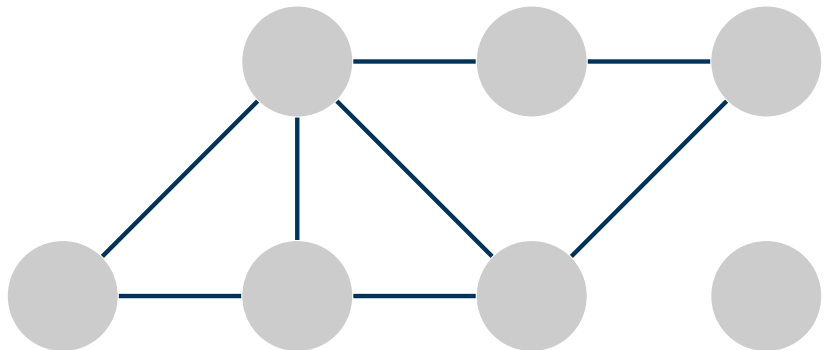
A More Reliable Greedy Heuristic for Maximum Matchings in Sparse Random Graphs

Martin Dietzfelbinger[1] Hendrik Peilke[2] Michael Rink[1]

[1]Technische Universität Ilmenau

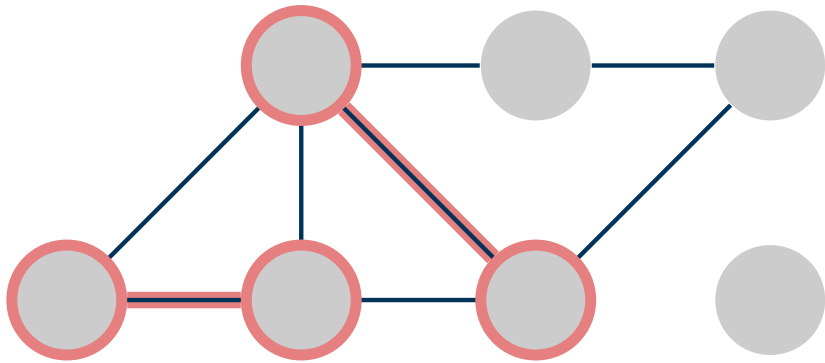
[2]IBYKUS AG

Maximum Cardinality Matching



undirected graph G:

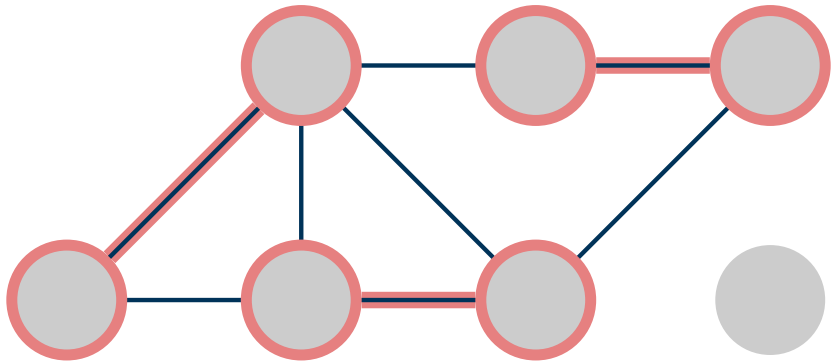
Maximum Cardinality Matching



undirected graph G:

- A matching M is set of pairwise disjoint edges from G.

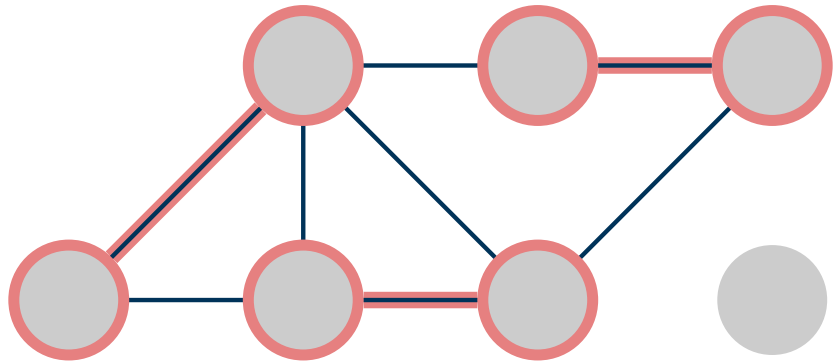
Maximum Cardinality Matching



undirected graph G:

- A matching M is set of pairwise disjoint edges from G.
- M is a maximum matching if it has largest possible cardinality.

Maximum Cardinality Matching



undirected graph G:

- A matching M is set of pairwise disjoint edges from G.
- M is a maximum matching if it has largest possible cardinality.
- The problem of finding a maximum matching is well understood.

Algorithms for Maximum Matchings

(arbitrary) graph $G = (V, E)$ with n nodes, m edges:

Algorithms for Maximum Matchings

(arbitrary) graph $G = (V, E)$ with n nodes, m edges:

- first polynomial time algorithm [Edmonds, 1965] $O(n^2 \cdot m)$

Algorithms for Maximum Matchings

(arbitrary) graph $G = (V, E)$ with n nodes, m edges:

- first polynomial time algorithm [Edmonds, 1965] $O(n^2 \cdot m)$
- many followed, e.g., [Micali and Vazirani, 1980] $O(n^{1/2} \cdot m)$

Algorithms for Maximum Matchings

(arbitrary) graph $G = (V, E)$ with n nodes, m edges:

- first polynomial time algorithm [Edmonds, 1965] $O(n^2 \cdot m)$
- many followed, e.g., [Micali and Vazirani, 1980] $O(n^{1/2} \cdot m)$
- for dense graphs $m = \Theta(n^2)$ improved to $O(n^\omega)$ (expected), $\omega < 2.376$, by [Mucha and Sankowski, 2004]

Algorithms for Maximum Matchings

(arbitrary) graph $G = (V, E)$ with n nodes, m edges:

- first polynomial time algorithm [Edmonds, 1965] $O(n^2 \cdot m)$
- many followed, e.g., [Micali and Vazirani, 1980] $O(n^{1/2} \cdot m)$
- for dense graphs $m = \Theta(n^2)$ improved to $O(n^\omega)$ (expected), $\omega < 2.376$, by [Mucha and Sankowski, 2004]

random graph $G(n; c)$ with n nodes, constant expected degree c:

Algorithms for Maximum Matchings

(arbitrary) graph $G = (V, E)$ with n nodes, m edges:

- first polynomial time algorithm [Edmonds, 1965] $O(n^2 \cdot m)$
- many followed, e.g., [Micali and Vazirani, 1980] $O(n^{1/2} \cdot m)$
- for dense graphs $m = \Theta(n^2)$ improved to $O(n^\omega)$ (expected), $\omega < 2.376$, by [Mucha and Sankowski, 2004]

random graph $G(n; c)$ with n nodes, constant expected degree c:

- [Bast et al., 2006] showed that if $c > 32.67$ then maximum matching can be found in time $O(n \cdot \log n)$ (expected)
  - conjectured that this holds for all constants $c > 0$

Algorithms for Maximum Matchings

(arbitrary) graph $G = (V, E)$ with n nodes, m edges:

- first polynomial time algorithm [Edmonds, 1965] $O(n^2 \cdot m)$
- many followed, e.g., [Micali and Vazirani, 1980] $O(n^{1/2} \cdot m)$
- for dense graphs $m = \Theta(n^2)$ improved to $O(n^\omega)$ (expected), $\omega < 2.376$, by [Mucha and Sankowski, 2004]

random graph $G(n; c)$ with n nodes, constant expected degree c:

- [Bast et al., 2006] showed that if $c > 32.67$ then maximum matching can be found in time $O(n \cdot \log n)$ (expected)
  - conjectured that this holds for all constants $c > 0$
- [Chebolu et al., 2010] gave algorithm with running time $O(n)$ (expected)

Algorithms for Maximum Matchings

(arbitrary) graph $G = (V, E)$ with n nodes, m edges:

- first polynomial time algorithm [Edmonds, 1965] $O(n^2 \cdot m)$
- many followed, e.g., [Micali and Vazirani, 1980] $O(n^{1/2} \cdot m)$
- for dense graphs $m = \Theta(n^2)$ improved to $O(n^\omega)$ (expected), $\omega < 2.376$, by [Mucha and Sankowski, 2004]

random graph $G(n; c)$ with n nodes, constant expected degree c:

- [Bast et al., 2006] showed that if $c > 32.67$ then maximum matching can be found in time $O(n \cdot \log n)$ (expected)
  - conjectured that this holds for all constants $c > 0$
- [Chebolu et al., 2010] gave algorithm with running time $O(n)$ (expected)

The (exact) algorithms remain complicated!

Algorithms for Maximum Matchings

(arbitrary) graph $G = (V, E)$ with n nodes, m edges:

- first polynomial time algorithm [Edmonds, 1965] $O(n^2 \cdot m)$
- many followed, e.g., [Micali and Vazirani, 1980] $O(n^{1/2} \cdot m)$
- for dense graphs $m = \Theta(n^2)$ improved to $O(n^\omega)$ (expected), $\omega < 2.376$, by [Mucha and Sankowski, 2004]

random graph $G(n; c)$ with n nodes, constant expected degree c:

- [Bast et al., 2006] showed that if $c > 32.67$ then maximum matching can be found in time $O(n \cdot \log n)$ (expected)
  - conjectured that this holds for all constants $c > 0$
- [Chebolu et al., 2010] gave algorithm with running time $O(n)$ (expected)

The (exact) algorithms remain complicated!

What can be achieved with simpler algorithms?

Matching Heuristics for Sparse Random Graphs

greedy heuristics for $G(n; c)$:

Matching Heuristics for Sparse Random Graphs

greedy heuristics for $G(n; c)$:

- Karp-Sipser heuristic [Karp and Sipser, 1981, Aronson et al., 1998]
  - if $c < e \approx 2.718$ finds maximum matching (whp)
  - if $c > e \approx 2.718$ size of matching found is within $n^{1/5+o(1)}$ of maximum cardinality

Matching Heuristics for Sparse Random Graphs

greedy heuristics for $G(n; c)$:

- Karp-Sipser heuristic [Karp and Sipser, 1981, Aronson et al., 1998]
  - if $c < e \approx 2.718$ finds maximum matching (whp)
  - if $c > e \approx 2.718$ size of matching found is within $n^{1/5+o(1)}$ of maximum cardinality
- experimental studies of several heuristics, e.g., [Magun, 1998]
  - There are good greedy heuristics with linear running time that are likely to find maximum matchings for a wide range of c.
  - Even the best heuristic often fails in the range of about $2.6 \leq c \leq 3.8$.

Matching Heuristics for Sparse Random Graphs

greedy heuristics for $G(n; c)$:

- Karp-Sipser heuristic [Karp and Sipser, 1981, Aronson et al., 1998]
  - if $c < e \approx 2.718$ finds maximum matching (whp)
  - if $c > e \approx 2.718$ size of matching found is within $n^{1/5+o(1)}$ of maximum cardinality
- experimental studies of several heuristics, e.g., [Magun, 1998]
  - There are good greedy heuristics with linear running time that are likely to find maximum matchings for a wide range of c.
  - Even the best heuristic often fails in the range of about $2.6 \leq c \leq 3.8$.

There is some region for c that seems critical for known greedy matching heuristics!

Matching Heuristics for Sparse Random Graphs

greedy heuristics for $G(n; c)$:

- Karp-Sipser heuristic [Karp and Sipser, 1981, Aronson et al., 1998]
  - if $c < e \approx 2.718$ finds maximum matching (whp)
  - if $c > e \approx 2.718$ size of matching found is within $n^{1/5+o(1)}$ of maximum cardinality
- experimental studies of several heuristics, e.g., [Magun, 1998]
  - There are good greedy heuristics with linear running time that are likely to find maximum matchings for a wide range of c.
  - Even the best heuristic often fails in the range of about $2.6 \leq c \leq 3.8$.

There is some region for c that seems critical for known greedy matching heuristics!

Is there a greedy heuristic with no critical region?

Result

We describe a new greedy heuristic with (close to) linear running time and give experimental evidence that this heuristic is likely to find a maximum matching in $G(n; c)$ for all ranges of c.

Result

We describe a new greedy heuristic with (close to) linear running time and give experimental evidence that this heuristic is likely to find a maximum matching in $G(n; c)$ for all ranges of c.

- Our approach is motivated by the "selfless algorithm" of [Sanders, 2004] for orienting undirected graphs.

Result

We describe a new greedy heuristic with (close to) linear running time and give experimental evidence that this heuristic is likely to find a maximum matching in $G(n; c)$ for all ranges of c.

- Our approach is motivated by the "selfless algorithm" of [Sanders, 2004] for orienting undirected graphs.
- We compared (quality of solution) our new heuristic with several good heuristics commonly used.

