

Thinking about a map for Rails applications

@makicamel

東京 Ruby 会議 12 前夜祭

2025.01.17

自己紹介

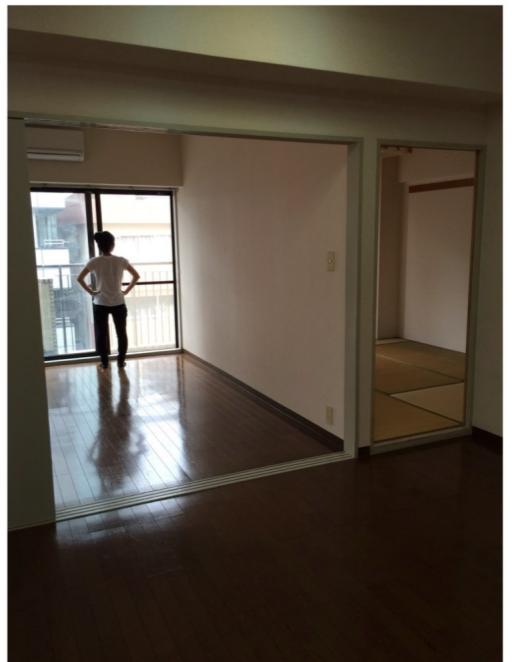
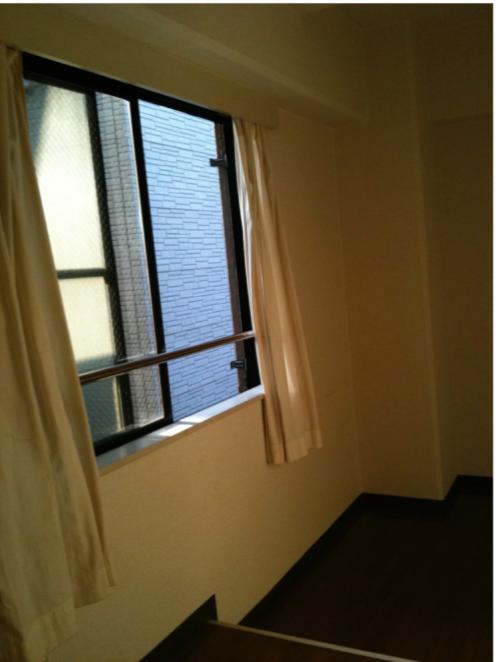
- @makicamel / 川原万季
- Ruby 💎 とビール 🍺 とお酒が好き
- (株)アンドパット 🧑
- 好きな言葉は「エイってやってバーン」 🐱



自己紹介

■ 東京歴

- 上京（2009 年前後）～2021 年居住
 - 若松河田・恵比寿・巣鴨・戸越銀座
 - リモートワーク全盛を機に東京脱出



■ 神奈川県民



Thinking about a map for Rails applications

-  Array#map
-  地図
- 今日は Rails アプリケーションの「地図」の話

Rails アプリケーションの「地図」

- ある程度大きな Rails アプリでは道しるべになるものがほしい

Rails アプリケーションの「地図」

■ 道しるべがないと

-  新しいメンバー
 - オンボーディングで教わる以外のアプリの知識の獲得は手探り
-  慣れたメンバー
 - 普段触るサービスにアプリの知識が閉じがち
 - いつの間にか知らないドメインが増えている

Rails アプリケーションの「地図」

- ドキュメントが欲しい
 - アプリの歩き方がわかるもの
 - 常に最新で正確で検索が容易なもの
- そんなドキュメントはない

Rails アプリケーションの「地図」

- ドキュメントは難しい
 - 書き忘れ
 - 書かれないがち
 - 更新忘れ
 - ドキュメントとコンテキストと現実のすり合わせが必要がち
 - 検索勘が必要
 - どのドキュメントシステムにあるかまちまち
 - 同じ項目で複数のページ作られがち

Rails アプリケーションの「地図」

- ドキュメントが欲しい
 - アプリの歩き方がわかるもの
 - 常に最新で正確で検索が容易なもの

Rails アプリケーションの「地図」

- Rails アプリの歩き方
 - とは？

Rails アプリケーションの「地図」

- Rails アプリの歩き方
 - モデル

Rails アプリケーションのモデル

- Rails アプリにおけるモデル (ActiveRecord)

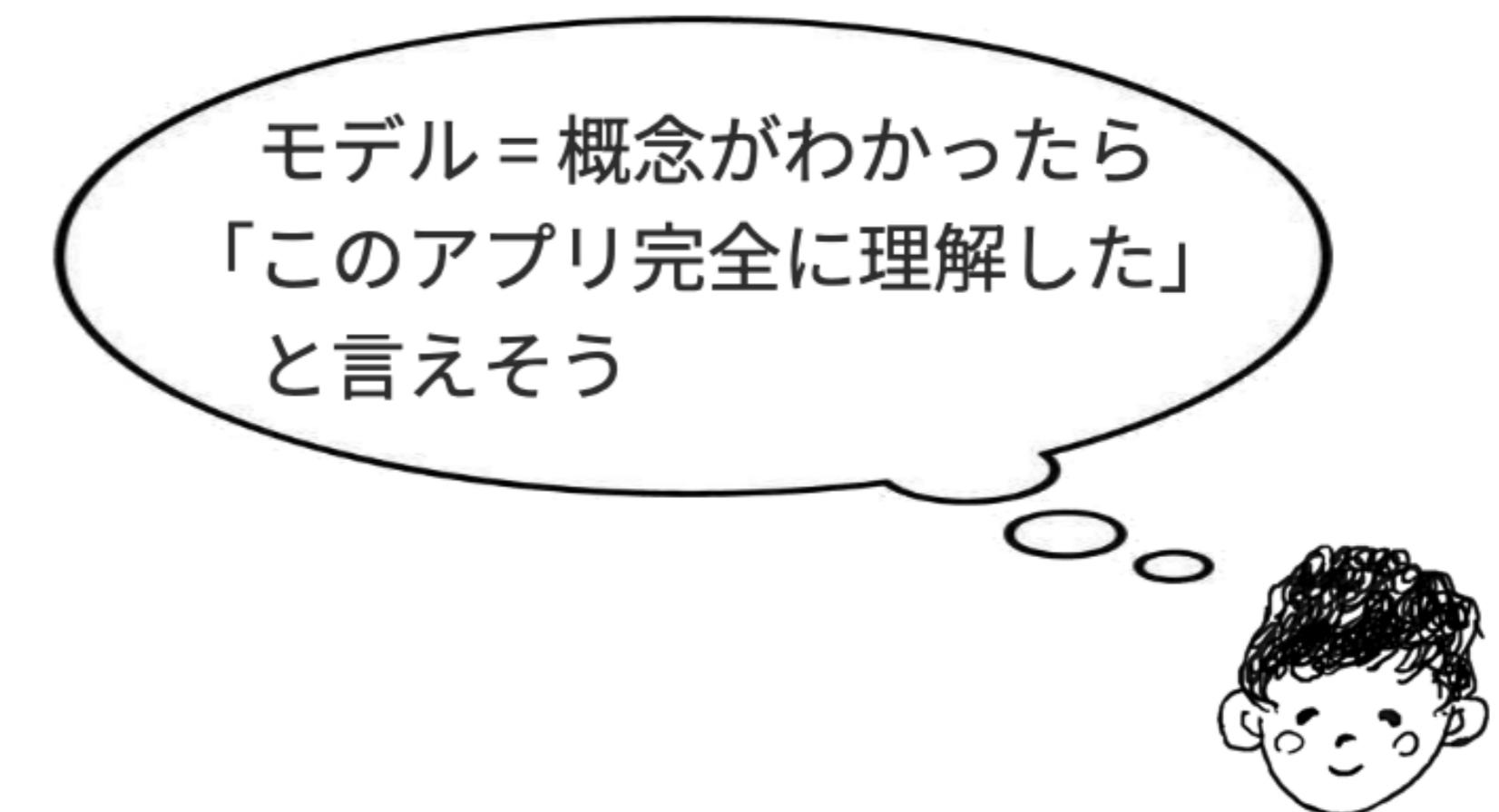
- ビジネスロジック

- 現実世界のルールやフローをコードに落とし込んだもの

- e.g. ユーザーの認証認可、在庫出庫管理

- 現実世界の概念

- e.g. ユーザー、商品、在庫 ...



Rails アプリケーションのモデル

- あるモノリシックなリポジトリのモデル数
 - 1,407 ^{[^1][^2]}

```
> rails stats
```

Name	Lines	LOC	Classes	Methods	M/C	LOC/M
Controllers	143687	112455	1656	10661	6	8
Models	225756	161799	1407	11681	8	11
Views	4136	3202	0	0	0	0
#
Total	1805333	1434727	4962	32224	6	42

[^1]: rails stats なのでモジュールや PORE も含みます

[^2]: 2025年1月14日時点

Rails アプリケーションのモデル

- 1,400 個のモデルを理解するのは無理

Rails アプリケーションのモデル

- ~~1,400 個のモデルを理解するのは無理~~
- 全部のモデルを理解したいわけではない
- アプリにとって重要なモデルを知りたい
 - 重要順に知っていくとうまく Rails アプリを歩けそう

Rails アプリケーションのモデル

■ ある日



アプリの地図ほしいですよね

画面表示時は重要なモデルだけ表示して
ズームインすると次のモデルを表示してほしい

なるほどほしい!!!!



