# Greedy `MaxCut` Algorithms and their Information Content

**Yatao Bian**, Alexey Gronskiy and Joachim M. Buhmann

Machine Learning Institute, ETH Zurich

April 27, 2015

# Contents

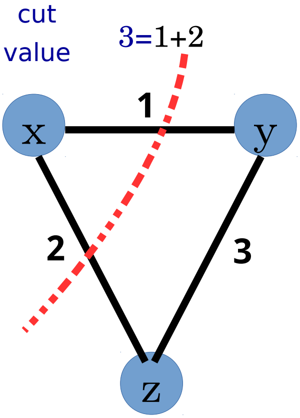
# Contents

# MaxCut

`MaxCut`: classical NP-hard problem

MaxCut: classical NP-hard problem

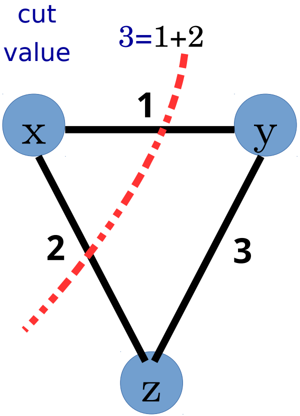
- $G = (V, E)$, vertex set $V$, edge set $E$, weights $w_{ij} \geq 0$



# MaxCut

MaxCut: classical NP-hard problem

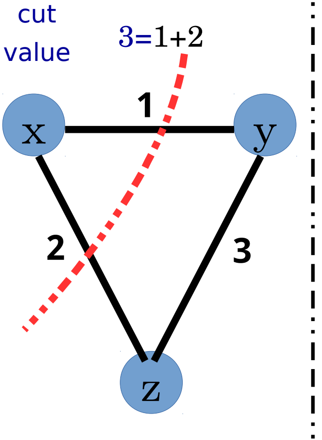
- $G = (V, E)$, vertex set $V$, edge set $E$, weights $w_{ij} \geq 0$
- CUT $c := (S, V \backslash S)$, cut space $\mathcal{C}$ ($|\mathcal{C}| = 2^{n-1} - 1$)



# MaxCut

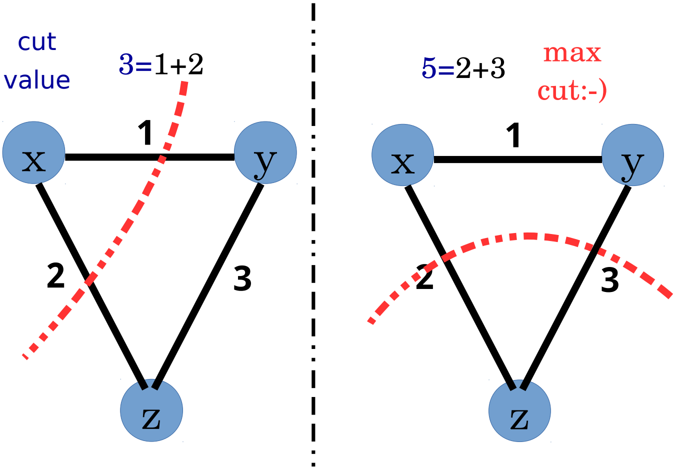
MaxCut: classical NP-hard problem

- $G = (V, E)$, vertex set $V$, edge set $E$, weights $w_{ij} \geq 0$
- CUT $c := (S, V \backslash S)$, cut space $\mathcal{C}$ ($|\mathcal{C}| = 2^{n-1} - 1$)
- Cut value: $\mathrm{cut}(c, G) := \sum_{i \in S, j \in V \backslash S} w_{ij}$



MaxCut: classical NP-hard problem

- $G = (V, E)$, vertex set $V$, edge set $E$, weights $w_{ij} \geq 0$
- CUT $c := (S, V \backslash S)$, cut space $\mathcal{C}$ ($|\mathcal{C}| = 2^{n-1} - 1$)
- Cut value: $\text{cut}(c, G) := \sum_{i \in S, j \in V \backslash S} w_{ij}$



# Greedy Algorithms for MaxCut

| Name | Greedy Heuristic | Techniques | |
|---|---|---|---|
| | | Sorting | Init. Vertices |
| Deterministic Double Greedy | Double | | |
| SG (Sahni & Gonzales) | | | ✓ |
| SG3 (variant of SG) | | ✓ | ✓ |
| Edge Contraction (EC) | Backward | ✓ | |

# Double Greedy Taxonomy

# Double Greedy Taxonomy

**D**eterministic **D**ouble Greedy (D2Greedy)

# Double Greedy Taxonomy

**D**eterministic **D**ouble Greedy (D2Greedy)

**Require:** graph $G = (V, E)$
**Ensure:** cut and the cut value

# Double Greedy Taxonomy

**D**eterministic **D**ouble Greedy (D2Greedy)

**Require:** graph $G = (V, E)$
**Ensure:** cut and the cut value
1: init. 2 solutions $S := \emptyset$, $T := V$

- works on 2 solutions simultaneously

# Double Greedy Taxonomy

Deterministic Double Greedy (D2Greedy)

**Require:** graph $G = (V, E)$
**Ensure:** cut and the cut value
init. 2 solutions $S := \emptyset$, $T := V$
    `//in random order`
**for** each vertex $v_i \in V$ **do**
**end for**

- works on 2 solutions simultaneously
- for each vertex, decides whether it should be added to $S$, or removed from $T$

# Double Greedy Taxonomy

Deterministic Double Greedy (D2Greedy)

**Require:** graph $G = (V, E)$
**Ensure:** cut and the cut value

init. 2 solutions $S := \emptyset$, $T := V$
    `//in random order`
**for** each vertex $v_i \in V$ **do**
    $a_i :=$ gain of adding $v_i$ to $S$
    $b_i :=$ gain of removing $v_i$ from $T$
**end for**

- works on 2 solutions simultaneously
- for each vertex, decides whether it should be added to $S$, or removed from $T$

# Double Greedy Taxonomy

**D**eterministic **D**ouble Greedy (D2Greedy)

```
Require: graph G = (V, E)
Ensure: cut and the cut value
init. 2 solutions S := ∅, T := V
    //in random order
for each vertex v_i ∈ V do
    a_i := gain of adding v_i to S
    b_i := gain of removing v_i from T
    if a_i ≥ b_i then
        add v_i to S
    else
        remove v_i from T
    end if
end for
```

- works on 2 solutions simultaneously
- for each vertex, decides whether it should be added to $S$, or removed from $T$

# Double Greedy Taxonomy

**D**eterministic **D**ouble Greedy (D2Greedy)

**Require:** graph $G = (V, E)$
**Ensure:** cut and the cut value

init. 2 solutions $S := \emptyset$, $T := V$
    //in random order
**for** each vertex $v_i \in V$ **do**
    $a_i :=$ gain of adding $v_i$ to $S$
    $b_i :=$ gain of removing $v_i$ from $T$
    **if** $a_i \geq b_i$ **then**
        add $v_i$ to $S$
    **else**
        remove $v_i$ from $T$
    **end if**
**end for**
**return** cut: $(S, V \backslash S)$, cut value

- works on 2 solutions simultaneously
- for each vertex, decides whether it should be added to $S$, or removed from $T$

# Double Greedy Taxonomy

Deterministic Double Greedy (D2Greedy)

**Require:** graph $G = (V, E)$
**Ensure:** cut and the cut value
init. 2 solutions $S := \emptyset$, $T := V$
    //in random order
**for** each vertex $v_i \in V$ **do**
    $a_i :=$ gain of adding $v_i$ to $S$
    $b_i :=$ gain of removing $v_i$ from $T$
    **if** $a_i \geq b_i$ **then**
        add $v_i$ to $S$
    **else**
        remove $v_i$ from $T$
    **end if**
**end for**
**return** cut: $(S, V \backslash S)$, cut value

- works on 2 solutions simultaneously
- for each vertex, decides whether it should be added to $S$, or removed from $T$

Differences between the double greedy algorithms:

# Double Greedy Taxonomy

Deterministic Double Greedy (D2Greedy)