Basic Structure

heuristics:

Basic Structure

heuristics:

- set of simple reduction steps with a strict order of priority
- reduction step: select an edge and shrink the graph

Basic Structure

heuristics:

- set of simple reduction steps with a strict order of priority
- reduction step: select an edge and shrink the graph

algorithm

while G has an edge
    select applicable reduction step with highest priority
    apply the reduction

Basic Structure

heuristics:

- set of simple reduction steps with a strict order of priority
- reduction step: select an edge and shrink the graph

algorithm

while G has an edge
    select applicable reduction step with highest priority
    apply the reduction

two kinds of reduction steps:

Basic Structure

heuristics:

- set of simple reduction steps with a strict order of priority
- reduction step: select an edge and shrink the graph

algorithm

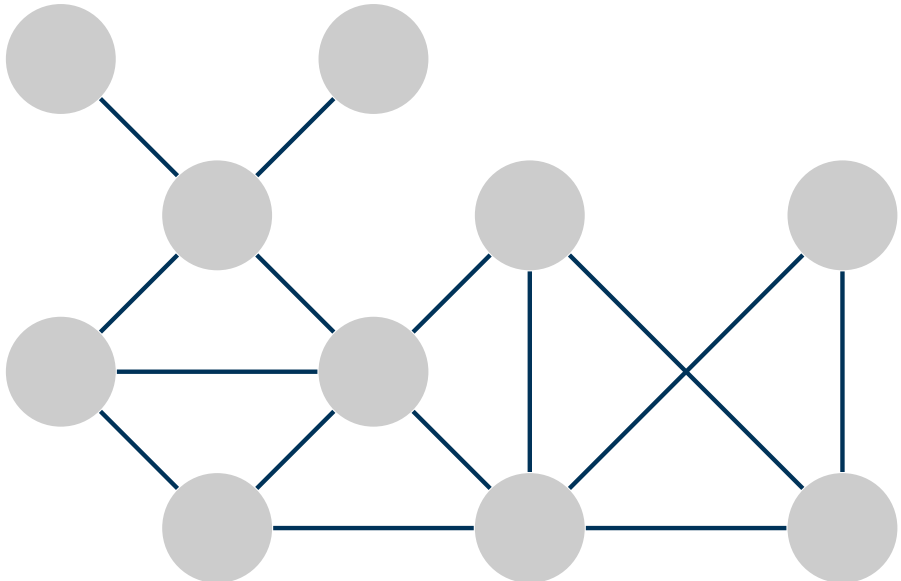
while G has an edge
    select applicable reduction step with highest priority
    apply the reduction

two kinds of reduction steps:

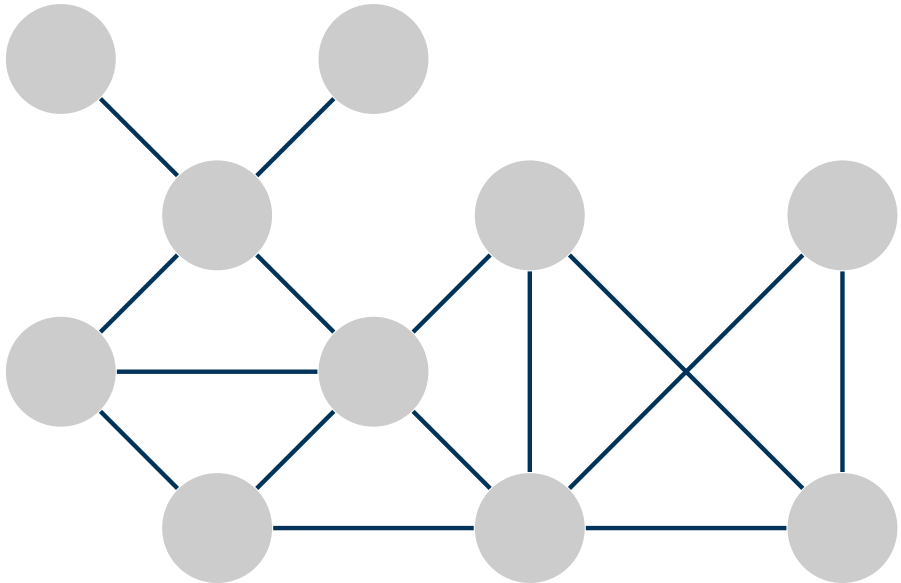
OPT never decrease the size of a maximum matching [Karp and Sipser, 1981]

HEU can decrease the size of a maximum matching

Optimal Reduction Steps

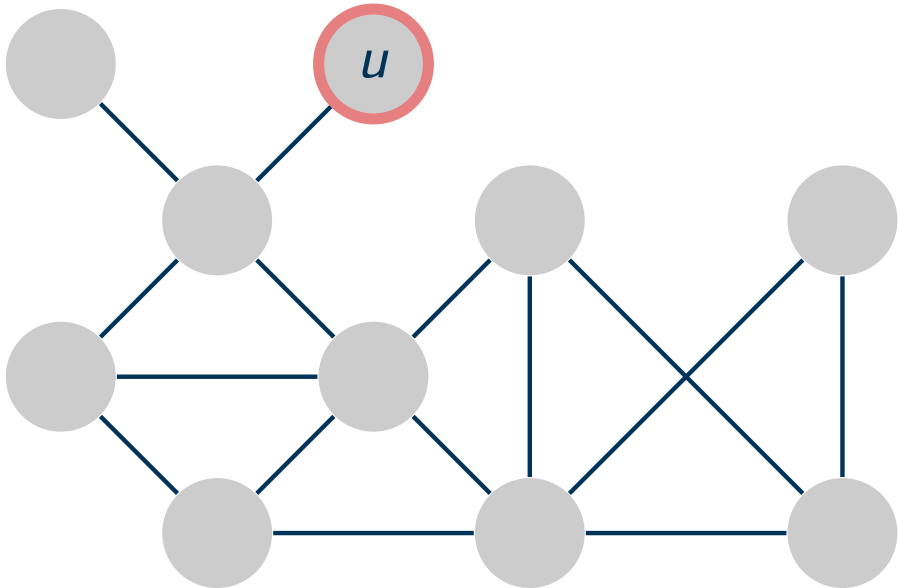


Optimal Reduction Steps



OPT(1)

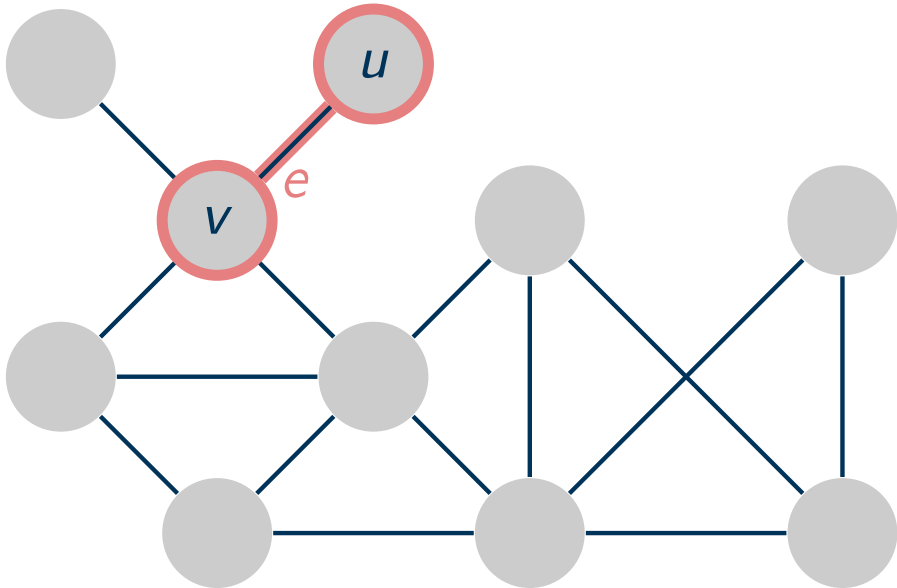
Optimal Reduction Steps



OPT(1)

- choose node u of degree 1

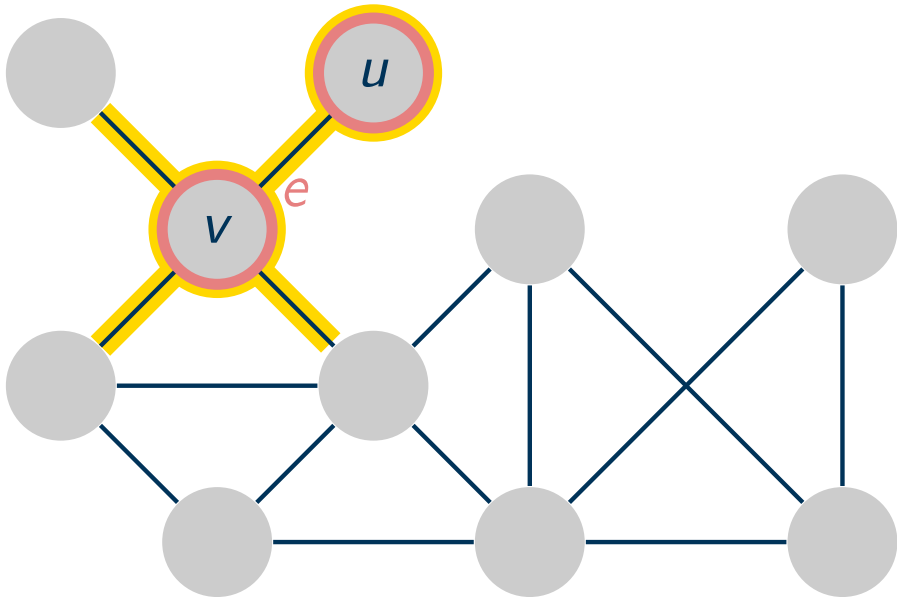
Optimal Reduction Steps



OPT(1)

- choose node u of degree 1
- incident edge $e = \{u, v\}$ belongs to matching

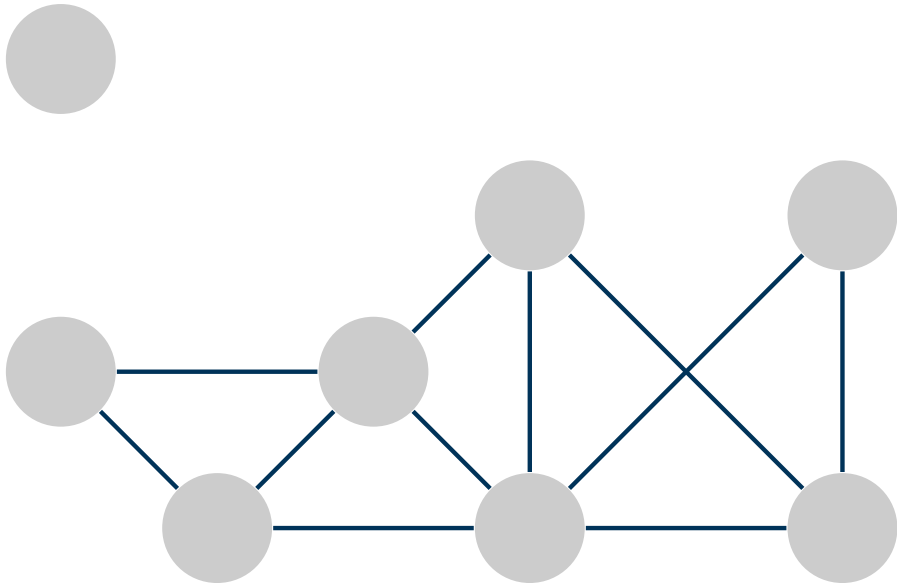
Optimal Reduction Steps



OPT(1)

- choose node u of degree 1
- incident edge $e = \{u, v\}$ belongs to matching
- shrink graph

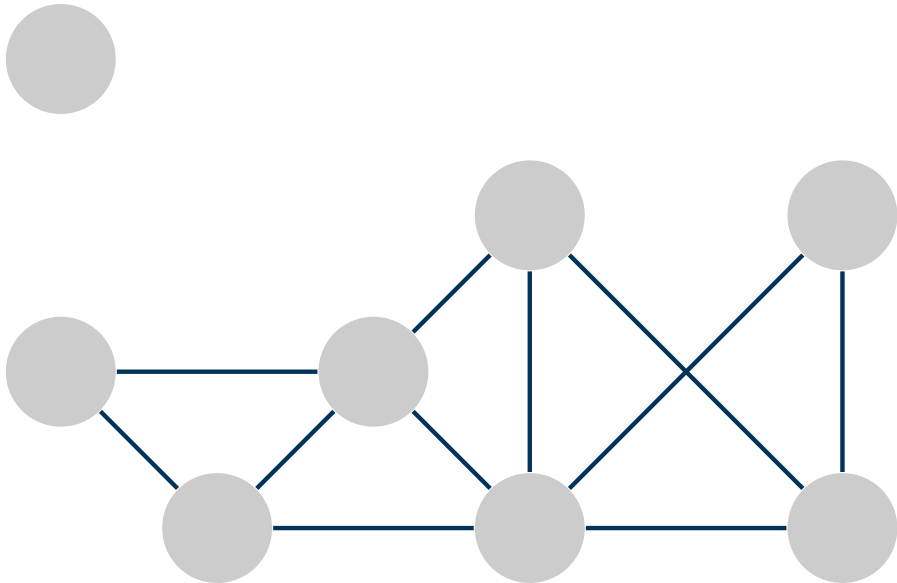
Optimal Reduction Steps



OPT(1)

- choose node u of degree 1
- incident edge $e = \{u, v\}$ belongs to matching
- shrink graph

Optimal Reduction Steps

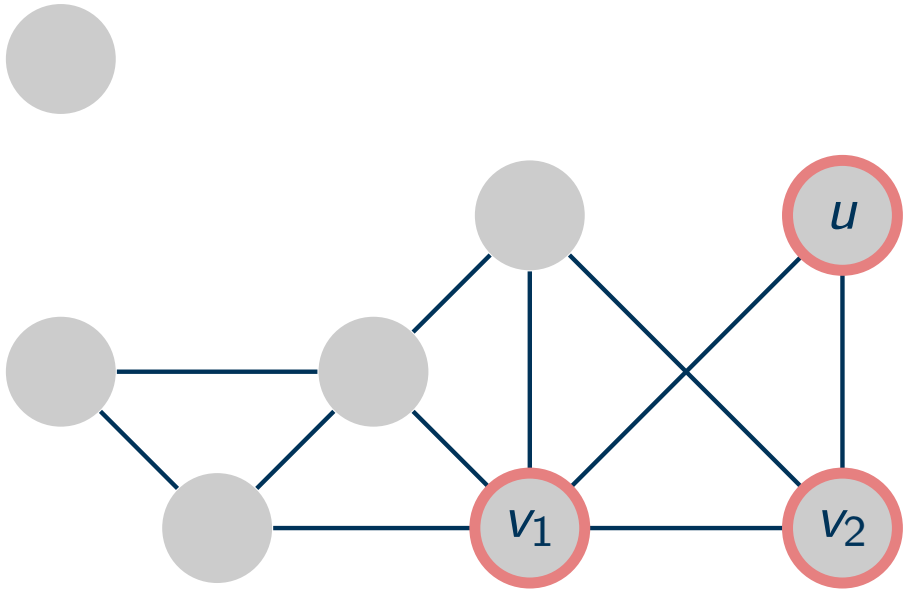


OPT(1)

- choose node u of degree 1
- incident edge $e = \{u, v\}$ belongs to matching
- shrink graph

OPT(2)

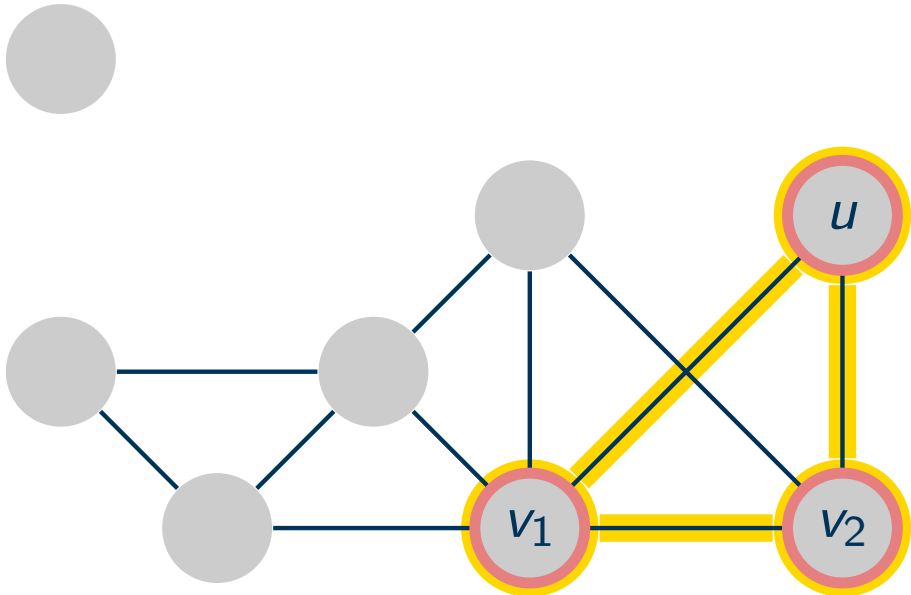
Optimal Reduction Steps



OPT(1)
- choose node u of degree 1
- incident edge $e = \{u, v\}$ belongs to matching
- shrink graph

