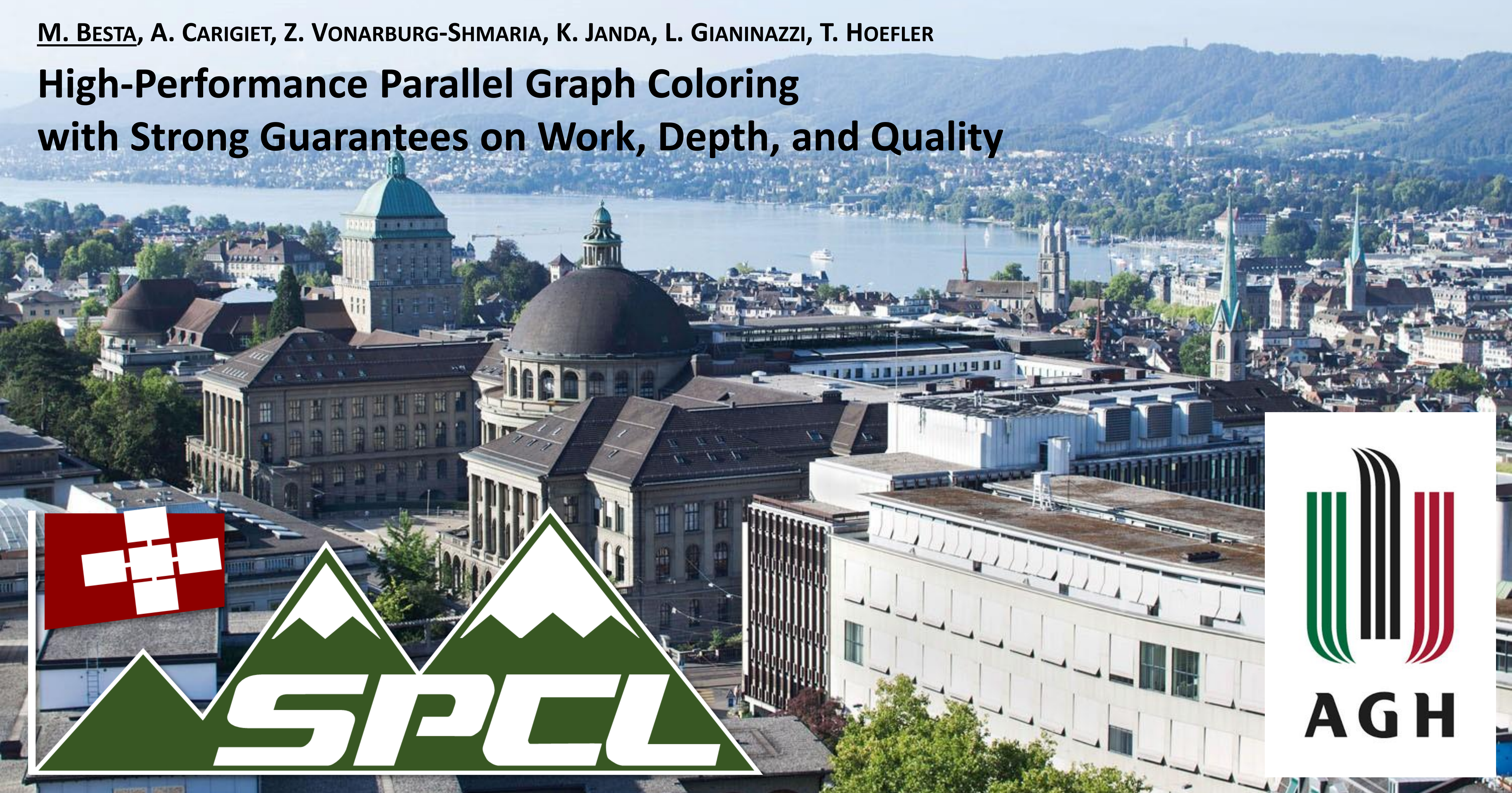
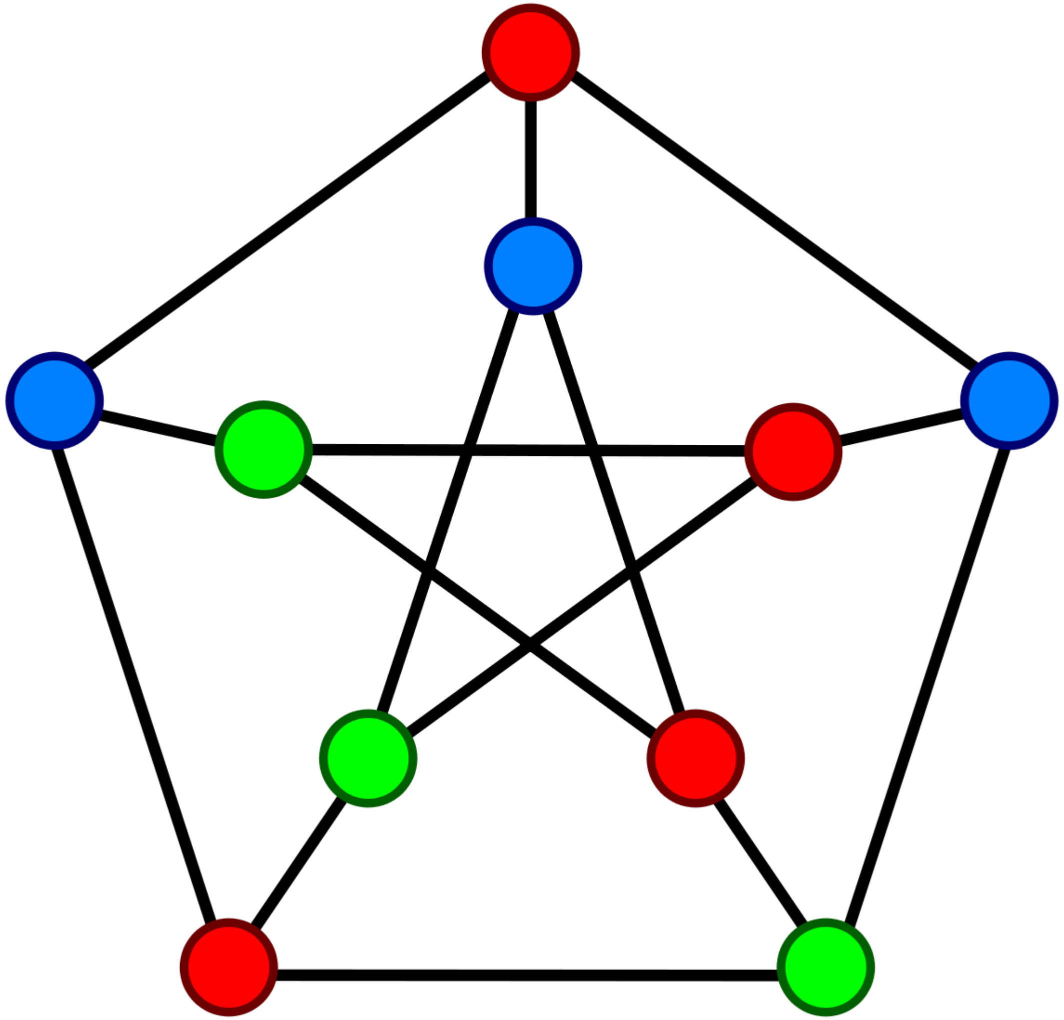
M. Besta, A. Carigiet, Z. Vonarburg-Shmaria, K. Janda, L. Gianinazzi, T. Hoefler

High-Performance Parallel Graph Coloring with Strong Guarantees on Work, Depth, and Quality

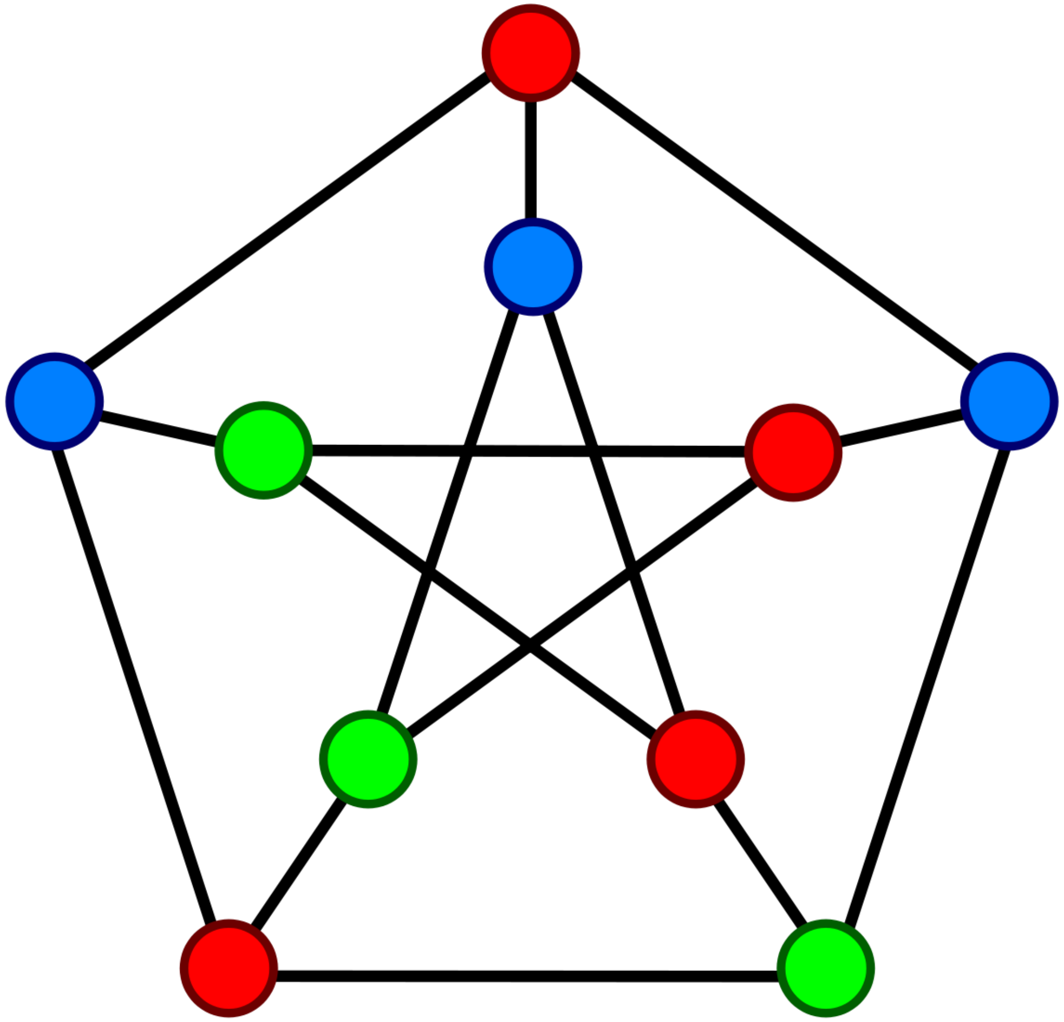


Graph coloring



Graph coloring

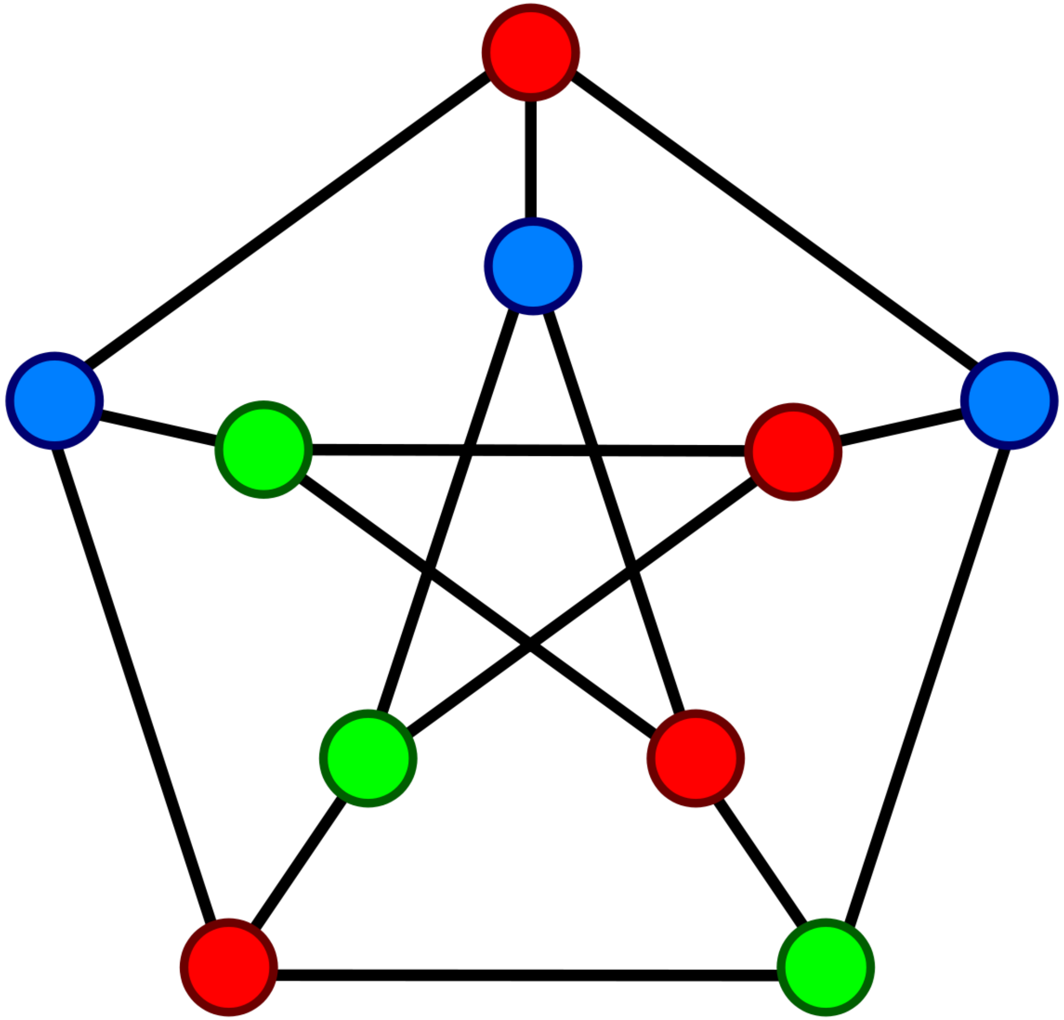
Fundamental graph problem



Graph coloring

Fundamental graph problem

Assign numbers, i.e., **colors**, to each vertex, such that **no adjacent vertices have the same color.**

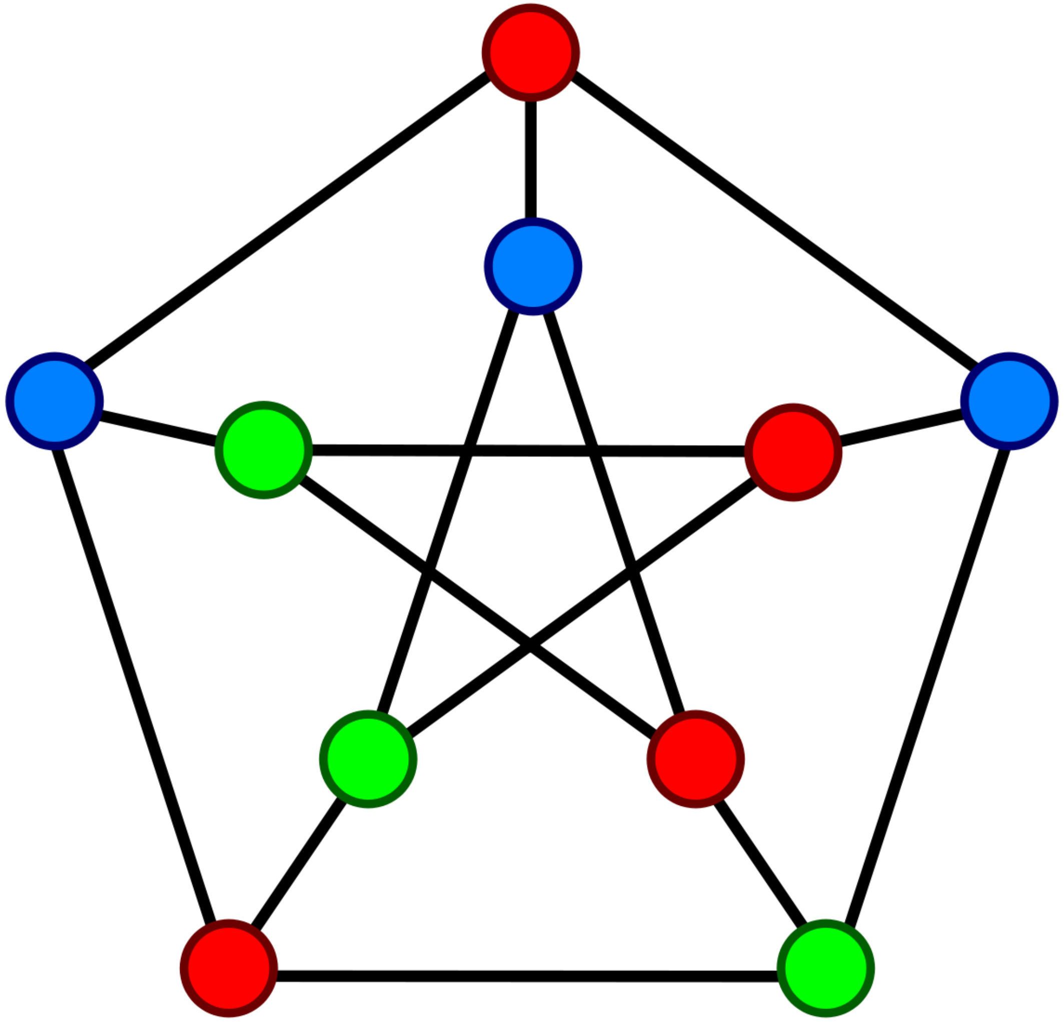


Graph coloring

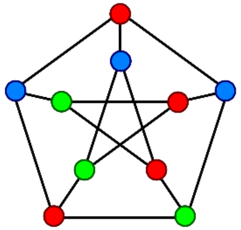
Fundamental graph problem

Assign numbers, i.e., **colors**, to each vertex, such that **no adjacent vertices have the same color.**

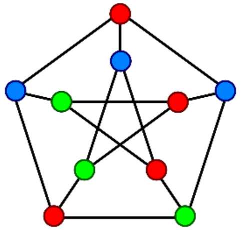
Goal: minimize the number of used colors



Graph coloring: applications

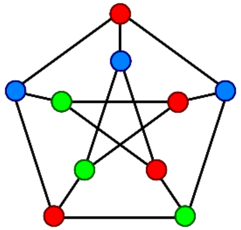


Graph coloring: applications

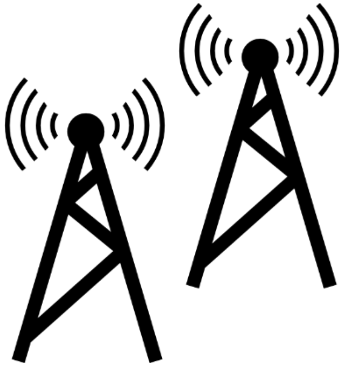


Constructing a schedule or a time-table

Graph coloring: applications

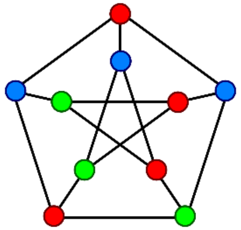


Assigning frequencies to radio towers

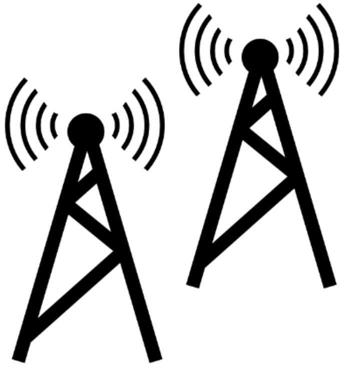


Constructing a schedule or a time-table

Graph coloring: applications

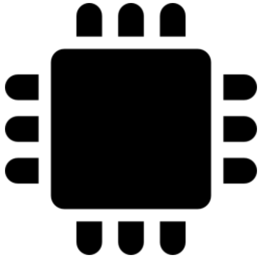


Assigning frequencies to radio towers

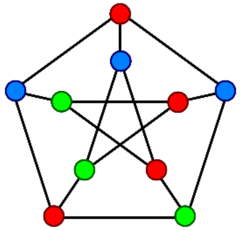


Constructing a schedule or a time-table

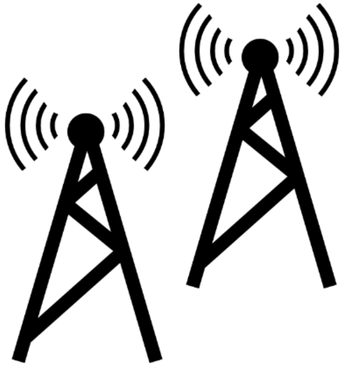
Allocating registers



Graph coloring: applications

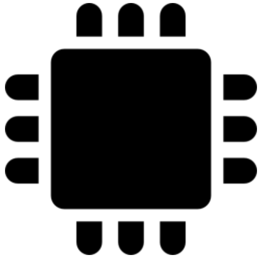


Assigning frequencies to radio towers



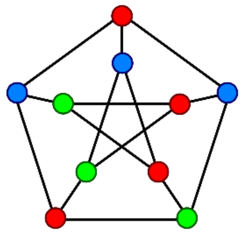
Constructing a schedule or a time-table

Allocating registers



Coloring maps

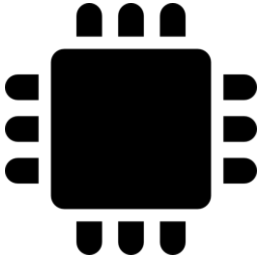
Graph coloring: applications



Assigning frequencies to radio towers

Constructing a schedule or a time-table

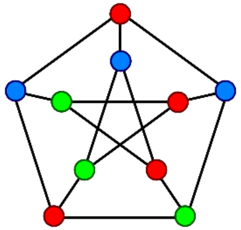
Allocating registers



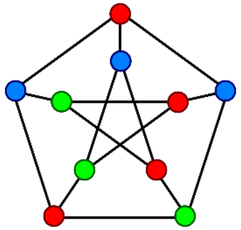
Coloring maps

Solving Sudoku ☺

Graph coloring & today's graph computations

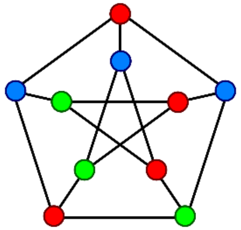


Graph coloring & today's graph computations



Graph datasets are **huge**

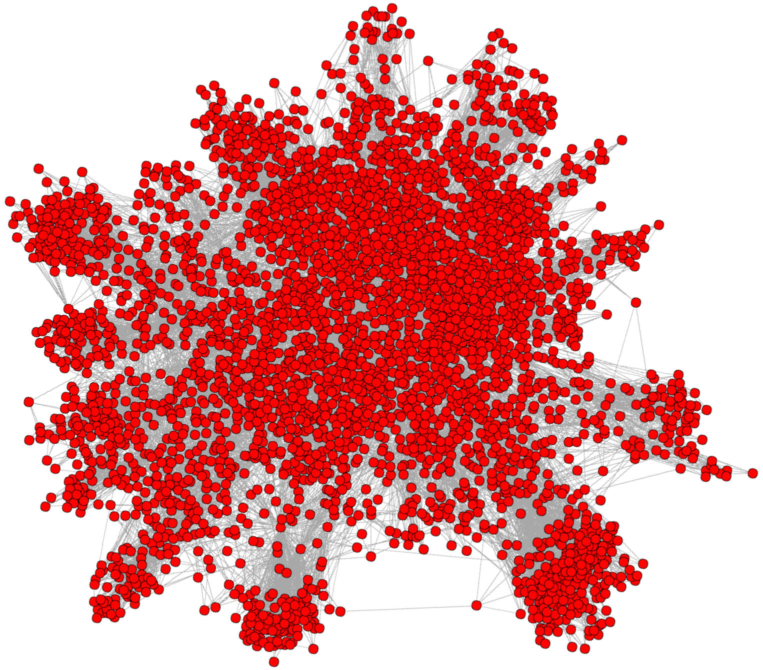
Graph coloring & today's graph computations



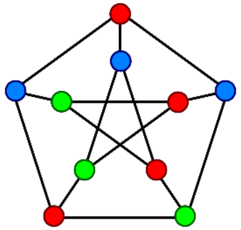
Graph datasets are **huge**

[1] Heng Lin et al.: ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds, **SC18, Gordon Bell Finalist**

> 233 TB
271 billion vertices,
12 trillion edges [1]



Graph coloring & today's graph computations

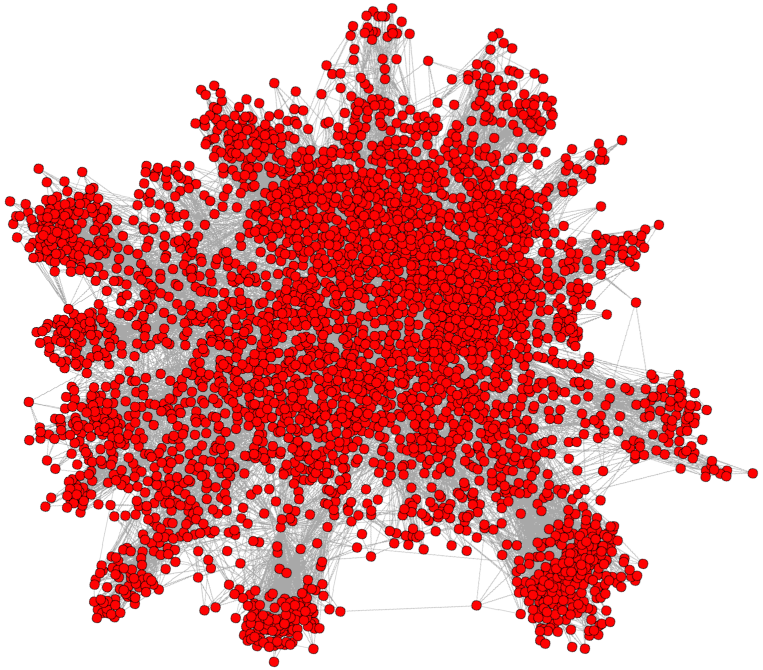


Graph datasets are **huge**

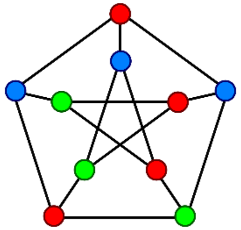
Optimal graph coloring is **NP-complete**

[1] Heng Lin et al.: ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds, **SC18, Gordon Bell Finalist**

> 233 TB
271 billion vertices,
12 trillion edges [1]



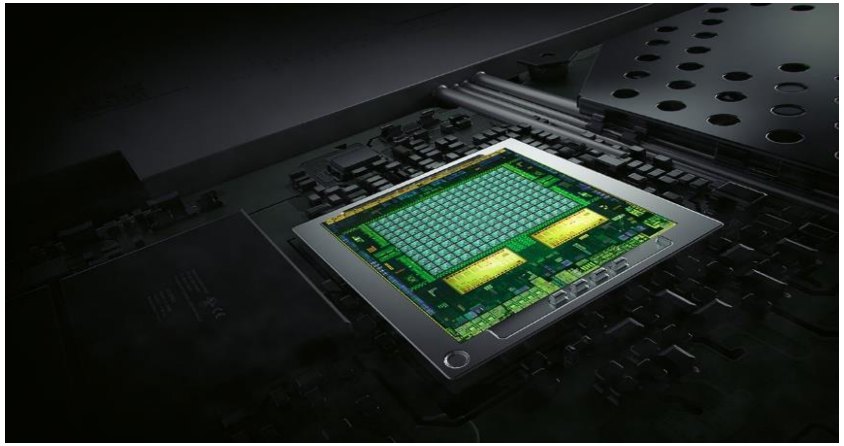
Graph coloring & today's graph computations



Graph datasets are **huge**

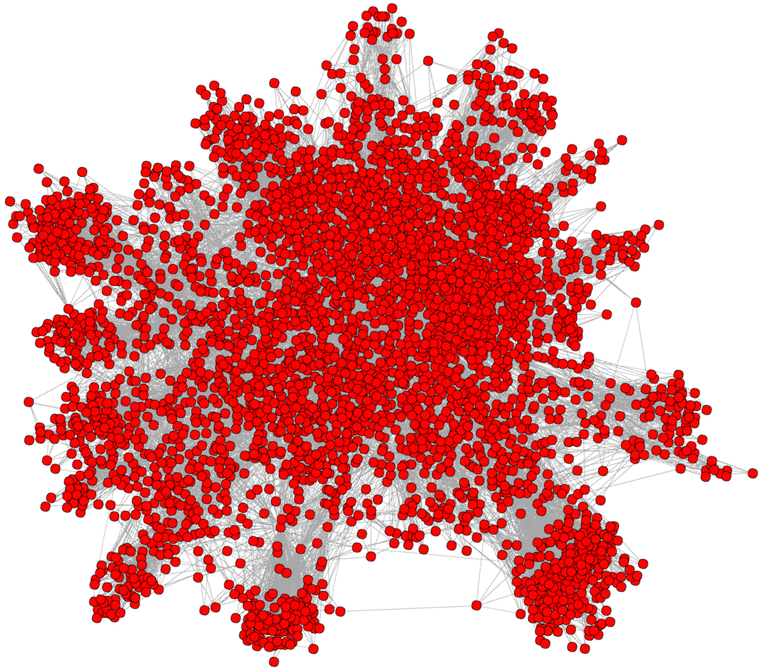
Optimal graph coloring is **NP-complete**

We have massive **parallelism**

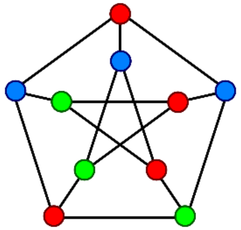


[1] Heng Lin et al.: ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds, **SC18, Gordon Bell Finalist**

> 233 TB
271 billion vertices,
12 trillion edges [1]



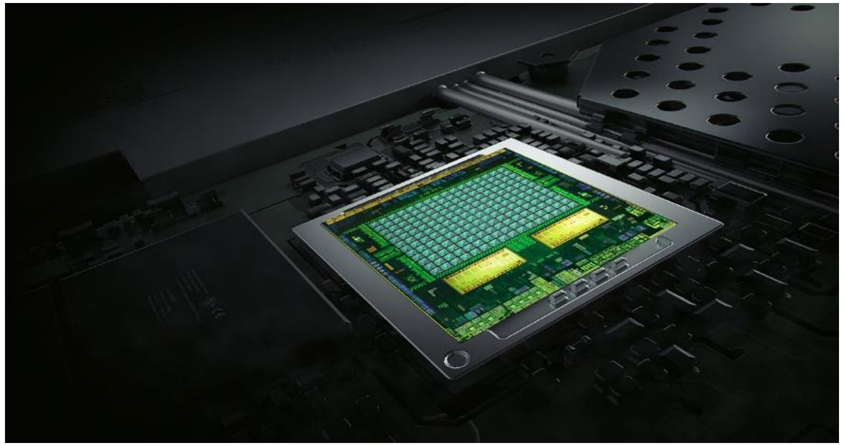
Graph coloring & today's graph computations

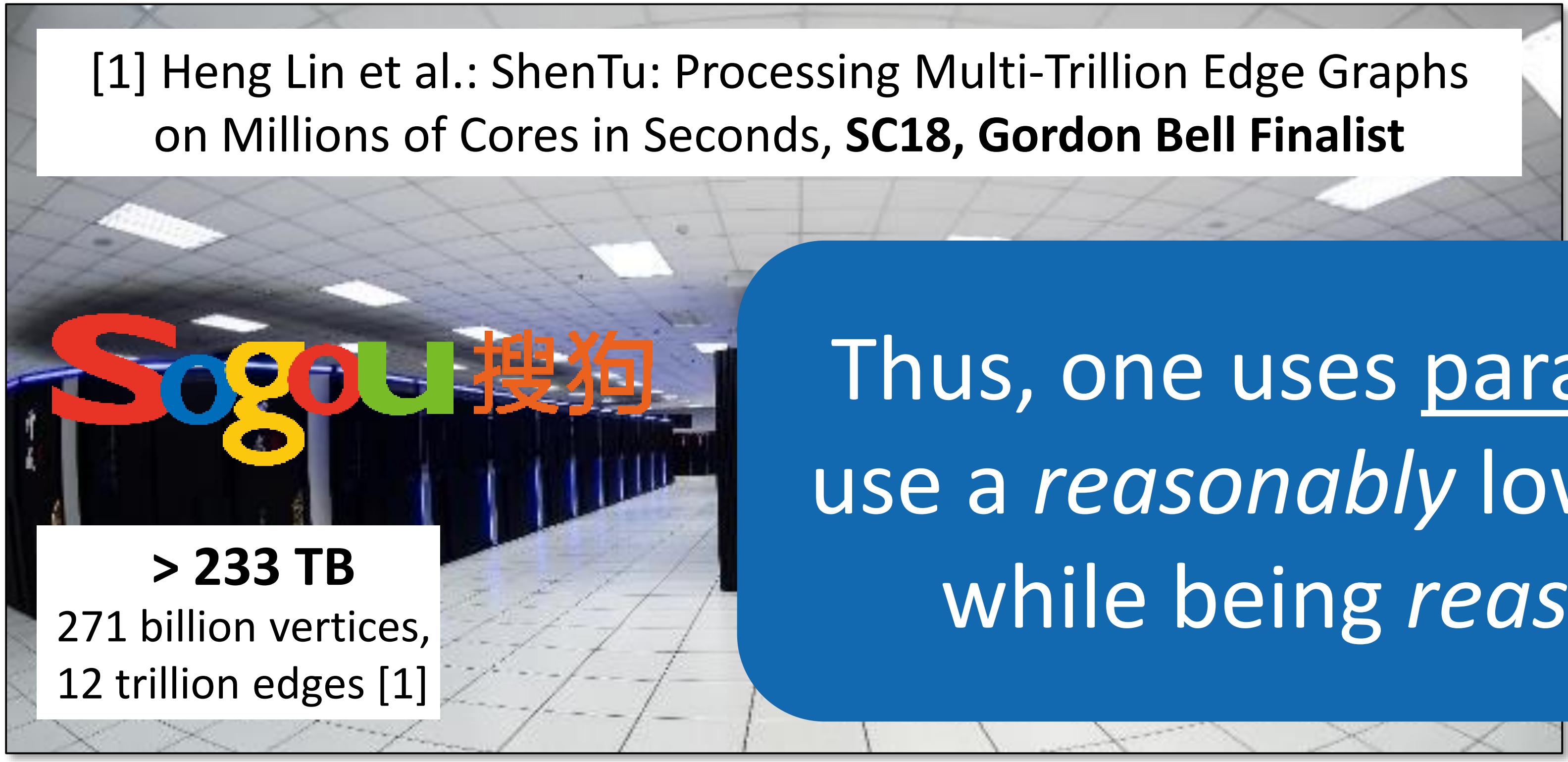


Graph datasets are **huge**

Optimal graph coloring is **NP-complete**

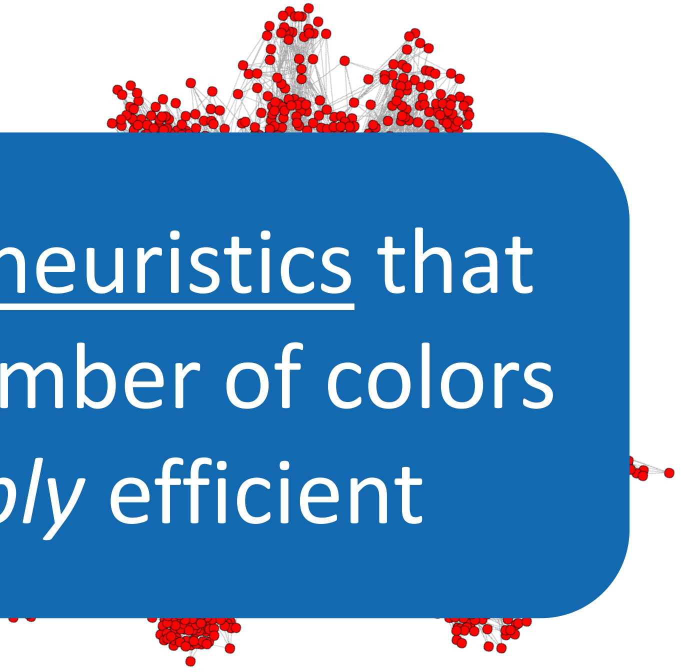
We have massive **parallelism**





[1] Heng Lin et al.: ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds, **SC18, Gordon Bell Finalist**

> 233 TB
271 billion vertices,
12 trillion edges [1]



Thus, one uses <u>parallel</u> <u>heuristics</u> that use a *reasonably* low number of colors while being *reasonably* efficient

Parallel graph coloring heuristics

```
/* n: number of vertices,
   m: number of edges,
   Δ: maximum vertex degree */
```

Parallel graph coloring heuristics

```
/* n: number of vertices,
   m: number of edges,
   Δ: maximum vertex degree */
```

They have a common structure

Parallel graph coloring heuristics

```
/* n: number of vertices,
   m: number of edges,
   Δ: maximum vertex degree */
```

They have a common structure

```
for each vertex $v_i$ in ($v_1$ … $v_n$):
  find smallest color c not used by the neighbors of $v_i$;
  assign c to $v_i$;
```

Parallel graph coloring heuristics

```
/* n: number of vertices,
   m: number of edges,
   Δ: maximum vertex degree */
```

They have a common structure

This immediately ensures using at most Δ+1 colors

```
for each vertex $v_i$ in ($v_1$ … $v_n$):
  find smallest color c not used by the neighbors of $v_i$;
  assign c to $v_i$;
```

Parallel graph coloring heuristics

```
/* n: number of vertices,
   m: number of edges,
   Δ: maximum vertex degree */
```

They have a common structure

This immediately ensures using at most Δ+1 colors

```
for each vertex $v_i$ in $(v_1 … v_n)$:
  find smallest color c not used by the neighbors of $v_i$;
  assign c to $v_i$;
```

Parallel graph coloring heuristics

```
/* n: number of vertices,
   m: number of edges,
   Δ: maximum vertex degree */
```

They have a common structure

This immediately ensures using at most Δ+1 colors

The order of picking vertices impacts coloring quality

```
for each vertex $v_i$ in ($v_1$ … $v_n$):
  find smallest color c not used by the neighbors of $v_i$;
  assign c to $v_i$;
```

Parallel graph coloring heuristics

```
/* n: number of vertices,
   m: number of edges,
   Δ: maximum vertex degree */
```

They have a common structure

This immediately ensures using at most Δ+1 colors

The order of picking vertices impacts coloring quality

This sounds inherently sequential…

```
for each vertex $v_i$ in ($v_1$ … $v_n$):
  find smallest color c not used by the neighbors of $v_i$;
  assign c to $v_i$;
```

Parallel graph coloring heuristics

```
/* n: number of vertices,
   m: number of edges,
   Δ: maximum vertex degree */
```

They have a common structure

This immediately ensures using at most Δ+1 colors

The order of picking vertices impacts coloring quality

This sounds inherently sequential...

```
for each vertex $v_i$ in ($v_1$ … $v_n$):
    find smallest color c not used by the neighbors of $v_i$;
    assign c to $v_i$;
```

...Parallelism is enabled by coloring in parallel <u>groups of vertices</u> that are <u>not</u> adjacent (i.e., form an independent set).

