Computing A Near-Maximum Independent Set in Linear Time by Reducing-Peeling

Never Stand Still | Engineering | Computer Science and Engineering

Lijun Chang

University of New South Wales, Australia
Lijun.Chang@unsw.edu.au

Joint work with Wei Li, Wenjie Zhang

Outline

- **Introduction**
- Existing Works
- Our Reducing-Peeling Framework
- Our Approaches
- Experimental Studies
- Conclusion

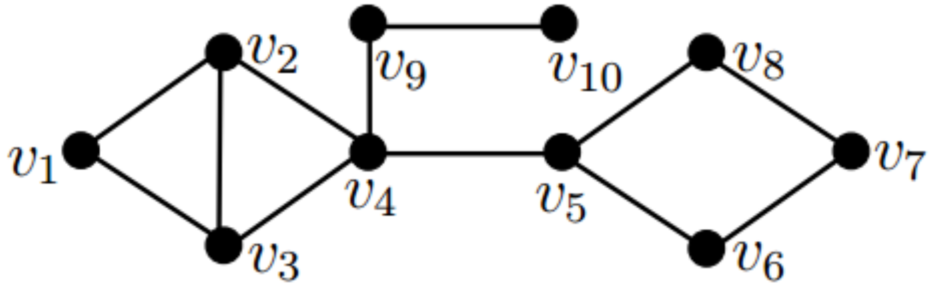
Introduction

Independent Set

Given a graph $G = (V, E)$, a vertex subset $I \subseteq V$ is an independent set if for any two vertices u and v in I, there is no edge between u and v in G.

Maximum Independent Set

An independent set I of G is a maximum independent set if its size is the largest among all independent sets of G.



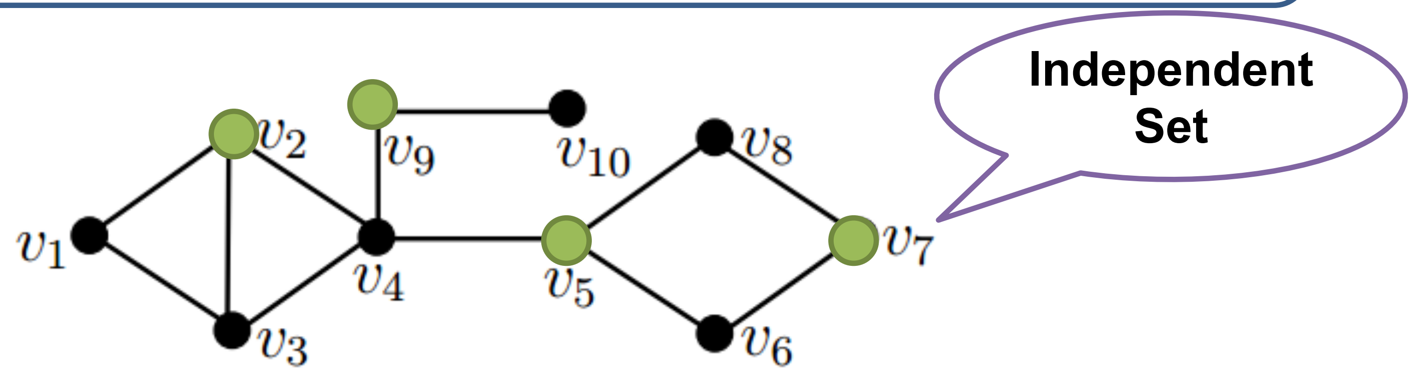
Introduction

Independent Set

Given a graph $G = (V, E)$, a vertex subset $I \subseteq V$ is an independent set if for any two vertices u and v in I, there is no edge between u and v in G.

Maximum Independent Set

An independent set I of G is a maximum independent set if its size is the largest among all independent sets of G.



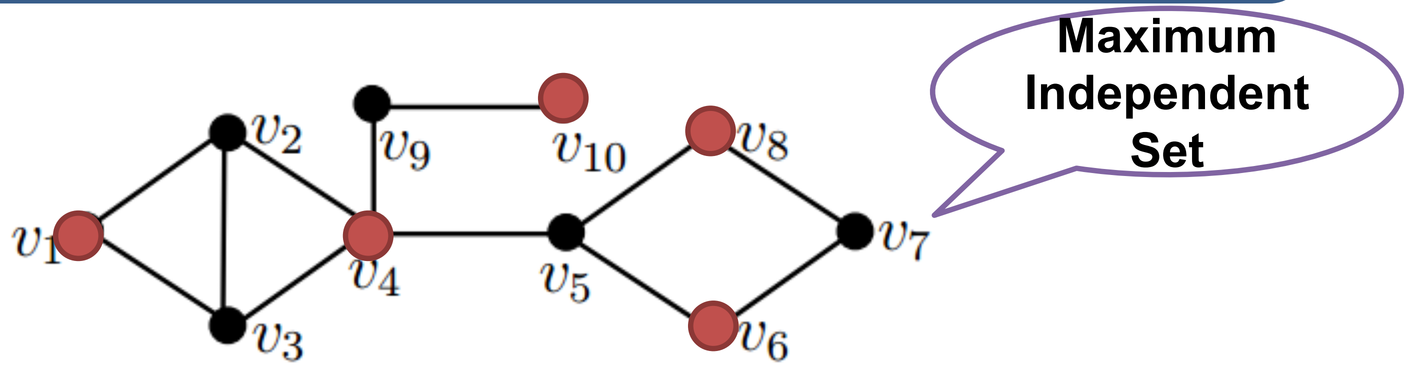
Introduction

Independent Set

Given a graph $G = (V, E)$, a vertex subset $I \subseteq V$ is an independent set if for any two vertices u and v in I, there is no edge between u and v in G.

Maximum Independent Set

An independent set I of G is a maximum independent set if its size is the largest among all independent sets of G.



Introduction

Applications

- Build index for shortest path/distance queries [Cheng et al. *SIGMOD'12*, Fu et al. *VLDB'13*]
- Refine the result of matching two graphs [Zhu et al. *VLDB J'13*]
- Social network coverage [Puthal et al. *BigData'15*]; vertex cover

Hardness

- NP-hard to compute a maximum independent set [Garey et al. *Book'79*]
- Hard to approximate
  - NP-hard to approximate within a factor of $n^{1-\varepsilon}$ for any $0 < \varepsilon < 1$ [J. Håstad. *FOCS'96*]

Outline

- ❑ Introduction
- ❑ **Existing Works**
- ❑ Our Reducing-Peeling Framework
- ❑ Our Approaches
- ❑ Experimental Studies
- ❑ Conclusion

UNSW AUSTRALIA

Existing Works

Exact algorithms -- *branch-and-reduce paradigm*

- [F. V. Fomin et al .*J.ACM'09*]
  - Theoretically runs in $O^*(1.2201^n)$ time
- [T. Akiba et al. *Theor. Comput. Sci.'16*]
  - Practically computes the exact solution for many small and medium-sized graphs

Approximation algorithms

- [U. Feige *J. Discrete Math'04*, M. M. Halldórsson et al. *Algorithmica'97*, P. Berman. *Theor.Comput. Sys.'99*]
  - Approximation ratio largely depends on n or Δ
  - Not practically useful

Existing Works

Heuristic algorithms for large graphs

- Linear-time algorithms
  - Greedy, dynamic update
  - Efficient, but can only find small independent sets in practice
- Iterative randomized searching
  - Local search algorithm: **ARW** [D. V. Andrade. *J.Heuristics'12*]
  - Evolutionary algorithm: **ReduMIS** [S. Lamm. *ALENEX'16*]
  - Local search + simple reduction rules: **OnlineMIS** [J. Dahlum. *SEA'16*]
  - Can find large independent sets, but take long time

Our goal: find large independent sets in a time-efficient and space-effective manner

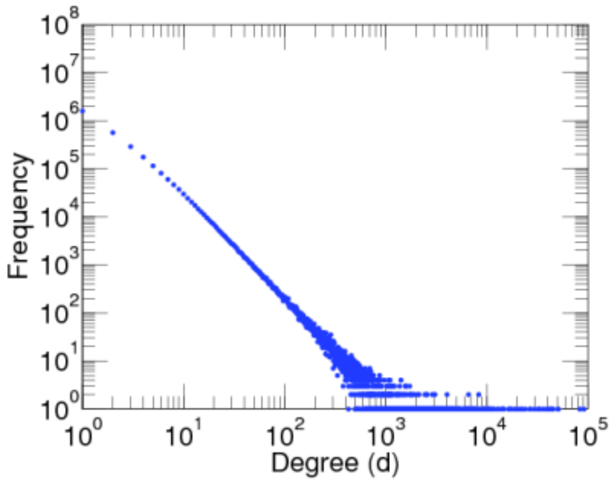
Outline

- Introduction
- Existing Works
- **Our Reducing-Peeling Framework**
- Our Approaches
- Experimental Studies
- Conclusion