つくれた



makicamel / erd_map

Code Issues Pull requests Actions Projects Security ...

Eye icon Watch icon Star icon

Visualize, explore, and interact with your Rails application's models and associations with ease.

MIT license

Code of conduct

0 stars 0 forks 1 watching 1 Branch 1 Tag Activity

Public repository

main ▾

Go to file + <> Code ▾

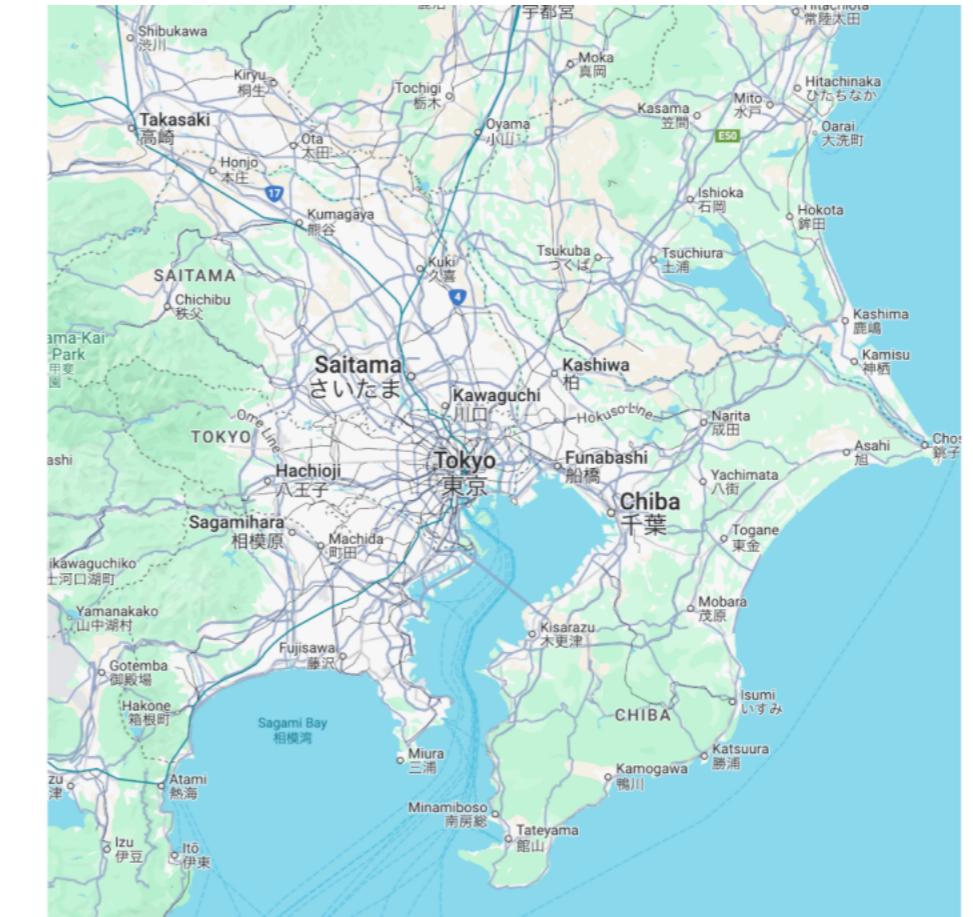
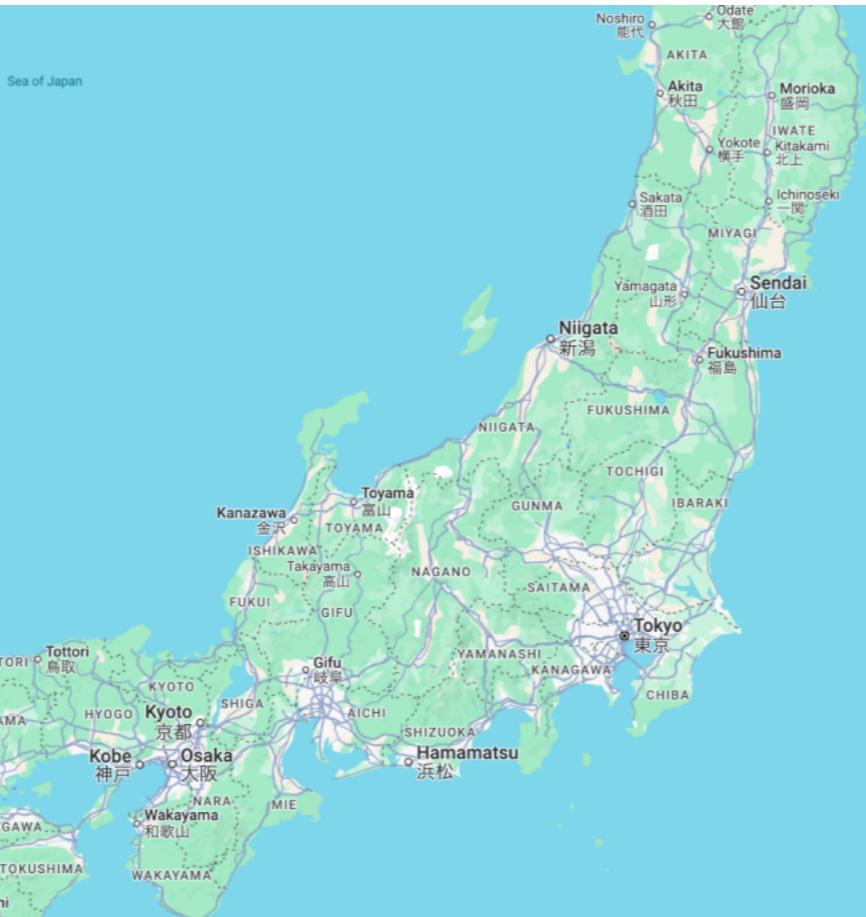
makicamel Add Changelog ✓	caa0fdf · 4 minutes ago	
.github	rails plugin new erd_map --mountable -T	last month
app	Not compute if already computed	4 hours ago

