

git2net - Mining Time-Stamped Co-Editing Networks from Large git Repositories

git2net - 大規模なgitリポジトリからの タイムスタンプ付き共同編集ネットワークのマイニング

著者

Christoph Gote, Frank Schweitzer
Chair of Systems Design ETH Zurich
Zurich, Switzerland

Ingo Scholtes
Data Analytics Group Department of Informatics
University of Zurich Zurich, Switzerland

紹介者：堀口日向

1. 背景

ソフトウェアリポジトリのデータは、
ソフトウェアエンジニアリングプロセスの
研究のためよく使われる



GitHub

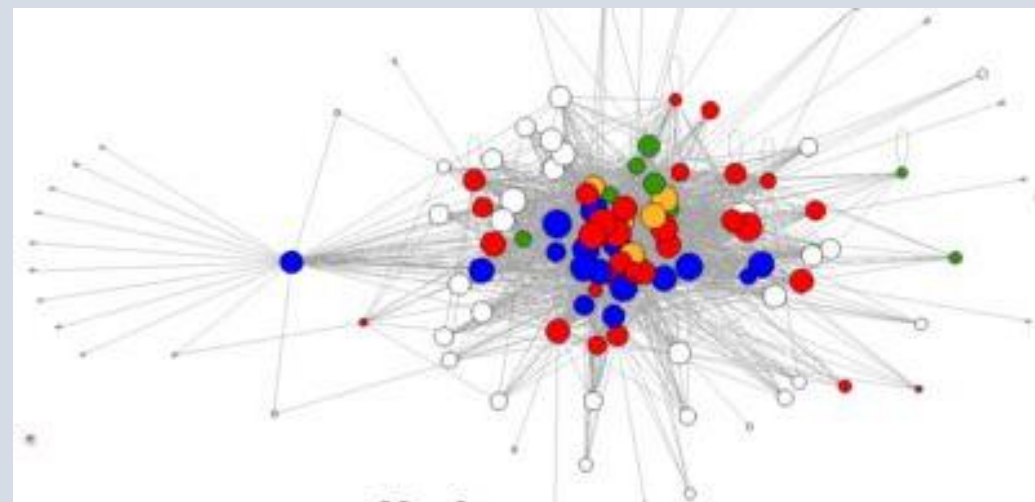


GitLab



Bitbucket

プロジェクトのコミット履歴を使えば
開発者間のコラボレーション、調整
コミュニケーション構造を
ソーシャルネットワークで構築できる



しかし、現状ではそのようなコラボレーションネットワークを
容易に抽出するためのツールがない

2. 関連研究

ソフトウェア開発チームにおける開発者間のコラボレーションネットワークを分析するこれまでの研究は開発者が共通のファイルを編集した場合に開発者間にリンクを張る無向ネットワークを構築し分析

そのようなネットワークでは

- ・複数の開発者が同じファイルを編集してはいるが、実際の編集箇所は異なる場合
- ・ある開発者が他の開発者によって以前に作成されたソースコードの特定の行を編集した場合

これらのリンクの違いが区別できない。前者と後者ではネットワークの解釈が異なる

後者のリンクは、開発者が同じファイルまたはモジュールを編集したという単なる事実よりも調整とコミュニケーションの必要性が強いことに関連している

- ・ M. Joblinら[1]は、ファイルベースのコラボレーションネットワークを使用すると意味のあるコミュニティを識別できないと主張
- ・ I. Scholtesら[2]は、ファイル単位ではなく、行単位で編集箇所を区別してリンクを張るネットワークを構築
そのような分析は
 - ・ ソフトウェアチームの調整構造への洞察をもたらす
 - ・ 社会心理学からのコラボレーションに関する長年の仮説を検証する新しい方法を提供すると主張

しかし、行単位での分析には、多数のgit操作を必要とするので、処理が複雑であると同時にghTorrentなどを使った方法に比べて、実行に時間がかかる
非常に大規模なリポジトリでこのタスクを容易にするツールはない

3. 目的

git2netを開発し、以下の機能を実装

- コミットログに含まれるすべてのファイルの変更を分析して、共同編集されたコードに関する行単位で抽出するデータベースを生成する
- タイムスタンプ付きコラボレーションネットワークの生成
 - (i) タイムスタンプ付き共同編集ネットワーク
 - (ii) 開発者を編集済みファイルにリンクするタイムスタンプ付き2部ネットワーク
 - (iii) 相互に構築された連続した編集の「パス」を推測できるコード編集の非循環グラフ
- 既存のネットワーク分析ツールで読み取り可能な形式のネットワークとしてエクスポート
- 並列処理で巨大なリポジトリの分析をサポート

git2netを利用して、OSSと商用の2つのプロジェクトのケーススタディを行い
このツールをどのように使用できるかを示す

4. git2netの実装 (1/4)

