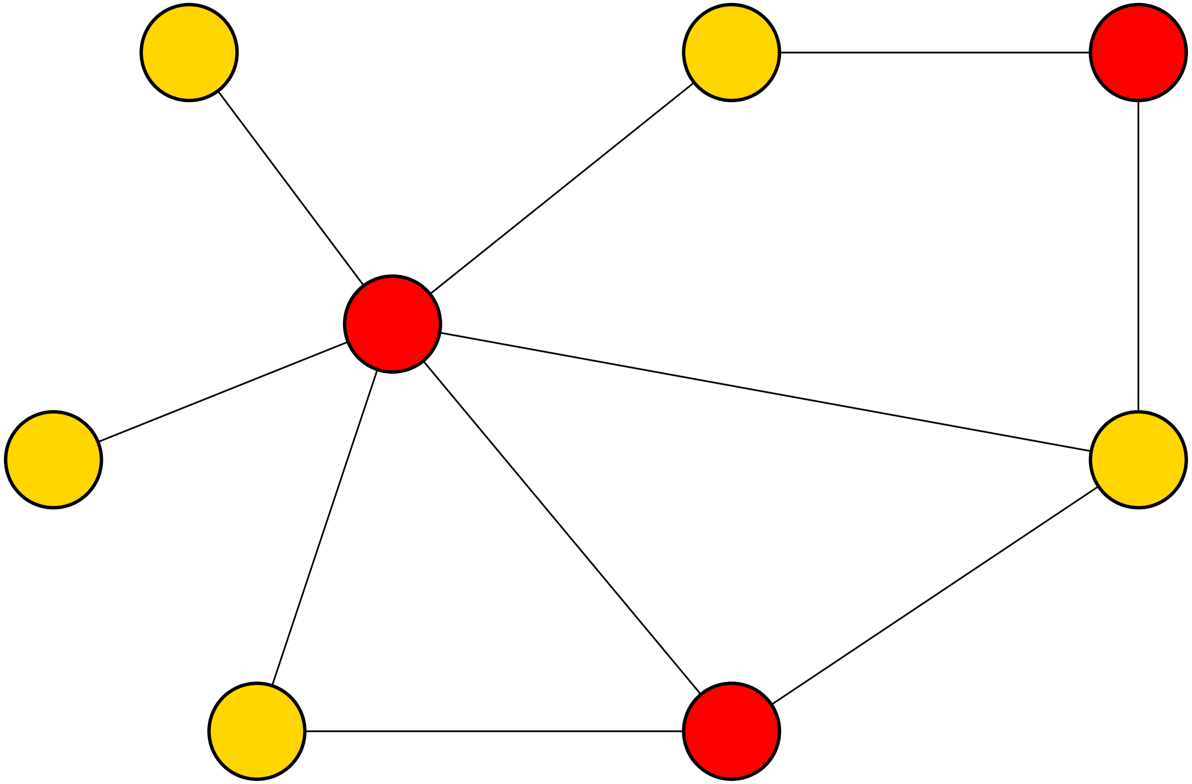
In this talk…

- Problems
  - Optimization problems on graphs
  - Examples: vertex cover, maximum matching, dominating set, …
  - Maximum (or average) degree bounded by $d = O(1)$
- Local Distributed Algorithms
  - compute a solution in a constant number of communication rounds
- Constant-Time Approximation Algorithms
  - Approximate the optimal solution size by only looking at a small fraction of the graph

Sample Problem: Vertex Cover

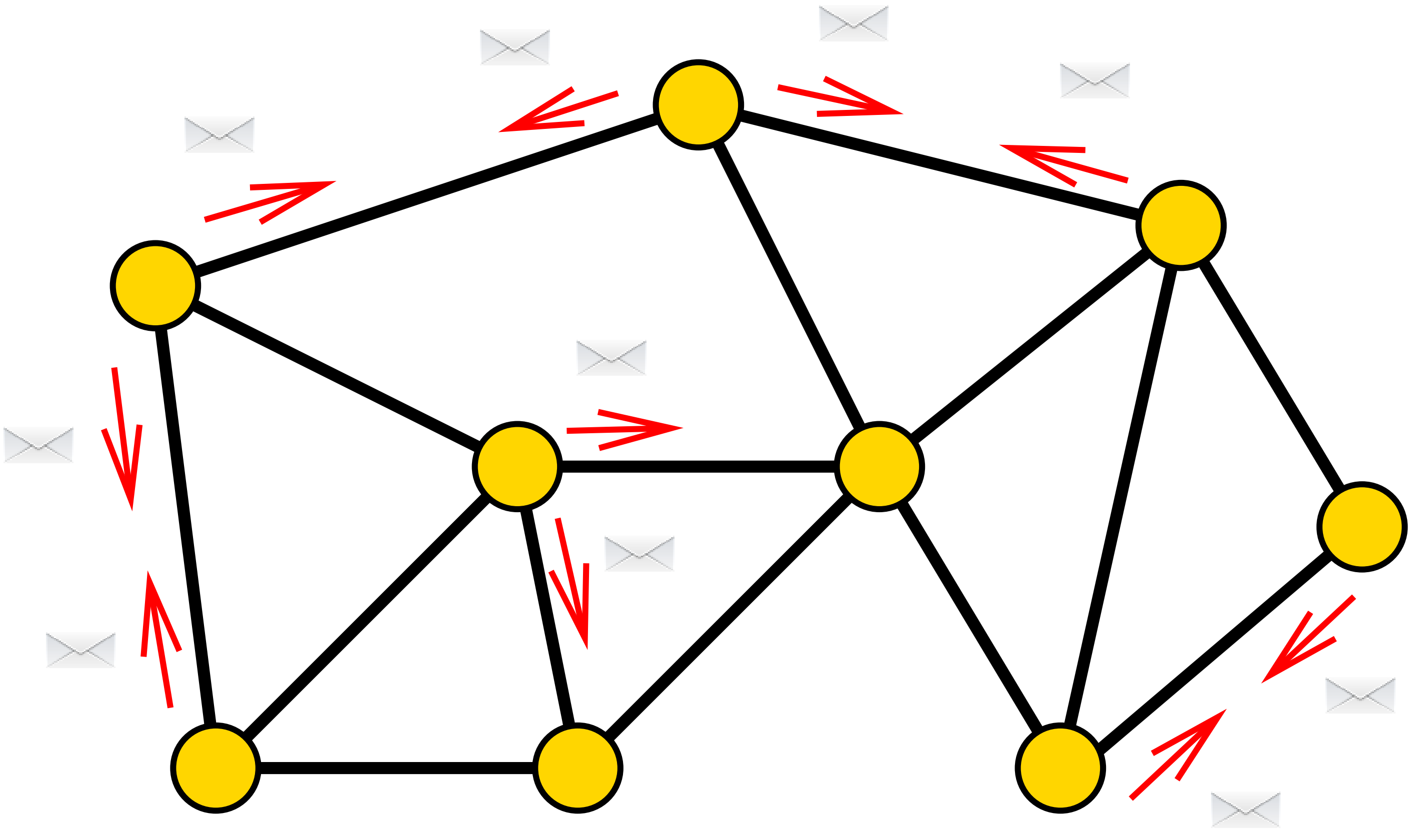
Graph $G = (V, E)$

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



Local Distributed Algorithms

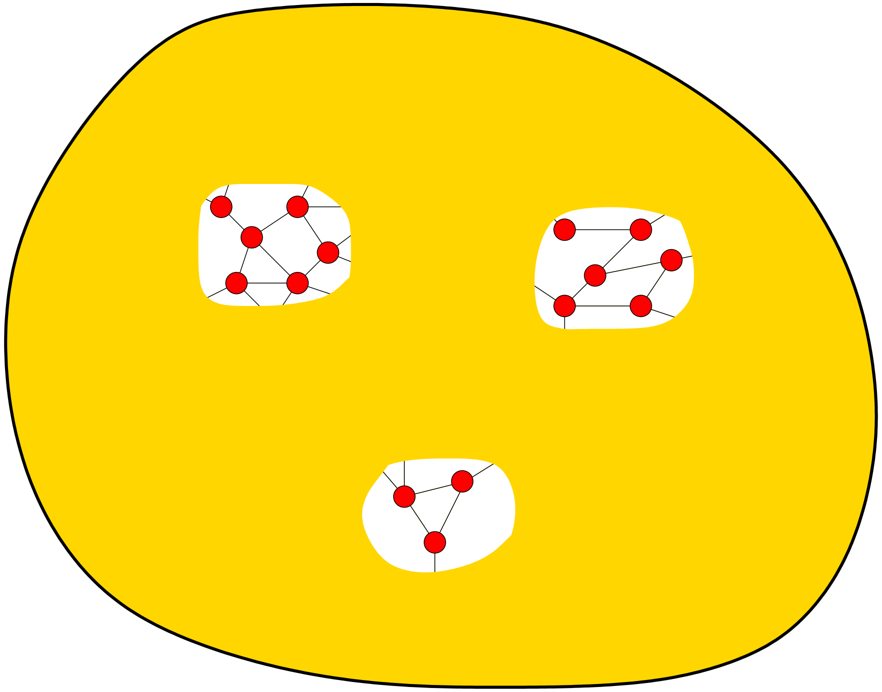
local $\equiv$ constant number of communication rounds
(can be a function of d)



Vertex Cover: finally every vertex knows if it is in the cover

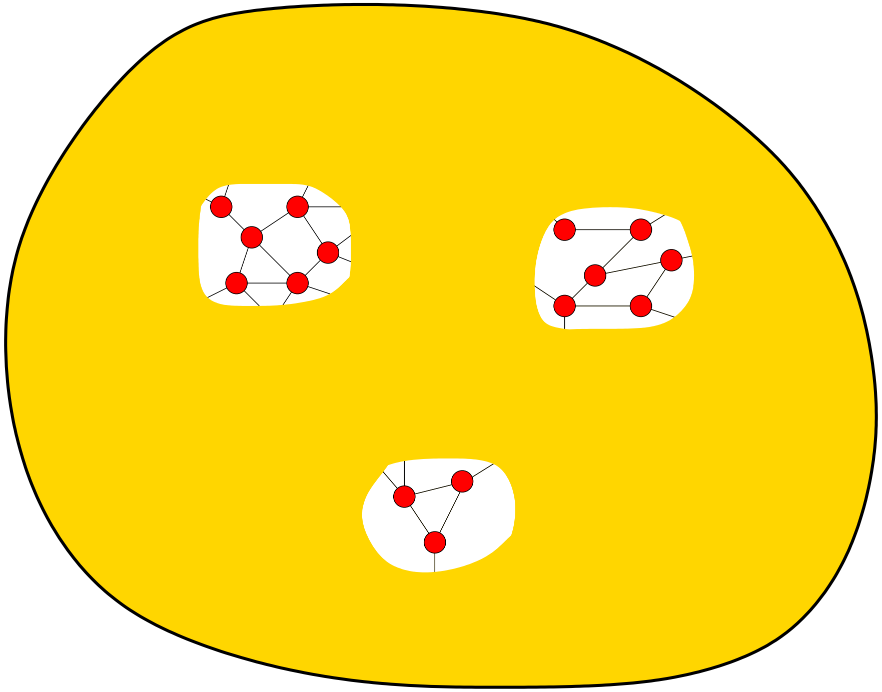
Constant-Time Algorithms

constant time $\equiv$ function of d and approximation quality parameter



Constant-Time Algorithms

constant time $\equiv$ function of d and approximation quality parameter

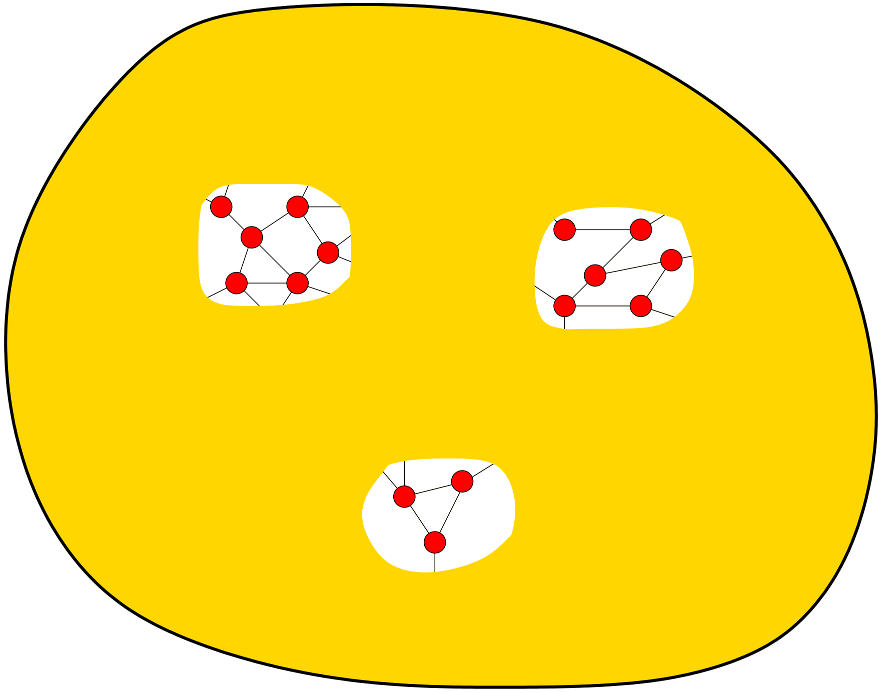


Approximation notion:

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

Constant-Time Algorithms

constant time $\equiv$ function of d and approximation quality parameter



Approximation notion:

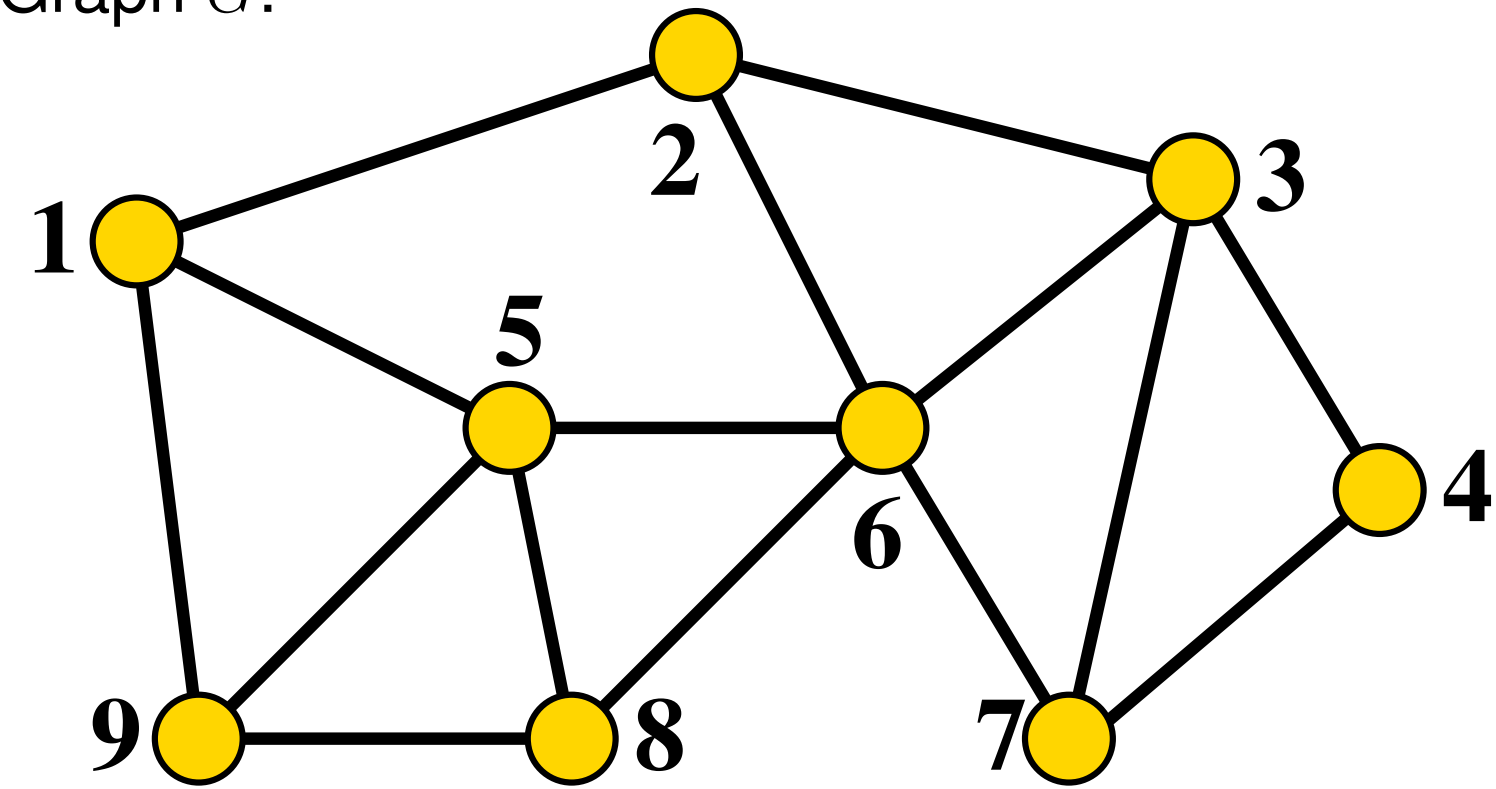
Y is an (α, β)-approximation to X if $X \leq Y \leq \alpha \cdot X + \beta$

We'll see:

constant-time $(2, \epsilon n)$-approximation algorithms
for vertex cover size

Query Model

Graph G:



Query access to adjacency list of each node

What is the 3rd neighbor of node 6?