https://github.com/makicamel/erd_map

地図

<https://www.google.com/maps>

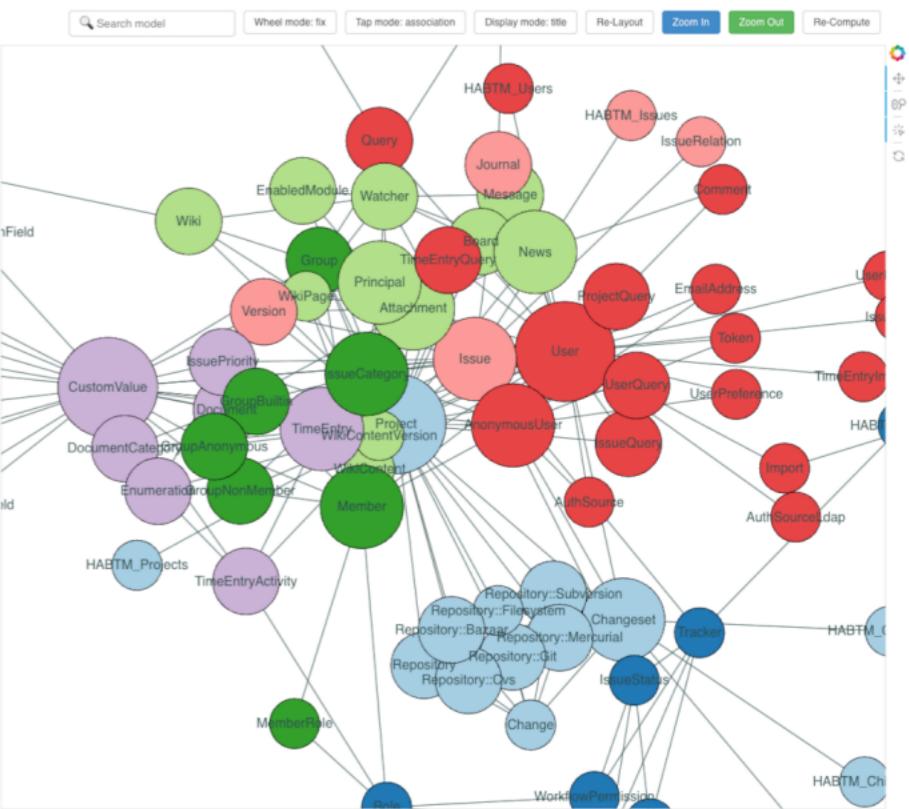
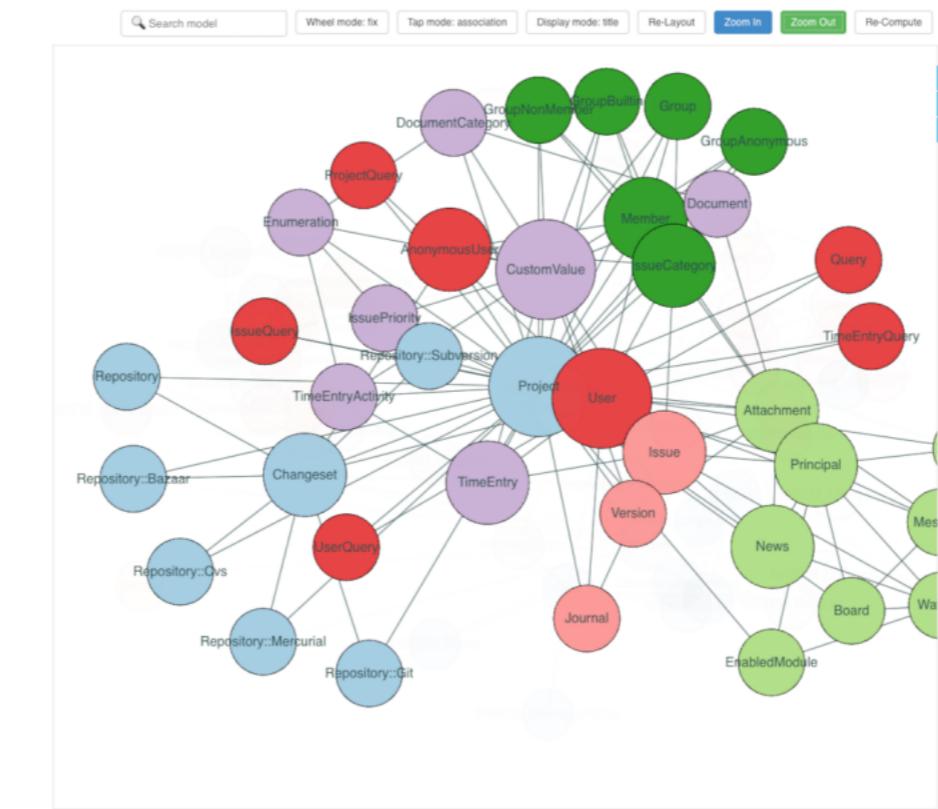
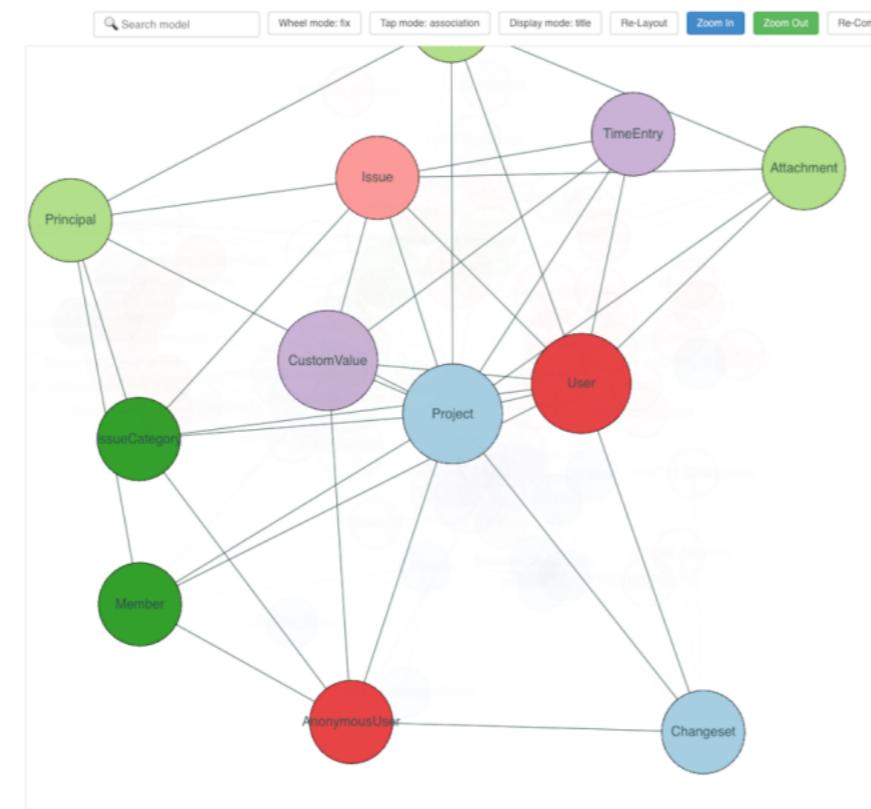
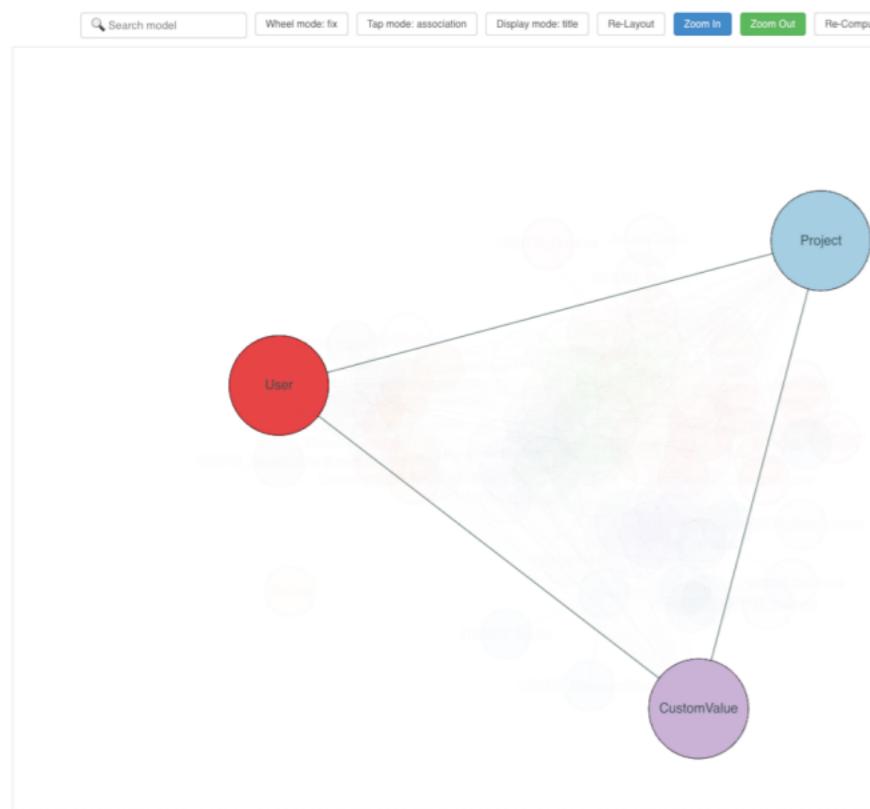
- ズームインすると詳細が見える
- ズームアウトすると全体が見通せる



ErdMap



- ズームインすると詳細が見える
 - ズームアウトすると概要が見える





- 最初は3つの"重要"なモデルのみ表示
- ズームインすると次に重要なモデルを表示
 - 重要度の高いモデルは大きく、重要度の低いモデルは小さく表示
- モデルを"コミュニティ"ごとに色分け



- ErdMap づくりに必要なこと
 - モデルの"重要"度の評価
 - "コミュニティ"分割

ErdMap のつくり方



ネットワーク分析

ネットワーク分析

■ ネットワークとは

- ノード（点）とエッジ（線）で構成されるデータ構造
- e.g. SNS のユーザー関係
 - ノード: ユーザー、エッジ: フォロー関係

ネットワーク分析

- ネットワーク分析とは
 - ネットワーク構造およびそのノードやエッジの関係性の分析
 - e.g. ノードの重要度の評価
 - e.g. コミュニティ検出

ノードの重要度の評価

- e.g. 電力ネットワークの重要地点特定
 - ノード: 変電所、エッジ: 送電線
 - 都市全体への電力供給の中継地点である変電所の重要度を高く評価
 - 活用例: システム障害時の影響を最小限に抑える設計やリスク評価を行う

コミュニティ検出

- e.g. 「この商品を買った人はこんな商品も買っています」
 - ノード: 商品、エッジ: 同じユーザーに購入された商品同士を繋ぐ
 - EC サイトでよく見るアレ
 - 購入頻度、同時購入の確率で重みづけ可
 - 活用例: 密につながった商品コミュニティを特定しレコメンドやパーソナライズを行う

やってみよう

ErdMap に必要なこと



- モデルの"重要"度の評価 
- "コミュニティ"分割

重要度の評価指標

- 次数中心性
- 固有ベクトル中心性
- 媒介中心性
- etc

固有ベクトル中心性を
使います



重要度の評価指標: 次数中心性

- 直接関連している他モデルの数（次数）を評価
- シンプルで高速

重要度の評価指標: 固有ベクトル中心性

- どれだけ重要なモデルと関連しているか^[^3]を評価
- 次数中心性の「数」の評価に対し「質」を評価
 - e.g. 請求データ
 - ユーザーや注文といった中核モデルと紐づくため固有ベクトル中心性が高い
 - 紐づくモデルが少ないため次数中心性が低い
 - e.g. 履歴データ
 - 中核モデルと紐づいても関連の終端になりやすく固有ベクトル中心性が低い
 - 多数の関連を持つため次数中心性が高い

[^3]: 関連する重要モデルの数ではなく相手モデルのスコアの合計

重要度の評価指標: 媒介中心性

- 他のモデル間の最短経路にどれだけ頻繁に登場するかを評価
 - e.g. 請求データ
 - ユーザーや注文といった中核モデルと紐づくため固有ベクトル中心性が高い
 - 他のノード間の最短経路にあまり登場しないため媒介中心性が低い