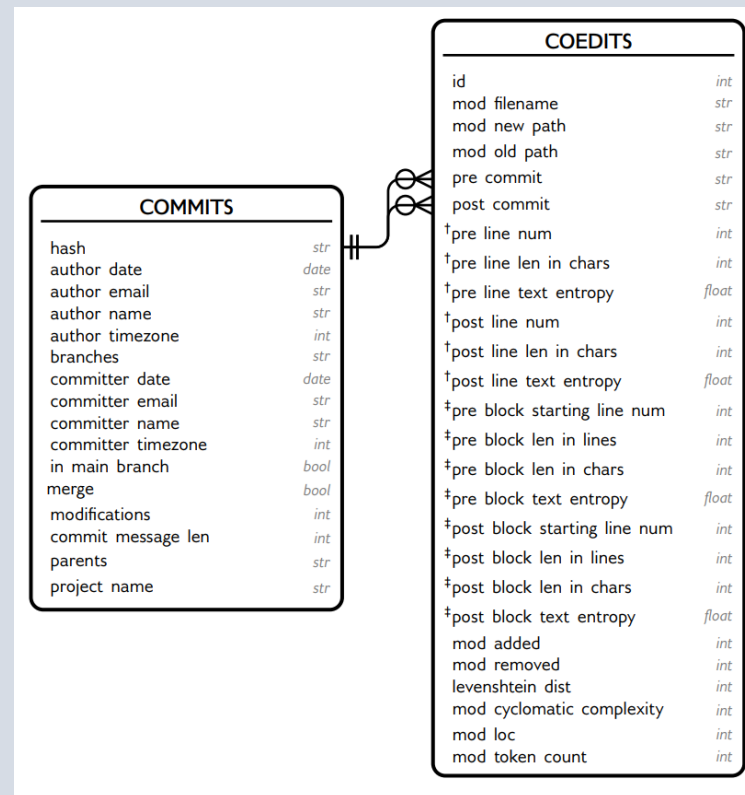
- コミットログを分析して開発者のコラボレーションを抽出
 - リポジトリから各コミットのメタデータ
(作成者、コミット時刻、ブランチなど) を記録
 - コミットには複数のファイルの変更が含まれる可能性があるため
各ファイルの変更を個別に分析
 1. どのコード行が追加または削除されたかが判断
 2. 編集されたすべてのコード行の元の作者を記録
 - 変更の大きさをレーベンシュタイン距離として記録
リンクの重みづけに使う

記録はsqliteデータベースに保存

これらの記録を利用することで
時刻tに編集をコミットした開発者Aと
編集された各コード行iの元の開発者Biとの間に、重みwの有向リンク
 $A \rightarrow B_i$ が得られる

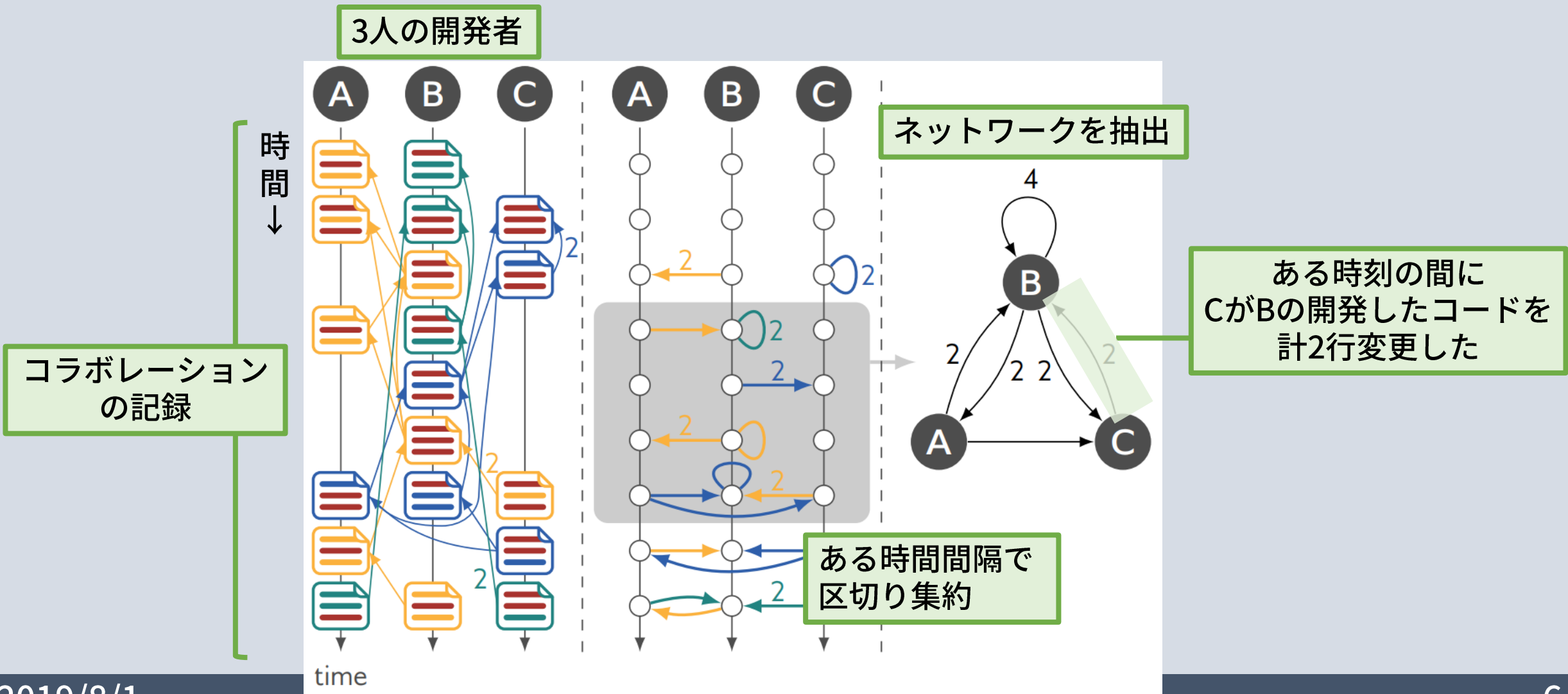
以下のコマンドで抽出 (coedits.dbへ記録)

```
./git2net.py mine <repo> coedits.db
```



4. git2netの実装 (2/4)