OPT(2)
- choose node u of degree 2, adjacent to v_1 and v_2

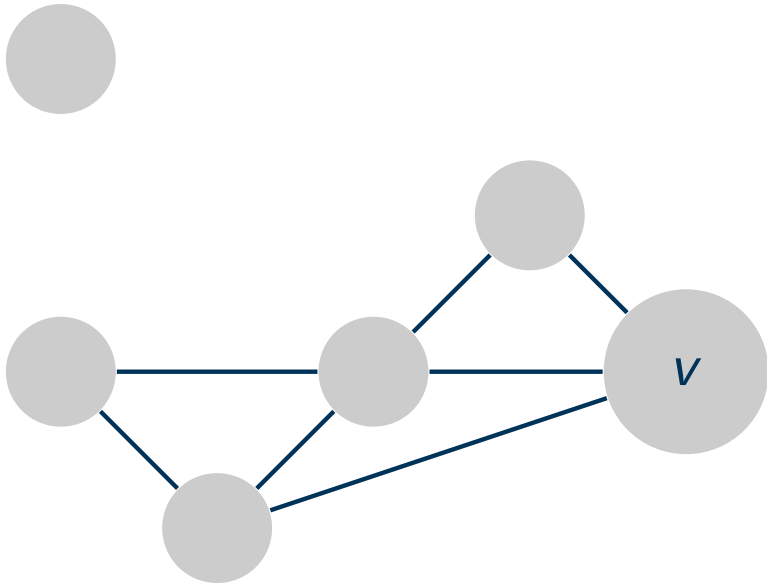
Optimal Reduction Steps



OPT(1)
- choose node u of degree 1
- incident edge $e = \{u, v\}$ belongs to matching
- shrink graph

OPT(2)
- choose node u of degree 2, adjacent to v_1 and v_2
- contract nodes into a single node v

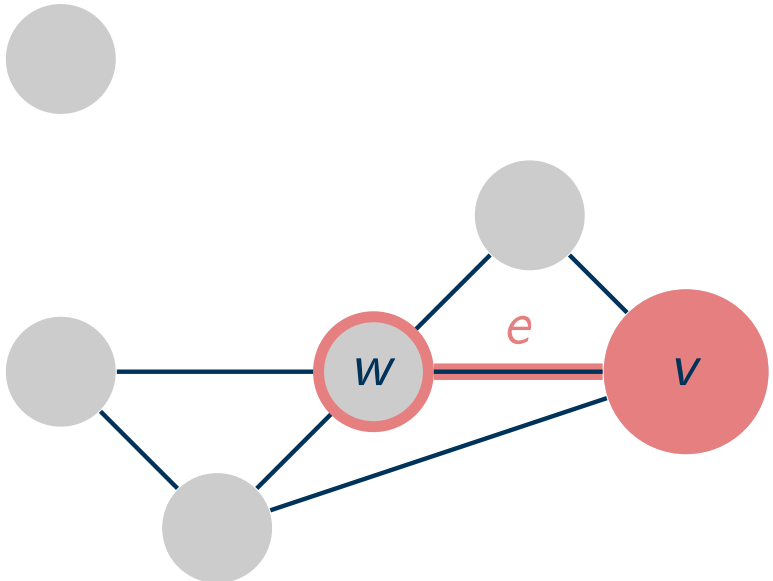
Optimal Reduction Steps



OPT(1)
- choose node u of degree 1
- incident edge $e = \{u, v\}$ belongs to matching
- shrink graph

OPT(2)
- choose node u of degree 2, adjacent to v_1 and v_2
- contract nodes into a single node v

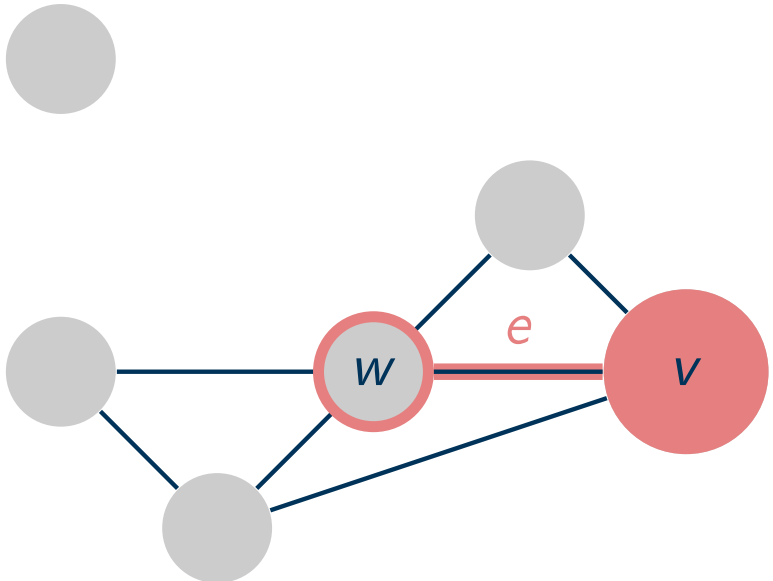
Optimal Reduction Steps



OPT(1)
- choose node u of degree 1
- incident edge $e = \{u, v\}$ belongs to matching
- shrink graph

OPT(2)
- choose node u of degree 2, adjacent to v_1 and v_2
- contract nodes into a single node v
- if $\{v, w\}$ becomes matching edge, by recursion,

Optimal Reduction Steps



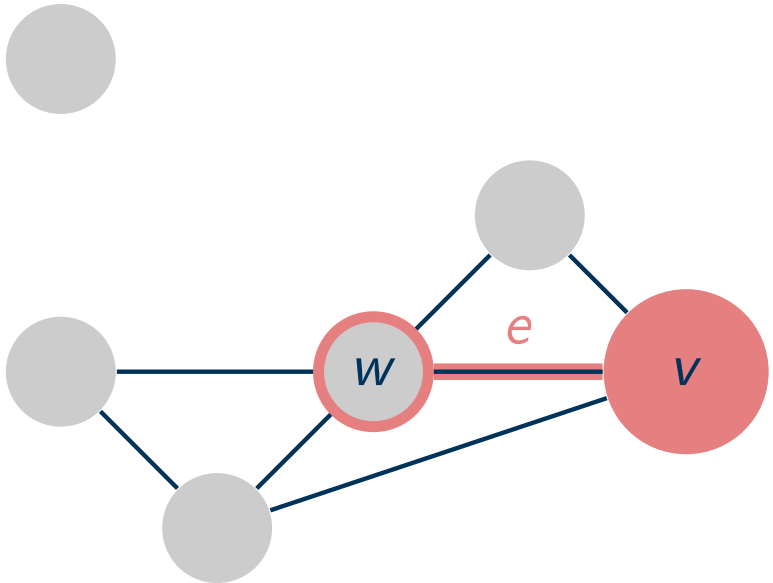
OPT(1)

- choose node u of degree 1
- incident edge $e = \{u, v\}$ belongs to matching
- shrink graph

OPT(2)

- choose node u of degree 2, adjacent to v_1 and v_2
- contract nodes into a single node v
- if $\{v, w\}$ becomes matching edge, by recursion, replace $\{v, w\}$ with $\{v_1, w\}$ and add $\{u, v_2\}$

Optimal Reduction Steps



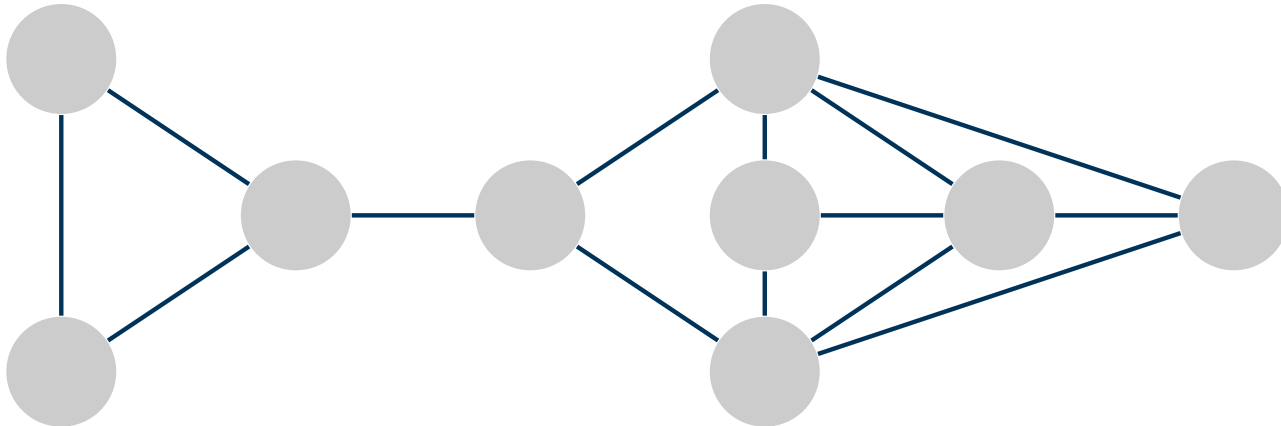
OPT(1)

- choose node u of degree 1
- incident edge $e = \{u, v\}$ belongs to matching
- shrink graph

OPT(2)

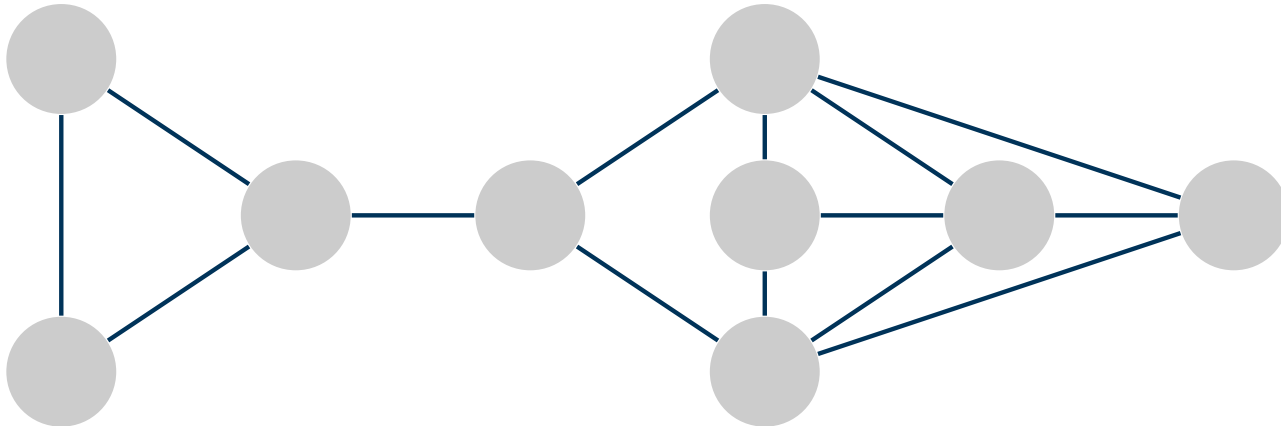
- choose node u of degree 2, adjacent to v_1 and v_2
- contract nodes into a single node v
- if $\{v, w\}$ becomes matching edge, by recursion, replace $\{v, w\}$ with $\{v_1, w\}$ and add $\{u, v_2\}$, or replace $\{v, w\}$ with $\{v_2, w\}$ and add $\{u, v_1\}$

Heuristic Reduction Steps



Choose an edge $e = \{u, v\}$ to put in the matching and shrink the graph.

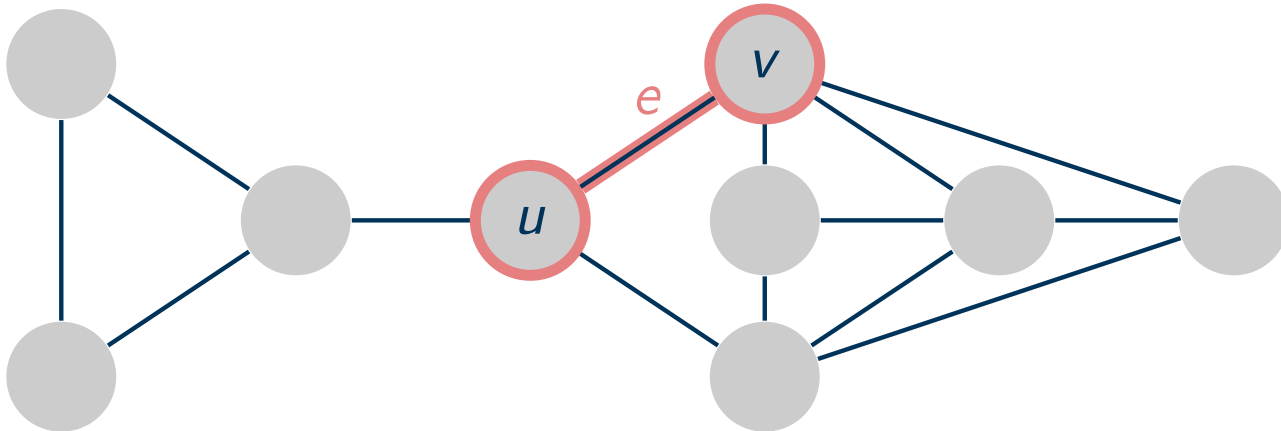
Heuristic Reduction Steps



Choose an edge $e = \{u, v\}$ to put in the matching and shrink the graph.