**Require:** graph $G = (V, E)$
**Ensure:** cut and the cut value

init. 2 solutions $S := \emptyset$, $T := V$
    //in random order
**for** each vertex $v_i \in V$ **do**
    $a_i :=$ gain of adding $v_i$ to $S$
    $b_i :=$ gain of removing $v_i$ from $T$
    **if** $a_i \geq b_i$ **then**
        add $v_i$ to $S$
    **else**
        remove $v_i$ from $T$
    **end if**
**end for**
**return** cut: $(S, V \setminus S)$, cut value

- works on 2 solutions simultaneously
- for each vertex, decides whether it should be added to $S$, or removed from $T$

Differences between the double greedy algorithms:

| | | | | |
|---|---|---|---|---|
| D2Greedy | $\rightarrow$ | select the first 2 vertices | $\rightarrow$ | SG |

# Double Greedy Taxonomy

**D**eterministic **D**ouble Greedy (D2Greedy)

**Require:** graph $G = (V, E)$
**Ensure:** cut and the cut value

```
init. 2 solutions S := ∅, T := V
    //in random order
for each vertex v_i ∈ V do
    a_i := gain of adding v_i to S
    b_i := gain of removing v_i from T
    if a_i ≥ b_i then
        add v_i to S
    else
        remove v_i from T
    end if
end for
return cut: (S, V\S), cut value
```

- works on 2 solutions simultaneously
- for each vertex, decides whether it should be added to $S$, or removed from $T$

Differences between the double greedy algorithms:

| | | | | |
|---|---|---|---|---|
| D2Greedy | $\rightarrow$ | select the first 2 vertices | $\rightarrow$ | SG |
| SG | $\rightarrow$ | sort the candidates | $\rightarrow$ | SG3 |

# Backward Greedy – Edge Contraction Algorithm

# Backward Greedy – Edge Contraction Algorithm

**E**dge **C**ontraction (EC)

# Backward Greedy – Edge Contraction Algorithm

**E**dge **C**ontraction (EC)

**Require:** graph $G = (V, E)$
**Ensure:** cut, cut value

# Backward Greedy – Edge Contraction Algorithm

**E**dge **C**ontraction (EC)

**Require:** graph $G = (V, E)$
**Ensure:** cut, cut value

```
repeat

until 2 "super" vertices left
```

# Backward Greedy – Edge Contraction Algorithm

**E**dge **C**ontraction (EC)

**Require:** graph $G = (V, E)$
**Ensure:** cut, cut value
1: **repeat**
5: **until** 2 "super" vertices left

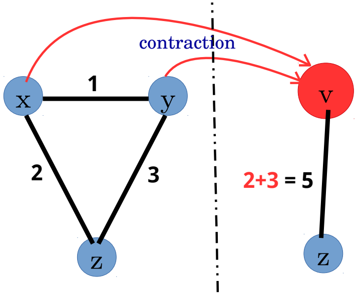
- contract the lightest edge in each step

# Backward Greedy – Edge Contraction Algorithm

**E**dge **C**ontraction (EC)

**Require:** graph $G = (V, E)$
**Ensure:** cut, cut value
1: **repeat**
5: **until** 2 "super" vertices left

- contract the lightest edge in each step



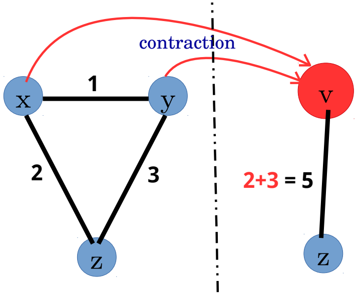
# Backward Greedy – Edge Contraction Algorithm

**E**dge **C**ontraction (EC)

**Require:** graph $G = (V, E)$
**Ensure:** cut, cut value
1: **repeat**
2: find the lightest edge $(x, y)$ in $G$

5: **until** 2 "super" vertices left

- contract the lightest edge in each step



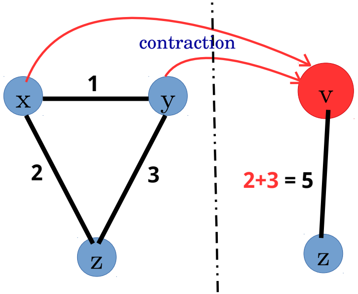
# Backward Greedy – Edge Contraction Algorithm

**E**dge **C**ontraction (EC)

**Require:** graph $G = (V, E)$
**Ensure:** cut, cut value
1: **repeat**
2: find the lightest edge $(x, y)$ in $G$
3: contract $x, y$ to be a super vertex $v$

5: **until** 2 "super" vertices left

- contract the lightest edge in each step



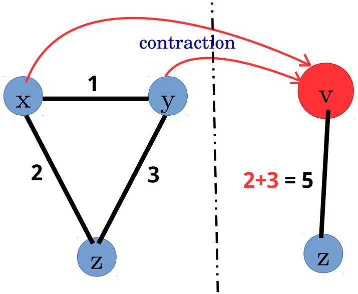
# Backward Greedy – Edge Contraction Algorithm

**E**dge **C**ontraction (EC)

**Require:** graph $G = (V, E)$
**Ensure:** cut, cut value

1: **repeat**
2: find the lightest edge $(x, y)$ in $G$
3: contract $x, y$ to be a super vertex $v$
4: set the edge weights connecting $v$
5: **until** 2 "super" vertices left

- contract the lightest edge in each step

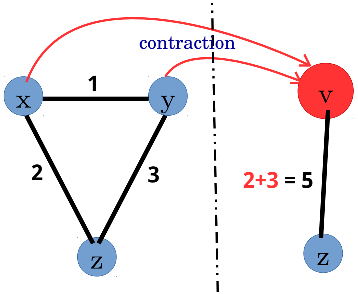


# Backward Greedy – Edge Contraction Algorithm

**E**dge **C**ontraction (EC)

**Require:** graph $G = (V, E)$
**Ensure:** cut, cut value
1: **repeat**
2: find the lightest edge $(x, y)$ in $G$
3: contract $x, y$ to be a super vertex $v$
4: set the edge weights connecting $v$
5: **until** 2 "super" vertices left
6: **return** the 2 super vertices

- contract the lightest edge in each step

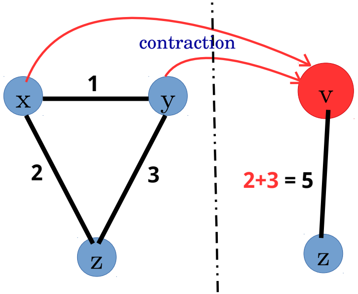


# Backward Greedy – Edge Contraction Algorithm

**E**dge **C**ontraction (EC)

**Require:** graph $G = (V, E)$
**Ensure:** cut, cut value
1: **repeat**
2: find the lightest edge $(x, y)$ in $G$
3: contract $x, y$ to be a super vertex $v$
4: set the edge weights connecting $v$
5: **until** 2 "super" vertices left
6: **return** the 2 super vertices

- contract the lightest edge in each step




Backward greedy: EC tries to remove the lightest edge from the cut set in each step

# Contents

# Glance of Approximation Set Coding (ASC)

How to measure the robustness of these algorithms facing noise?

# Glance of Approximation Set Coding (ASC)

How to measure the robustness of these algorithms facing noise?

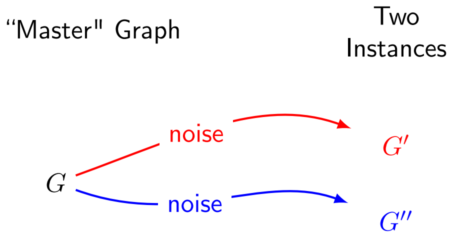
- **ASC**: an analogy to Shannon's communication theory
  learning procedure $\Leftrightarrow$ communication process [Buhmann 2010]

# Glance of Approximation Set Coding (ASC)

How to measure the robustness of these algorithms facing noise?