- データベースから開発者のコラボレーションネットワークを抽出



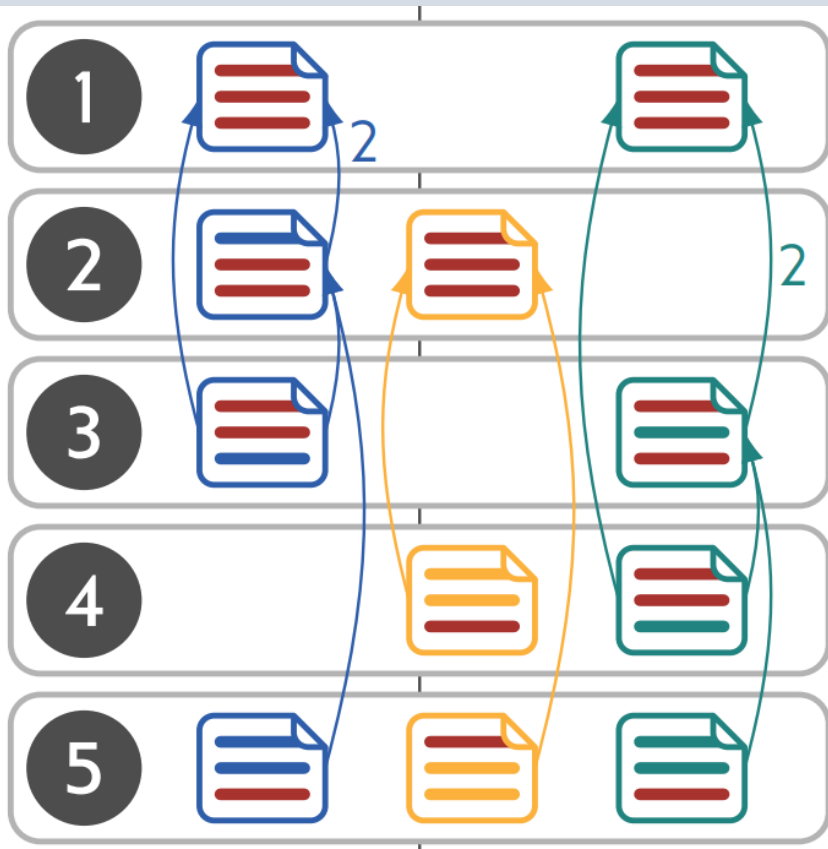
4. git2netの実装 (3 / 4)

- 一連のファイル編集を表すネットワークの抽出

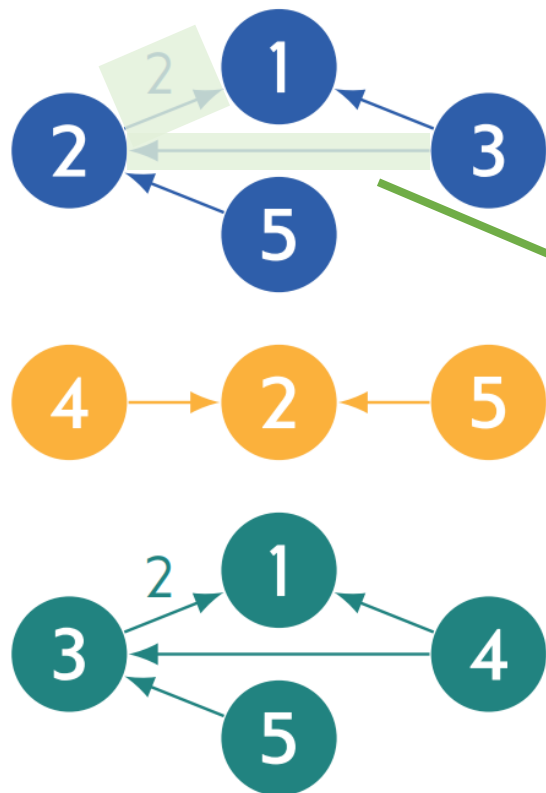
3つのファイル

時間
↓

各番号は
コミットの
記録



各ファイルの
ネットワーク



①のコミットに対して
2行の変更した
②のコミットに対して
1行の変更をした

パスは
あるファイル編集の
一連の流れ

ネットワークは
開発の知識の流れを
追跡可能

4. git2netの実装 (4/4)

- ファイルと開発者をリンクした2部ネットワークの抽出
方法は省略されていた (ケーススタディで抽出されたネットワークについて説明)

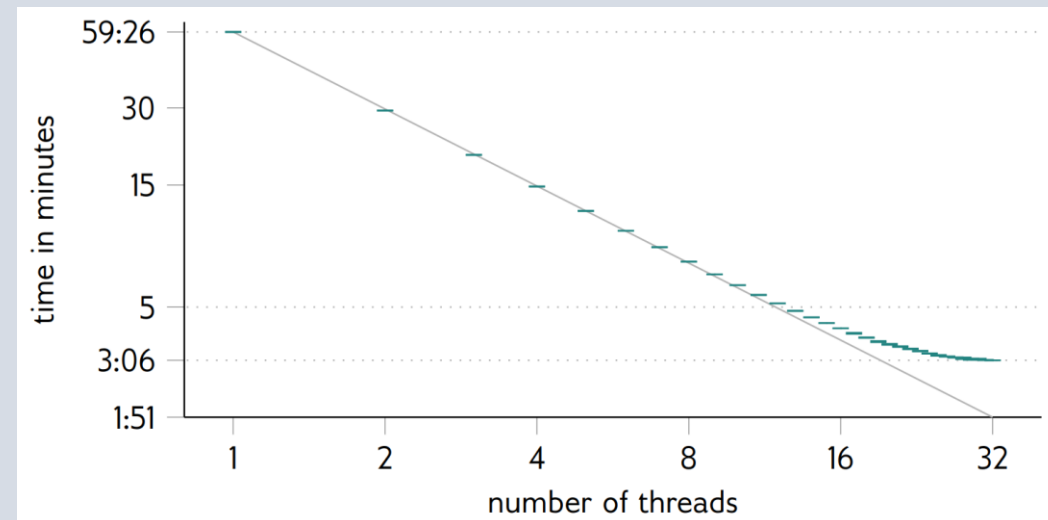
- ネットワークの抽出コマンド

```
./git2net.py graph [type] coedits.db graph.csv
```