HEU(rand)

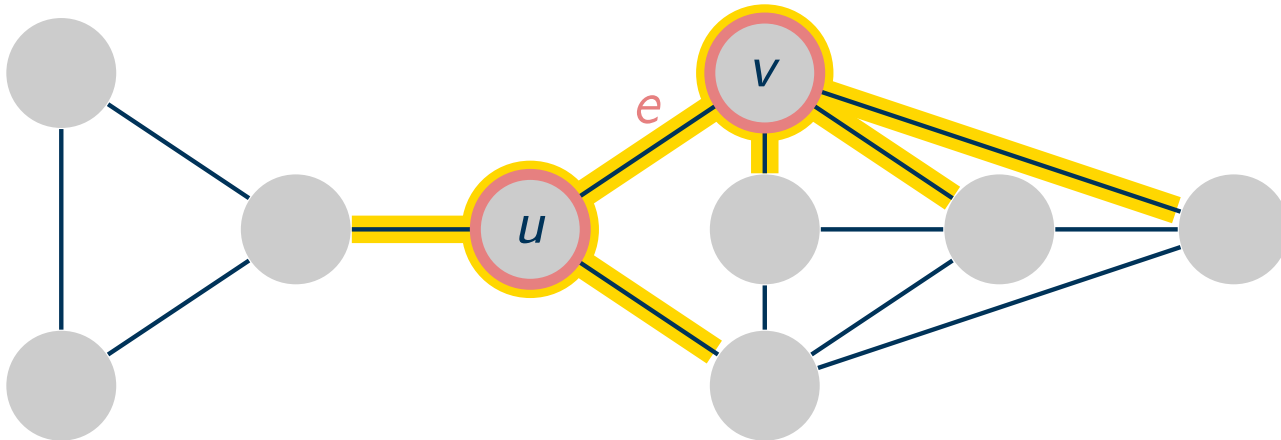
Heuristic Reduction Steps



Choose an edge $e = \{u, v\}$ to put in the matching and shrink the graph.

HEU(rand)
- e is randomly chosen

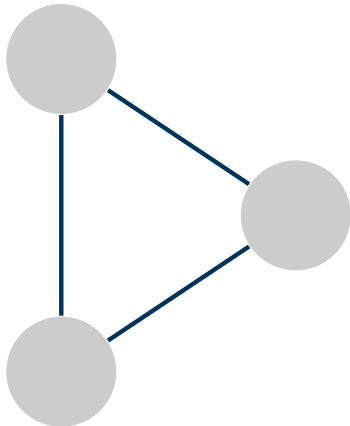
Heuristic Reduction Steps

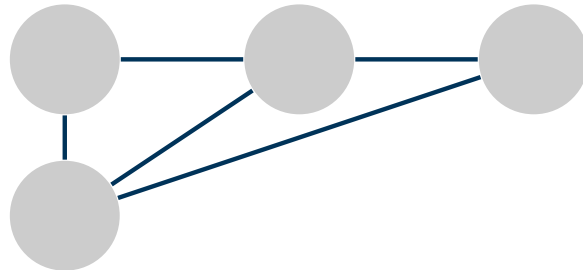


Choose an edge $e = \{u, v\}$ to put in the matching and shrink the graph.

HEU(rand)
- e is randomly chosen

Heuristic Reduction Steps

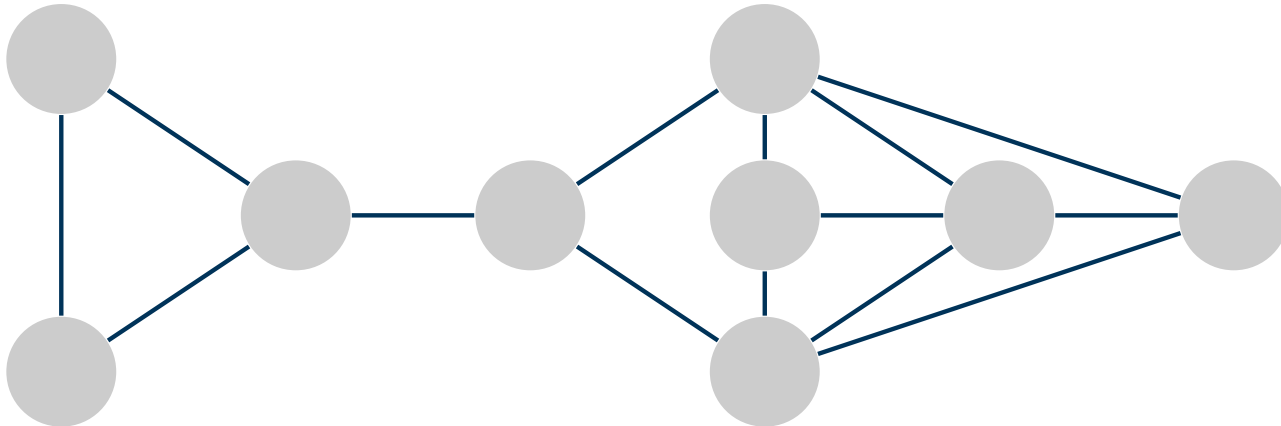




Choose an edge $e = \{u, v\}$ to put in the matching and shrink the graph.

HEU(rand)
- e is randomly chosen

Heuristic Reduction Steps

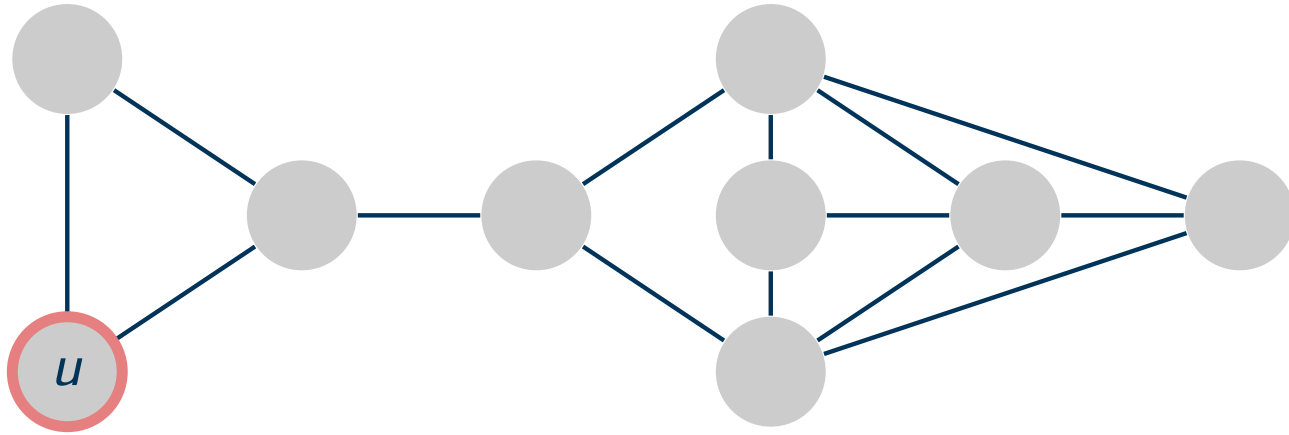


Choose an edge $e = \{u, v\}$ to put in the matching and shrink the graph.

HEU(rand) ▸ e is randomly chosen

HEU(deg,deg)

Heuristic Reduction Steps

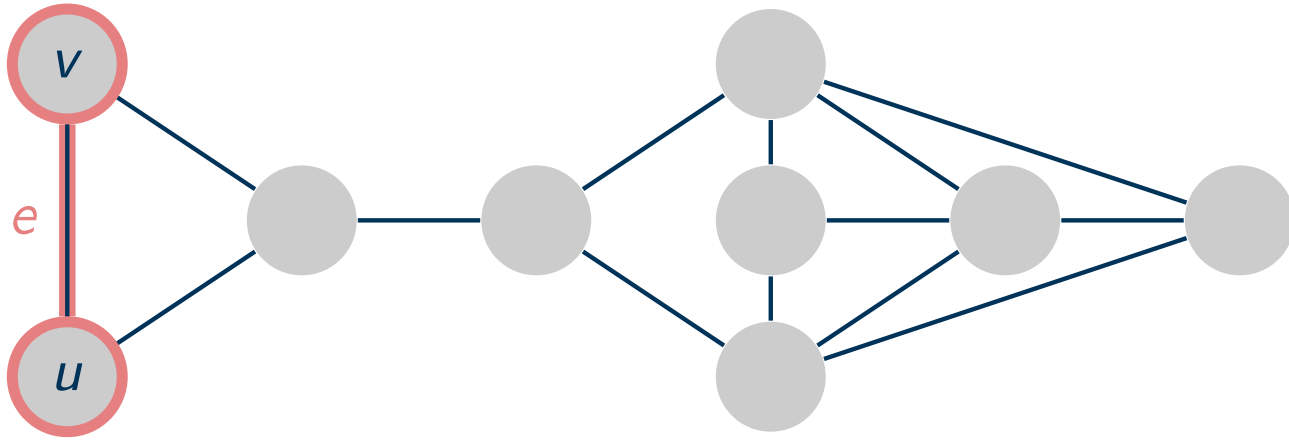


Choose an edge $e = \{u, v\}$ to put in the matching and shrink the graph.

HEU(rand) ▸ e is randomly chosen

HEU(deg,deg) ▸ u is a node of smallest degree

Heuristic Reduction Steps

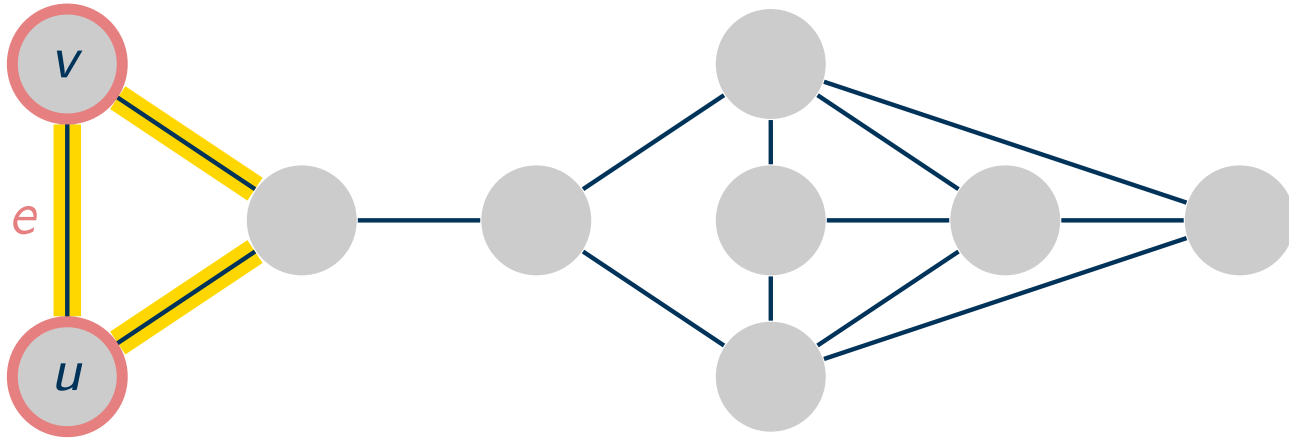


Choose an edge $e = \{u, v\}$ to put in the matching and shrink the graph.

HEU(rand)
- e is randomly chosen

HEU(deg,deg)
- u is a node of smallest degree
- v is a neighbor of u of smallest degree

Heuristic Reduction Steps

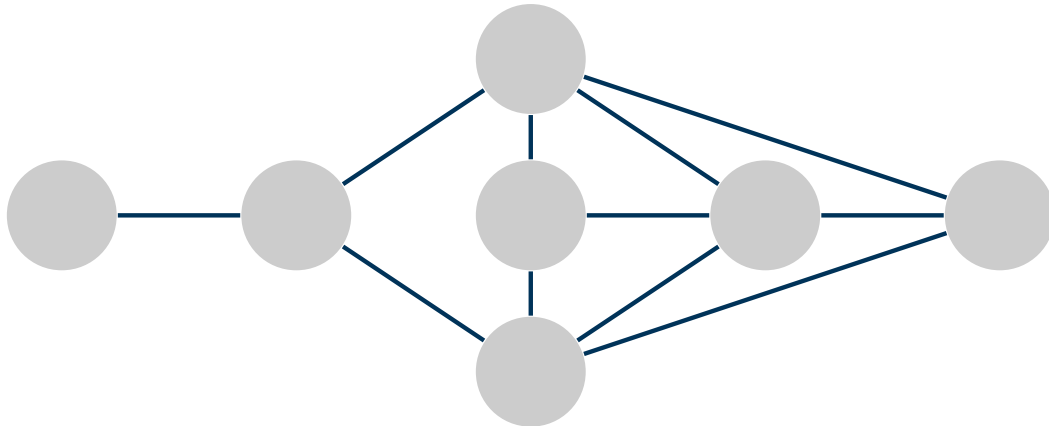


Choose an edge $e = \{u, v\}$ to put in the matching and shrink the graph.

HEU(rand)
- e is randomly chosen

HEU(deg,deg)
- u is a node of smallest degree
- v is a neighbor of u of smallest degree

Heuristic Reduction Steps

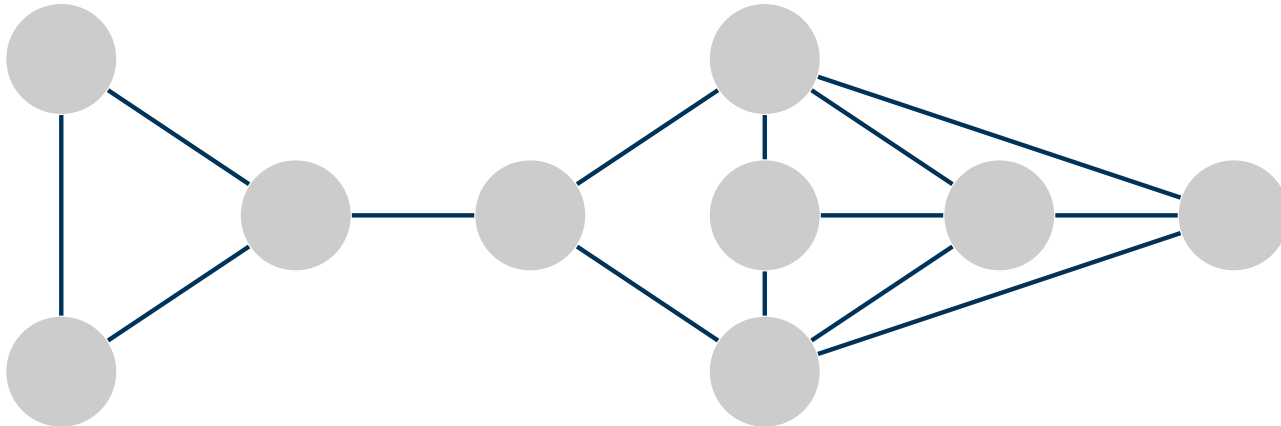


Choose an edge $e = \{u, v\}$ to put in the matching and shrink the graph.