Sampling from a Distributed Algorithm’s Solution

[Parnas, Ron 2007]

Approximation Algorithm

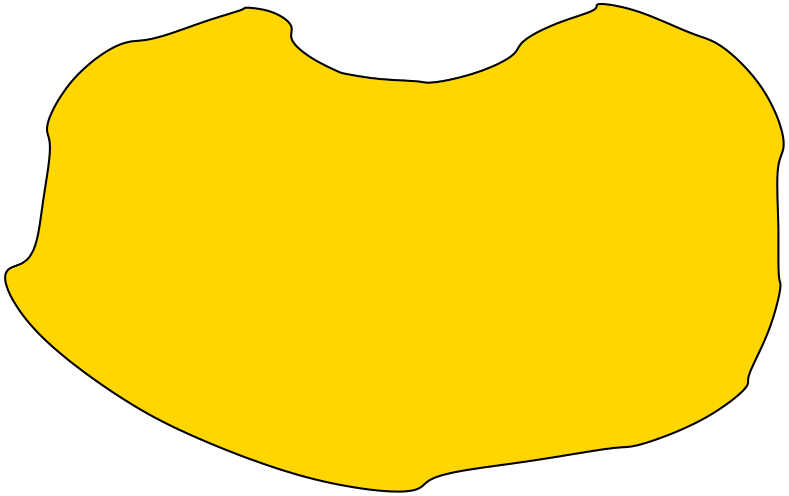
Local distributed approximation algorithm $\mathcal{A}$ for vertex cover:

- $\alpha =$ approximation factor
- $t =$ number of rounds

Approximation Algorithm

Local distributed approximation algorithm $\mathcal{A}$ for vertex cover:

- $\alpha =$ approximation factor
- $t =$ number of rounds

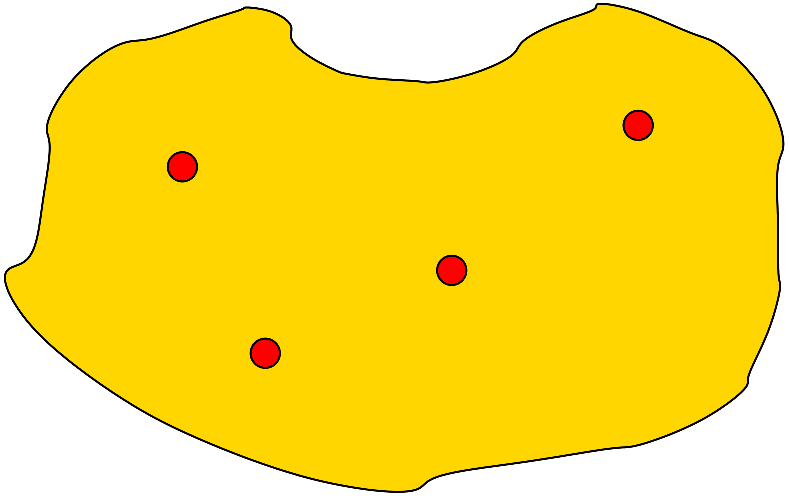


Constant-time algorithm:

Approximation Algorithm

Local distributed approximation algorithm $\mathcal{A}$ for vertex cover:

- $\alpha =$ approximation factor
- $t =$ number of rounds



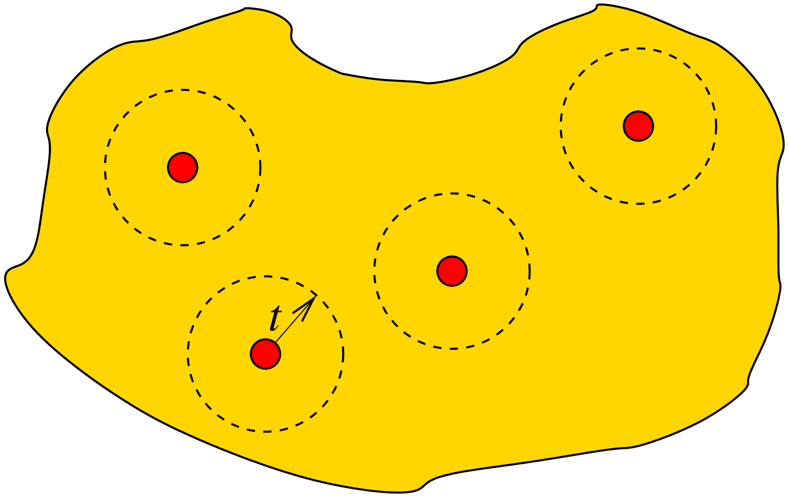
Constant-time algorithm:

1. Sample $O(1/\epsilon^2)$ vertices v

Approximation Algorithm

Local distributed approximation algorithm $\mathcal{A}$ for vertex cover:

- $\alpha =$ approximation factor
- $t =$ number of rounds



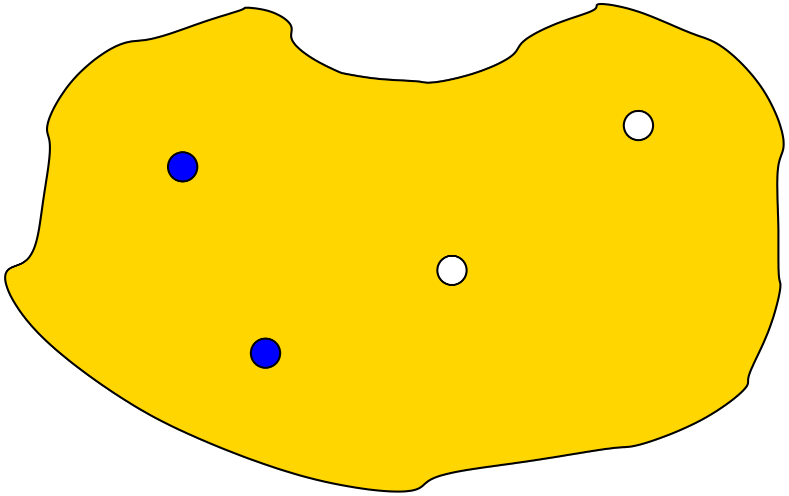
Constant-time algorithm:

1. Sample $O(1/\epsilon^2)$ vertices v
2. Simulate $\mathcal{A}$ on the neighborhood of each v of radius t

Approximation Algorithm

Local distributed approximation algorithm $\mathcal{A}$ for vertex cover:

- $\alpha =$ approximation factor
- $t =$ number of rounds



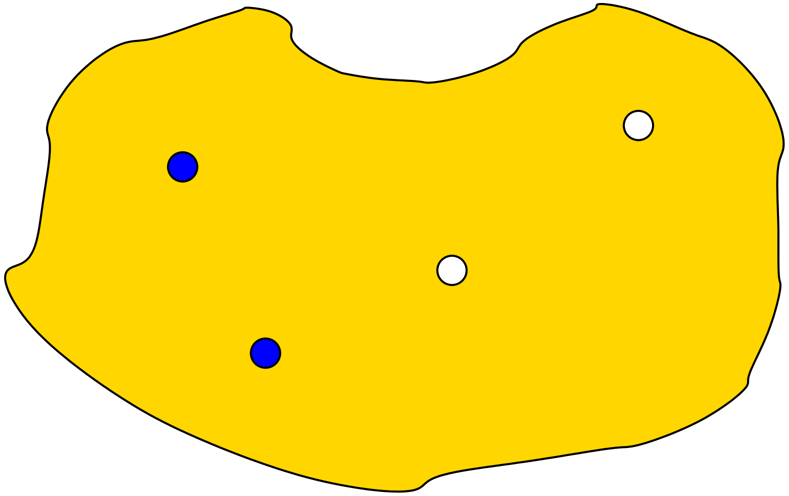
Constant-time algorithm:

1. Sample $O(1/\epsilon^2)$ vertices v
2. Simulate $\mathcal{A}$ on the neighborhood of each v of radius t
3. Return the fraction of vertices that are in $\mathcal{A}$'s cover ($+\epsilon n/2$)

Approximation Algorithm

Local distributed approximation algorithm $\mathcal{A}$ for vertex cover:

- $\alpha =$ approximation factor
- $t =$ number of rounds



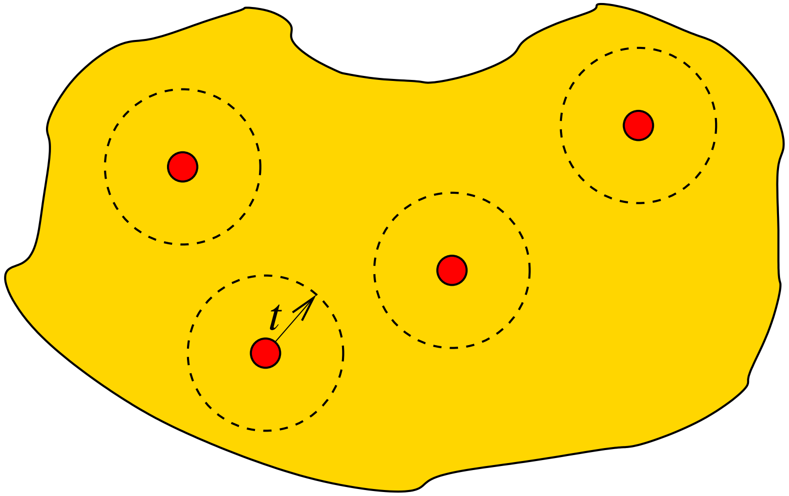
Constant-time algorithm:

1. Sample $O(1/\epsilon^2)$ vertices v
2. Simulate $\mathcal{A}$ on the neighborhood of each v of radius t
3. Return the fraction of vertices that are in $\mathcal{A}$'s cover ($+\epsilon n/2$)

Output: $(\alpha, \epsilon n)$-approximation with constant probability

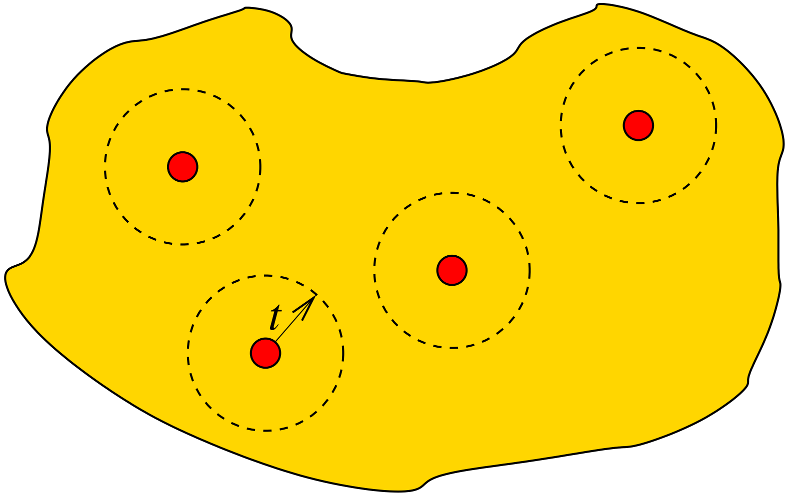
Complexity of the Algorithm

Query complexity: $O(1/\epsilon^2) \cdot d^{O(t)}$



Complexity of the Algorithm

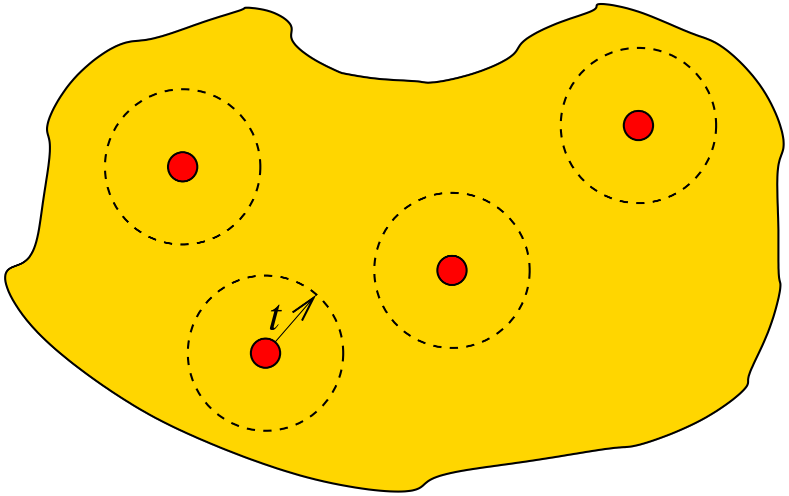
Query complexity: $O(1/\epsilon^2) \cdot d^{O(t)}$



Parnas and Ron applied algorithms of Kuhn, Moscibroda, Wattenhofer (2006) to vertex cover

Complexity of the Algorithm

Query complexity: $O(1/\epsilon^2) \cdot d^{O(t)}$

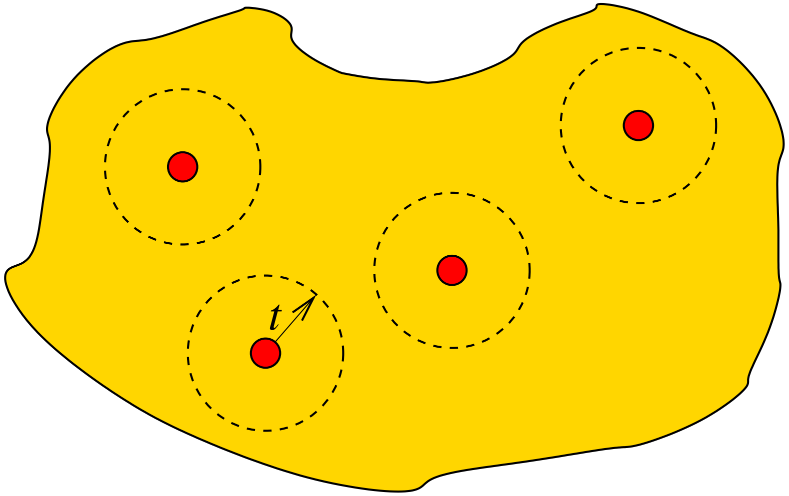


Parnas and Ron applied algorithms of Kuhn, Moscibroda, Wattenhofer (2006) to vertex cover:

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

Complexity of the Algorithm

Query complexity: $O(1/\epsilon^2) \cdot d^{O(t)}$



Parnas and Ron applied algorithms of Kuhn, Moscibroda, Wattenhofer (2006) to vertex cover:

- $\forall c > 2$, $(c, \epsilon n)$-approximation with $d^{O(\log(d))}/\epsilon^2$ queries
- $(2, \epsilon n)$-approximation with $d^{O(\log(d)/\epsilon^3)}$ queries

Slightly Better Algorithms

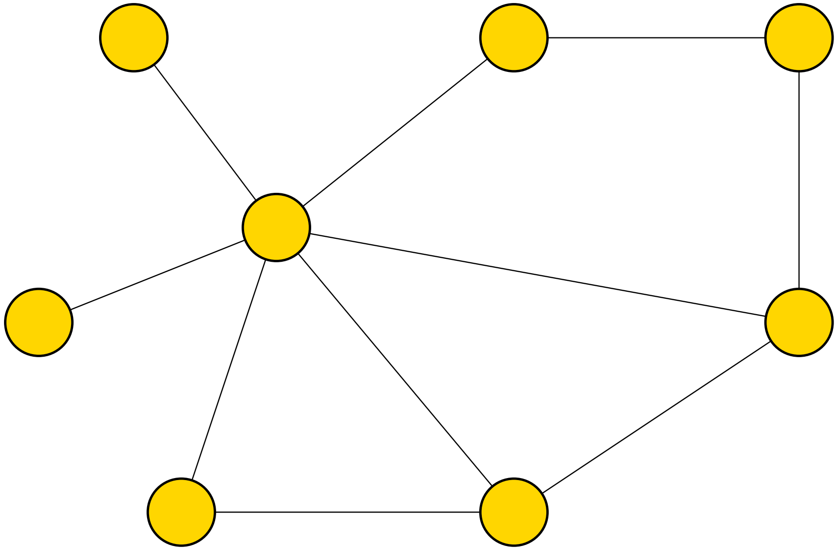
[Marko, Ron 2007]

Vertex Cover

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

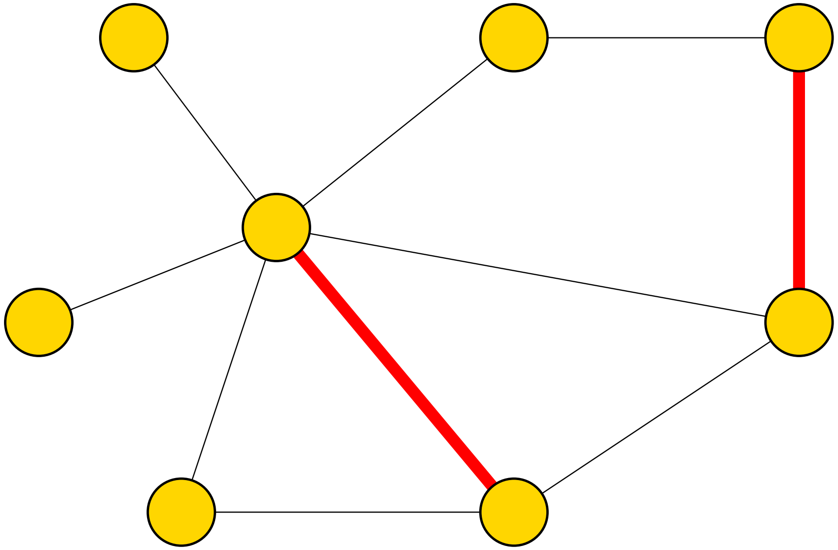


Vertex Cover

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

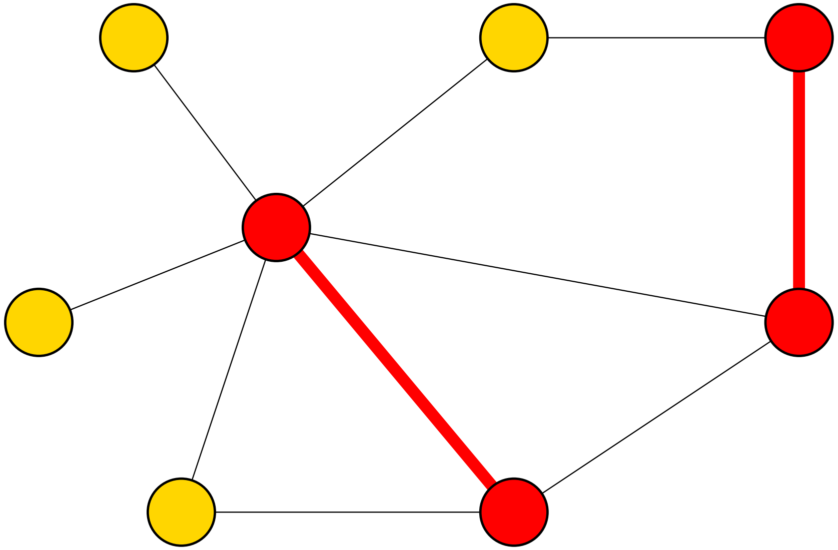


Vertex Cover

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

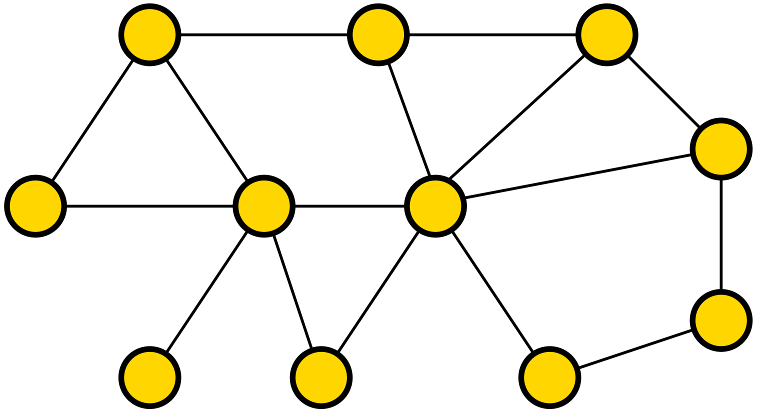
Classical 2-approximation algorithm [Gavril]:

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



Algorithm of Marko & Ron

via Luby (1986)

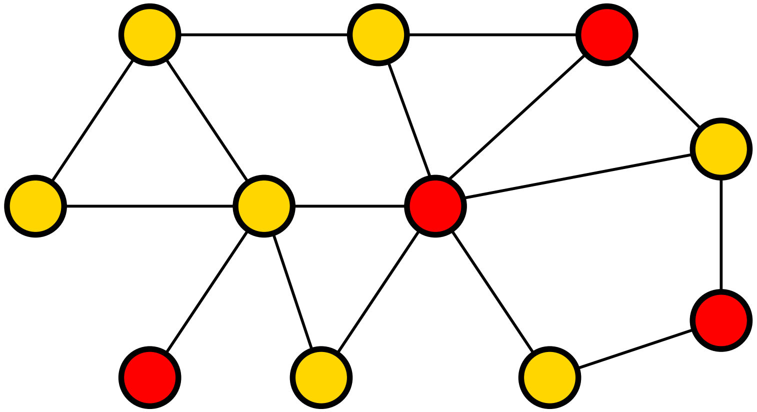


Repeat:

- select each node v with probability $\Theta(1/d(v))$
- deselect a node if a neighbor selected
- add selected nodes to independent set
- remove selected nodes and their neighbors from graph

Algorithm of Marko & Ron

via Luby (1986)

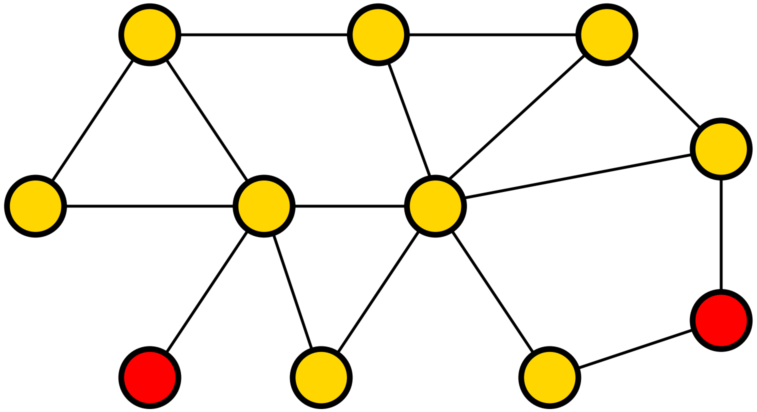


Repeat:

- select each node v with probability $\Theta(1/d(v))$
- deselect a node if a neighbor selected
- add selected nodes to independent set
- remove selected nodes and their neighbors from graph

Algorithm of Marko & Ron

via Luby (1986)

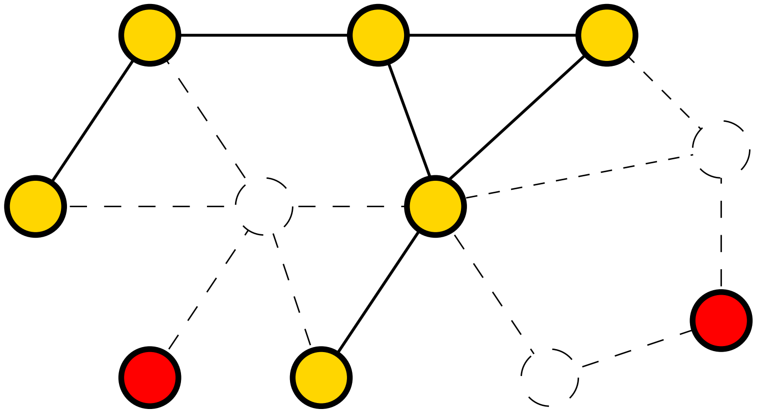


Repeat:

- select each node v with probability $\Theta(1/d(v))$
- deselect a node if a neighbor selected
- add selected nodes to independent set
- remove selected nodes and their neighbors from graph

Algorithm of Marko & Ron

via Luby (1986)

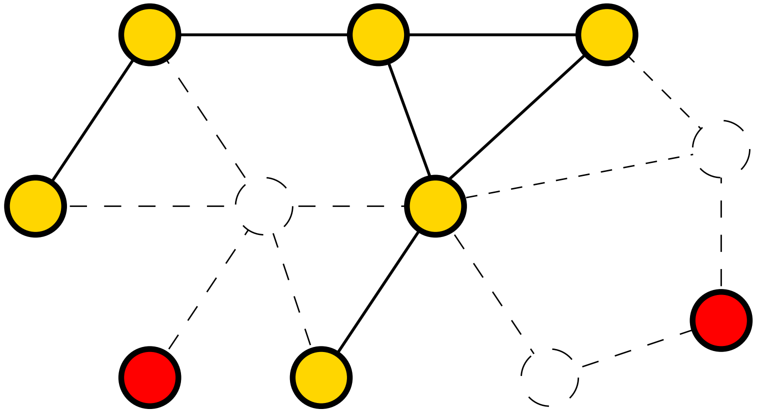


Repeat:

- select each node v with probability $\Theta(1/d(v))$
- deselect a node if a neighbor selected
- add selected nodes to independent set
- remove selected nodes and their neighbors from graph

Algorithm of Marko & Ron

via Luby (1986)



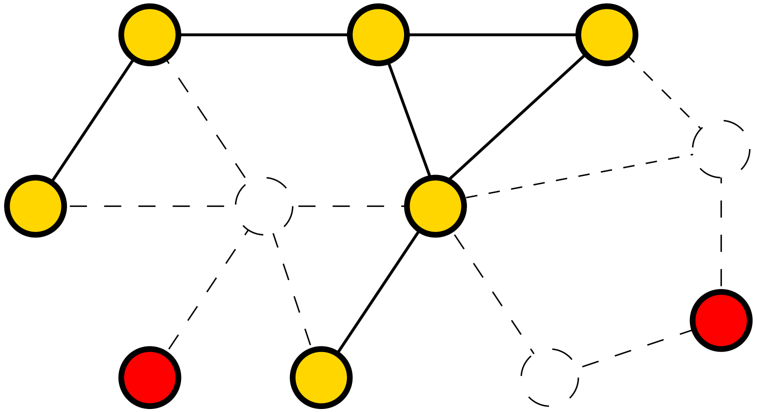
Repeat:

- select each node v with probability $\Theta(1/d(v))$
- deselect a node if a neighbor selected
- add selected nodes to independent set
- remove selected nodes and their neighbors from graph

Can show: $1-\delta$ fraction of vertices decided in $O(\log(d/\delta))$ rounds

Algorithm of Marko & Ron

via Luby (1986)



Repeat:

- select each node v with probability $\Theta(1/d(v))$
- deselect a node if a neighbor selected
- add selected nodes to independent set
- remove selected nodes and their neighbors from graph

Can show: $1 - \delta$ fraction of vertices decided in $O(\log(d/\delta))$ rounds

Ramifications for vertex cover:

- distributed: $(2 + \delta)$-approximation in $O(\log(d/\delta))$ rounds
- sublinear: $(2, \epsilon n)$-approximation with $d^{O(\log(d/\epsilon))}$ queries

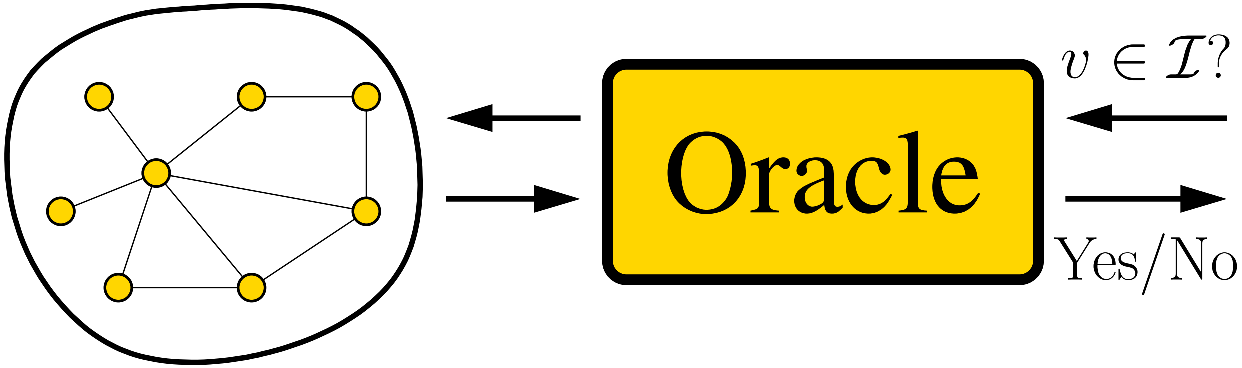
Local Greedy Computation

[Nguyen, O. 2008]

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 $\mathcal{I} \subseteq V$
- $\mathcal{I}$ independent of queries