固有ベクトル中心性のアルゴリズム

1. 初期設定

- すべてのノードに初期スコアを与える

2. スコアの更新

- 各ノードのスコアの「接続先ノードのスコアの合計」を算出し正規化する

3. 繰り返し

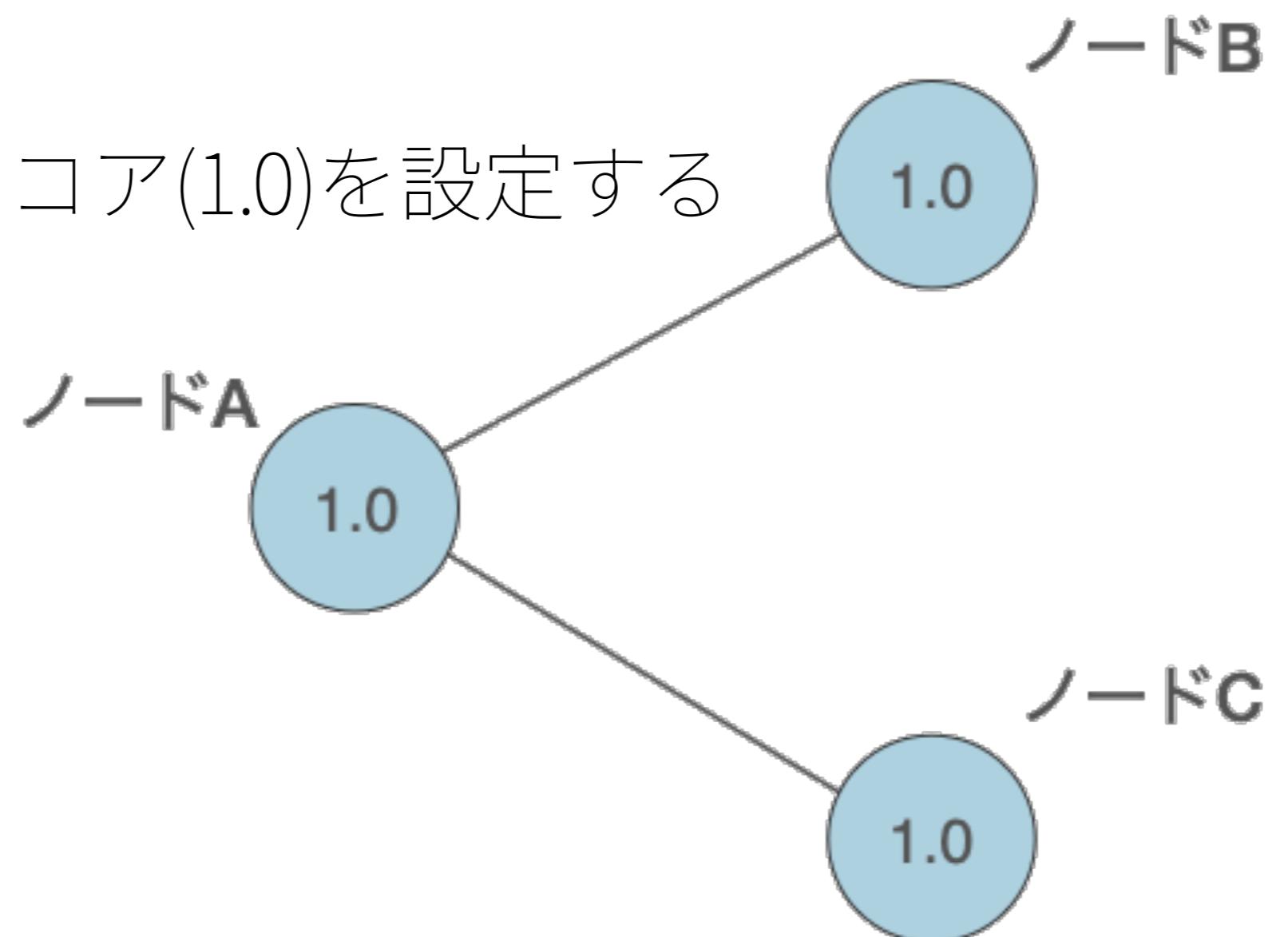
- スコアが収束するまで2の計算を繰り返す

固有ベクトル中心性のアルゴリズム

- 例えば「A-B」「A-C」が接続するネットワークで考えてみる

