
Detecting Automatic Software Plagiarism via Token Sequence Normalization

.....

Timur Sağlam, Moritz Brödel, Larissa Schmid, Sebastian Hahner

firstname.lastname@kit.edu

Karlsruhe Institute of Technology (KIT)

Karlsruhe, Germany

ソフトウェア盗用

| コンピュータサイエンス教育で多く発見される

- 学生は不正だと認識しながらも盗用をおこなう学生も存在
 - ここでの盗用は学生間での盗用を指す

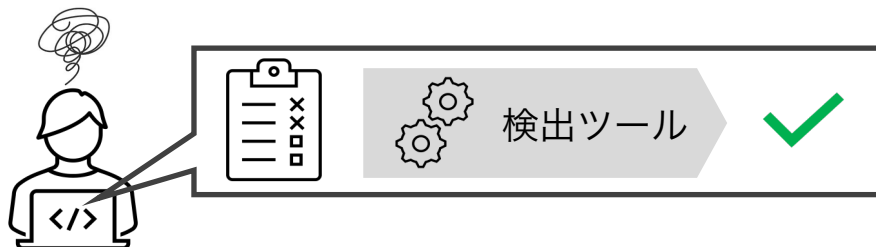
| 盗用検出

- 初学者向けの授業は規模が大きいため手作業による検出は実用的でない
 - データ活用への関心の高まり，プログラミング教育の義務化
- 手作業による検出は一人一人の検出精度が低下する
- MOSSやJplagなどの盗用検出ツールが世界中の大学で使用されている