typeは開発者のコラボレーションネットワーク、一連のファイル編集を表すネットワーク
ファイルと開発者をリンクした2部ネットワークのどれを抽出するか指定

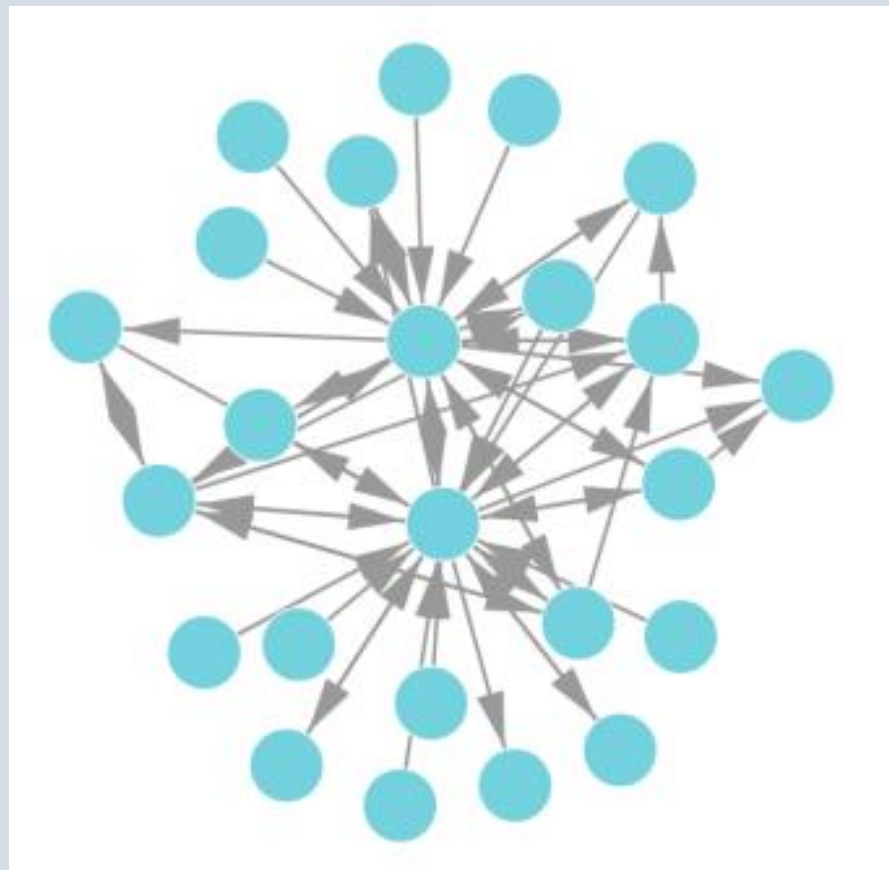
- 並列処理
スレッドの数を増やすほどネットワーク抽出の時間を短縮

Linuxのgitコミットの歴史から18年間で80万回以上の
コミットを含むコラボレーションネットワークを抽出
16個の処理コアを持つマシン上でgit2netを実行し
4日間で6,000万以上のネットワーク抽出を完了



5. git2netを使ったケーススタディ（1/4）

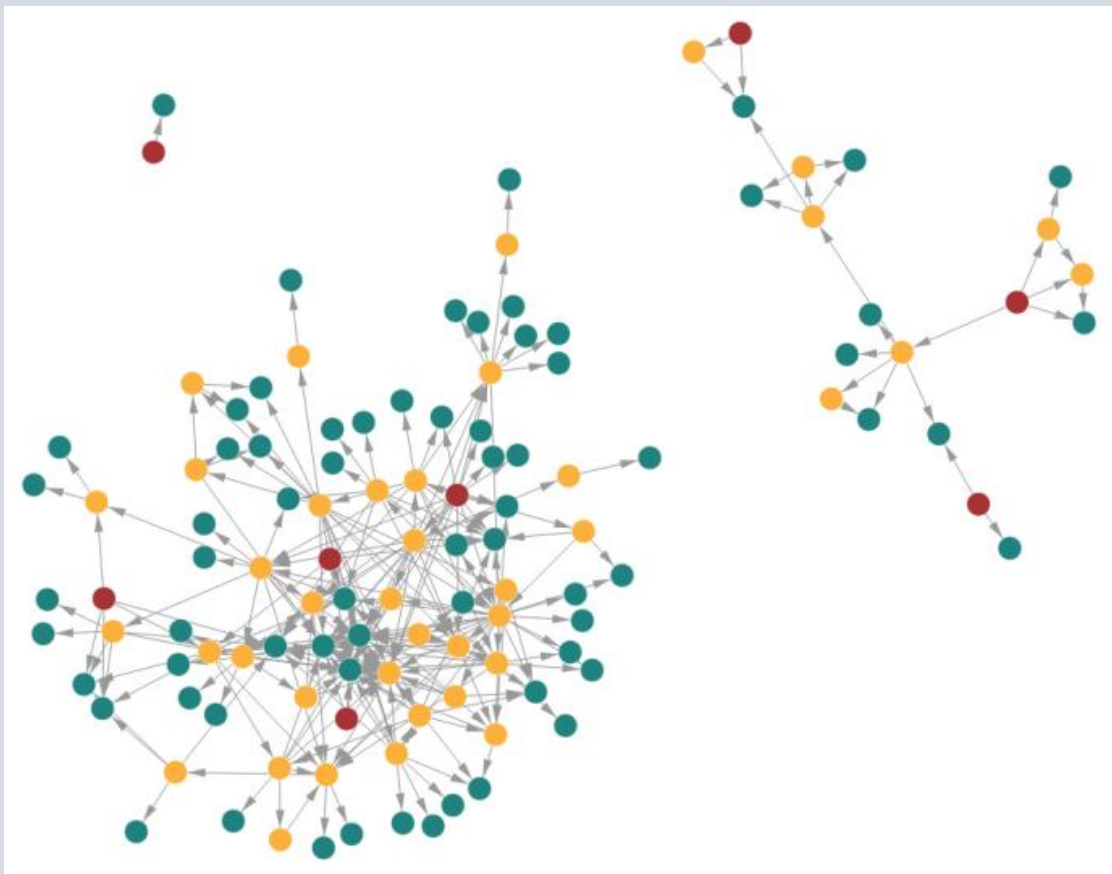
- gitリポジトリからネットワークの抽出
igraph（ネットワークの作成、可視化、分析を行うライブラリ）のgitリポジトリからgit2netを使って3種類のネットワークを抽出