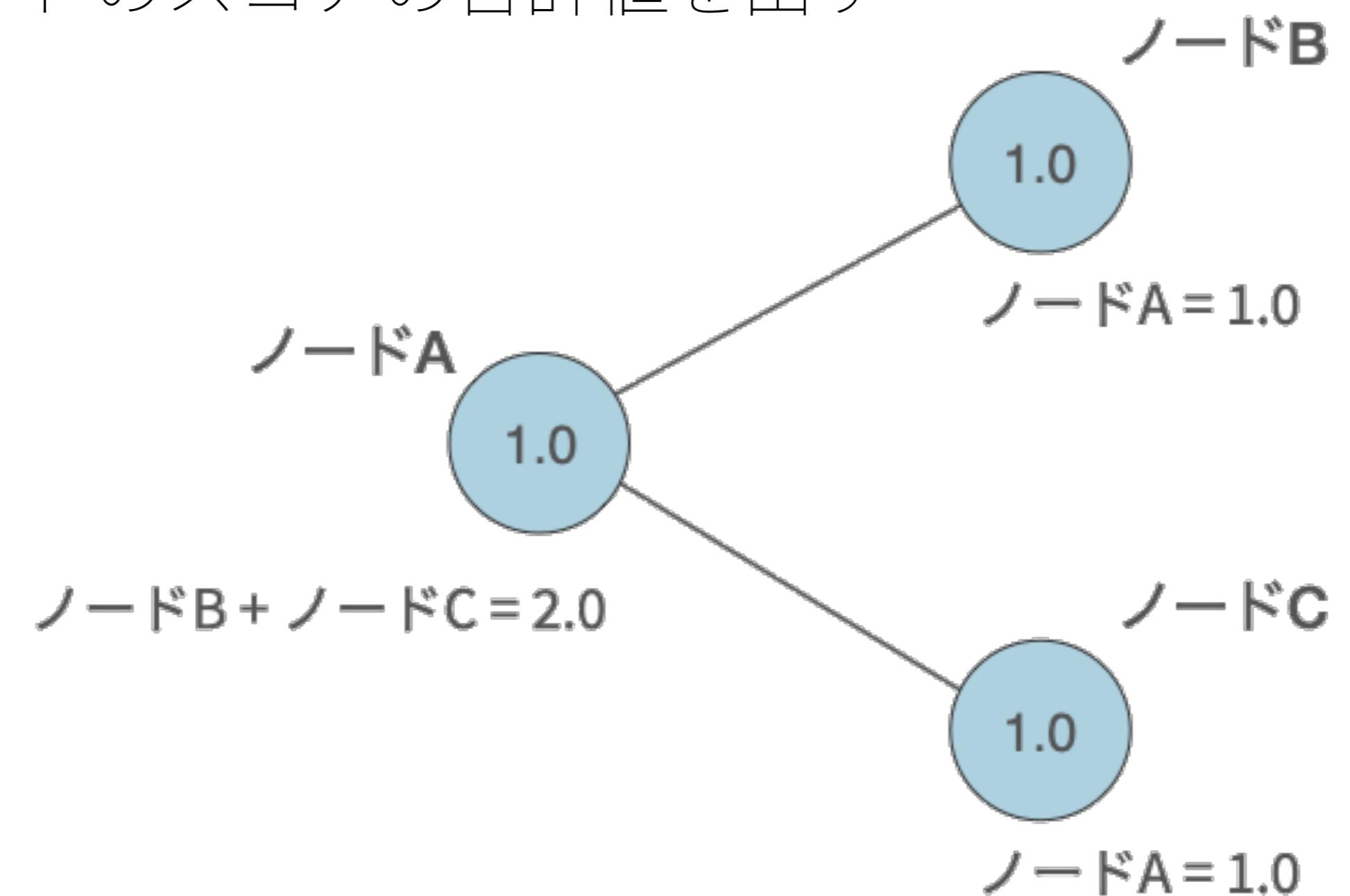
- 初期設定

- 各ノードに初期スコア(1.0)を設定する



固有ベクトル中心性のアルゴリズム

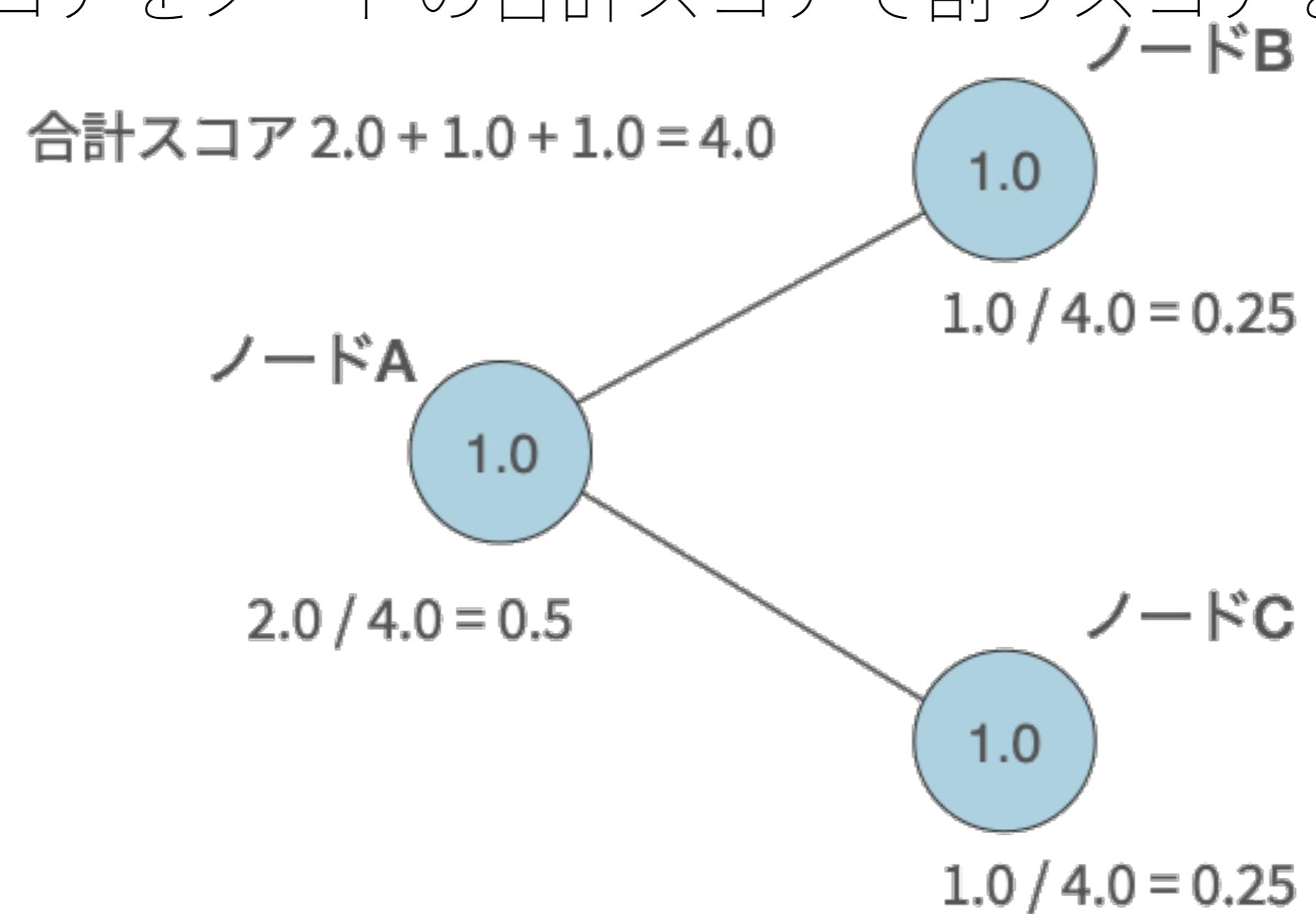
- スコアの算出(1回目)
 - 接続先ノードのスコアの合計値を出す



固有ベクトル中心性のアルゴリズム

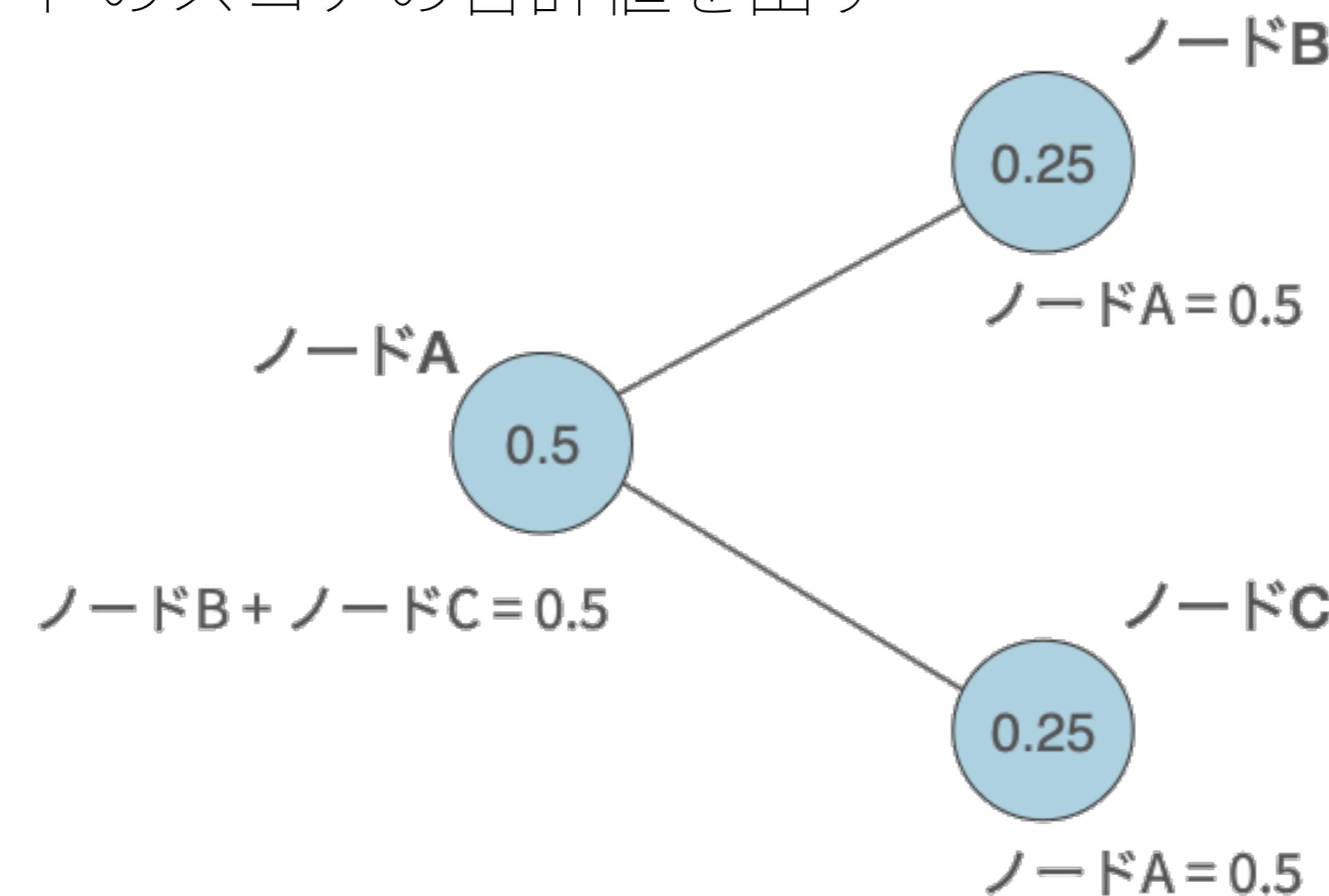
- スコアの正規化(1回目)