- **ASC**: an analogy to Shannon's communication theory
  learning procedure ⇔ communication process [Buhmann 2010]

2 instances scenario: training $G'$, test $G''$ (noisy instaces of $G$)

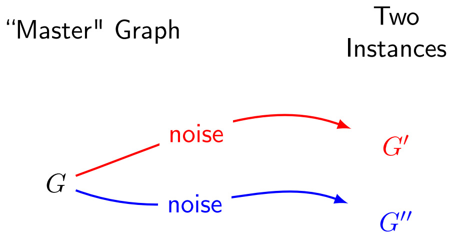


# Glance of Approximation Set Coding (ASC)

How to measure the robustness of these algorithms facing noise?

- **ASC**: an analogy to Shannon's communication theory
  learning procedure ⇔ communication process [Buhmann 2010]

2 instances scenario: training $G'$, test $G''$ (noisy instaces of $G$)




- Models/algorithms should generalize well from $G'$ to $G''$

# Approximate Solving and Algorithmic Approx. Set

- Empirical risk minimizer

$c^{\perp}(G) := \arg\min_c R(c, G)$

# Approximate Solving and Algorithmic Approx. Set

- Empirical risk minimizer

$c^{\perp}(G) := \arg\min_c R(c, G)$

$c^{\perp}(G') \overset{\text{noise}}{\neq} c^{\perp}(G'')$

# Approximate Solving and Algorithmic Approx. Set

- Empirical risk minimizer

$c^{\perp}(G) := \arg\min_c R(c, G)$

$c^{\perp}(G') \overset{\text{noise}}{\neq} c^{\perp}(G'')$

- $\gamma$-approximation set (solutions $\gamma$ distant from $c^{\perp}$): $C_\gamma(G) := \left\{ c \in \mathcal{C} \;\middle|\; R(c, G) - R(c^{\perp}, G) \leq \gamma \right\}$

$\gamma$: resolution

# Approximate Solving and Algorithmic Approx. Set

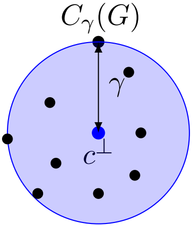
- Empirical risk minimizer

$c^{\perp}(G) := \arg\min_c R(c, G)$

$c^{\perp}(G') \overset{\text{noise}}{\neq} c^{\perp}(G'')$

- $\gamma$-approximation set (solutions $\gamma$ distant from $c^{\perp}$): $C_\gamma(G) := \{c \in \mathcal{C} \mid R(c, G) - R(c^{\perp}, G) \leq \gamma\}$

$\gamma$: resolution



# Approximate Solving and Algorithmic Approx. Set

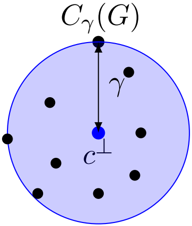
- Empirical risk minimizer

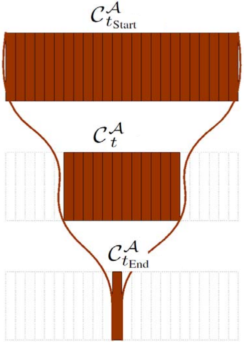
$c^{\perp}(G) := \arg\min_c R(c, G)$

$c^{\perp}(G') \overset{\text{noise}}{\neq} c^{\perp}(G'')$

- $\gamma$-approximation set (solutions $\gamma$ distant from $c^{\perp}$): $C_\gamma(G) := \{c \in \mathcal{C} \mid R(c, G) - R(c^{\perp}, G) \le \gamma\}$

$\gamma$: resolution



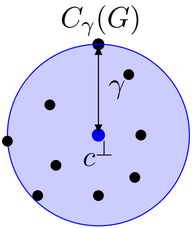



# Approximate Solving and Algorithmic Approx. Set

- Empirical risk minimizer

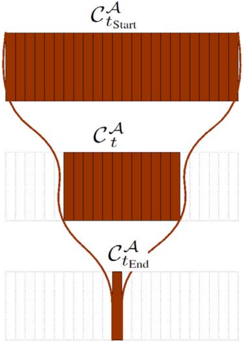
$c^{\perp}(G) := \arg\min_c R(c, G)$

$c^{\perp}(G') \overset{\text{noise}}{\neq} c^{\perp}(G'')$




- $\gamma$-approximation set (solutions $\gamma$ distant from $c^{\perp}$): $C_\gamma(G) := \{c \in \mathcal{C} \mid R(c, G) - R(c^{\perp}, G) \leq \gamma\}$

$\gamma$: resolution



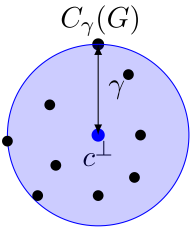

- Flow of *contractive* $\mathscr{A}$: sequence of the available solution sets in each step $t$

# Approximate Solving and Algorithmic Approx. Set

- Empirical risk minimizer

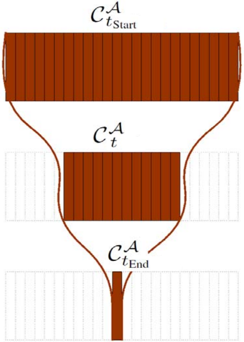
$c^{\perp}(G) := \arg\min_c R(c, G)$

$c^{\perp}(G') \overset{\text{noise}}{\neq} c^{\perp}(G'')$




- $\gamma$-approximation set (solutions $\gamma$ distant from $c^{\perp}$): $C_\gamma(G) := \{c \in \mathcal{C} \mid R(c, G) - R(c^{\perp}, G) \leq \gamma\}$

$\gamma$: resolution




- Flow of *contractive* $\mathscr{A}$: sequence of the available solution sets in each step $t$

Algorithmic $t$-approximation set [Gronskiy and Buhmann 2014]:

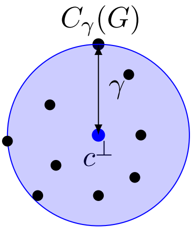
$$C_t^{\mathscr{A}}(G)$$

# Approximate Solving and Algorithmic Approx. Set

- Empirical risk minimizer

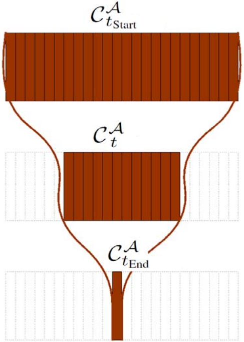
$c^{\perp}(G) := \arg\min_c R(c, G)$

$c^{\perp}(G') \overset{\text{noise}}{\neq} c^{\perp}(G'')$




- $\gamma$-approximation set (solutions $\gamma$ distant from $c^{\perp}$): $C_\gamma(G) := \{c \in \mathcal{C} \mid R(c, G) - R(c^{\perp}, G) \leq \gamma\}$

$\gamma$: resolution




- Flow of *contractive* $\mathscr{A}$: sequence of the available solution sets in each step $t$

Algorithmic $t$-approximation set [Gronskiy and Buhmann 2014]:

$$C_t^{\mathscr{A}}(G)$$

$\nearrow$ step $t \Leftrightarrow \searrow$ resolution $\gamma$

# Analogy of Communication System

# Analogy of Communication System

(Not going into detail here)

# Analogy of Communication System

(Not going into detail here)

## Analogical mutual information in step $t$

$$I_t^{\mathscr{A}} := \mathbb{E}_{G',G''}\left[\log\left(\frac{|\mathcal{C}|\cdot|\Delta C_t^{\mathscr{A}}(G',G'')|}{|C_t^{\mathscr{A}}(G')|\cdot|C_t^{\mathscr{A}}(G'')|}\right)\right]$$

$$\Delta C_t^{\mathscr{A}}(G',G'') = C_t^{\mathscr{A}}(G') \cap C_t^{\mathscr{A}}(G'')$$

# Analogy of Communication System

(Not going into detail here)

## Analogical mutual information in step $t$

$$I_t^{\mathscr{A}} := \mathbb{E}_{G',G''}\left[\log\left(\frac{|\mathcal{C}|\cdot|\Delta C_t^{\mathscr{A}}(G',G'')|}{|C_t^{\mathscr{A}}(G')|\cdot|C_t^{\mathscr{A}}(G'')|}\right)\right]$$

$$\Delta C_t^{\mathscr{A}}(G',G'') = C_t^{\mathscr{A}}(G') \cap C_t^{\mathscr{A}}(G'')$$