Parallel graph coloring heuristics

```
/* n: number of vertices,
   m: number of edges,
   Δ: maximum vertex degree */
```

They have a common structure

This immediately ensures using at most Δ+1 colors

The order of picking vertices impacts coloring quality

This sounds inherently sequential…

```
for each vertex $v_i$ in ($v_1$ … $v_n$):
    find smallest color c not used by the neighbors of $v_i$;
    assign c to $v_i$;
```

…Parallelism is enabled by coloring in parallel <u>groups of vertices</u> that are <u>not</u> adjacent (i.e., form an independent set).

"Scheduled coloring" – the vertex order determines ("schedules") when vertices are picked for being colored

Parallel graph coloring heuristics

```
/* n: number of vertices,
   m: number of edges,
   Δ: maximum vertex degree */
```

They have a common structure

This immediately ensures using at most Δ+1 colors

The order of picking vertices impacts coloring quality

This sounds inherently sequential…

```
for each vertex $v_i$ in $(v_1 … v_n)$:
    find smallest color c not used by the neighbors of $v_i$;
    assign c to $v_i$;
```

…Parallelism is enabled by coloring in parallel groups of vertices that are not adjacent (i.e., form an independent set).

“Scheduled coloring” – the vertex order determines (“schedules”) when vertices are picked for being colored

Parallel graph coloring heuristics

```
/* n: number of vertices,
   m: number of edges,
   Δ: maximum vertex degree */
```

They h immediately
a comm
structur

Goal 1

Maximize quality
(i.e., minimize
#used colors)

The order of picking vertices impacts coloring quality

This sounds inherently sequential…

```
for v_1 … v_n):
    find smallest color c not used by the neighbors of v_i;
    assign c to v_i;
```

…Parallelism is enabled by coloring in parallel groups of vertices that are not adjacent (i.e., form an independent set).

“Scheduled coloring” – the vertex order determines (“schedules”) when vertices are picked for being colored

Parallel graph coloring heuristics

```
/* n: number of vertices,
   m: number of edges,
   Δ: maximum vertex degree */
```

They h immediately
a comm
structur

Goal 1

Maximize quality (i.e., minimize #used colors)

The order of picking vertices impacts coloring quality

This sounds inherently sequential…

```
for v1 … vn):
    find smallest color c not used by the neighbors of vi;
    assign c to vi;
```

Goal 2

Maximize performance

...Paralle parallel
groups ent (i.e.,

"Scheduled coloring" – the vertex order determines ("schedules") when vertices are picked for being colored

Parallel graph coloring heuristics

```
/* n: number of vertices,
   m: number of edges,
   Δ: maximum vertex degree */
```

They h immediately a comm structur

The order of picking vertices impacts coloring quality

This sounds inherently

```
for v_1 … v_n>:
  find smallest color c not used b
  assign c to v_i;
```

...Paralle groups parallel ent (i.e.,

e (“schedules”) when vertices are picked for being colored

Goal 1

Maximize quality (i.e., minimize #used colors)

Goal 2

Maximize performance

Both empirically and with theoretical properties (i.e., minimize work, depth, and theoretical limit on #used colors)

Parallel graph coloring heuristics

They h… immediately
a comm…
structur…

Goal 1

Maximize quality (i.e., minimize #used colors)

The order of pi…
vertices impa…
coloring quality

work: total number of operations
depth: longest chain of sequential dependencies

This sounds inherently

```
for … v_1 … v_n);
find smallest color c not used b…
assign c to v_i;
```

Both empirically and with theoretical properties (i.e., minimize work, depth, and theoretical limit on #used colors)

Goal 2

Maximize performance

…Paralle…
groups …
…parallel
…ent (i.e.,

…e
…s
(“schedules”) when vertices are picked for being colored

Parallel graph coloring heuristics

```
/* n: number of vertices,
   m: number of edges,
   Δ: maximum vertex degree,
   d: graph’s degeneracy */
```

Parallel graph coloring heuristics

A lot of heuristics were introduced, offering different work-depth-quality **tradeoffs**

```
/* n: number of vertices,
   m: number of edges,
   Δ: maximum vertex degree,
   d: graph's degeneracy */
```

Parallel graph coloring heuristics

```
/* n: number of vertices,
   m: number of edges,
   Δ: maximum vertex degree,
   d: graph’s degeneracy */
```

A lot of heuristics were introduced, offering different work-depth-quality **tradeoffs**

Ordering

Parallel graph coloring heuristics

```
/* n: number of vertices,
   m: number of edges,
   Δ: maximum vertex degree,
   d: graph’s degeneracy */
```

A lot of heuristics were introduced, offering different work-depth-quality **tradeoffs**

Ordering

“First fit” (i.e., any order)

“Largest degree first”

“Smallest degree last”

Random

Random

“Largest log-degree first”

“Smallest log-degree last”

Parallel graph coloring heuristics

A lot of heuristics were introduced, offering different work-depth-quality **tradeoffs**

```
/* n: number of vertices,
   m: number of edges,
   Δ: maximum vertex degree,
   d: graph’s degeneracy */
```

The associated coloring heuristics:

Ordering	Depth	Work	Quality
“First fit” (i.e., any order)			
“Largest degree first”			
“Smallest degree last”			
Random			
Random			
“Largest log-degree first”			
“Smallest log-degree last”			

Parallel graph coloring heuristics

```
/* n: number of vertices,
   m: number of edges,
   Δ: maximum vertex degree,
   d: graph’s degeneracy */
```

A lot of heuristics were introduced, offering different work-depth-quality **tradeoffs**

The associated coloring heuristics:

Ordering	Depth	Work	Quality
“First fit” (i.e., any order)	No general bounds; $\Omega(n)$ for some graphs	$O(n+m)$	$\Delta+1$
“Largest degree first”	No general bounds; $\Omega(\Delta^2)$ for some graphs	$O(n+m)$	$\Delta+1$
“Smallest degree last”	No general bounds; $\Omega(n)$ for some graphs	$O(n+m)$	$d+1$
Random	$\mathbb{E}\ O\left(\frac{\log n}{\log\log n}\right)$	$O(n+m)$	$\Delta+1$
Random	$\mathbb{E}\ O\left(\log n + \log\Delta\cdot\min\left\{\sqrt{m}, \Delta + \frac{\log\Delta\log n}{\log\log n}\right\}\right)$	$O(n+m)$	$\Delta+1$
“Largest log-degree first”	$\mathbb{E}\ O\left(\log n + \log\Delta\cdot\left(\min\{\Delta, \sqrt{m}\} + \frac{\log^2\Delta\log n}{\log\log n}\right)\right)$	$O(n+m)$	$\Delta+1$
“Smallest log-degree last”	$\mathbb{E}\ O\left(\log\Delta\log n + \log\Delta\cdot\left(\min\{\Delta, \sqrt{m}\} + \frac{\log^2\Delta\log n}{\log\log n}\right)\right)$	$O(n+m)$	$\Delta+1$

Parallel graph coloring heuristics

A lot of heuristics were introduced, offering different work-depth-quality **tradeoffs**

```
/* n: number of vertices,
   m: number of edges,
   Δ: maximum vertex degree,
   d: graph's degeneracy */
```

The associated coloring heuristics:

Ordering	Depth	Work	Quality
"First fit" (i.e., any order)	No general bounds; $\Omega(n)$ for some graphs	$O(n+m)$	$\Delta+1$
"Largest degree first"	No general bounds; $\Omega(\Delta^2)$ for some graphs	$O(n+m)$	$\Delta+1$
"Smallest degree last"	No general bounds; $\Omega(n)$ for some graphs	$O(n+m)$	$d+1$
Random	$\mathbb{E}\, O\left(\frac{\log n}{\log\log n}\right)$	$O(n+m)$	$\Delta+1$
Random	$\mathbb{E}\, O\left(\log n + \log\Delta \cdot \min\left\{\sqrt{m}, \Delta + \frac{\log\Delta \log n}{\log\log n}\right\}\right)$	$O(n+m)$	$\Delta+1$
"Largest log-degree first"	$\mathbb{E}\, O\left(\log n + \log\Delta \left(\min\{\Delta, \sqrt{m}\} + \frac{\log^2\Delta \log n}{\log\log n}\right)\right)$	$O(n+m)$	$\Delta+1$
"Smallest log-degree last"	$\mathbb{E}\, O\left(\log\Delta \log n + \log\Delta \cdot \left(\min\{\Delta, \sqrt{m}\} + \frac{\log^2\Delta \log n}{\log\log n}\right)\right)$	$O(n+m)$	$\Delta+1$

No need for going over these details (for now ☺)

Parallel graph coloring heuristics

```
/* n: number of vertices,
   m: number of edges,
   Δ: maximum vertex degree,
   d: graph's degeneracy */
```

A lot of heuristics were introduced, offering different work-depth-quality **tradeoffs**

The associated coloring heuristics:

Ordering	**Depth**	**Work**	**Quality**
“First fit” (i.e., any order)	No general bounds; $\Omega(n)$ for some graphs	$O(n+m)$	$\Delta + 1$
“Largest degree first”	No general bounds; $\Omega(\Delta^2)$ for some graphs	$O(n+m)$	$\Delta + 1$
“Smallest degree last”	No general bounds; $\Omega(n)$ for some graphs	$O(n+m)$	$d + 1$
Random	$\mathbb{E}\, O\left(\frac{\log n}{\log\log n}\right)$	$O(n+m)$	$\Delta + 1$
Random	$\mathbb{E}\, O\left(\log n + \log\Delta \cdot \min\left\{\sqrt{m}, \Delta + \frac{\log\Delta\log n}{\log\log n}\right\}\right)$	$O(n+m)$	$\Delta + 1$
“Largest log-degree first”	$\mathbb{E}\, O\left(\log n + \log\Delta \cdot \left(\min\{\Delta, \sqrt{m}\} + \frac{\log^2\Delta\log n}{\log\log n}\right)\right)$	$O(n+m)$	$\Delta + 1$
“Smallest log-degree last”	$\mathbb{E}\, O\left(\log\Delta\log n + \log\Delta \cdot \left(\min\{\Delta, \sqrt{m}\} + \frac{\log^2\Delta\log n}{\log\log n}\right)\right)$	$O(n+m)$	$\Delta + 1$

No need for going over these details (for now ☺)

Parallel graph coloring heuristics

A lot of heuristics were introduced, offering different work-depth-quality **tradeoffs**

```
/* n: number of vertices,
   m: number of edges,
   Δ: maximum vertex degree,
   d: graph’s degeneracy */
```

The associated coloring heuristics:

Ordering	Depth	Work	Quality
“First fit” (i.e., any order)	No general bounds; $\Omega(n)$ for some graphs	$O(n+m)$	$\Delta+1$
“Largest degree first”	No general bounds; $\Omega(\Delta^2)$ for some graphs	$O(n+m)$	$\Delta+1$
“Smallest degree last”	No general bounds; $\Omega(n)$ for some graphs	$O(n+m)$	$d+1$
Random	$\mathbb{E}\, O\left(\frac{\log n}{\log\log n}\right)$	$O(n+m)$	$\Delta+1$
Random	$\mathbb{E}\, O\left(\log n + \log\Delta \cdot \min\left\{\sqrt{m}, \Delta + \frac{\log\Delta \log n}{\log\log n}\right\}\right)$	$O(n+m)$	$\Delta+1$
“Largest log-degree first”	$\mathbb{E}\, O\left(\log n + \log\Delta\left(\min\{\Delta,\sqrt{m}\} + \frac{\log^2\Delta \log n}{\log\log n}\right)\right)$	$O(n+m)$	$\Delta+1$
“Smallest log-degree last”	$\mathbb{E}\, O\left(\log\Delta \log n + \log\Delta \cdot \left(\min\{\Delta,\sqrt{m}\} + \frac{\log^2\Delta \log n}{\log\log n}\right)\right)$	$O(n+m)$	$\Delta+1$

No need for going over these details (for now ☺)

Almost all schemes have only trivial quality bounds

Parallel graph coloring heuristics

```
/* n: number of vertices,
   m: number of edges,
   Δ: maximum vertex degree,
   d: graph’s degeneracy */
```

A lot of heuristics were introduced, offering different work-depth-quality **tradeoffs**

The only scheme with good quality bounds offers no depth bounds

ciated coloring heuristics:

Ordering	Depth	Work	Quality
“First fit” (i.e., any order)	No general bounds; $\Omega(n)$ for some graphs	$O(n+m)$	$\Delta+1$
“Largest degree first”	No general bounds; $\Omega(\Delta^2)$ for some graphs	$O(n+m)$	$\Delta+1$
“Smallest degree last”	No general bounds; $\Omega(n)$ for some graphs	$O(n+m)$	$d+1$
Random	$\mathbb{E}\, O\left(\frac{\log n}{\log\log n}\right)$	$O(n+m)$	$\Delta+1$
Random	$\mathbb{E}\, O\left(\log n + \log\Delta \cdot \min\left\{\sqrt{m}, \Delta + \frac{\log\Delta \log n}{\log\log n}\right\}\right)$	$O(n+m)$	$\Delta+1$
“Largest log-degree first”	$\mathbb{E}\, O\left(\log n + \log\Delta \cdot \left(\min\{\Delta, \sqrt{m}\} + \frac{\log^2\Delta \log n}{\log\log n}\right)\right)$	$O(n+m)$	$\Delta+1$
“Smallest log-degree last”	$\mathbb{E}\, O\left(\log\Delta \log n + \log\Delta \cdot \left(\min\{\Delta, \sqrt{m}\} + \frac{\log^2\Delta \log n}{\log\log n}\right)\right)$	$O(n+m)$	$\Delta+1$

No need for going over these details (for now ☺)

Almost all schemes have only trivial quality bounds

Parallel graph coloring heuristics

```
/* n: number of vertices,
   m: number of edges,
   Δ: maximum vertex degree,
   d: graph’s degeneracy */
```

A lot of heuristics were introduced, offering different work-depth-quality **tradeoffs**

The only scheme with good quality bounds offers no depth bounds

ciated coloring heuristics:

Let’s use it as a starting point…

Ordering		Work	Quality
“First fit” (i.e., any order)	No general bounds; $\Omega($	$O(n+m)$	$\Delta + 1$
“Largest degree first”	No general bounds; $\Omega(\Delta^2)$ for some graphs	$O(n+m)$	$\Delta + 1$
“Smallest degree last”	No general bounds; $\Omega(n)$ for some graphs	$O(n+m)$	$d + 1$
Random	$\mathbb{E}\, O\left(\frac{\log n}{\log\log n}\right)$	$O(n+m)$	$\Delta + 1$
Random	$\mathbb{E}\, O\left(\log n + \log\Delta \cdot \min\left\{\sqrt{m}, \Delta + \frac{\log\Delta\log n}{\log\log n}\right\}\right)$	$O(n+m)$	$\Delta + 1$
“Largest log-degree first”	$\mathbb{E}\, O\left(\log n + \log\Delta \cdot \left(\min\{\Delta, \sqrt{m}\} + \frac{\log^2\Delta\log n}{\log\log n}\right)\right)$	$O(n+m)$	$\Delta + 1$
“Smallest log-degree last”	$\mathbb{E}\, O\left(\log\Delta\log n + \log\Delta \cdot \left(\min\{\Delta, \sqrt{m}\} + \frac{\log^2\Delta\log n}{\log\log n}\right)\right)$	$O(n+m)$	$\Delta + 1$

No need for going over these details (for now ☺)

Almost all schemes have only trivial quality bounds

"Smallest degree last": fundamentals

“Smallest degree last”: fundamentals

→ Iterate over vertices in the **degeneracy ordering**

“Smallest degree last”: fundamentals

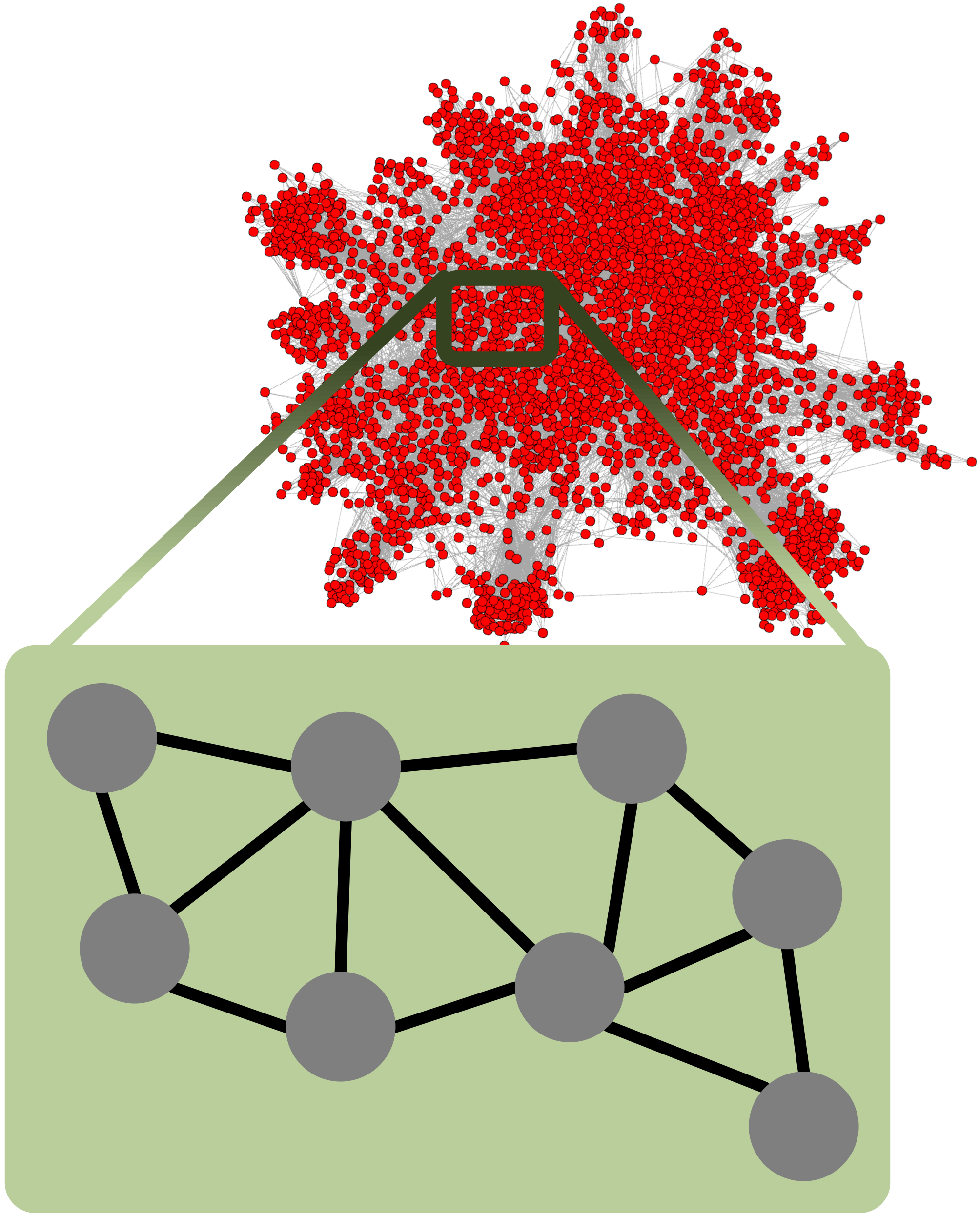
→ Iterate over vertices in the **degeneracy ordering**

→ A graph G is ***s*-degenerate** if, in each of its induced subgraphs, there is a vertex with a degree of at most *s*

“Smallest degree last”: fundamentals

→ Iterate over vertices in the **degeneracy ordering**

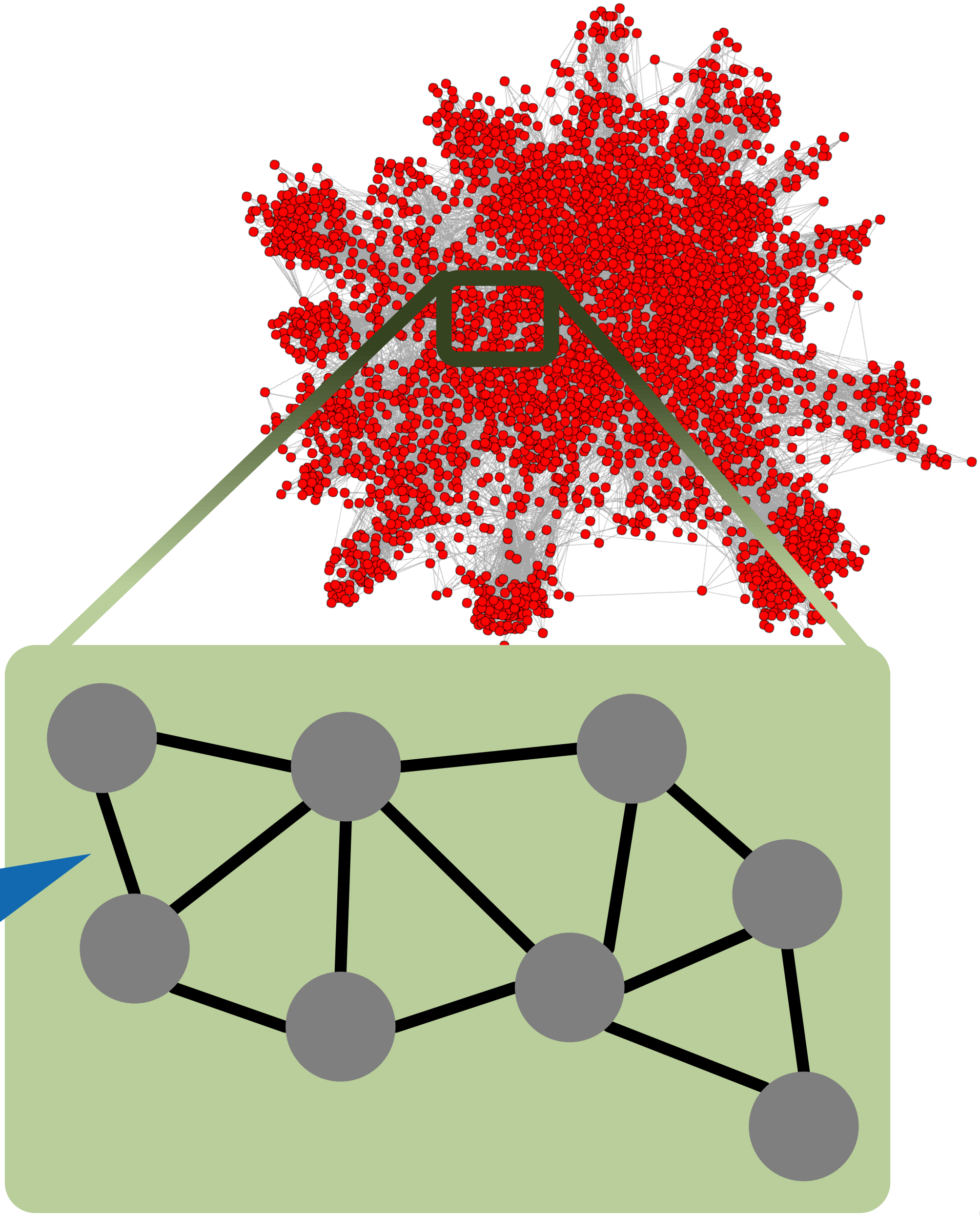
→ A graph G is ***s*-degenerate** if, in each of its induced subgraphs, there is a vertex with a degree of at most *s*



“Smallest degree last”: fundamentals

→ Iterate over vertices in the **degeneracy ordering**

→ A graph G is ***s*-degenerate** if, in each of its induced subgraphs, there is a vertex with a degree of at most *s*

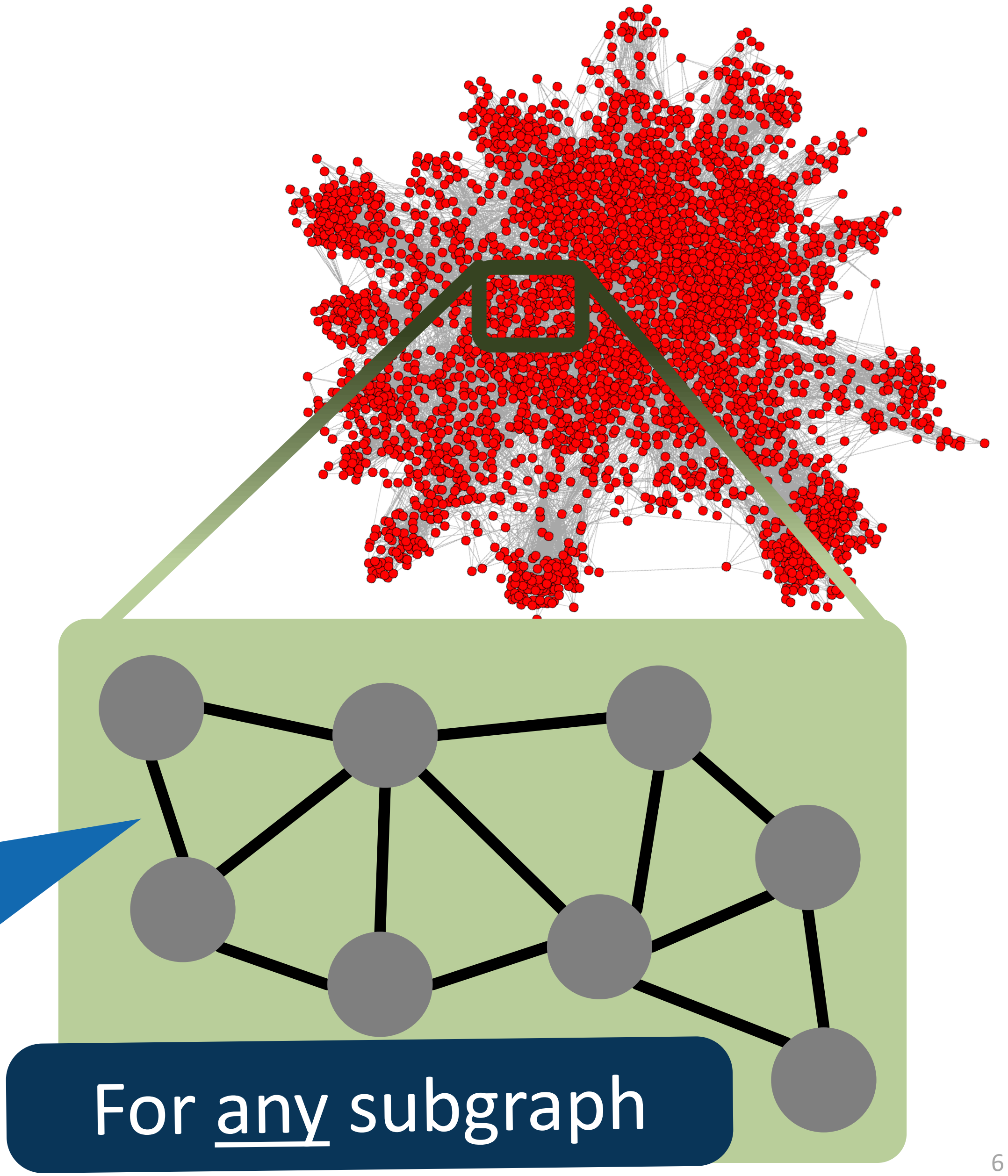


At least one vertex will have degree at most *s*

"Smallest degree last": fundamentals

→ Iterate over vertices in the **degeneracy ordering**

→ A graph G is ***s*-degenerate** if, in each of its induced subgraphs, there is a vertex with a degree of at most *s*



At least one vertex will have degree at most *s*

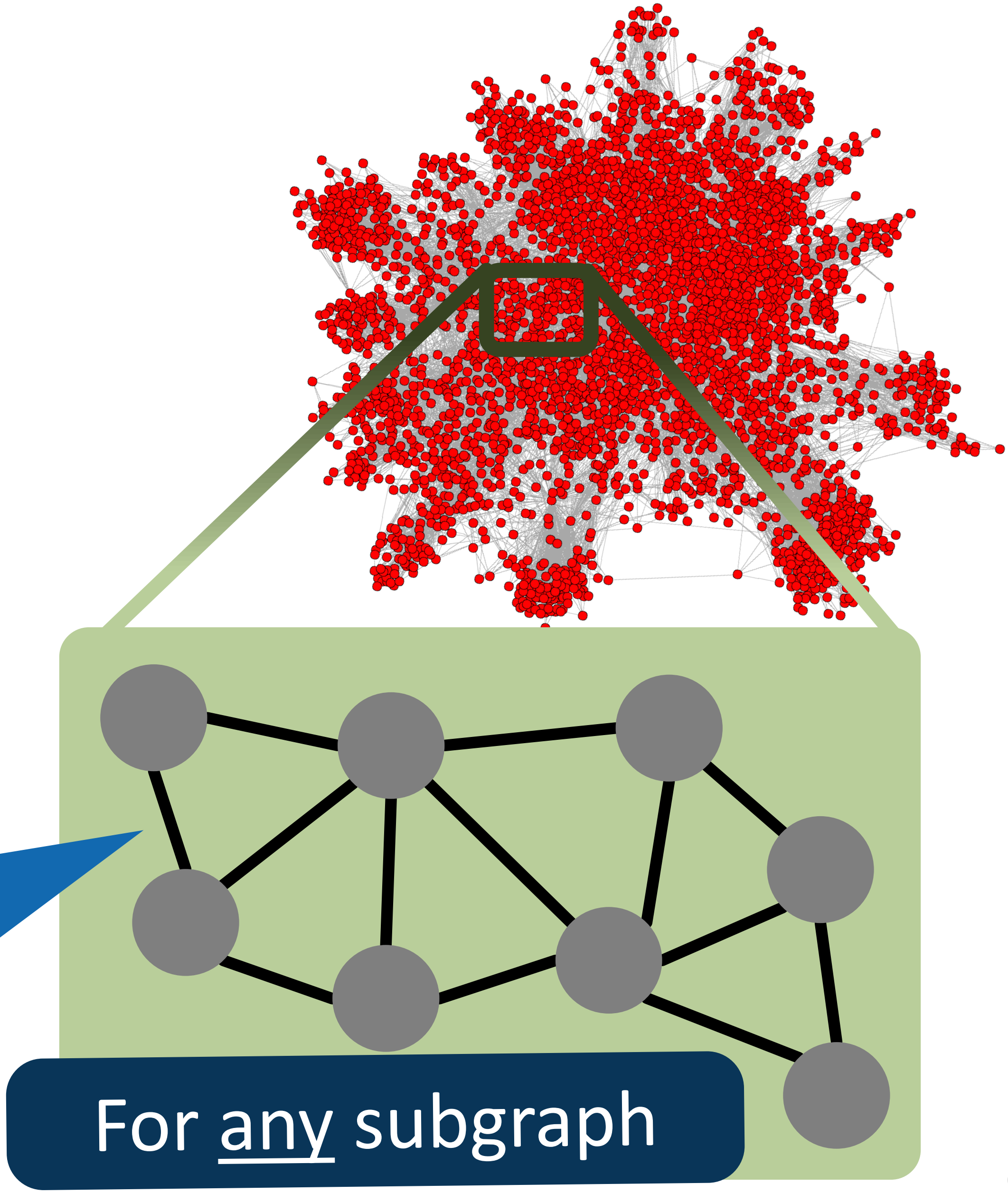
For any subgraph

“Smallest degree last”: fundamentals

→ Iterate over vertices in the **degeneracy ordering**

→ A graph G is ***s*-degenerate** if, in each of its induced subgraphs, there is a vertex with a degree of at most *s*

→ The **degeneracy *d*** of a graph G is the smallest *s*, such that G is still *s*-degenerate



At least one vertex will have degree at most *s*

For any subgraph

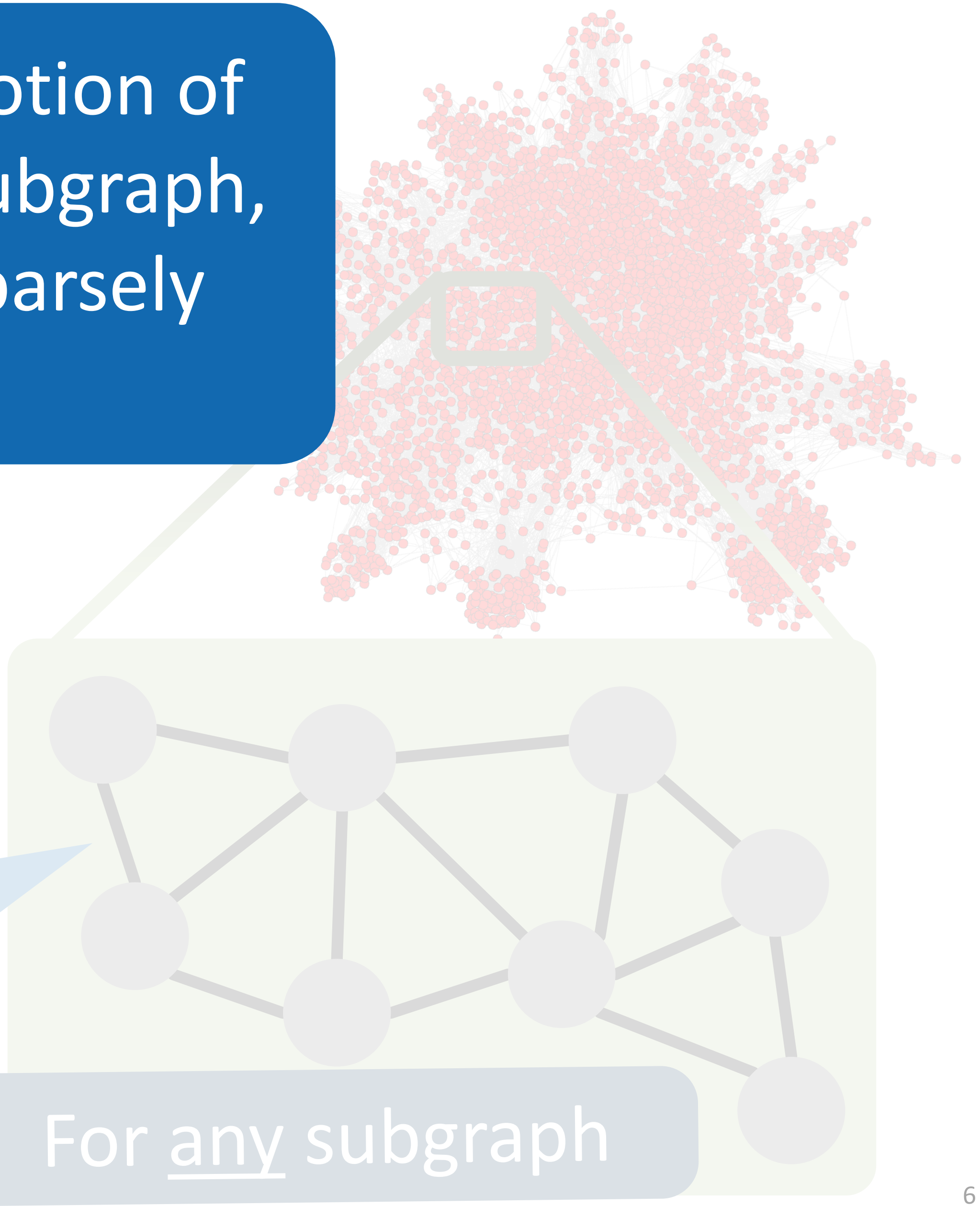
“Smallest degree last”: fundamentals

Intuitively, degeneracy captures the notion of graph sparsity „at any level”: in each subgraph, we will always find a low-degree (=sparsely connected) vertex

degree of at most s

→ The **degeneracy *d*** of a graph G is the smallest *s*, such that G is still *s*-degenerate

At least one vertex will have degree at most *s*



For any subgraph

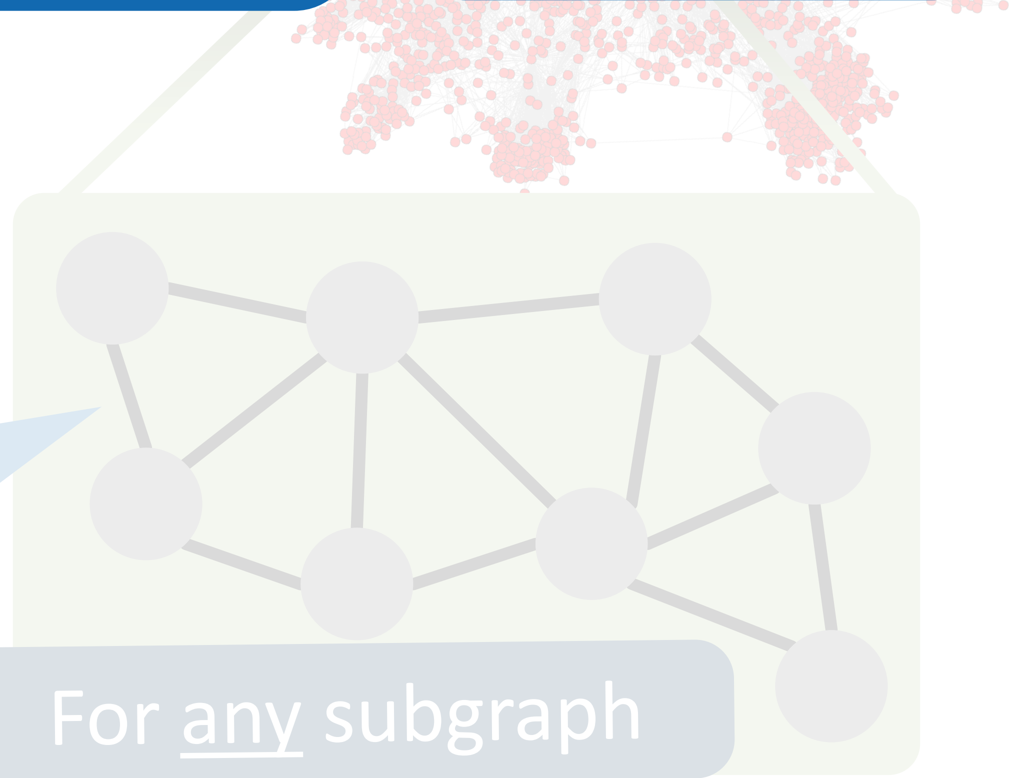
“Smallest degree last”: fundamentals

Intuitively, degeneracy captures the notion of graph sparsity „at any level”: in each subgraph, we will always find a low-degree (=sparsely connected) vertex