Three Observations Utilized in Our Framework

- **Observation–I:** Real networks are usually power-law graphs with many low-degree vertices

$$\boldsymbol{Pr(deg = k) \propto \frac{1}{k^{\beta}}}$$



- Observation-II: Reduction rules have been effectively used for low-degree vertices

- Observation-III: High-degree vertices are less likely to be in a maximum independent set

Three Observations Utilized in Our Framework

- Observation–I: Real networks are usually power-law graphs with many low-degree vertices

- **Observation-II:** Reduction rules have been effectively used for low-degree vertices

Degree-one Reduction

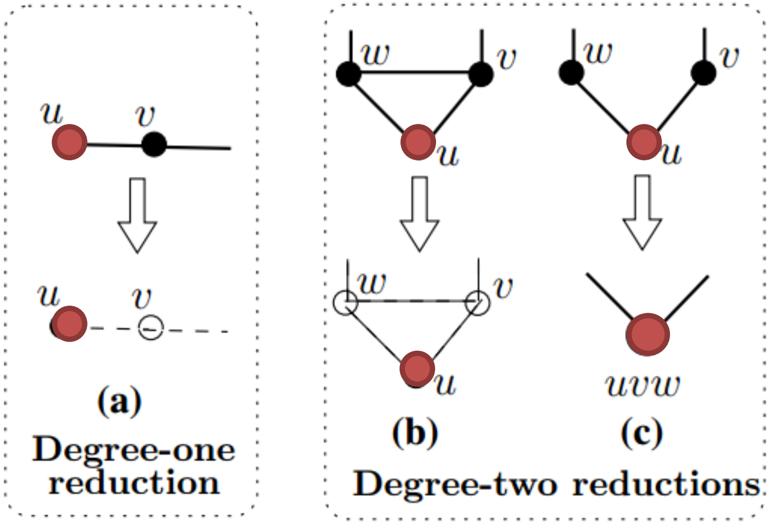
(a) $\alpha(G) = \alpha(G \backslash \{v\})$

$\alpha(G)$: $independence\ number\ of\ G$

Degree-two Reductions

(b) Isolation $\alpha(G) = \alpha(G \backslash \{v, w\})$

(c) Folding $\alpha(G) = \alpha(G / \{u, v, w\}) + 1$



- Observation-III: High-degree vertices are less likely to be in a maximum independent set

Three Observations Utilized in Our Framework

- Observation–I: Real networks are usually power-law graphs with many low-degree vertices

- Observation-II: Reduction rules have been effectively used for low-degree vertices

- **Observation-III:** High-degree vertices are less likely to be in a maximum independent set

  - If a high-degree vertex is added into the independent set, then all its neighbors, which are of a large quantity, are ruled out from the independent set [J. Dahlum et al *SEA'16*]

  - Removing/peeling high-degree vertices can further sparsify the graph [Y. Lim et al *TKDE'14*]

The Reducing-Peeling Framework

Definition 3.1: (Inexact Reduction) Given a graph G, we remove/peel the vertex with the highest degree from G.

- Phase 1: **Reducing**
  - **While** a reduction rule can be applied on a vertex u **then**
    Apply the exact reduction rule on u

- Phase 2: **Peeling**
  - Apply the inexact reduction rule to temporarily remove a high-degree vertex

- Repeat the above two phases until there is no edge in the graph

- Post-process: Iteratively add a temporarily removed vertex to the solution if the independence requirement is not violated

Outline

- ❑ Introduction
- ❑ Existing Works
- ❑ Our Reducing-Peeling Framework
- ❑ **Our Approaches**
- ❑ Experimental Studies
- ❑ Conclusion

Overview of Our Approaches

❖ Compute large independent set for large graphs in a *time-efficient* and *space-effective* manner

- Subquadratic (or even linear) time.
- $2m + O(n)$ space: m is the number of undirected edges.
  - A graph is stored in $2m + n + O(1)$ space by the adjacency array (aka, Compressed Sparse Row) graph representation
  - A graph with one billion edges takes slightly more than *8GB* memory

Algorithm	Time Complexity	Space Complexity	Exact Reduction Rules Used
BDOne	$O(m)$	$2m + O(n)$	Degree-one reduction [21]
BDTwo	$O(n \times m)$	$6m + O(n)$	Degree-one reduction [21] & Degree-two vertex reductions [21]
LinearTime	$O(m)$	$2m + O(n)$	Degree-one reduction [21] & Degree-two path reduction (this paper)
NearLinear	$O(m \times \Delta)$	$4m + O(n)$	Dominance reduction [21] & Degree-two path reduction (this paper)

Table 1: Overview of our approaches (n: number of vertices, m: number of edges, Δ: maximum vertex degree)

An Efficient Baseline Algorithm

- **BDOne**

Step 1:

While $V_{=1} \neq \emptyset$ or $V_{\geq 2} \neq \emptyset$

If $V_{=1} \neq \emptyset$ **then**

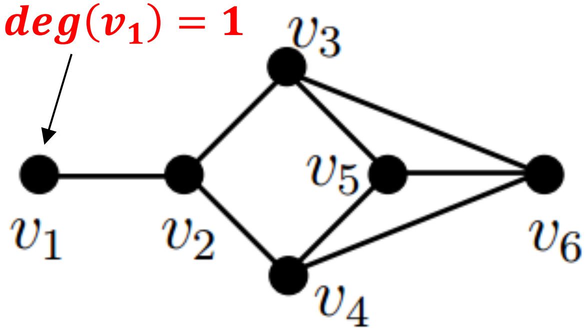
DegreeOne-Reduction

Else

Inexact-Reduction

Step 2:

Recover temporarily removed vertices



An Efficient Baseline Algorithm

❖ **BDOne**

Step 1:

While $V_{=1} \neq \emptyset$ or $V_{\geq 2} \neq \emptyset$

If $V_{=1} \neq \emptyset$ **then**

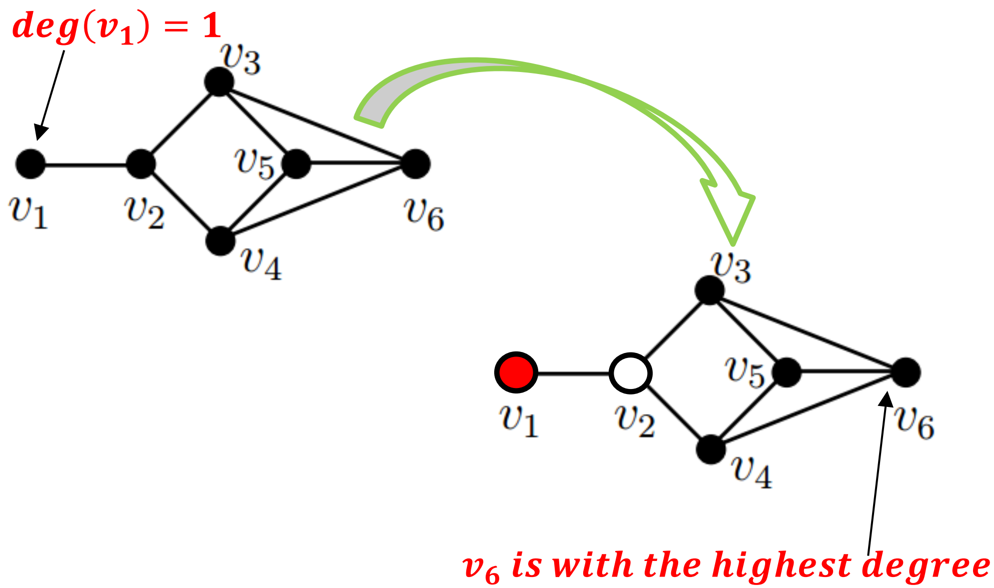
DegreeOne-Reduction

Else

Inexact-Reduction

Step 2:

Recover temporarily removed vertices



An Efficient Baseline Algorithm

- **BDOne**

Step 1:

While $V_{=1} \neq \emptyset$ or $V_{\geq 2} \neq \emptyset$

If $V_{=1} \neq \emptyset$ **then**

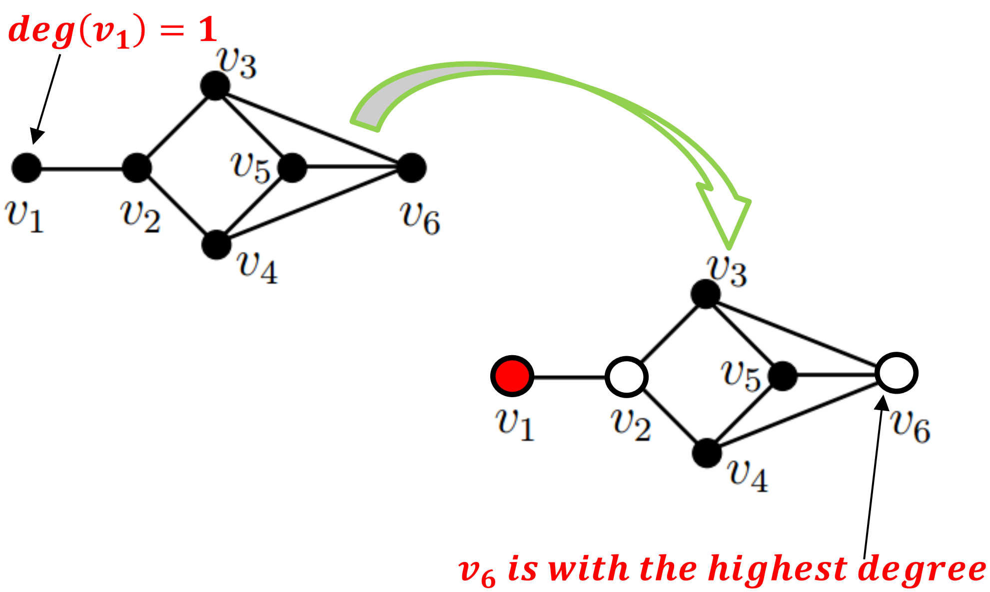
DegreeOne-Reduction

Else

Inexact-Reduction

Step 2:

Recover temporarily removed vertices



An Efficient Baseline Algorithm

❖ **BDOne**

Step 1:

While $V_{=1} \neq \emptyset$ or $V_{\geq 2} \neq \emptyset$

If $V_{=1} \neq \emptyset$ **then**

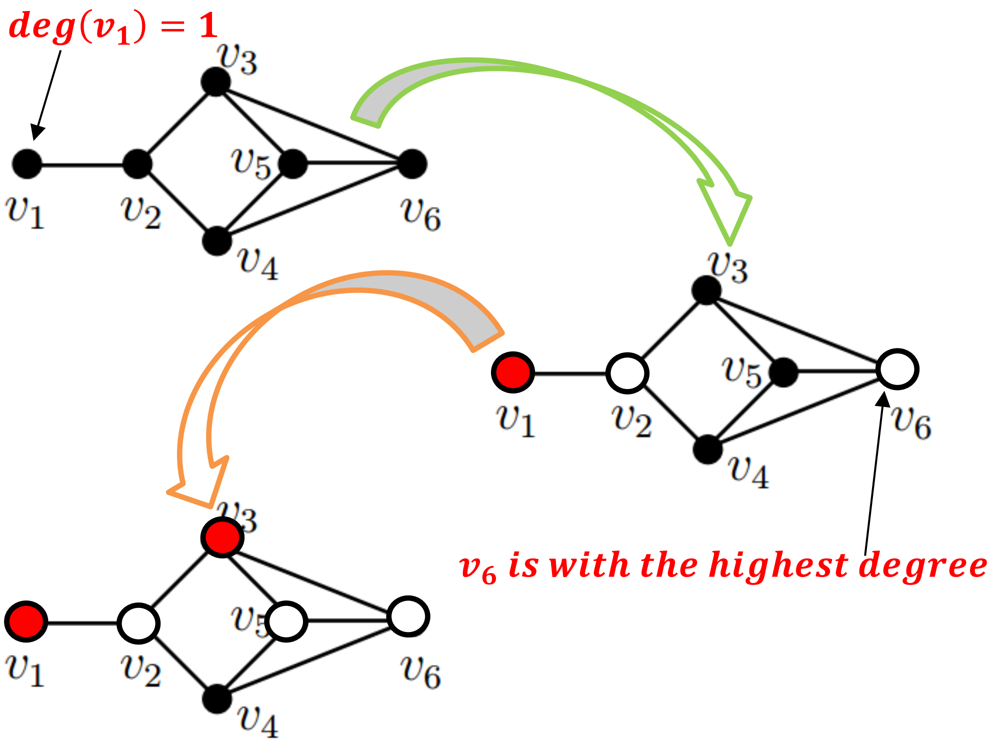
DegreeOne-Reduction

Else

Inexact-Reduction

Step 2:

Recover temporarily removed vertices



An Efficient Baseline Algorithm

- **BDOne**

Step 1:

While $V_{=1} \neq \emptyset$ or $V_{\geq 2} \neq \emptyset$

If $V_{=1} \neq \emptyset$ **then**

DegreeOne-Reduction

Else

Inexact-Reduction

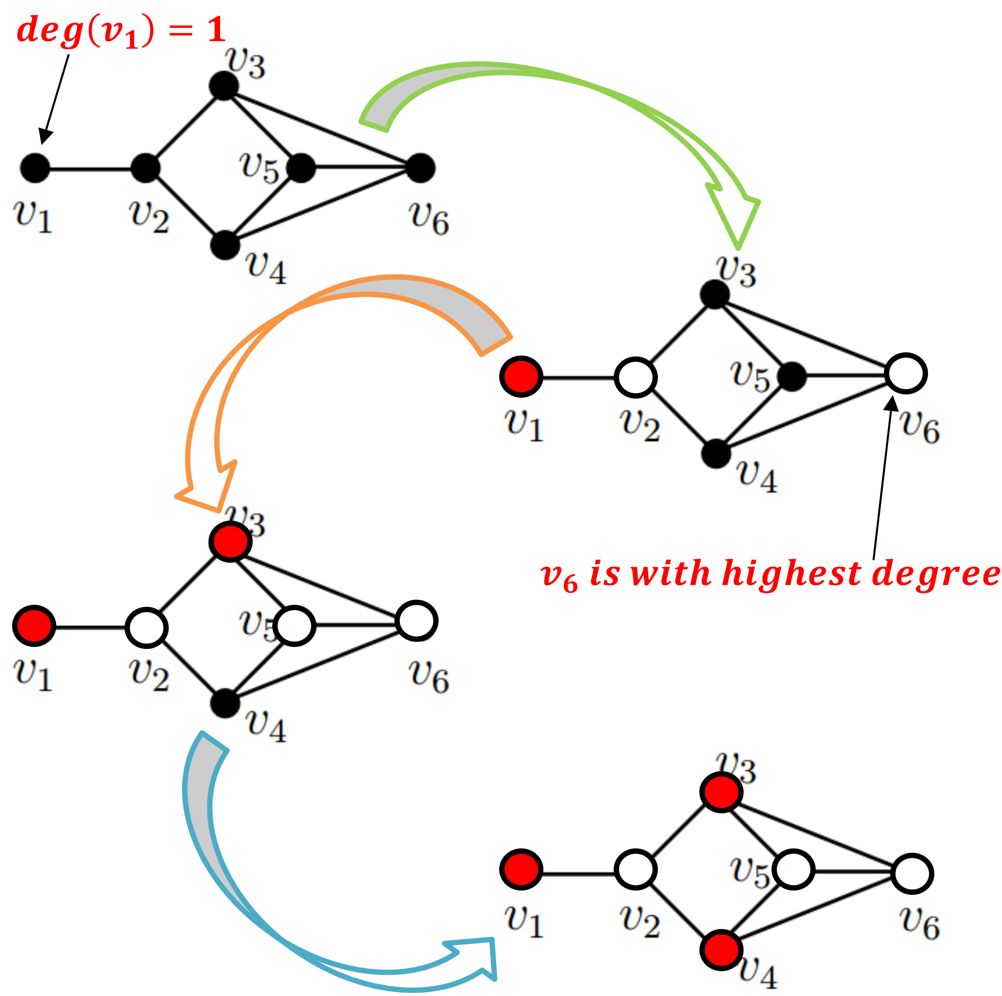
Step 2:

Recover temporarily removed vertices

Complexity Analysis

Time: $O(m)$

Space: $2m + O(n)$



An Effective Baseline Algorithm

- **BDTwo**

Step 1:

While $V_{=1} \neq \emptyset$ or $V_{=2} \neq \emptyset$ or $V_{\geq 3} \neq \emptyset$

If $V_{=1} \neq \emptyset$ **then**

DegreeOne-Reduction

Else if $V_{=2} \neq \emptyset$ **then**

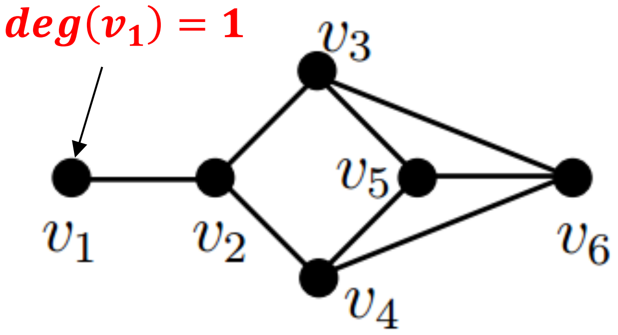
DegreeTwo-Reduction

Else

Inexact-Reduction

Step 2:

Recover temporarily removed vertices



An Effective Baseline Algorithm

- **BDTwo**

Step 1:

While $V_{=1} \neq \emptyset$ or $V_{=2} \neq \emptyset$ or $V_{\geq 3} \neq \emptyset$

  If $V_{=1} \neq \emptyset$ **then**

    DegreeOne-Reduction

  Else if $V_{=2} \neq \emptyset$ **then**

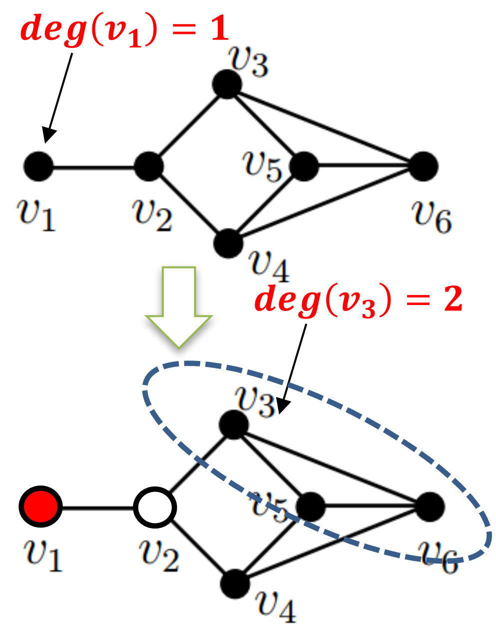
    DegreeTwo-Reduction

  Else

    Inexact-Reduction

Step 2:

Recover temporarily removed vertices



An Effective Baseline Algorithm

- **BDTwo**

Step 1:

While $V_{=1} \neq \emptyset$ or $V_{=2} \neq \emptyset$ or $V_{\geq 3} \neq \emptyset$

If $V_{=1} \neq \emptyset$ **then**

DegreeOne-Reduction

Else if $V_{=2} \neq \emptyset$ **then**

DegreeTwo-Reduction

Else

Inexact-Reduction

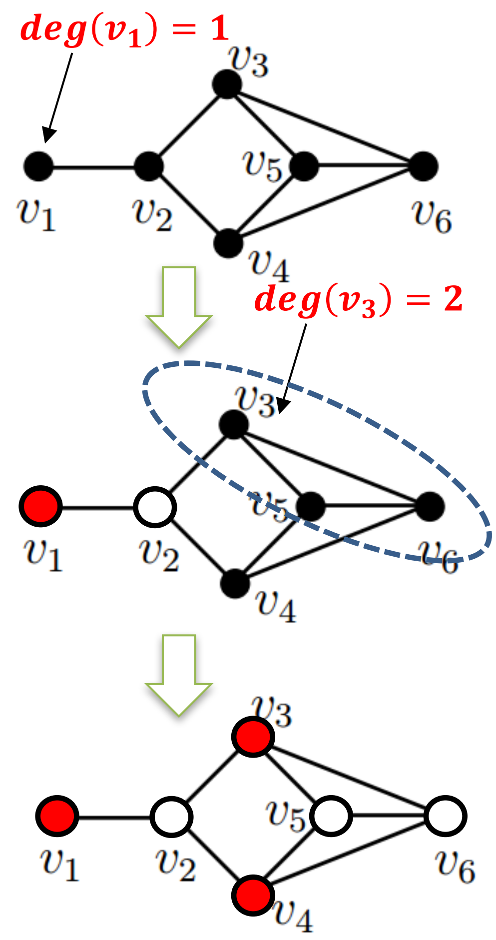
Step 2:

Recover temporarily removed vertices

Complexity Analysis

Time: $O(n \times m)$ and $\Omega(m + nlogn)$

Space: $6m + O(n)$



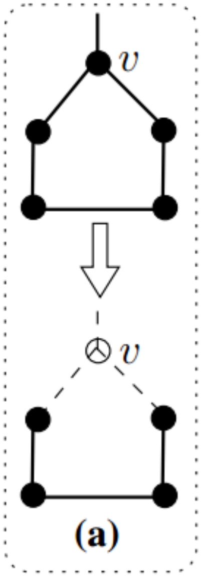
An Effective Linear-Time Algorithm

❖ **LinearTime**

Lemma 4.1: (Degree-two Path Reductions) Consider a graph $G = (V, E)$ with minimum degree two. For a maximal degree-two path $P = \{v_1, v_2, \dots, v_l\}$, let $v \notin P$ and $w \notin P$ be the unique vertices connected to v_1 and v_l, respectively.

Case 1: $v = w$

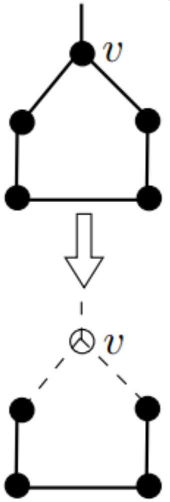
$\Longrightarrow \alpha(G) = \alpha(G \backslash \{v\})$



An Effective Linear-Time Algorithm

❖ **LinearTime**

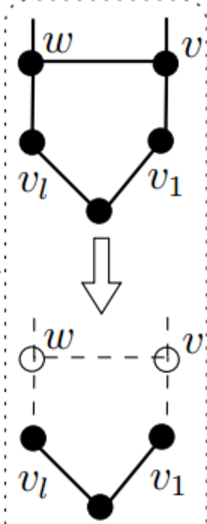
Lemma 4.1: (Degree-two Path Reductions) Consider a graph $G = (V, E)$ with minimum degree two. For a maximal degree-two path $P = \{v_1, v_2, \dots, v_l\}$, let $v \notin P$ and $w \notin P$ be the unique vertices connected to v_1 and v_l, respectively.



(a)

Case 2: $|P|$ is odd and $(v, w) \in E$

$$\Longrightarrow \alpha(G) = \alpha(G \backslash \{v, w\})$$

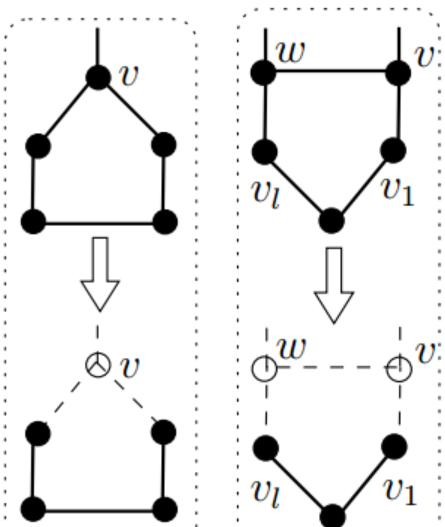


(b)

An Effective Linear-Time Algorithm

❖ **LinearTime**

Lemma 4.1: (Degree-two Path Reductions) Consider a graph $G = (V, E)$ with minimum degree two. For a maximal degree-two path $P = \{v_1, v_2, \dots, v_l\}$, let $v \notin P$ and $w \notin P$ be the unique vertices connected to v_1 and v_l, respectively.