## Information content of $\mathscr{A}$

# Analogy of Communication System

(Not going into detail here)

## Analogical mutual information in step $t$

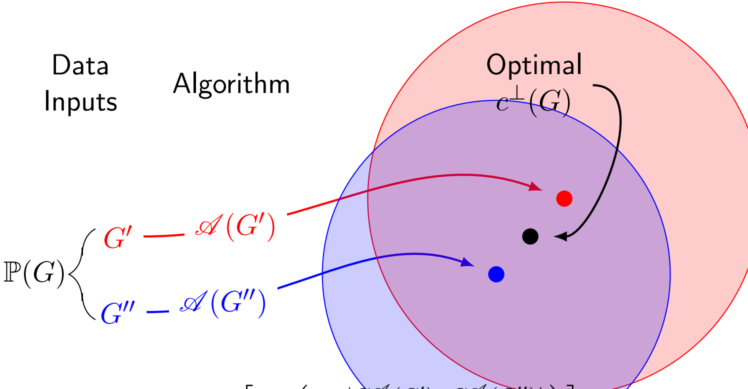
$$I_t^{\mathscr{A}} := \mathbb{E}_{G',G''}\left[\log\left(\frac{|\mathcal{C}|\cdot|\Delta C_t^{\mathscr{A}}(G',G'')|}{|C_t^{\mathscr{A}}(G')|\cdot|C_t^{\mathscr{A}}(G'')|}\right)\right]$$

$$\Delta C_t^{\mathscr{A}}(G',G'') = C_t^{\mathscr{A}}(G') \cap C_t^{\mathscr{A}}(G'')$$

## Information content of $\mathscr{A}$

channel capacity $I^{\mathscr{A}} := \max_t I_t^{\mathscr{A}}$

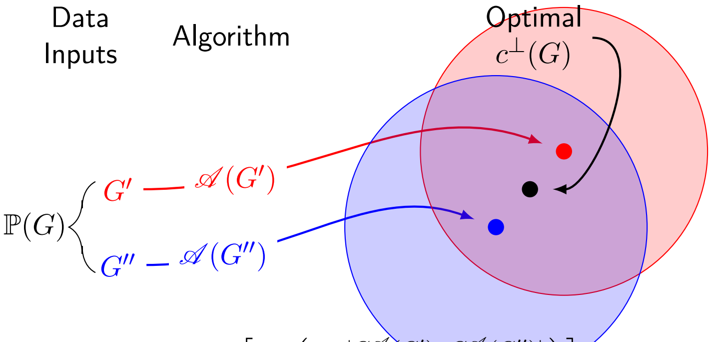
# Information Content of an Algorithm $\mathscr{A}$




mutual information: $I_t^{\mathscr{A}} := \mathbb{E}\left[\log\left(|\mathcal{C}|\frac{|C_t^{\mathscr{A}}(G') \cap C_t^{\mathscr{A}}(G'')|}{|C_t^{\mathscr{A}}(G')| \cdot |C_t^{\mathscr{A}}(G'')|}\right)\right]$
(stepwise information)

Information content of $\mathscr{A}$: channel capacity $I^{\mathscr{A}} := \max_t I_t^{\mathscr{A}}$

# Information Content of an Algorithm $\mathscr{A}$




mutual information: $I_t^{\mathscr{A}} := \mathbb{E}\left[\log\left(|\mathcal{C}|\frac{|C_t^{\mathscr{A}}(G')\cap C_t^{\mathscr{A}}(G'')|}{|C_t^{\mathscr{A}}(G')|\cdot|C_t^{\mathscr{A}}(G'')|}\right)\right]$

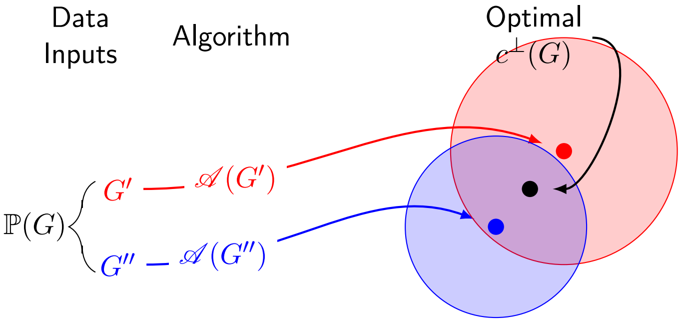
(stepwise information)

$\nearrow$ step $t \Leftrightarrow \searrow$ resolution $\gamma$

less informative but more robust

Information content of $\mathscr{A}$: channel capacity $I^{\mathscr{A}} := \max_t I_t^{\mathscr{A}}$

# Information Content of an Algorithm $\mathscr{A}$




mutual information: $I_t^{\mathscr{A}} := \mathbb{E}\left[\log\left(|\mathcal{C}|\frac{|C_t^{\mathscr{A}}(G')\cap C_t^{\mathscr{A}}(G'')|}{|C_t^{\mathscr{A}}(G')|\cdot|C_t^{\mathscr{A}}(G'')|}\right)\right]$

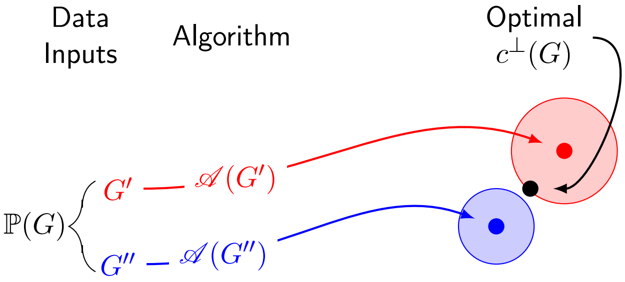
(stepwise information)

$\nearrow$ step $t$ $\Leftrightarrow$ $\searrow$ resolution $\gamma$

less informative but more robust

Information content of $\mathscr{A}$: channel capacity $I^{\mathscr{A}} := \max_t I_t^{\mathscr{A}}$

# Information Content of an Algorithm $\mathscr{A}$




mutual information: $I_t^{\mathscr{A}} := \mathbb{E}\left[\log\left(|\mathcal{C}|\frac{|C_t^{\mathscr{A}}(G') \cap C_t^{\mathscr{A}}(G'')|}{|C_t^{\mathscr{A}}(G')| \cdot |C_t^{\mathscr{A}}(G'')|}\right)\right]$

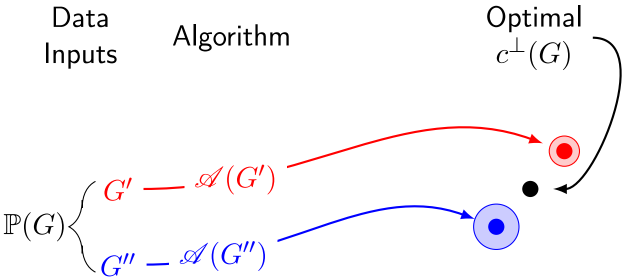
(stepwise information)

$\nearrow$ step $t \Leftrightarrow \searrow$ resolution $\gamma$

less informative but more robust

Information content of $\mathscr{A}$: channel capacity $I^{\mathscr{A}} := \max_t I_t^{\mathscr{A}}$

# Information Content of an Algorithm $\mathscr{A}$




mutual information: $I_t^{\mathscr{A}} := \mathbb{E}\left[\log\left(|\mathcal{C}|\frac{|C_t^{\mathscr{A}}(G')\cap C_t^{\mathscr{A}}(G'')|}{|C_t^{\mathscr{A}}(G')|\cdot|C_t^{\mathscr{A}}(G'')|}\right)\right]$

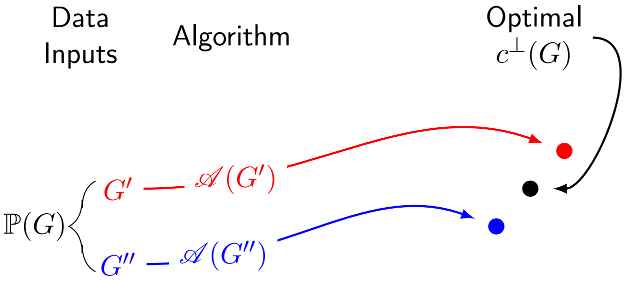
(stepwise information)