Goal: Minimize the query processing time

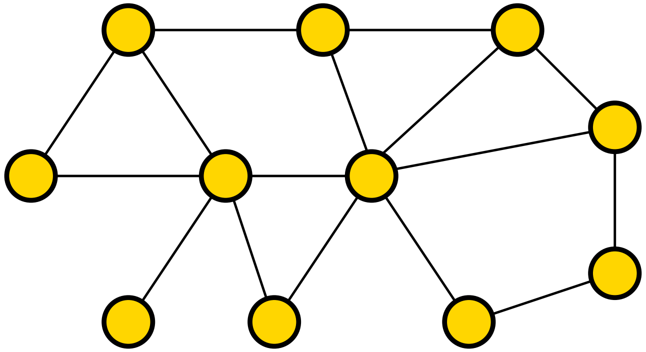
Local Greedy Computation

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



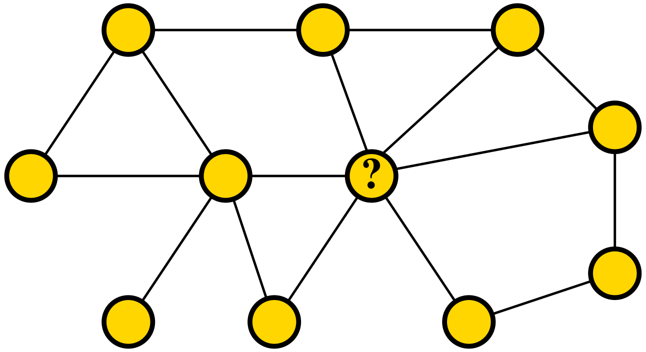
Local Greedy Computation

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 $\mathcal{I}$

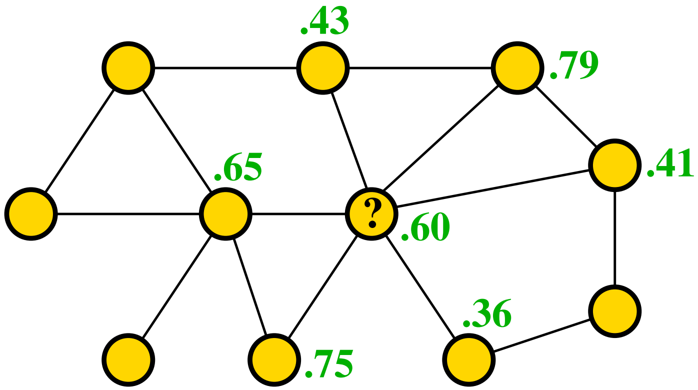
Local Greedy Computation

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 $\mathcal{I}$

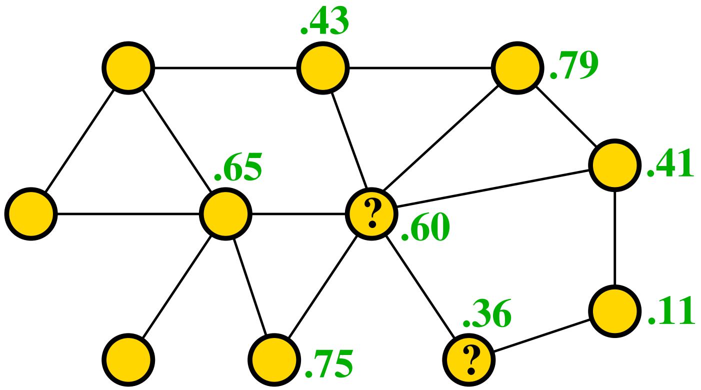
Local Greedy Computation

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 $\mathcal{I}$

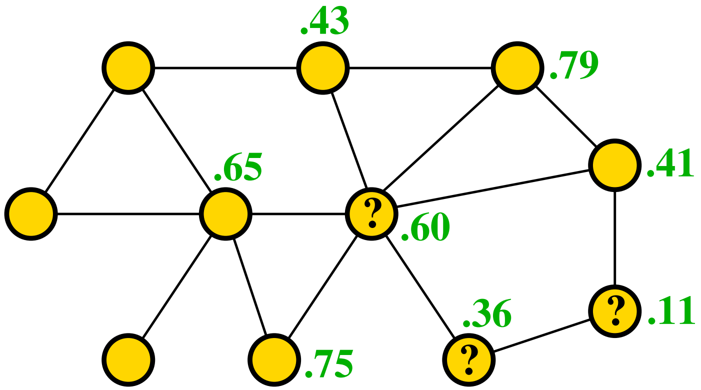
Local Greedy Computation

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 $\mathcal{I}$

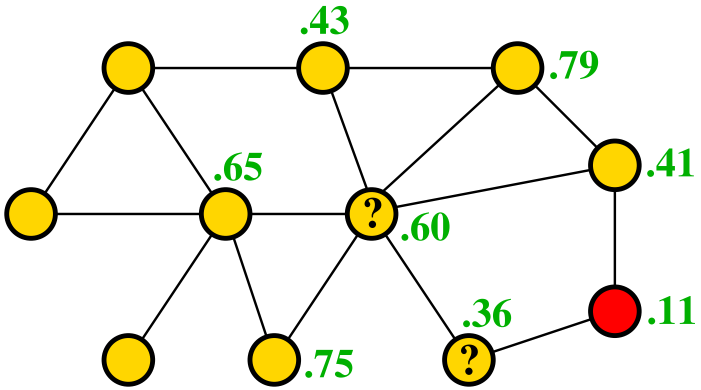
Local Greedy Computation

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 $\mathcal{I}$

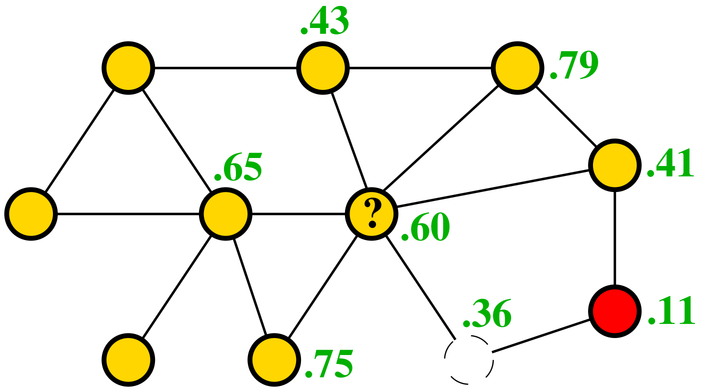
Local Greedy Computation

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 $\mathcal{I}$

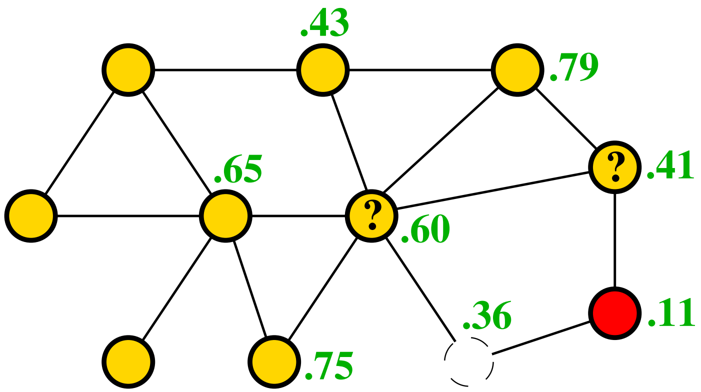
Local Greedy Computation

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 $\mathcal{I}$

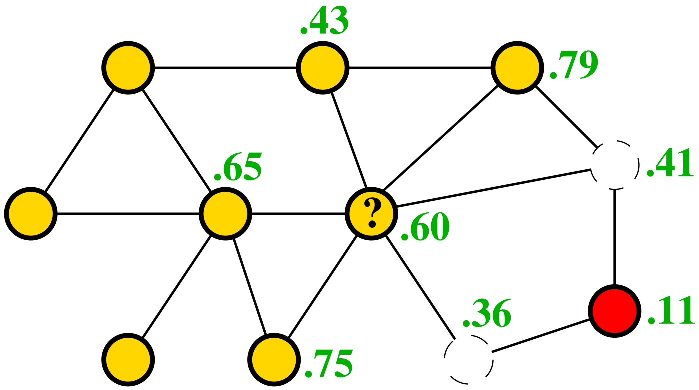
Local Greedy Computation

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 $\mathcal{I}$

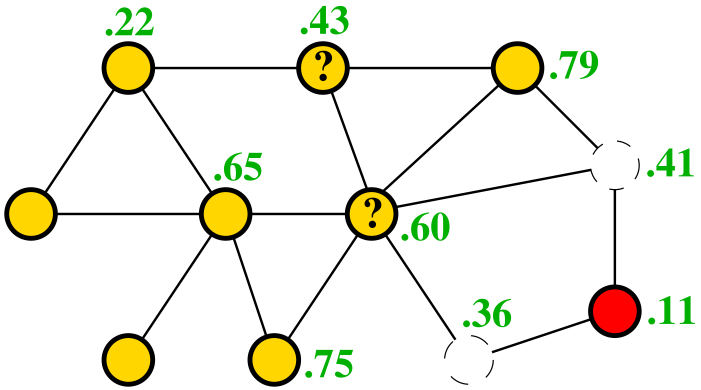
Local Greedy Computation

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 $\mathcal{I}$

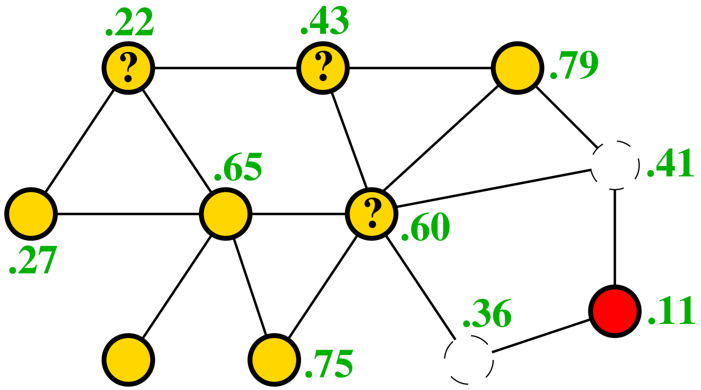
Local Greedy Computation

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 $\mathcal{I}$

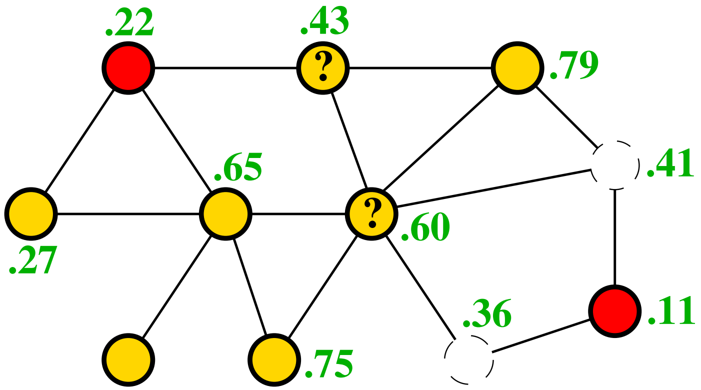
Local Greedy Computation

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 $\mathcal{I}$

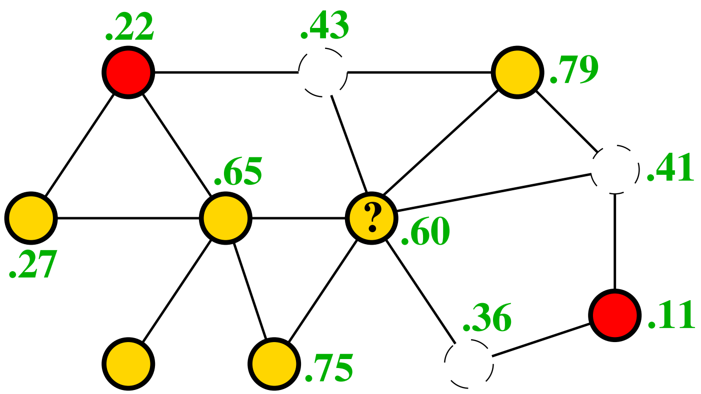
Local Greedy Computation

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 $\mathcal{I}$

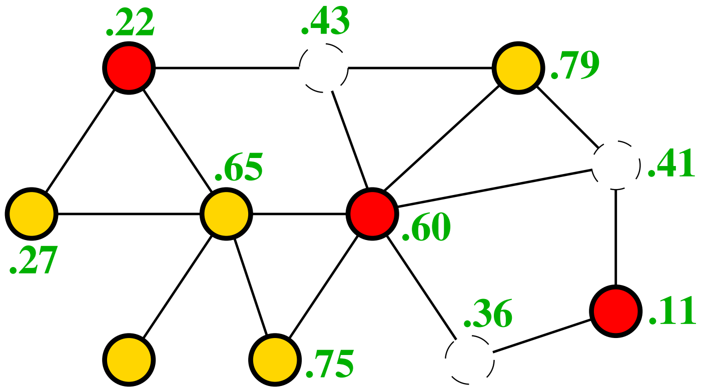
Local Greedy Computation

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 $\mathcal{I}$

Bounding Expected Query Complexity

- $\Pr[\text{a given path of length } k \text{ is explored}] \le 1/(k+1)!$

Bounding Expected Query Complexity

- $\Pr[\text{a given path of length } k \text{ is explored}] \leq 1/(k+1)!$
- number of neighbors at distance $k \leq d^k$

Bounding Expected Query Complexity

- $\Pr[$a given path of length k is explored$] \leq 1/(k+1)!$
- number of neighbors at distance $k \leq d^k$
- $E[$number of vertices explored at distance $k] \leq d^k/(k+1)!$

Bounding Expected Query Complexity

- $\mathrm{Pr}[$a given path of length k is explored$] \leq 1/(k+1)!$
- number of neighbors at distance $k \leq d^k$
- $E[$number of vertices explored at distance $k] \leq d^k/(k+1)!$
- $E[$number of explored vertices$] \leq \sum_{k=0}^{\infty} d^k/(k+1)!$
  $\leq e^d/d$

Bounding Expected Query Complexity

- $\Pr[\text{a given path of length } k \text{ is explored}] \leq 1/(k+1)!$
- number of neighbors at distance $k \leq d^k$
- $E[\text{number of vertices explored at distance } k] \leq d^k/(k+1)!$
- $E[\text{number of explored vertices}] \leq \sum_{k=0}^{\infty} d^k/(k+1)!$
  $\leq e^d/d$
- Expected query complexity $= O(d) \cdot e^d/d = O(e^d)$

Recent Improvement

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)

Recent Improvement

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:

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

Recent Improvement

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 = $O(d^2)$

$(2, \epsilon n)$-Approximation for Vertex Cover

Parnas, Ron (2007) + Kuhn, Moscibroda, Wattenhofer (2006):

$$d^{O(\log(d)/\epsilon^3)} \text{ queries}$$

$(2, \epsilon n)$-Approximation for Vertex Cover

Parnas, Ron (2007) + Kuhn, Moscibroda, Wattenhofer (2006):

$$d^{O(\log(d)/\epsilon^3)} \text{ queries}$$

Marko, Ron (2007):

$$d^{O(\log(d/\epsilon))} \text{ queries}$$

$(2, \epsilon n)$-Approximation for Vertex Cover

Parnas, Ron (2007) + Kuhn, Moscibroda, Wattenhofer (2006):

$$d^{O(\log(d)/\epsilon^3)} \text{ queries}$$

Marko, Ron (2007):

$$d^{O(\log(d/\epsilon))} \text{ queries}$$

Nguyen, O. (2008):

$$2^{O(d)}/\epsilon^2 \text{ queries}$$

$(2, \epsilon n)$-Approximation for Vertex Cover

Parnas, Ron (2007) + Kuhn, Moscibroda, Wattenhofer (2006):

$$d^{O(\log(d)/\epsilon^3)} \text{ queries}$$

Marko, Ron (2007):

$$d^{O(\log(d/\epsilon))} \text{ queries}$$

Nguyen, O. (2008):

$$2^{O(d)}/\epsilon^2 \text{ queries}$$

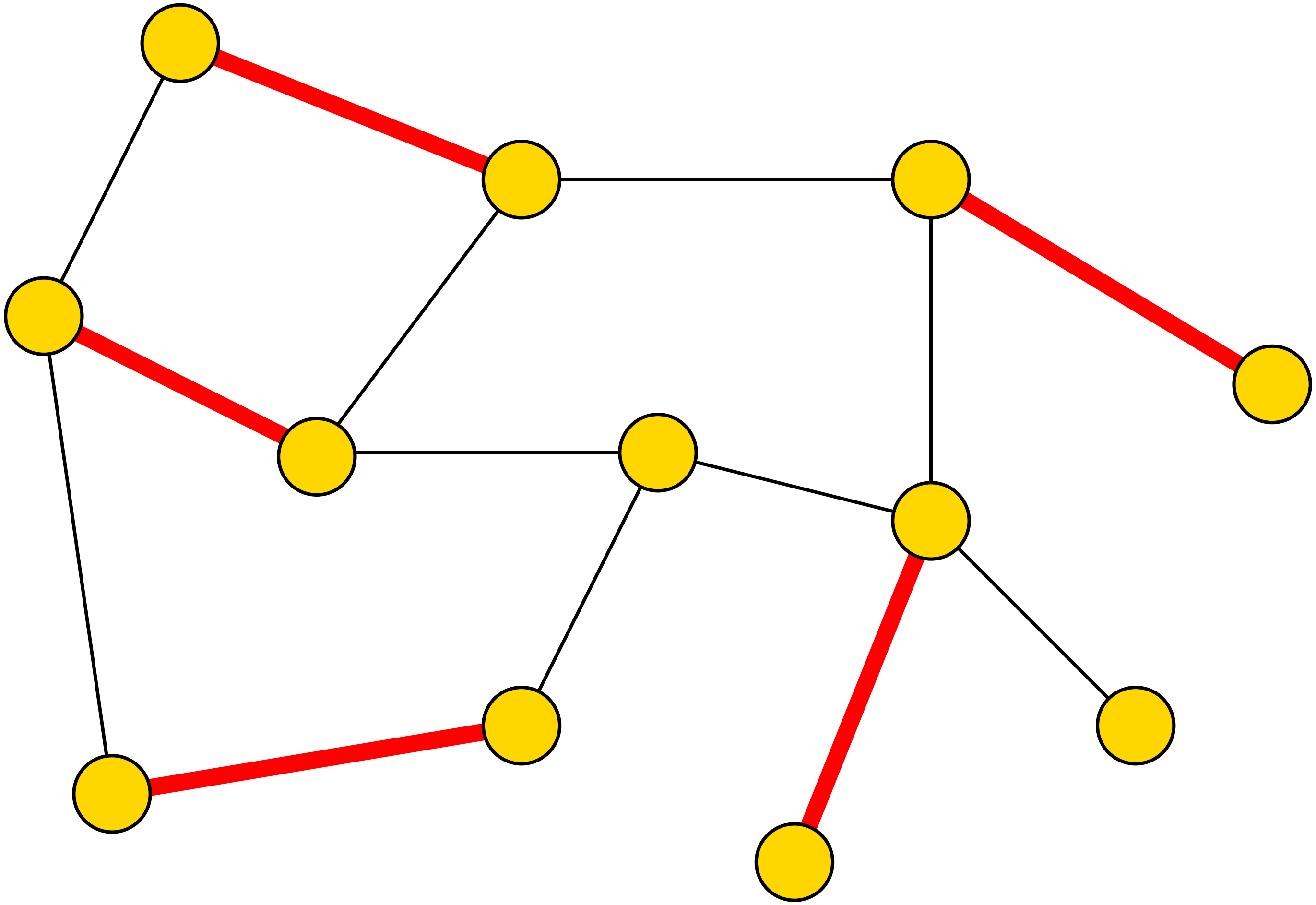
Yoshida, Yamamoto, Ito (2009):

$$O(d^3/\epsilon^2) \text{ queries}$$

$(1, \epsilon n)$-Approximation for Maximum Matching

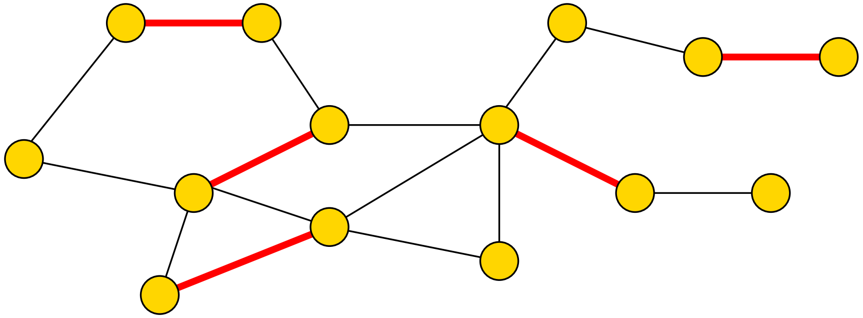
Maximum Matching

Goal: find a set of disjoint edges of maximum cardinality



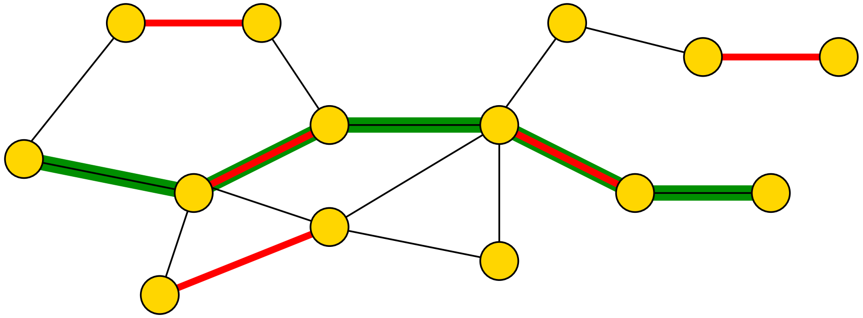
Review of Properties

Augmenting Path: a path that improves matching



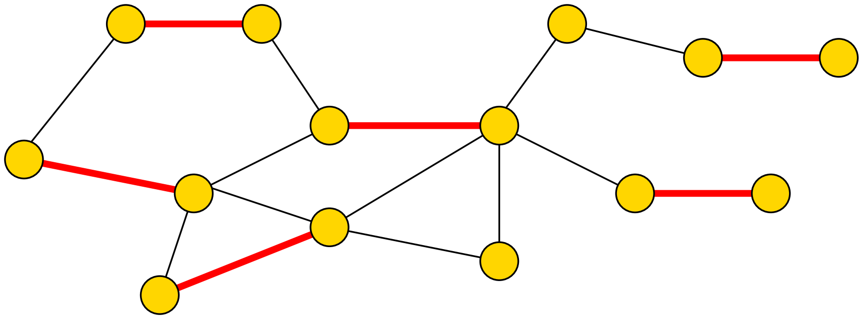
Review of Properties

Augmenting Path: a path that improves matching



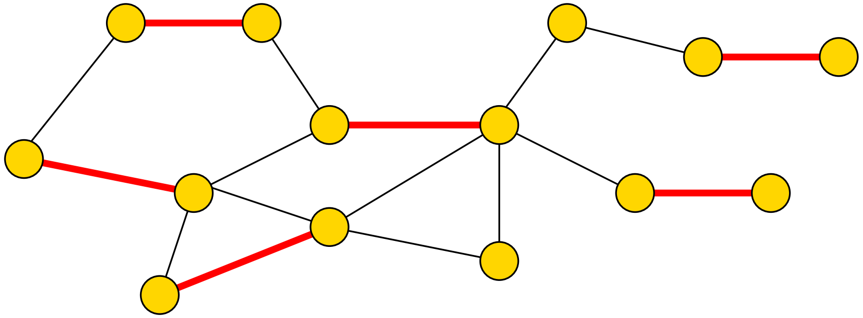
Review of Properties

Augmenting Path: a path that improves matching



Review of Properties

Augmenting Path: a path that improves matching

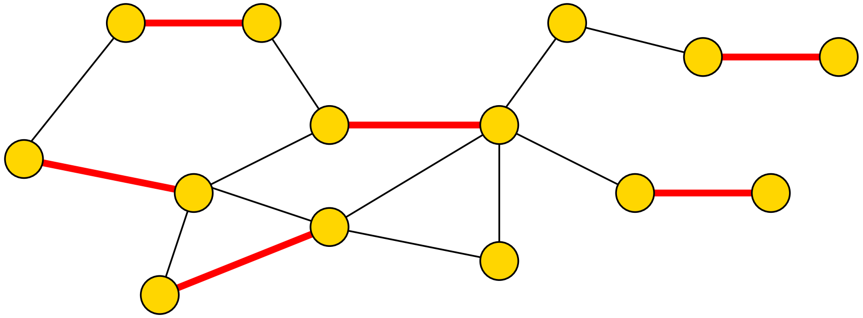


M = matching, M^* = maximum matching

Fact: There are $|M^*| - |M|$ disjoint augmenting paths for M

Review of Properties

Augmenting Path: a path that improves matching



M = matching, M^* = maximum matching

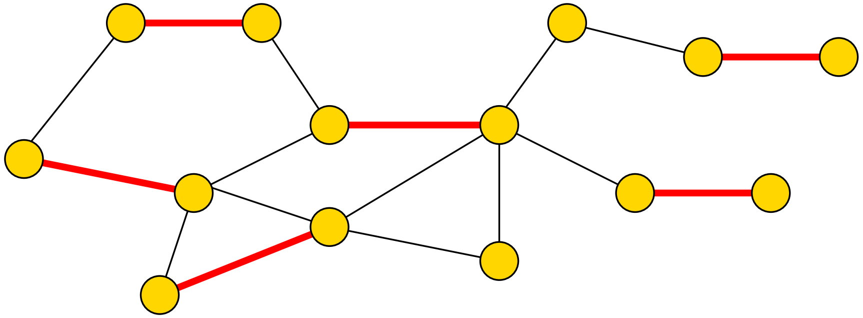
Fact: There are $|M^*| - |M|$ disjoint augmenting paths for M

Fact:

No augmenting paths of length $< 2k+1 \ \Rightarrow \ |M| \geq \frac{k}{k+1}|M^*|$

Review of Properties

Augmenting Path: a path that improves matching



M = matching, M^* = maximum matching

Fact: There are $|M^*| - |M|$ disjoint augmenting paths for M

Fact:
No augmenting paths of length $< 2k+1 \;\Rightarrow\; |M| \geq \frac{k}{k+1}|M^*|$

To get $(1+\epsilon)$-approximation, set $k = \lceil 1/\epsilon \rceil$

Standard Algorithm

Lemma [Hopcroft, Karp 1973]:

M = matching with no augmenting paths of length $< t$

P = maximal set of vertex-disjoint augmenting paths of length t for M

M' = M with all paths in P applied

Claim: M' has only augmenting paths of length $> t$

Standard Algorithm

Lemma [Hopcroft, Karp 1973]:

M = matching with no augmenting paths of length $< t$

P = maximal set of vertex-disjoint augmenting paths of length t for M

M' = M with all paths in P applied

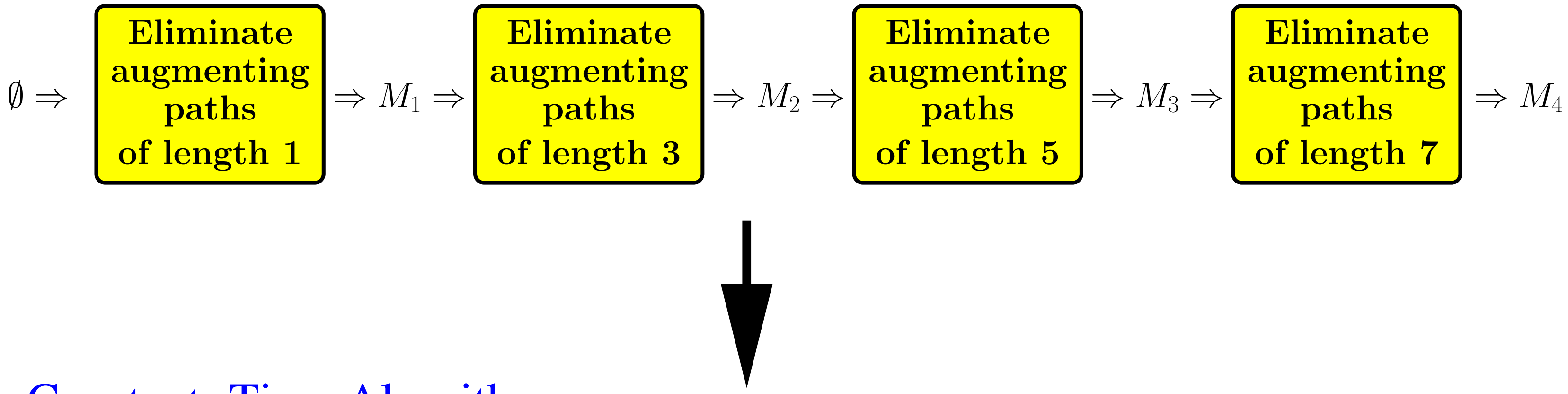
Claim: M' has only augmenting paths of length $> t$