HEU(rand)
- e is randomly chosen

HEU(deg,deg)
- u is a node of smallest degree
- v is a neighbor of u of smallest degree

Heuristic Reduction Steps



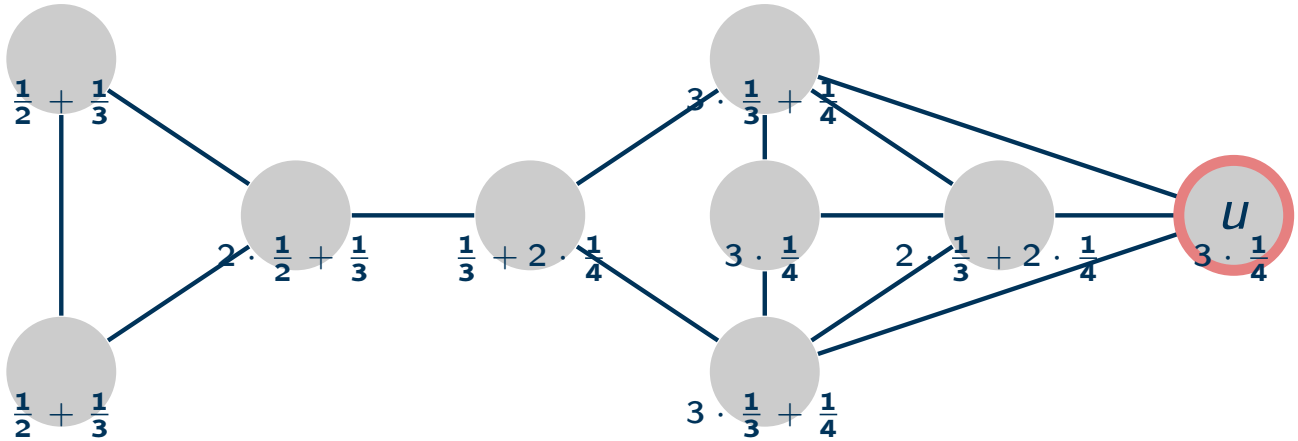
Choose an edge $e = \{u, v\}$ to put in the matching and shrink the graph.

HEU(rand)
- e is randomly chosen

HEU(deg,deg)
- u is a node of smallest degree
- v is a neighbor of u of smallest degree

HEU(pot,deg)

Heuristic Reduction Steps



Choose an edge $e = \{u, v\}$ to put in the matching and shrink the graph.

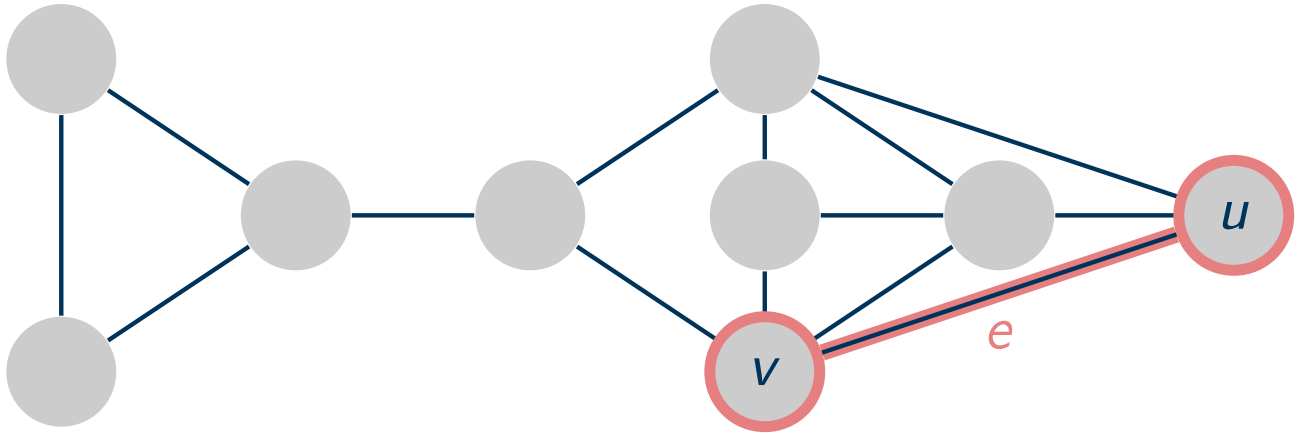
HEU(rand)
- e is randomly chosen

HEU(deg,deg)
- u is a node of smallest degree
- v is a neighbor of u of smallest degree

HEU(pot,deg)
- u is a node of smallest potential $\pi(u)$, where

$$\pi(u) := \sum_{\{u,v\}\in E} \frac{1}{\deg(v)}$$

Heuristic Reduction Steps



Choose an edge $e = \{u, v\}$ to put in the matching and shrink the graph.

HEU(rand)
- e is randomly chosen

HEU(deg,deg)
- u is a node of smallest degree
- v is a neighbor of u of smallest degree

HEU(pot,deg)
- u is a node of smallest potential $\pi(u)$, where

$$\pi(u) := \sum_{\{u,v\} \in E} \frac{1}{\deg(v)}$$

- v is a neighbor of u of smallest degree

Greedy Heuristics

"random edge":

- **OPT(1):HEU(rand)** — Karp-Sipser heuristic [Karp and Sipser, 1981]
- **OPT(1,2):HEU(rand)**

Greedy Heuristics

"random edge":

- **OPT(1):HEU(rand)** — Karp-Sipser heuristic [Karp and Sipser, 1981]
- **OPT(1,2):HEU(rand)**

"double minimum degree":

- **OPT(1):HEU(deg,deg)**
- **OPT(1,2):HEU(deg,deg)** — heuristic with highest quality of solution from [Magun, 1998]

Greedy Heuristics

"random edge":

- **OPT(1):HEU(rand)** — Karp-Sipser heuristic [Karp and Sipser, 1981]
- **OPT(1,2):HEU(rand)**

"double minimum degree":

- **OPT(1):HEU(deg,deg)**
- **OPT(1,2):HEU(deg,deg)** — heuristic with highest quality of solution from [Magun, 1998]

"minimum potential, minimum degree":
new algorithms — adaptation of selfless algorithm by [Sanders, 2004]

- **OPT(1):HEU(pot,deg)**
- **OPT(1,2):HEU(pot,deg)** — heuristic that probably has no critical region

Experimental Setup

graphs:

Experimental Setup

graphs:

- random graphs $G(n; c)$ with n nodes and edge probability $p = c/(n-1)$
- $n = 10^6$, $c \in [1, 10]$, step size 0.1

Experimental Setup

graphs:

- random graphs $G(n; c)$ with n nodes and edge probability $p = c/(n-1)$
- $n = 10^6$, $c \in [1, 10]$, step size 0.1

For results regarding graphs with fewer nodes and random bipartite graphs — see paper and the full version on arXiv.

Experimental Setup

graphs:

- random graphs $G(n; c)$ with n nodes and edge probability $p = c/(n-1)$
- $n = 10^6$, $c \in [1, 10]$, step size 0.1

For results regarding graphs with fewer nodes and random bipartite graphs — see paper and the full version on arXiv.

Experimental Setup

graphs:

- random graphs $G(n; c)$ with n nodes and edge probability $p = c/(n-1)$
- $n = 10^6$, $c \in [1, 10]$, step size 0.1

For results regarding graphs with fewer nodes and random bipartite graphs — see paper and the full version on arXiv.

measurements:

Experimental Setup

graphs:

- random graphs $G(n; c)$ with n nodes and edge probability $p = c/(n-1)$
- $n = 10^6$, $c \in [1, 10]$, step size 0.1

For results regarding graphs with fewer nodes and random bipartite graphs — see paper and the full version on arXiv.

measurements:

λ: failure rate — fraction of graphs where matching found is not a maximum matching

Experimental Setup

graphs:

- random graphs $G(n; c)$ with n nodes and edge probability $p = c/(n-1)$
- $n = 10^6$, $c \in [1, 10]$, step size 0.1

For results regarding graphs with fewer nodes and random bipartite graphs — see paper and the full version on arXiv.

measurements:

λ: failure rate — fraction of graphs where matching found is not a maximum matching

ρ: number of lost edges — average number of edges missing from maximum matching if failure occurs

Experimental Setup

graphs:

- random graphs $G(n; c)$ with n nodes and edge probability $p = c/(n-1)$
- $n = 10^6$, $c \in [1, 10]$, step size 0.1

For results regarding graphs with fewer nodes and random bipartite graphs — see paper and the full version on arXiv.

measurements:

λ: failure rate — fraction of graphs where matching found is not a maximum matching

ρ: number of lost edges — average number of edges missing from maximum matching if failure occurs

$\bar{t}$: avg. running time

Experimental Setup

graphs:

- random graphs $G(n; c)$ with n nodes and edge probability $p = c/(n-1)$
- $n = 10^6$, $c \in [1, 10]$, step size 0.1

For results regarding graphs with fewer nodes and random bipartite graphs — see paper and the full version on arXiv.

measurements:

λ: failure rate — fraction of graphs where matching found is not a maximum matching

$\bar{t}$: avg. running time

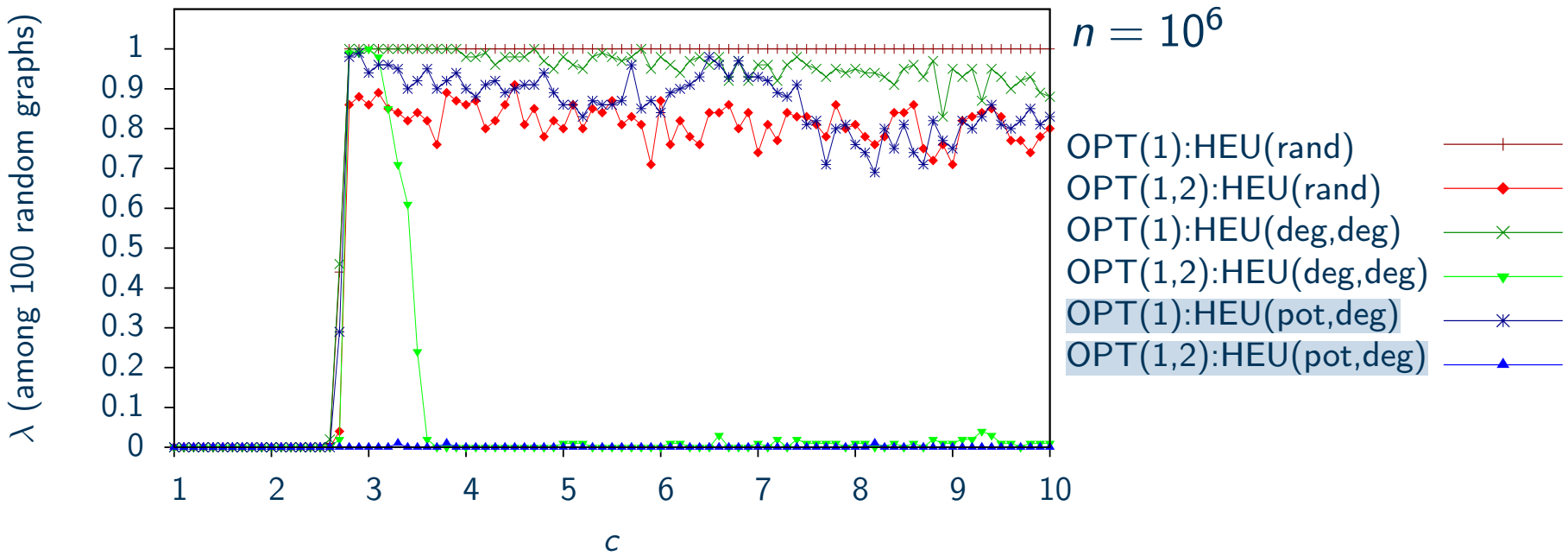
ρ: number of lost edges — average number of edges missing from maximum matching if failure occurs

$\overline{\#o1}$: avg. fraction of **OPT(1)** steps

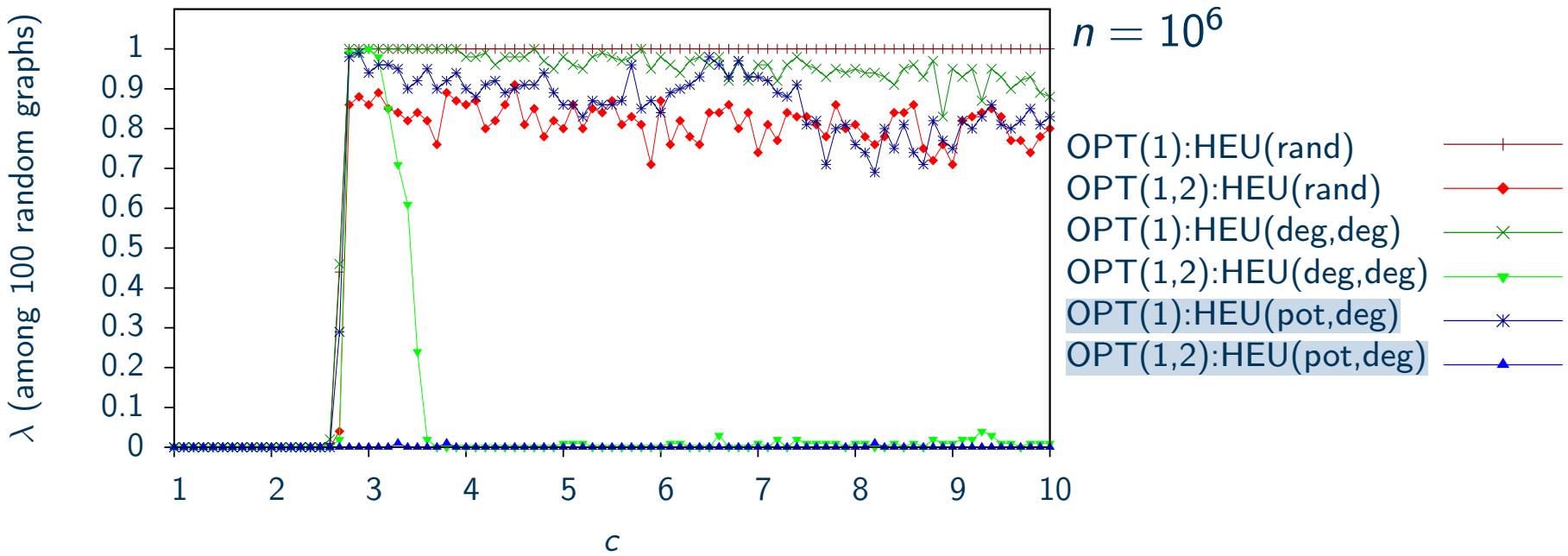
$\overline{\#o2}$: avg. fraction of **OPT(2)** steps