$\nearrow$ step $t \Leftrightarrow \searrow$ resolution $\gamma$

less informative but more robust

Information content of $\mathscr{A}$: channel capacity $I^{\mathscr{A}} := \max_t I_t^{\mathscr{A}}$

# Information Content of an Algorithm $\mathscr{A}$




mutual information: $I_t^{\mathscr{A}} := \mathbb{E}\left[\log\left(|\mathcal{C}| \frac{|C_t^{\mathscr{A}}(G') \cap C_t^{\mathscr{A}}(G'')|}{|C_t^{\mathscr{A}}(G')| \cdot |C_t^{\mathscr{A}}(G'')|}\right)\right]$

(stepwise information)

$\nearrow$ step $t$ $\Leftrightarrow$ $\searrow$ resolution $\gamma$

less informative but more robust

Information content of $\mathscr{A}$: channel capacity $I^{\mathscr{A}} := \max_t I_t^{\mathscr{A}}$

# Contents

# Counting – Double Greedy Algorithms

Counting methods similar for double greedy algorithms (D2Greedy, SG, SG3)

# Counting – Double Greedy Algorithms

Counting methods similar for double greedy algorithms (D2Greedy, SG, SG3)

- SG3: assume $k$ vertices unlabeled in step $t$, $|C_t^{\mathscr{A}}(G^{'})| = |C_t^{\mathscr{A}}(G^{''})| = 2^k$

# Counting – Double Greedy Algorithms

Counting methods similar for double greedy algorithms (D2Greedy, SG, SG3)

- SG3: assume $k$ vertices unlabeled in step $t$, $|C_t^{\mathscr{A}}(G^{'})| = |C_t^{\mathscr{A}}(G^{''})| = 2^k$
- $|C_t^{\mathscr{A}}(G^{'}) \cap C_t^{\mathscr{A}}(G^{''})|$

Counting methods similar for double greedy algorithms (D2Greedy, SG, SG3)

- SG3: assume $k$ vertices unlabeled in step $t$, $|C_t^{\mathscr{A}}(G')| = |C_t^{\mathscr{A}}(G'')| = 2^k$
- $|C_t^{\mathscr{A}}(G') \cap C_t^{\mathscr{A}}(G'')|$
  We propose (and prove correctness) polynomial time algorithm to count (not going in detail here):

For the SG3 (Alg. 6, see Supplement), after step $t$ ($t = 1, \cdots, n-1$) there are $k = n - t - 1$ unlabelled vertices, and it is clear that $|C(G')| = |C(G'')| = 2^k$.

To count the intersection set $\Delta(G', G'')$, assume the solution set pair of $G'$ is $(S_1', S_2')$, the solution set pair of $G''$ is $(S_1'', S_2'')$, so the unlabelled vertex sets are $T' = V \backslash \{S_1' \cup S_2'\}$, $T'' = V \backslash \{S_1'' \cup S_2''\}$, respectively. Denote $L := T' \cap T''$ be the common vertices of the two unlabelled vertex sets, so $l = |L|$ ($0 \le l \le k$) is the number of common vertices in the unlabelled $k$ vertices. Denote $M' := T' \backslash L$, $M'' := T'' \backslash L$ be the sets of different vertex sets between the two unlabelled vertex sets. Then,

$$\Delta(G', G'') = \begin{cases} 2^l & \text{if } (S_1'' \backslash M', S_2'' \backslash M') \text{ is matched by} \\ & (S_1' \backslash M'', S_2' \backslash M'') \text{ or } (S_2' \backslash M'', S_1' \backslash M'') \\ 0 & \text{otherwise} \end{cases}$$

# Counting – Edge Contraction Algorithm

- In step $t$, there are $k$ "**super**" vertices, get $|C_t^{\mathscr{A}}(G^{'})| = |C_t^{\mathscr{A}}(G^{''})| = 2^{k-1} - 1$

# Counting – Edge Contraction Algorithm

- In step $t$, there are $k$ "**super**" vertices, get $|C_t^{\mathscr{A}}(G^{'})| = |C_t^{\mathscr{A}}(G^{''})| = 2^{k-1} - 1$

- We propose polynomial time algorithm (and prove correctness) to exactly count $|C_t^{\mathscr{A}}(G^{'}) \cap C_t^{\mathscr{A}}(G^{''})|$

# Counting – Edge Contraction Algorithm

- In step $t$, there are $k$ "**super**" vertices, get $|C_t^{\mathscr{A}}(G')| = |C_t^{\mathscr{A}}(G'')| = 2^{k-1} - 1$

- We propose polynomial time algorithm (and prove correctness) to exactly count $|C_t^{\mathscr{A}}(G') \cap C_t^{\mathscr{A}}(G'')|$

---

**Algorithm 3:** Common Super Vertex Counting

**Input**: Two distinct super vertex sets $P$, $Q$
**Output**: Maximum number of common super vertices after all possible contractions

$c := 0$;
**while** $P \neq \emptyset$ **do**
  Randomly pick $\mathbf{p}_i \in P$;
  Find $\mathbf{q}_j \in Q$ s.t. $\mathbf{p}_i \cap \mathbf{q}_j \neq \emptyset$;
  **if** $\mathbf{q}_j \backslash \mathbf{p}_i \neq \emptyset$ **then**
    For $\mathbf{p}_i$, find $\mathbf{p}_{i'} \in P \backslash \{\mathbf{p}_i\}$ s.t. $\mathbf{p}_{i'} \cap (\mathbf{q}_j \backslash \mathbf{p}_i) \neq \emptyset$;
    $\mathbf{p_{ii'}} := \mathbf{p}_i \cup \mathbf{p}_{i'}$, $P := P \cup \{\mathbf{p_{ii'}}\} \backslash \{\mathbf{p}_i, \mathbf{p}_{i'}\}$ ;
  **if** $\mathbf{p}_i \backslash \mathbf{q}_j \neq \emptyset$ **then**
    For $\mathbf{q}_j$, find $\mathbf{q}_{j'} \in Q \backslash \{\mathbf{q}_j\}$ s.t. $\mathbf{q}_{j'} \cap (\mathbf{p}_i \backslash \mathbf{q}_j) \neq \emptyset$;
    $\mathbf{q_{jj'}} := \mathbf{q}_j \cup \mathbf{q}_{j'}$, $Q := Q \cup \{\mathbf{q_{jj'}}\} \backslash \{\mathbf{q}_j, \mathbf{q}_{j'}\}$ ;
  **if** $\mathbf{p_{ii'}} == \mathbf{q_{jj'}}$ **then**
    Remove $\mathbf{p_{ii'}}$, $\mathbf{q_{jj'}}$ from $P$, $Q$, respectively;
    $c := c + 1$;
**return** $c$

# Counting – Edge Contraction Algorithm

- In step $t$, there are $k$ "**super**" vertices, get $|C_t^{\mathscr{A}}(G')| = |C_t^{\mathscr{A}}(G'')| = 2^{k-1} - 1$
- We propose polynomial time algorithm (and prove correctness) to exactly count $|C_t^{\mathscr{A}}(G') \cap C_t^{\mathscr{A}}(G'')|$
- Involves calculating max. number of common super vertices between 2 super vertex sets (details in the paper)

**Algorithm 3:** Common Super Vertex Counting

**Input**: Two distinct super vertex sets $P$, $Q$
**Output**: Maximum number of common super vertices after all possible contractions