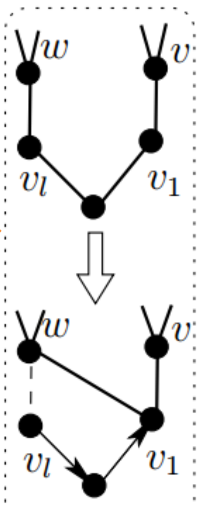


(a) (b)

Case 3: $|P|$ is odd and $(v, w) \notin E$

$$\Longrightarrow \alpha(G)$$

$$= \alpha(G \backslash \{v_2, \dots, v_l\} \cup \{(v_1, w)\}) + \frac{|P| - 1}{2}$$

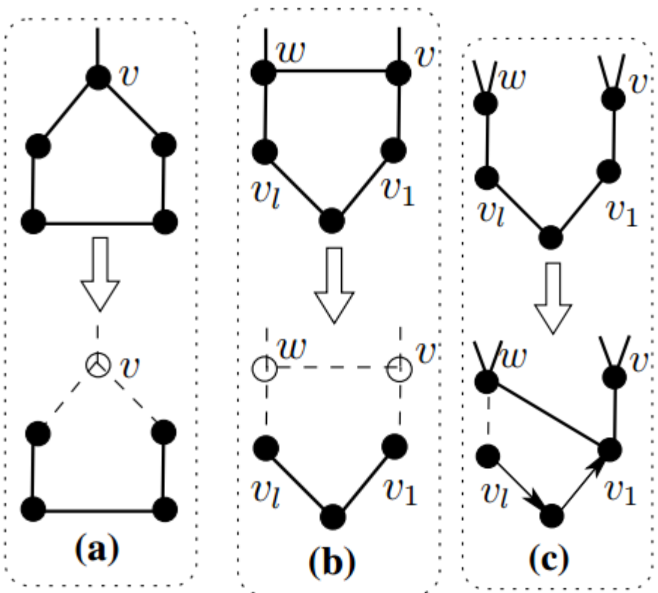


(c)

An Effective Linear-Time Algorithm

❖ **LinearTime**

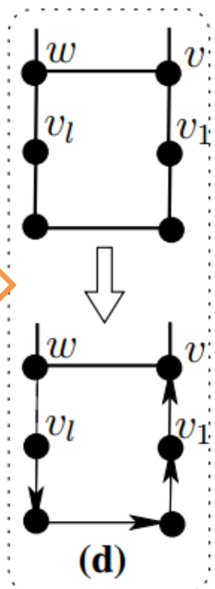
Lemma 4.1: (Degree-two Path Reductions) Consider a graph $G = (V, E)$ with minimum degree two. For a maximal degree-two path $P = \{v_1, v_2, \ldots, v_l\}$, let $v \notin P$ and $w \notin P$ be the unique vertices connected to v_1 and v_l, respectively.



Case 4: $|P|$ is even and $(v, w) \in E$

$$\Longrightarrow \alpha(G)$$

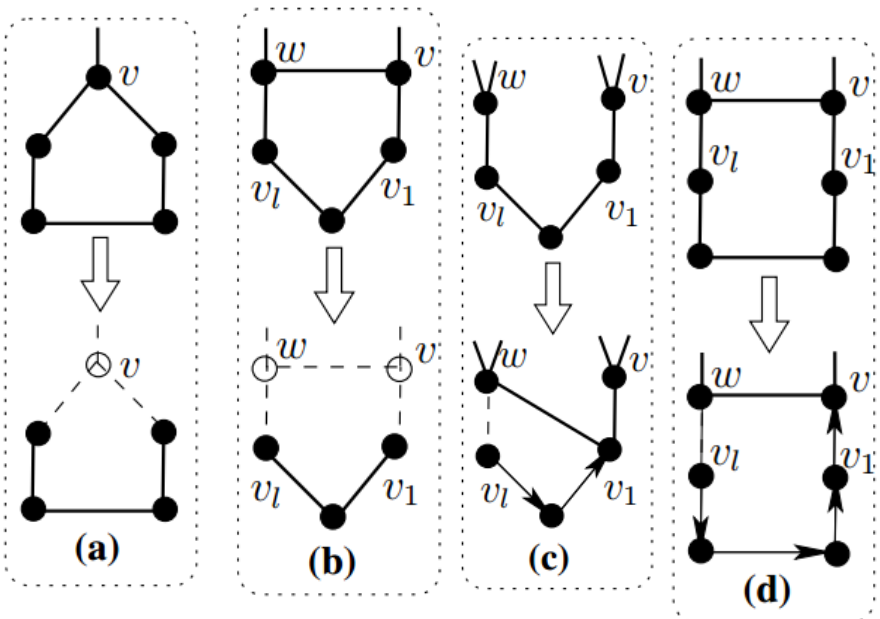
$$= \alpha(G \backslash \{v_1, \ldots, v_l\}) + \frac{|P|}{2}$$



An Effective Linear-Time Algorithm

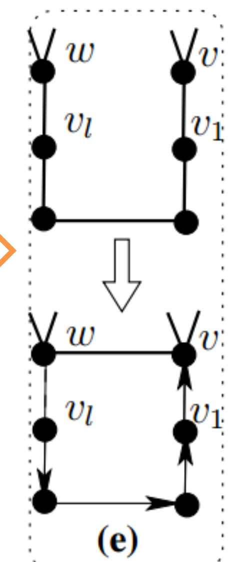
❖ **LinearTime**

Lemma 4.1: (Degree-two Path Reductions) Consider a graph $G = (V, E)$ with minimum degree two. For a maximal degree-two path $P = \{v_1, v_2, \dots, v_l\}$, let $v \notin P$ and $w \notin P$ be the unique vertices connected to v_1 and v_l, respectively.



Case 5: $|P|$ is even and $(v, w) \notin E$

$$\Rightarrow \alpha(G) = \alpha(G \backslash \{v_1, \dots, v_l\} \cup \{(v, w)\}) + \frac{|P|}{2}$$



An Effective Linear-Time Algorithm

- **LinearTime**

Step 1:

While $V_{=1} \neq \emptyset$ or $V_{=2} \neq \emptyset$ or $V_{\geq 3} \neq \emptyset$

  If $V_{=1} \neq \emptyset$ **then**

    DegreeOne-Reduction

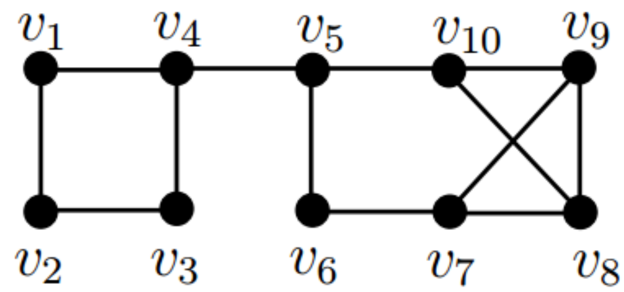
  Else if $V_{=2} \neq \emptyset$ **then**

    DegreeTwoPath-Reduction

  Else

    Inexact-Reduction

Step 2: Recover temporarily removed vertices



An Effective Linear-Time Algorithm

❖ **LinearTime**

Step 1:

While $V_{=1} \neq \emptyset$ or $V_{=2} \neq \emptyset$ or $V_{\geq 3} \neq \emptyset$

If $V_{=1} \neq \emptyset$ **then**

DegreeOne-Reduction

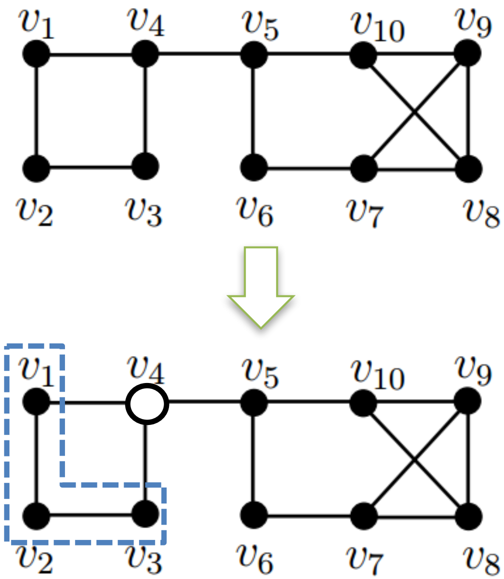
Else if $V_{=2} \neq \emptyset$ **then**