- 開発者のコラボレーションネットワーク
各ノードは開発者、開発者Aによって以前書かれたコード行を開発者Bが編集していたら、A→Bのエッジを作成

5. git2netを使ったケーススタディ（1/4）

- gitリポジトリからネットワークの抽出
igraph（ネットワークの作成、可視化、分析を行うライブラリ）のgitリポジトリからgit2netを使って3種類のネットワークを抽出



- 一連のファイル編集を表すネットワークの抽出
あるソースファイル（flow.c）の編集の有向グラフ

ノードは開発者によるコミット

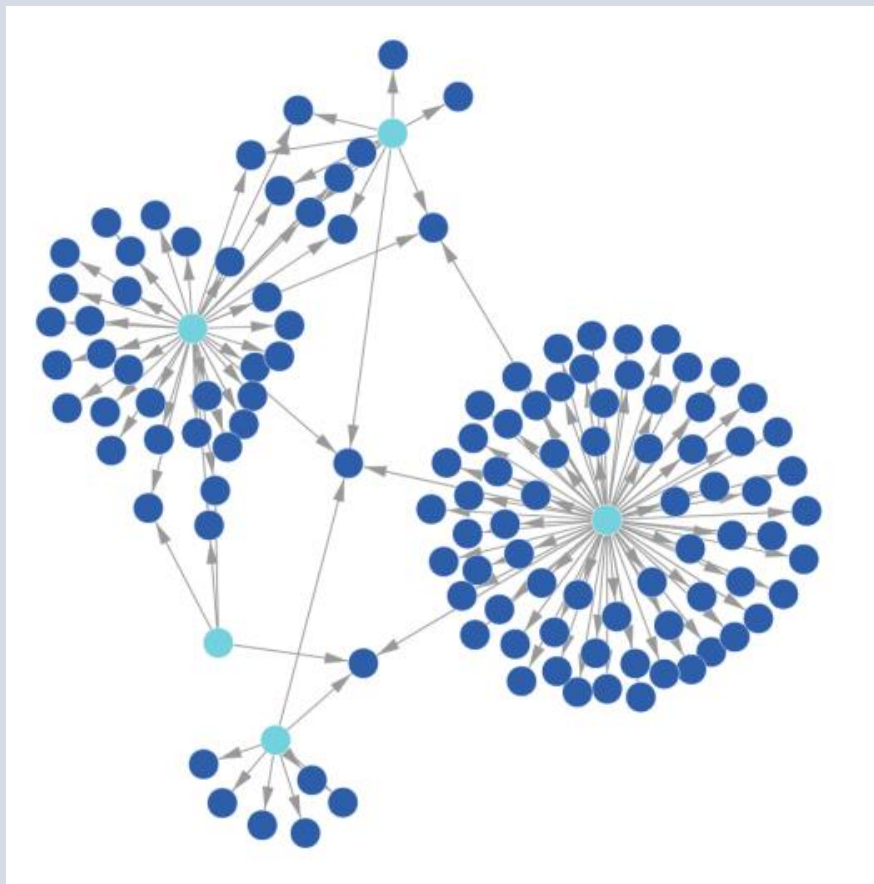
赤ノード：入力次数0

緑ノード：出力次数0

黄ノード：赤ノードと緑ノードの間

5. git2netを使ったケーススタディ（1/4）

- gitリポジトリからネットワークの抽出
igraph（ネットワークの作成、可視化、分析を行うライブラリ）のgitリポジトリからgit2netを使って3種類のネットワークを抽出