$c := 0$;
**while** $P \neq \emptyset$ **do**
  Randomly pick $\mathbf{p}_i \in P$;
  Find $\mathbf{q}_j \in Q$ s.t. $\mathbf{p}_i \cap \mathbf{q}_j \neq \emptyset$;
  **if** $\mathbf{q}_j \backslash \mathbf{p}_i \neq \emptyset$ **then**
    For $\mathbf{p}_i$, find $\mathbf{p}_{i'} \in P \backslash \{\mathbf{p}_i\}$ s.t. $\mathbf{p}_{i'} \cap (\mathbf{q}_j \backslash \mathbf{p}_i) \neq \emptyset$;
    $\mathbf{p}_{\mathbf{ii'}} := \mathbf{p}_i \cup \mathbf{p}_{i'}$, $P := P \cup \{\mathbf{p}_{\mathbf{ii'}}\} \backslash \{\mathbf{p}_i, \mathbf{p}_{i'}\}$ ;
  **if** $\mathbf{p}_i \backslash \mathbf{q}_j \neq \emptyset$ **then**
    For $\mathbf{q}_j$, find $\mathbf{q}_{j'} \in Q \backslash \{\mathbf{q}_j\}$ s.t. $\mathbf{q}_{j'} \cap (\mathbf{p}_i \backslash \mathbf{q}_j) \neq \emptyset$;
    $\mathbf{q}_{\mathbf{jj'}} := \mathbf{q}_j \cup \mathbf{q}_{j'}$, $Q := Q \cup \{\mathbf{q}_{\mathbf{jj'}}\} \backslash \{\mathbf{q}_j, \mathbf{q}_{j'}\}$ ;
  **if** $\mathbf{p}_{\mathbf{ii'}} == \mathbf{q}_{\mathbf{jj'}}$ **then**
    Remove $\mathbf{p}_{\mathbf{ii'}}$, $\mathbf{q}_{\mathbf{jj'}}$ from $P$, $Q$, respectively;
    $c := c + 1$;
**return** $c$

# Counting – Edge Contraction Algorithm

- In step $t$, there are $k$ "**super**" vertices, get $|C_t^{\mathscr{A}}(G')| = |C_t^{\mathscr{A}}(G'')| = 2^{k-1} - 1$

- We propose polynomial time algorithm (and prove correctness) to exactly count $|C_t^{\mathscr{A}}(G') \cap C_t^{\mathscr{A}}(G'')|$

- Involves calculating max. number of common super vertices between 2 super vertex sets (details in the paper)

**Algorithm 3:** Common Super Vertex Counting

**Input**: Two distinct super vertex sets $P$, $Q$
**Output**: Maximum number of common super vertices after all possible contractions

$c := 0$;
**while** $P \neq \emptyset$ **do**
  Randomly pick $\mathbf{p}_i \in P$;
  Find $\mathbf{q}_j \in Q$ s.t. $\mathbf{p}_i \cap \mathbf{q}_j \neq \emptyset$;
  **if** $\mathbf{q}_j \backslash \mathbf{p}_i \neq \emptyset$ **then**
    For $\mathbf{p}_i$, find $\mathbf{p}_{i'} \in P \backslash \{\mathbf{p}_i\}$ s.t. $\mathbf{p}_{i'} \cap (\mathbf{q}_j \backslash \mathbf{p}_i) \neq \emptyset$;
    $\mathbf{p_{ii'}} := \mathbf{p}_i \cup \mathbf{p}_{i'}$, $P := P \cup \{\mathbf{p_{ii'}}\} \backslash \{\mathbf{p}_i, \mathbf{p}_{i'}\}$ ;
  **if** $\mathbf{p}_i \backslash \mathbf{q}_j \neq \emptyset$ **then**
    For $\mathbf{q}_j$, find $\mathbf{q}_{j'} \in Q \backslash \{\mathbf{q}_j\}$ s.t. $\mathbf{q}_{j'} \cap (\mathbf{p}_i \backslash \mathbf{q}_j) \neq \emptyset$;
    $\mathbf{q_{jj'}} := \mathbf{q}_j \cup \mathbf{q}_{j'}$, $Q := Q \cup \{\mathbf{q_{jj'}}\} \backslash \{\mathbf{q}_j, \mathbf{q}_{j'}\}$ ;
  **if** $\mathbf{p_{ii'}} == \mathbf{q_{jj'}}$ **then**
    Remove $\mathbf{p_{ii'}}$, $\mathbf{q_{jj'}}$ from $P$, $Q$, respectively;
    $c := c + 1$;
**return** $c$

**Theorem 1.** *Given two distinct super vertex sets* $P := \{\mathbf{p}_1, \mathbf{p}_2, \cdots, \mathbf{p}_h\}$, $Q := \{\mathbf{q}_1, \mathbf{q}_2, \cdots, \mathbf{q}_h\}$ *(any 2 super vertices inside* $P$ *or* $Q$ *do not intersect, and there is no common super vertex between* $P$ *and* $Q$*), such that* $\mathbf{p}_1 \cup \mathbf{p}_2 \cup \cdots \cup \mathbf{p}_h = \mathbf{q}_1 \cup \mathbf{q}_2 \cup \cdots \cup \mathbf{q}_h$*, Alg. 3 returns the maximum number of common super vertices between* $P$ *and* $Q$ *after all possible contractions.*

# Contents

# Noise Model: Gaussian Edge Weights

**Master Graph** $G$
Gaussian distributed edge weights:

$$W_{ij} \sim N(\mu, \sigma_m^2), \mu = 600, \sigma_m = 50$$

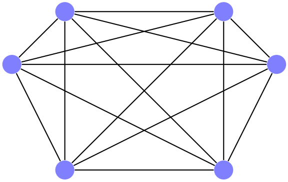
Negative edges are set to be $\mu$.

# Noise Model: Gaussian Edge Weights

**Master Graph** $G$
Gaussian distributed edge weights:

$$W_{ij} \sim N(\mu, \sigma_m^2), \mu = 600, \sigma_m = 50$$

Negative edges are set to be $\mu$.



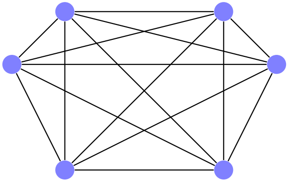
Master graph $G$ with Gaussian weights

# Noise Model: Gaussian Edge Weights

**Master Graph $G$**
Gaussian distributed edge weights:

$$W_{ij} \sim N(\mu, \sigma_m^2), \mu = 600, \sigma_m = 50$$

Negative edges are set to be $\mu$.



Master graph $G$ with Gaussian weights

**Noisy Graphs $G^{'}$, $G^{''}$**
$G^{'}$, $G^{''}$ are obtained by adding Gaussian distributed noise.
Negative edges are set to be 0.

# Noise Model: Edge Reversal

**Master Graph** $G$

# Noise Model: Edge Reversal

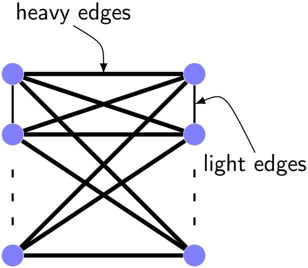
**Master Graph** $G$

1. approximate bipartite $G'_b$: *light edges*, *heavy edges*

# Noise Model: Edge Reversal

**Master Graph** $G$

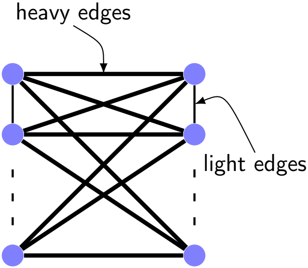
1. approximate bipartite $G'_b$: *light edges*, *heavy edges*




Approximate bipartite graph $G'_b$

**Master Graph** $G$

1. approximate bipartite $G'_b$: *light edges*, *heavy edges*
2. randomly flip edges in $G'_b \Rightarrow G$,
   flipping: heavy (light) $\Rightarrow$ light (heavy)
   $(\text{flip } e_{ij}) \sim \text{Ber}(p_m)$; $p_m = 0.2$

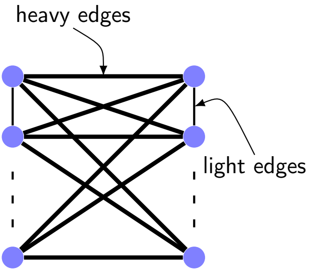



Approximate bipartite graph $G'_b$

# Noise Model: Edge Reversal

**Master Graph $G$**

1. approximate bipartite $G_b'$: *light edges*, *heavy edges*
2. randomly flip edges in $G_b' \Rightarrow G$, flipping: heavy (light) $\Rightarrow$ light (heavy) $(\text{flip } e_{ij}) \sim \text{Ber}(p_m)$; $p_m = 0.2$