ソフトウェア盗用の自動検出ツール

自動検出ツール活用目的

- 検出ツールを回避するため（難読化）の労力 > 自身でコードを書く労力
 - 検出を手作業で回避することができるなら既に十分なスキルを持っている



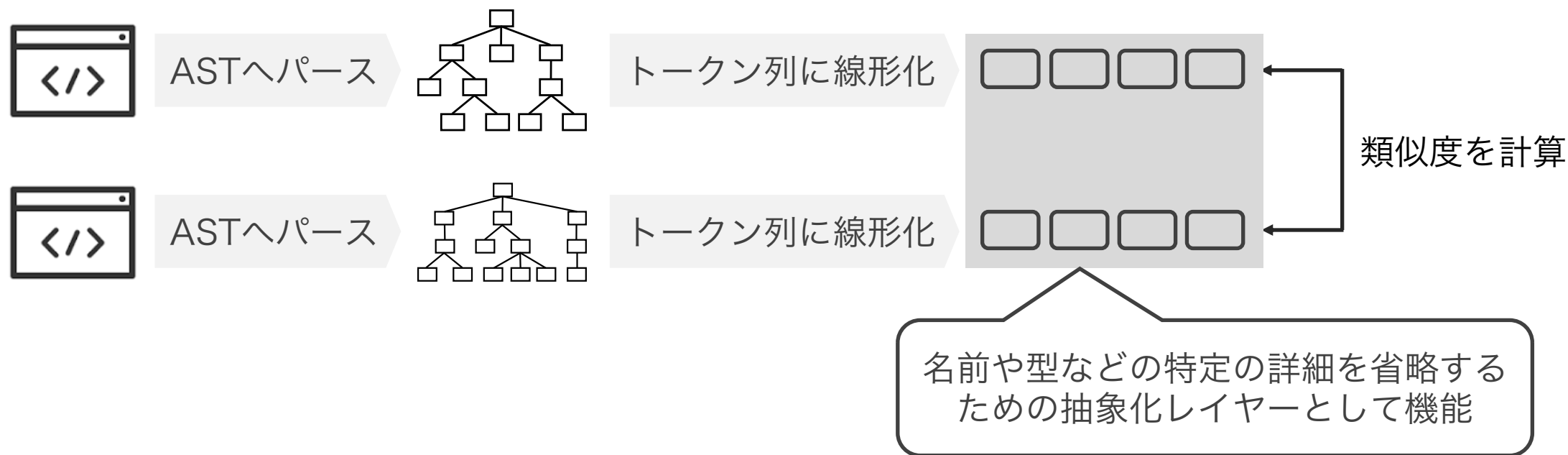
盗用ジェネレータの台頭

- ほとんどの検出ツールは難読化攻撃に頑健に作用
- MOSSAD（最新の盗用ジェネレータ）はトークンベースアプローチの検出ツールの脆弱性を利用して検出を回避

最先端の盗用検出ツール

最先端の盗用検出ツール

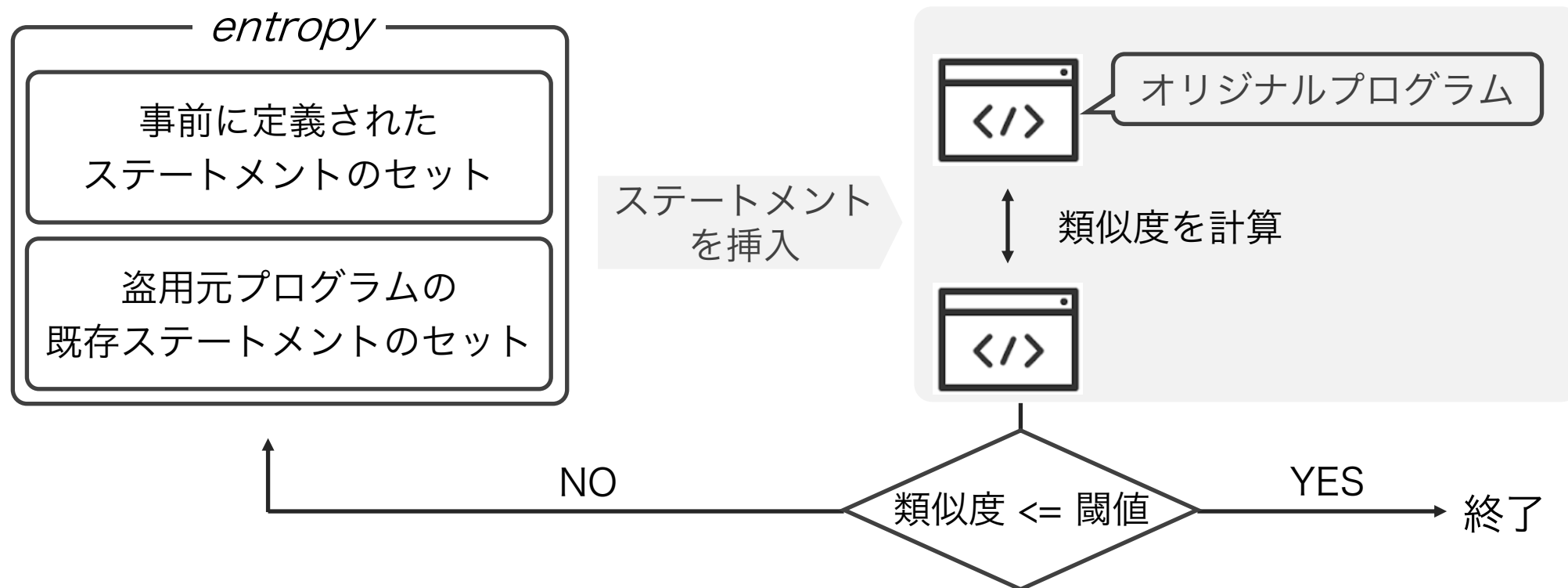
- トークンベースでコード構造を比較し、コード間の類似性を計算



- 抽象化をおこなうことで変数名のリネーム、再入力などの難読化に強くなる

最先端の盗用ジェネレータによる難読化

- 盗用元のプログラムにステートメントを繰り返し挿入をおこなう



トークンベース検出ツールの性質上の欠点を利用
トークンの並びを比較するので挿入、並び替えなどの難読化に弱い

トークン化

1	<code>void printSquares() {</code>		<code>void printSquares() {</code>
2	<code>int i = 1;</code>		<code>int i = 1;</code>
3		(+)	<code>boolean debug = false;</code>
4	<code>while (i <= 10) {</code>		<code>while (i <= 10) {</code>
5	<code>int square = i * i;</code>		<code>int square = i * i;</code>
6	<code>println(square);</code>	(~)	<code>i++;</code>
7	<code>i++;</code>	(~)	<code>println(square);</code>
8	<code>}</code>		<code>}</code>
9	<code>}</code>		<code>}</code>



トークン列

#	Original Tokens	→	Variant Tokens
1	method start		method start
2	variable	挿入	variable
3		(+)	variable
4	loop start		loop start
5	variable	並び替え	variable
6	apply	(~)	assignment
7	assignment	(~)	apply
8	loop end		loop end
9	method end		method end