The lower the degeneracy is, the sparser graph is

degree of at most *s*

→ The **degeneracy *d*** of a graph G is the smallest *s*, such that G is still *s*-degenerate



At least one vertex will have degree at most *s*

For any subgraph

“Smallest degree last”: fundamentals

Intuitively, degeneracy captures the notion of graph sparsity „at any level”: in each subgraph, we will always find a low-degree (=sparsely connected) vertex

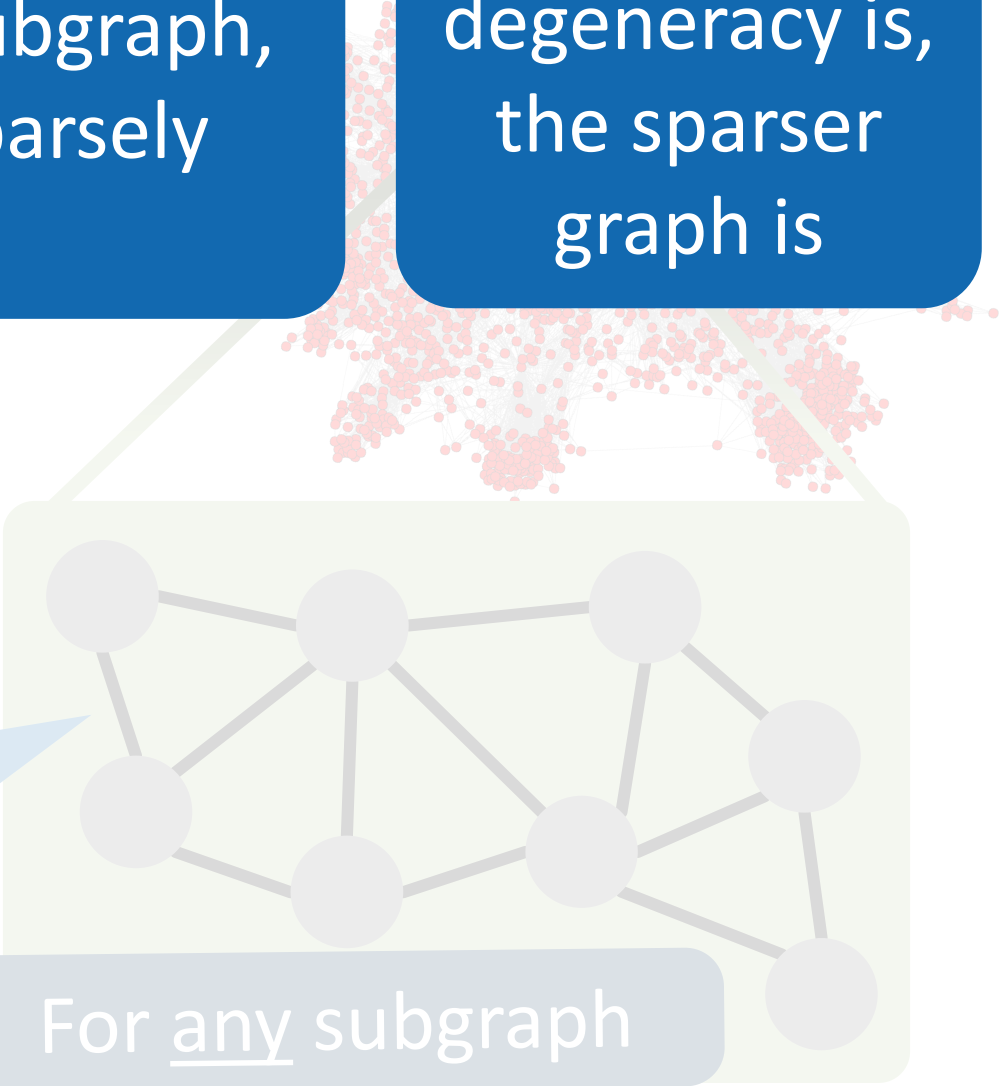
The lower the degeneracy is, the sparser graph is

degree of at most *s*

allest *s*,

Now, the coloring heuristics that uses the degeneracy order gives provable d+1 coloring quality

ast one vertex will have degree at most *s*



For any subgraph

"Smallest degree last": fundamentals

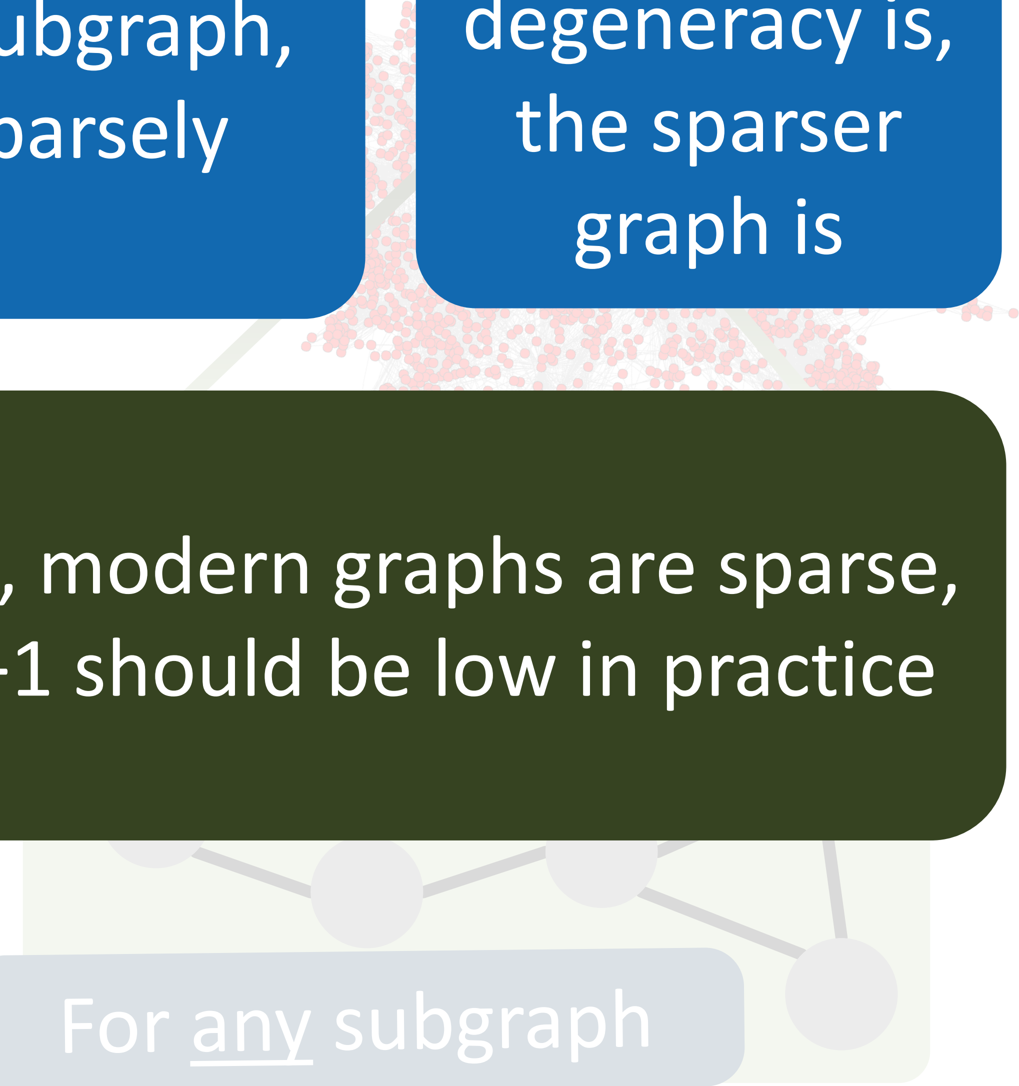
Intuitively, degeneracy captures the notion of graph sparsity „at any level": in each subgraph, we will always find a low-degree (=sparsely connected) vertex

The lower the degeneracy is, the sparser graph is

Now, the coloring heuristics that uses the degeneracy order gives provable d+1 coloring quality

Great, modern graphs are sparse, so d+1 should be low in practice

vertex will have degree at most *s*



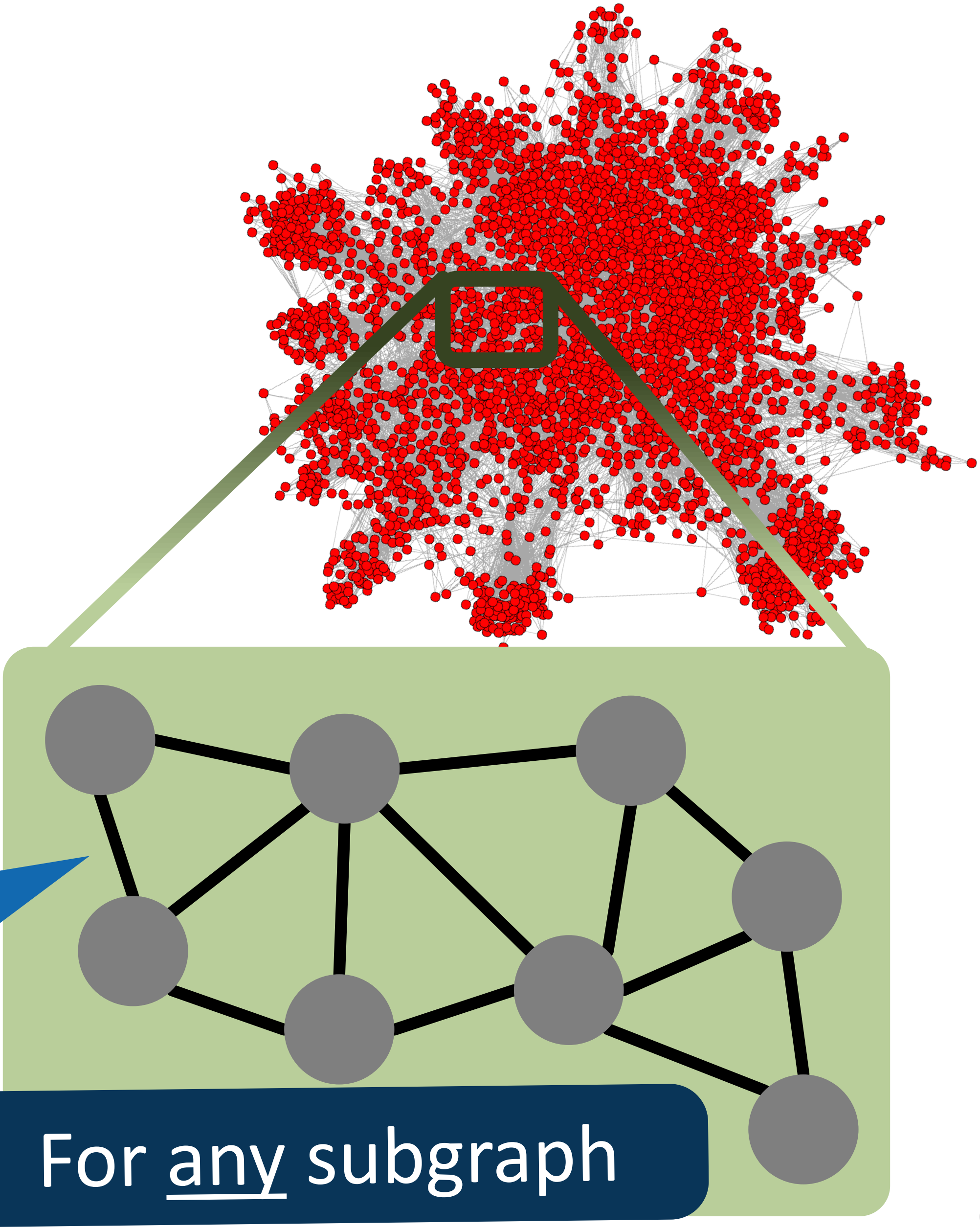
For any subgraph

"Smallest degree last": fundamentals

→ Iterate over vertices in the **degeneracy ordering**

→ A graph G is ***s*-degenerate** if, in each of its induced subgraphs, there is a vertex with a degree of at most *s*

→ The **degeneracy *d*** of a graph G is the smallest *s*, such that G is still *s*-degenerate



At least one vertex will have degree at most *s*

For any subgraph

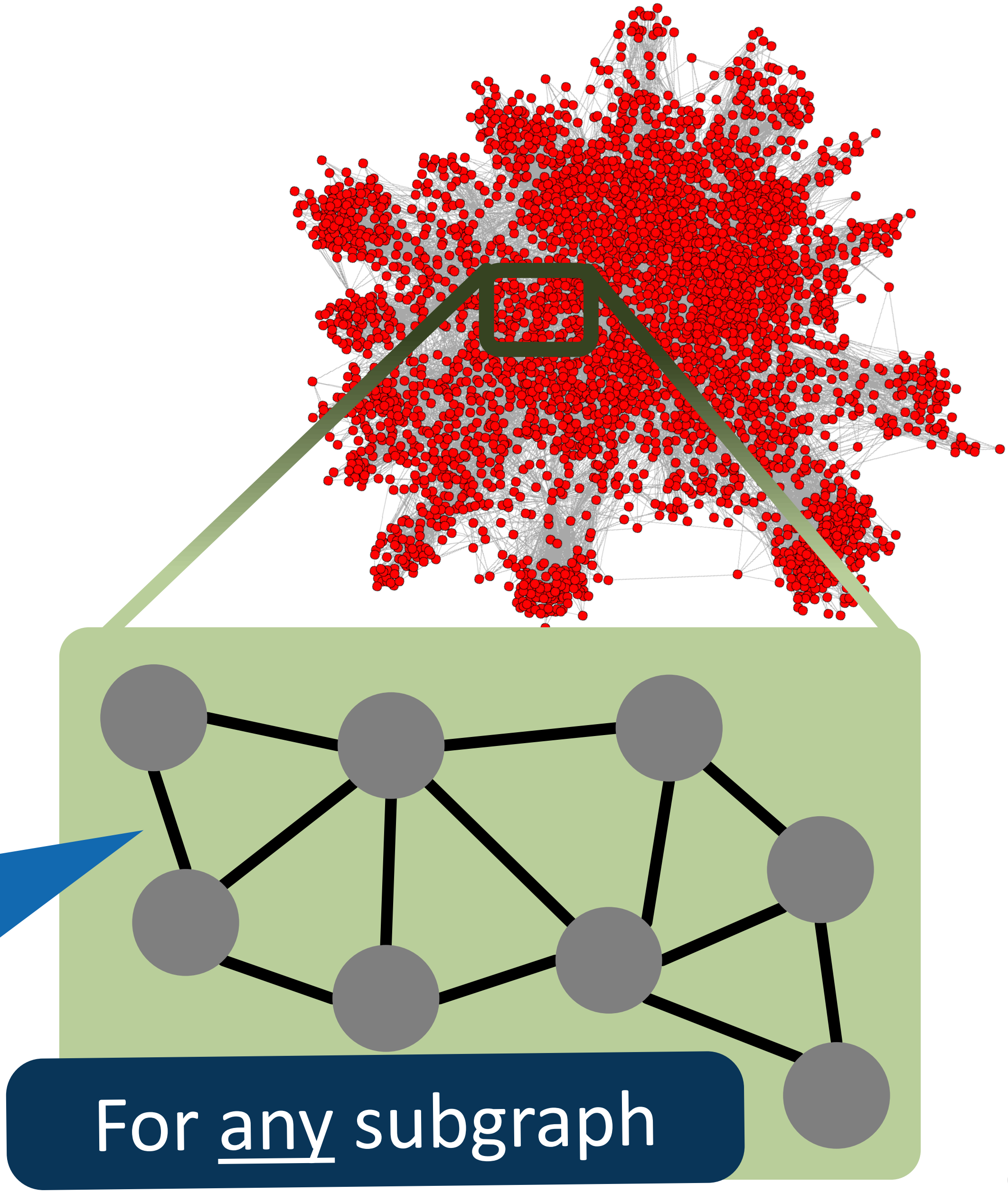
"Smallest degree last": fundamentals

→ Iterate over vertices in the **degeneracy ordering**

→ A graph G is ***s*-degenerate** if, in each of its induced subgraphs, there is a vertex with a degree of at most *s*

→ The **degeneracy *d*** of a graph G is the smallest *s*, such that G is still *s*-degenerate

→ The degeneracy ordering of a given graph is an ordering, where each vertex *v* has at most *d* neighbors that are ordered higher than *v*



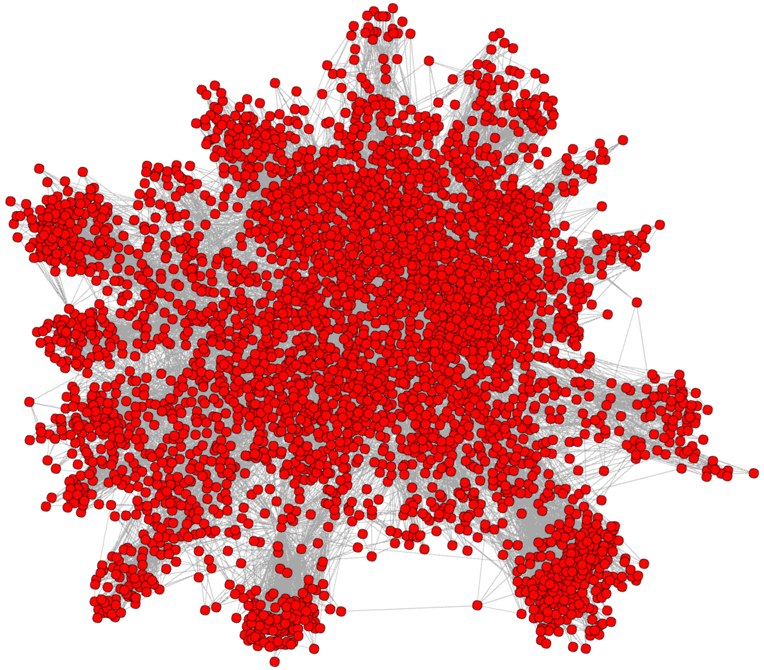
At least one vertex will have degree at most *s*

For any subgraph

Degeneracy ordering: example

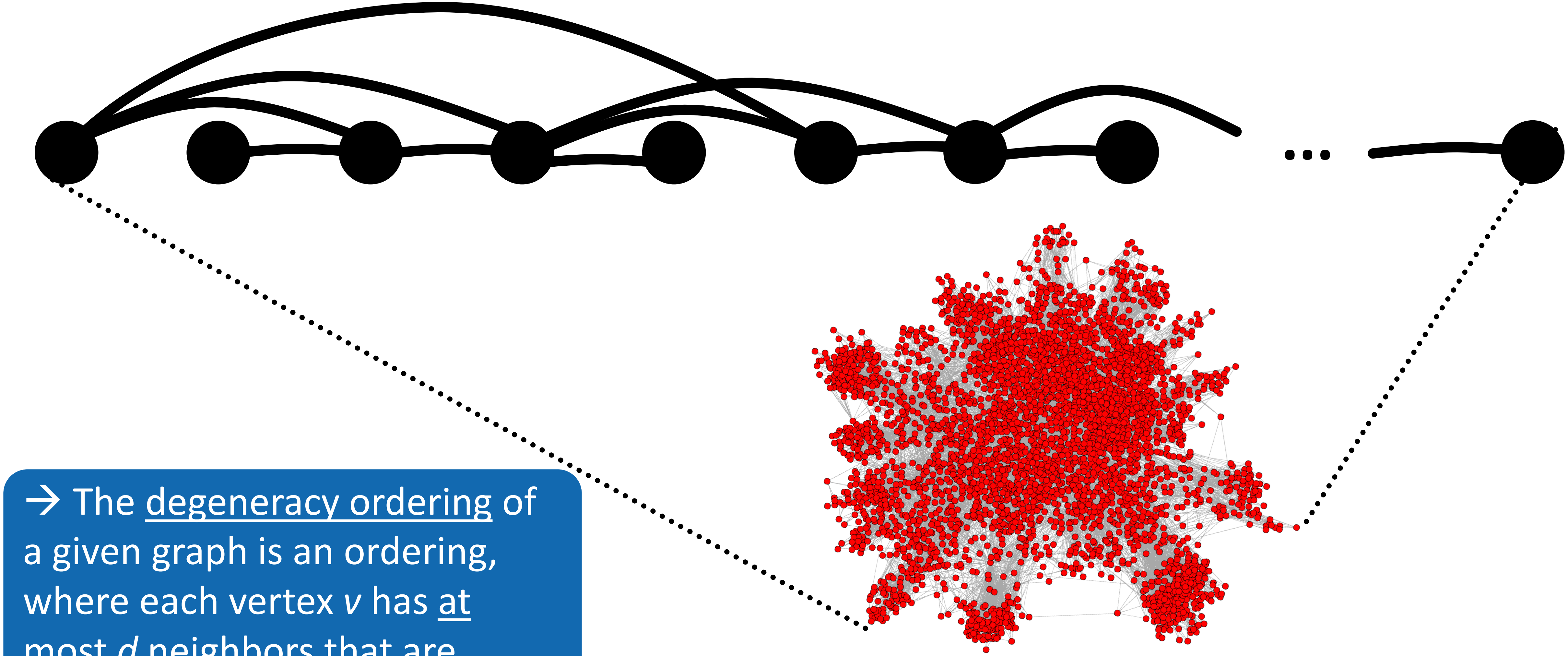
→ The <u>degeneracy ordering</u> of a given graph is an ordering, where each vertex *v* has <u>at most</u> *d* neighbors that are ordered higher than *v*

Degeneracy ordering: example



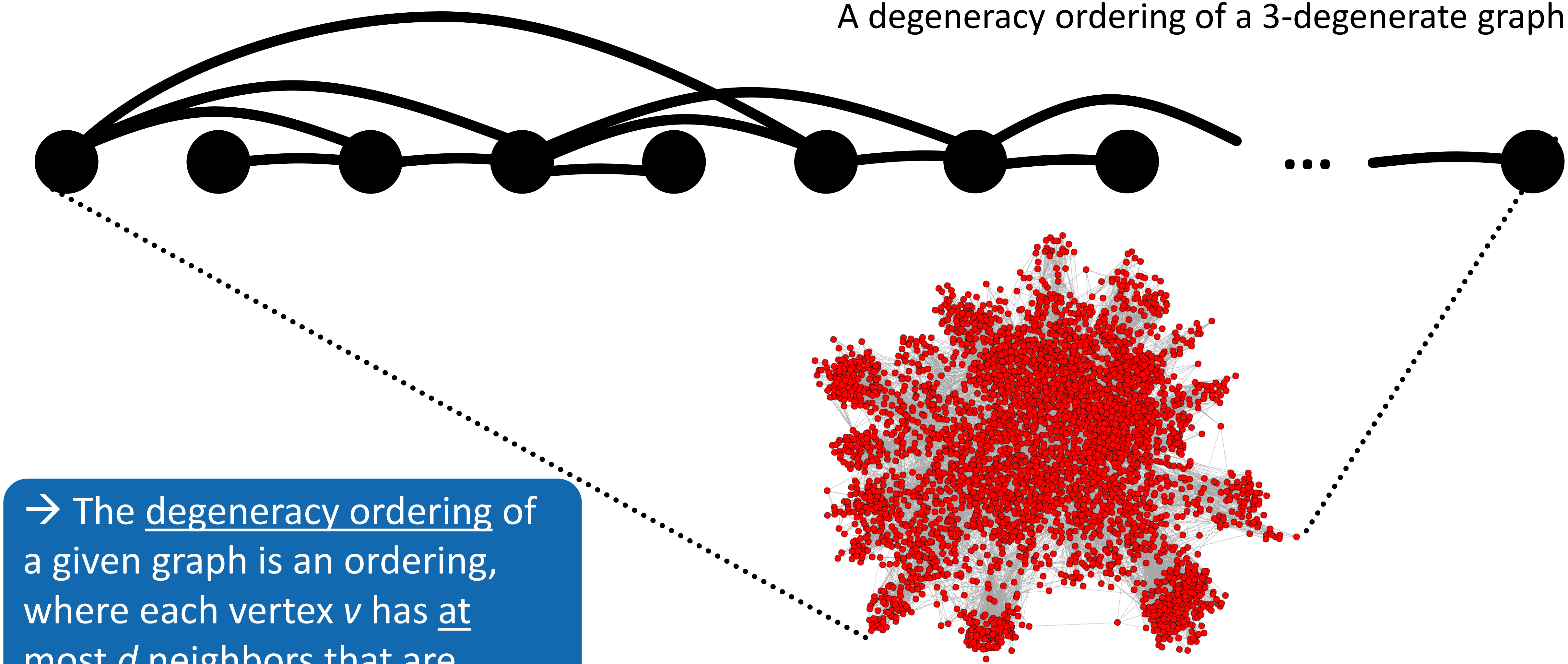
→ The degeneracy ordering of a given graph is an ordering, where each vertex *v* has at most *d* neighbors that are ordered higher than *v*

Degeneracy ordering: example



→ The <u>degeneracy ordering</u> of a given graph is an ordering, where each vertex *v* has <u>at most</u> *d* neighbors that are ordered higher than *v*

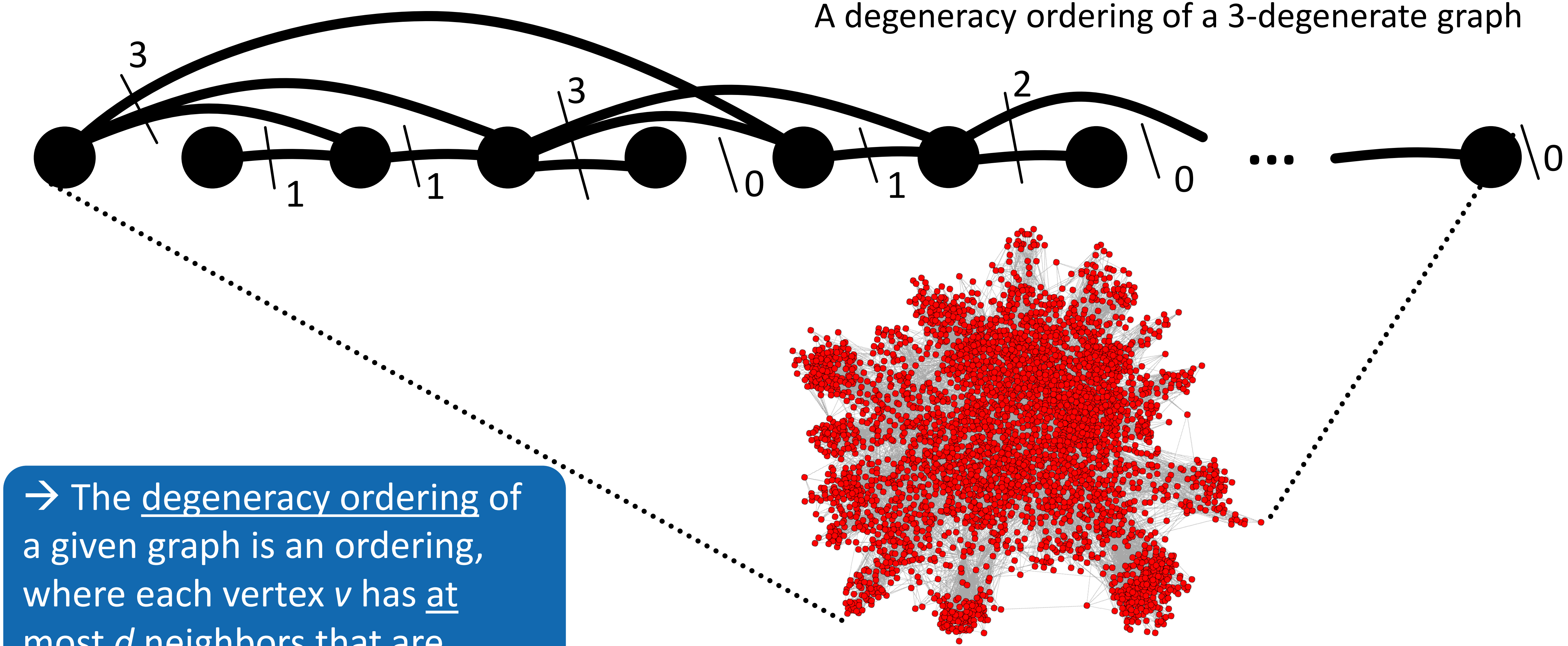
Degeneracy ordering: example



A degeneracy ordering of a 3-degenerate graph

→ The degeneracy ordering of a given graph is an ordering, where each vertex *v* has at most *d* neighbors that are ordered higher than *v*

Degeneracy ordering: example


A degeneracy ordering of a 3-degenerate graph

→ The degeneracy ordering of a given graph is an ordering, where each vertex *v* has at most *d* neighbors that are ordered higher than *v*

Degeneracy ordering: derivation

```
/* V: set of all vertices,
   d(v): degree of a vertex v */
```

Degeneracy ordering: derivation

```
/* V: set of all vertices,
   d(v): degree of a vertex v */
```

How to derive the degeneracy ordering?

Simple: Sequentially remove vertices of smallest degree, one by one.

Degeneracy ordering: derivation

```
/* V: set of all vertices,
   d(v): degree of a vertex v */
```

How to derive the degeneracy ordering?

Strict degeneracy order:

Simple: Sequentially remove vertices of smallest degree, one by one.

```
itr = 0;
while V ≠ Ø:
   $v_{min}$ = argmin $_{v \text{ in } V}$ d(v);
   V = V \ {$v_{min}$};
   rank[$v_{min}$] = itr++;
```

Degeneracy ordering: derivation

```
/* V: set of all vertices,
   d(v): degree of a vertex v */
```

How to derive the degeneracy ordering?

Strict degeneracy order:

Simple: Sequentially remove vertices of smallest degree, one by one.

```
itr = 0;
while V ≠ ∅:
   $v_{min}$ = argmin $_{v \text{ in } V}$ d(v);
   V = V \ {$v_{min}$};
   rank[$v_{min}$] = itr++;
```

Deriving the ordering takes O(n) depth (i.e., it is inherently sequential)

Degeneracy ordering: derivation

```
/* V: set of all vertices,
   d(v): degree of a vertex v */
```

How to derive the degeneracy ordering?

Strict degeneracy order:

Simple: Sequentially remove vertices of smallest degree, one by one.

```
itr = 0;
while $V \neq \emptyset$:
    $v_{min}$ = argmin $_{v \text{ in } V}$ d(v);
    V = V \ {$v_{min}$};
    rank[$v_{min}$] = itr++;
```

✗ Deriving the ordering takes O(n) depth (i.e., it is inherently sequential)

✗ The corresponding coloring heuristics is thus bottlenecked by the ordering derivation

Approximate degeneracy ordering

```
/* V: set of all vertices,
   d(v): degree of a vertex v,
   $d_{avg}$: average degree in V */
```

Strict degeneracy order:

```
itr = 0;
while V ≠ Ø:
   $v_{min}$ = argmin $_{v \text{ in } V}$ d(v);
   V = V \ {$v_{min}$};
   rank[$v_{min}$] = itr++;
```

Approximate degeneracy ordering

```
/* V: set of all vertices,
   d(v): degree of a vertex v,
   $d_{avg}$: average degree in V */
```

→ **Key idea**: try a relaxation of the strict degeneracy order, at the cost of (some) accuracy loss.

Strict degeneracy order:

```
itr = 0;
while V ≠ Ø:
   $v_{min}$ = argmin $_{v \text{ in } V}$ d(v);
   V = V \ {$v_{min}$};
   rank[$v_{min}$] = itr++;
```

Approximate degeneracy ordering

```
/* V: set of all vertices,
   d(v): degree of a vertex v,
   $d_{avg}$: average degree in V */
```

→ **Key idea**: try a relaxation of the strict degeneracy order, at the cost of (some) accuracy loss.

Strict degeneracy order:

```
itr = 0;
while V ≠ Ø:
   $v_{min}$ = argmin $_{v \text{ in } V}$ d(v);
   V = V \ {$v_{min}$};
   rank[$v_{min}$] = itr++;
```

ADG: approximate degeneracy order:

```
itr = 0;
while V ≠ Ø:
   $R_{min}$ = {v | d(v) ≤ (1+ε)$d_{avg}$};
   V = V \ $R_{min}$;
   forall v in $R_{min}$ in parallel:
      rank[v] = itr;
   ++itr;
```

Approximate degeneracy ordering

```
/* V: set of all vertices,
   d(v): degree of a vertex v,
   $d_{avg}$: average degree in V */
```

→ **Key idea**: try a relaxation of the strict degeneracy order, at the cost of (some) accuracy loss.

Strict degeneracy order:

```
itr = 0;
while $V \neq \emptyset$:
   $v_{min}$ = argmin $_{v \text{ in } V}$ d(v);
   V = V \ {$v_{min}$};
   rank[$v_{min}$] = itr++;
```

ADG: approximate degeneracy order:

```
itr = 0;
while $V \neq \emptyset$:
   $R_{min}$ = {v | d(v) $\leq$ (1+$\varepsilon$)$d_{avg}$};
   V = V \ $R_{min}$;
   forall v in $R_{min}$ in parallel:
      rank[v] = itr;
   ++itr;
```

A user-specified parameter that controls a performance-quality tradeoff

Approximate degeneracy ordering

→ **Key idea**: try a relaxation of the strict degeneracy order, at the cost of (some) accuracy loss.

```
/* n: the number of all vertices
   V: set of all vertices,
   d(v): degree of a vertex v,
   $d_{avg}$: average degree in V */
```

```
itr = 0;
while V ≠ ∅:
    $R_{min}$ = {v | d(v) ≤ (1+ε)$d_{avg}$};
    V = V \ $R_{min}$;
    forall v in $R_{min}$ in parallel:
        rank[v] = itr;
    ++itr;
```

Approximate degeneracy ordering

→ **Key idea**: try a relaxation of the strict degeneracy order, at the cost of (some) accuracy loss.

```
/* n: the number of all vertices
   V: set of all vertices,
   d(v): degree of a vertex v,
   $d_{avg}$: average degree in V */
```

Constructing R_{min} takes O(log n) depth

```
itr = 0;
while V ≠ Ø:
   $R_{min}$ = {v | d(v) ≤ (1+ε)$d_{avg}$};
   V = V \ $R_{min}$;
   forall v in $R_{min}$ in parallel:
      rank[v] = itr;
   ++itr;
```

Approximate degeneracy ordering

```
/* n: the number of all vertices
   V: set of all vertices,
   d(v): degree of a vertex v,
   $d_{avg}$: average degree in V */
```

→ **Key idea**: try a relaxation of the strict degeneracy order, at the cost of (some) accuracy loss.

Constructing R_{min} takes O(log n) depth

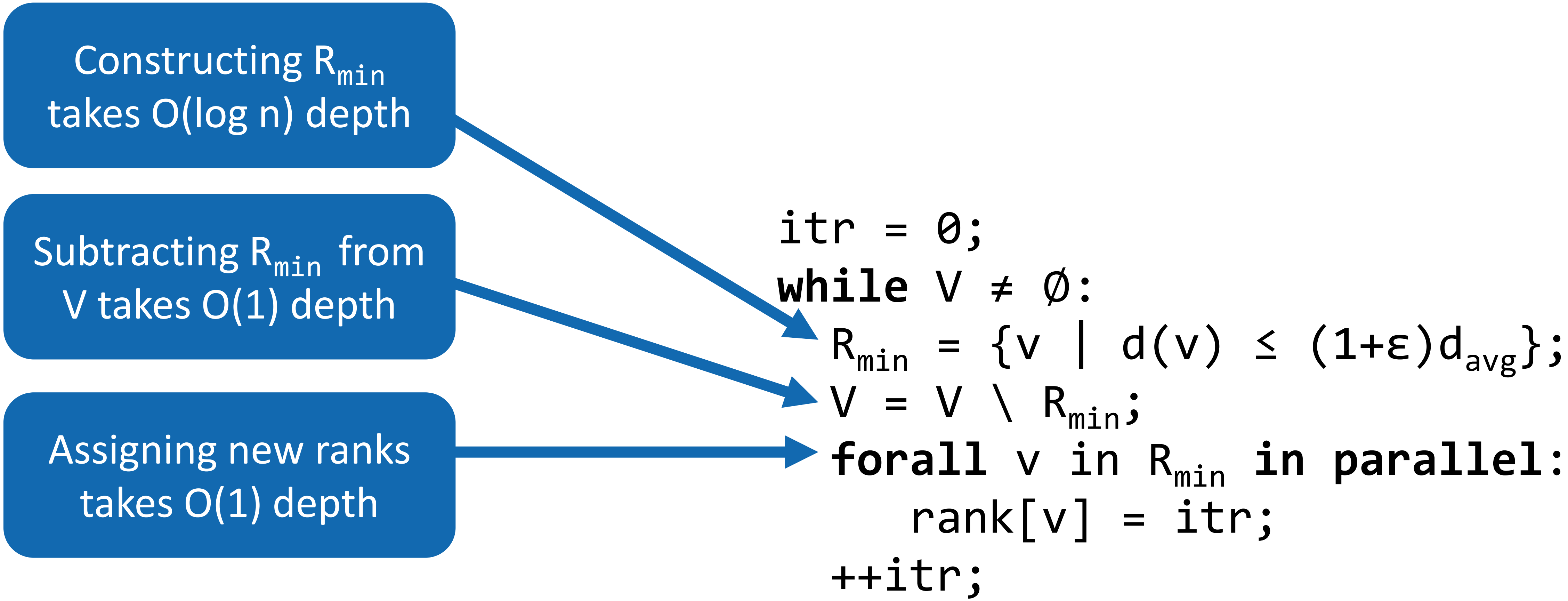
Subtracting R_{min} from V takes O(1) depth

```
itr = 0;
while V ≠ Ø:
    $R_{min}$ = {v | d(v) ≤ (1+ε)$d_{avg}$};
    V = V \ $R_{min}$;
    forall v in $R_{min}$ in parallel:
        rank[v] = itr;
    ++itr;
```

Approximate degeneracy ordering

```
/* n: the number of all vertices
   V: set of all vertices,
   d(v): degree of a vertex v,
   $d_{avg}$: average degree in V */
```

→ **Key idea**: try a relaxation of the strict degeneracy order, at the cost of (some) accuracy loss.