Algorithm:

```
$M :=$ empty matching
for $i = 1$ to $k$:
    find maximal set of disjoint augmenting paths of length $2i - 1$
    apply all paths to $M$
return $M$
```

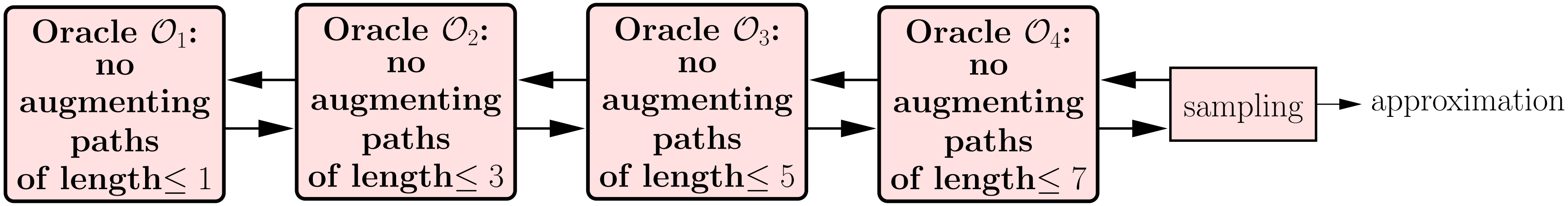
Transformation

Standard Algorithm:



$\emptyset \Rightarrow$ **Eliminate augmenting paths of length 1** $\Rightarrow M_1 \Rightarrow$ **Eliminate augmenting paths of length 3** $\Rightarrow M_2 \Rightarrow$ **Eliminate augmenting paths of length 5** $\Rightarrow M_3 \Rightarrow$ **Eliminate augmenting paths of length 7** $\Rightarrow M_4$

Constant–Time Algorithm:



Oracle $\mathcal{O}_1$: no augmenting paths of length≤ 1

Oracle $\mathcal{O}_2$: no augmenting paths of length≤ 3

Oracle $\mathcal{O}_3$: no augmenting paths of length≤ 5

Oracle $\mathcal{O}_4$: no augmenting paths of length≤ 7

sampling

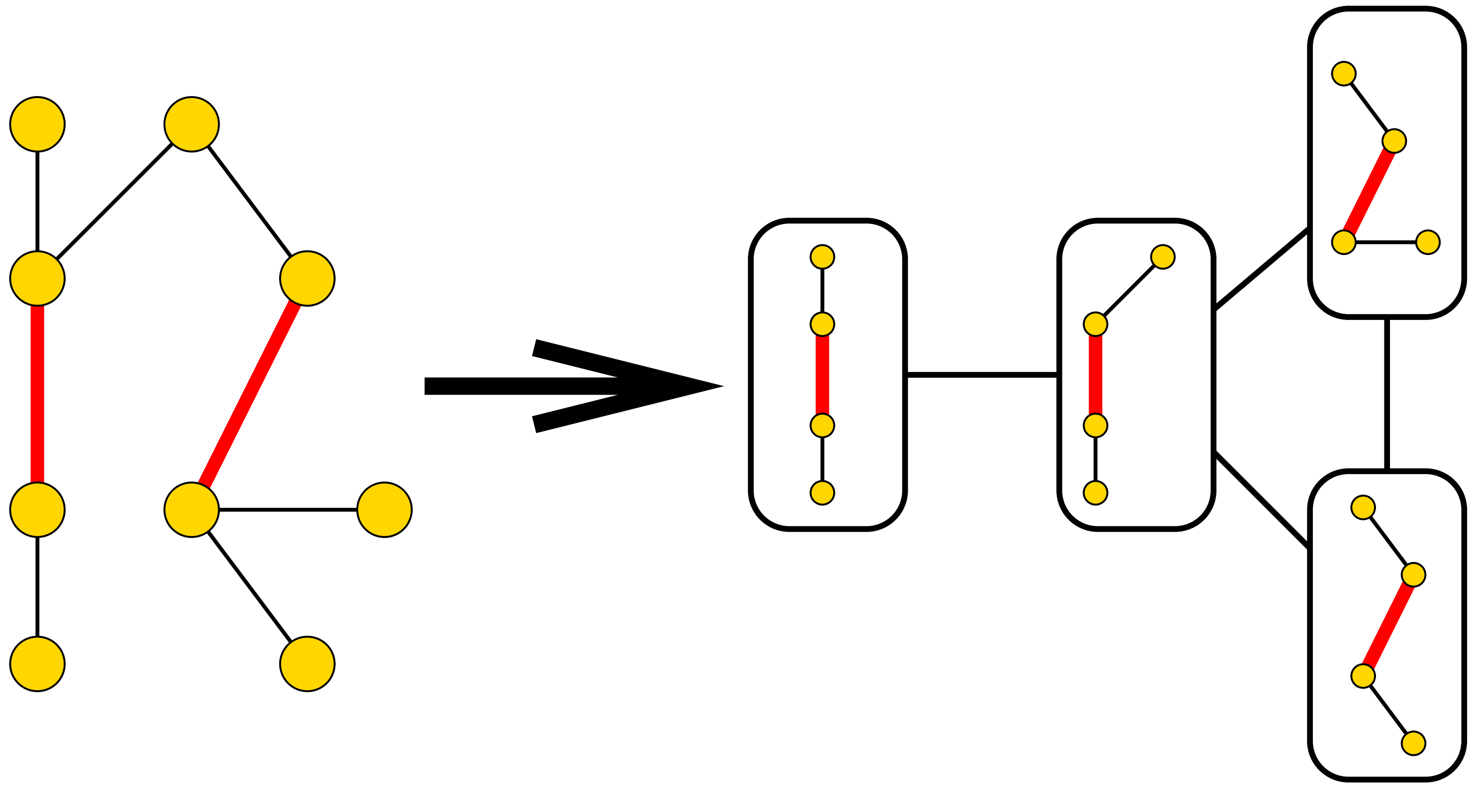
approximation

Oracle $\mathcal{O}_i$:

- provides query access to M_i
- simulates applying to M_{i-1} a maximal set of disjoint augmenting paths of length $2i-1$

Transformation

Sample graph considered by $\mathcal{O}_2$:



$\mathcal{O}_i$'s graph has degree $d^{O(i)}$

Query Complexity

Can’t apply the previous approach!

- every query may disclose some information about the random numbers
- algorithm could use it to form a malicious query

Query Complexity

Can't apply the previous approach!

- every query may disclose some information about the random numbers
- algorithm could use it to form a malicious query

Locality Lemma:

for q queries, needs to visit at most $q^2 \cdot 2^{O(d^4)}/\delta$ vertices with probability $1 - \delta$

Query Complexity

Can't apply the previous approach!

- every query may disclose some information about the random numbers
- algorithm could use it to form a malicious query

Locality Lemma:

for q queries, needs to visit at most $q^2 \cdot 2^{O(d^4)}/\delta$ vertices with probability $1 - \delta$

Query complexity: $2^{d^{O(1/\epsilon)}}$ queries for $(1, \epsilon n)$-approximation

Query Complexity

Can't apply the previous approach!

- every query may disclose some information about the random numbers
- algorithm could use it to form a malicious query

Locality Lemma:

for q queries, needs to visit at most $q^2 \cdot 2^{O(d^4)}/\delta$ vertices with probability $1-\delta$

Query complexity: $2^{d^{O(1/\epsilon)}}$ queries for $(1, \epsilon n)$-approximation

Yoshida, Yamamoto, Ito (2009)

- Query complexity: $d^{O(1/\epsilon^2)}$
- uniform on higher level ⇒ close to uniform on lower

Distributed Algorithms

- Can simulate the oracle locally for every vertex

Distributed Algorithms

- Can simulate the oracle locally for every vertex
- $(1-\epsilon)$-approximate maximum matching computable in $d^{O(1/\epsilon)}$ rounds

Lower Bounds

Relevant Lower Bounds

No constant-time $(\alpha, \epsilon n)$-approximation algorithm for:

- vertex cover if α constant less than 2 [Trevisan]

Relevant Lower Bounds

No constant-time $(\alpha, \epsilon n)$-approximation algorithm for:

- vertex cover if α constant less than 2 [Trevisan]
- dominating set if $\alpha = o(\log d)$ [Alon]

Relevant Lower Bounds

No constant-time $(\alpha, \epsilon n)$-approximation algorithm for:

- vertex cover if α constant less than 2 [Trevisan]
- dominating set if $\alpha = o(\log d)$ [Alon]
- maximum independent set if $\alpha = o\left(\frac{d}{\log d}\right)$ [Alon]

Relevant Lower Bounds

No constant-time $(\alpha, \epsilon n)$-approximation algorithm for:

- vertex cover if α constant less than 2 [Trevisan]
- dominating set if $\alpha = o(\log d)$ [Alon]
- maximum independent set if $\alpha = o\left(\frac{d}{\log d}\right)$ [Alon]

Ramifications:

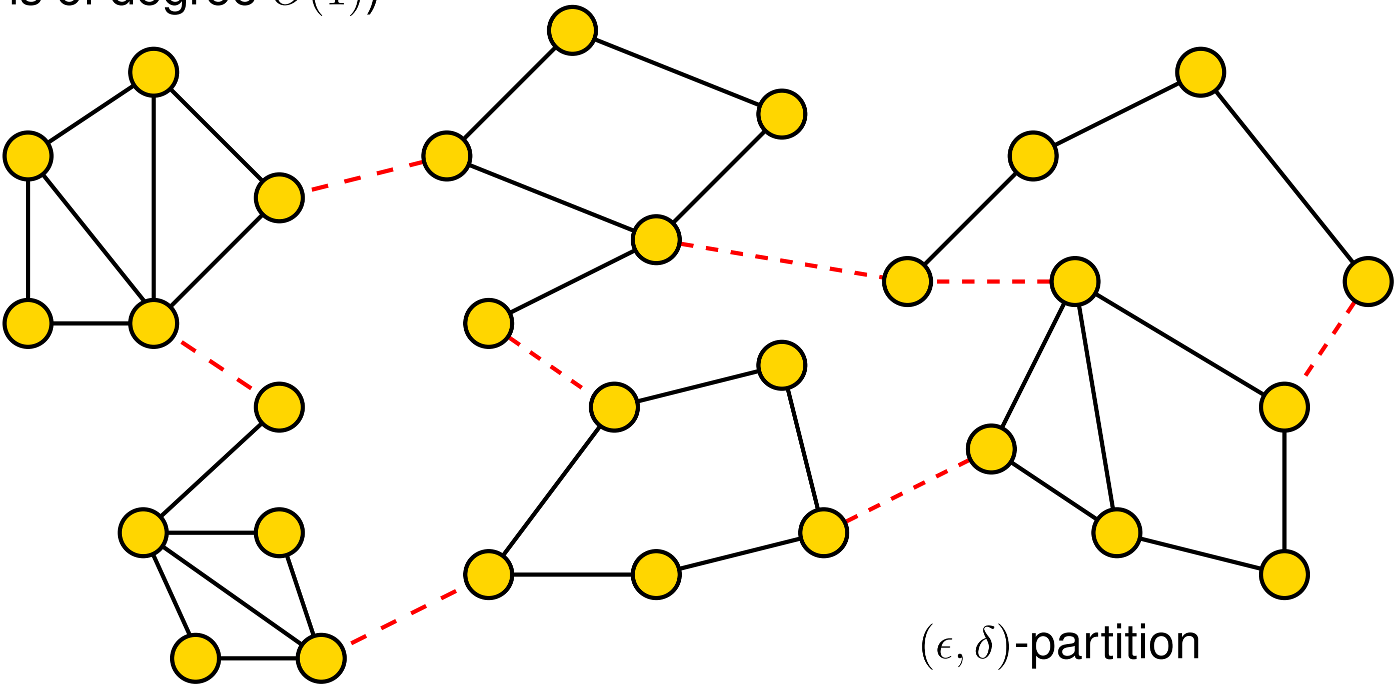
- no corresponding local distributed algorithm
- need $\Omega(\log n)$ rounds

Local Graph Partitions

[Hassidim, Kelner, Nguyen, O. 2009]

Hyperfinite Graphs

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

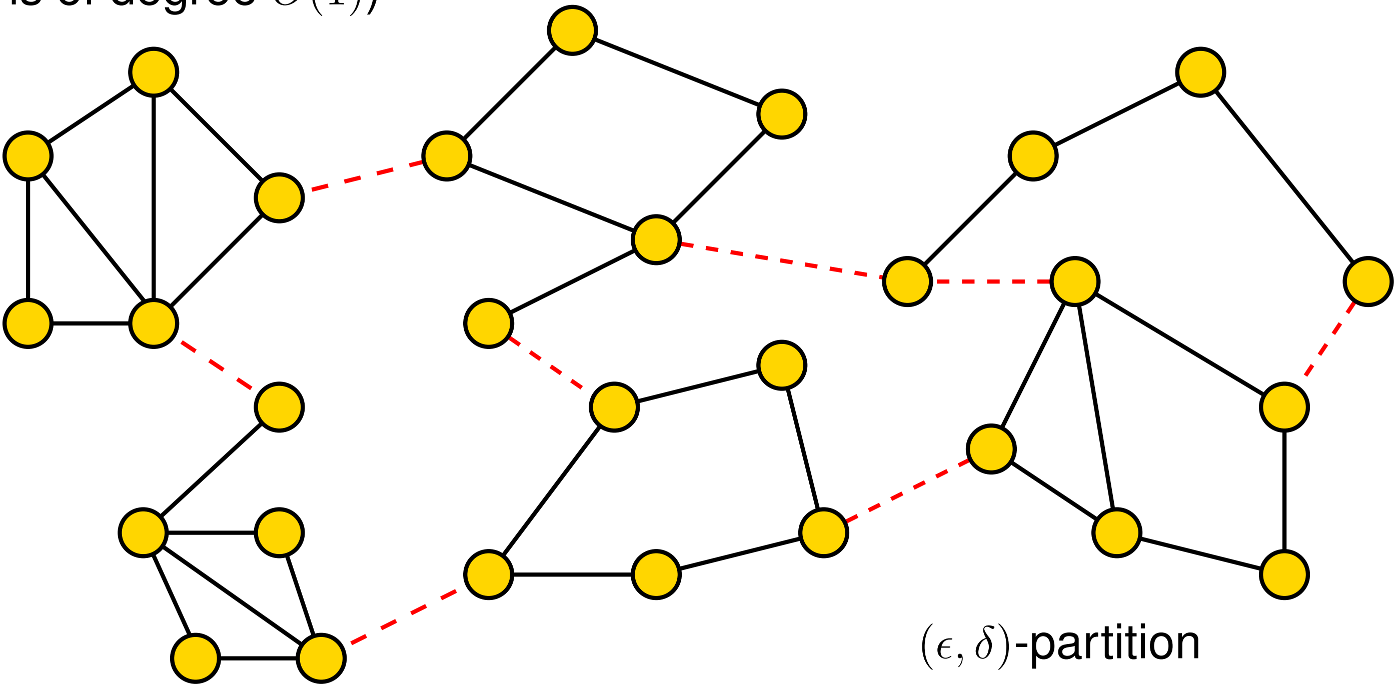


(ϵ, δ)-partition

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

Hyperfinite Graphs

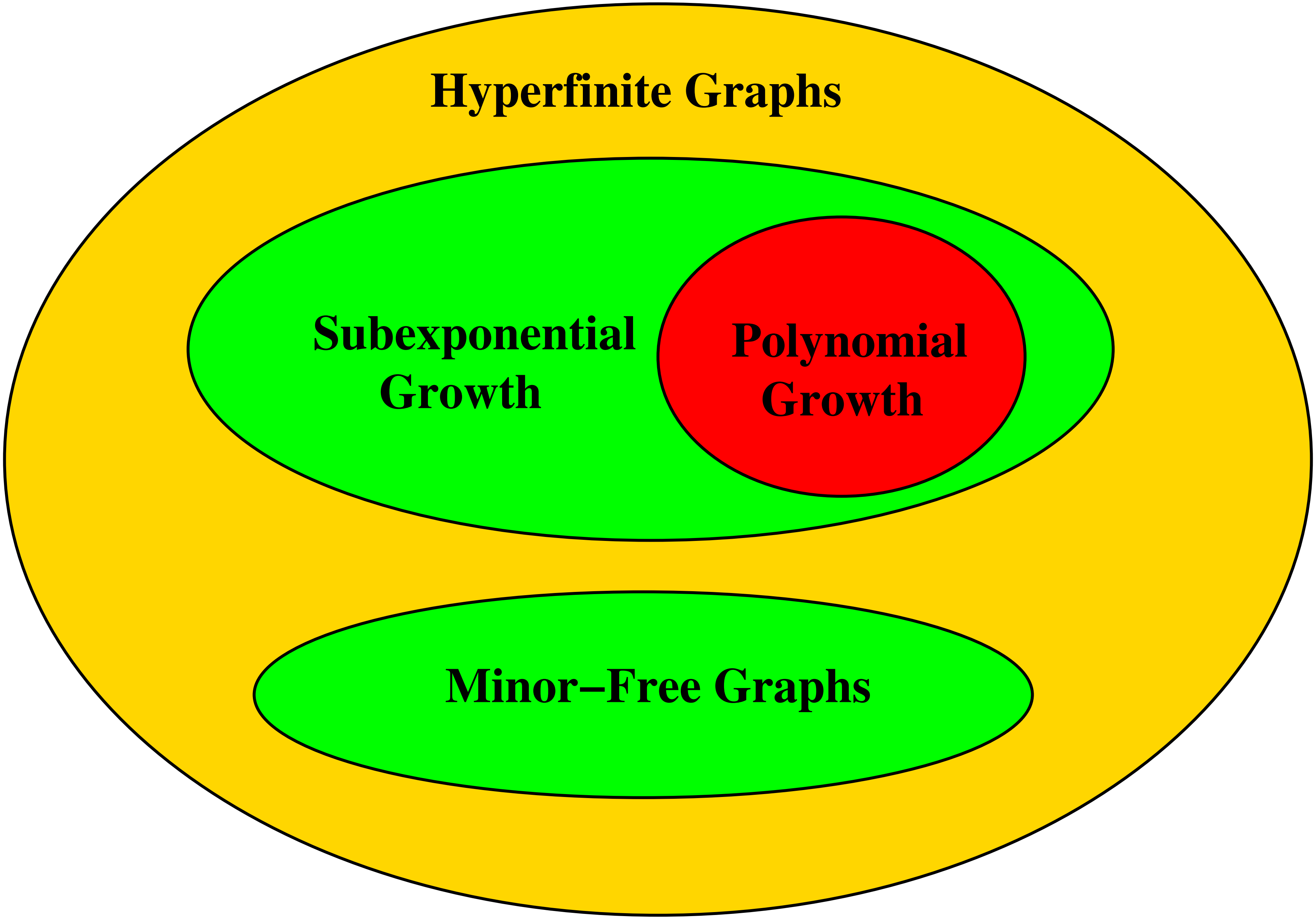
(All graphs of degree $O(1)$)