DegreeTwoPath-Reduction

Else

Inexact-Reduction

Step 2: Recover temporarily removed vertices



An Effective Linear-Time Algorithm

❖ **LinearTime**

Step 1:

While $V_{=1} \neq \emptyset$ or $V_{=2} \neq \emptyset$ or $V_{\geq 3} \neq \emptyset$

If $V_{=1} \neq \emptyset$ **then**

DegreeOne-Reduction

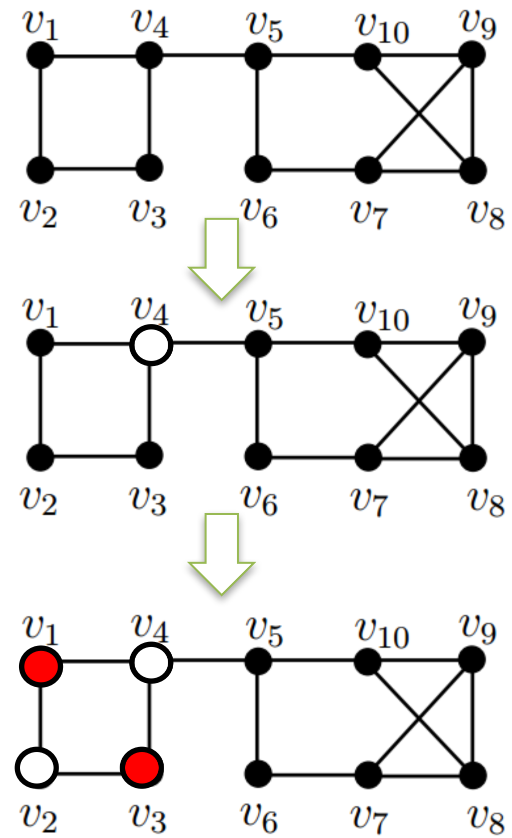
Else if $V_{=2} \neq \emptyset$ **then**

DegreeTwoPath-Reduction

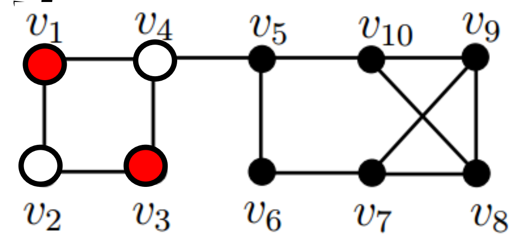
Else

Inexact-Reduction

Step 2: Recover temporarily removed vertices



An Effective Linear-Time Algorithm



- **LinearTime**

Step 1:

While $V_{=1} \neq \emptyset$ or $V_{=2} \neq \emptyset$ or $V_{\geq 3} \neq \emptyset$

If $V_{=1} \neq \emptyset$ **then**

DegreeOne-Reduction

Else if $V_{=2} \neq \emptyset$ **then**

DegreeTwoPath-Reduction

Else

Inexact-Reduction

Step 2: Recover temporarily removed vertices

An Effective Linear-Time Algorithm

❖ **LinearTime**

Step 1:

While $V_{=1} \neq \emptyset$ or $V_{=2} \neq \emptyset$ or $V_{\geq 3} \neq \emptyset$

If $V_{=1} \neq \emptyset$ **then**

DegreeOne-Reduction

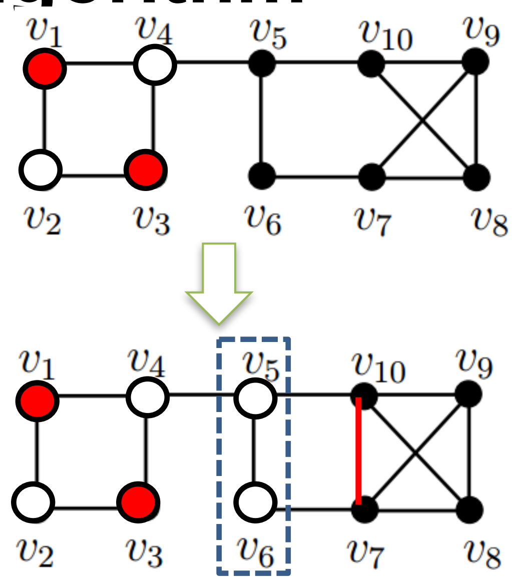
Else if $V_{=2} \neq \emptyset$ **then**

DegreeTwoPath-Reduction

Else

Inexact-Reduction

Step 2: Recover temporarily removed vertices



An Effective Linear-Time Algorithm

❖ **LinearTime**

Step 1:

While $V_{=1} \neq \emptyset$ or $V_{=2} \neq \emptyset$ or $V_{\geq 3} \neq \emptyset$

If $V_{=1} \neq \emptyset$ **then**

DegreeOne-Reduction

Else if $V_{=2} \neq \emptyset$ **then**

DegreeTwoPath-Reduction

Else

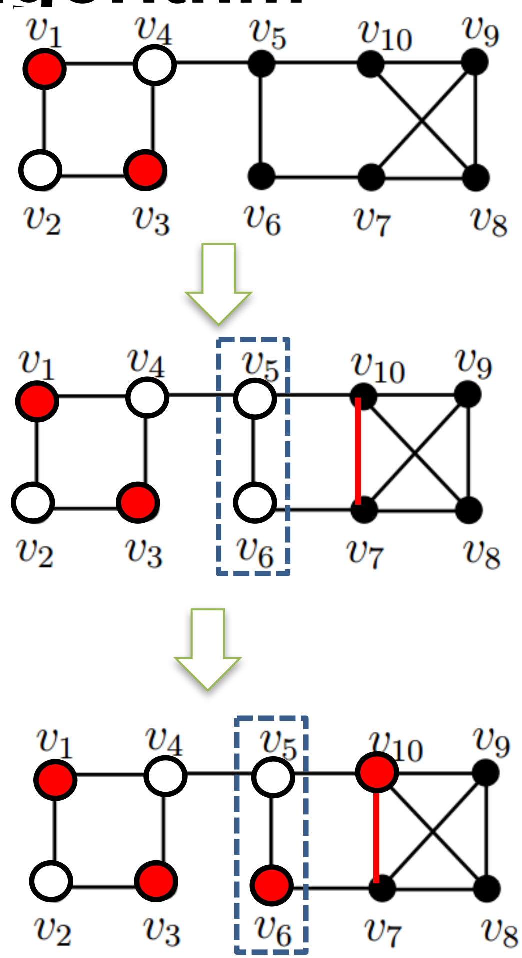
Inexact-Reduction

Step 2: Recover temporarily removed vertices

Complexity Analysis

Time: $O(m)$

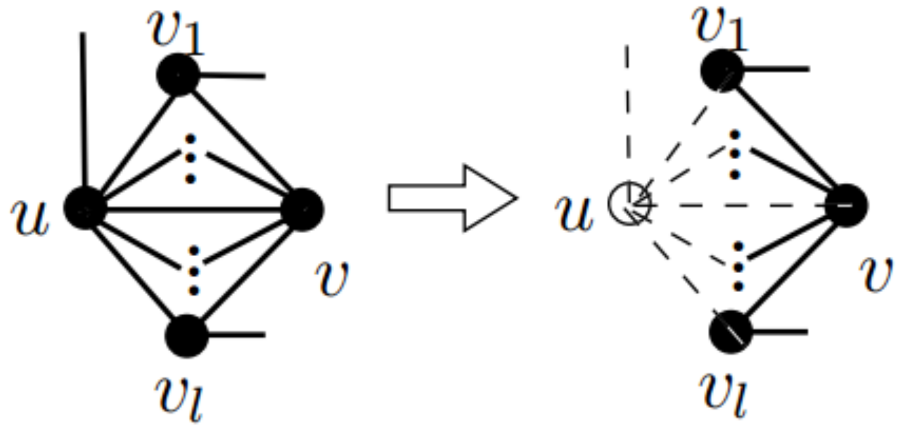
Space: $2m + O(n)$



A Near-Linear-Time Algorithm

❖ **NearLinear**

Lemma 5.1: (Dominance Reduction) [F. V. Fomin et al. *JACM'09*] Vertex v dominates vertex u if $(v, u) \in E$ and all neighbors of v other than u are also connected to u (i.e., $N(v)\backslash\{u\} \subseteq N(u)$). If v dominates u, then there exists a maximum independent set of G the excludes u; thus, we can remove u from G, and $\alpha(G) = \alpha(G\backslash\{u\})$.