Constructing R_{min} takes O(log n) depth

Subtracting R_{min} from V takes O(1) depth

Assigning new ranks takes O(1) depth

```
itr = 0;
while V ≠ ∅:
    $R_{min} = \{v \mid d(v) \le (1+\varepsilon)d_{avg}\}$;
    $V = V \setminus R_{min}$;
    forall v in $R_{min}$ in parallel:
        rank[v] = itr;
    ++itr;
```

Approximate degeneracy ordering

→ **Key idea**: try a relaxation of the strict degeneracy order, at the cost of (some) accuracy loss.

```
/* n: the number of all vertices
   V: set of all vertices,
   d(v): degree of a vertex v,
   $d_{avg}$: average degree in V */
```

Constructing R_{min} takes O(log n) depth

Subtracting R_{min} from V takes O(1) depth

Assigning new ranks takes O(1) depth

Thus, a single iteration takes O(log n) depth

```
itr = 0;
while $V \neq \emptyset$:
    $R_{min} = \{v \mid d(v) \leq (1+\varepsilon)d_{avg}\}$;
    $V = V \setminus R_{min}$;
    forall v in $R_{min}$ in parallel:
        rank[v] = itr;
    ++itr;
```

Approximate degeneracy ordering

```
/* n: the number of all vertices
   V: set of all vertices,
   d(v): degree of a vertex v,
   $d_{avg}$: average degree in V */
```

→ **Key idea**: try a relaxation of the strict degeneracy order, at the cost of (some) accuracy loss.

One can prove that R_{min} forms a constant fraction of all vertices

Thus, a single iteration takes O(log n) depth

Constructing R_{min} takes O(log n) depth

Subtracting R_{min} from V takes O(1) depth

Assigning new ranks takes O(1) depth

```
itr = 0;
while V ≠ Ø:
    $R_{min}$ = {v | d(v) ≤ $(1+\varepsilon)d_{avg}$};
    V = V \ $R_{min}$;
    forall v in $R_{min}$ in parallel:
        rank[v] = itr;
    ++itr;
```

Approximate degeneracy ordering

→ **Key idea**: try a relaxation of the strict degeneracy order, at the cost of (some) accuracy loss.

```
/* n: the number of all vertices
   V: set of all vertices,
   d(v): degree of a vertex v,
   $d_{avg}$: average degree in V */
```

Constructing R_{min} takes O(log n) depth

Subtracting R_{min} from V takes O(1) depth

Assigning new ranks takes O(1) depth

Thus, a single iteration takes O(log n) depth

One can prove that R_{min} forms a constant fraction of all vertices

Thus, there are O(log n) iterations, giving O($\log^2 n$) total depth

```
itr = 0;
while V ≠ ∅:
  $R_{min} = \{v \mid d(v) \leq (1+\varepsilon)d_{avg}\}$;
  $V = V \setminus R_{min}$;
  forall v in $R_{min}$ in parallel:
     rank[v] = itr;
  ++itr;
```

Approximate degeneracy ordering

→ **Key idea**: try a relaxation of the strict degeneracy order, at the cost of (some) accuracy loss.

```
/* n: t              of all vertices
   V: s
   d(v                             ,
   d_avg: average                  /
```

All iterations take O(n+m) work

One can prove that R_{min} forms a constant fraction of all vertices

Constructing R_{min} takes O(log n) depth

Thus, a single iteration takes O(log n) depth

Subtracting R_{min} from V takes O(1) depth

Thus, there are O(log n) iterations, giving O($\log^2 n$) total depth

Assigning new ranks takes O(1) depth

```
itr = 0;
while V ≠ ∅:
    $R_{min} = \{v \mid d(v) \leq (1+\varepsilon)d_{avg}\}$;
    V = V \ $R_{min}$;
    forall v in $R_{min}$ in parallel:
        rank[v] = itr;
    ++itr;
```

Approximate degeneracy ordering

→ **Key idea**: try a relaxation of the strict degeneracy order, at the cost of (some) accuracy loss.

```
/* n: the number of all vertices
   V: set of all vertices,
   d(v): degree of a vertex v,
   d_avg: average degree in V */
```

Approximate degeneracy ordering

/* n: the number of all vertices
V: set of all vertices,
d(v): degree of a vertex v,
d_{avg}: average degree in V */

→ **Key idea**: try a relaxation of the strict degeneracy order, at the cost of (some) accuracy loss.

Strict degeneracy order:

ADG: approximate degeneracy order:

Approximate degeneracy ordering

```
/* n: the number of all vertices
   V: set of all vertices,
   d(v): degree of a vertex v,
   d_avg: average degree in V */
```

→ **Key idea**: try a relaxation of the strict degeneracy order, at the cost of (some) accuracy loss.

Strict degeneracy order:

ADG: approximate degeneracy order:

Work: O(n+m)

Depth: O(n)

Approximate degeneracy ordering

/* n: the number of all vertices
V: set of all vertices,
d(v): degree of a vertex v,
d_{avg}: average degree in V */

→ **Key idea**: try a relaxation of the strict degeneracy order, at the cost of (some) accuracy loss.

Strict degeneracy order: → **ADG: approximate degeneracy order:**

Work: O(n+m)

Depth: O(n)

Work: O(n+m)

Depth: O($\log^2$ n)

Approximate degeneracy ordering

```
/* n: the number of all vertices
   V: set of all vertices,
   d(v): degree of a vertex v,
   $d_{avg}$: average degree in V */
```

→ **Key idea**: try a relaxation of the strict degeneracy order, at the cost of (some) accuracy loss.

Strict degeneracy order: → **ADG: approximate degeneracy order:**

Work: O(n+m)

Depth: O(n)

Work: O(n+m)

Depth: $O(\log^2 n)$

Approximation: 2(1+ε)

Relaxation approximates the degeneracy by a 2(1+ε) multiplicative factor

Approximate degeneracy ordering

```
/* n: the number of all vertices
   V: set of all vertices,
   d(v): degree of a vertex v,
   d_avg: average degree in V */
```

→ **Key idea**: try a relaxation of the strict degeneracy order, at the cost of (some) accuracy loss.

Strict degeneracy order:

ADG: approximate deg

Work: O(

Depth: O

How does it impact the actual graph coloring?

2(1+ε)

Relaxation approximates the degeneracy by a 2(1+ε) multiplicative factor

Parallel graph coloring heuristics

Parallel graph coloring heuristics

Let's see how the coloring heuristic uses the orderings

Parallel graph coloring heuristics

Let’s see how the coloring heuristic uses the orderings

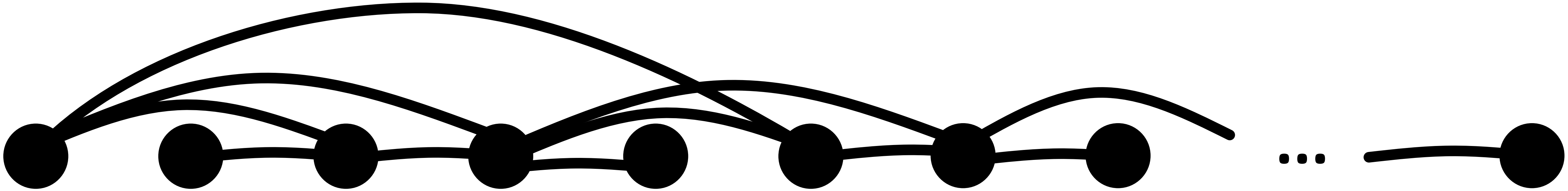
```
for each vertex $v_i$ in ($v_1$ … $v_n$):
  find smallest color c not
    used by the neighbors of $v_i$;
  assign c to $v_i$;
```

Parallel graph coloring heuristics

```
for each vertex $v_i$ in ($v_1$ … $v_n$):
  find smallest color c not
    used by the neighbors of $v_i$;
  assign c to $v_i$;
```

Let’s see how the coloring heuristic uses the orderings

→ The degeneracy ordering of a given graph is an ordering, where each vertex v has at most d neighbors that are ordered higher than v (d is G’s degeneracy).

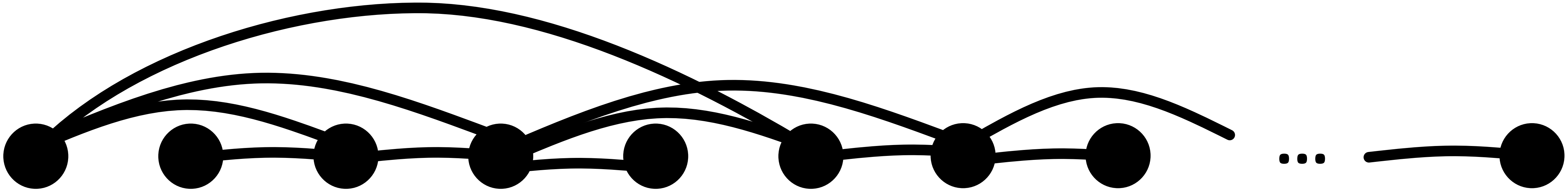


Parallel graph coloring heuristics

Let's see how the coloring heuristic uses the orderings

```
for each vertex $v_i$ in ($v_1$ … $v_n$):
  find smallest color c not
    used by the neighbors of $v_i$;
  assign c to $v_i$;
```

→ The degeneracy ordering of a given graph is an ordering, where each vertex v has at most d neighbors that are ordered higher than v (d is G's degeneracy).



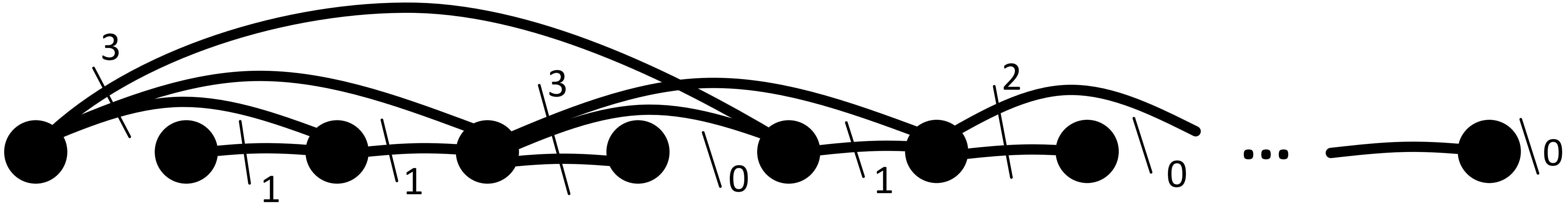
A degeneracy ordering of a 3-degenerate graph

Parallel graph coloring heuristics

```
for each vertex $v_i$ in ($v_1$ … $v_n$):
  find smallest color c not
    used by the neighbors of $v_i$;
  assign c to $v_i$;
```

Let's see how the coloring heuristic uses the orderings

→ The degeneracy ordering of a given graph is an ordering, where each vertex v has at most d neighbors that are ordered higher than v (d is G's degeneracy).



A degeneracy ordering of a 3-degenerate graph

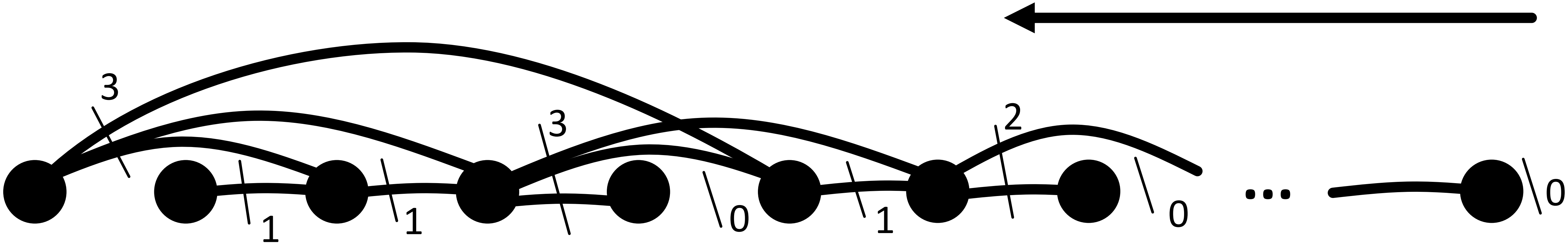
Parallel graph coloring heuristics

Let's see how the coloring heuristic uses the orderings

→ The degeneracy ordering of a given graph is an ordering, where each vertex v has at most d neighbors that are ordered higher than v (d is G's degeneracy).

```
for each vertex $v_i$ in ($v_1$ … $v_n$):
  find smallest color c not
    used by the neighbors of $v_i$;
  assign c to $v_i$;
```

Color vertices one by one, assigning a lowest color not used by the neighbors „on the right"



A degeneracy ordering of a 3-degenerate graph

Parallel graph coloring heuristics

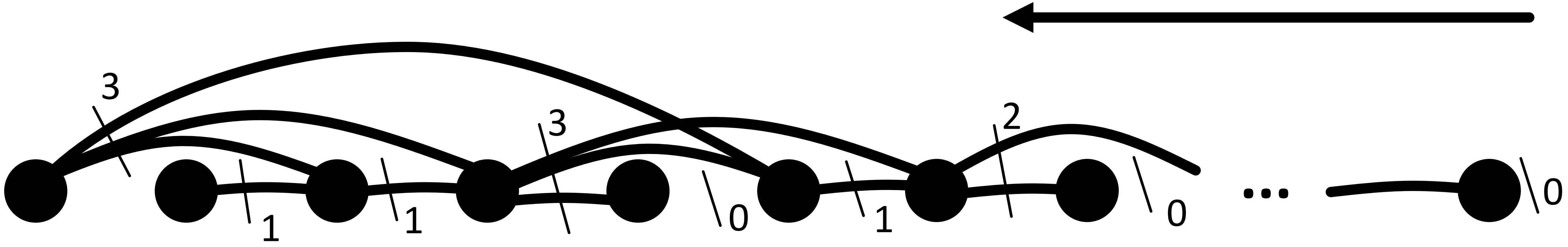
```
for each vertex $v_i$ in ($v_1$ … $v_n$):
  find smallest color c not
    used by the neighbors of $v_i$;
  assign c to $v_i$;
```

Let's see how the coloring heuristic uses the orderings

→ The degeneracy ordering of a given graph is an ordering, where each vertex v has at most d neighbors that are ordered higher than v (d is G's degeneracy).

Using the strict degeneracy ordering, we get at most d+1 colors

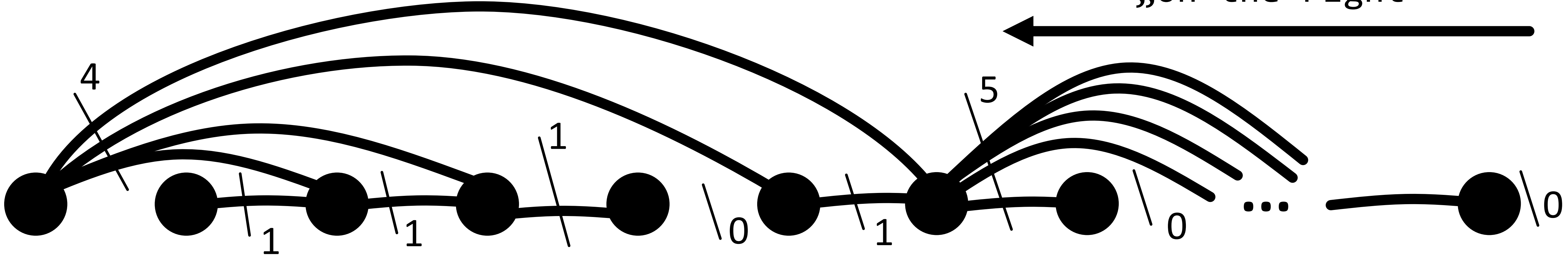
Color vertices one by one, assigning a lowest color not used by the neighbors „on the right"



A degeneracy ordering of a 3-degenerate graph

Parallel graph coloring heuristics + ADG

Color vertices one by one,
assigning a lowest color
not used by the neighbors
„on the right”

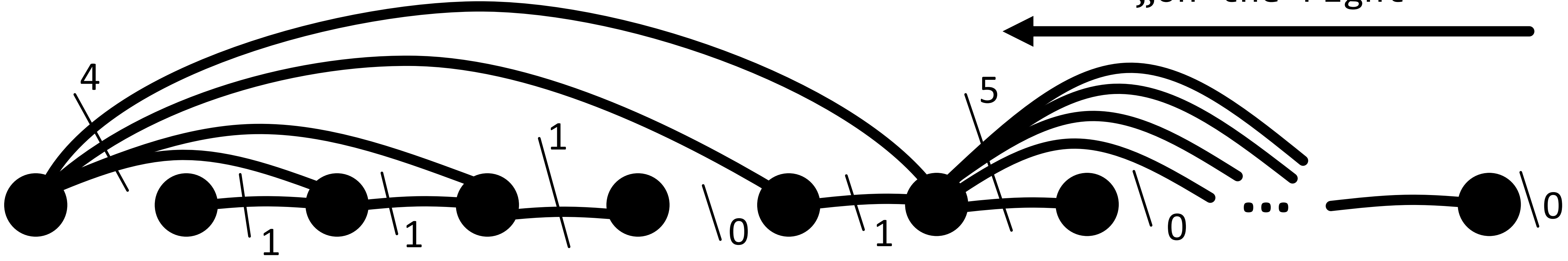


A 2(1+ε)-approximate degeneracy ordering of a 3-degenerate graph

Parallel graph coloring heuristics + ADG

In the ADG ordering, each vertex v has at most 2(1+ε)d neighbors that are ordered higher than v

Color vertices one by one, assigning a lowest color not used by the neighbors „on the right"



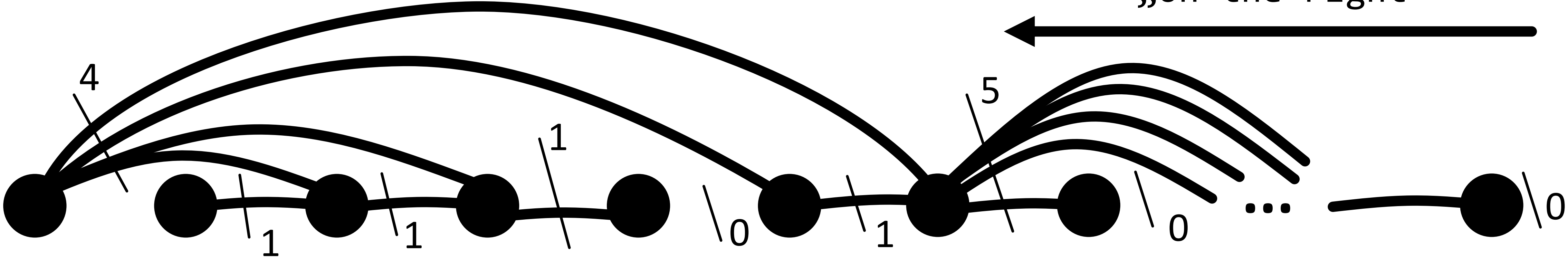
A 2(1+ε)-approximate degeneracy ordering of a 3-degenerate graph

Parallel graph coloring heuristics + ADG

In the ADG ordering, each vertex v has at most 2(1+ε)d neighbors that are ordered higher than v

Using ADG, we get at most 2(1+ε)d + 1 colors

Color vertices one by one, assigning a lowest color not used by the neighbors „on the right"



A 2(1+ε)-approximate degeneracy ordering of a 3-degenerate graph

Parallel graph coloring heuristics + ADG

```
/* n: number of vertices,
   m: number of edges,
   Δ: maximum vertex degree,
   d: graph’s degeneracy */
```

Ordering	Depth	Work	Quality
“First fit” (i.e., any order)	No general bounds; $\Omega(n)$ for some graphs	$O(n+m)$	$\Delta+1$
“Largest degree first”	No general bounds; $\Omega(\Delta^2)$ for some graphs	$O(n+m)$	$\Delta+1$
“Smallest degree last”	No general bounds; $\Omega(n)$ for some graphs	$O(n+m)$	$d+1$
Random	$\mathbb{E}\, O\left(\frac{\log n}{\log\log n}\right)$	$O(n+m)$	$\Delta+1$
Random	$\mathbb{E}\, O\left(\log n + \log\Delta \cdot \min\left\{\sqrt{m}, \Delta + \frac{\log\Delta \log n}{\log\log n}\right\}\right)$	$O(n+m)$	$\Delta+1$
“Largest log-degree first”	$\mathbb{E}\, O\left(\log n + \log\Delta \cdot \left(\min\left\{\Delta, \sqrt{m}\right\} + \frac{\log^2\Delta \log n}{\log\log n}\right)\right)$	$O(n+m)$	$\Delta+1$
“Smallest log-degree last”	$\mathbb{E}\, O\left(\log\Delta \log n + \log\Delta \cdot \left(\min\left\{\Delta, \sqrt{m}\right\} + \frac{\log^2\Delta \log n}{\log\log n}\right)\right)$	$O(n+m)$	$\Delta+1$

Parallel graph coloring heuristics + ADG

```
/* n: number of vertices,
   m: number of edges,
   Δ: maximum vertex degree,
   d: graph's degeneracy */
```

Ordering	Depth	Work	Quality
"First fit" (i.e., any order)	No general bounds; $\Omega(n)$ for some graphs	$O(n+m)$	$\Delta+1$
"Largest degree first"	No general bounds; $\Omega(\Delta^2)$ for some graphs	$O(n+m)$	$\Delta+1$
"Smallest degree last"	No general bounds; $\Omega(n)$ for some graphs	$O(n+m)$	$d+1$
Random	$\mathbb{E}\,O\left(\frac{\log n}{\log\log n}\right)$	$O(n+m)$	$\Delta+1$
Random	$\mathbb{E}\,O\left(\log n+\log\Delta\cdot\min\left\{\sqrt{m},\Delta+\frac{\log\Delta\log n}{\log\log n}\right\}\right)$	$O(n+m)$	$\Delta+1$
"Largest log-degree first"	$\mathbb{E}\,O\left(\log n+\log\Delta\cdot\left(\min\{\Delta,\sqrt{m}\}+\frac{\log^2\Delta\log n}{\log\log n}\right)\right)$	$O(n+m)$	$\Delta+1$
"Smallest log-degree last"	$\mathbb{E}\,O\left(\log\Delta\log n+\log\Delta\cdot\left(\min\{\Delta,\sqrt{m}\}+\frac{\log^2\Delta\log n}{\log\log n}\right)\right)$	$O(n+m)$	$\Delta+1$
ADG			

Parallel graph coloring heuristics + ADG

```
/* n: number of vertices,
   m: number of edges,
   Δ: maximum vertex degree,
   d: graph’s degeneracy */
```

Ordering	Depth	Work	Quality
“First fit” (i.e., any order)	No general bounds; $\Omega(n)$ for some graphs	$O(n+m)$	$\Delta+1$
“Largest degree first”	No general bounds; $\Omega(\Delta^2)$ for some graphs	$O(n+m)$	$\Delta+1$
“Smallest degree last”	No general bounds; $\Omega(n)$ for some graphs	$O(n+m)$	$d+1$
Random	$\mathbb{E}\, O\left(\frac{\log n}{\log\log n}\right)$	$O(n+m)$	$\Delta+1$
Random	$\mathbb{E}\, O\left(\log n + \log\Delta \cdot \min\left\{\sqrt{m}, \Delta + \frac{\log\Delta\log n}{\log\log n}\right\}\right)$	$O(n+m)$	$\Delta+1$
“Largest log-degree first”	$\mathbb{E}\, O\left(\log n + \log\Delta \cdot \left(\min\{\Delta, \sqrt{m}\} + \frac{\log^2\Delta\log n}{\log\log n}\right)\right)$	$O(n+m)$	$\Delta+1$
“Smallest log-degree last”	$\mathbb{E}\, O\left(\log\Delta\log n + \log\Delta \cdot \left(\min\{\Delta, \sqrt{m}\} + \frac{\log^2\Delta\log n}{\log\log n}\right)\right)$	$O(n+m)$	$\Delta+1$
ADG			$2(1+\varepsilon)d+1$

Parallel graph coloring heuristics + ADG

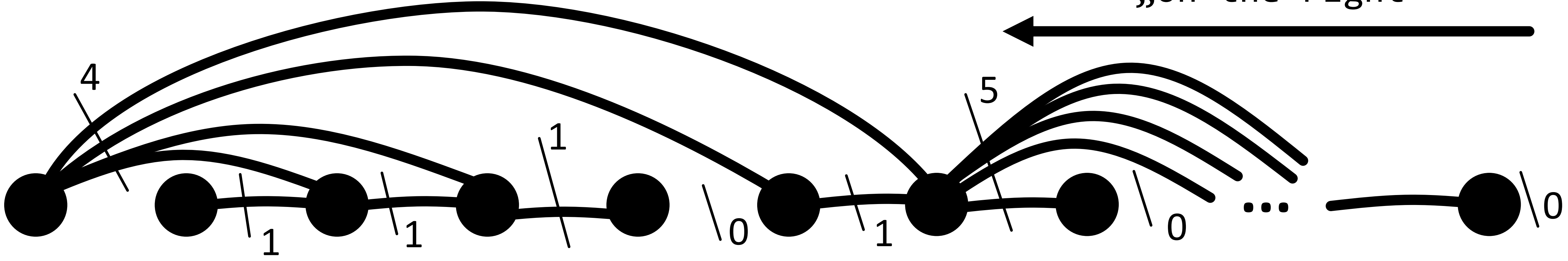
```
/* n: number of vertices,
   m: number of edges,
   Δ: maximum vertex degree,
   d: graph's degeneracy */
```

Ordering	Depth	Work	Quality
"First fit" (i.e., any order)	No general bounds; $\Omega(n)$ for some graphs	$O(n+m)$	$\Delta+1$
"Largest degree first"	No general bounds; $\Omega(\Delta^2)$ for some graphs	$O(n+m)$	$\Delta+1$
"Smallest degree last"	No general bounds; $\Omega(n)$ for some graphs	$O(n+m)$	$d+1$
Random	$\mathbb{E}\, O\left(\frac{\log n}{\log\log n}\right)$	$O(n+m)$	$\Delta+1$
Random	$\mathbb{E}\, O\left(\log n + \log\Delta \cdot \min\left\{\sqrt{m}, \Delta + \frac{\log\Delta\log n}{\log\log n}\right\}\right)$	$O(n+m)$	$\Delta+1$
"Largest log-degree first"	$\mathbb{E}\, O\left(\log n + \log\Delta \cdot \left(\min\{\Delta, \sqrt{m}\} + \frac{\log^2\Delta\log n}{\log\log n}\right)\right)$	$O(n+m)$	$\Delta+1$
"Smallest log-degree last"	$\mathbb{E}\, O\left(\log\Delta\log n + \log\Delta \cdot \left(\min\{\Delta, \sqrt{m}\} + \frac{\log^2\Delta\log n}{\log\log n}\right)\right)$	$O(n+m)$	$\Delta+1$
ADG	?	?	$2(1+\varepsilon)d+1$

Parallel graph coloring heuristics + ADG

In the ADG ordering, each vertex v has at most 2(1+ε)d neighbors that are ordered higher than v

Color vertices one by one, assigning a lowest color not used by the neighbors „on the right"

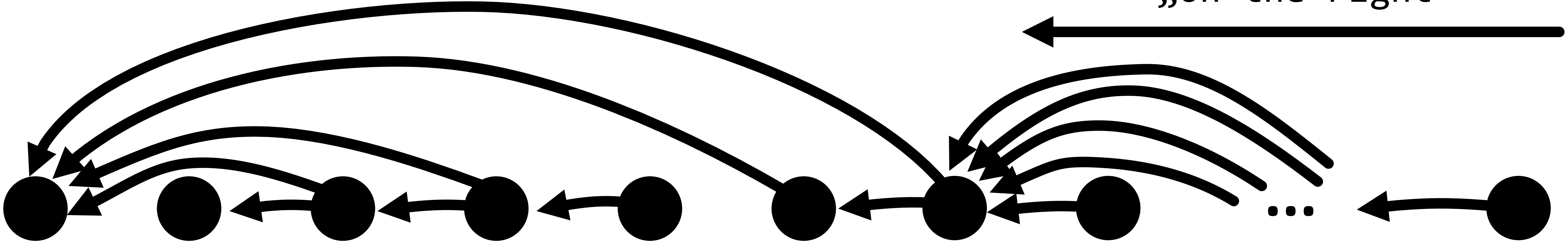


A 2(1+ε)-approximate degeneracy ordering of a 3-degenerate graph

Parallel graph coloring heuristics + ADG

In the ADG ordering, each vertex v has at most 2(1+ε)d neighbors that are ordered higher than v

We consider a DAG imposed over the input graph G, with directions assigned based on the used vertex ordering



Color vertices one by one, assigning a lowest color not used by the neighbors „on the right”

A 2(1+ε)-approximate degeneracy ordering of a 3-degenerate graph

Parallel graph coloring heuristics + ADG

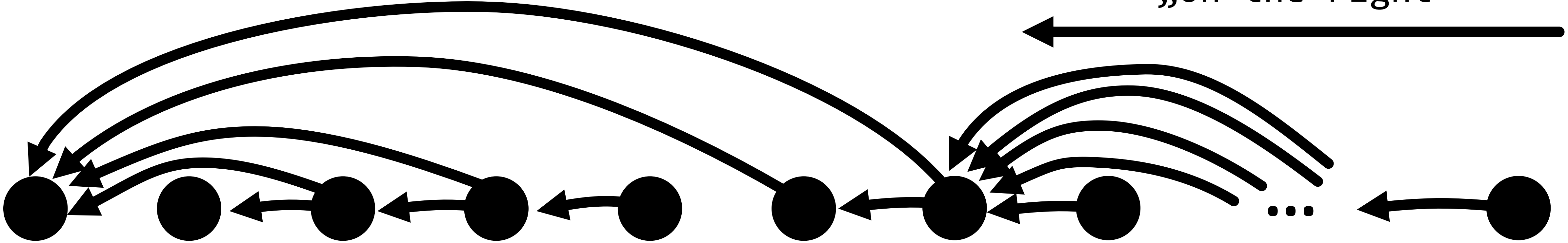
[1] W. Hasenplaugh, T. Kaler, T. B. Schardl, and C. E. Leiserson, "Ordering heuristics for parallel graph coloring". SPAA'14.

In the ADG ordering, each vertex v has at most 2(1+ε)d neighbors that are ordered higher than v

We consider a DAG imposed over the input graph G, with directions assigned based on the used vertex ordering

Now, it was proved that a parallel coloring heuristics runs in O(|P| log Δ + log n) depth and O(n+m) work [1].

`Color vertices one by one, assigning a lowest color not used by the neighbors „on the right"`



A 2(1+ε)-approximate degeneracy ordering of a 3-degenerate graph

Parallel graph coloring heuristics + ADG

[1] W. Hasenplaugh, T. Kaler, T. B. Schardl, and C. E. Leiserson, "Ordering heuristics for parallel graph coloring". SPAA'14.

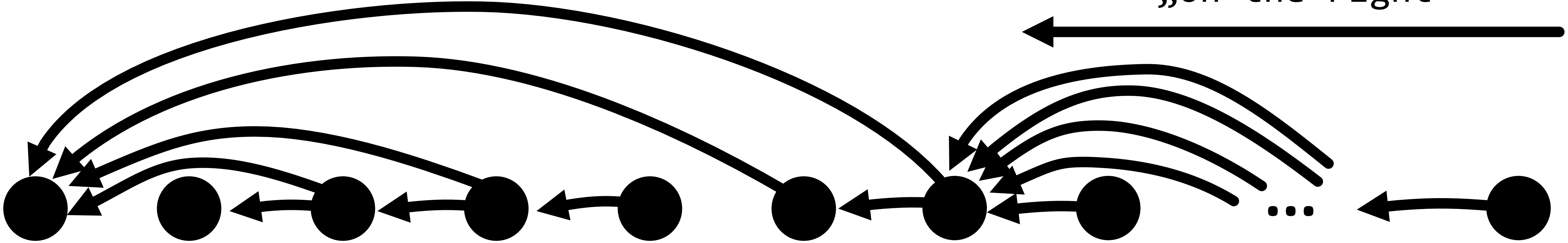
In the ADG ordering, each vertex v has at most 2(1+ε)d neighbors that are ordered higher than v

We consider a DAG imposed over the input graph G, with directions assigned based on the used vertex ordering

Now, it was proved that a parallel coloring heuristics runs in O(|P| log Δ + log n) depth and O(n+m) work [1].

What is |P| when using ADG?

Color vertices one by one, assigning a lowest color not used by the neighbors „on the right"



A 2(1+ε)-approximate degeneracy ordering of a 3-degenerate graph

Parallel graph coloring heuristics + ADG

[1] W. Hasenplaugh, T. Kaler, T. B. Schardl, and C. E. Leiserson, “Ordering heuristics for parallel graph coloring”. SPAA’14.

In the ADG ordering, each vertex v has at most 2(1+ε)d neighbors that are ordered higher than v

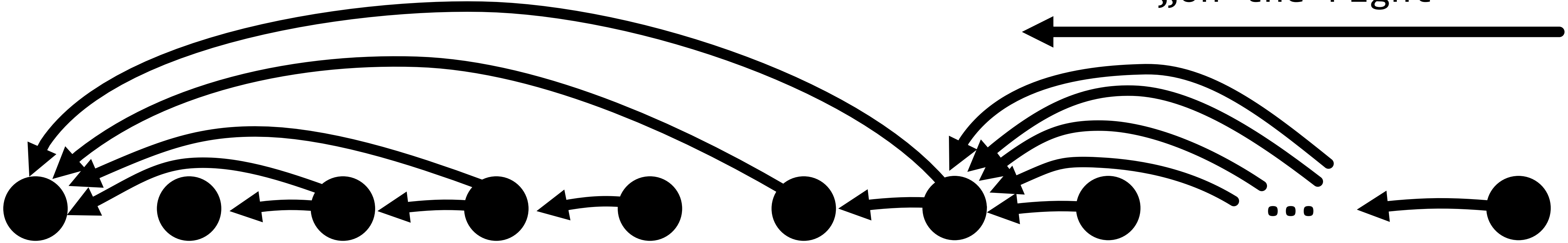
We consider a DAG imposed over the input graph G, with directions assigned based on the used vertex ordering

Now, it was proved that a parallel coloring heuristics runs in O(|P| log Δ + log n) depth and O(n+m) work [1].

What is |P| when using ADG?