構造情報は残っているが、
名前や型などの詳細は除外

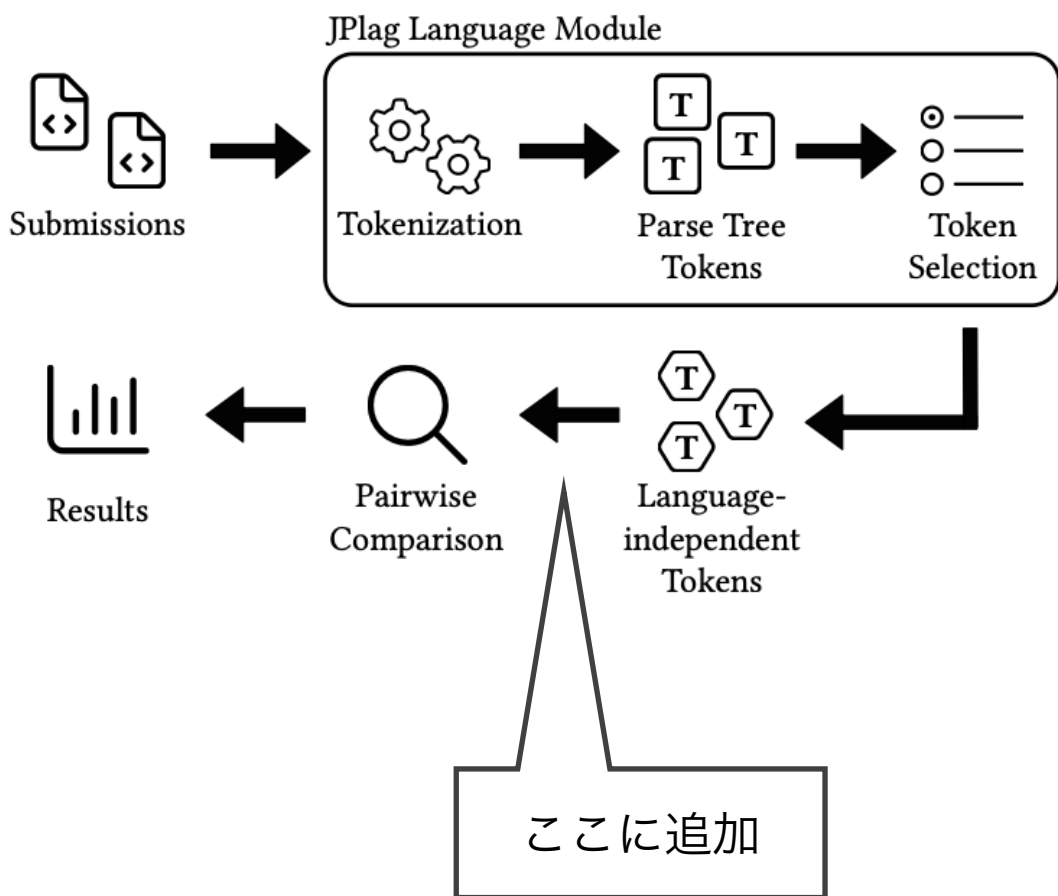
本研究のモチベーション

名前や型に対する難読化は抽象トークン化によって対応できているため
挿入や並び替えの難読化に対応したい

提案手法

アプローチ

- 既存の検出ツールにトークン正規化グラフ（TNG）を追加



追加するステップの概要

1. トークン列内のトークン間の相互依存関係情報を付与
2. トークン化されたプログラムの言語に依存しないグラフベースの表現（TNG）を構築
3. グラフを使って正規化されたトークン列を生成

トークン正規化グラフ (TNG)