Lemma 5.2: Vertex v dominates its neighbor u iff $\Delta(v, u) = d(v) - 1$, where $\Delta(v, u)$ is the number of triangles containing u and v

A Near-Linear-Time Algorithm

❖ **NearLinear**

Step1: Maintain the set D of candidate dominated vertices, and also maintain $\Delta(v, u)$ for every edge (v, u)

Step 2:

While $V_{=2} \neq \emptyset$ or $D \neq \emptyset$ or $V_{\geq 3} \neq \emptyset$

If $V_{=2} \neq \emptyset$ **then**

DegreeTwoPath-Reduction

Else if $D \neq \emptyset$ **then**

dominance reduction

Else

Inexact-Reduction

Step 2: Recover temporarily removed vertices

Complexity Analysis

Time: $O(m \times \Delta)$
(Δ is the maximum degree in G)

Space: $4m + O(n)$ in worst case and $2m + O(n)$ in practice

Extensions of Our Algorithms

- Accelerate **ARW**

- Compute Upper Bound of $\alpha(G)$

Outline

- ❑ Introduction
- ❑ Existing Works
- ❑ Our Reducing-Peeling Framework
- ❑ Our Approaches
- ❑ **Experimental Studies**
- ❑ Conclusion

Experimental Settings

❖ Datasets

Graph	#Vertices	#Edges	$\overline{d}$
GrQc	5,242	14,484	5.53
CondMat	23,133	93,439	8.08
AstroPh	18,772	198,050	21.10
Email	265,214	364,481	2.75
Epinions	75,879	405,740	10.69
cnr-2000	325,557	2,738,969	16.83
dblp	933,258	3,353,618	7.19
wiki-Talk	2,394,385	4,659,565	3.89
BerkStan	685,230	6,649,470	19.41
as-Skitter	1,696,415	11,095,398	13.08
in-2004	1,382,870	13,591,473	19.66
eu-2005	862,664	16,138,468	37.42
soc-pokec	1,632,803	22,301,964	27.32
LiveJ	4,847,571	42,851,237	17.68
hollywood	1,985,306	114,492,816	115.34
indochina	7,414,768	150,984,819	40.73
uk-2002	18,484,117	261,787,258	28.33
uk-2005	39,454,746	783,027,125	39.70
webbase	115,657,290	854,809,761	14.78
it-2004	41,290,682	1,027,474,947	49.77

❖ Environments

- All algorithms are implemented in C++
- All experiments are conducted on a machine with an Intel(R) Xeon(R) 3.4GHz CPU and 16GB main memory running Linux

Accuracy

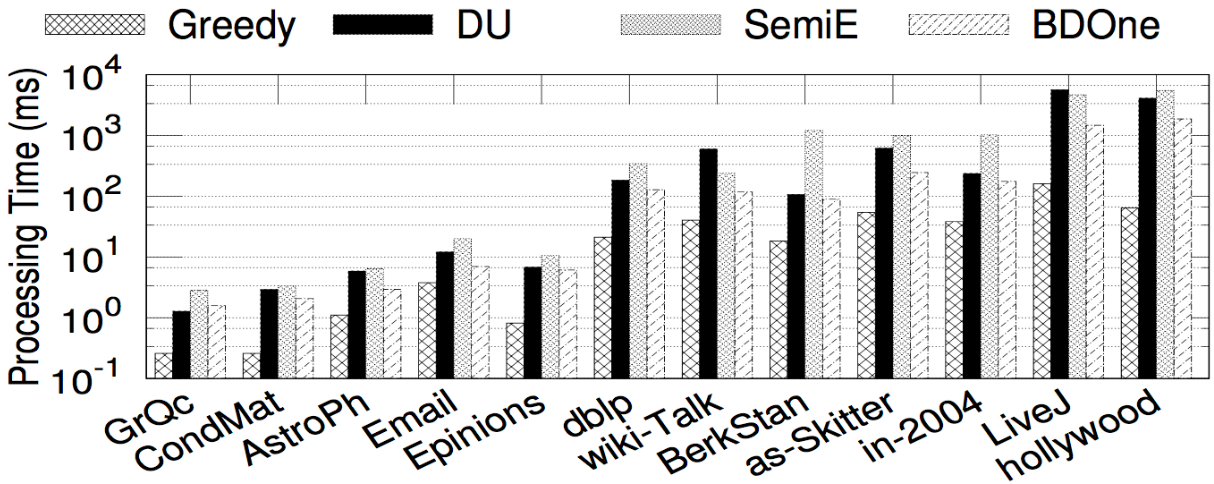
❖ Gap to the maximum independent set size

Graphs	Independence Number	Gap to the Independence Number							Accuracy of NearLinear	Kernel Graph Size by NearLinear
		Greedy	DU	SemiE	BDOne	BDTwo	LinearTime	NearLinear		
GrQc	2,459	5	1	1	0	0	0	**0***	100%	0
CondMat	9,612	17	5	1	4	2	1	**0***	100%	0
AstroPh	6,760	24	10	1	2	0	1	**0***	100%	0
Email	246,898	76	0	1	0	**0***	0	**0***	100%	0
Epinions	53,599	170	3	14	0	0	0	0	100%	6
dblp	434,289	484	63	53	45	5	4	**0***	100%	0
wiki-Talk	2,338,222	536	0	14	0	0	0	**0***	100%	0
BerkStan	408,482	11,092	3,000	4,458	1,088	385	766	428	99.895%	55,990
as-Skitter	1,170,580	34,591	2,336	5,886	319	55	170	39	99.997%	9,733
in-2004	896,724	14,832	3,553	5,918	656	351	412	57	99.993%	19,575
LiveJ	2,631,903	32,997	6,138	7,364	1,494	343	378	33	99.998%	10,173
hollywood	327,949	98	45	8	16	4	4	**0***	100%	0

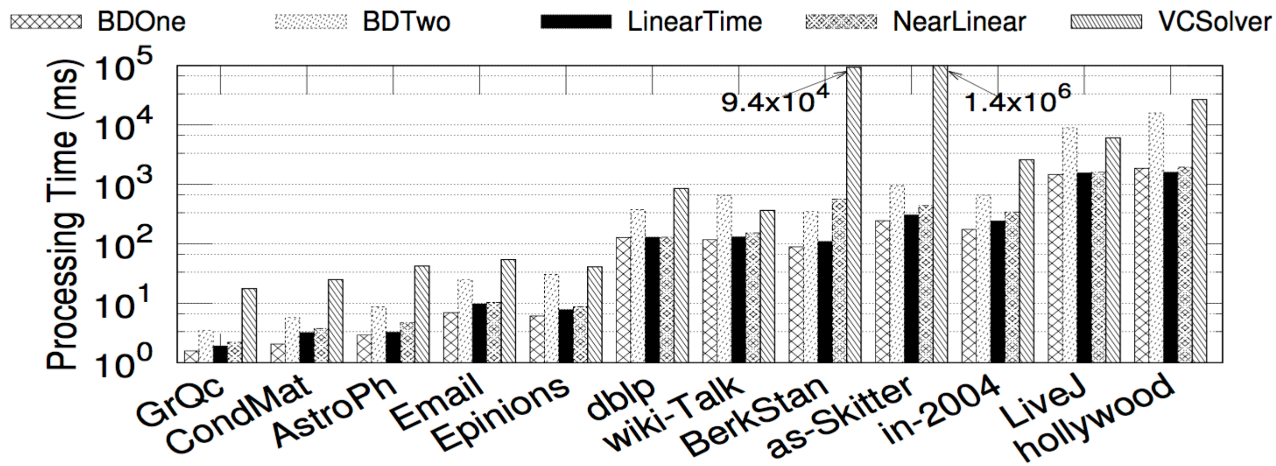
Table 3: The gap of the reported independent set size to the independence number computed by VCSolver [1] (* denotes that the independent set is reported as a maximum independent set by our algorithms)