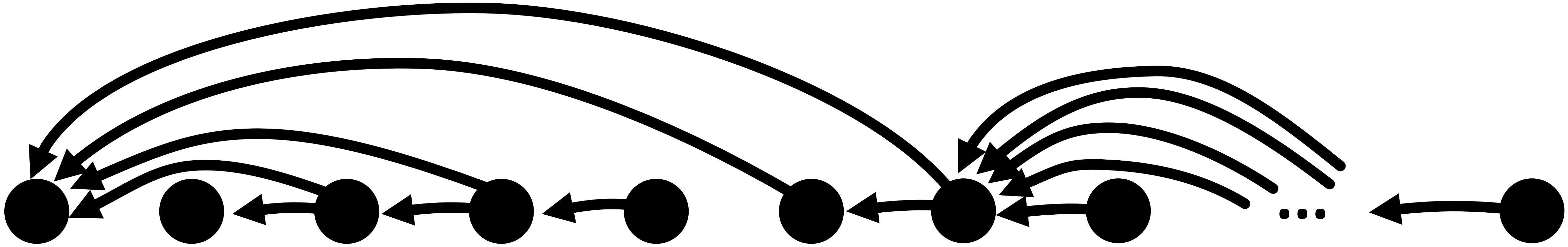
Let’s see some intuition

Color vertices one by one, assigning a lowest color not used by the neighbors „on the right”



A 2(1+ε)-approximate degeneracy ordering of a 3-degenerate graph

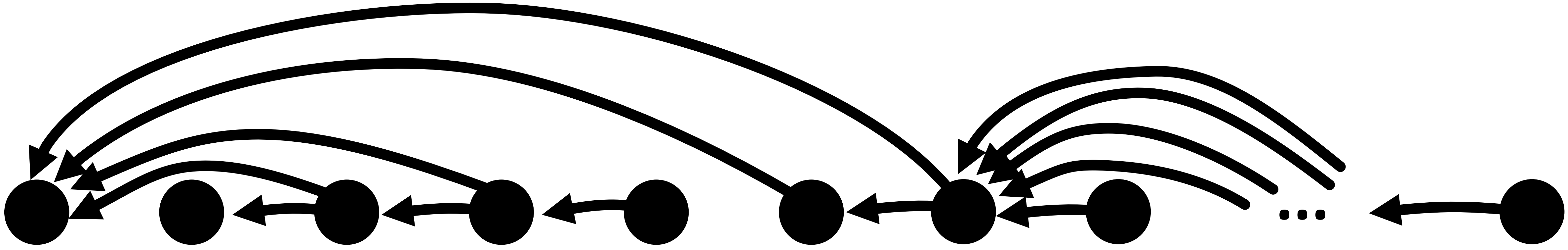
Parallel graph coloring heuristics + ADG



A 2(1+ε)-approximate degeneracy ordering of a 3-degenerate graph

Parallel graph coloring heuristics + ADG

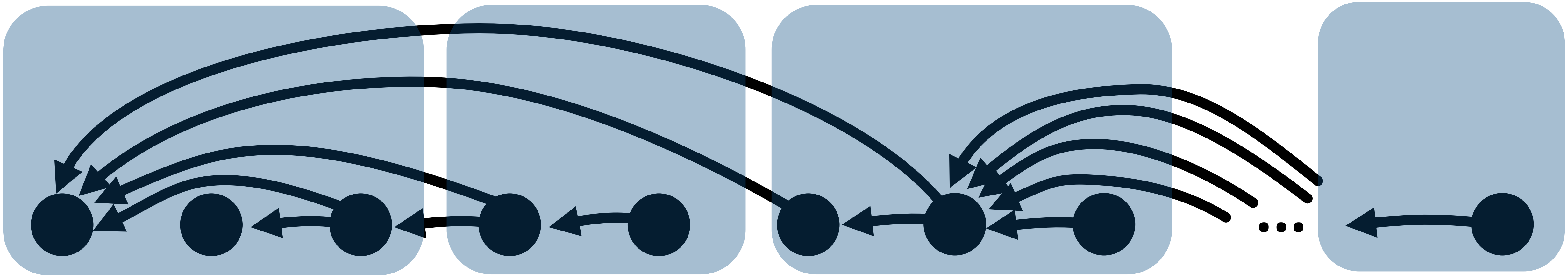
Vertices with the same ADG rank form subgraphs



A 2(1+ε)-approximate degeneracy ordering of a 3-degenerate graph

Parallel graph coloring heuristics + ADG

Vertices with the same ADG rank form subgraphs

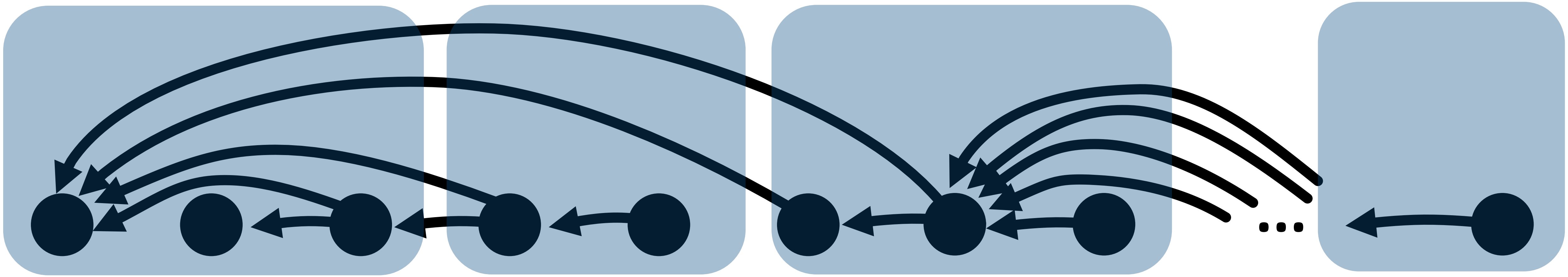


A 2(1+ε)-approximate degeneracy ordering of a 3-degenerate graph

Parallel graph coloring heuristics + ADG

Vertices with the same ADG rank form subgraphs

Analyze |P| by analyzing the lengths of its parts, going via each subgraph



A 2(1+ε)-approximate degeneracy ordering of a 3-degenerate graph

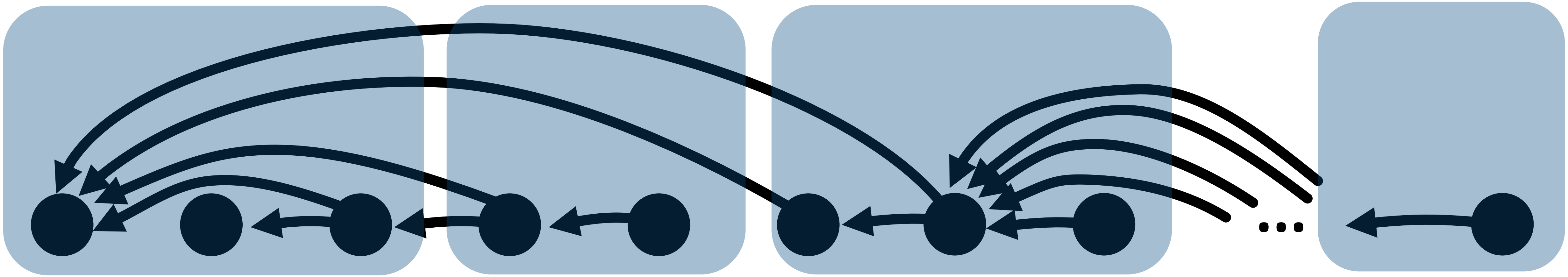
Parallel graph coloring heuristics + ADG

Vertices with the same ADG rank form subgraphs

Analyze |P| by analyzing the lengths of its parts, going via each subgraph

\+

By ADG, each vertex has a <u>bounded</u> degree in each subgraph



A 2(1+ε)-approximate degeneracy ordering of a 3-degenerate graph

Parallel graph coloring heuristics + ADG

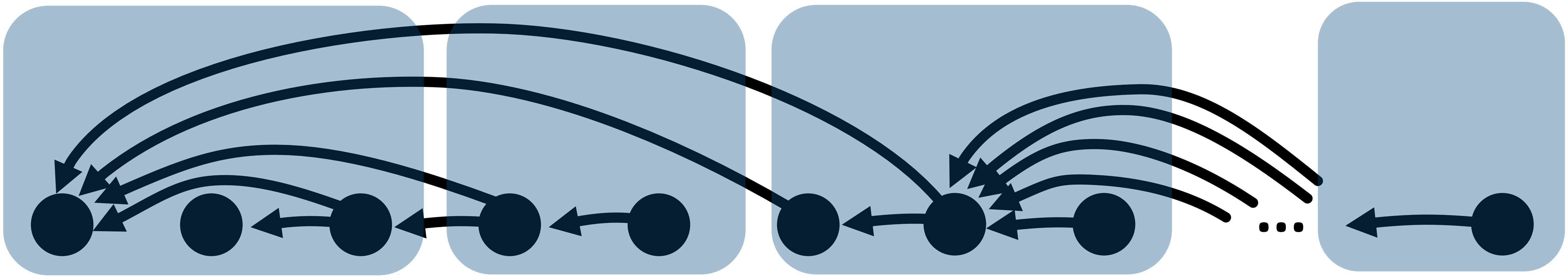
Vertices with the same ADG rank form subgraphs

Analyze |P| by analyzing the lengths of its parts, going via each subgraph

+

By ADG, each vertex has a <u>bounded</u> degree in each subgraph

“There is only as far (constant) as you can go in a subgraph”



A 2(1+ε)-approximate degeneracy ordering of a 3-degenerate graph

Parallel graph coloring heuristics

```
/* n: number of vertices,
   m: number of edges,
   Δ: maximum vertex degree,
   d: graph's degeneracy */
```

Ordering	Depth	Work	Quality
"First fit" (i.e., any order)	No general bounds; $\Omega(n)$ for some graphs	$O(n+m)$	$\Delta + 1$
"Largest degree first"	No general bounds; $\Omega(\Delta^2)$ for some graphs	$O(n+m)$	$\Delta + 1$
"Smallest degree last"	No general bounds; $\Omega(n)$ for some graphs	$O(n+m)$	$d + 1$
Random	$\mathbb{E}\, O\left(\frac{\log n}{\log\log n}\right)$	$O(n+m)$	$\Delta + 1$
Random	$\mathbb{E}\, O\left(\log n + \log \Delta \cdot \min\left\{\sqrt{m}, \Delta + \frac{\log \Delta \log n}{\log\log n}\right\}\right)$	$O(n+m)$	$\Delta + 1$
"Largest log-degree first"	$\mathbb{E}\, O\left(\log n + \log \Delta \cdot \left(\min\{\Delta, \sqrt{m}\} + \frac{\log^2 \Delta \log n}{\log\log n}\right)\right)$	$O(n+m)$	$\Delta + 1$
"Smallest log-degree last"	$\mathbb{E}\, O\left(\log \Delta \log n + \log \Delta \cdot \left(\min\{\Delta, \sqrt{m}\} + \frac{\log^2 \Delta \log n}{\log\log n}\right)\right)$	$O(n+m)$	$\Delta + 1$
ADG	?	?	$2(1+\varepsilon)d + 1$

Parallel graph coloring heuristics

```
/* n: number of vertices,
   m: number of edges,
   Δ: maximum vertex degree,
   d: graph’s degeneracy */
```

Ordering	Depth	Work	Quality
“First fit” (i.e., any order)	No general bounds; $\Omega(n)$ for some graphs	$O(n+m)$	$\Delta+1$
“Largest degree first”	No general bounds; $\Omega(\Delta^2)$ for some graphs	$O(n+m)$	$\Delta+1$
“Smallest degree last”	No general bounds; $\Omega(n)$ for some graphs	$O(n+m)$	$d+1$
Random	$\mathbb{E}\, O\left(\frac{\log n}{\log\log n}\right)$	$O(n+m)$	$\Delta+1$
Random	$\mathbb{E}\, O\left(\log n + \log\Delta \cdot \min\left\{\sqrt{m}, \Delta + \frac{\log\Delta \log n}{\log\log n}\right\}\right)$	$O(n+m)$	$\Delta+1$
“Largest log-degree first”	$\mathbb{E}\, O\left(\log n + \log\Delta \cdot \left(\min\{\Delta, \sqrt{m}\} + \frac{\log^2\Delta \log n}{\log\log n}\right)\right)$	$O(n+m)$	$\Delta+1$
“Smallest log-degree last”	$\mathbb{E}\, O\left(\log\Delta \log n + \log\Delta \cdot \left(\min\{\Delta, \sqrt{m}\} + \frac{\log^2\Delta \log n}{\log\log n}\right)\right)$	$O(n+m)$	$\Delta+1$
ADG	$\mathbb{E}\, O\left(\log^2 n + \log\Delta \cdot \left(d\log n + \frac{\log d \cdot \log^2 n}{\log\log n}\right)\right)$	$O(n+m)$	$2(1+\varepsilon)d+1$

Parallel graph coloring heuristics

```
/* n: number of vertices,
   m: number of edges,
   Δ: maximum vertex degree,
   d: graph’s degeneracy */
```

Ordering	**Depth**	**Work**	**Quality**
“First fit” (i.e., any order)	No general bounds; $\Omega(n)$ for some graphs	$O(n+m)$	$\Delta+1$
“Largest degree first”	No general bounds; $\Omega(\Delta^2)$ for some graphs	$O(n+m)$	$\Delta+1$
“Smallest degree last”	No general bounds; $\Omega(n)$ for some graphs	$O(n+m)$	$d+1$
Random	$\mathbb{E}\ O\left(\frac{\log n}{\log\log n}\right)$	$O(n+m)$	$\Delta+1$
Random	$\mathbb{E}\ O\left(\log n+\log\Delta\cdot\min\left\{\sqrt{m},\Delta+\frac{\log\Delta\log n}{\log\log n}\right\}\right)$	$O(n+m)$	$\Delta+1$
“Largest log-degree first”	$\mathbb{E}\ O\left(\log n+\log\Delta\cdot\left(\min\{\Delta,\sqrt{m}\}+\frac{\log^2\Delta\log n}{\log\log n}\right)\right)$	$O(n+m)$	$\Delta+1$
“Smallest log-degree last”	$\mathbb{E}\ O\left(\log\Delta\log n+\log\Delta\cdot\left(\min\{\Delta,\sqrt{m}\}+\frac{\log^2\Delta\log n}{\log\log n}\right)\right)$	$O(n+m)$	$\Delta+1$
ADG	$\mathbb{E}\ O\left(\log^2 n+\log\Delta\cdot\left(d\log n+\frac{\log d\cdot\log^2 n}{\log\log n}\right)\right)$	$O(n+m)$	$2(1+\varepsilon)d+1$

Parallel graph coloring heuristics

```
/* n: number of vertices,
   m: number of edges,
   Δ: maximum vertex degree,
   d: graph’s degeneracy */
```

Ordering	Depth	Work	Quality
“First fit” (i.e., any order)	No general bounds; $\Omega(n)$ for some graphs	$O(n+m)$	$\Delta+1$
“Largest degree first”	No general bounds; $\Omega(\Delta^2)$ for some graphs	$O(n+m)$	$\Delta+1$
“Smallest degree last”	No general bounds; $\Omega(n)$ for some graphs	$O(n+m)$	$d+1$
Random	$\mathbb{E}\, O\left(\frac{\log n}{\log\log n}\right)$	$O(n+m)$	$\Delta+1$
Random	$\mathbb{E}\, O\left(\log n + \log\Delta \cdot \min\left\{\sqrt{m}, \Delta + \frac{\log\Delta \log n}{\log\log n}\right\}\right)$	$O(n+m)$	$\Delta+1$
“Largest log-degree first”	$\mathbb{E}\, O\left(\log n + \log\Delta \cdot \left(\min\{\Delta, \sqrt{m}\} + \frac{\log^2\Delta \log n}{\log\log n}\right)\right)$	$O(n+m)$	$\Delta+1$
“Smallest log-degree last”	$\mathbb{E}\, O\left(\log\Delta \log n + \log\Delta \cdot \left(\min\{\Delta, \sqrt{m}\} + \frac{\log^2\Delta \log n}{\log\log n}\right)\right)$	$O(n+m)$	$\Delta+1$
ADG	$\mathbb{E}\, O\left(\log^2 n + \log\Delta \cdot \left(d\log n + \frac{\log d \cdot \log^2 n}{\log\log n}\right)\right)$	$O(n+m)$	$2(1+\varepsilon)d+1$

Parallel graph coloring heuristics

```
/* n: number of vertices,
   m: number of edges,
   Δ: maximum vertex degree,
   d: graph's degeneracy */
```

Ordering	Depth	Work	Quality
"First fit" (i.e., any order)	No general bounds; $\Omega(n)$ for some graphs	$O(n+m)$	$\Delta+1$
"Largest degree first"	No general bounds; $\Omega(\Delta^2)$ for some graphs	$O(n+m)$	$\Delta+1$
"Smallest degree last"	No general bounds; $\Omega(n)$ for some graphs	$O(n+m)$	$d+1$
Random	$\mathbb{E}\, O\left(\frac{\log n}{\log\log n}\right)$	$O(n+m)$	$\Delta+1$
Random	$\mathbb{E}\, O\left(\log n + \log\Delta \cdot \min\left\{\sqrt{m}, \Delta + \frac{\log\Delta \log n}{\log\log n}\right\}\right)$	$O(n+m)$	$\Delta+1$
"Largest log-degree first"	$\mathbb{E}\, O\left(\log n + \log\Delta \cdot \left(\min\{\Delta, \sqrt{m}\} + \frac{\log^2\Delta \log n}{\log\log n}\right)\right)$	$O(n+m)$	$\Delta+1$
"Smallest log-degree last"	$\mathbb{E}\, O\left(\log\Delta \log n + \log\Delta \cdot \left(\min\{\Delta, \sqrt{m}\} + \frac{\log^2\Delta \log n}{\log\log n}\right)\right)$	$O(n+m)$	$\Delta+1$
ADG	$\mathbb{E}\, O\left(\log^2 n + \log\Delta \cdot \left(d \log n + \frac{\log d \cdot \log^2 n}{\log\log n}\right)\right)$	$O(n+m)$	$2(1+\varepsilon)d+1$

Parallel graph coloring heuristics

```
/* n: number of vertices,
   m: number of edges,
   Δ: maximum vertex degree,
   d: graph's degeneracy */
```

Ordering	Depth	Work	Quality
"First fit" (i.e., any order)	No general bounds; $\Omega(n)$ for some graphs	$O(n+m)$	$\Delta+1$
"Largest degree first"	No general bounds; $\Omega(\Delta^2)$ for some graphs	$O(n+m)$	$\Delta+1$
"Smallest degree last"	No general bounds; $\Omega(n)$ for some graphs	$O(n+m)$	$d+1$
Random	$\mathbb{E}\, O\left(\frac{\log n}{\log\log n}\right)$	$O(n+m)$	$\Delta+1$
Random	$\mathbb{E}\, O\left(\log n + \log\Delta \cdot \min\left\{\sqrt{m}, \Delta + \frac{\log\Delta\log n}{\log\log n}\right\}\right)$	$O(n+m)$	$\Delta+1$
"Largest log-degree first"	$\mathbb{E}\, O\left(\log n + \log\Delta\cdot\left(\min\{\Delta,\sqrt{m}\} + \frac{\log^2\Delta\log n}{\log\log n}\right)\right)$	$O(n+m)$	$\Delta+1$
"Smallest log-degree last"	$\mathbb{E}\, O\left(\log\Delta\log n + \log\Delta\cdot\left(\min\{\Delta,\sqrt{m}\} + \frac{\log^2\Delta\log n}{\log\log n}\right)\right)$	$O(n+m)$	$\Delta+1$
ADG	$\mathbb{E}\, O\left(\log^2 n + \log\Delta\cdot\left(d\log n + \frac{\log d\cdot\log^2 n}{\log\log n}\right)\right)$	$O(n+m)$	$2(1+\varepsilon)d+1$

Parallel graph coloring heuristics

```
/* n: number of vertices,
   m: number of edges,
   Δ: maximum vertex degree,
   d: graph’s degeneracy */
```

Anything else?

Ordering	**Depth**	**Work**	**Quality**
“First fit” (i.e., any order)	No general bounds; $\Omega(n)$ for some graphs	$O(n+m)$	$\Delta+1$
“Largest degree first”	No general bounds; $\Omega(\Delta^2)$ for some graphs	$O(n+m)$	$\Delta+1$
“Smallest degree last”	No general bounds; $\Omega(n)$ for some graphs	$O(n+m)$	$d+1$
Random	$\mathbb{E}\, O\left(\frac{\log n}{\log\log n}\right)$	$O(n+m)$	$\Delta+1$
Random	$\mathbb{E}\, O\left(\log n + \log\Delta \cdot \min\left\{\sqrt{m}, \Delta + \frac{\log\Delta \log n}{\log\log n}\right\}\right)$	$O(n+m)$	$\Delta+1$
“Largest log-degree first”	$\mathbb{E}\, O\left(\log n + \log\Delta \cdot \left(\min\{\Delta, \sqrt{m}\} + \frac{\log^2\Delta \log n}{\log\log n}\right)\right)$	$O(n+m)$	$\Delta+1$
“Smallest log-degree last”	$\mathbb{E}\, O\left(\log\Delta \log n + \log\Delta \cdot \left(\min\{\Delta, \sqrt{m}\} + \frac{\log^2\Delta \log n}{\log\log n}\right)\right)$	$O(n+m)$	$\Delta+1$
ADG ✓	$\mathbb{E}\, O\left(\log^2 n + \log\Delta \cdot \left(d \log n + \frac{\log d \cdot \log^2 n}{\log\log n}\right)\right)$	✓ $O(n+m)$	$2(1+\varepsilon)d+1$

Parallel graph coloring heuristics

Based on „speculative coloring”

Parallel graph coloring heuristics

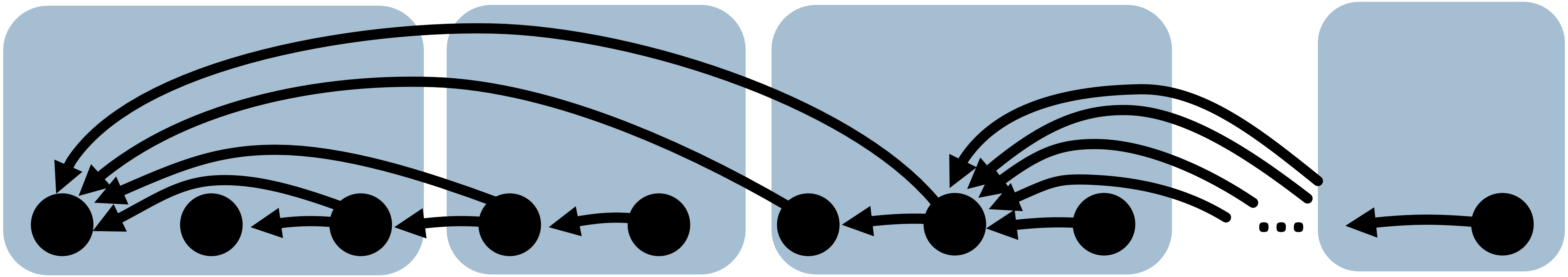
Based on „speculative coloring”

→ Construct the ADG-induced partitioning

Parallel graph coloring heuristics

Based on „speculative coloring"

→ Construct the ADG-induced partitioning

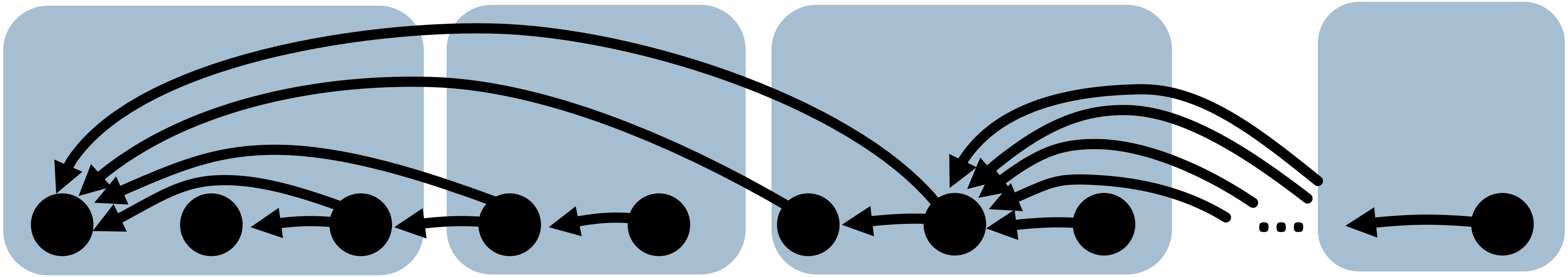


Parallel graph coloring heuristics

Based on „speculative coloring”

→ Construct the ADG-induced partitioning

→ Now, color each partition independently (“speculative coloring”)

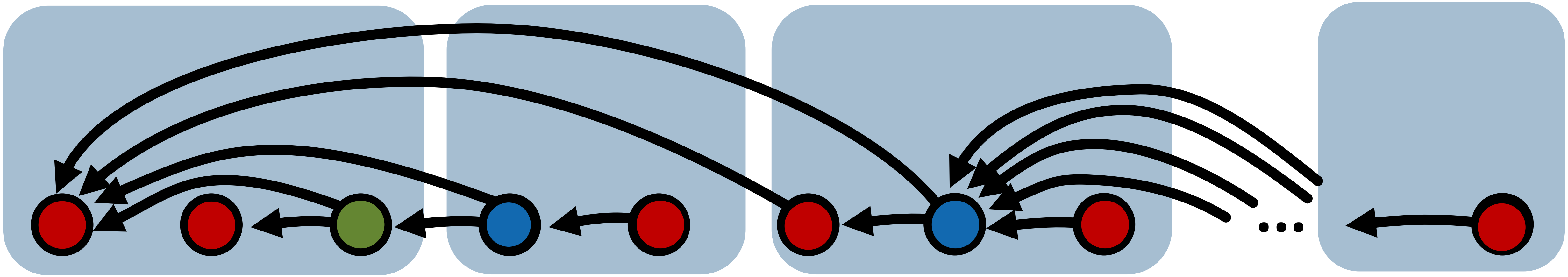


Parallel graph coloring heuristics

Based on „speculative coloring”

→ Construct the ADG-induced partitioning

→ Now, color each partition independently (“speculative coloring”)

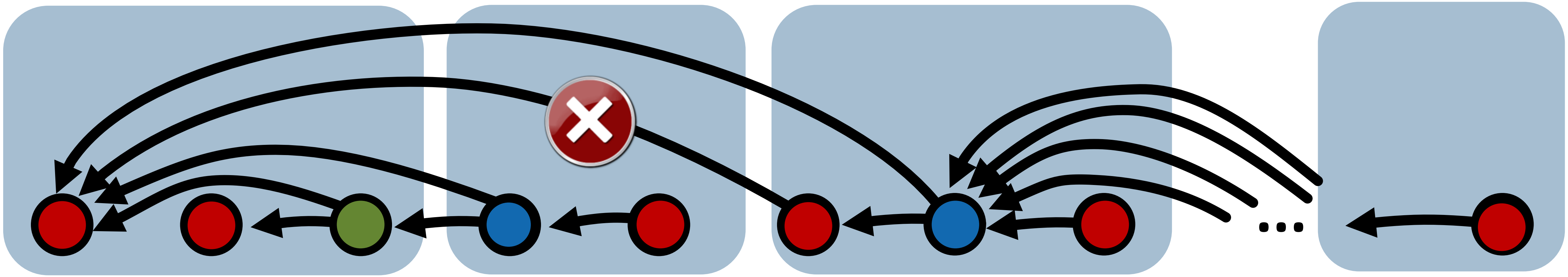


Parallel graph coloring heuristics

Based on „speculative coloring”

→ Construct the ADG-induced partitioning

→ Now, color each partition independently (“speculative coloring”)



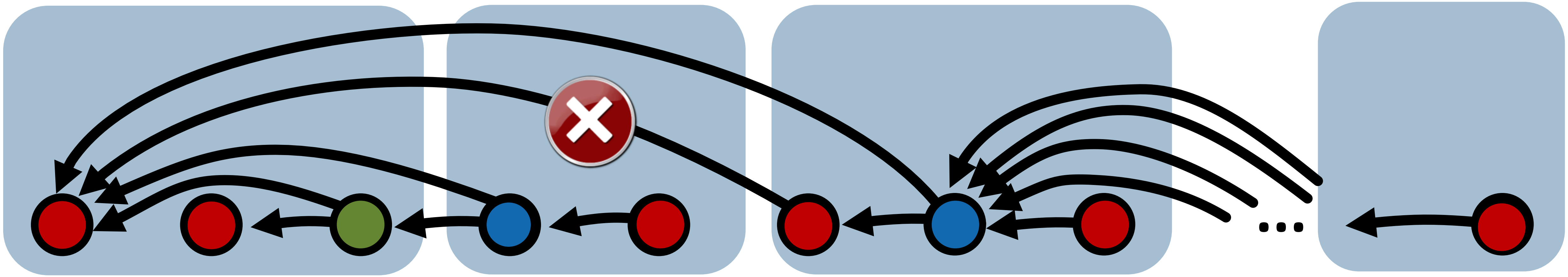
Parallel graph coloring heuristics

Based on „speculative coloring”

→ Construct the ADG-induced partitioning

→ Now, color each partition independently (“speculative coloring”)

→ Any coloring „conflicts” (vertices with the same colors) are by repeating the coloring on conflicting vertices as many times as needed



Parallel graph coloring heuristics

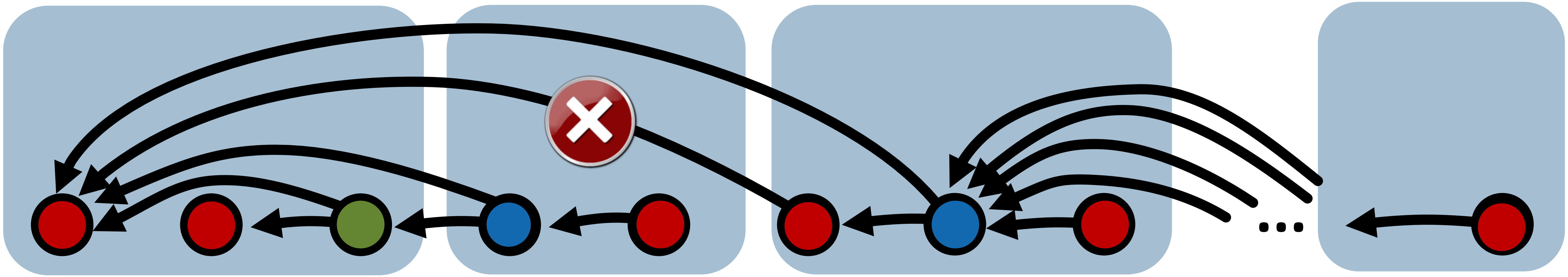
Based on „speculative coloring”

→ Construct the ADG-induced partitioning

→ Now, color each partition independently (“speculative coloring”)

→ Any coloring „conflicts” (vertices with the same colors) are by repeating the coloring on conflicting vertices as many times as needed

→ Each such partition is “low-degree”: it has a bounded number of edges to any other such partitions (by the definition of ADG)



Parallel graph coloring heuristics

Ordering	Depth	Work	Quality
“First fit” (i.e., any order)	No general bounds; $\Omega(n)$ for some graphs	$O(n+m)$	$\Delta+1$
“Largest degree first”	No general bounds; $\Omega(\Delta^2)$ for some graphs	$O(n+m)$	$\Delta+1$
“Smallest degree last”	No general bounds; $\Omega(n)$ for some graphs	$O(n+m)$	$d+1$
Random	$\mathbb{E}\ O\left(\frac{\log n}{\log\log n}\right)$	$O(n+m)$	$\Delta+1$
Random	$\mathbb{E}\ O\left(\log n+\log\Delta\cdot\min\left\{\sqrt{m},\Delta+\frac{\log\Delta\log n}{\log\log n}\right\}\right)$	$O(n+m)$	$\Delta+1$
“Largest log-degree first”	$\mathbb{E}\ O\left(\log n+\log\Delta\cdot\left(\min\{\Delta,\sqrt{m}\}+\frac{\log^2\Delta\log n}{\log\log n}\right)\right)$	$O(n+m)$	$\Delta+1$
“Smallest log-degree last”	$\mathbb{E}\ O\left(\log\Delta\log n+\log\Delta\cdot\left(\min\{\Delta,\sqrt{m}\}+\frac{\log^2\Delta\log n}{\log\log n}\right)\right)$	$O(n+m)$	$\Delta+1$
ADG (scheduling)	$\mathbb{E}\ O\left(\log^2 n+\log\Delta\cdot\left(d\log n+\frac{\log d\cdot\log^2 n}{\log\log n}\right)\right)$	$O(n+m)$	$2(1+\varepsilon)d+1$

Parallel graph coloring heuristics

Ordering	Depth	Work	Quality
"First fit" (i.e., any order)	No general bounds; $\Omega(n)$ for some graphs	$O(n+m)$	$\Delta+1$
"Largest degree first"	No general bounds; $\Omega(\Delta^2)$ for some graphs	$O(n+m)$	$\Delta+1$
"Smallest degree last"	No general bounds; $\Omega(n)$ for some graphs	$O(n+m)$	$d+1$
Random	$\mathbb{E}\, O\left(\frac{\log n}{\log\log n}\right)$	$O(n+m)$	$\Delta+1$
Random	$\mathbb{E}\, O\left(\log n + \log\Delta \cdot \min\left\{\sqrt{m}, \Delta + \frac{\log\Delta\log n}{\log\log n}\right\}\right)$	$O(n+m)$	$\Delta+1$
"Largest log-degree first"	$\mathbb{E}\, O\left(\log n + \log\Delta \cdot \left(\min\{\Delta, \sqrt{m}\} + \frac{\log^2\Delta\log n}{\log\log n}\right)\right)$	$O(n+m)$	$\Delta+1$
"Smallest log-degree last"	$\mathbb{E}\, O\left(\log\Delta\log n + \log\Delta \cdot \left(\min\{\Delta, \sqrt{m}\} + \frac{\log^2\Delta\log n}{\log\log n}\right)\right)$	$O(n+m)$	$\Delta+1$
ADG (scheduling)	$\mathbb{E}\, O\left(\log^2 n + \log\Delta \cdot \left(d\log n + \frac{\log d \cdot \log^2 n}{\log\log n}\right)\right)$	$O(n+m)$	$2(1+\varepsilon)d+1$
ADG (speculative)	$O\left(\log d \log^2 n\right)$ w.h.p.	$O(n+m)$ (w.h.p.)	$(2+\varepsilon)d$
ADG (speculative)	$O\left(I \cdot d\log n\right)$		$2(1+\varepsilon)d+1$

Parallel graph coloring heuristics

Ordering	Depth	Work	Quality
"First fit" (i.e., any order)	No general bounds; $\Omega(n)$ for some graphs	$O(n+m)$	$\Delta+1$
"Largest degree first"	No general bounds; $\Omega(\Delta^2)$ for some graphs	$O(n+m)$	$\Delta+1$
"Smallest degree last"	No general bounds; $\Omega(n)$ for some graphs	$O(n+m)$	$d+1$
Random	$\mathbb{E}\, O\left(\frac{\log n}{\log\log n}\right)$	$O(n+m)$	$\Delta+1$
Random	$\mathbb{E}\, O\left(\log n + \log\Delta \cdot \min\left\{\sqrt{m}, \Delta + \frac{\log\Delta\log n}{\log\log n}\right\}\right)$	$O(n+m)$	$\Delta+1$
"Largest log-degree first"	$\mathbb{E}\, O\left(\log n + \log\Delta \cdot \left(\min\{\Delta, \sqrt{m}\} + \frac{\log^2\Delta\log n}{\log\log n}\right)\right)$	$O(n+m)$	$\Delta+1$
"Smallest log-degree last"	$\mathbb{E}\, O\left(\log\Delta\log n + \log\Delta \cdot \left(\min\{\Delta, \sqrt{m}\} + \frac{\log^2\Delta\log n}{\log\log n}\right)\right)$	$O(n+m)$	$\Delta+1$
ADG (scheduling)	$\mathbb{E}\, O\left(\log^2 n + \log\Delta \cdot \left(d\log n + \frac{\log d \cdot \log^2 n}{\log\log n}\right)\right)$	$O(n+m)$	$2(1+\varepsilon)d+1$
ADG (speculative)	$O\left(\log d \log^2 n\right)$ w.h.p.	$O(n+m)$ (w.h.p.)	$(2+\varepsilon)d$
ADG (speculative)	$O\left(I \cdot d\log n\right)$		$2(1+\varepsilon)d+1$

ADG (speculative)

All details, proofs, etc., are in the paper ☺

Evaluation

Evaluation

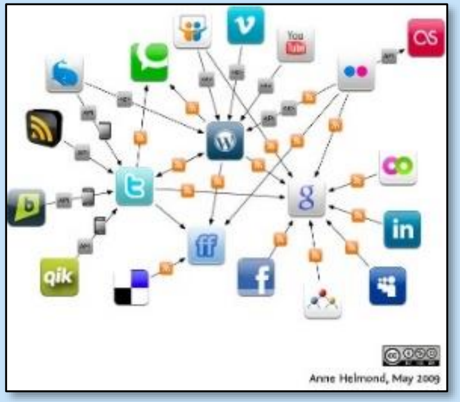
Graphs

Evaluation

Graphs

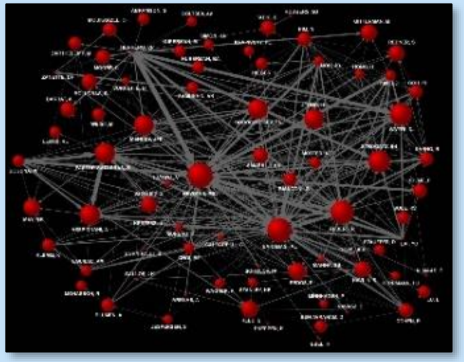
Road networks

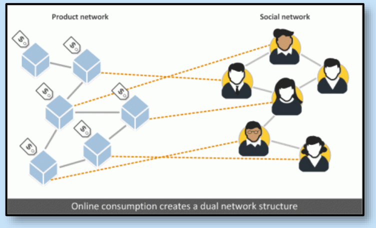
Social networks



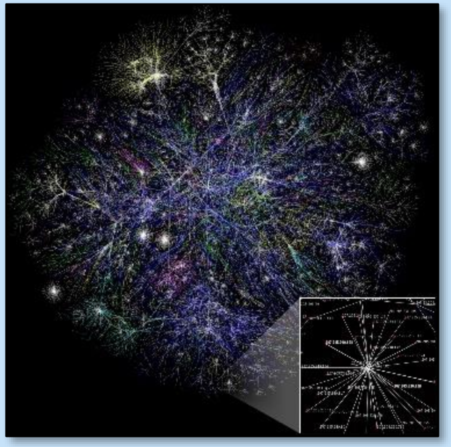
Communication graphs

Citation & collaboration graphs





Purchase networks



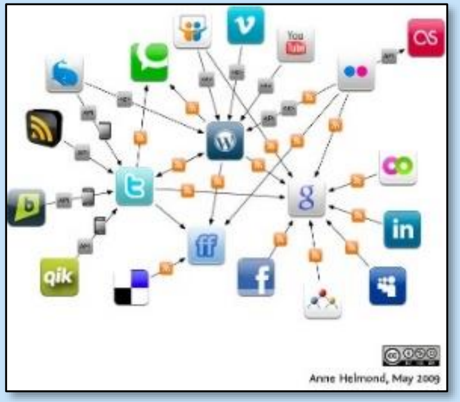
Web graphs

Evaluation

Graphs

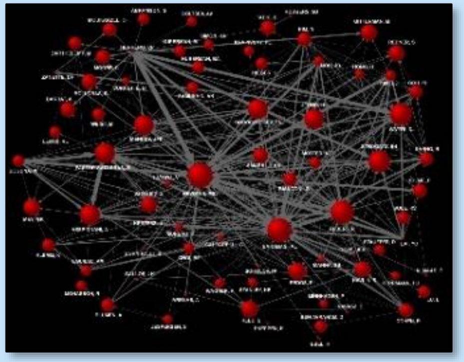
Road networks

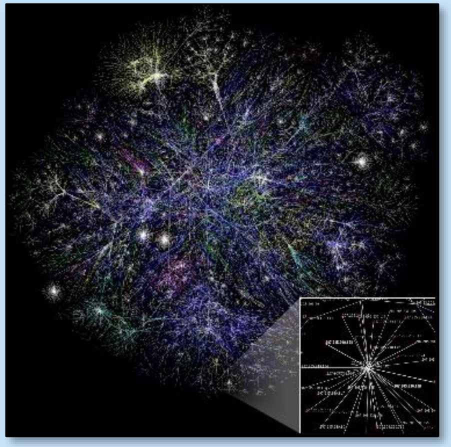
Social networks



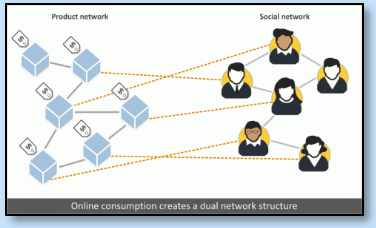
Communication graphs

Citation & collaboration graphs





Web graphs



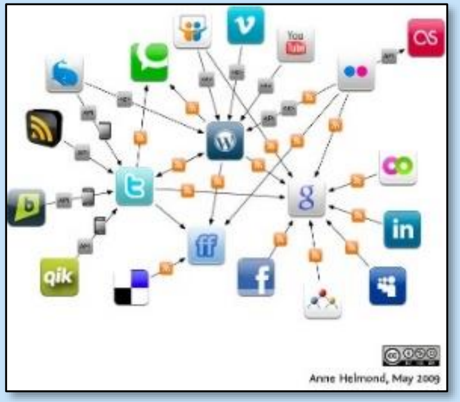
Purchase networks

Evaluation

Graphs

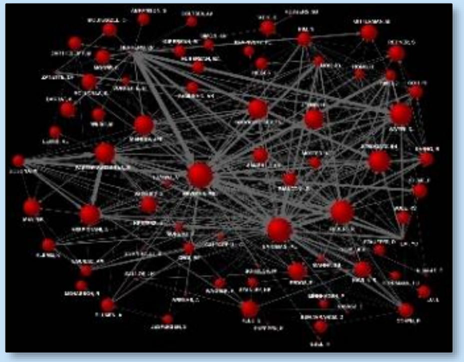
Road networks

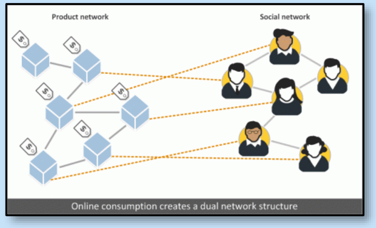
Social networks



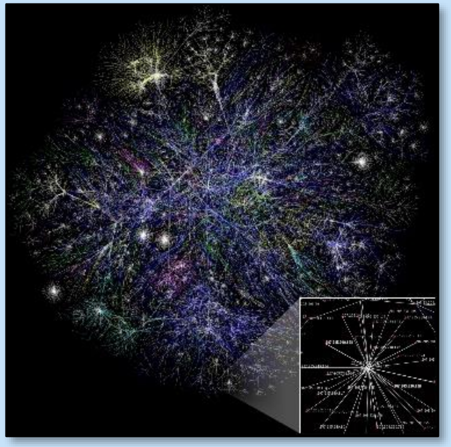
Communication graphs

Citation & collaboration graphs





Purchase networks



Web graphs

Machines

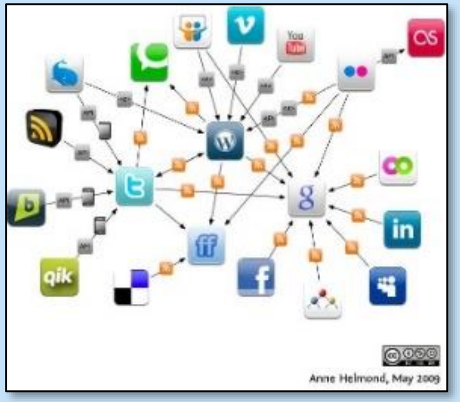
In-house Dell PowerEdge R910 server
(Intel Xeon X7550, 32 cores, 1TiB RAM)