- ノードのスコアをノードの合計スコアで割りスコアを更新する



固有ベクトル中心性のアルゴリズム

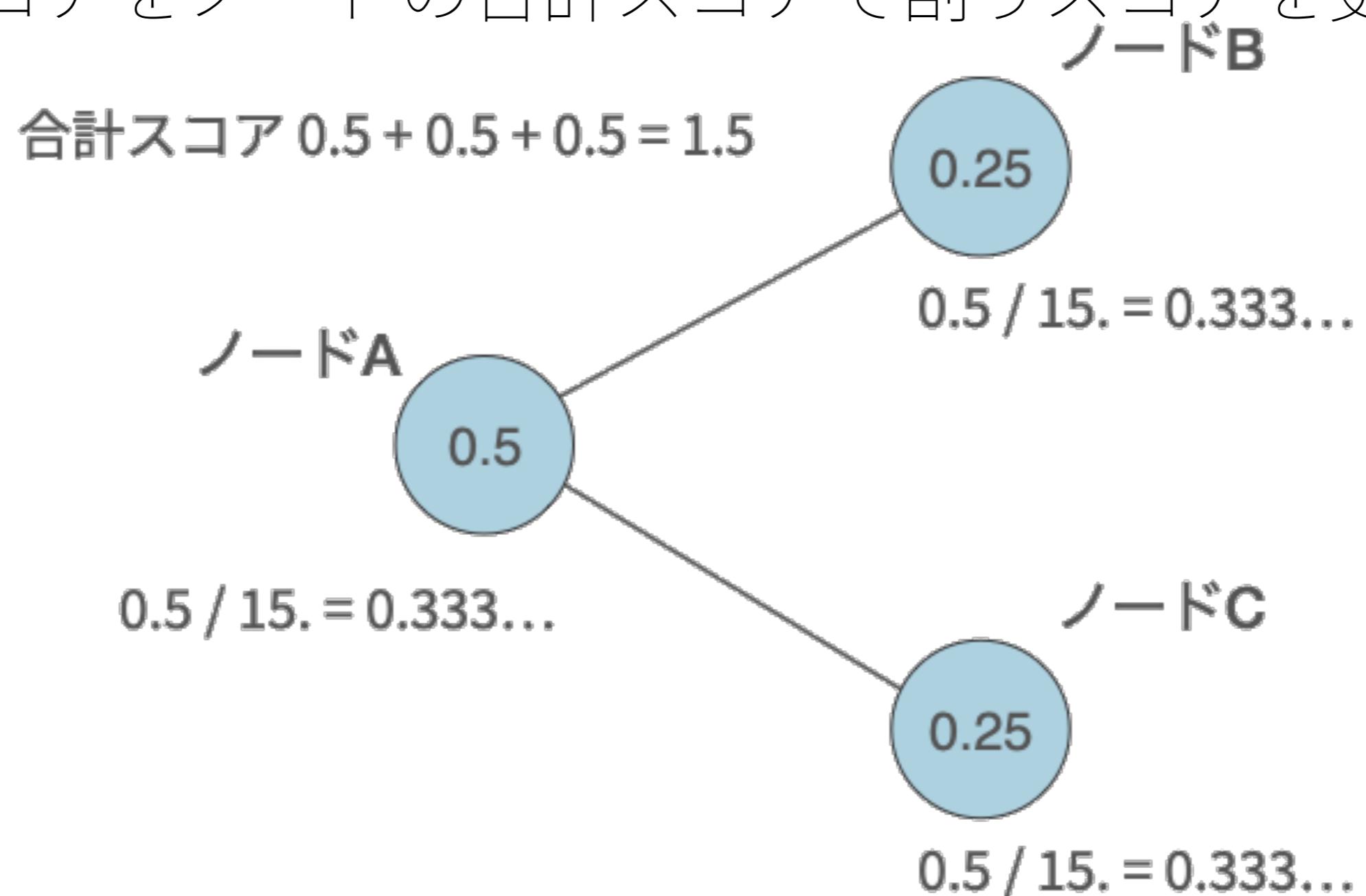
- スコアの算出(2回目)
 - 接続先ノードのスコアの合計値を出す



固有ベクトル中心性のアルゴリズム

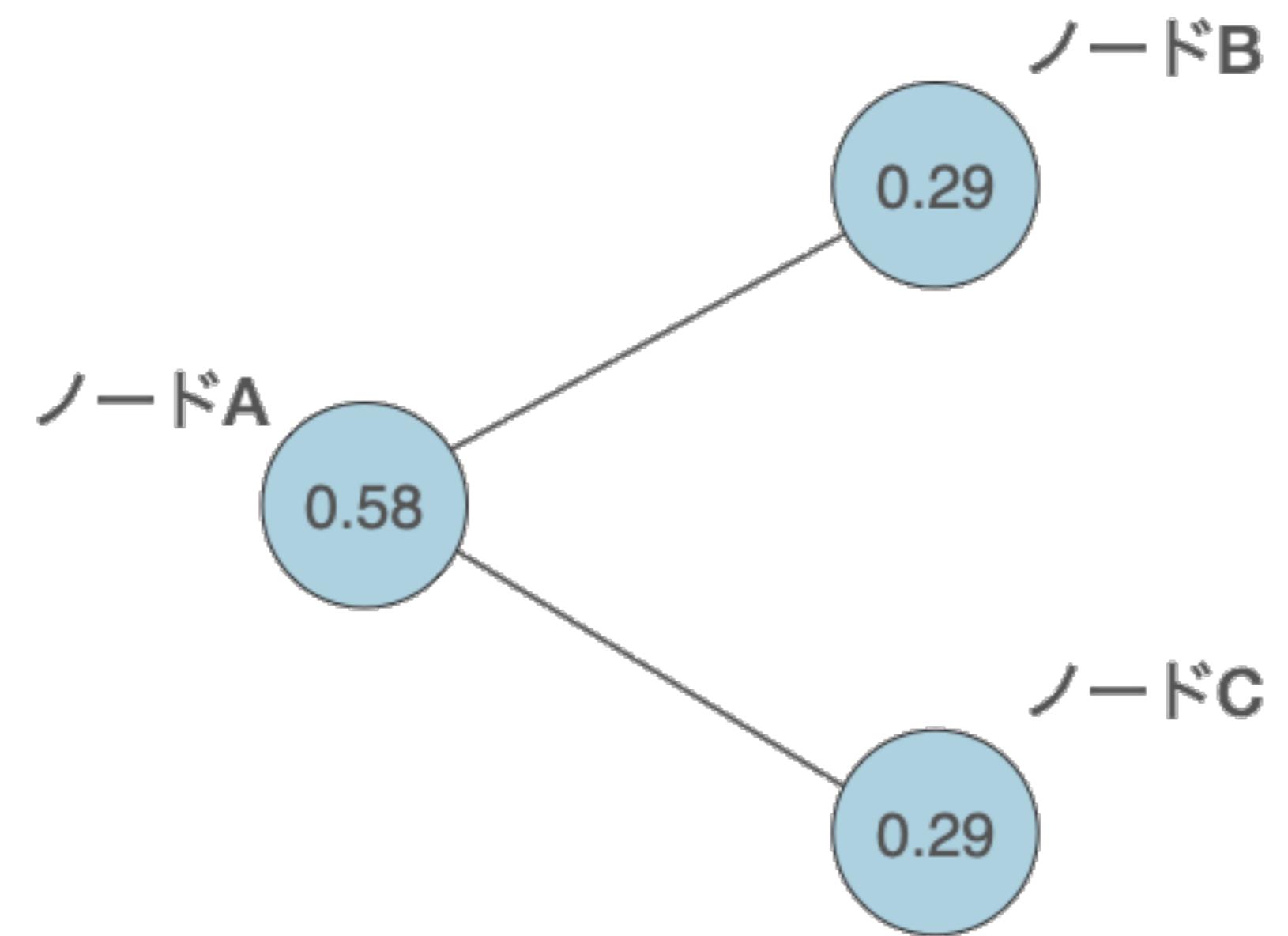
- スコアの正規化(2回目)