$\overline{\#h}$: avg. fraction of **HEU($\star$)** steps

Failure Rate

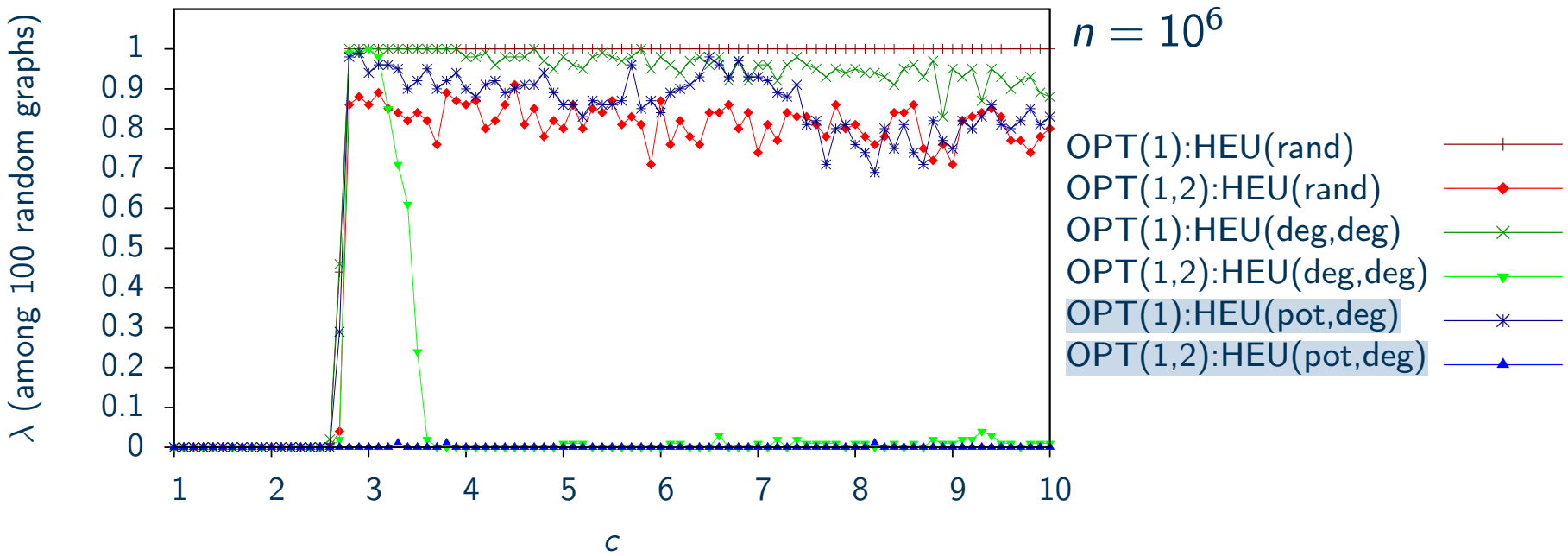


Failure Rate



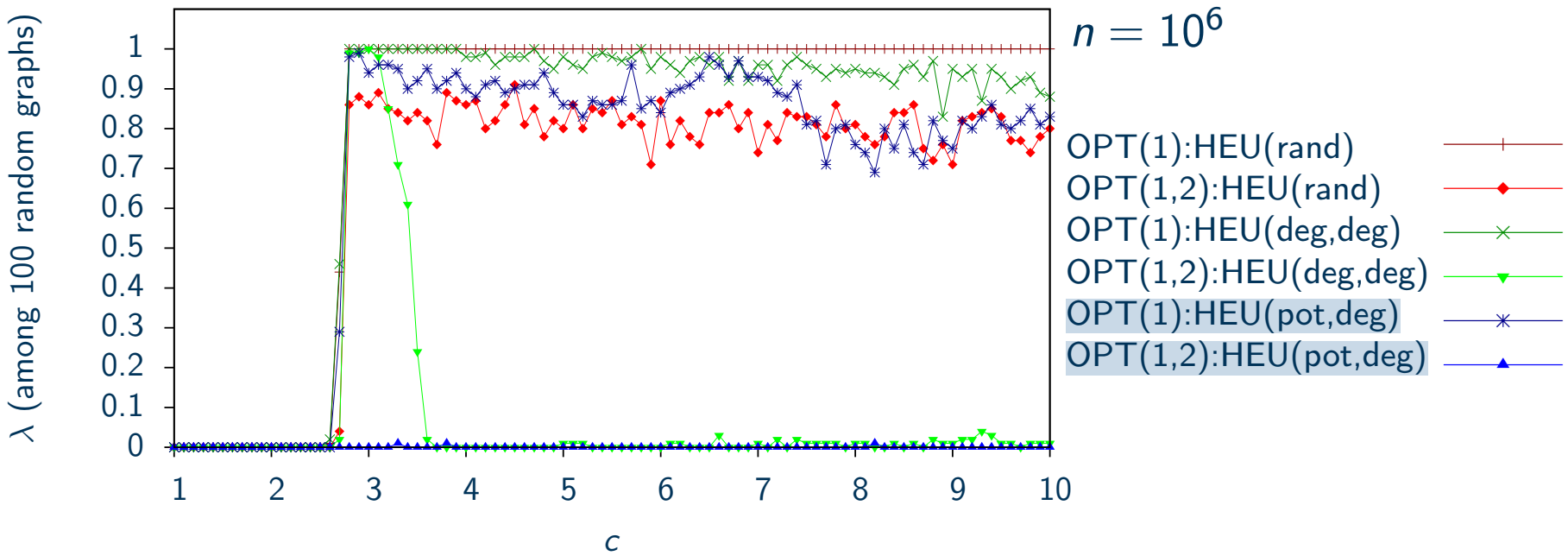
- For $c \leq 2.5$ no failure occurred in any of the heuristics. (well known behavior — almost only **OPT(1)** reductions)

Failure Rate



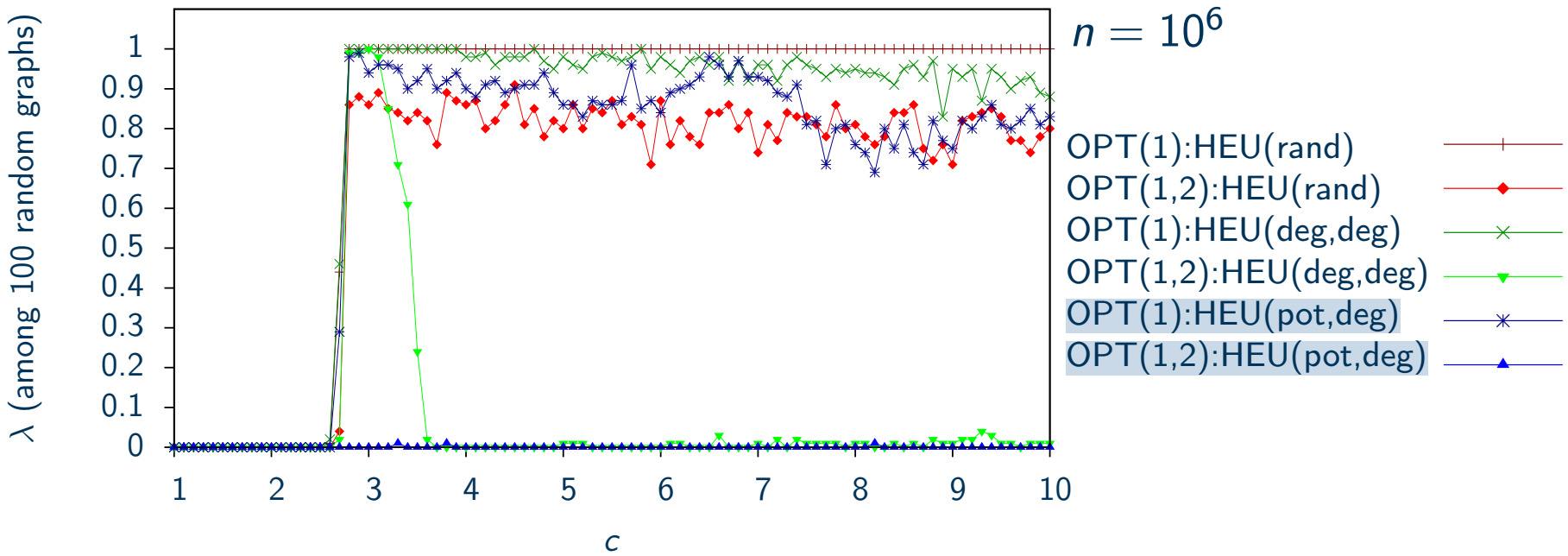
- For $c \leq 2.5$ no failure occurred in any of the heuristics. (well known behavior — almost only **OPT(1)** reductions)
- **OPT(2)** does not influence "begin of failure" much.

Failure Rate



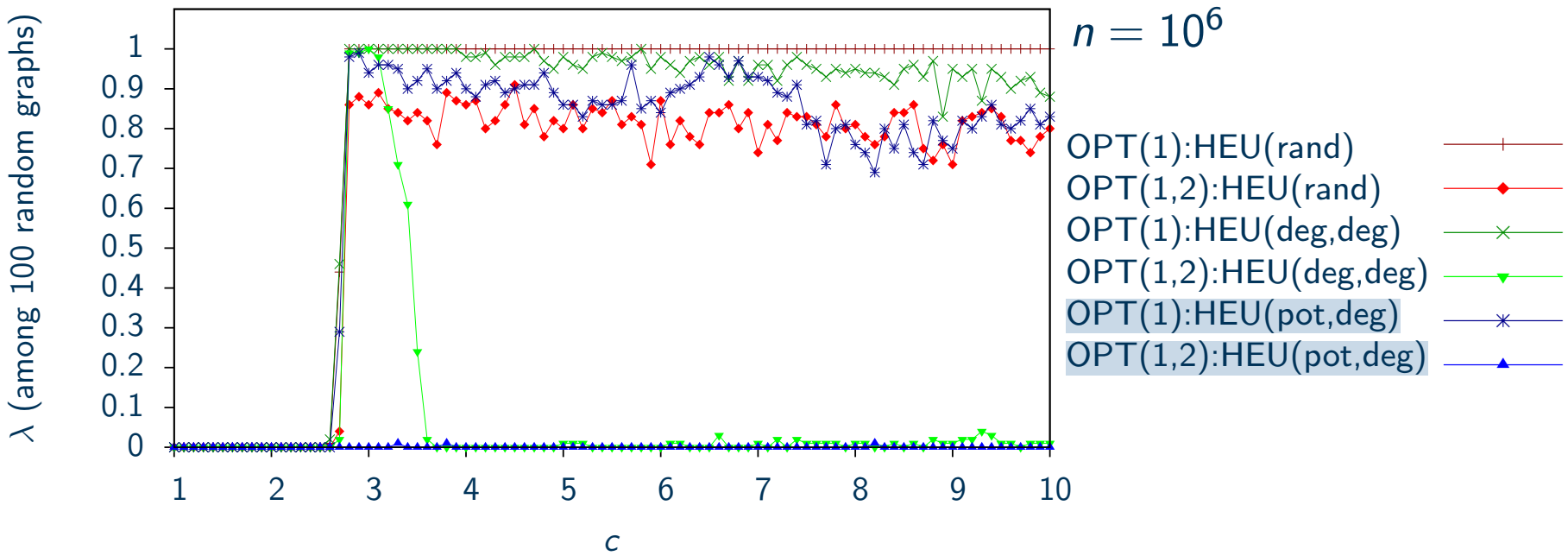
- For $c \leq 2.5$ no failure occurred in any of the heuristics. (well known behavior — almost only **OPT(1)** reductions)
- **OPT(2)** does not influence "begin of failure" much.
- Heuristics with **OPT(1,2)** outperform counterparts using **OPT(1)**.

Failure Rate



- For $c \leq 2.5$ no failure occurred in any of the heuristics. (well known behavior — almost only **OPT(1)** reductions)
- **OPT(2)** does not influence "begin of failure" much.
- Heuristics with **OPT(1,2)** outperform counterparts using **OPT(1)**.
- critical region of $2.6 < c < 3.7$ for **OPT(1,2):HEU(deg,deg)** reproduced