- ファイルと開発者をリンクした2部ネットワークの抽出
開発者を、開発者が編集したファイルにリンク

開発者：薄い青
ファイル：青

5. git2netを使ったケーススタディ（2/4）

- オープンソース(igraph)と商用(非公開)のコラボレーションネットワークの比較
すべての時刻のコラボレーションを集約

リンクの張り方が異なるだけなのでノード数は同じ

	network	nodes	edges	δ
Open Source	co-authorship	24	67	1.22
	co-editing	24	55	
commercial	co-authorship	84	2551	2.11
	co-editing	84	1211	

$$\delta = \frac{\text{ファイルベースのネットワークのリンク数}}{\text{ラインベースのネットワークのリンク数}}$$

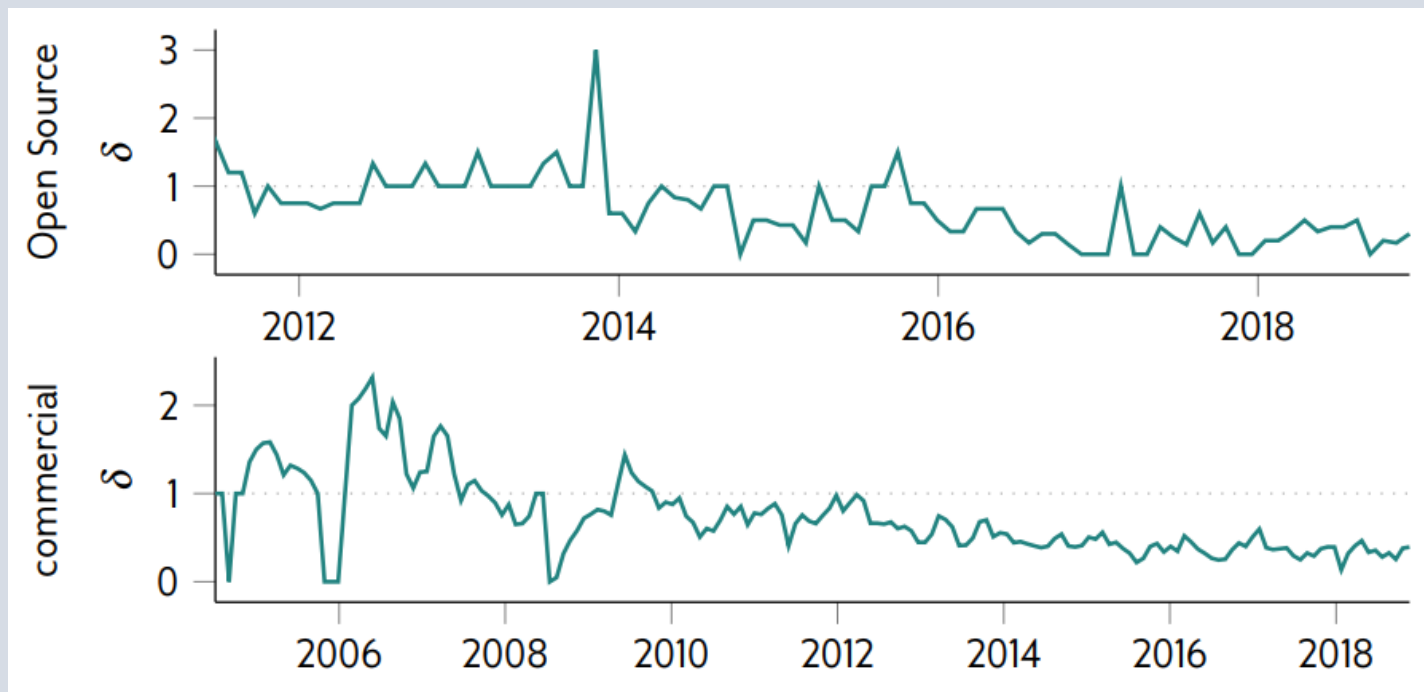
オープンソース、商用限らず $\delta > 1$ → ファイルベースのリンク数 > ラインベースのリンク数

ファイルベースでは、2人の開発者が同じファイルを編集したにもかかわらず
同じコードに対して実際にコラボレーションが行われなかったのに、リンクが張られている

追加のリンクの多くが誤検出であると予想される

5. git2netを使ったケーススタディ (2/4)