Evaluation

Graphs

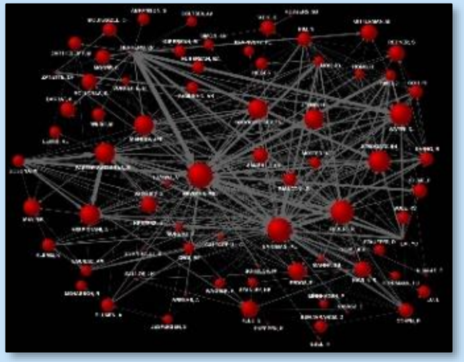
Road networks

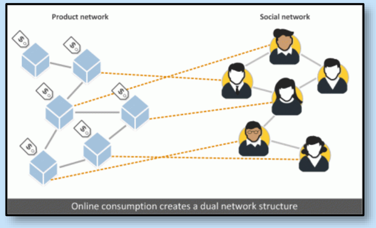
Social networks



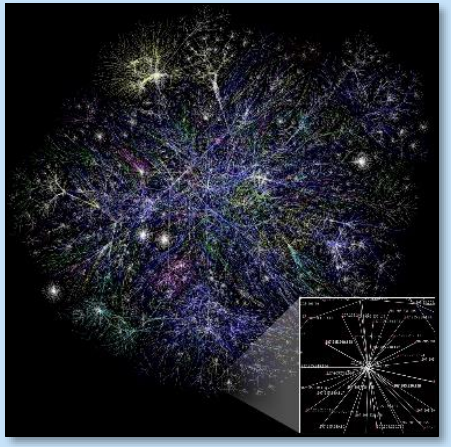
Communication graphs

Citation & collaboration graphs





Purchase networks



Web graphs

Machines

In-house Dell PowerEdge R910 server (Intel Xeon X7550, 32 cores, 1TiB RAM)

CSCS Ault, Intel Xeon Gold 6140, 18 cores, 768 GiB RAM

Evaluation

Evaluation

Comparison targets: 16 algorithms

Evaluation

Comparison targets: 16 algorithms

“First fit” (i.e., any order)

“Largest degree first”

“Smallest degree last”

Random

“Largest log-degree first”

“Smallest log-degree last”

(scheduling and speculative variants)

Evaluation

Comparison targets: 16 algorithms

“First fit” (i.e., any order)	Random
“Largest degree first”	“Largest log-degree first”
“Smallest degree last”	“Smallest log-degree last”

(scheduling and speculative variants)

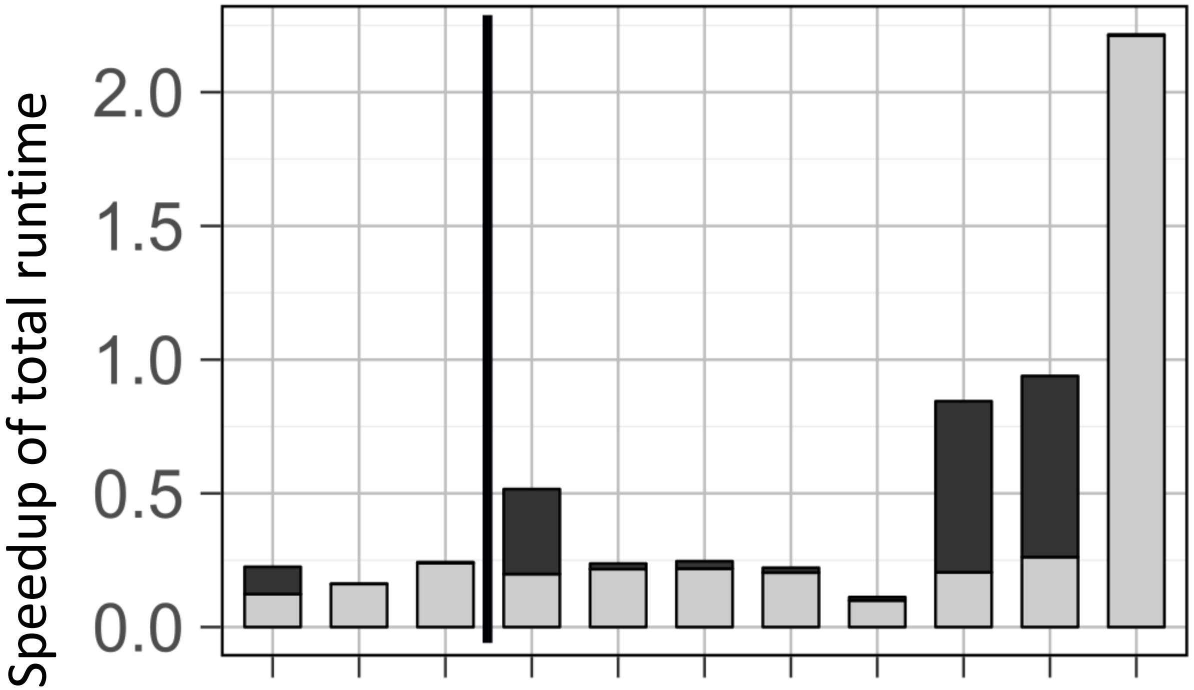
Taken from four libraries / codes

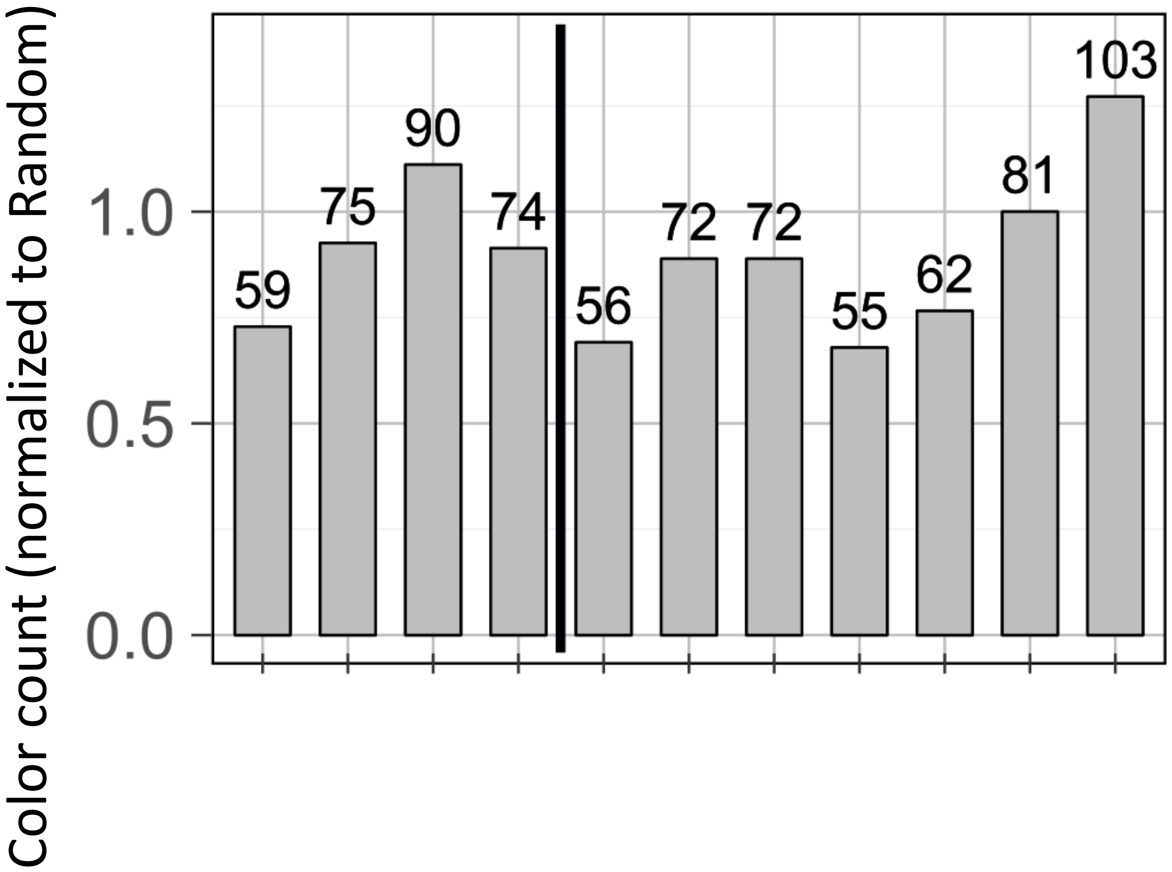
- Colpack [1]
- Zoltan [2]
- GBBS/Ligra [3]
- HP code [4]

[1] A. H. Gebremedhin, D. Nguyen, M. M. A. Patwary, and A. Pothen, “Colpack: Software for graph coloring and related problems in scientific computing”. TOMS’13.
[2] D. Bozdag, A. H. Gebremedhin, F. Manne, E. G. Boman, and U. V. ˘ Catalyurek, “A framework for scalable greedy coloring on distributedmemory parallel computers”. JPDC’08.
[3] L. Dhulipala, G. E. Blelloch, and J. Shun, “Theoretically efficient parallel graph algorithms can be fast and scalable” . SPAA’18.
[4] W. Hasenplaugh, T. Kaler, T. B. Schardl, and C. E. Leiserson, “Ordering heuristics for parallel graph coloring”. SPAA’14.

Runtime & quality analysis

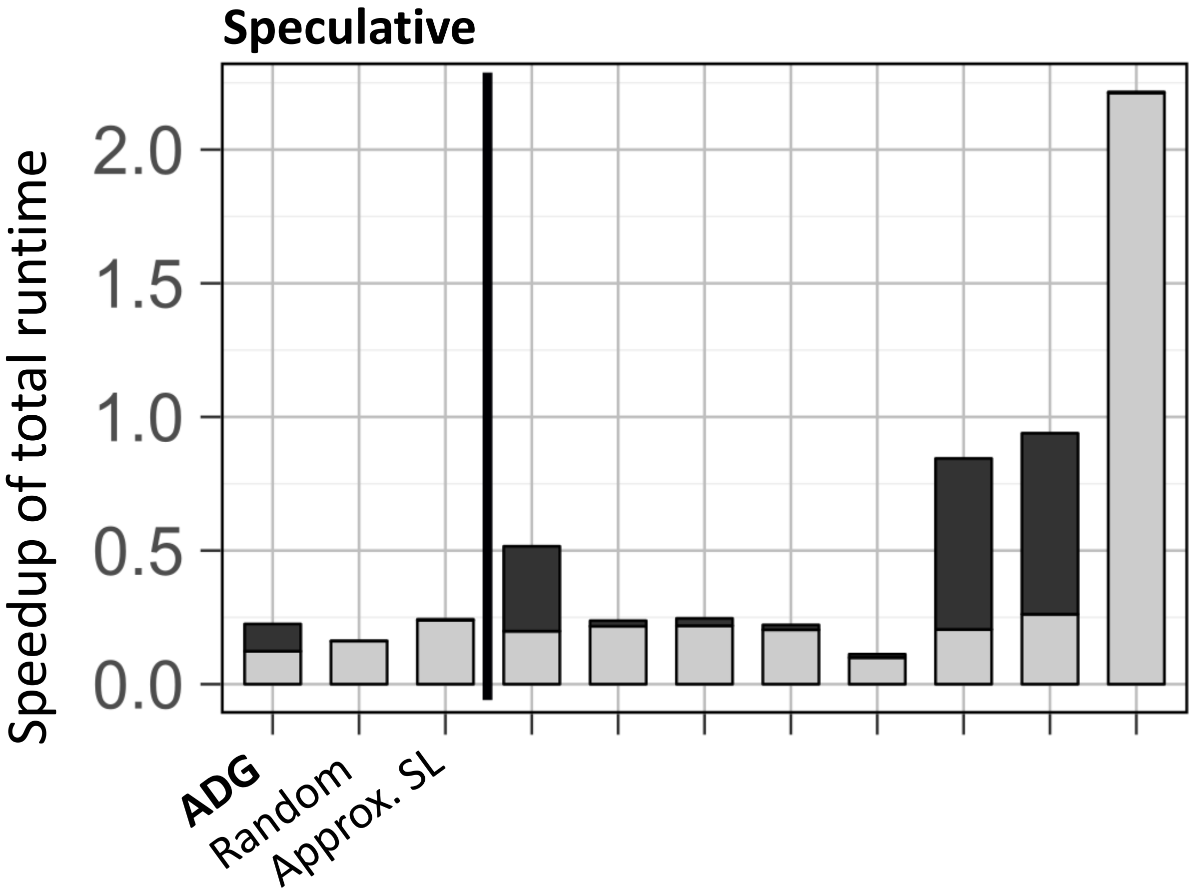
Smaller graphs; 5M edges (used in online settings)

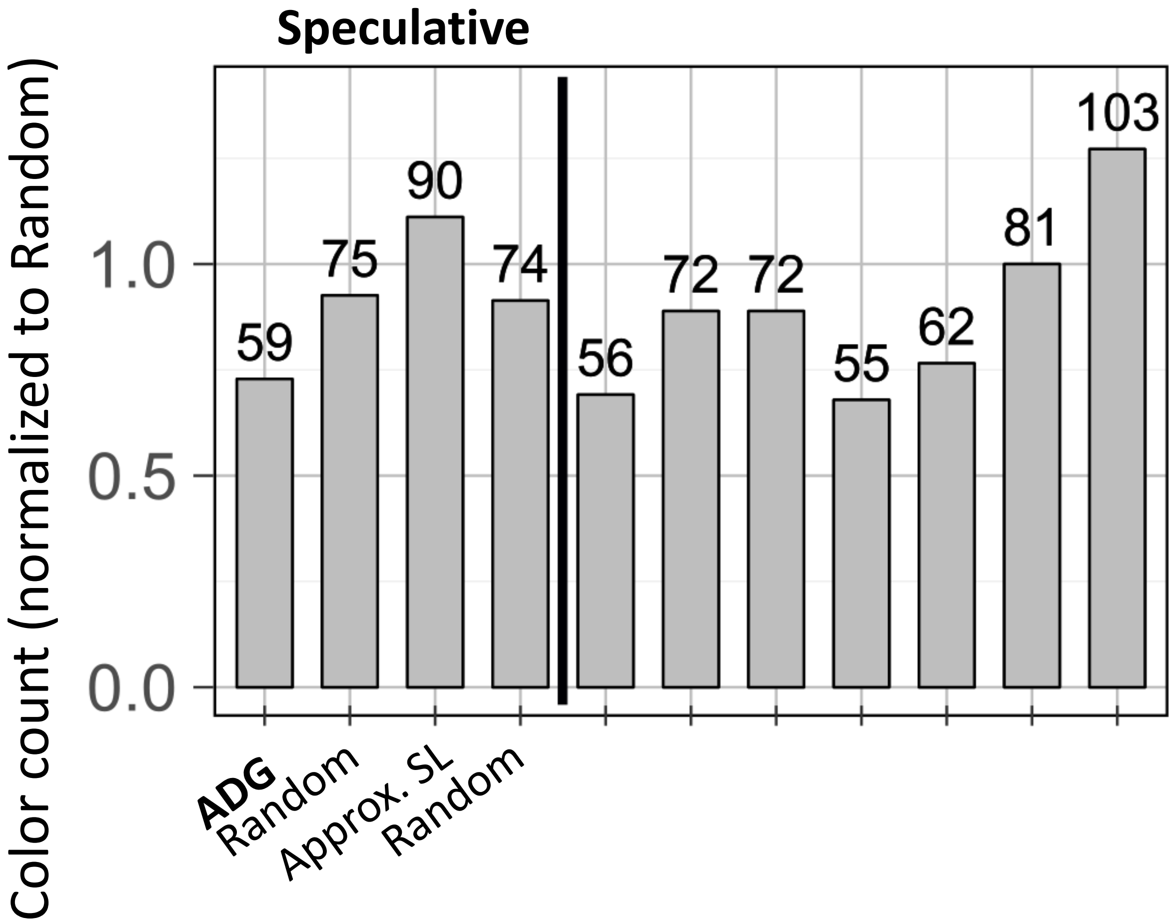




Runtime & quality analysis

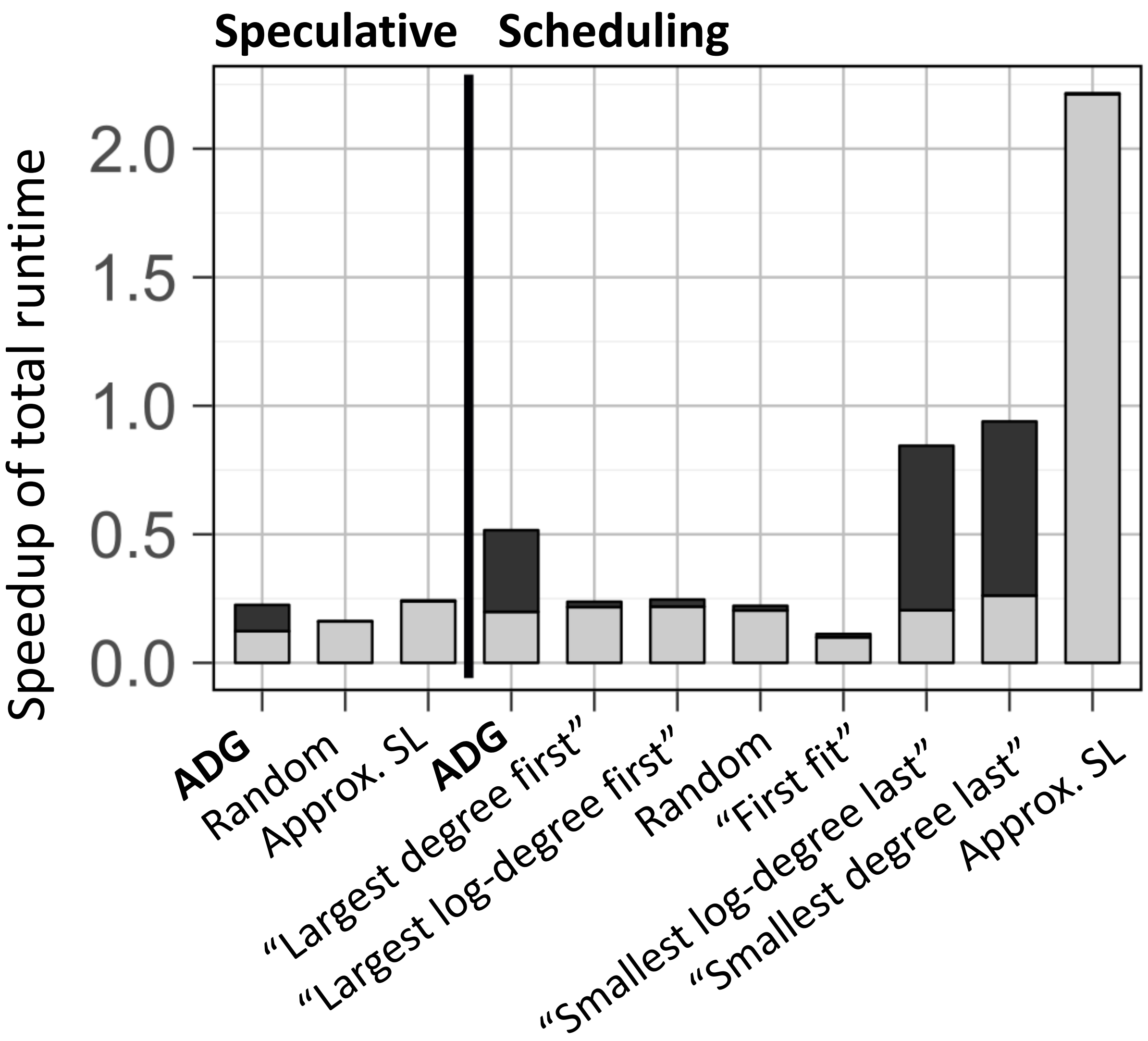
Smaller graphs; 5M edges (used in online settings)

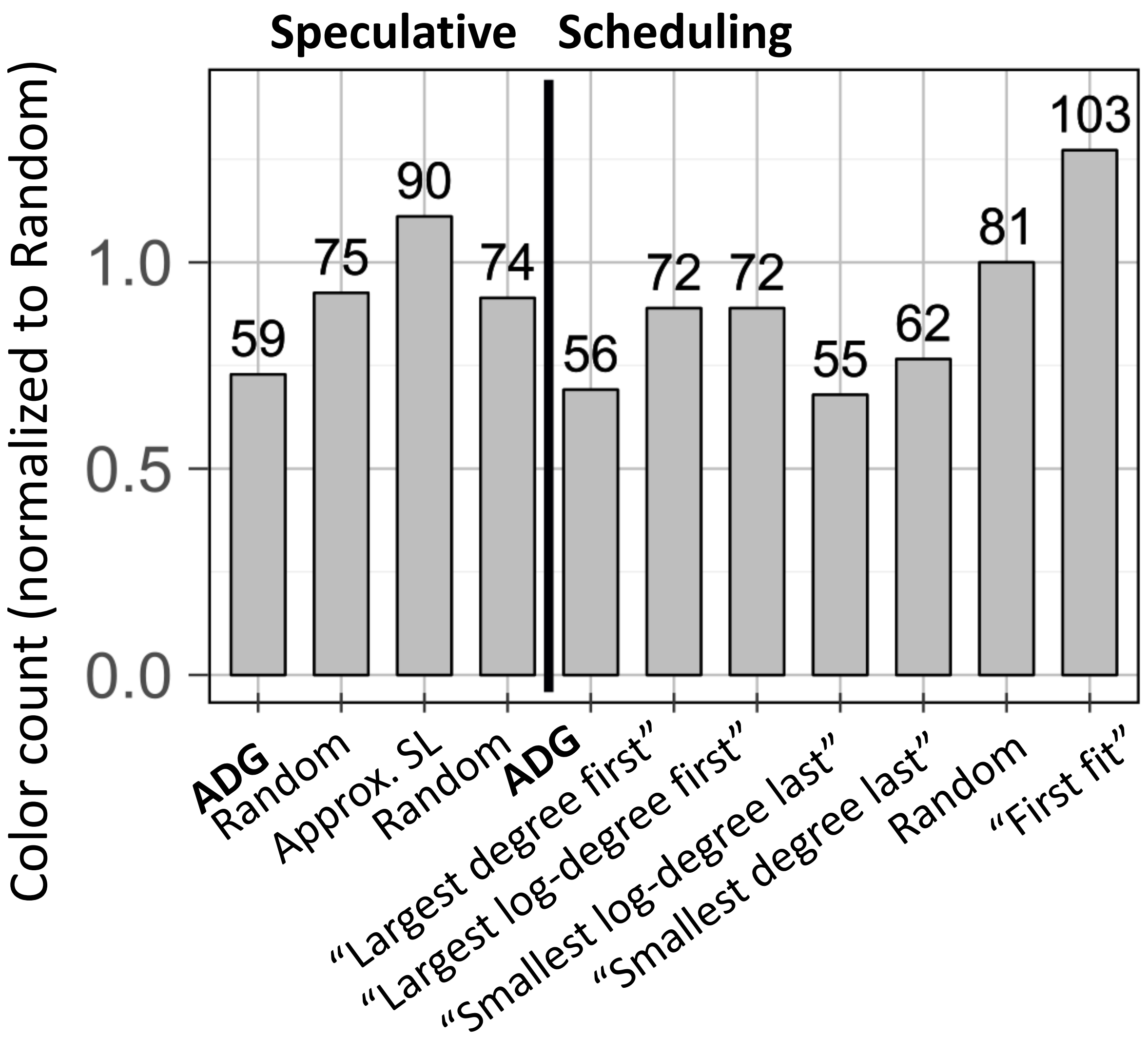

Speculative


Speculative

Runtime & quality analysis

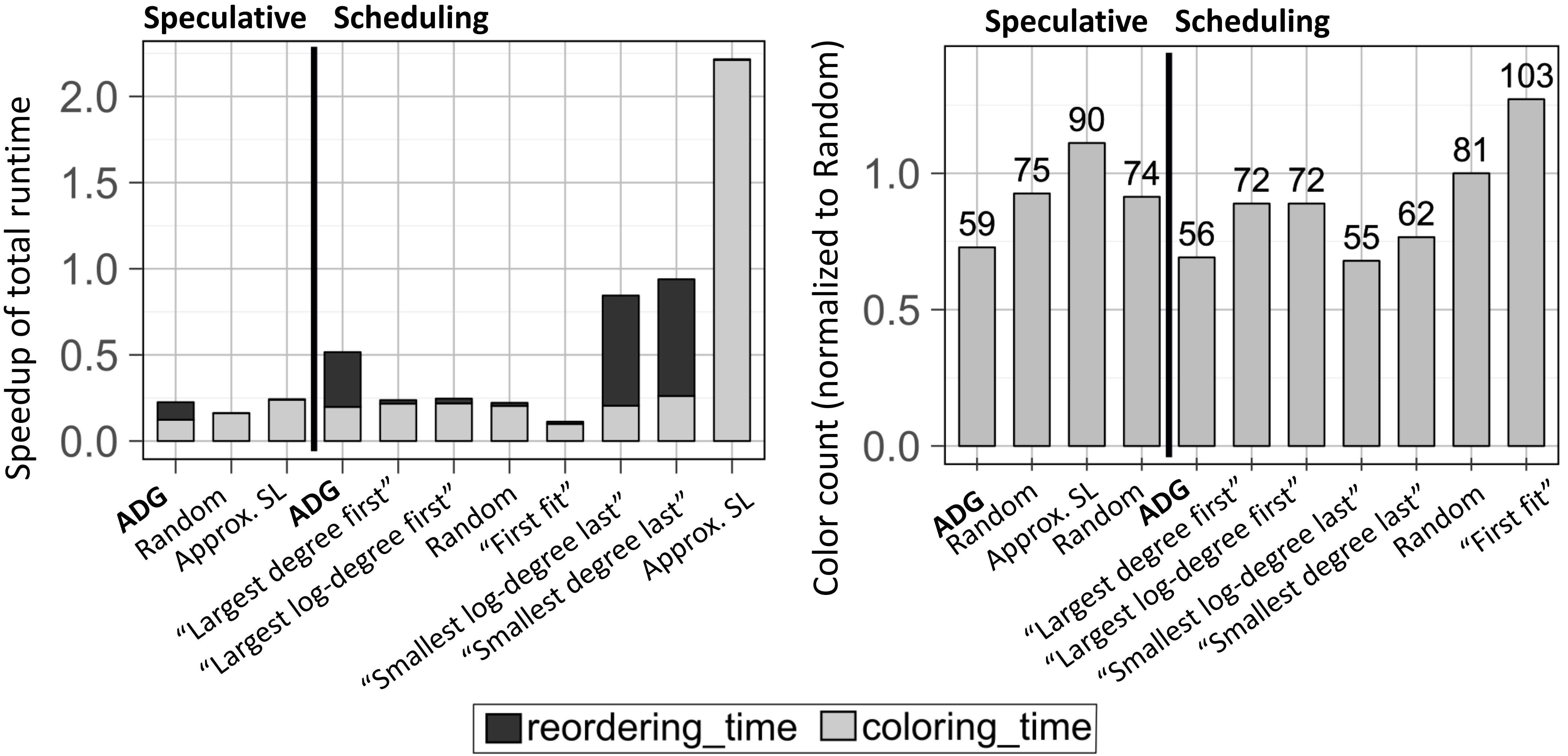
Smaller graphs; 5M edges (used in online settings)





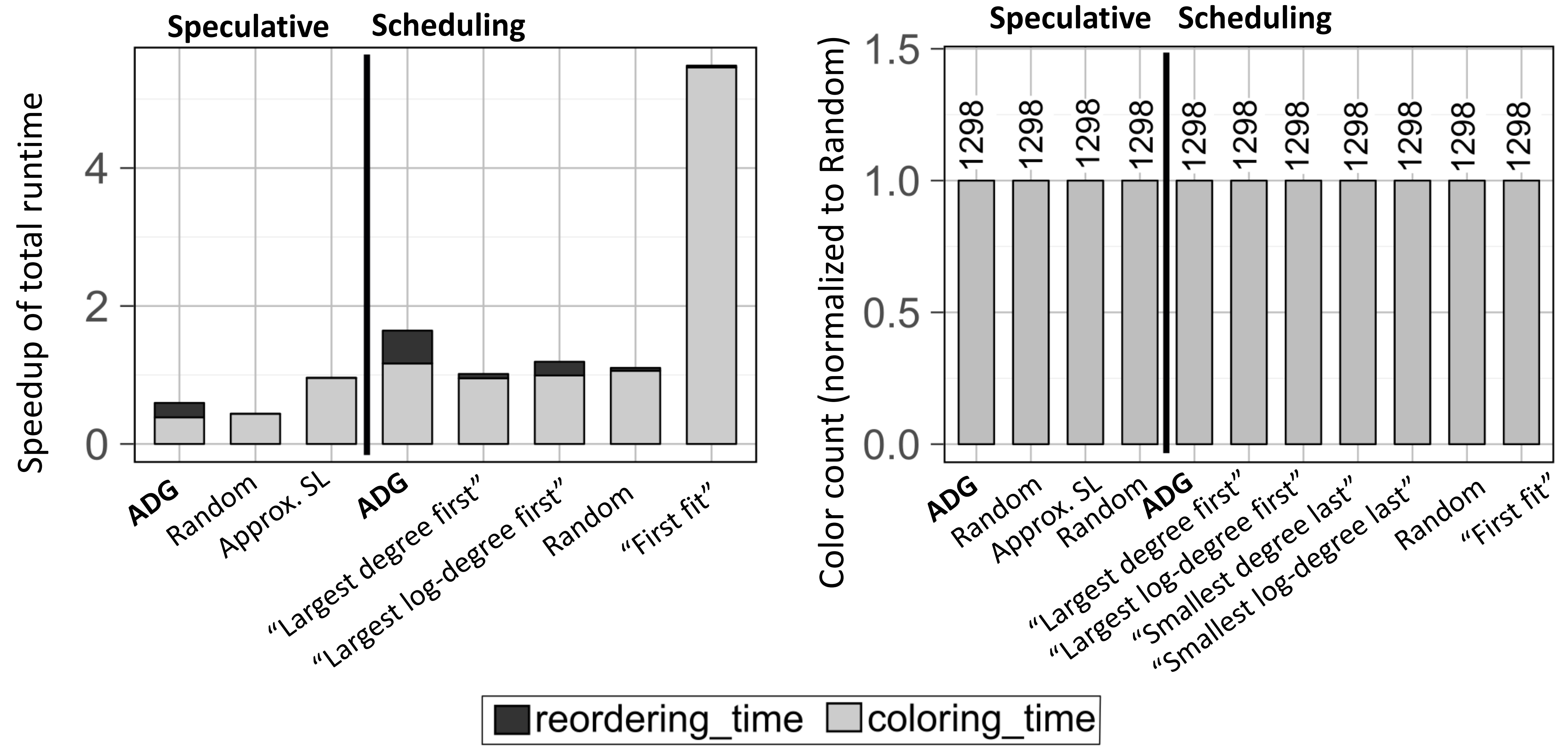
Runtime & quality analysis

Smaller graphs; 5M edges (used in online settings)



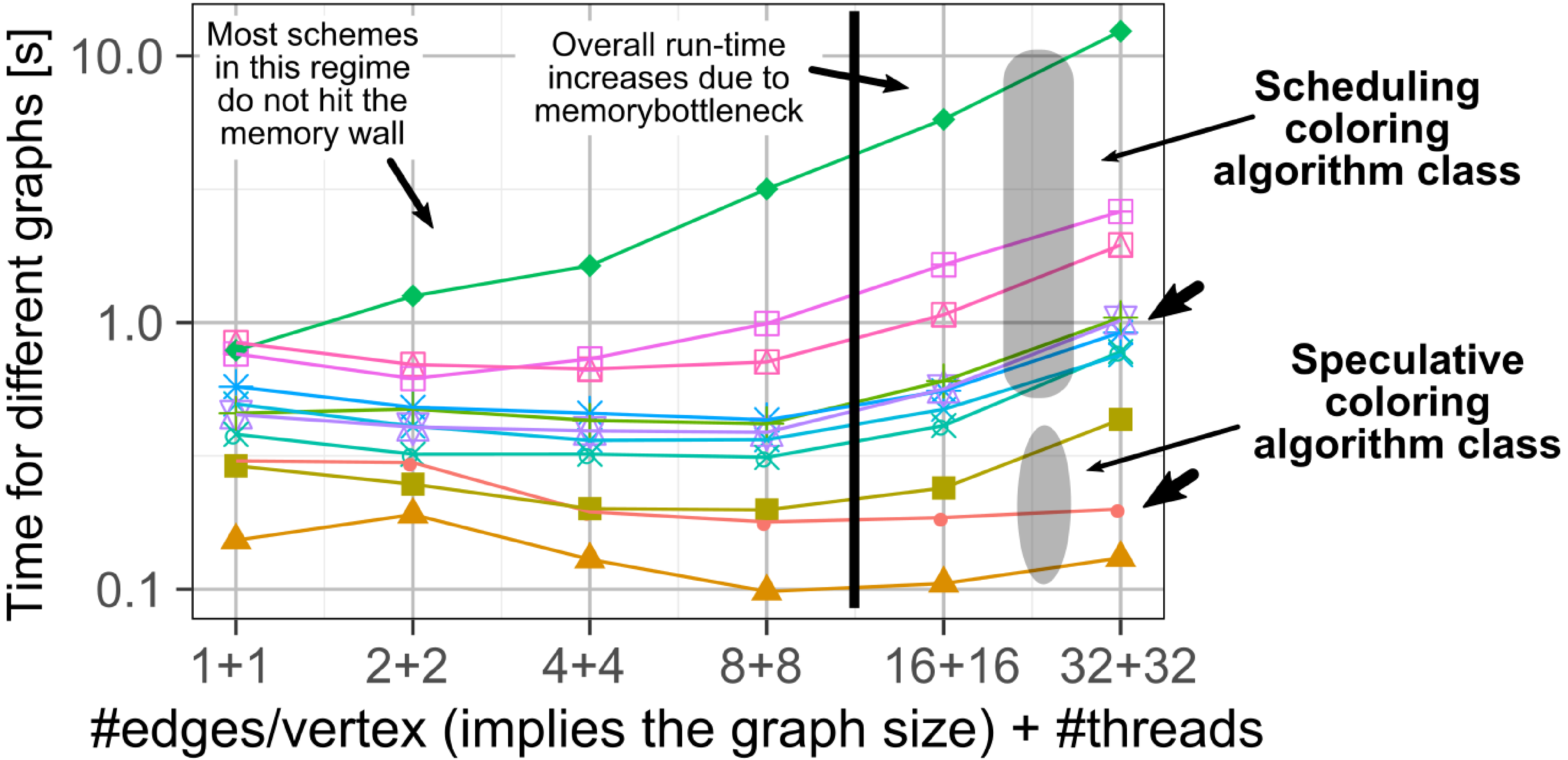
Runtime & quality analysis

Larger graphs, 230M edges (used in offline data analytics)



Scaling

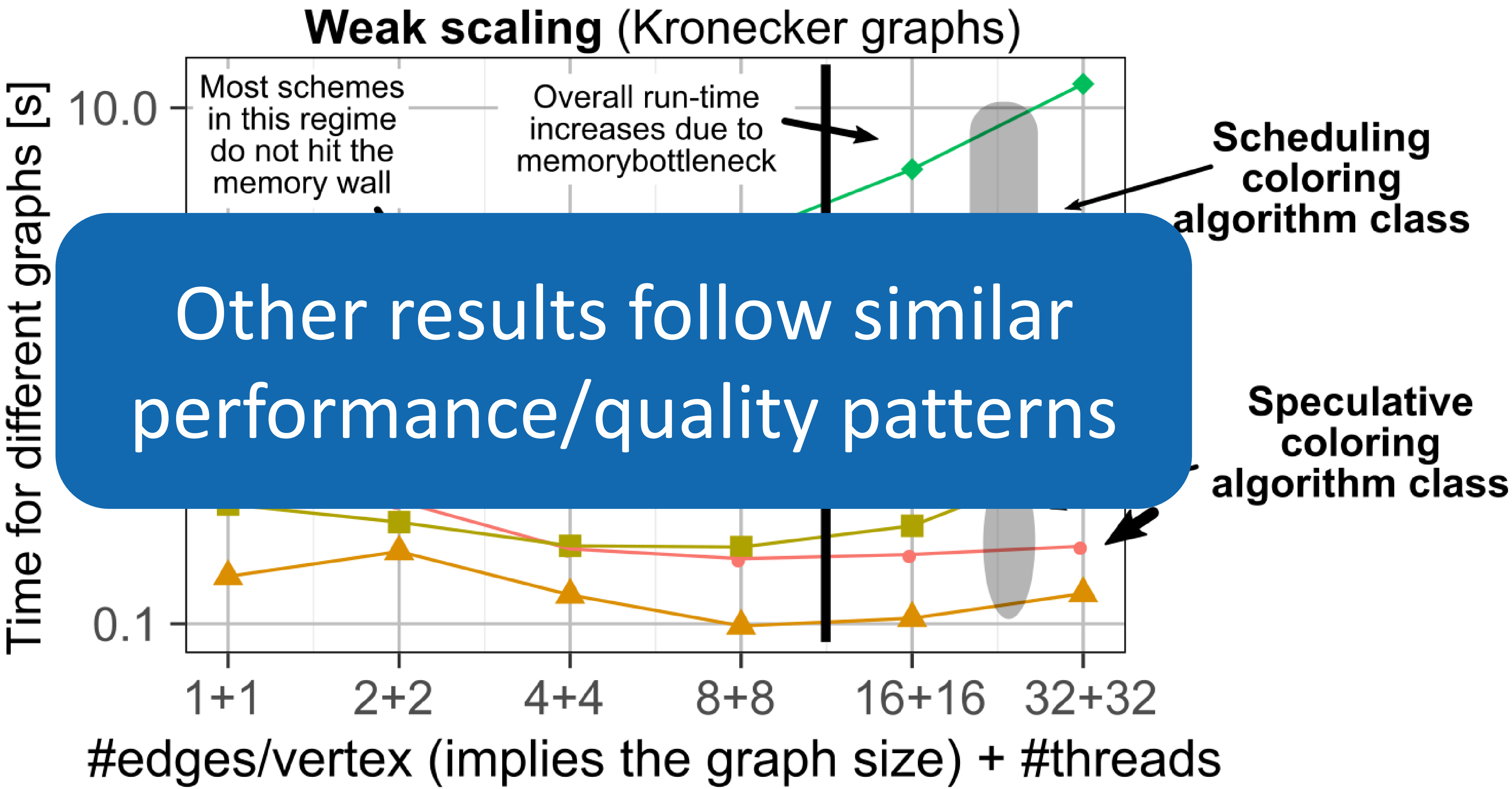
Weak scaling (Kronecker graphs)