相互依存関係に関する意味情報の付与

- トークンはプログラムを最小単位に分割しただけなので意味的な情報がない
- 以下の4つの情報をトークンレベルで付与

Criticality

変数以外でプログラムの挙動に寄与するトークンであり、削除してはいけないトークン

Loop

ステートメントの実行順序が宣言順序からズれる可能性のあるトークン

Fixed Order

意味上の等価性を保証するために、相対的な順序を維持する必要があるトークン

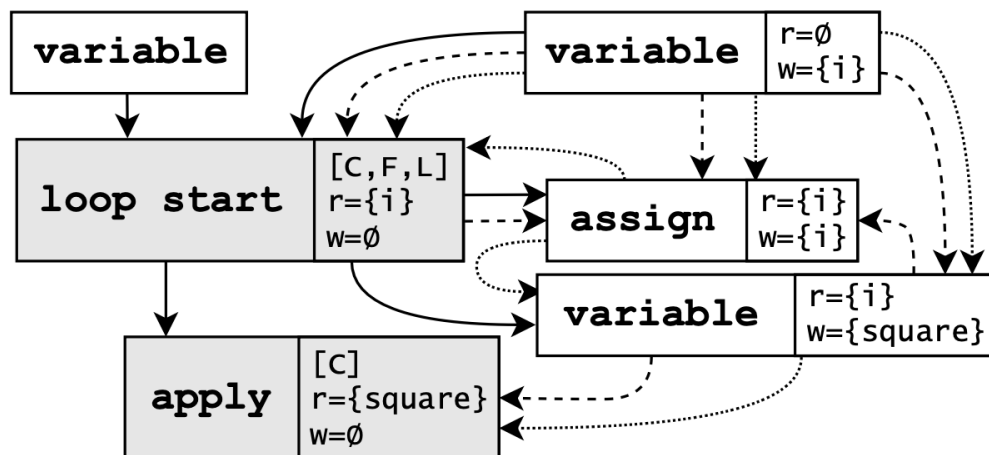
Variable Access

各ステートメントの変数アクセスを記述し、他のどのステートメントに依存しているかを示す

トークン正規化グラフ (TNG)

TNGの構築

- ステートメント内の全てのトークンをノードしてグループ化
- “Fixed Order”ラベルを持つトークンを使用して順序エッジを作成する
- “Variable Access”と”Loop”を使用して可変フローエッジを作成
- “Loop”を無視して”Variable Access”だけで可変順序エッジを作成



- 可変フローエッジ（点線） : あるステートメントが別のステートメントの変数に書き込む
- 可変順序エッジ（破線） : 2つのステートメントの順序を変更すると挙動が変更する可能性がある
- 固定順序エッジ（実線） : 2つのステートメントの順序を変更すると変数以外の理由で挙動が変更する可能性がある

トークン正規化グラフ (TNG)

トークン正規化列の生成

1. デッドノードと可変フローエッジの削除

- 可変フローエッジを経由して”Criticality”ラベルを持つノードに辿り着けない全てのノード
- デッドノードは冗長な挿入がされた場合に存在する

2. トポロジカルソートを用いて正規化されたトークン列の生成

#	Original	→	Obfuscated	→	Nodes Removed	→	Top. Sorted
1	method start		method start		method start		method start
2	variable		variable		variable		variable
3		(+)	variable	(-)			
4	loop start		loop start		loop start		loop start
5	variable		variable		variable		variable
6	apply	(~)	assignment		assignment	(~)	apply
7	assignment	(~)	apply		apply	(~)	assignment
8	loop end		loop end		loop end		loop end
9	method end		method end		method end		method end

- | RQ1 : 提案手法は, MOSSADによる難読化を効果的に検出できるか
- | RQ2 : 一般的な挿入/並び替え/複合などの難読化に対する耐性があるか
- | RQ3 : 提案手法は検出ツールの誤検出にどのような影響を与えるか
 - TNGの追加によって関連しないプログラムの類似度が向上することはない
- | RQ4 : 提案手法は検出ツールのパフォーマンスにどう影響を与えるか
 - TNGの有無による実行時間を比較してパフォーマンスはほぼ影響しない

データセット

| PROGprediaから2つの課題

- 多数ある課題の中で他の課題よりもプログラムが長く解答数が多かった
- Javaの解答に限定
 - 1つ目の課題：27の投稿 / 平均LOC=131
 - 2つ目の課題：28の投稿 / 平均LOC=85