(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

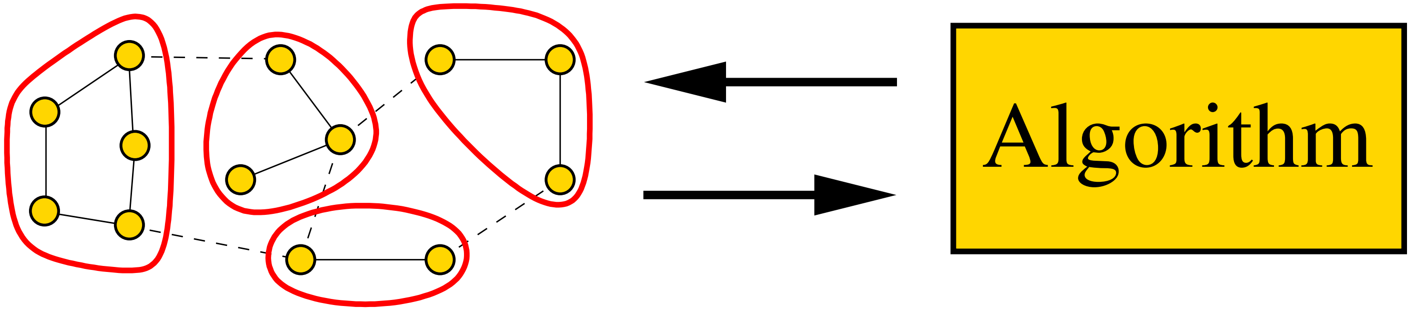
Subexponential Growth

Polynomial Growth

Minor–Free Graphs

Using a Partition

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

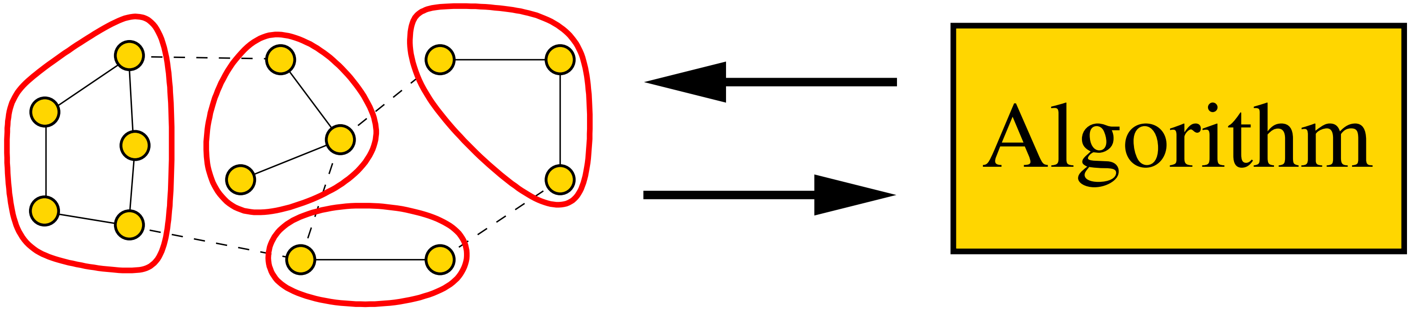


Algorithm

- 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:



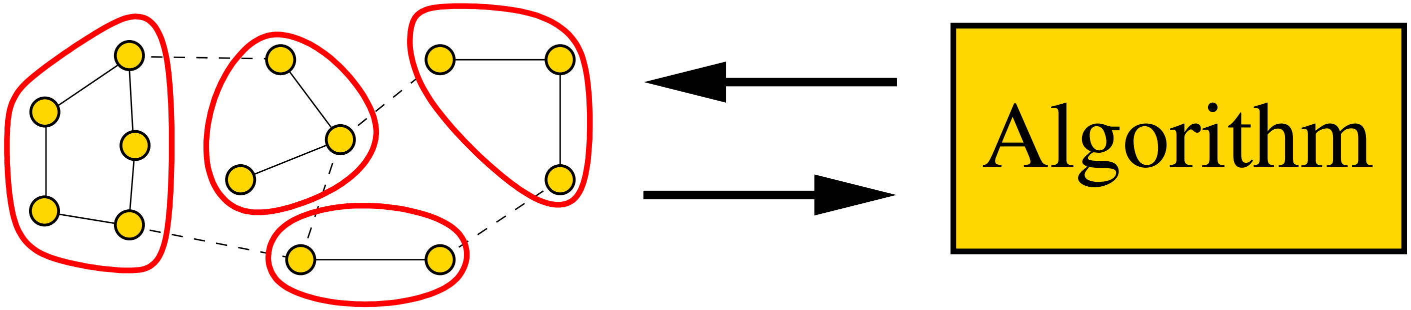
- Sample $O(1/\epsilon^2)$ vertices
- Compute minimum vertex cover for the sampled components
- Return the fraction of the sampled vertices in the covers

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:

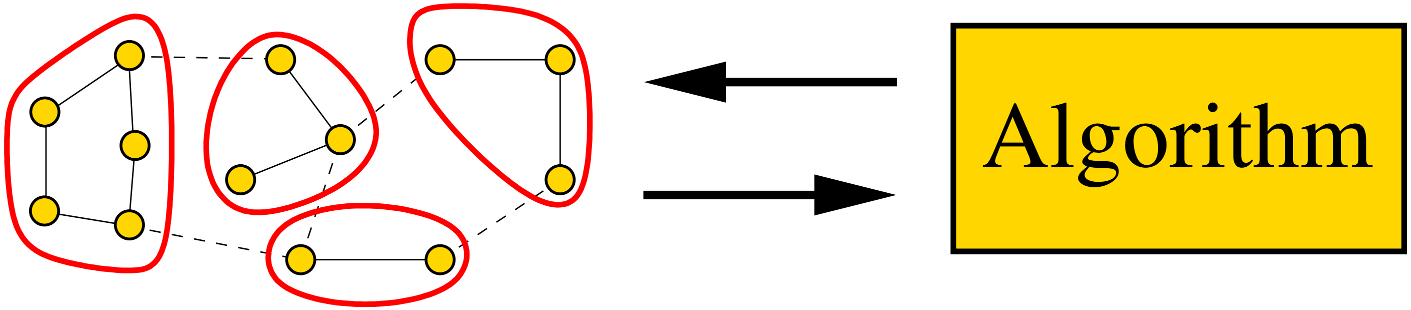


Bad news:

We don’t have a partition

Using a Partition

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



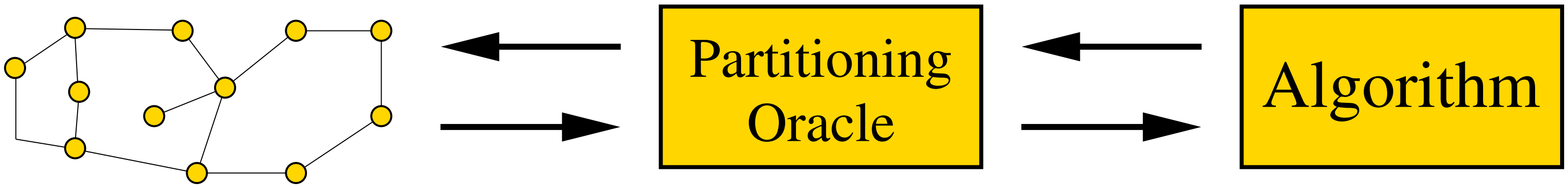
Bad news:

We don't have a partition

Good news:

We can compute it ourselves
without looking at the entire graph

Using a Partition



Partitioning Oracle

Algorithm

Bad news:

We don't have a partition

Good news:

We can compute it ourselves
without looking at the entire graph

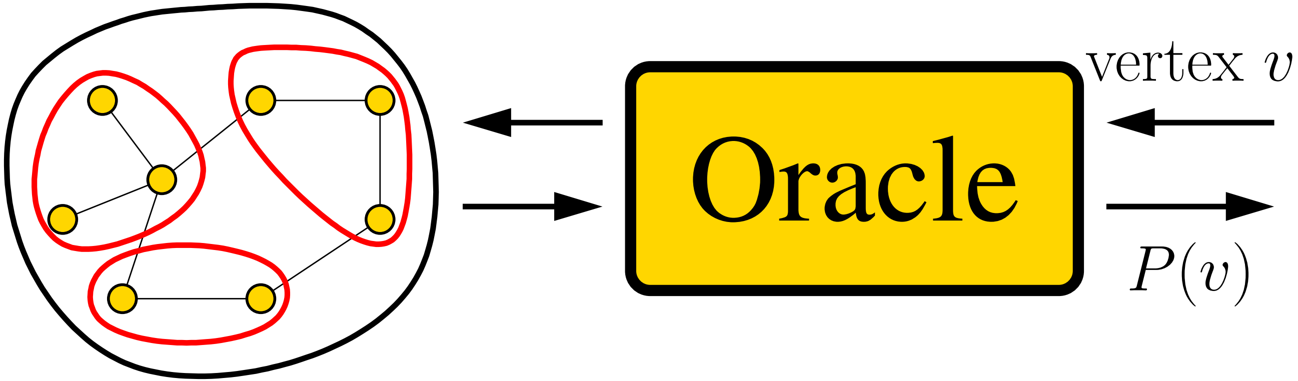
New Tool: Partitioning Oracles

Partitioning Oracle

$\mathcal{C} =$ fixed hyperfinite class

- oracle has query access to $G = (V, E)$

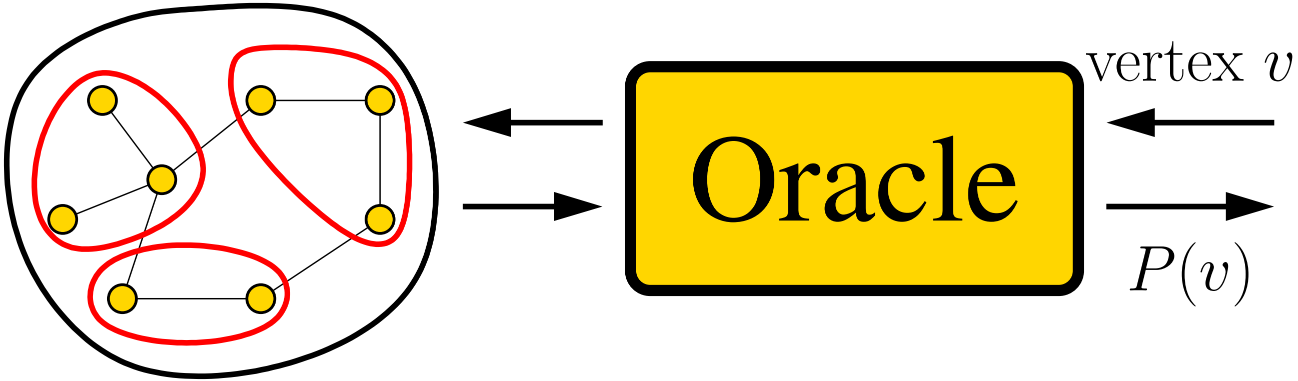
  (G need not be in $\mathcal{C}$)



Partitioning Oracle

$\mathcal{C} =$ fixed hyperfinite class

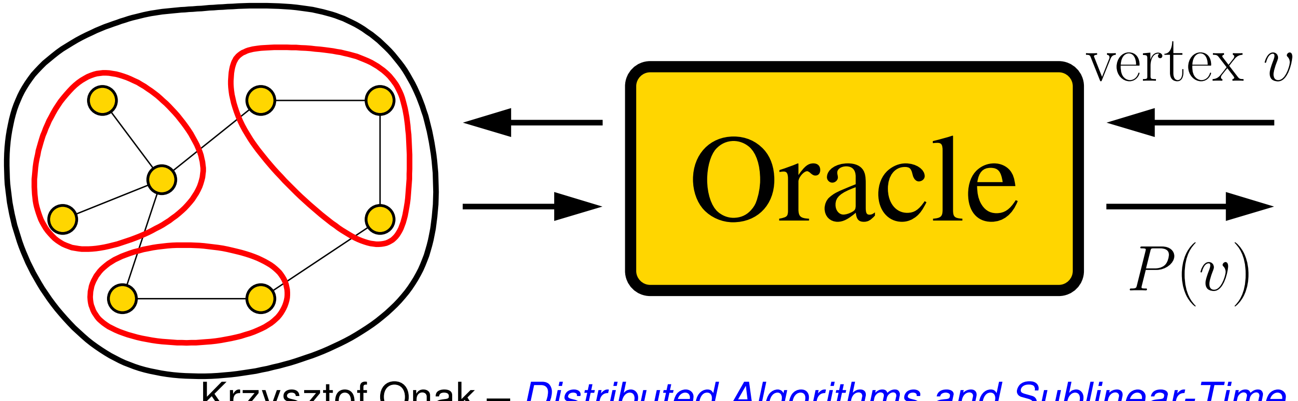
- oracle has query access to $G = (V, E)$
  (G need not be in $\mathcal{C}$)
- oracle provides query access to partition P of V;
  for each v, oracle returns $P(v) \subseteq V$ s.t. $v \in P(v)$



Partitioning Oracle

$\mathcal{C} =$ fixed hyperfinite class

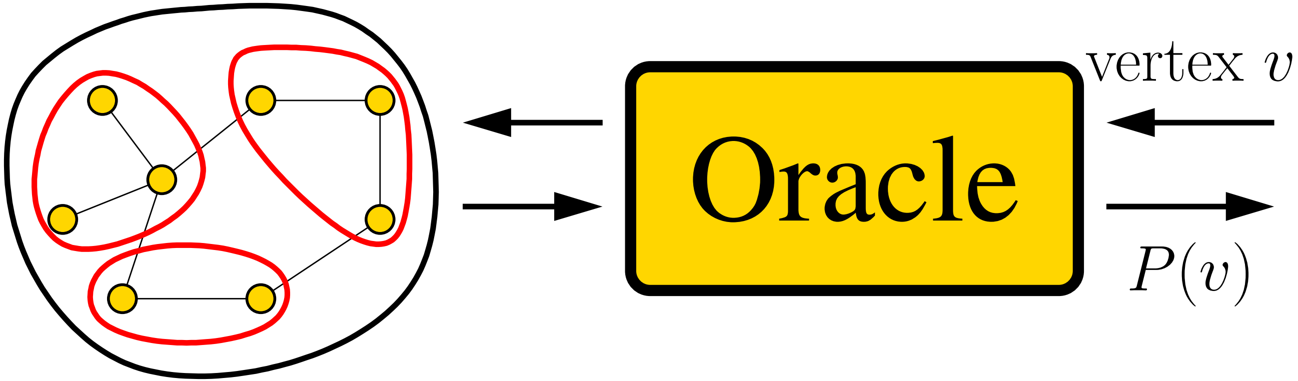
- oracle has query access to $G = (V, E)$
  (G need not be in $\mathcal{C}$)
- oracle provides query access to partition P of V;
  for each v, oracle returns $P(v) \subseteq V$ s.t. $v \in P(v)$
- Properties of P:
  - each $|P(v)| = O(1)$



Partitioning Oracle

$\mathcal{C} =$ fixed hyperfinite class

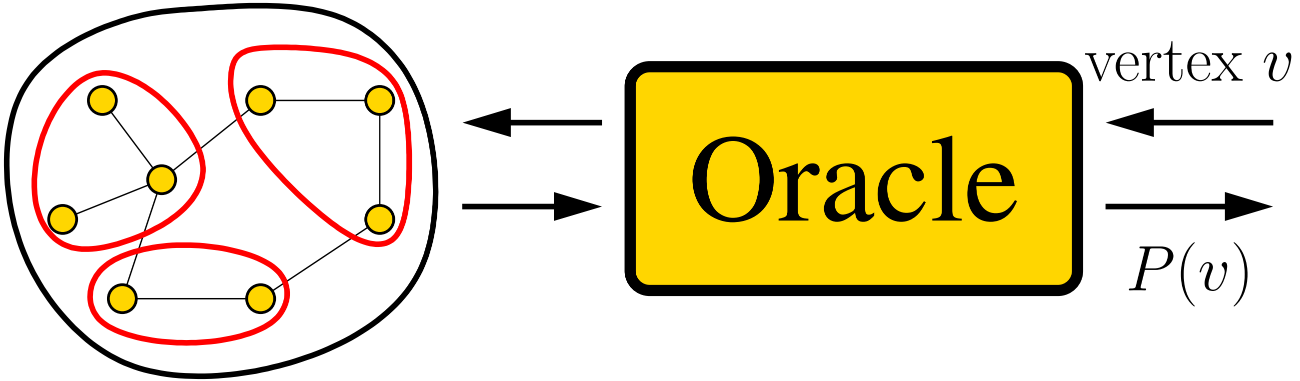
- oracle has query access to $G = (V, E)$
  (G need not be in $\mathcal{C}$)
- oracle provides query access to partition P of V; for each v, oracle returns $P(v) \subseteq V$ s.t. $v \in P(v)$
- Properties of P:
  - each $|P(v)| = O(1)$
  - If $G \in \mathcal{C}$, number of cut edges $\leq \epsilon n$ 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: $2^{\mathrm{poly}(d/\epsilon)}$

Also :

- For polynomial growth [Jung, Shah]:
  - Query complexity: $\mathrm{poly}(d/\epsilon)$