- ファイルベースとラインベースのコラボレーションネットワークの比較
オープンソース(igraph)と商用(非公開)の2つのリポジトリで比較



$$\delta = \frac{\text{ファイルベースのネットワークのリンク数}}{\text{ラインベースのネットワークのリンク数}}$$

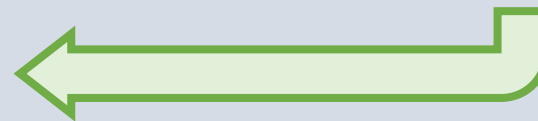
90日間ごとに区切ってネットワークを抽出
 δ の時系列データをプロット

時間が経つほど、 $\delta < 1$ の傾向

→ ファイルベースのネットワークは
現在分析されている時間より前に行われた
コミットを考慮していない

ある時刻に発生した開発者間の
コラボレーションや調整の必要性を
排除してしまう可能性

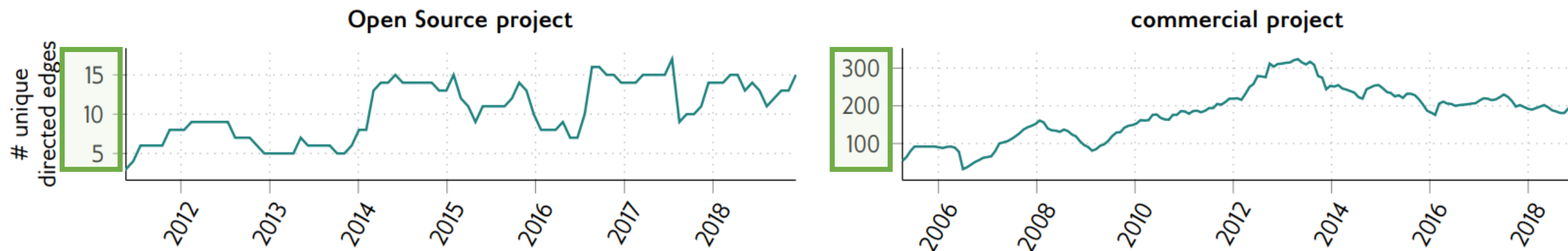
単純なファイルベースの
コラボレーションネットワーク抽出の限界



5. git2netを使ったケーススタディ（3/4）

- テンポラルネットワークの分析（1/3）

365日ごとに区切り、30日ずつスライドさせてネットワークを抽出し、4つの指標を測定
オープンソース(igraph)と商用(非公開)で比較

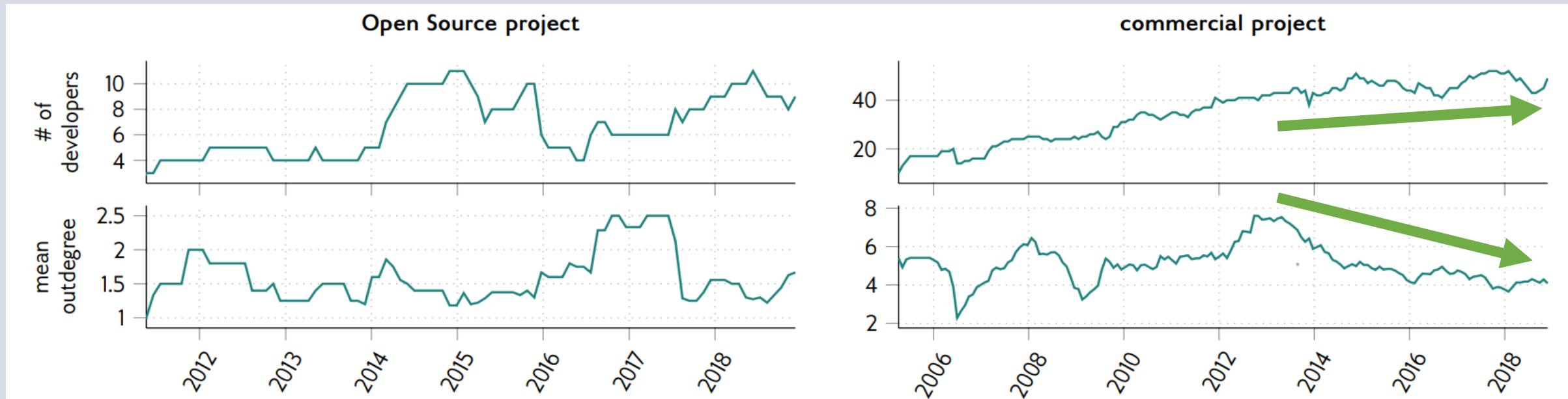


- 複数の開発者が同じコード領域を編集した際のコード領域の数 → コラボレーション量

開発者間のコラボレーションの数は、オープンソースプロジェクトよりも商用プロジェクトのほうが多い

5. git2netを使ったケーススタディ（3/4）

・テンポラルネットワークの分析（2/3）



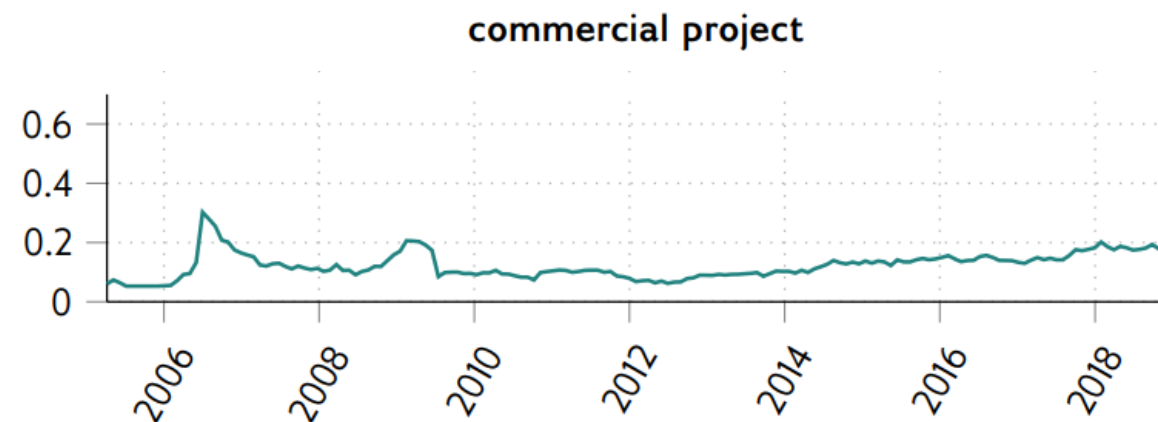
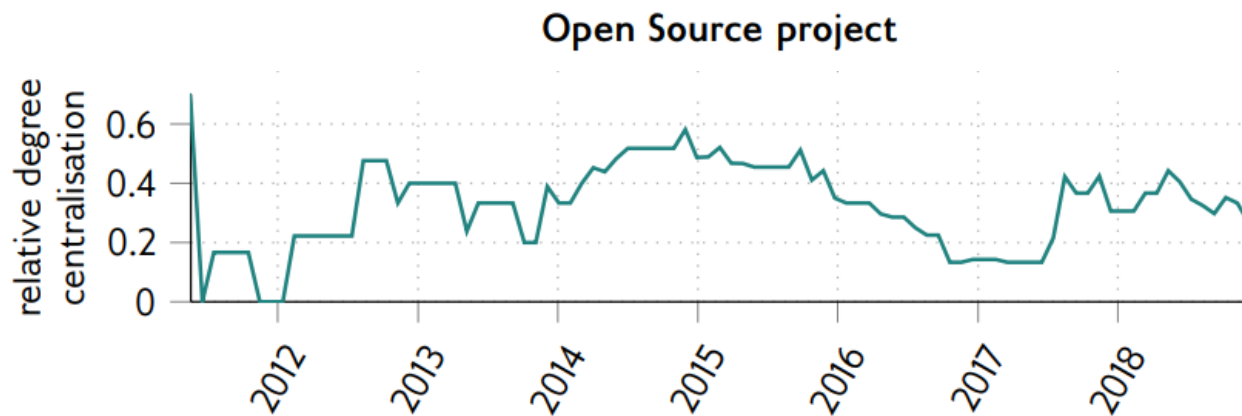
- ・プロジェクトに取り組んでいる開発者の数
- ・ノードの平均出力次数 → 関わった他の開発者の数