| 著者が担当している大学のJavaプログラミング演習から2つの課題

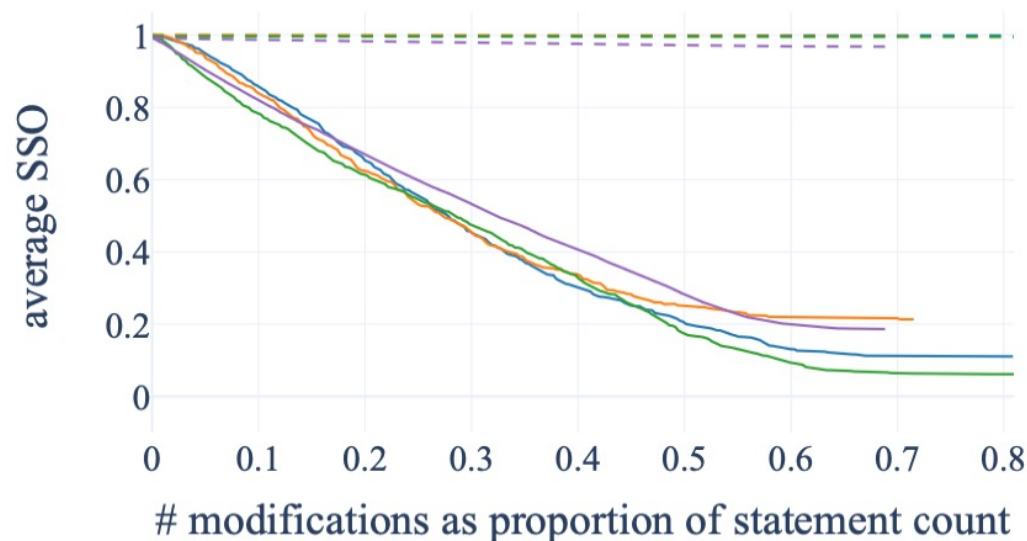
- PROGprediaはコードの規模が比較的小さいため
 - 1つ目の課題：626の投稿 / 平均LOC=236
 - 2つ目の課題：434の投稿 / 平均LOC=1529

盗用ジェネレータにはMOSSADを使用

RQ1 | 提案手法は、MOSSADによる難読化を効果的に検出できるか

結果

- 提案手法がない場合、Jplagは4つのデータセットの平均SSOを25%未満と計算
- グラフより、提案手法有りの場合提案手法無しの場合の平均SSOに並ぶには50から60%のステートメント挿入が必要
 - 冗長なステートメントが50%も追加されたコードは十分難読化を防御できていると言える



a) insertion-based

※ SSO : Sililarity score to Original

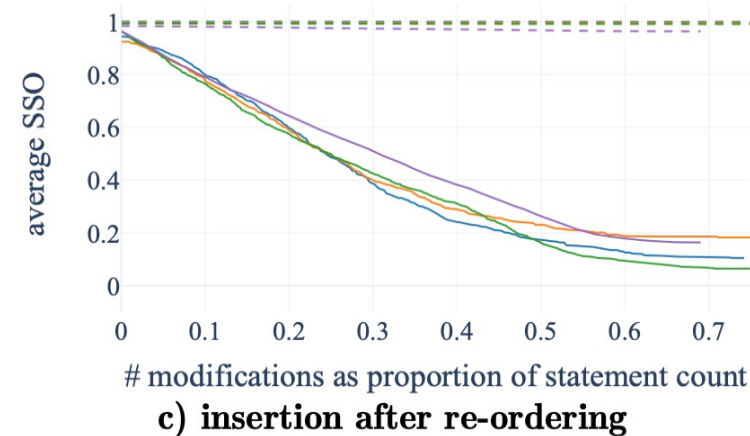
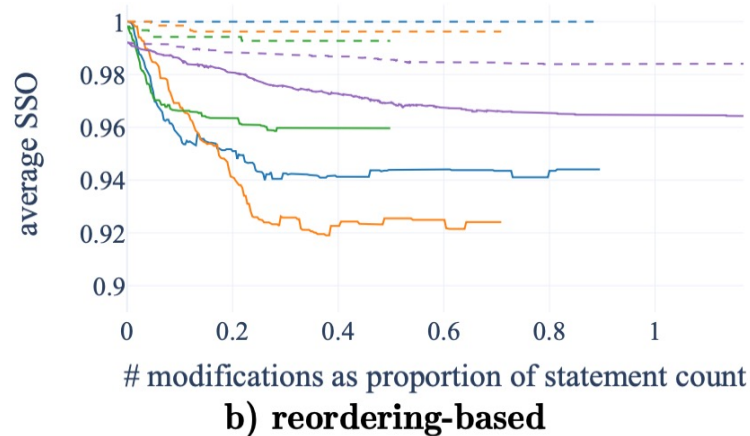
RQ2 | 一般的な並び替え/複合などの難読化に対する耐性があるか

| 並び替え

- 並び替え難読化に対しては効果が大い
- どのデータセットでも平均SSOが0.9を下回ることはない

| 複合（挿入と並び替え）

- 挿入の結果と酷似しており挿入の結果に大きく影響を受けている可能性が高い
- 提案手法なしの場合と比べて平均SSOは高い値を取っている



- | 4つの意味情報でここまで高性能な正規化グラフを構築できるのはなかなか良くできているなと感心した
- | TNGが変数のリネームなど既存手法で対応できていたものにも対応できることを示してほしかった
 - TNGという余計な正規化を挟むことで今まで保持できていた情報が失われる可能性はない？