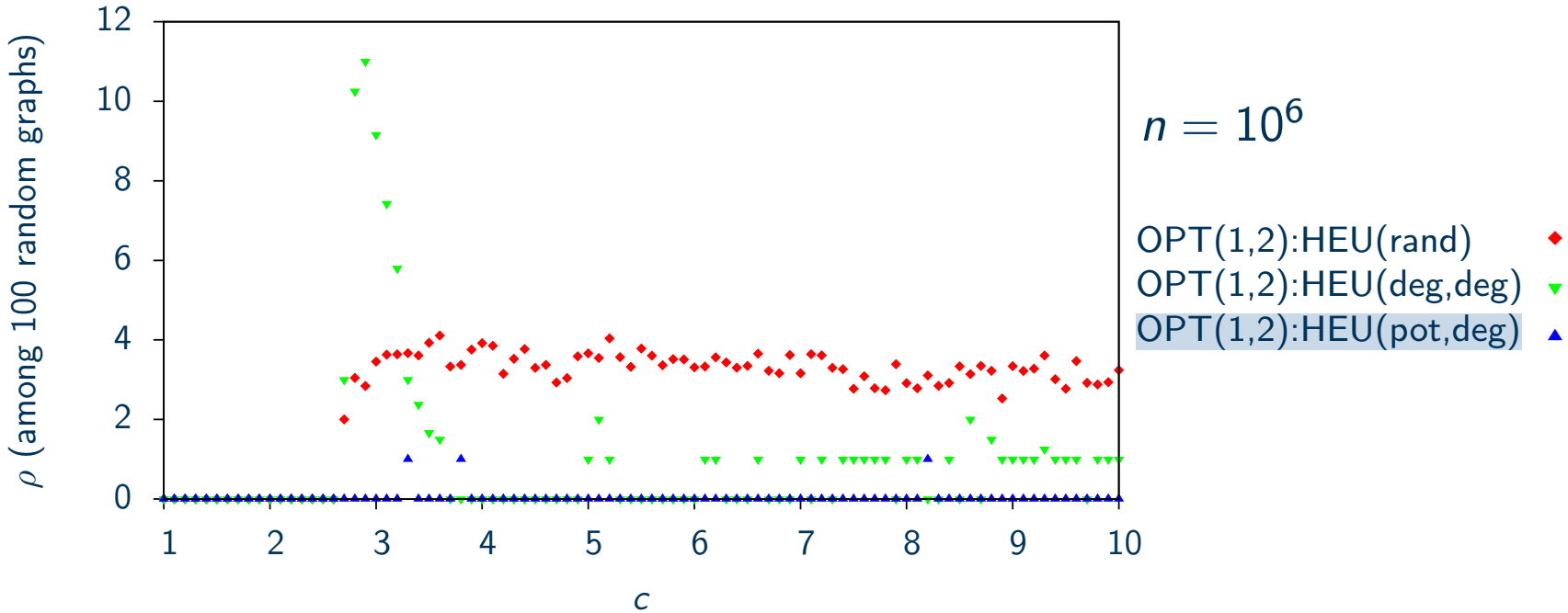
Failure Rate



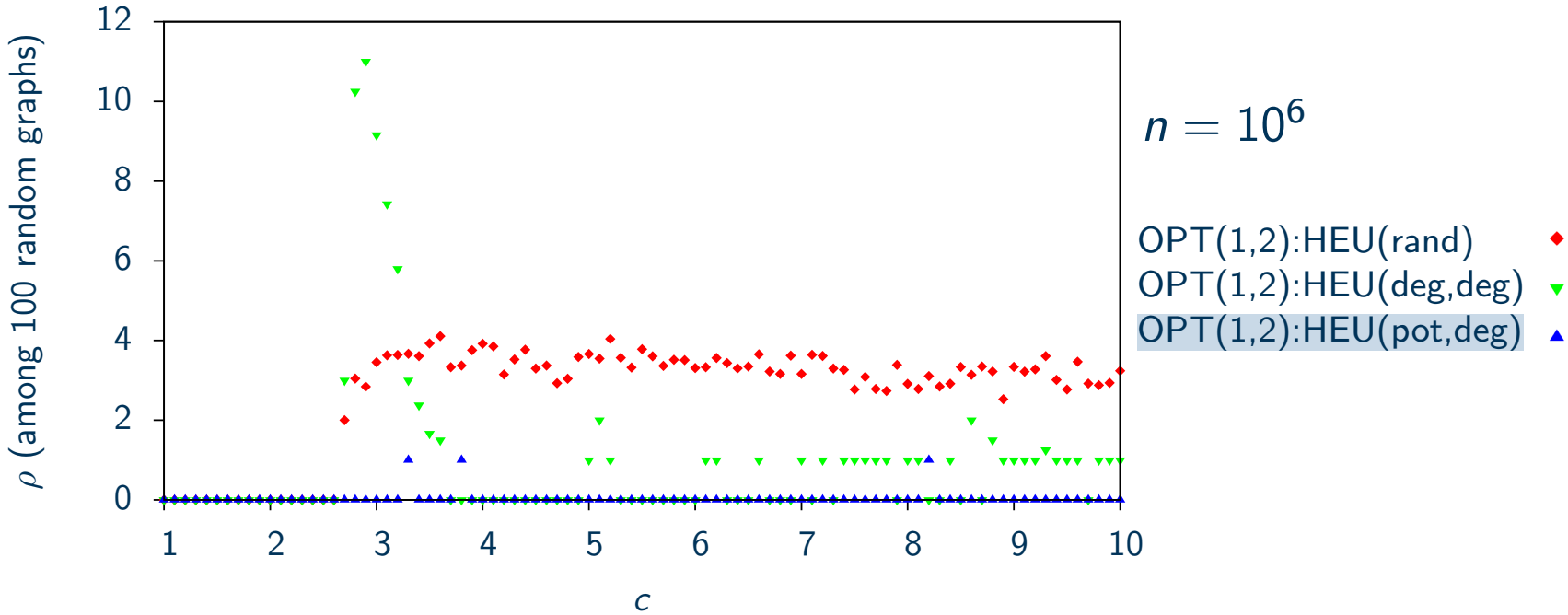
- For $c \leq 2.5$ no failure occurred in any of the heuristics. (well known behavior — almost only **OPT(1)** reductions)
- **OPT(2)** does not influence "begin of failure" much.
- Heuristics with **OPT(1,2)** outperform counterparts using **OPT(1)**.
- critical region of $2.6 < c < 3.7$ for **OPT(1,2):HEU(deg,deg)** reproduced

OPT(1,2):HEU(pot,deg) fails only 3 times out of 9100.

Number of Lost Edges

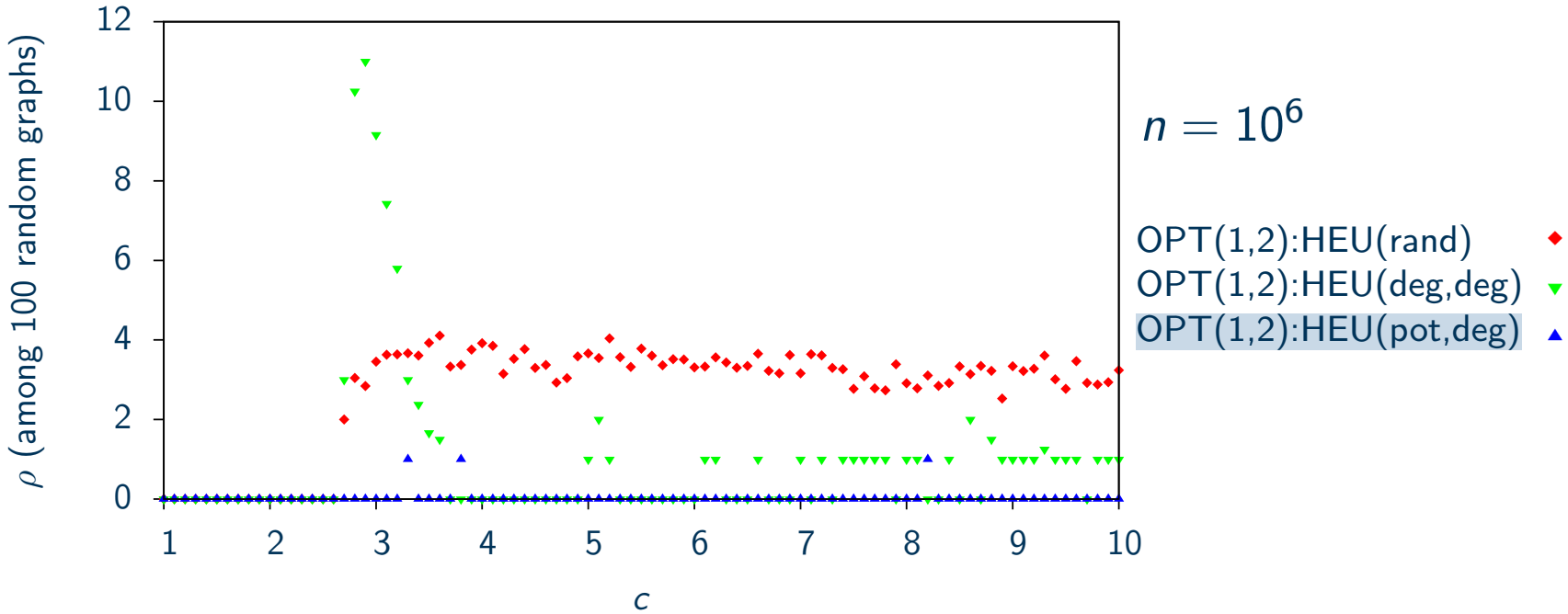


Number of Lost Edges



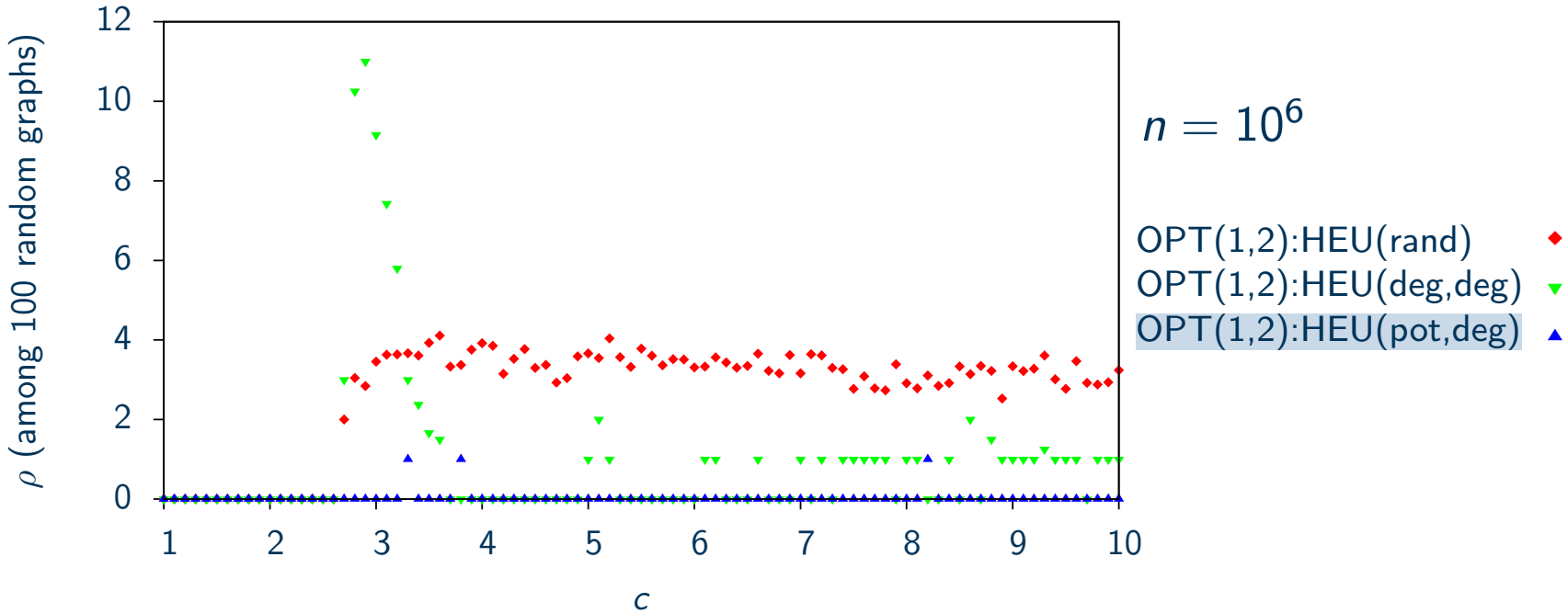
- From $c \geq 2.7$ number of lost edges for **OPT(1,2):HEU(rand)** is smaller than $n^{1/5}$ (and relatively stable).

Number of Lost Edges



- From $c \geq 2.7$ number of lost edges for **OPT(1,2):HEU(rand)** is smaller than $n^{1/5}$ (and relatively stable).
- Outside its critical range **OPT(1,2):HEU(deg,deg)** loses mostly zero or one edge.

Number of Lost Edges



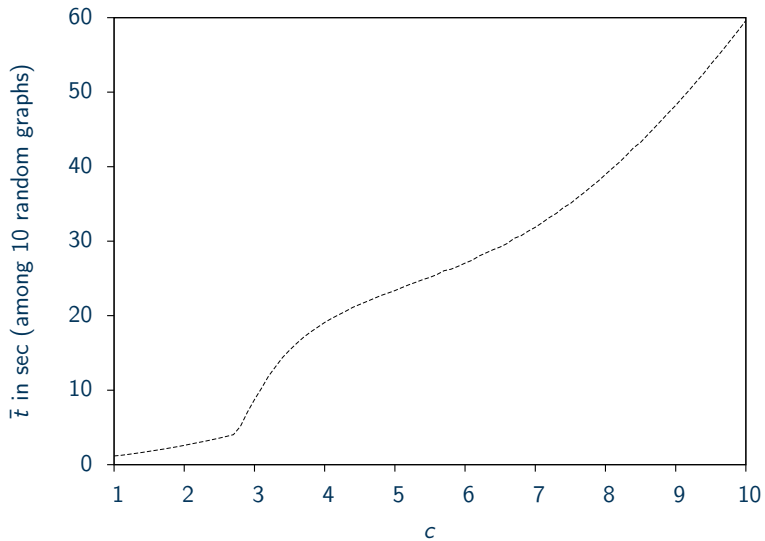
- From $c \geq 2.7$ number of lost edges for **OPT(1,2):HEU(rand)** is smaller than $n^{1/5}$ (and relatively stable).
- Outside its critical range **OPT(1,2):HEU(deg,deg)** loses mostly zero or one edge.

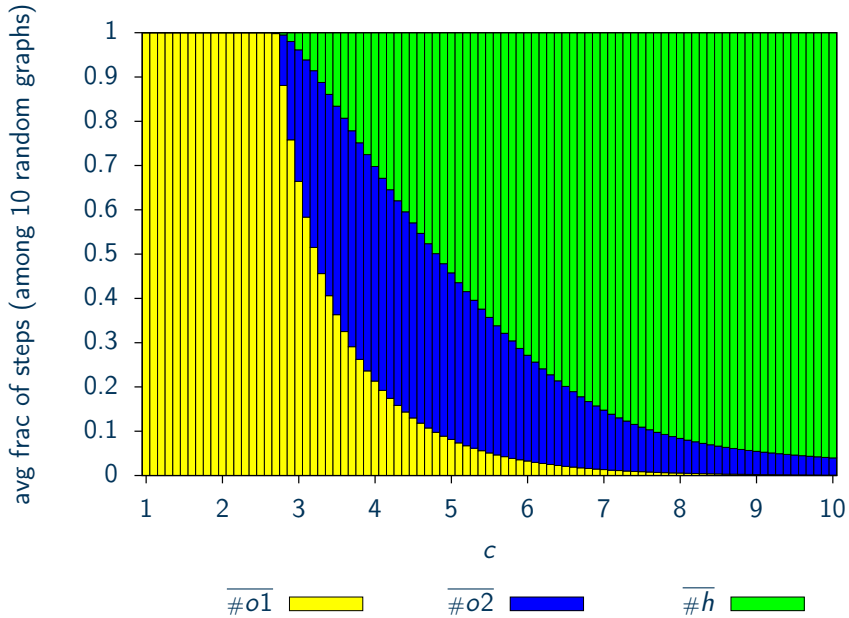
OPT(1,2):HEU(pot,deg) loses one edge 3 times out of 9100.