Scheduling coloring algorithm class

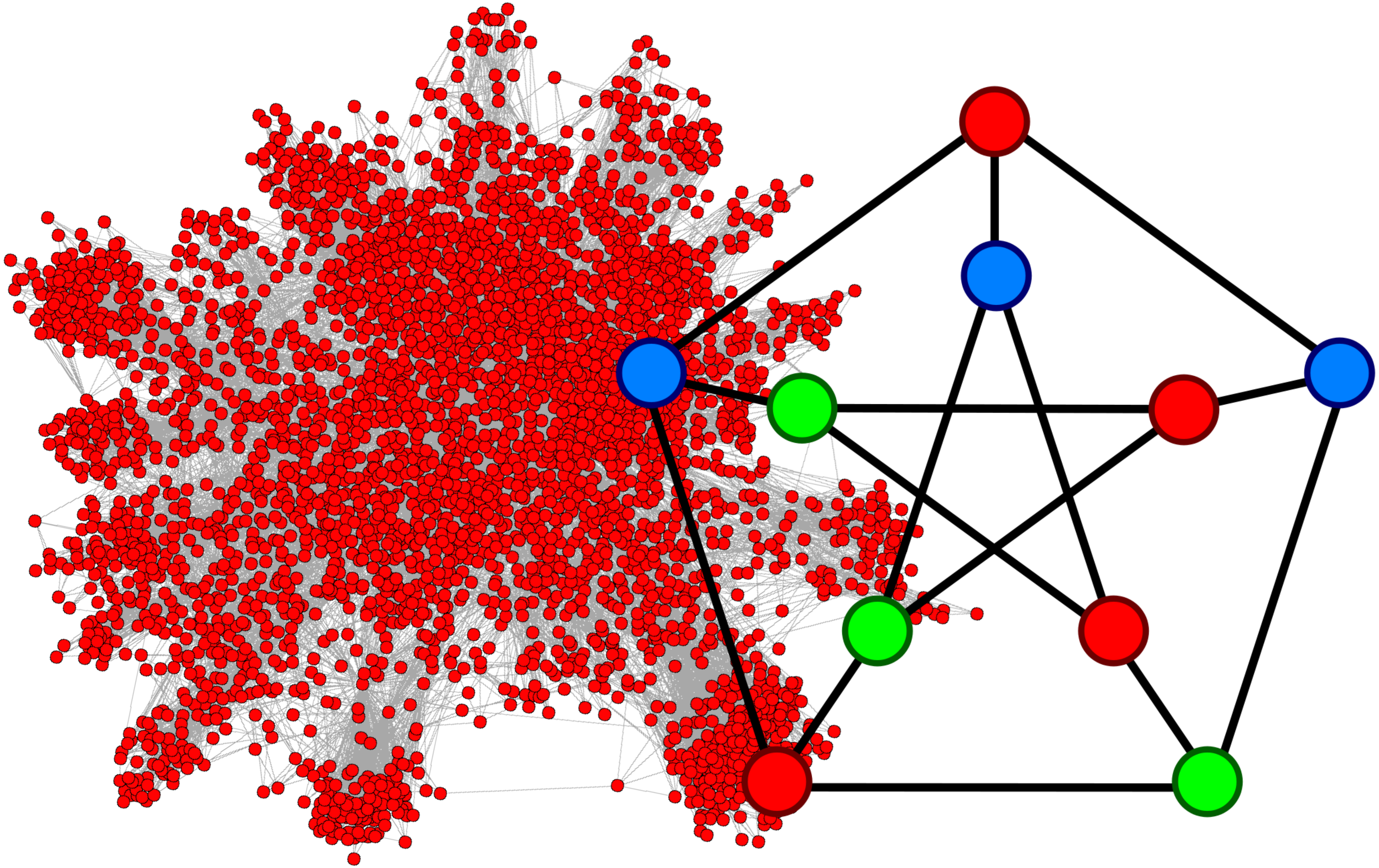
Speculative coloring algorithm class

Scaling



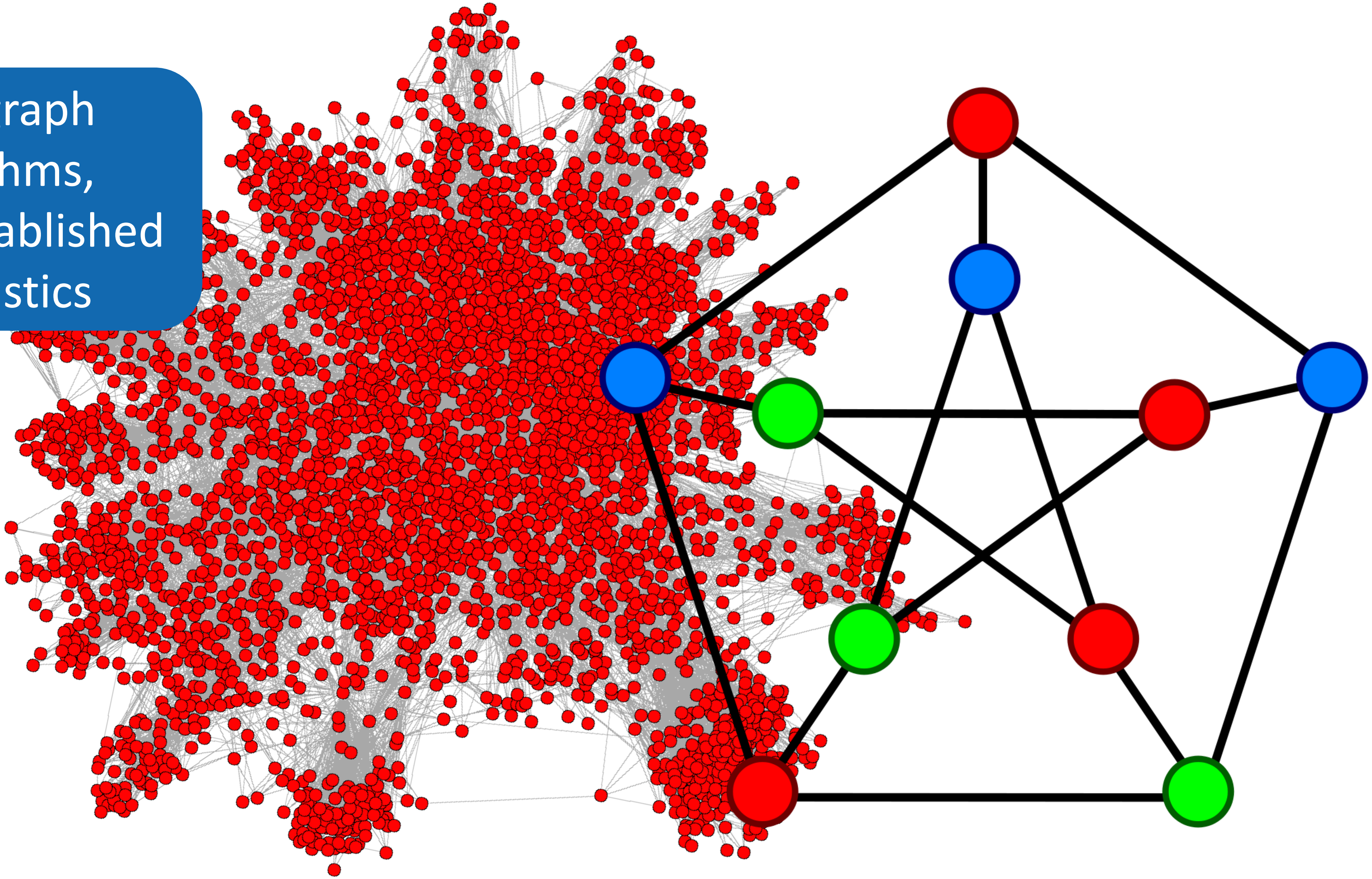
Other results follow similar performance/quality patterns

Conclusion

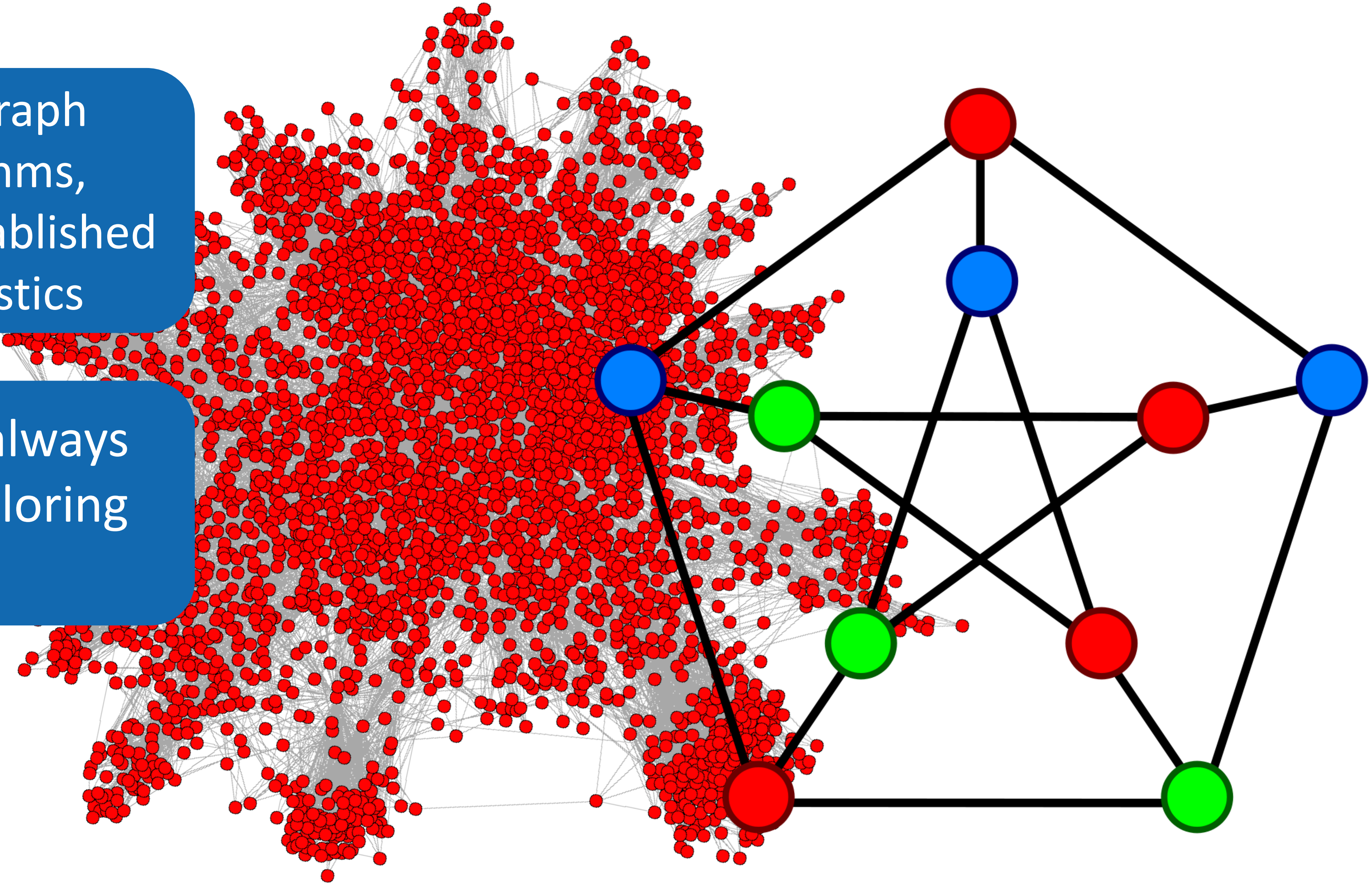


Conclusion

Novel parallel graph coloring algorithms, enhancing two established classes of heuristics



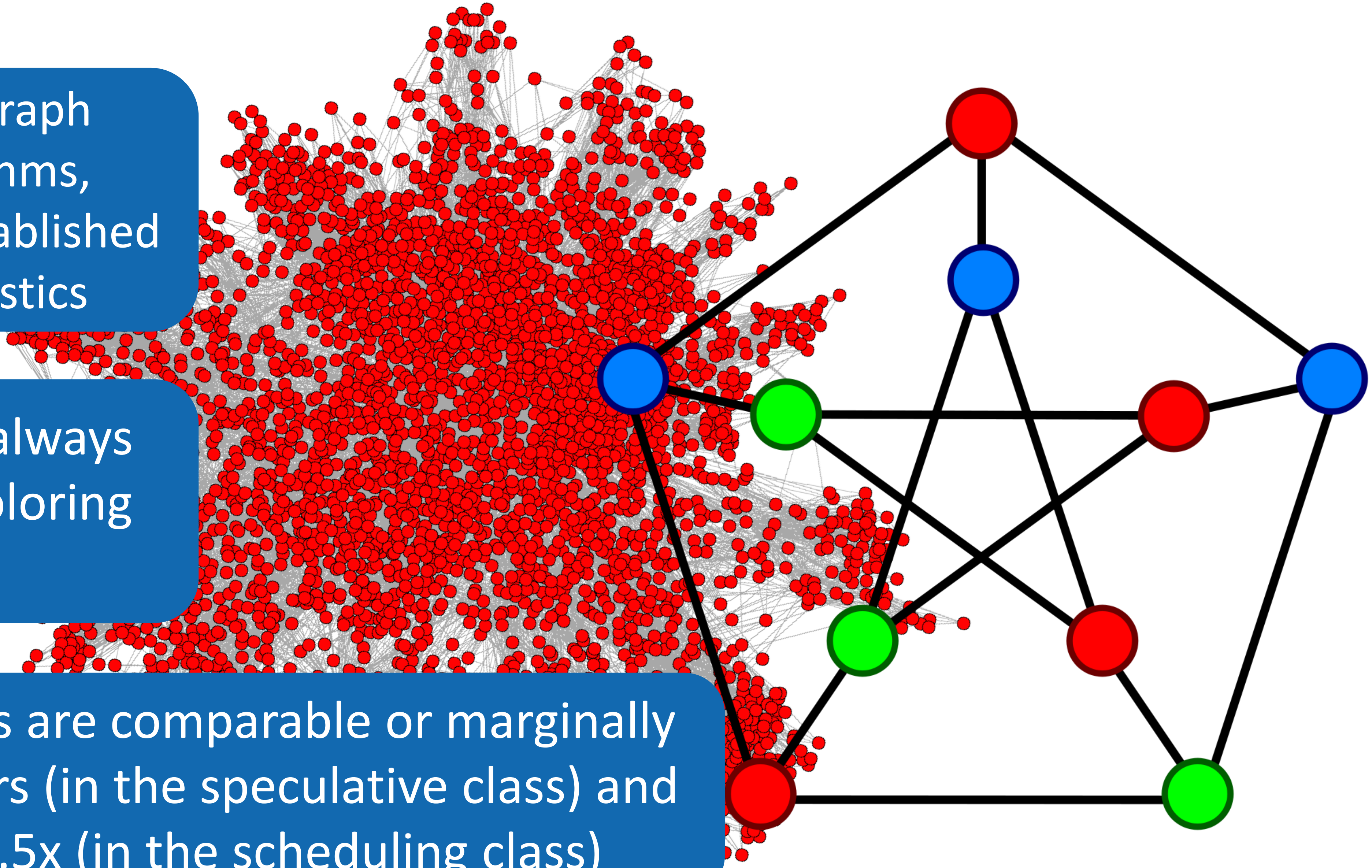
Conclusion



Novel parallel graph coloring algorithms, enhancing two established classes of heuristics

→ They almost always offer superior coloring quality

Conclusion

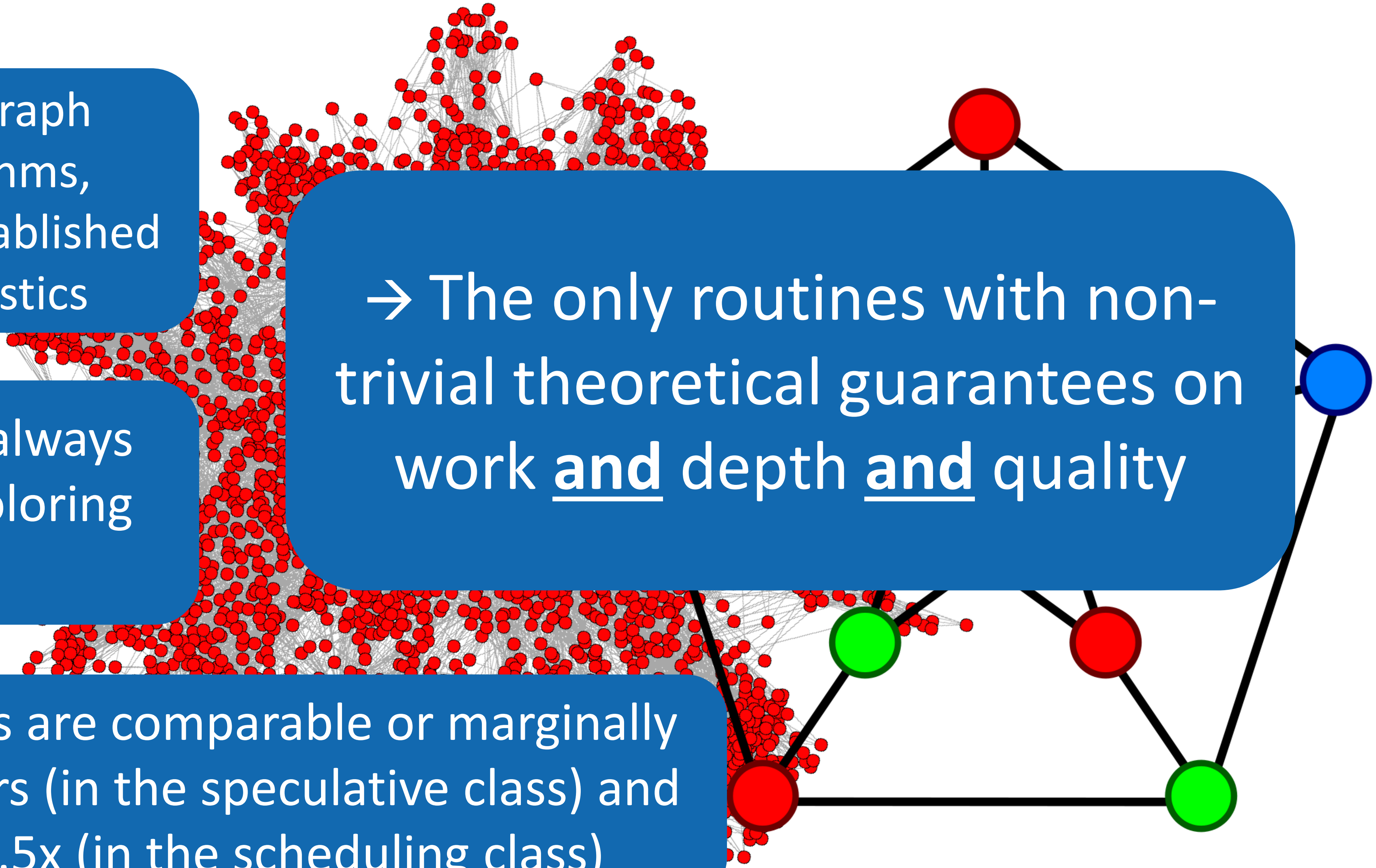


Novel parallel graph coloring algorithms, enhancing two established classes of heuristics

→ They almost always offer superior coloring quality

→ Their runtimes are comparable or marginally higher than others (in the speculative class) and within 1.1 – 1.5x (in the scheduling class)

Conclusion



Novel parallel graph coloring algorithms, enhancing two established classes of heuristics

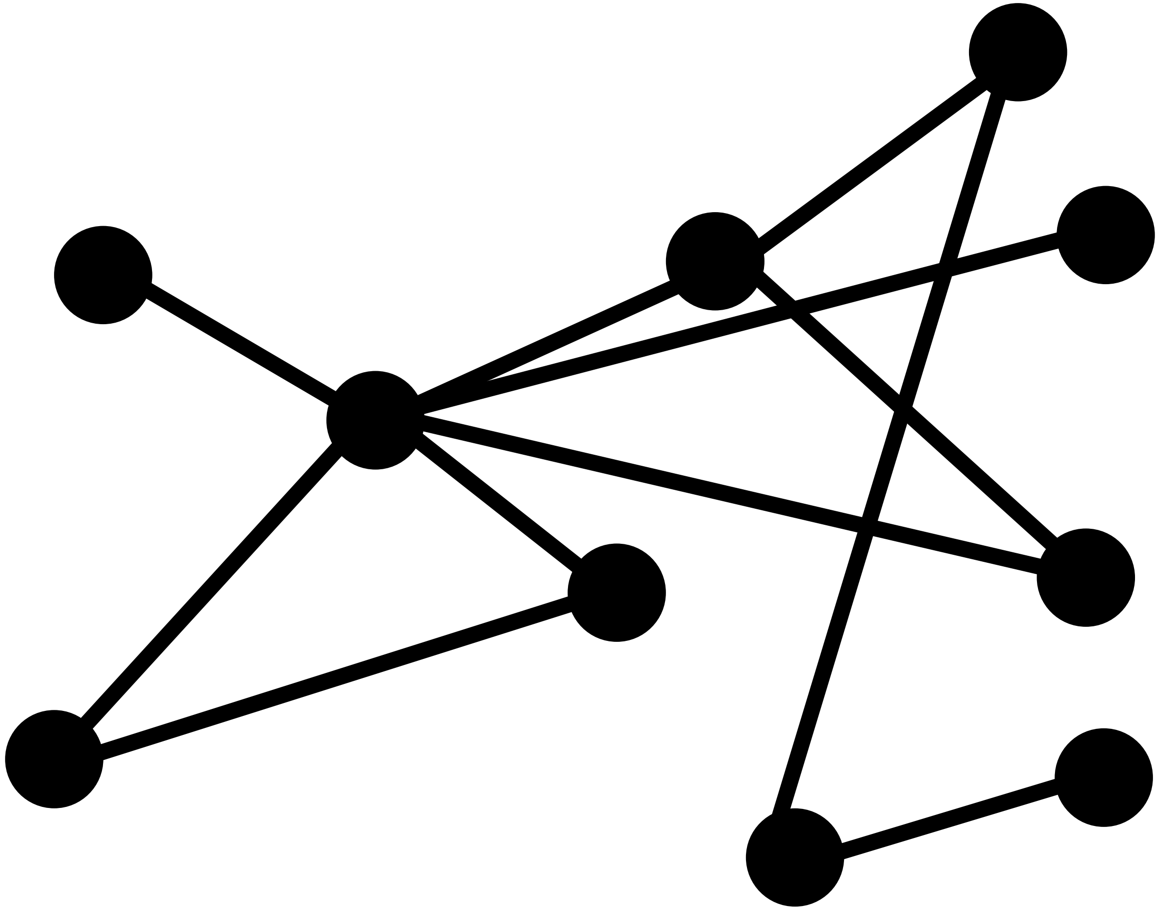
→ They almost always offer superior coloring quality

→ The only routines with non-trivial theoretical guarantees on work **and** depth **and** quality

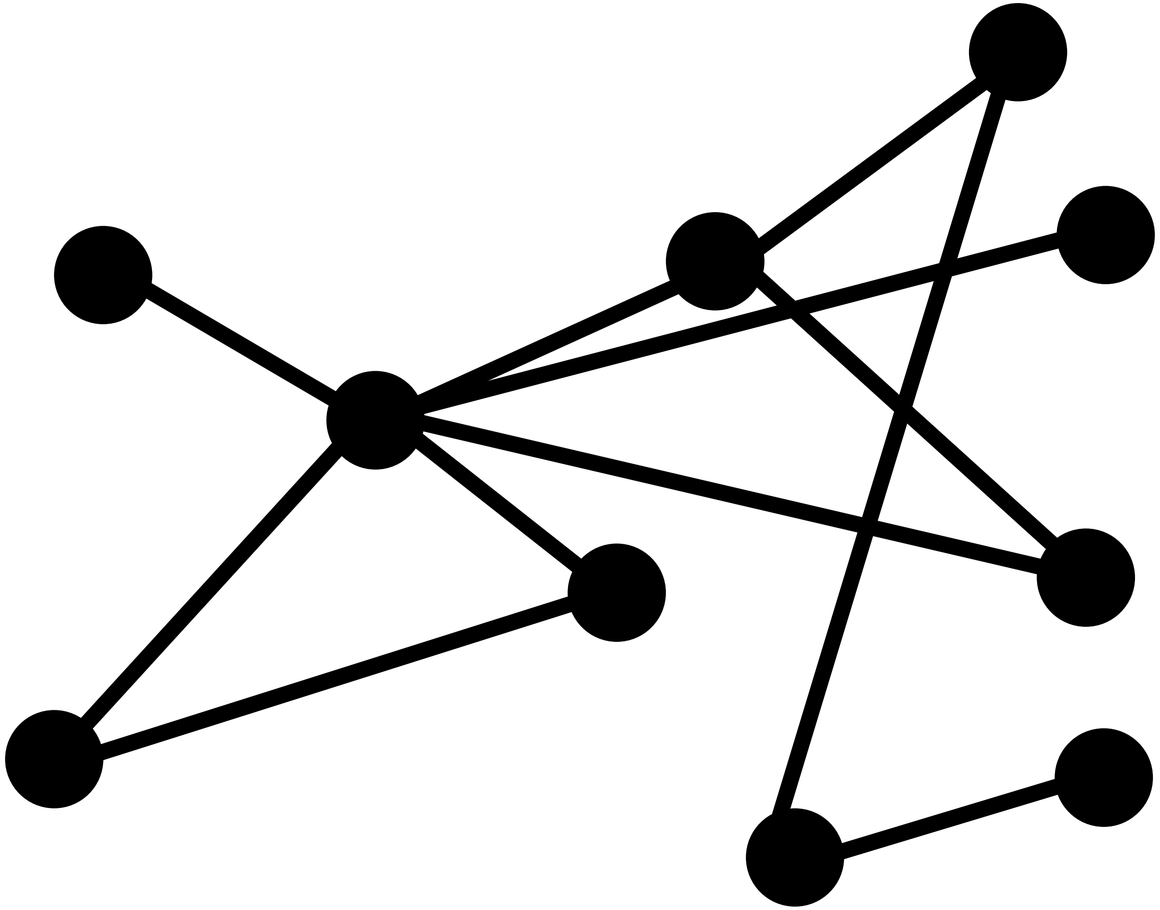
→ Their runtimes are comparable or marginally higher than others (in the speculative class) and within 1.1 – 1.5x (in the scheduling class)

Backup Slides and Slides’ Variants

Graph degeneracy: an example

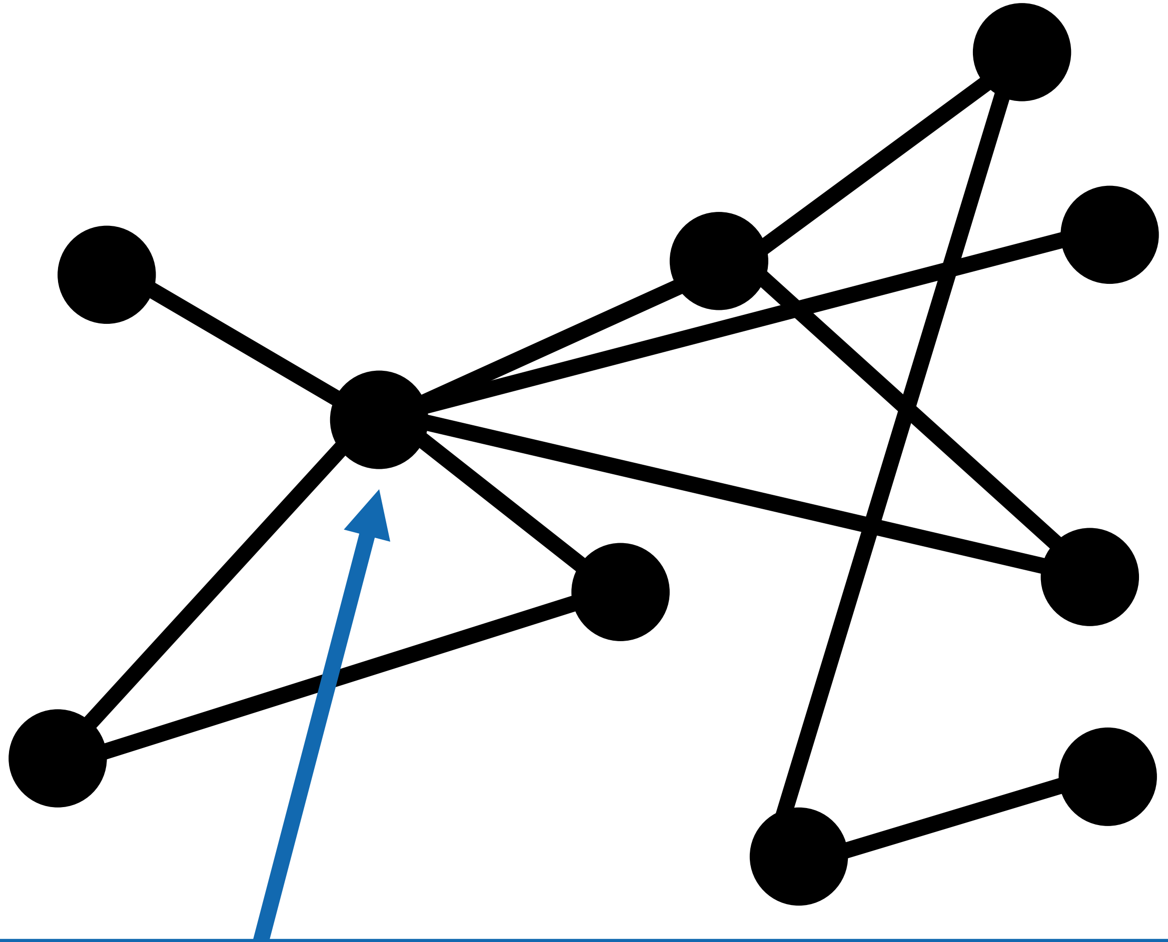


Graph degeneracy: an example



This graph has degeneracy of 3

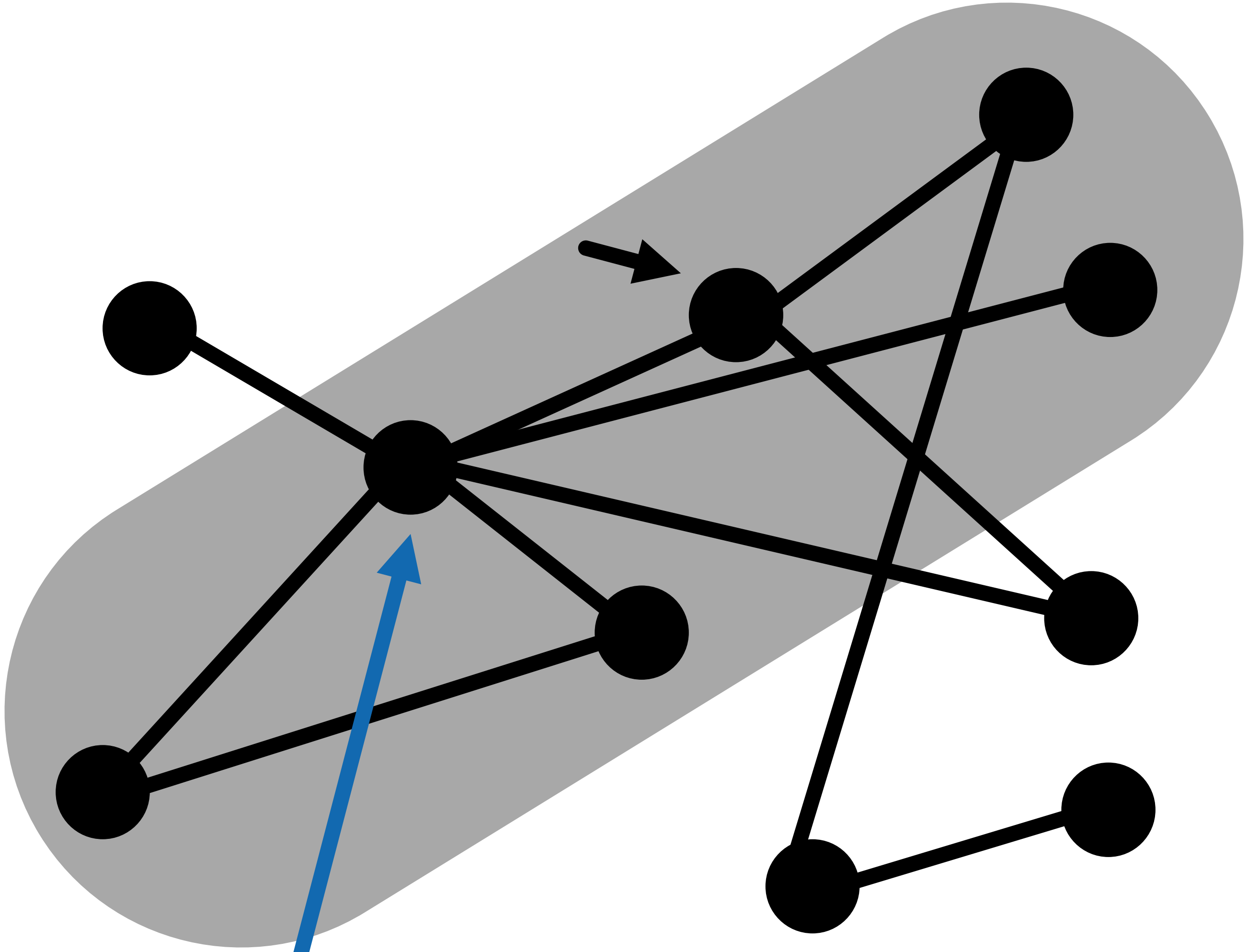
Graph degeneracy: an example



This graph has degeneracy of 3

Despite the high-degree vertex, each induced subgraph with this vertex contains some other vertex with degree at most 3

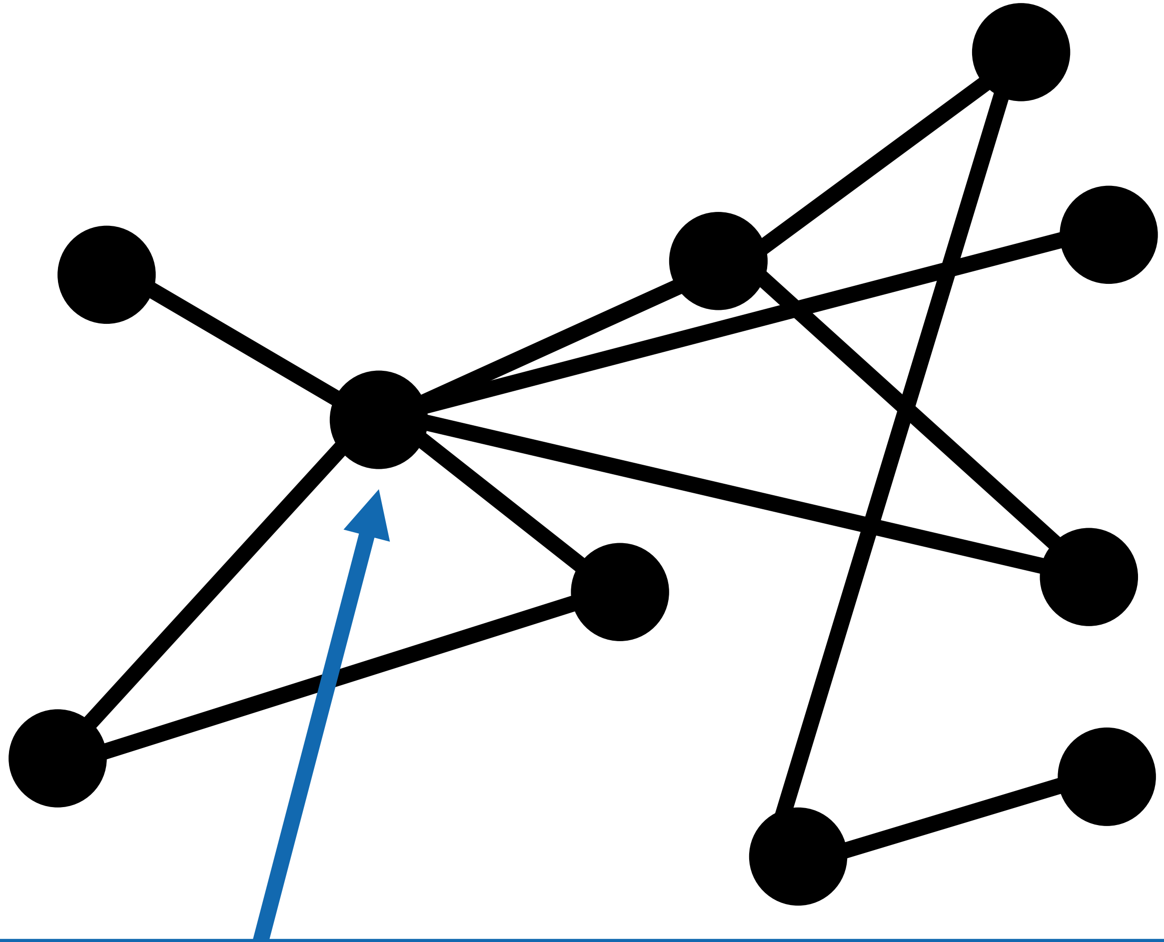
Graph degeneracy: an example



This graph has degeneracy of 3

Despite the high-degree vertex, each induced subgraph with this vertex contains some other vertex with degree at most 3