オープンソースの開発者は平均して1~2人の他の開発者のコードを編集するのに対し
商用プロジェクトでは4~8人

2013年以降は、開発者の数が増えているにもかかわらず、関わった他の開発者の数低下
これは、ソフトウェア開発プロセスやチームの社会的組織の変化を示す可能性がある

5. git2netを使ったケーススタディ（3/4）

・テンポラルネットワークの分析（3/3）



・正規化した次数中心性

最小値0は、ネットワーク内のすべてのノードが同じ次数を持つことを示す

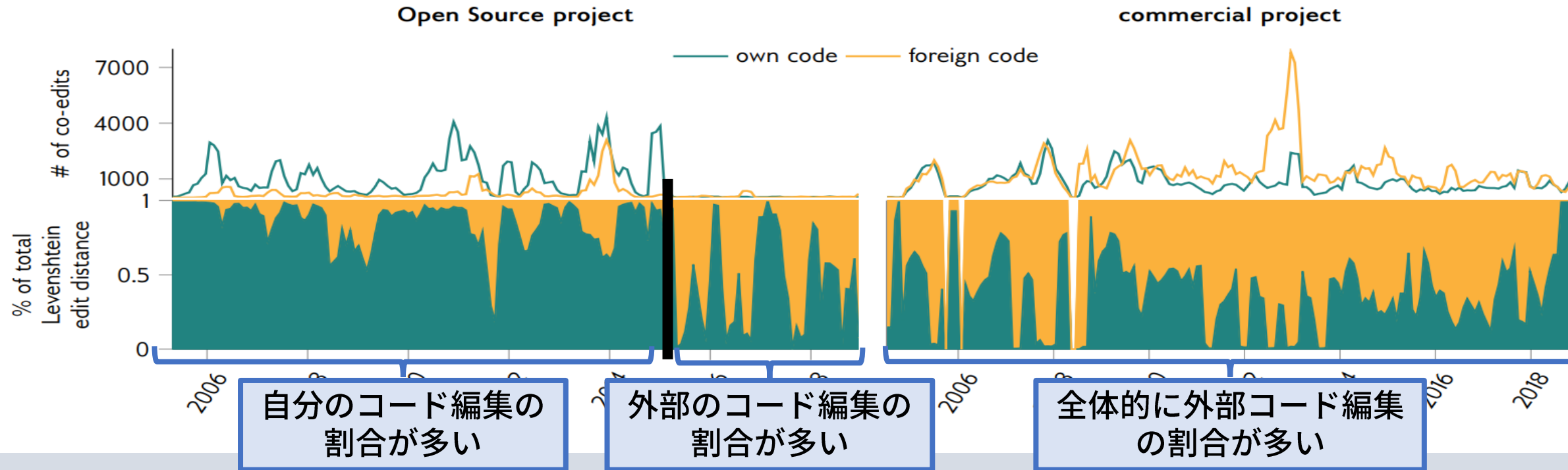
最大値1は、ハブノードを除くすべてのノードが次数1を持つ完全なスター型ネットワークになる

オープンソースプロジェクトは商用ソフトウェアプロジェクトよりかなり集中度が高い

オープンソースプロジェクトにおけるコードの貢献度の偏りの大きい

5. git2netを使ったケーススタディ（4/4）

- ・ 自分自身が編集したコードと他の開発者が編集したコードの作業割合の比較



- ・ 編集されたソースコードブロックの総数
- ・ 自分のコード編集と外部のコード編集の割合

オープンソースプロジェクトの初期段階では開発者は他の開発者のコードにあまり触れていない
商用プロジェクトでもコードレビューが行われていた可能性が高い

6. 結論

これまでの研究では、ソフトウェア開発チームの開発者同士が同じファイルを編集した場合は開発者同士がコラボレーションをしたと定義するコラボレーションネットワークを使った分析が行われてきた

しかし、この方法では

- ・ 同じファイルを編集しているが、同じ箇所は編集しておらず
実際にはコラボレーションが起きていないのに開発者間のリンクを張ってしまう
- ・ ある時間間隔ごとにネットワークを抽出して分析した際、それより前に行われたファイルへのコミットを考慮できていない

これらの問題により、コラボレーションや調整の必要性を誤認する可能性がある

本研究で提案したgit2netを使えば

開発者の編集箇所を行単位で区別してリンクを張ることで粒度の細かいネットワークを容易に抽出でき
編集前後のバージョン間のレーベンシュタイン距離を計算できる
さらに、並列処理で大規模なリポジトリも分析できる

ケーススタディでは、git2netを使ってネットワークを抽出し、分析して
開発者のコラボレーションに関連する研究に役立てることができると確認できた

2.A. 関連研究 | 参考文献リスト

- [1] M. Joblin, W. Mauerer, S. Apel, J. Siegmund, and D. Riehle,
“From developer networks to verified communities: A fine-grained approach,” 2015
- [2] I. Scholtes, P. Mavrodiev, and F. Schweitzer,
“From aristotle to ringelmann: a large-scale analysis of team productivity and coordination in open source software projects,” 2016.