- ノードのスコアをノードの合計スコアで割りスコアを更新する



固有ベクトル中心性のアルゴリズム

- スコアの更新を繰り返すとスコアがほとんど変化しなくなり、最終的に収束する



ErdMap に必要なこと



- モデルの"重要"度の評価
- "コミュニティ"分割 

コミュニティ分割

- とは？

コミュニティ分割

- 重要なモデルとそれに関連するモデルを知りたい
 - ひとつのモデルはひとつの概念を表す
 - ビジネスロジックは複数のモデルから成ることが多い

アプリケーションの
構造が見えそう

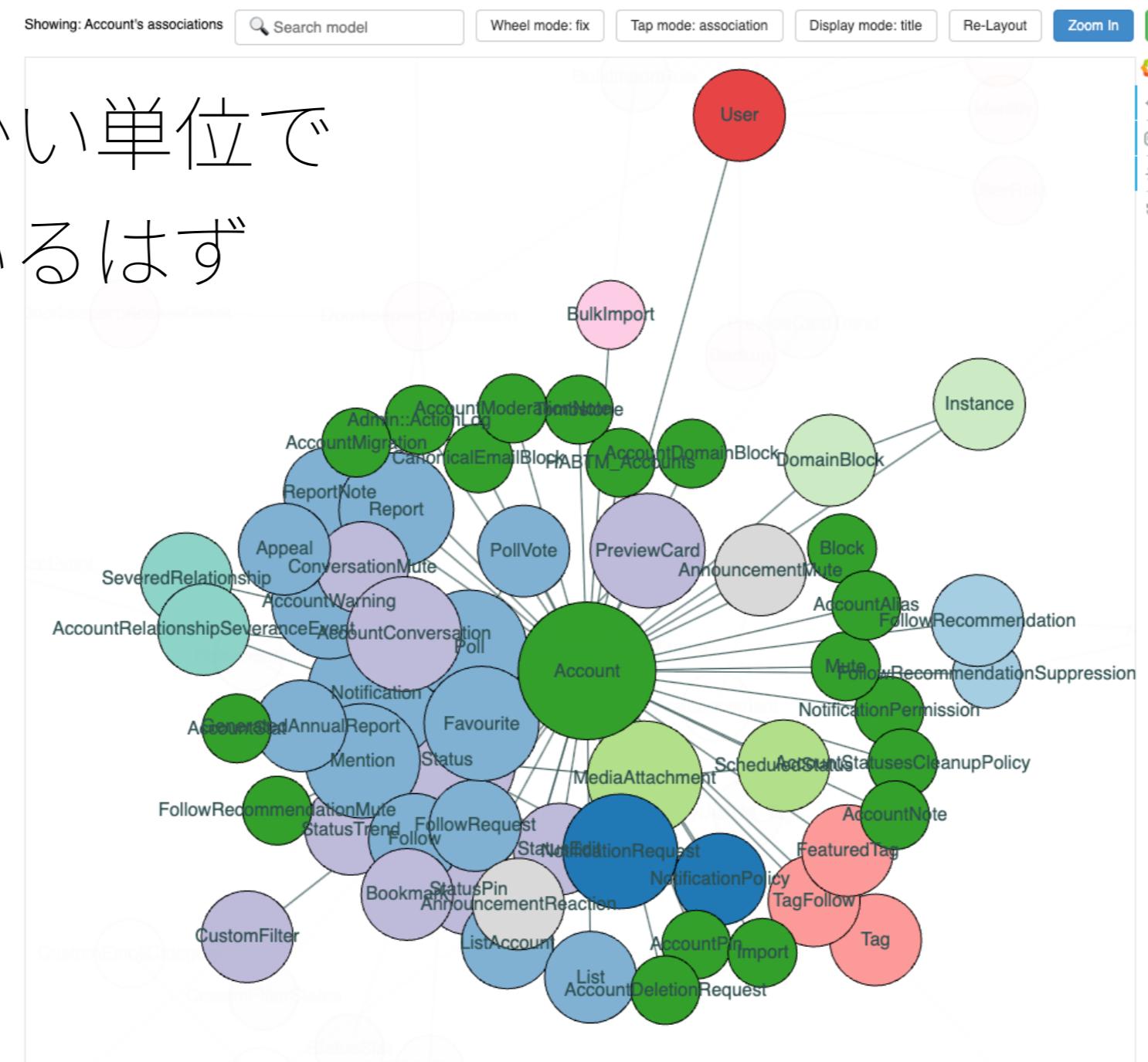


ユニティ分割

■ Mastodon の Account モデルの隣接モデル

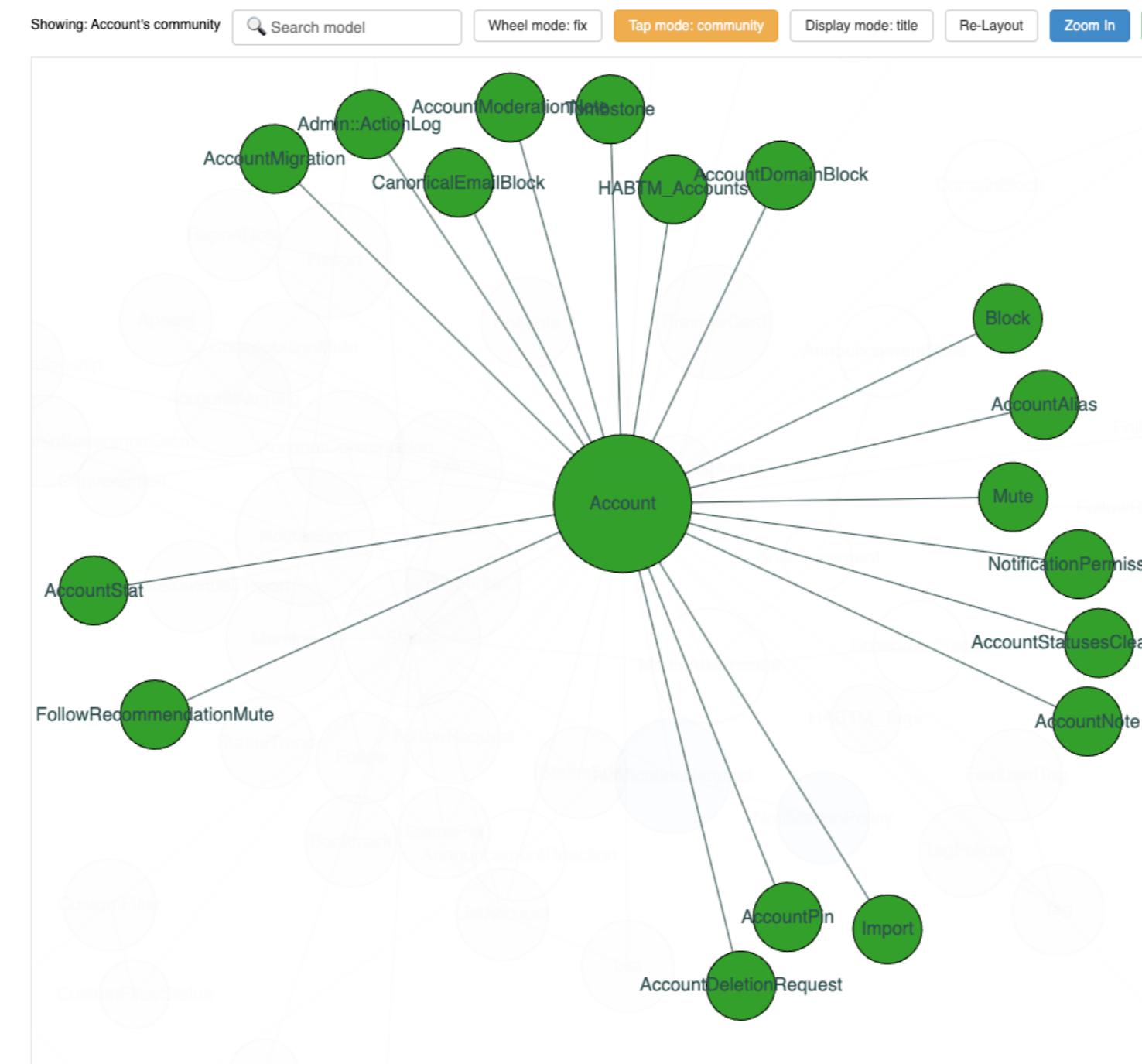
- 情報が多くて情報を読み取りづらい

- 我々はもっと細かい単位で概念を認識しているはず



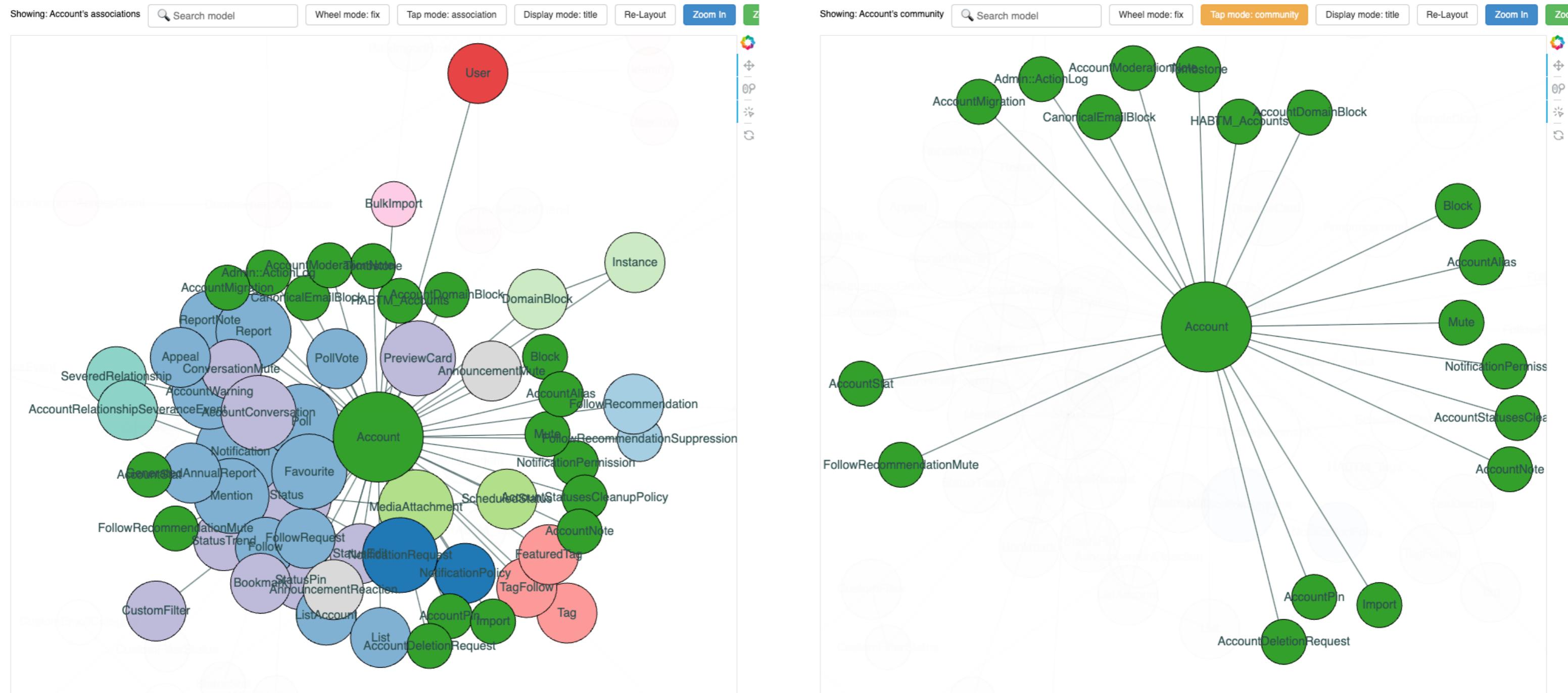
コミュニティ分割

- Mastdon の Account モデルの「コミュニティ」モデル
 - Accountにまつわる概念が見えていそう



コミュニティ分割

■ モデルのグループをいい感じに分割する



コミュニティ検出

■ アルゴリズム

- Louvain 法
- Girvan-Newman 法 ^[^4]
- etc

[^4]: Louvain法は高速、Girvan-Newman法は計算コストが高いですが、Railsアプリのモデルデータくらいなら関係なさそうです

両方試して