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

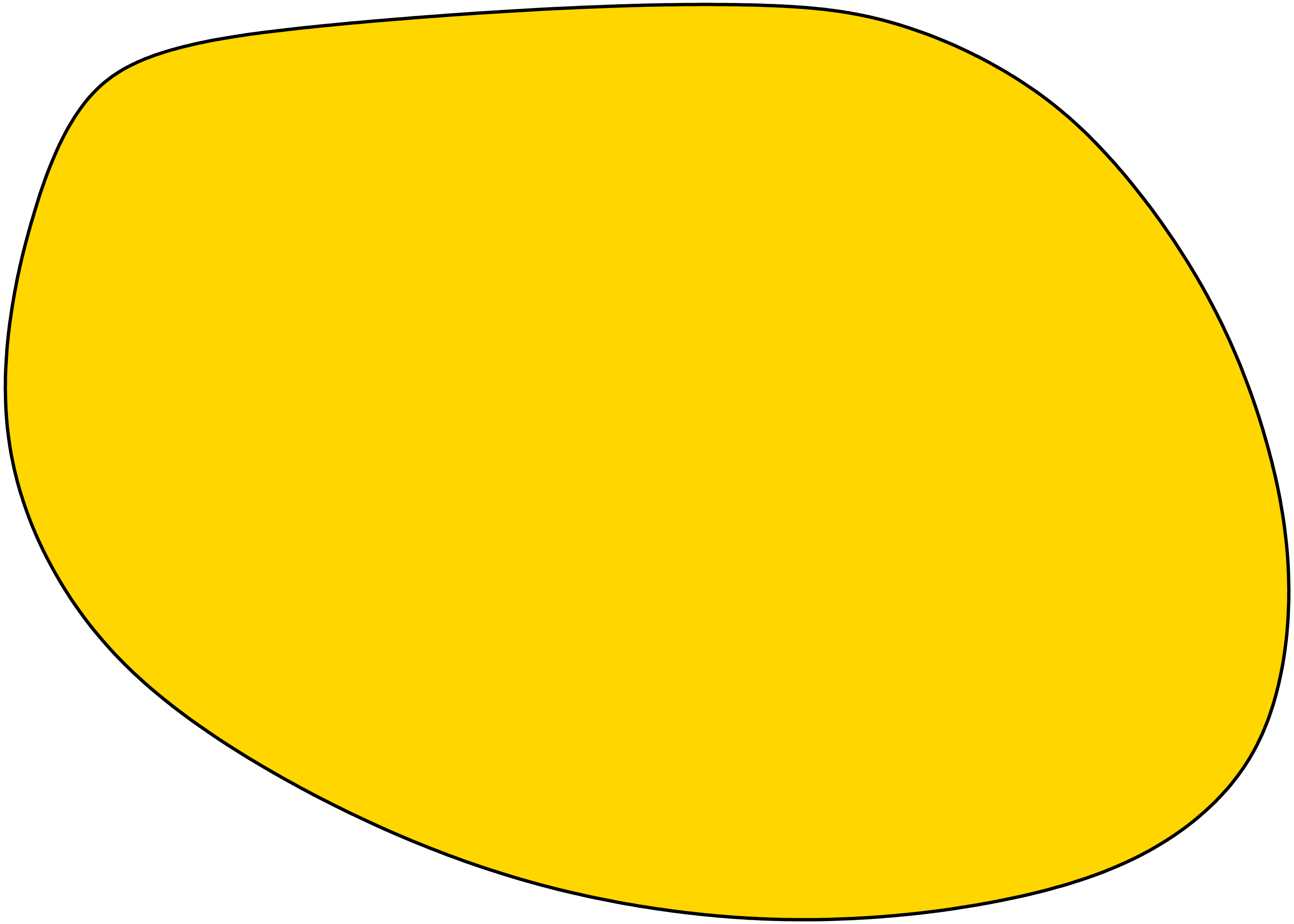
- For hyperfinite graphs, can get $\pm\epsilon n$ approximation to:
  - minimum vertex cover size
    (that is also the independence number)
  - minimum dominating set size

  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

Simplest Oracle

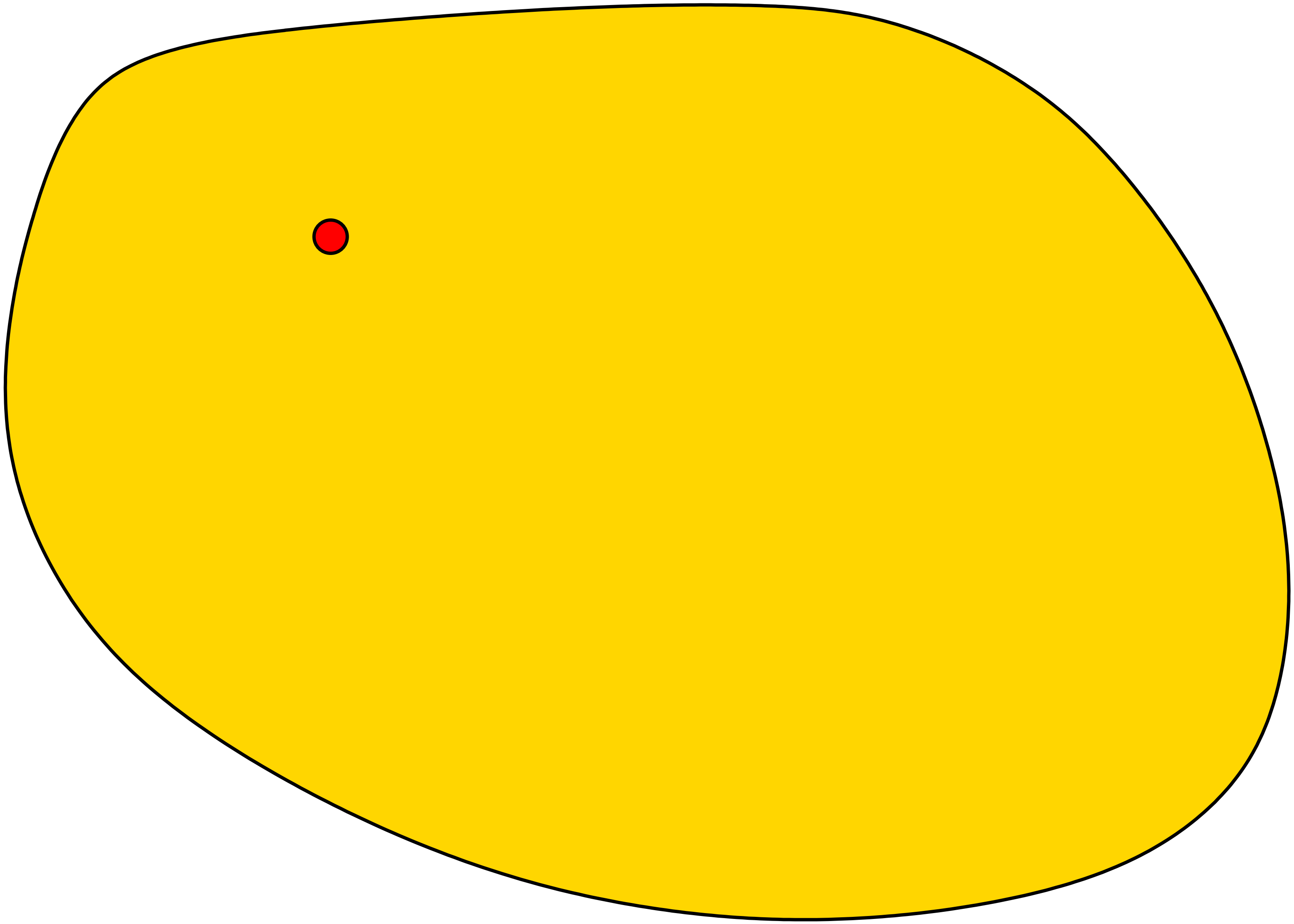
Iterative Procedure

Global procedure:



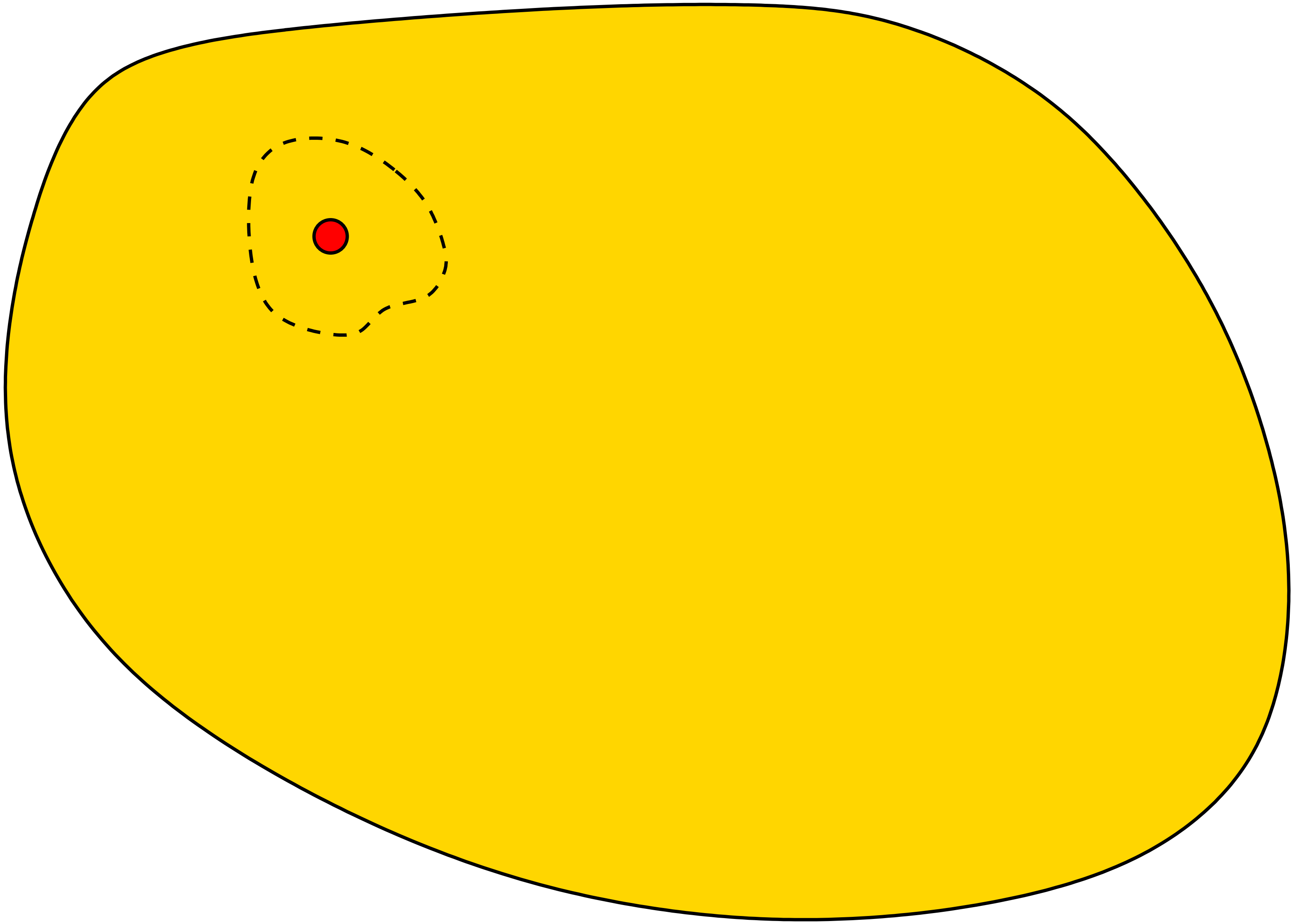
Iterative Procedure

Global procedure:



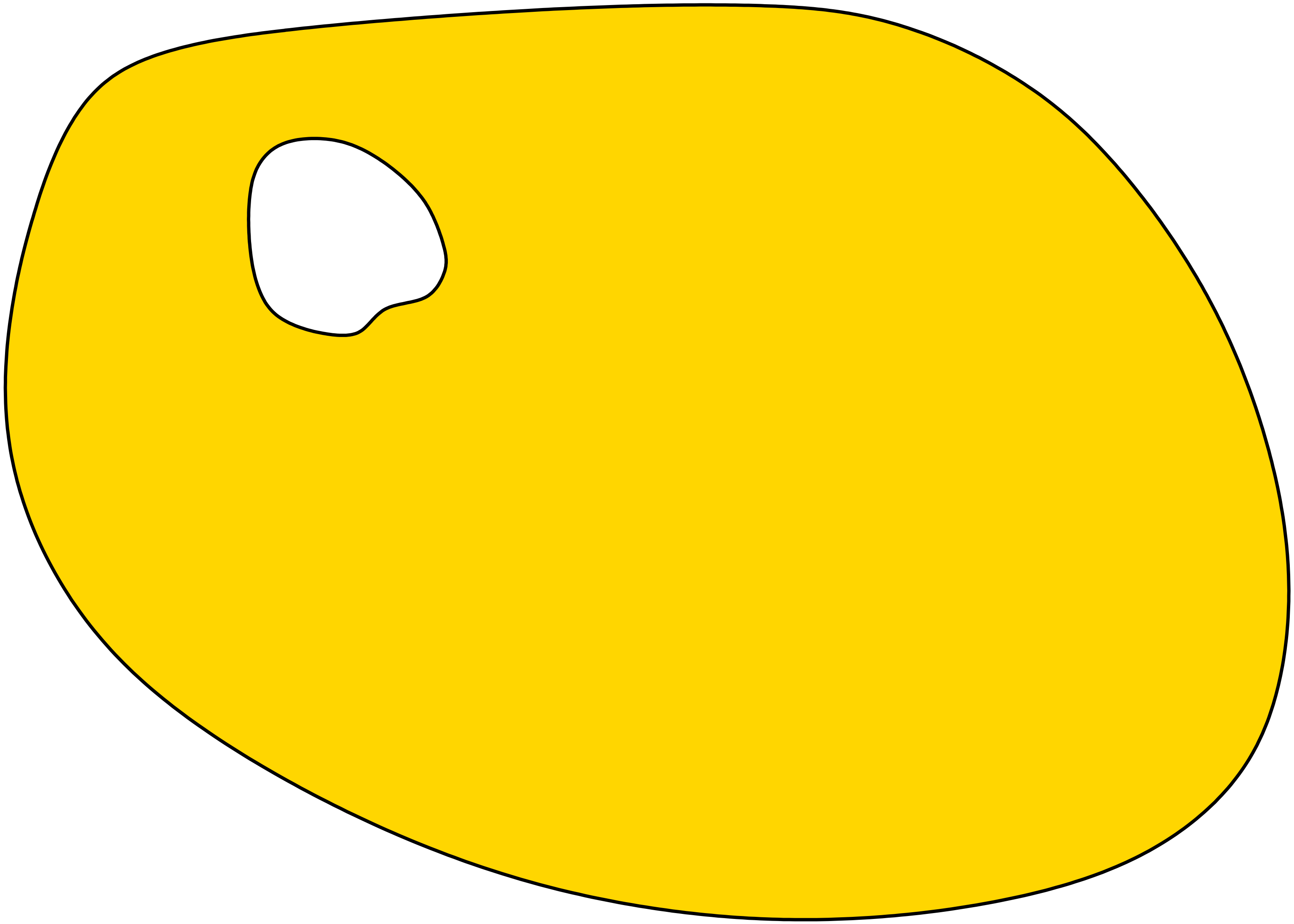
Iterative Procedure

Global procedure:



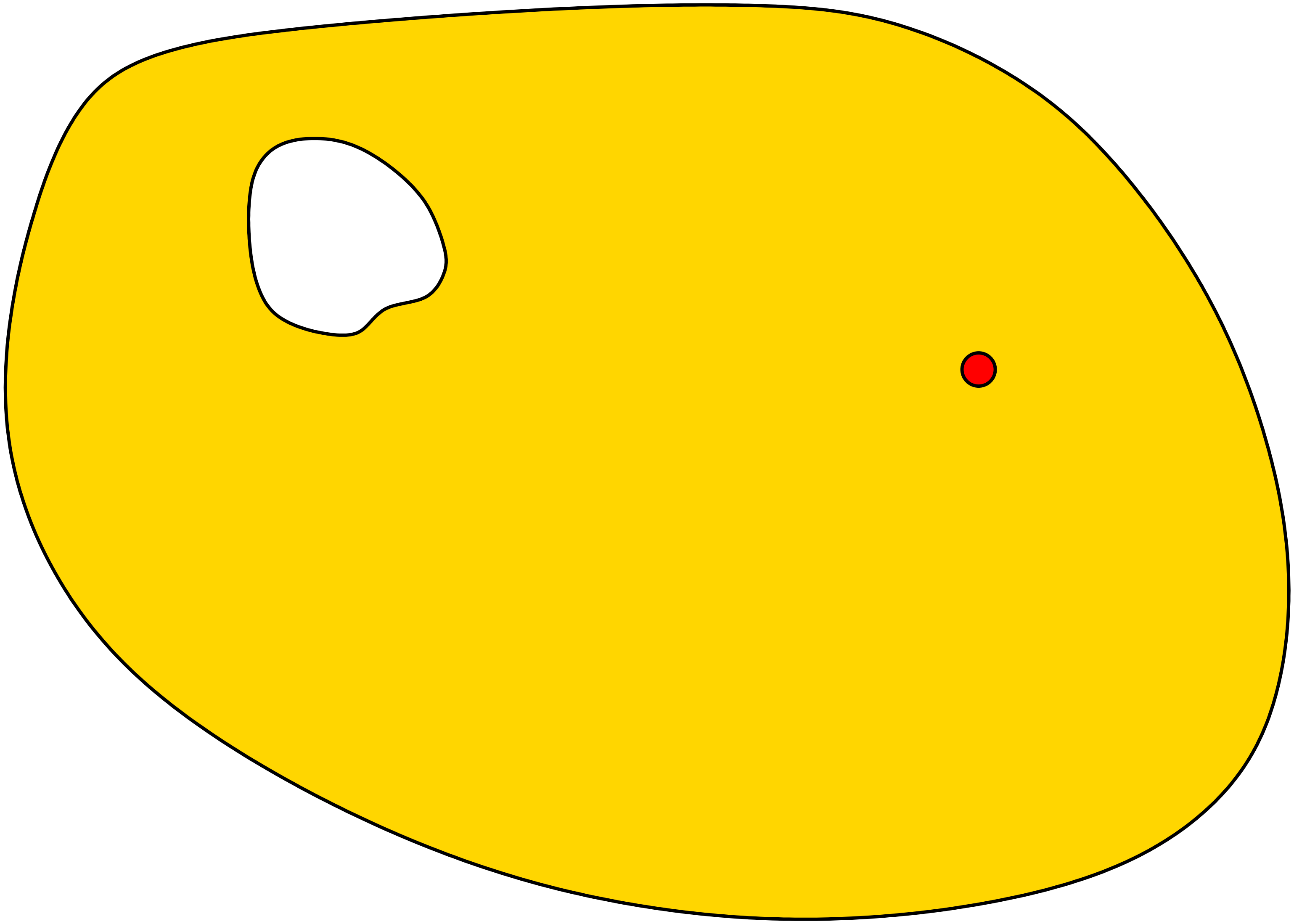
Iterative Procedure

Global procedure:



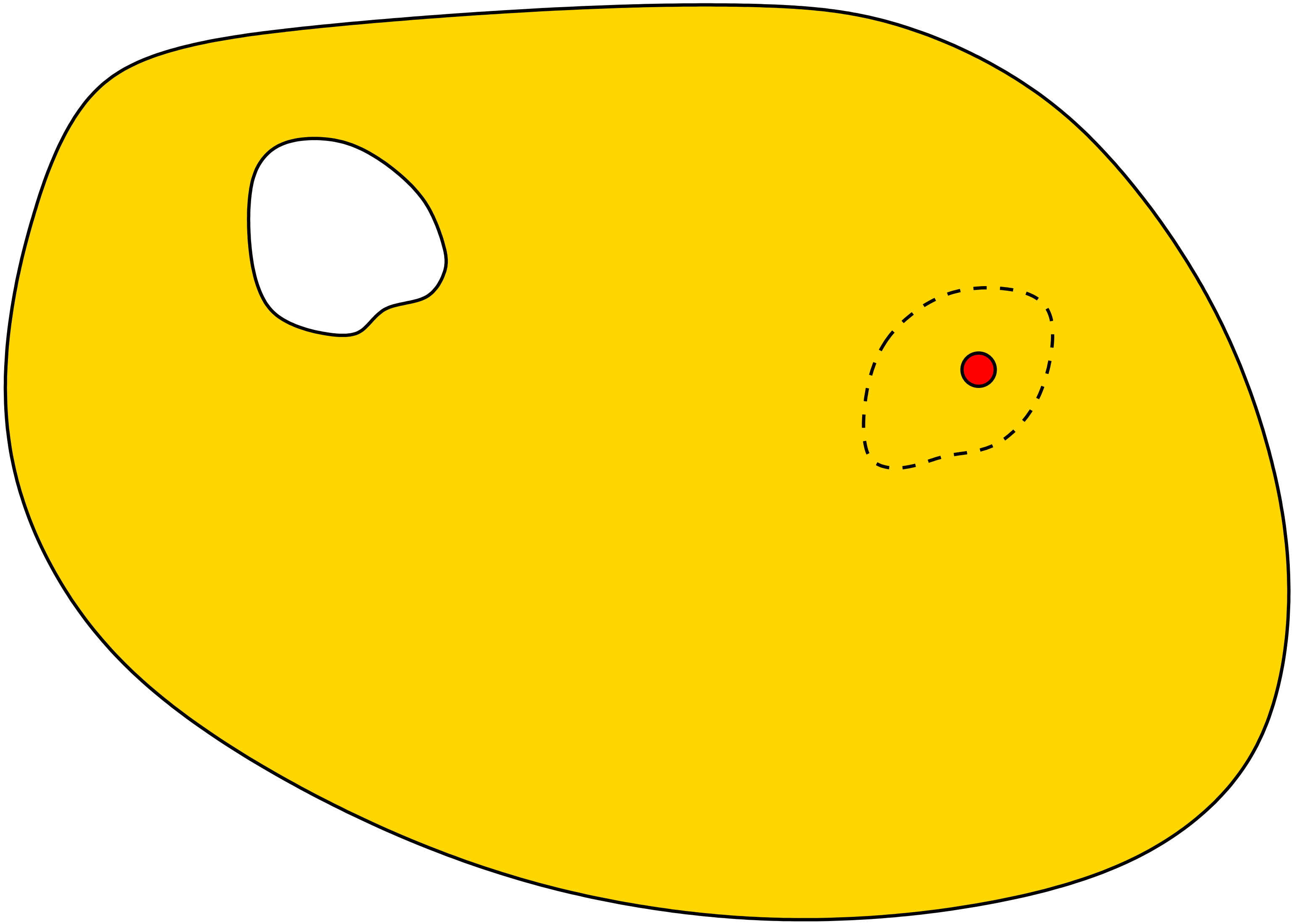
Iterative Procedure

Global procedure:



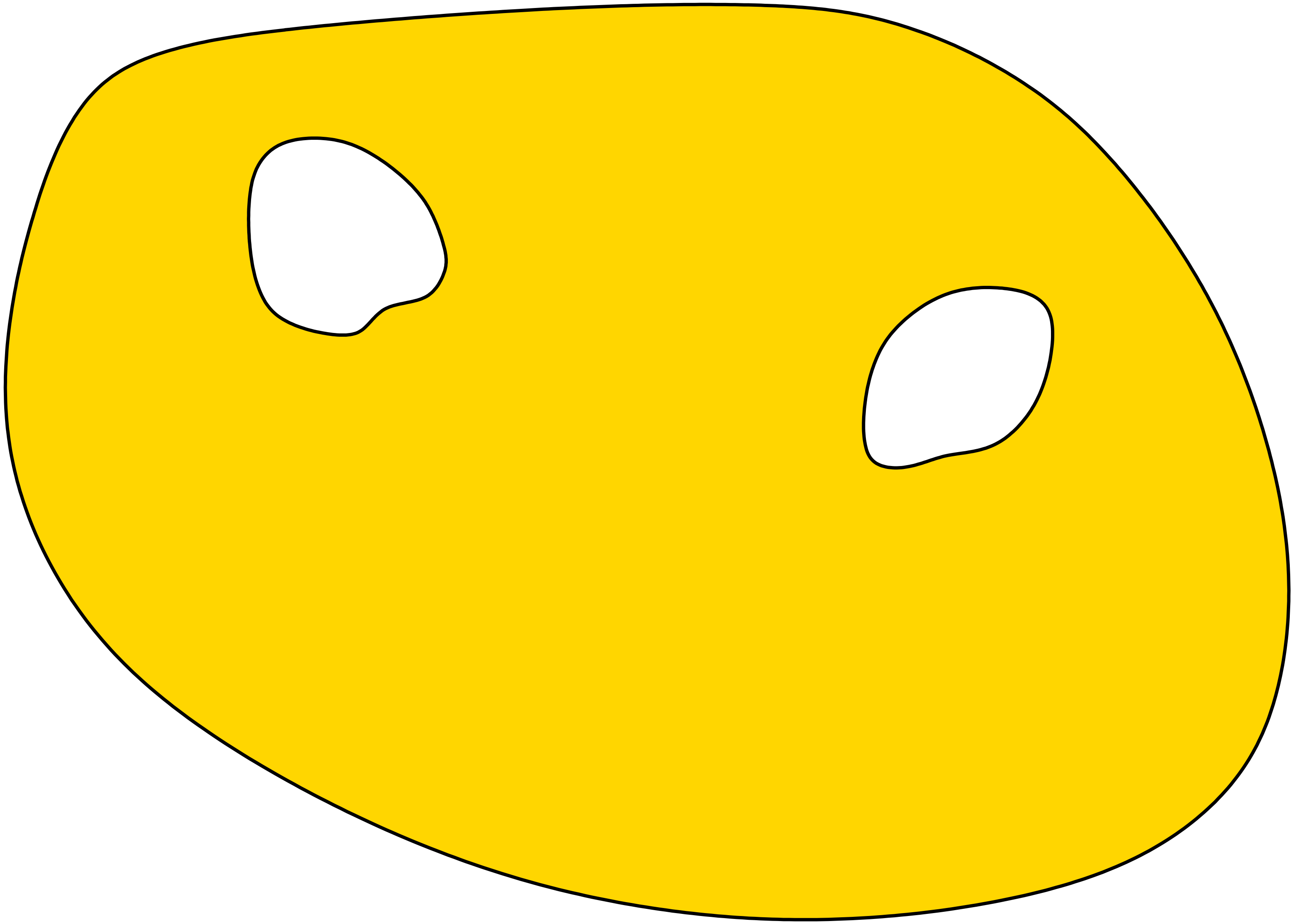
Iterative Procedure

Global procedure:



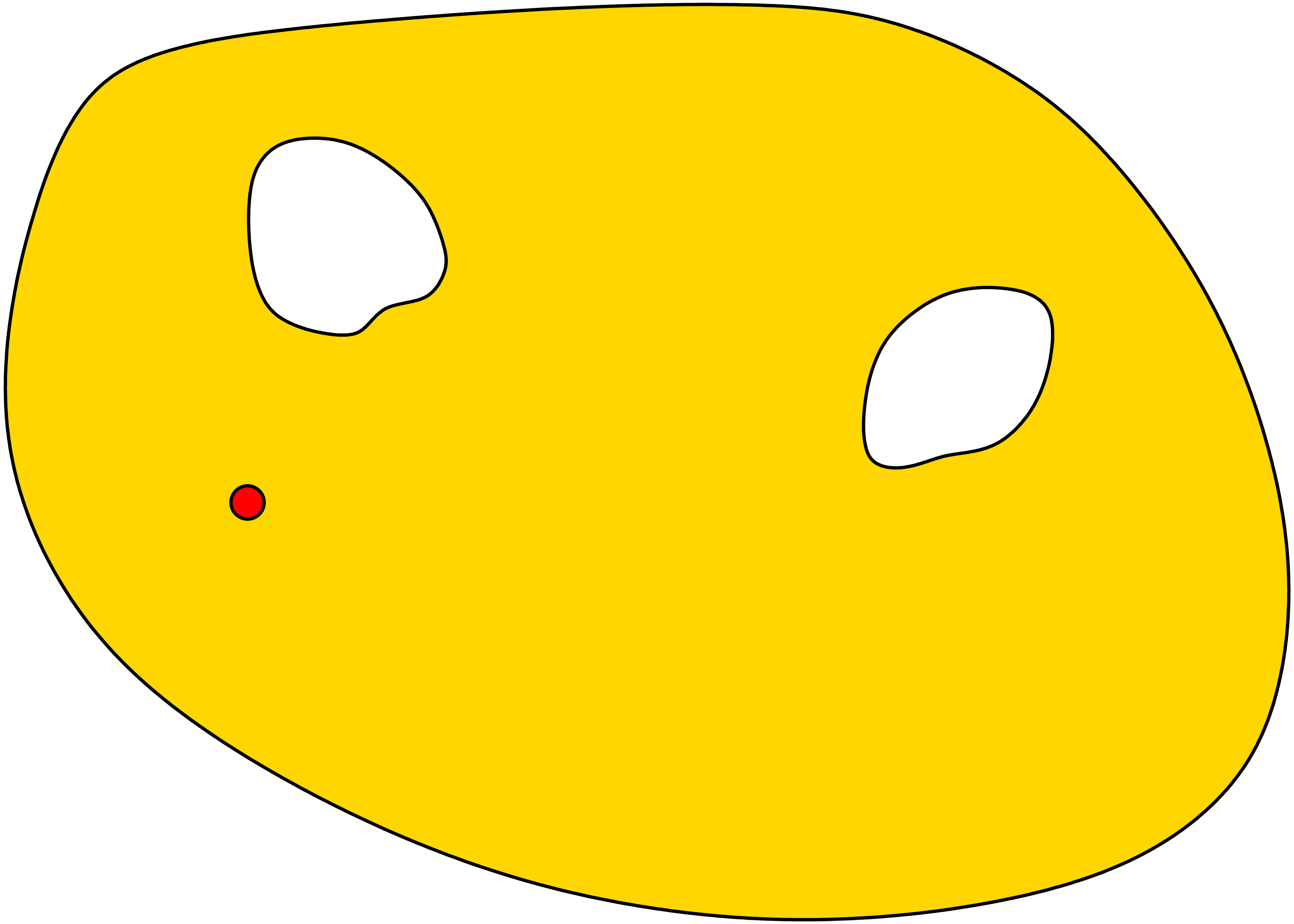
Iterative Procedure

Global procedure:



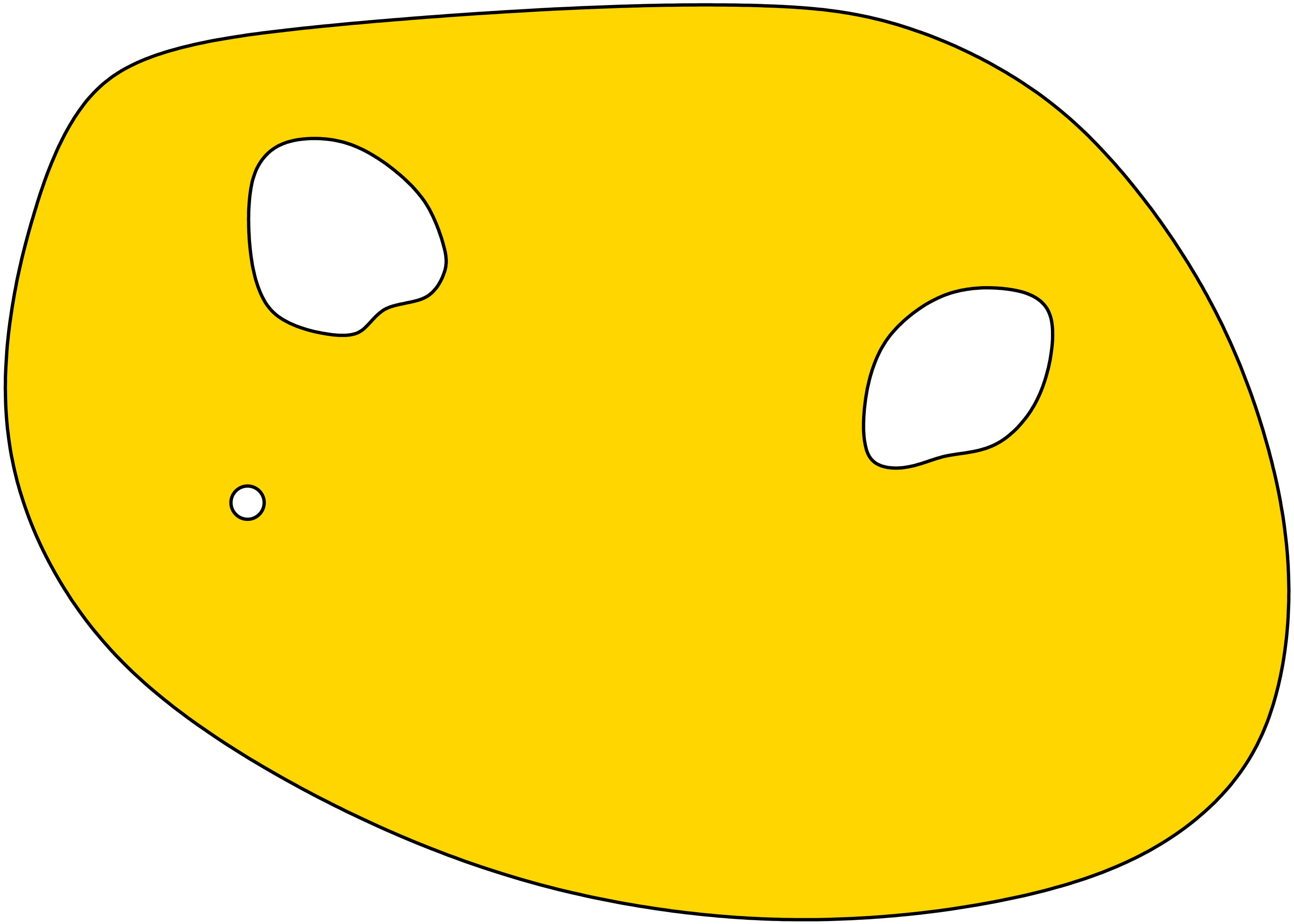
Iterative Procedure

Global procedure:

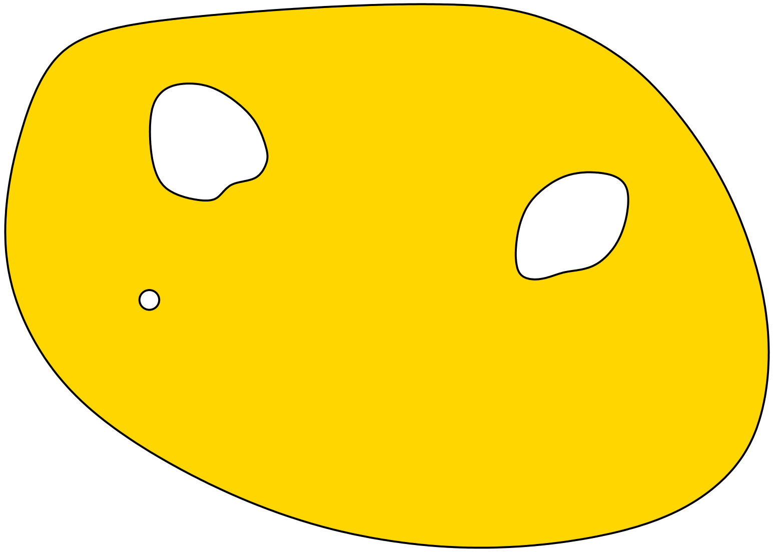


Iterative Procedure

Global procedure:



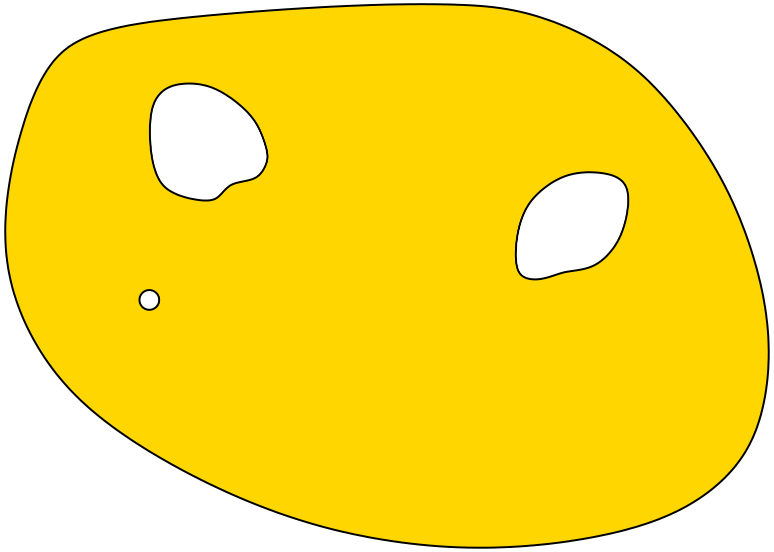
Local simulation



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 to 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

Thank you!