Processing Time

(a) Compared with Existing Techniques



(b) Compare Our Techniques

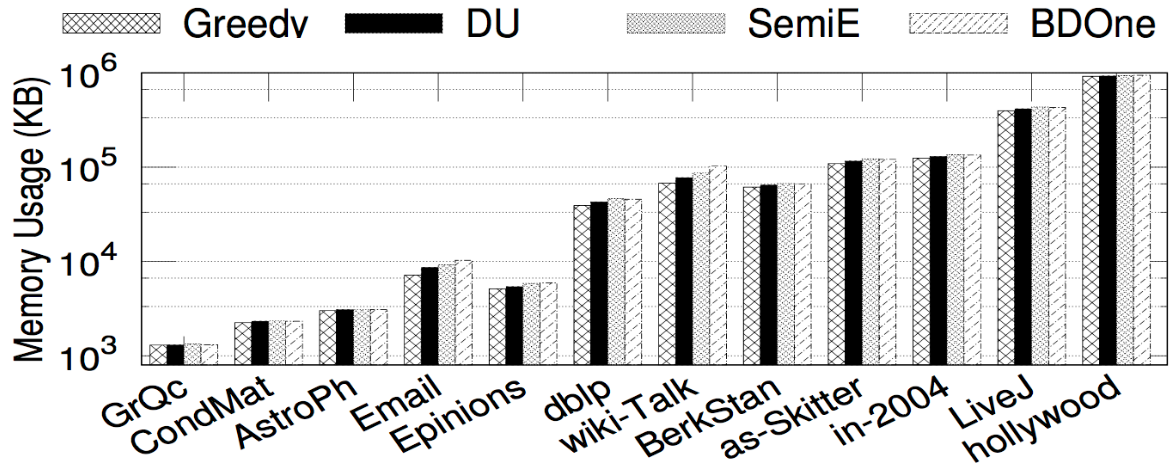


Algorithm	Time Complexity	Space Complexity	Exact Reduction Rules Used
BDOne	$O(m)$	$2m + O(n)$	Degree-one reduction [21]
BDTwo	$O(n \times m)$	$6m + O(n)$	Degree-one reduction [21] & Degree-two vertex reductions [21]
LinearTime	$O(m)$	$2m + O(n)$	Degree-one reduction [21] & Degree-two path reduction (this paper)
NearLinear	$O(m \times \Delta)$	$4m + O(n)$	Dominance reduction [21] & Degree-two path reduction (this paper)

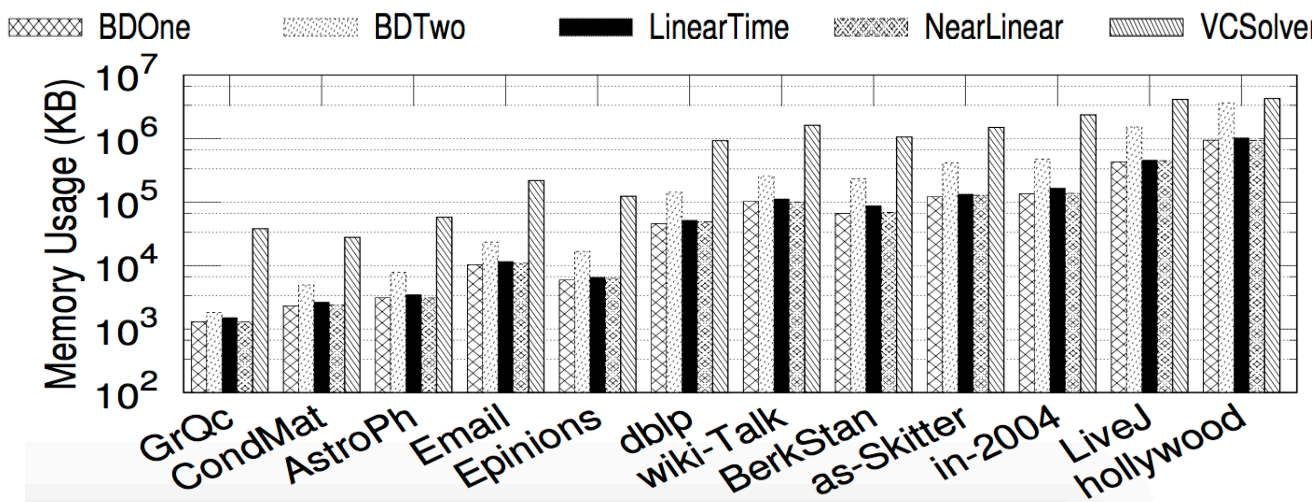
Table 1: Overview of our approaches (n: number of vertices, m: number of edges, Δ: maximum vertex degree)

Memory Usage

(a) Compared with Existing Techniques



(b) Compare Our Techniques

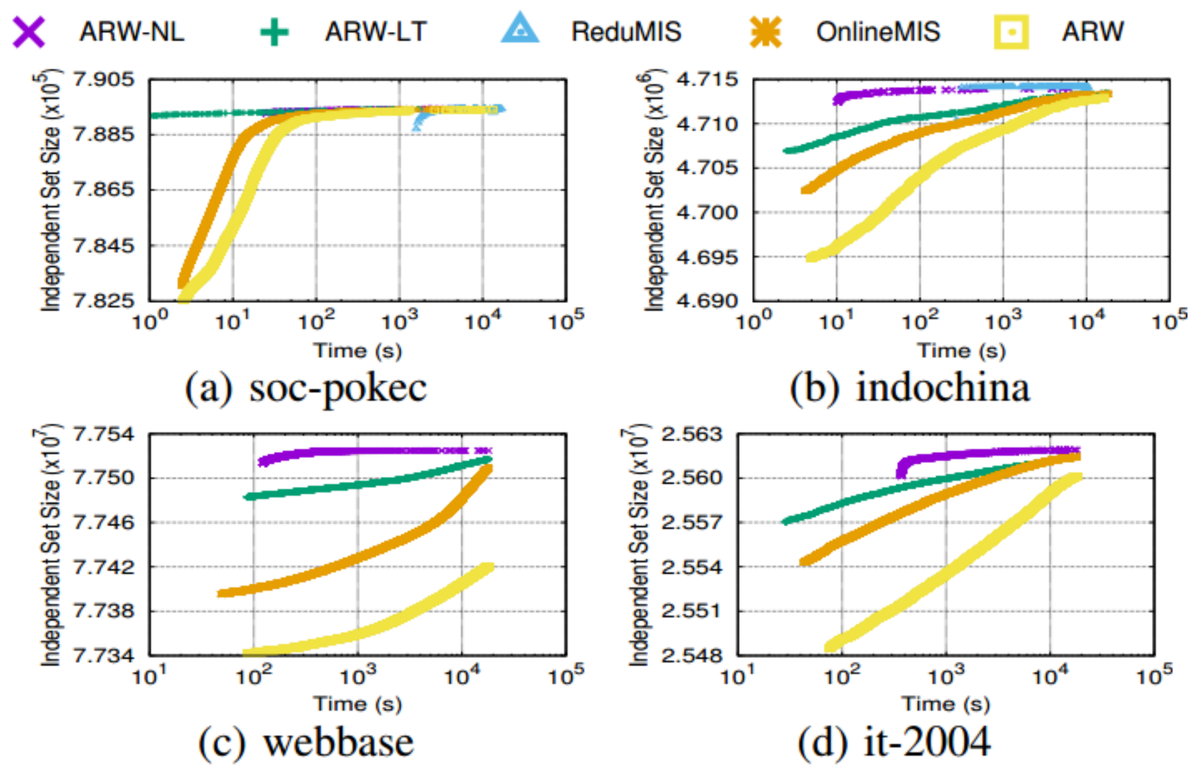


Algorithm	Time Complexity	Space Complexity	Exact Reduction Rules Used
BDOne	$O(m)$	$2m + O(n)$	Degree-one reduction [21]
BDTwo	$O(n \times m)$	$6m + O(n)$	Degree-one reduction [21] & Degree-two vertex reductions [21]
LinearTime	$O(m)$	$2m + O(n)$	Degree-one reduction [21] & Degree-two path reduction (this paper)
NearLinear	$O(m \times \Delta)$	$4m + O(n)$	Dominance reduction [21] & Degree-two path reduction (this paper)

Table 1: Overview of our approaches (n: number of vertices, m: number of edges, Δ: maximum vertex degree)

Boost ARW

ARW-NL, ARW-LT: ARW boosted by NearLinear and LinearTime, respectively.



Convergence plots of local search algorithms

Outline

- ❑ Introduction
- ❑ Existing Works
- ❑ Our Reducing-Peeling Framework
- ❑ Our Approaches
- ❑ Experimental Studies
- ❑ **Conclusion**

Conclusion

- A new Reducing-Peeling framework
- Time-efficient and space-effective techniques to implement the reducing-peeling framework
- Find large independent sets efficiently for large real-world graphs with billions of edges

Thank you !

Question?

Lijun.Chang@unsw.edu.au