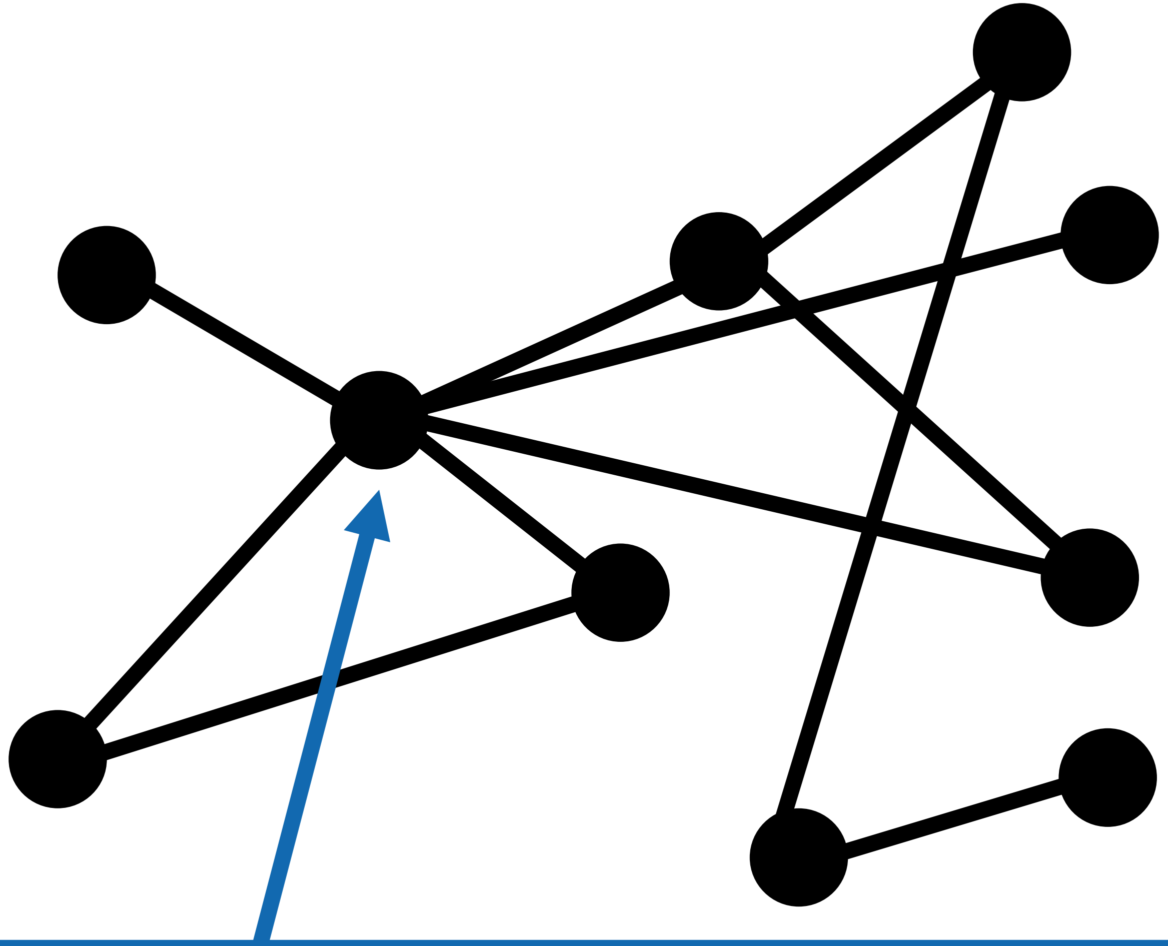
Graph degeneracy: an example



This graph has degeneracy of 3

Despite the high-degree vertex, each induced subgraph with this vertex contains some other vertex with degree at most 3

Graph degeneracy: an example

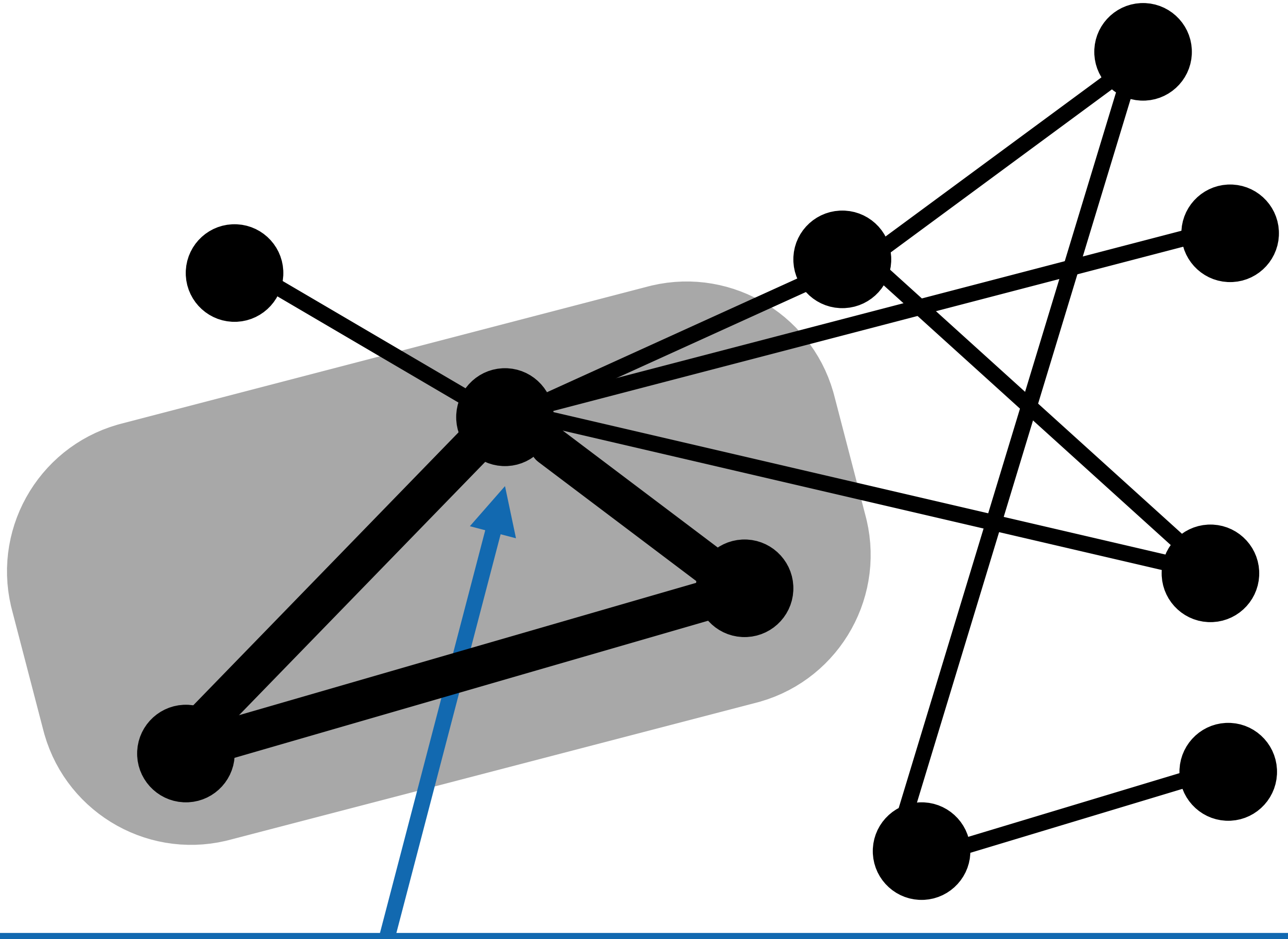


This graph has degeneracy of 3

Once a subgraph gets smaller, the degree becomes smaller than 3 anyway (as one considers induced subgraphs)

Despite the high-degree vertex, each induced subgraph with this vertex contains some other vertex with degree at most 3

Graph degeneracy: an example



This graph has degeneracy of 3

Once a subgraph gets smaller, the degree becomes smaller than 3 anyway (as one considers induced subgraphs)

Despite the high-degree vertex, each induced subgraph with this vertex contains some other vertex with degree at most 3

“Smallest degree last”: fundamentals

→ Iterate over vertices in the **degeneracy ordering**

```
/* n: number of vertices,
   m: number of edges,
   Δ: maximum vertex degree,
   d: graph’s degeneracy */
```

“Smallest degree last”: fundamentals

```
/* n: number of vertices,
   m: number of edges,
   Δ: maximum vertex degree,
   d: graph’s degeneracy */
```

→ Iterate over vertices in the **degeneracy ordering**

? How to derive the degeneracy ordering?

“Smallest degree last”: fundamentals

```
/* n: number of vertices,
   m: number of edges,
   Δ: maximum vertex degree,
   d: graph’s degeneracy */
```

→ Iterate over vertices in the **degeneracy ordering**

How to derive the degeneracy ordering?

Simple: Sequentially remove vertices of smallest degree, one by one.

"Smallest degree last": fundamentals

```
/* n: number of vertices,
   m: number of edges,
   Δ: maximum vertex degree,
   d: graph's degeneracy */
```

→ Iterate over vertices in the **degeneracy ordering**

? How to derive the degeneracy ordering?

Simple: Sequentially remove vertices of smallest degree, one by one.

✓ Deriving the ordering takes O(n+m) work

"Smallest degree last": fundamentals

```
/* n: number of vertices,
   m: number of edges,
   Δ: maximum vertex degree,
   d: graph's degeneracy */
```

→ Iterate over vertices in the **degeneracy ordering**

? How to derive the degeneracy ordering?

Simple: Sequentially remove vertices of smallest degree, one by one.

✓ Deriving the ordering takes O(n+m) work

✓ The corresponding coloring heuristics takes O(n+m) work and gives d+1 quality

"Smallest degree last": fundamentals

```
/* n: number of vertices,
   m: number of edges,
   Δ: maximum vertex degree,
   d: graph's degeneracy */
```

→ Iterate over vertices in the **degeneracy ordering**

? How to derive the degeneracy ordering?

Simple: Sequentially remove vertices of smallest degree, one by one.

✓ Deriving the ordering takes O(n+m) work

✓ The corresponding coloring heuristics takes O(n+m) work and gives d+1 quality

✗ Deriving the ordering takes O(n) depth (i.e., it is inherently sequential)

"Smallest degree last": fundamentals

```
/* n: number of vertices,
   m: number of edges,
   Δ: maximum vertex degree,
   d: graph's degeneracy */
```

→ Iterate over vertices in the **degeneracy ordering**

How to derive the degeneracy ordering?

The corresponding
cs is
d by
the ordering
derivation

Simple: Sequentially remove vertices of smallest degree, one by one.

✓ Deriving the ordering takes O(n+m) work

✓ The corresponding coloring heuristics takes O(n+m) work and gives d+1 quality

✗ Deriving the ordering takes O(n) depth (i.e., it is inherently sequential)

✗ The corresponding coloring heuristics is thus bottlenecked by the ordering derivation

“Smallest degree last”: fundamentals

```
/* n: number of vertices,
   m: number of edges,
   Δ: maximum vertex degree,
   d: graph’s degeneracy */
```

→ Iterate over vertices in the **degeneracy ordering**

? How to derive the degeneracy ordering?

Simple: Sequentially remove ver
degree, one by on

The corresponding coloring heuristics is thus bottlenecked by the ordering derivation

? Can we have both good degeneracy-based quality and low depth & work ?

✓ Deriving the ordering takes O(n+m) work

✓ The corresponding coloring heuristics takes O(n+m) work and gives d+1 quality

✗ Deriving the ordering takes O(n) depth (i.e., it is inherently sequential)

✗ The corresponding coloring heuristics is thus bottlenecked by the ordering derivation

Approximate degeneracy ordering

→ **Key idea**: try a relaxation of the strict degeneracy order, at the cost of (some) accuracy loss.

```
/* n: the number of all vertices
   V: set of all vertices,
   d(v): degree of a vertex v,
   $d_{avg}$: average degree in V */
```

One can prove that R_{min} forms a constant fraction of all vertices

```
itr = 0;
while V ≠ Ø:
   $R_{min} = \{v \mid d(v) \le (1+\varepsilon)d_{avg}\}$;
   $V = V \setminus R_{min}$;
   forall v in $R_{min}$ in parallel:
      rank[v] = itr;
   ++itr;
```

Approximate degeneracy ordering

```
/* n: the number of all vertices
   V: set of all vertices,
   d(v): degree of a vertex v,
   $d_{avg}$: average degree in V */
```

→ **Key idea**: try a <u>relaxation</u> of the strict degeneracy order, at the cost of (some) accuracy loss.

One can prove that R_{min} forms a constant fraction of all vertices

Constructing R_{min} takes O(|V|) work

```
itr = 0;
while V ≠ Ø:
    $R_{min}$ = {v | d(v) ≤ (1+ε)$d_{avg}$};
    V = V \ $R_{min}$;
    forall v in $R_{min}$ in parallel:
        rank[v] = itr;
    ++itr;
```

Approximate degeneracy ordering

→ **Key idea**: try a <u>relaxation</u> of the strict degeneracy order, at the cost of (some) accuracy loss.

```
/* n: the number of all vertices
   V: set of all vertices,
   d(v): degree of a vertex v,
   $d_{avg}$: average degree in V */
```

One can prove that R_{min} forms a constant fraction of all vertices

Constructing R_{min} takes O(|V|) work

Subtracting R_{min} from V takes O($|R_{min}|$) work

```
itr = 0;
while V ≠ ∅:
    $R_{min}$ = {v | d(v) ≤ $(1+\varepsilon)d_{avg}$};
    V = V \ $R_{min}$;
    forall v in $R_{min}$ in parallel:
        rank[v] = itr;
    ++itr;
```

Approximate degeneracy ordering

→ **Key idea**: try a relaxation of the strict degeneracy order, at the cost of (some) accuracy loss.

```
/* n: the number of all vertices
   V: set of all vertices,
   d(v): degree of a vertex v,
   $d_{avg}$: average degree in V */
```

One can prove that R_{min} forms a constant fraction of all vertices

Constructing R_{min} takes O(|V|) work

Subtracting R_{min} from V takes O($|R_{min}|$) work

Assigning new ranks takes O($|R_{min}|$) work

```
itr = 0;
while V ≠ Ø:
    $R_{min}$ = {v | d(v) ≤ (1+ε)$d_{avg}$};
    V = V \ $R_{min}$;
    forall v in $R_{min}$ in parallel:
        rank[v] = itr;
    ++itr;
```

Approximate degeneracy ordering

→ **Key idea**: try a relaxation of the strict degeneracy order, at the cost of (some) accuracy loss.

```
/* n: the number of all vertices
   V: set of all vertices,
   d(v): degree of a vertex v,
   $d_{avg}$: average degree in V */
```

One can prove that R_{min} forms a constant fraction of all vertices

Constructing R_{min} takes O(|V|) work

Subtracting R_{min} from V takes O($|R_{min}|$) work

Assigning new ranks takes O($|R_{min}|$) work

|V| gets smaller by a constant fraction in each iteration

```
itr = 0;
while V ≠ ∅:
  $R_{min}$ = {v | d(v) ≤ (1+ε)$d_{avg}$};
  V = V \ $R_{min}$;
  forall v in $R_{min}$ in parallel:
     rank[v] = itr;
  ++itr;
```

Approximate degeneracy ordering

```
/* n: the number of all vertices
   V: set of all vertices,
   d(v): degree of a vertex v,
   $d_{avg}$: average degree in V */
```

→ **Key idea**: try a relaxation of the strict degeneracy order, at the cost of (some) accuracy loss.

All iterations take O(n+m) work

One can prove that R_{min} forms a constant fraction of all vertices

Constructing R_{min} takes O(|V|) work

Subtracting R_{min} from V takes O(|R_{min}|) work

Assigning new ranks takes O(|R_{min}|) work

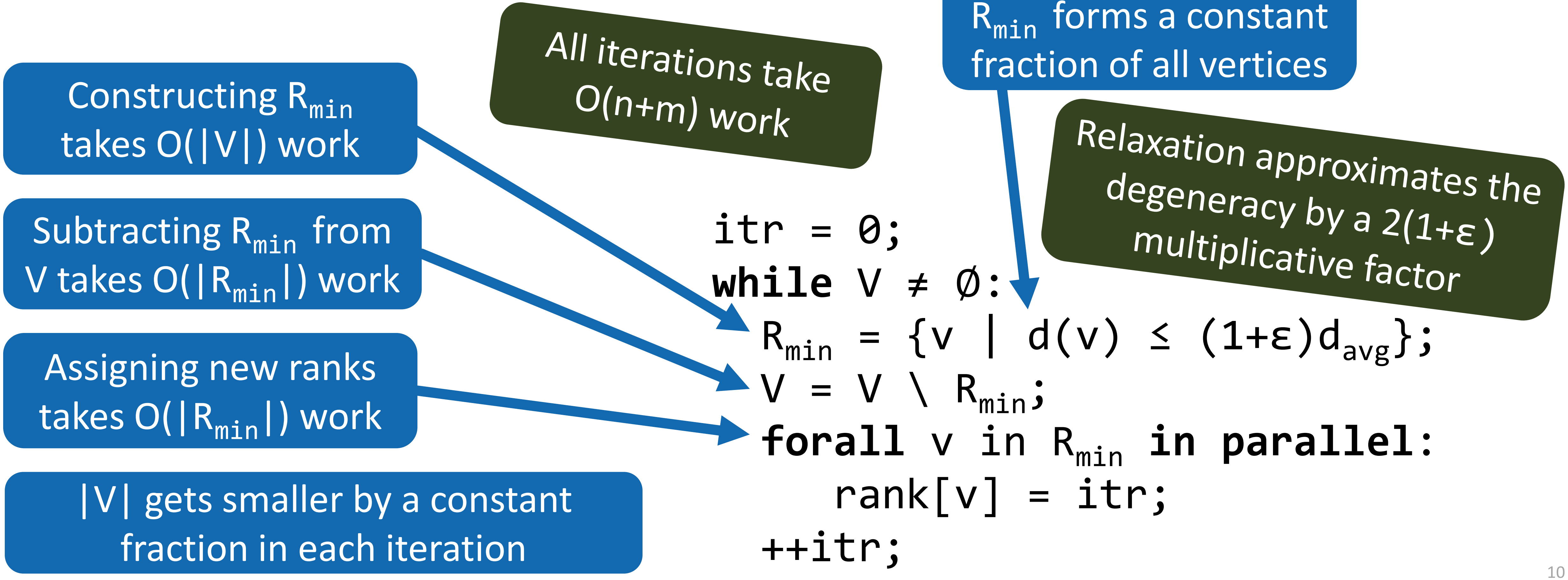
|V| gets smaller by a constant fraction in each iteration

```
itr = 0;
while V ≠ ∅:
  $R_{min}$ = {v | d(v) ≤ (1+ε)$d_{avg}$};
  V = V \ $R_{min}$;
  forall v in $R_{min}$ in parallel:
     rank[v] = itr;
  ++itr;
```

Approximate degeneracy ordering

```
/* n: the number of all vertices
   V: set of all vertices,
   d(v): degree of a vertex v,
   $d_{avg}$: average degree in V */
```

→ **Key idea**: try a relaxation of the strict degeneracy order, at the cost of (some) accuracy loss.



All iterations take O(n+m) work

One can prove that R_{min} forms a constant fraction of all vertices

Relaxation approximates the degeneracy by a 2(1+ε) multiplicative factor

Constructing R_{min} takes O(|V|) work

Subtracting R_{min} from V takes O($|R_{min}|$) work

Assigning new ranks takes O($|R_{min}|$) work

|V| gets smaller by a constant fraction in each iteration

```
itr = 0;
while V ≠ ∅:
  $R_{min}$ = {v | d(v) ≤ (1+ε)$d_{avg}$};
  V = V \ $R_{min}$;
  forall v in $R_{min}$ in parallel:
     rank[v] = itr;
  ++itr;
```

Parallel graph coloring heuristics

Ordering	Depth	Work	Quality
“First fit” (i.e., any order)	No general bounds; $\Omega(n)$ for some graphs	$O(n+m)$	$\Delta+1$
“Largest degree first”	No general bounds; $\Omega(\Delta^2)$ for some graphs	$O(n+m)$	$\Delta+1$
“Smallest degree last”	No general bounds; $\Omega(n)$ for some graphs	$O(n+m)$	$d+1$
Random	$\mathbb{E}\, O\left(\frac{\log n}{\log\log n}\right)$	$O(n+m)$	$\Delta+1$
Random	$\mathbb{E}\, O\left(\log n + \log\Delta \cdot \min\left\{\sqrt{m}, \Delta + \frac{\log\Delta\log n}{\log\log n}\right\}\right)$	$O(n+m)$	$\Delta+1$
“Largest log-degree first”	$\mathbb{E}\, O\left(\log n + \log\Delta \cdot \left(\min\{\Delta, \sqrt{m}\} + \frac{\log^2\Delta\log n}{\log\log n}\right)\right)$	$O(n+m)$	$\Delta+1$
“Smallest log-degree last”	$\mathbb{E}\, O\left(\log\Delta\log n + \log\Delta \cdot \left(\min\{\Delta, \sqrt{m}\} + \frac{\log^2\Delta\log n}{\log\log n}\right)\right)$	$O(n+m)$	$\Delta+1$
ADG (scheduling)	$\mathbb{E}\, O\left(\log^2 n + \log\Delta \cdot \left(d\log n + \frac{\log d \cdot \log^2 n}{\log\log n}\right)\right)$	$O(n+m)$	$2(1+\varepsilon)d+1$

Parallel graph coloring heuristics

Ordering	Depth	Work	Quality
“First fit” (i.e., any order)	No general bounds; $\Omega(n)$ for some graphs	$O(n+m)$	$\Delta+1$
“Largest degree first”	No general bounds; $\Omega(\Delta^2)$ for some graphs	$O(n+m)$	$\Delta+1$
“Smallest degree last”	No general bounds; $\Omega(n)$ for some graphs	$O(n+m)$	$d+1$
Random	$\mathbb{E}\, O\left(\frac{\log n}{\log\log n}\right)$	$O(n+m)$	$\Delta+1$
Random	$\mathbb{E}\, O\left(\log n+\log\Delta\cdot\min\left\{\sqrt{m},\Delta+\frac{\log\Delta\log n}{\log\log n}\right\}\right)$	$O(n+m)$	$\Delta+1$
“Largest log-degree first”	$\mathbb{E}\, O\left(\log n+\log\Delta\cdot\left(\min\{\Delta,\sqrt{m}\}+\frac{\log^2\Delta\log n}{\log\log n}\right)\right)$	$O(n+m)$	$\Delta+1$
“Smallest log-degree last”	$\mathbb{E}\, O\left(\log\Delta\log n+\log\Delta\cdot\left(\min\{\Delta,\sqrt{m}\}+\frac{\log^2\Delta\log n}{\log\log n}\right)\right)$	$O(n+m)$	$\Delta+1$
ADG (scheduling)	$\mathbb{E}\, O\left(\log^2 n+\log\Delta\cdot\left(d\log n+\frac{\log d\cdot\log^2 n}{\log\log n}\right)\right)$	$O(n+m)$	$2(1+\varepsilon)d+1$
ADG (scheduling)	$\mathbb{E}\, O\left(\log^2 n+\log\Delta\cdot\left(d\log n+\frac{\log d\cdot\log^2 n}{\log\log n}\right)\right)$	$O(n+m)$	$4d+1$
ADG (speculative)	$O\left(\log d\log^2 n\right)$ w.h.p.	$O(n+m)$ (w.h.p.)	$(2+\varepsilon)d$
ADG (speculative)	$O\left(\log d\log^2 n\right)$ w.h.p.	$O(n+m)$ (w.h.p.)	$(4+\varepsilon)d$
ADG (speculative)	$O\left(I\cdot d\log n\right)$		$2(1+\varepsilon)d+1$

Parallel graph coloring heuristics

Ordering	Depth	Work	Quality
"First fit" (i.e., any order)	No general bounds; $\Omega(n)$ for some graphs	$O(n+m)$	$\Delta+1$
"Largest degree first"	No general bounds; $\Omega(\Delta^2)$ for some graphs	$O(n+m)$	$\Delta+1$
"Smallest degree last"	No general bounds; $\Omega(n)$ for some graphs	$O(n+m)$	$d+1$
Random	$\mathbb{E}\, O\left(\frac{\log n}{\log\log n}\right)$	$O(n+m)$	$\Delta+1$
Random	$\mathbb{E}\, O\left(\log n + \log\Delta \cdot \min\left\{\sqrt{m}, \Delta + \frac{\log\Delta\log n}{\log\log n}\right\}\right)$	$O(n+m)$	$\Delta+1$
"Largest log-degree first"	$\mathbb{E}\, O\left(\log n + \log\Delta \cdot \left(\min\{\Delta, \sqrt{m}\} + \frac{\log^2\Delta\log n}{\log\log n}\right)\right)$	$O(n+m)$	$\Delta+1$
"Smallest log-degree last"	$\mathbb{E}\, O\left(\log\Delta\log n + \log\Delta \cdot \left(\min\{\Delta, \sqrt{m}\} + \frac{\log^2\Delta\log n}{\log\log n}\right)\right)$	$O(n+m)$	$\Delta+1$
ADG (scheduling)	$\mathbb{E}\, O\left(\log^2 n + \log\Delta \cdot \left(d\log n + \frac{\log d \cdot \log^2 n}{\log\log n}\right)\right)$	$O(n+m)$	$2(1+\varepsilon)d+1$
ADG (scheduling)	$\mathbb{E}\, O\left(\log^2 n + \log\Delta \cdot \left(d\log n + \frac{\log d \cdot \log^2 n}{\log\log n}\right)\right)$	$O(n+m)$	$4d+1$
ADG (speculative)	$O\left(\log d \log^2 n\right)$ w.h.p.	$O(n+m)$ (w.h.p.)	$(2+\varepsilon)d$
ADG (speculative)	$O\left(\log d \log^2 n\right)$ w.h.p.	$O(n+m)$ (w.h.p.)	$(4+\varepsilon)d$
ADG (speculative)	$O\left(I \cdot d\log n\right)$		$2(1+\varepsilon)d+1$

All details, proofs, etc., are in the paper ☺

Approximate degeneracy ordering

→ **Key idea**: try a relaxation of the strict degeneracy order, at the cost of (some) accuracy loss.

```
/* n: the number of all vertices
   V: set of all vertices,
   d(v): degree of a vertex v,
   $d_{avg}$: average degree in V */
```

One can prove that R_{min} forms a constant fraction of all vertices

```
itr = 0;
while $V \neq \emptyset$:
   $R_{min} = \{v \mid d(v) \leq (1+\varepsilon)d_{avg}\}$;
   $V = V \setminus R_{min}$;
   forall v in $R_{min}$ in parallel:
      rank[v] = itr;
   ++itr;
```

Approximate degeneracy ordering

```
/* n: the number of all vertices
   V: set of all vertices,
   d(v): degree of a vertex v,
   $d_{avg}$: average degree in V */
```

→ **Key idea**: try a <u>relaxation</u> of the strict degeneracy order, at the cost of (some) accuracy loss.

One can prove that R_{min} forms a constant fraction of all vertices

Constructing R_{min} takes O(|V|) work

```
itr = 0;
while V ≠ Ø:
    $R_{min}$ = {v | d(v) ≤ (1+ε)$d_{avg}$};
    V = V \ $R_{min}$;
    forall v in $R_{min}$ in parallel:
        rank[v] = itr;
    ++itr;
```

Approximate degeneracy ordering

```
/* n: the number of all vertices
   V: set of all vertices,
   d(v): degree of a vertex v,
   $d_{avg}$: average degree in V */
```

→ **Key idea**: try a relaxation of the strict degeneracy order, at the cost of (some) accuracy loss.

One can prove that R_{min} forms a constant fraction of all vertices

Constructing R_{min} takes O(|V|) work

Subtracting R_{min} from V takes O(|R_{min}|) work

```
itr = 0;
while V ≠ ∅:
    $R_{min}$ = {v | d(v) ≤ (1+ε)$d_{avg}$};
    V = V \ $R_{min}$;
    forall v in $R_{min}$ in parallel:
        rank[v] = itr;
    ++itr;
```

Approximate degeneracy ordering

```
/* n: the number of all vertices
   V: set of all vertices,
   d(v): degree of a vertex v,
   $d_{avg}$: average degree in V */
```

→ **Key idea**: try a relaxation of the strict degeneracy order, at the cost of (some) accuracy loss.

One can prove that R_{min} forms a constant fraction of all vertices

Constructing R_{min} takes O(|V|) work

Subtracting R_{min} from V takes O(|R_{min}|) work

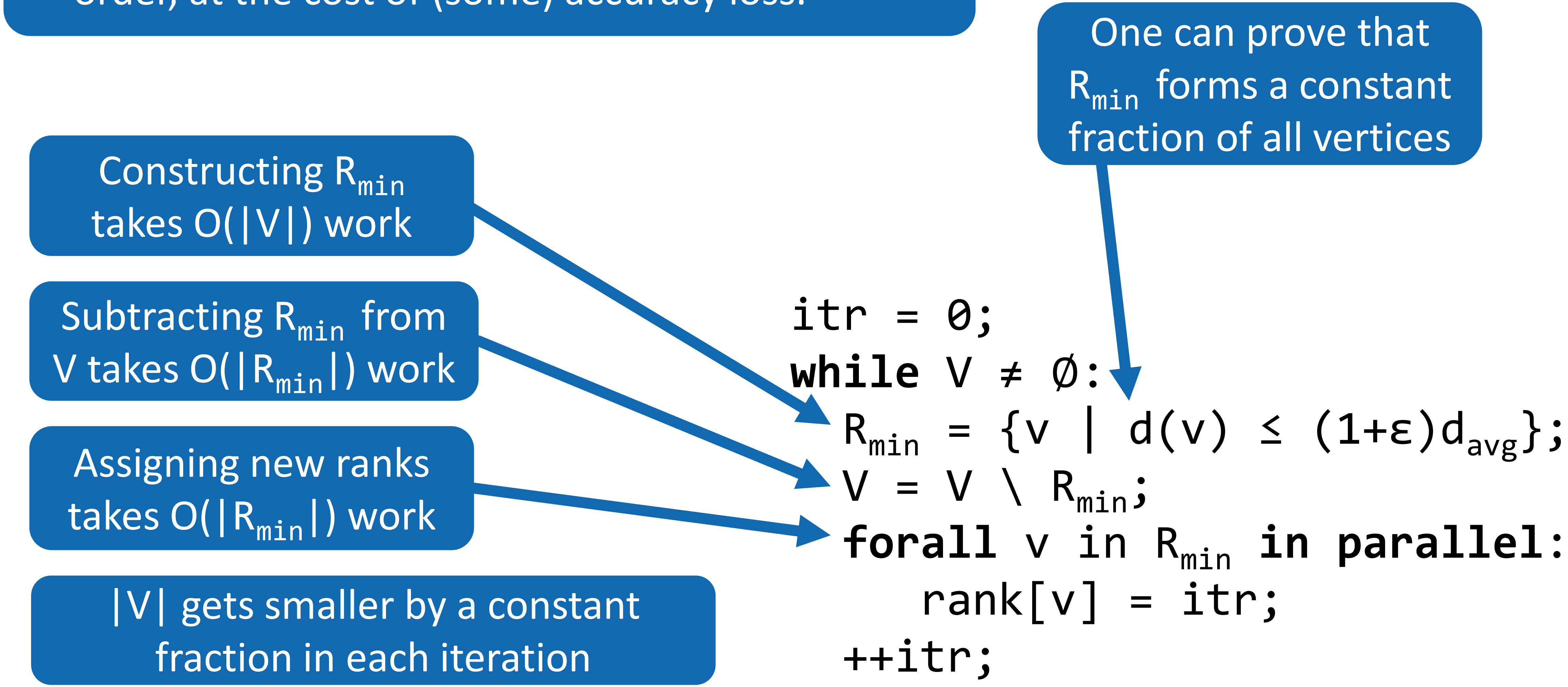
Assigning new ranks takes O(|R_{min}|) work

```
itr = 0;
while V ≠ ∅:
    $R_{min}$ = {v | d(v) ≤ (1+ε)$d_{avg}$};
    V = V \ $R_{min}$;
    forall v in $R_{min}$ in parallel:
        rank[v] = itr;
    ++itr;
```

Approximate degeneracy ordering

```
/* n: the number of all vertices
   V: set of all vertices,
   d(v): degree of a vertex v,
   $d_{avg}$: average degree in V */
```

→ **Key idea**: try a relaxation of the strict degeneracy order, at the cost of (some) accuracy loss.



One can prove that R_{min} forms a constant fraction of all vertices

Constructing R_{min} takes O(|V|) work

Subtracting R_{min} from V takes O($|R_{min}|$) work

Assigning new ranks takes O($|R_{min}|$) work

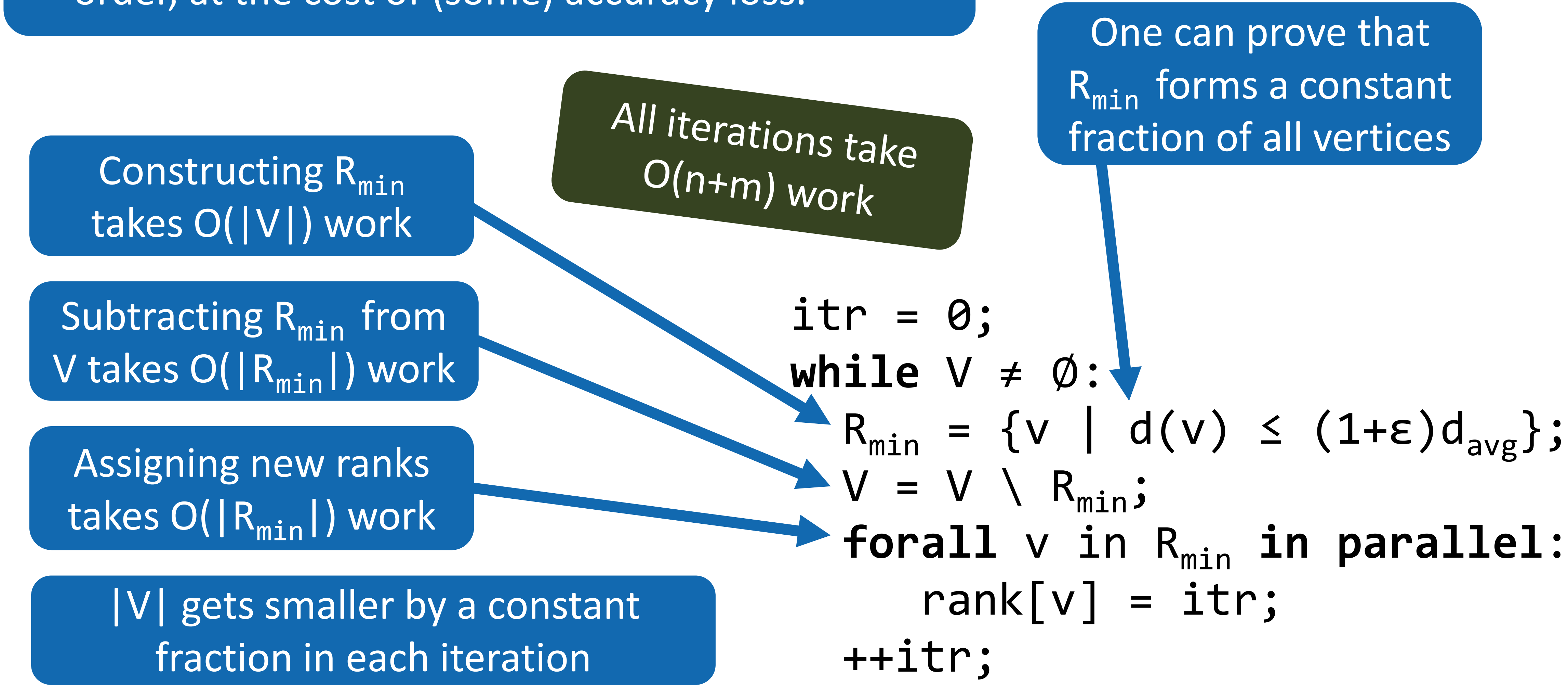
|V| gets smaller by a constant fraction in each iteration

```
itr = 0;
while V ≠ Ø:
    $R_{min}$ = {v | d(v) ≤ (1+ε)$d_{avg}$};
    V = V \ $R_{min}$;
    forall v in $R_{min}$ in parallel:
        rank[v] = itr;
    ++itr;
```

Approximate degeneracy ordering

→ **Key idea**: try a relaxation of the strict degeneracy order, at the cost of (some) accuracy loss.

```
/* n: the number of all vertices
   V: set of all vertices,
   d(v): degree of a vertex v,
   $d_{avg}$: average degree in V */
```



One can prove that R_{min} forms a constant fraction of all vertices

All iterations take O(n+m) work

Constructing R_{min} takes O(|V|) work

Subtracting R_{min} from V takes O(|R_{min}|) work

Assigning new ranks takes O(|R_{min}|) work

|V| gets smaller by a constant fraction in each iteration

```
itr = 0;
while V ≠ ∅:
    $R_{min}$ = {v | d(v) ≤ (1+ε)$d_{avg}$};
    V = V \ $R_{min}$;
    forall v in $R_{min}$ in parallel:
        rank[v] = itr;
    ++itr;
```

Approximate degeneracy ordering

```
/* n: the number of all vertices
   V: set of all vertices,
   d(v): degree of a vertex v,
   $d_{avg}$: average degree in V */
```

→ **Key idea**: try a relaxation of the strict degeneracy order, at the cost of (some) accuracy loss.

All iterations take O(n+m) work

One can prove that R_{min} forms a constant fraction of all vertices

Relaxation approximates the degeneracy by a 2(1+ε) multiplicative factor

Constructing R_{min} takes O(|V|) work

Subtracting R_{min} from V takes O(|R_{min}|) work

Assigning new ranks takes O(|R_{min}|) work

|V| gets smaller by a constant fraction in each iteration

```
itr = 0;
while V ≠ ∅:
    $R_{min}$ = {v | d(v) ≤ (1+ε)$d_{avg}$};
    V = V \ $R_{min}$;
    forall v in $R_{min}$ in parallel:
        rank[v] = itr;
    ++itr;
```