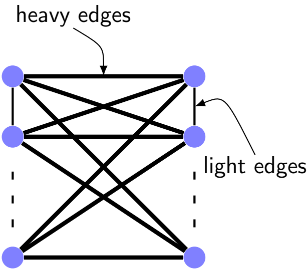

Approximate bipartite graph $G_b'$

**Noisy Graphs $G^{'}$, $G^{''}$**

# Noise Model: Edge Reversal

**Master Graph** $G$

1. approximate bipartite $G'_b$: *light edges*, *heavy edges*
2. randomly flip edges in $G'_b \Rightarrow G$,
   flipping: heavy (light) $\Rightarrow$ light (heavy)
   $(\text{flip } e_{ij}) \sim \text{Ber}(p_m)$; $p_m = 0.2$




`Approximate bipartite graph` $G'_b$

**Noisy Graphs** $G^{'}$, $G^{''}$

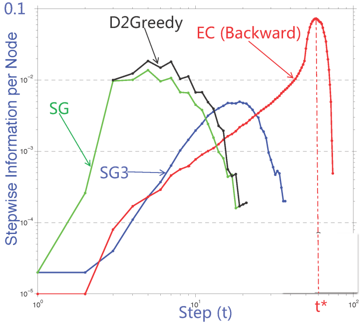
- Flip $G \Rightarrow G^{'}$ and $G^{''}$.
  Probability of flipping an edge: Bernoulli distribution with $p$,

$$(\text{flip } e_{ij}) \sim \text{Ber}(p)$$

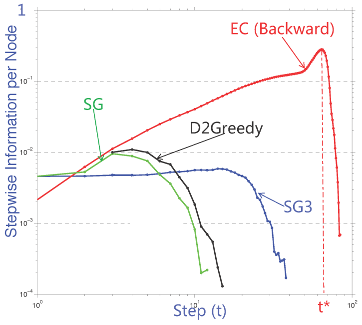
$p$: noise level

# Stepwise Information $I_t^{\mathscr{A}}$

$$I_t^{\mathscr{A}} := \mathbb{E}_{G',G''}\left[\log\left(\frac{|\mathcal{C}|\cdot|\Delta C_t^{\mathscr{A}}(G',G'')|}{|C_t^{\mathscr{A}}(G')|\cdot|C_t^{\mathscr{A}}(G'')|}\right)\right]$$



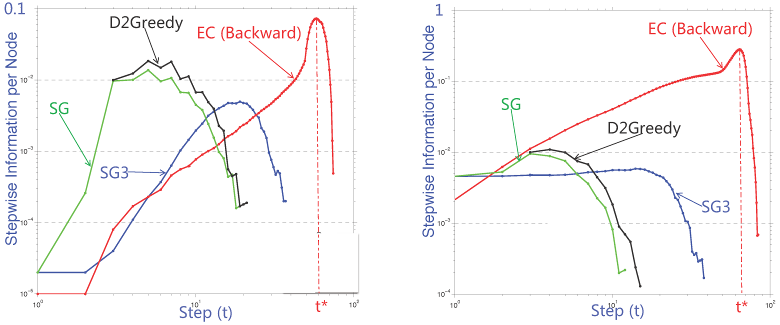

Gaussian Model, $\sigma = 125$




Edge Reversal, $p = 0.65$

# Stepwise Information $I_t^{\mathscr{A}}$

$$I_t^{\mathscr{A}} := \mathbb{E}_{G',G''}\left[\log\left(\frac{|\mathcal{C}|\cdot|\Delta C_t^{\mathscr{A}}(G',G'')|}{|C_t^{\mathscr{A}}(G')|\cdot|C_t^{\mathscr{A}}(G'')|}\right)\right]$$

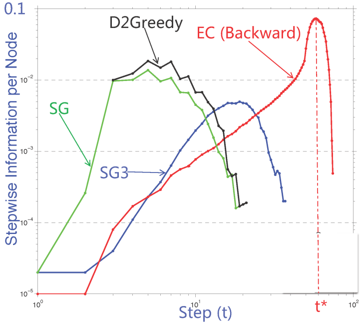



Gaussian Model, $\sigma = 125$ Edge Reversal, $p = 0.65$

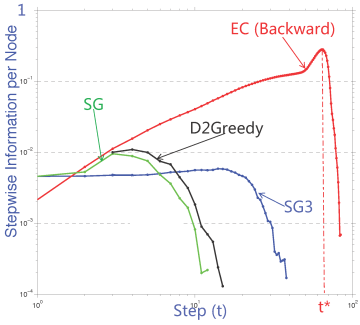
- $I_t^{\mathscr{A}}$ behavior: increase initially $\Rightarrow$ reach the optimal step $t^*$ $\Rightarrow$ decreases $\Rightarrow$ vanishes.

# Stepwise Information $I_t^{\mathscr{A}}$

$$I_t^{\mathscr{A}} := \mathbb{E}_{G',G''}\left[\log\left(\frac{|\mathcal{C}|\cdot|\Delta C_t^{\mathscr{A}}(G',G'')|}{|C_t^{\mathscr{A}}(G')|\cdot|C_t^{\mathscr{A}}(G'')|}\right)\right]$$




Gaussian Model, $\sigma = 125$

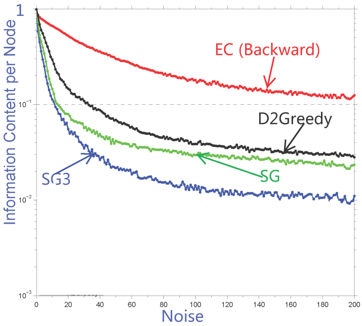



Edge Reversal, $p = 0.65$

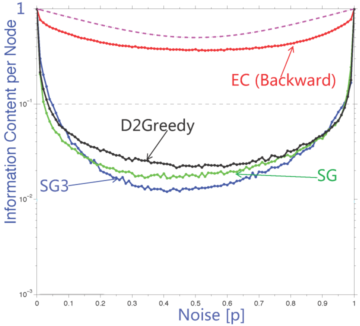
- $I_t^{\mathscr{A}}$ behavior: increase initially $\Rightarrow$ reach the optimal step $t^*$ $\Rightarrow$ decreases $\Rightarrow$ vanishes.
- consistent with analysis: $\nearrow t \Rightarrow$ tradeoff of roubstness and informativeness

# Information Content $I^{\mathscr{A}}$

$I^{\mathscr{A}} := \max_t I_t^{\mathscr{A}}$ (channel capacity)



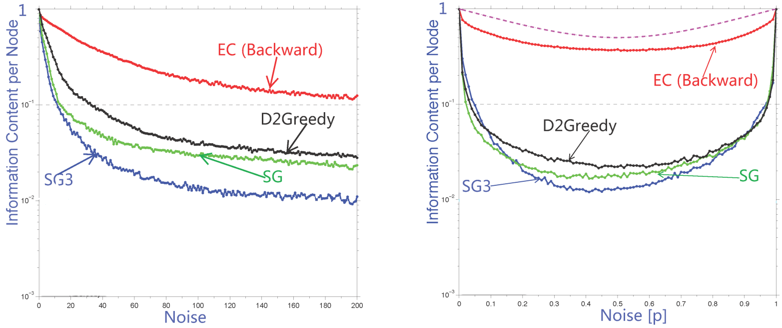

Gaussian Edge Weights Model




Edge Reversal Model

# Information Content $I^{\mathscr{A}}$

$I^{\mathscr{A}} := \max_t I_t^{\mathscr{A}}$ (channel capacity)