Running Time Behavior

$n = 10^6$

OPT(1,2):HEU(pot,deg)

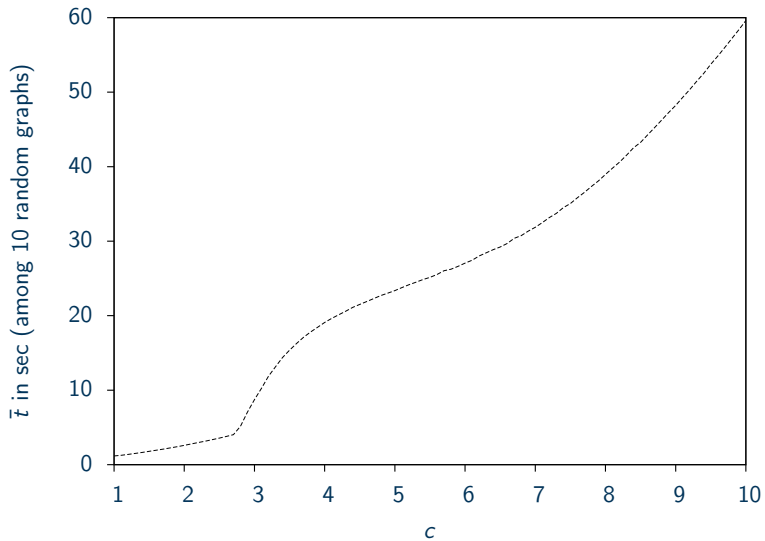


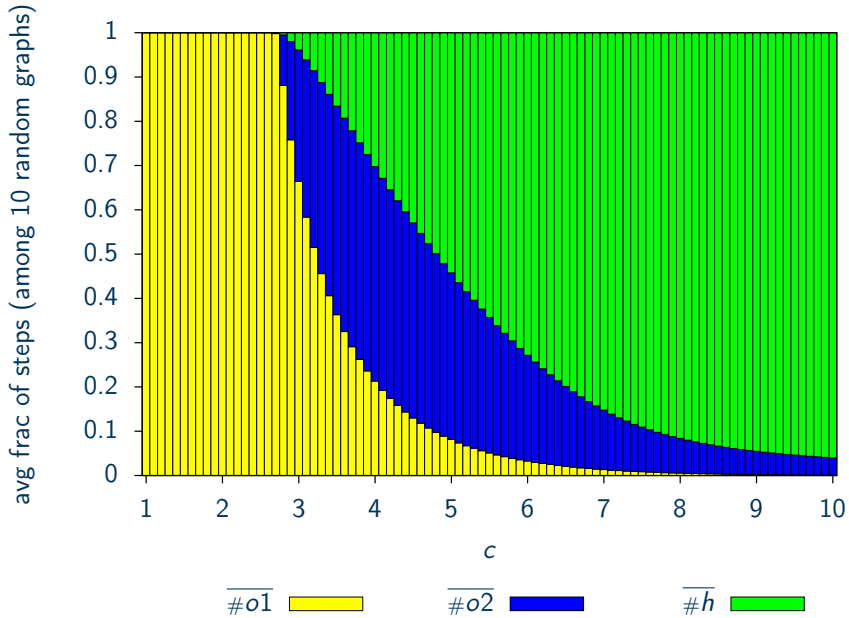


Running Time Behavior

$n = 10^6$

OPT(1,2):HEU(pot,deg)



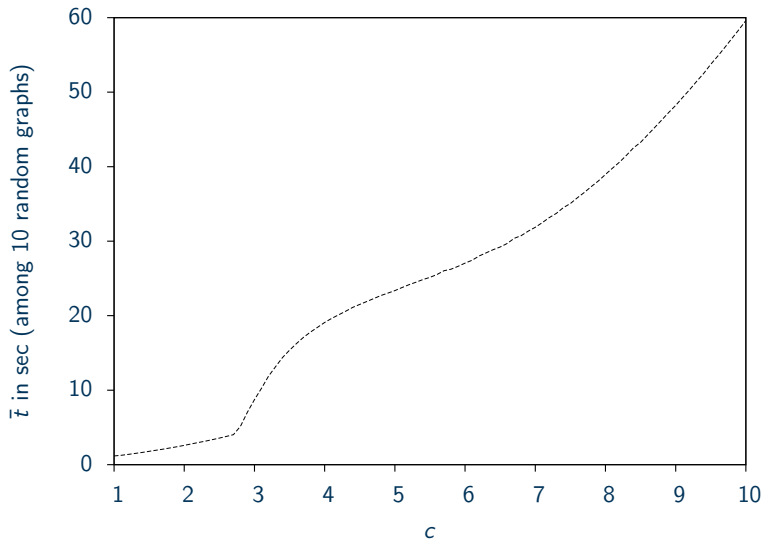


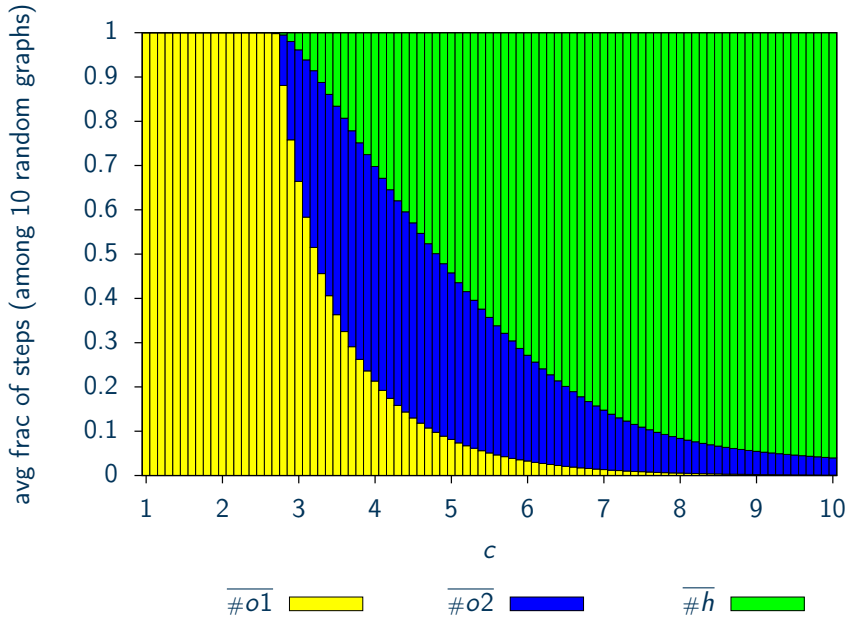
- $1 \leq c \leq 2.7$: linear slope — almost only **OPT(1)** steps

Running Time Behavior

$n = 10^6$

OPT(1,2):HEU(pot,deg)



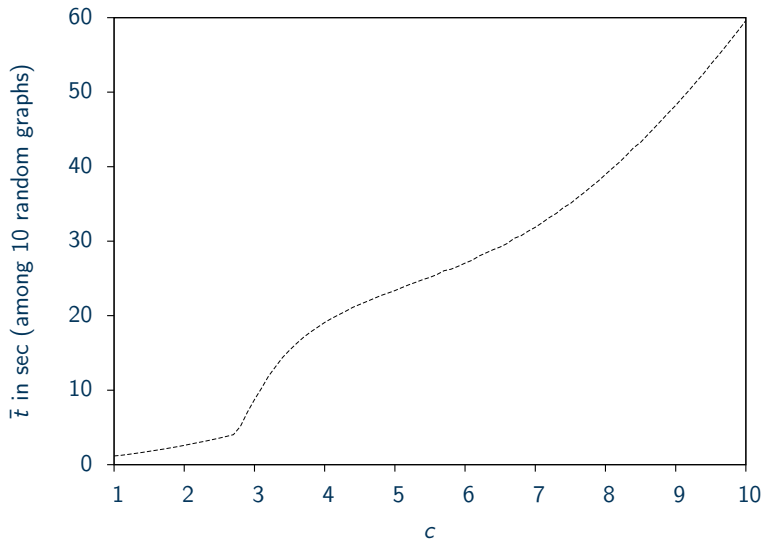


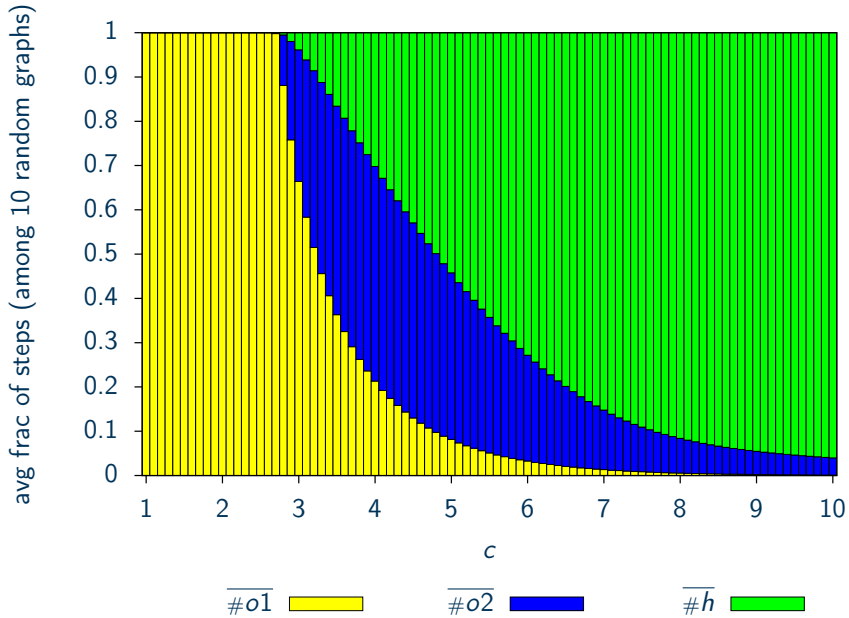
- $1 \leq c \leq 2.7$: linear slope — almost only **OPT(1)** steps
- $2.7 < c < 6$: non-linear slope — strong decrease of $\overline{\#o1}$, non-linear increase of $\overline{\#o2}$ and linear increase of $\overline{\#h}$

Running Time Behavior

$n = 10^6$

OPT(1,2):HEU(pot,deg)





- $1 \leq c \leq 2.7$: linear slope — almost only **OPT(1)** steps
- $2.7 < c < 6$: non-linear slope — strong decrease of $\overline{\#o1}$, non-linear increase of $\overline{\#o2}$ and linear increase of $\overline{\#h}$
- $c \geq 6$: slightly non-linear slope — dominated by $\overline{\#h}$

Running Time Behavior

exemplary comparison with exact algorithm:

$\bar{t}$ in seconds (average among 10 random graphs), $n = 10^6$

c	$\bar{t}(H)$	$\bar{t}(E)$
1		
5		
9		

H: **OPT(1,2):HEU(pot,deg)**

E: Edmonds' algorithm from Boost C++

Running Time Behavior

exemplary comparison with exact algorithm:

$\bar{t}$ in seconds (average among 10 random graphs), $n = 10^6$

c	$\bar{t}(H)$	$\bar{t}(E)$
1	1.1	73.1
5	23.5	948.6
9	48.4	1216.5

H: **OPT(1,2):HEU(pot,deg)**

E: Edmonds' algorithm from Boost C++

Running Time Behavior

exemplary comparison with exact algorithm:

$\bar{t}$ in seconds (average among 10 random graphs), $n = 10^6$

c	$\bar{t}(H)$	$\bar{t}(E)$	$\bar{t}(E+i)$
1	1.1	73.1	0.8
5	23.5	948.6	6.6
9	48.4	1216.5	8.3

H: **OPT(1,2):HEU(pot,deg)**

E: Edmonds' algorithm from Boost C++

$+i$: with initial matching from **OPT(1):HEU(rand)**

Running Time Behavior

exemplary comparison with exact algorithm:

$\bar{t}$ in seconds (average among 10 random graphs), $n = 10^6$

c	$\bar{t}(H)$	$\bar{t}(E)$	$\bar{t}(E+i)$
1	1.1	73.1	0.8
5	23.5	948.6	6.6
9	48.4	1216.5	8.3

H: **OPT(1,2):HEU(pot,deg)**

E: Edmonds' algorithm from Boost C++

$+i$: with initial matching from **OPT(1):HEU(rand)**

OPT(2) and **HEU(pot,deg)** steps are slow!
(At least in our implementation.)

Conclusion

We proposed a new greedy heuristic for maximum cardinality matchings on sparse random graphs. The algorithm showed a very low failure rate in experiments.

Conclusion

We proposed a new greedy heuristic for maximum cardinality matchings on sparse random graphs. The algorithm showed a very low failure rate in experiments.

More (detailed) experimental results can be found in the paper or the full version on arXiv.

Conclusion

We proposed a new greedy heuristic for maximum cardinality matchings on sparse random graphs. The algorithm showed a very low failure rate in experiments.

More (detailed) experimental results can be found in the paper or the full version on arXiv.

Future work:

- Prove that this behavior is to be expected.

Conclusion

We proposed a new greedy heuristic for maximum cardinality matchings on sparse random graphs. The algorithm showed a very low failure rate in experiments.

More (detailed) experimental results can be found in the paper or the full version on arXiv.

Future work:

- Prove that this behavior is to be expected.
- Improve the running time behavior.

Conclusion

We proposed a new greedy heuristic for maximum cardinality matchings on sparse random graphs. The algorithm showed a very low failure rate in experiments.

More (detailed) experimental results can be found in the paper or the full version on arXiv.

Future work:

- Prove that this behavior is to be expected.
- Improve the running time behavior.
- Study performance on other classes of sparse random graphs, e.g., with no almost perfect matching [Bordenave et al., 2011].

Conclusion

We proposed a new greedy heuristic for maximum cardinality matchings on sparse random graphs. The algorithm showed a very low failure rate in experiments.

More (detailed) experimental results can be found in the paper or the full version on arXiv.

Future work:

- Prove that this behavior is to be expected.
- Improve the running time behavior.
- Study performance on other classes of sparse random graphs, e.g., with no almost perfect matching [Bordenave et al., 2011].
- Apply "selfless approach" to other (harder) problems, like graph coloring.

Thank you!

References (1)

Aronson, J., Frieze, A. M., and Pittel, B. (1998).
Maximum matchings in sparse random graphs: Karp-Sipser revisited.
Random Struct. Algorithms, 12(2):111–177.

Bast, H., Mehlhorn, K., Schäfer, G., and Tamaki, H. (2006).
Matching Algorithms Are Fast in Sparse Random Graphs.
Theory Comput. Syst., 39(1):3–14.

Bordenave, C., Lelarge, M., and Salez, J. (2011).
Matchings on infinite graphs.
CoRR, arXiv:1102.0712.

Chebolu, P., Frieze, A. M., and Melsted, P. (2010).
Finding a Maximum Matching in a Sparse Random Graph in $O(n)$ Expected Time.
J. ACM, 57(4).

Edmonds, J. (1965).
Paths, trees, and flowers.
Canadian Journal of Mathematics, 17:449–467.

References (2)

Karp, R. M. and Sipser, M. (1981).
Maximum Matchings in Sparse Random Graphs.
In *Proc. 22nd FOCS*, pages 364–375. IEEE Computer Society.

Magun, J. (1998).
Greedy Matching Algorithms: An Experimental Study.
ACM Journal of Experimental Algorithmics, 3:6.

Micali, S. and Vazirani, V. V. (1980).
An $O(\sqrt{|v|} \cdot |E|)$ Algorithm for Finding Maximum Matching in General Graphs.
In *Proc. 21st FOCS*, pages 17–27. IEEE Computer Society.

Mucha, M. and Sankowski, P. (2004).
Maximum Matchings via Gaussian Elimination.
In *Proc. 45th FOCS*, pages 248–255. IEEE Computer Society.

Sanders, P. (2004).
Algorithms for Scalable Storage Servers.
In *Proc. 30th SOFSEM*, LNCS, pages 82–101. Springer.