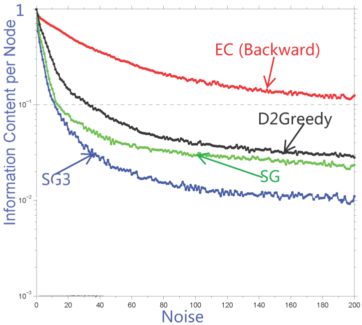



Gaussian Edge Weights Model        Edge Reversal Model

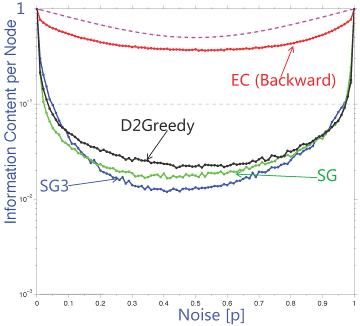
- All reach max. information content in the noise free limit ($G' = G''$) ($p = 0, 1$ in edge reversal model, $\sigma = 0$ in Gaussian model)

# Information Content $I^{\mathscr{A}}$

$I^{\mathscr{A}} := \max_t I_t^{\mathscr{A}}$ (channel capacity)




Gaussian Edge Weights Model

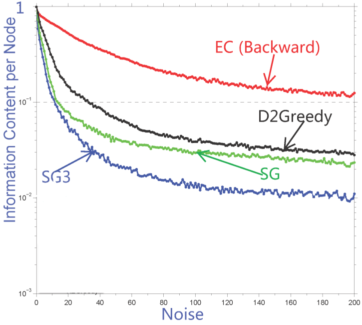



Edge Reversal Model

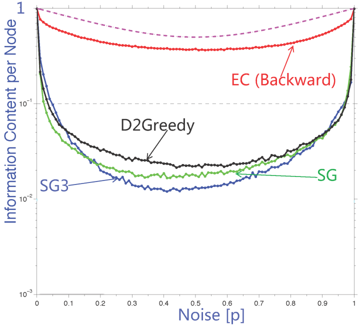
- All reach max. information content in the noise free limit $(G' = G'')$ $(p = 0, 1$ in edge reversal model, $\sigma = 0$ in Gaussian model)
- 1 node transmits about 1 bit information

# Effect of Greedy Heuristics

Backward greedy $\succcurlyeq$ double greedy



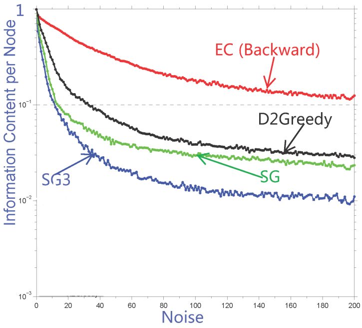

Gaussian Edge Weights Model



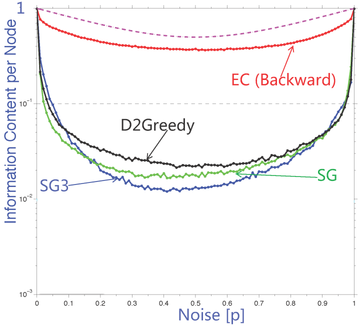

Edge Reversal Model

# Effect of Greedy Heuristics

Backward greedy $\succcurlyeq$ double greedy




**Gaussian Edge Weights Model**

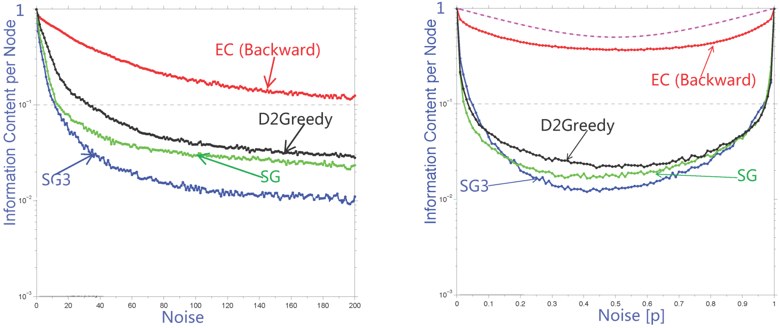



**Edge Reversal Model**

- Delayed decision making of backward greedy

# Effect of Greedy Heuristics

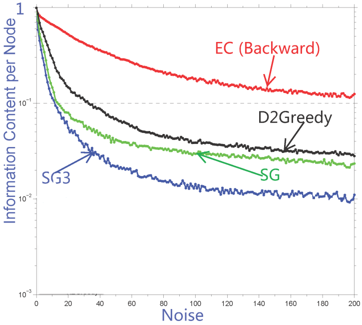
Backward greedy $\succcurlyeq$ double greedy



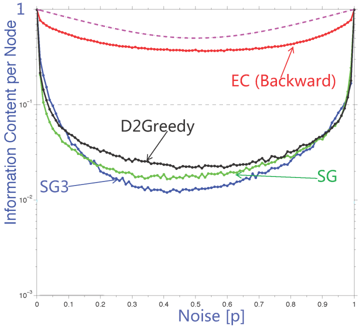

Gaussian Edge Weights Model    Edge Reversal Model

- Delayed decision making of backward greedy
- EC preserves consistent solutions by contracting lightest edge (having low probability to be included in the cut)

# Effect of Greedy Techniques

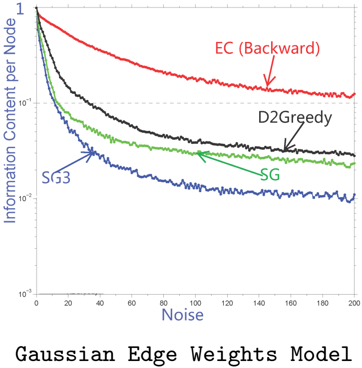



Gaussian Edge Weights Model



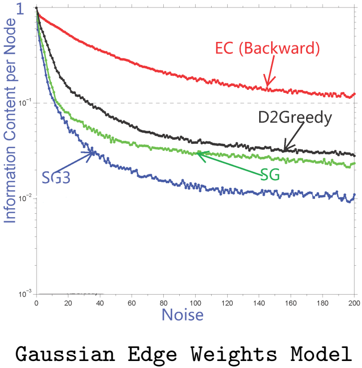

Edge Reversal Model

# Effect of Greedy Techniques




- Initializing (D2Greedy ⇒ SG): ↘, due to early decision making

# Effect of Greedy Techniques




- Initializing (D2Greedy ⇒ SG): ↘, due to early decision making
- Sorting candidates (SG ⇒ SG3): ↘, due to early decision making

# Discussion

- **Observation**:
  Different greedy heuristics (backward, double) and different processing techniques (sorting candidates, initializing the first 2 vertices) sensitively influence the information content of $\mathscr{A}$.

# Discussion

- **Observation**:
  Different greedy heuristics (backward, double) and different processing techniques (sorting candidates, initializing the first 2 vertices) sensitively influence the information content of $\mathscr{A}$.
- **Conjecture**:
  Backward greedy $\overset{\text{delayed decision making}}{\succeq}$ double greedy
  for different noise models and noise levels.

# Thank you!

Qs?

# Supplement: Analogy of Communication System

Imaginary communication system:

- message: permutations $\sigma_s \in \Sigma$ on the data space
- encoder: encoding $\sigma_s$ using $C_t^{\mathscr{A}}(\sigma_s \circ G')$ (codebook vector)
- channel: noisy instances $G'$, $G''$
- decoder: max. overlap of approx. sets:
  $\hat{\sigma} := \arg\max_{\sigma \in \Sigma} |C_t^{\mathscr{A}}(\sigma \circ G'') \cap C_t^{\mathscr{A}}(\sigma_s \circ G')|$

## Analogical mutual information in step $t$

$$I_t^{\mathscr{A}}(\sigma_s; \hat{\sigma}) := \mathbb{E}_{G', G''}\left[\log\left(|\mathcal{C}| \frac{|C_t^{\mathscr{A}}(G') \cap C_t^{\mathscr{A}}(G'')|}{|C_t^{\mathscr{A}}(G')| \cdot |C_t^{\mathscr{A}}(G'')|}\right)\right]$$

channel capacity $I^{\mathscr{A}} := \max_t I_t^{\mathscr{A}}$ (Information content of $\mathscr{A}$ )