分割結果がしつくりきた
Louvain 法を採用

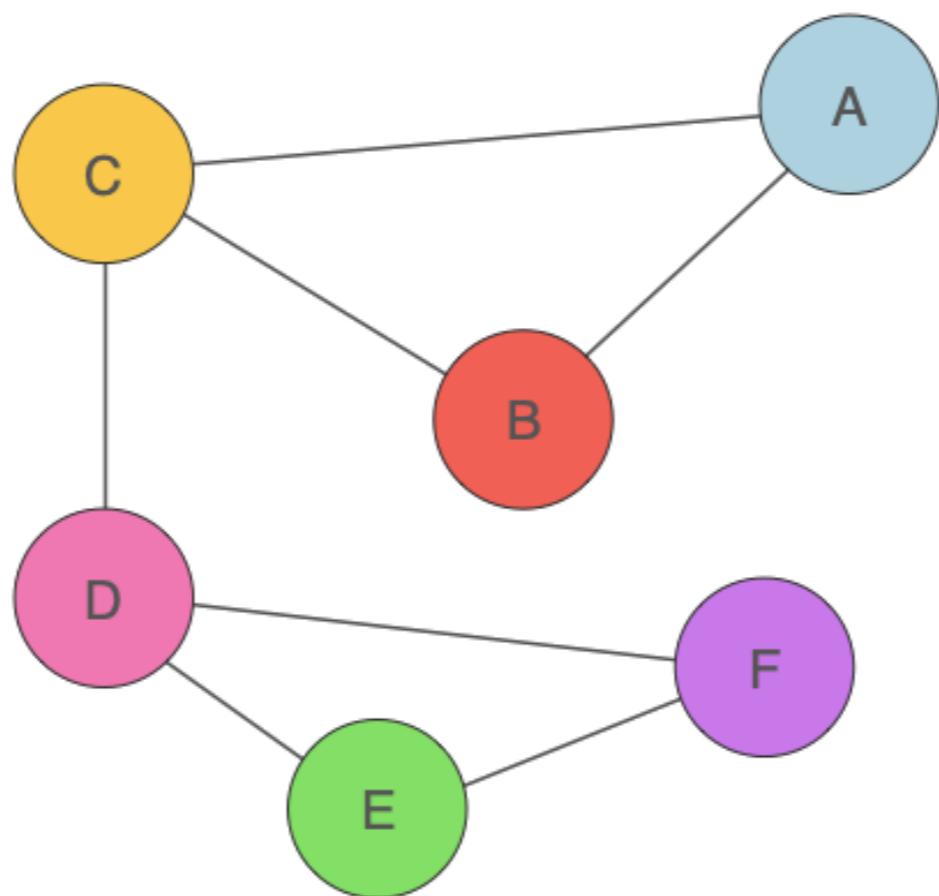


コミュニティ検出:Louvain 法

- ネットワーク内でつながりの強いコミュニティを発見するアルゴリズム
- モジュラリティを最大化するようノードを移動・統合し、最適化を繰り返すことで自然なまとまりに分割する
 - モジュラリティ = つながりの密度

コミュニティ検出:Louvain 法

- 6つのノード、7つのエッジを持つネットワークがあるとする
- 初期状態ではすべてのノードは独立したコミュニティ



コミュニティ検出:Louvain 法

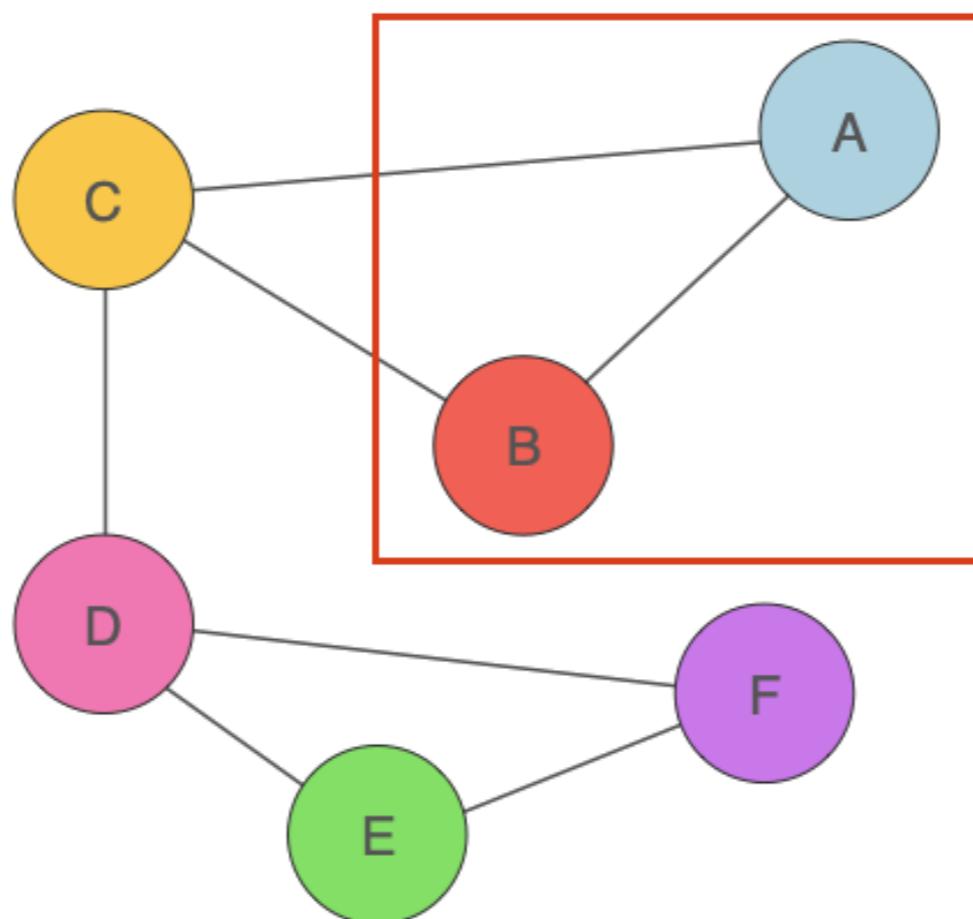
- AとBを同じコミュニティにした場合のモジュラリティの変化を見る

AとBをひとつのコミュニティにする

$$\text{コミュニティ内のエッジ数} = 1$$

$$\frac{\text{Aのエッジ数} * \text{Bのエッジ数}}{\text{エッジの総数} * 2} = \frac{2 * 2}{14} = 0.286$$

$$\text{モジュラリティ変化} = \frac{1}{\text{エッジの総数} * 2} * (1 - 0.286) \approx 0.051$$



コミュニティ検出:Louvain 法

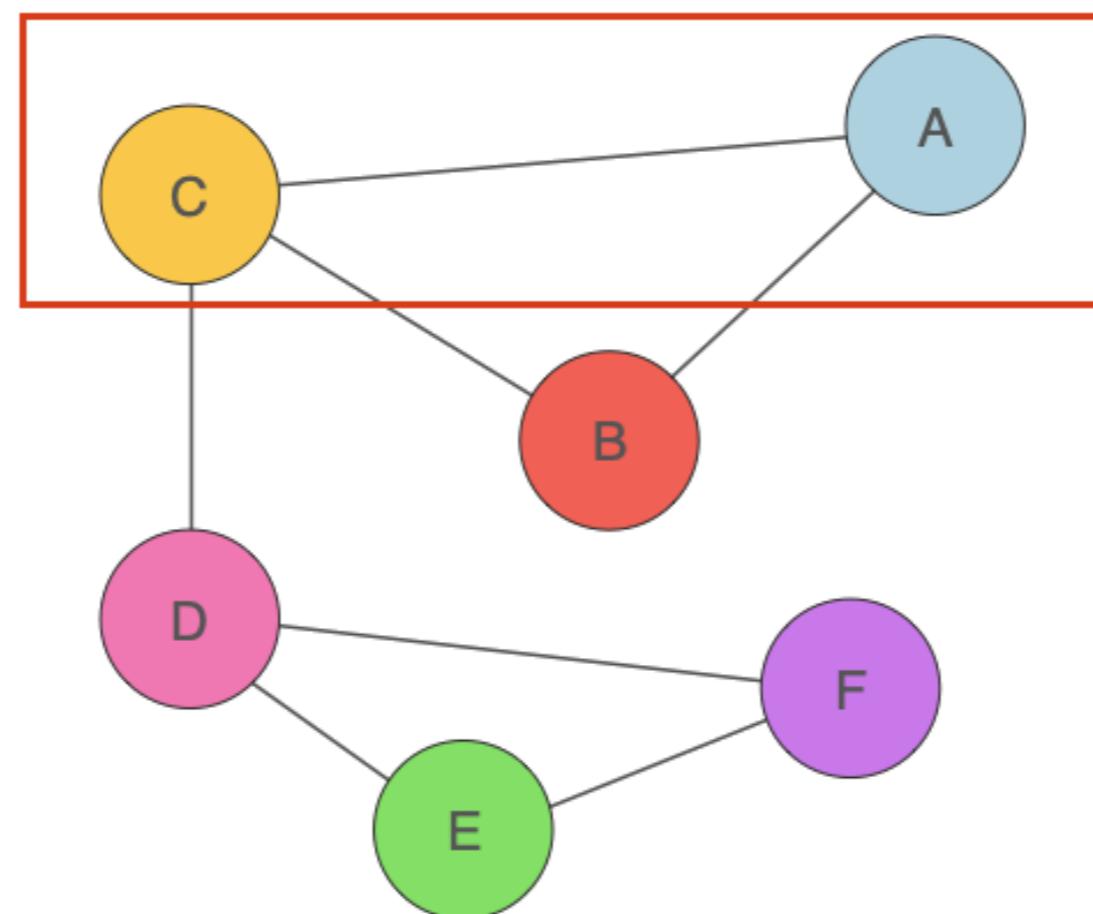
- AとCを同じコミュニティにした場合のモジュラリティの変化を見る

AとCをひとつのコミュニティにする

$$\text{コミュニティ内のエッジ数} = 1$$

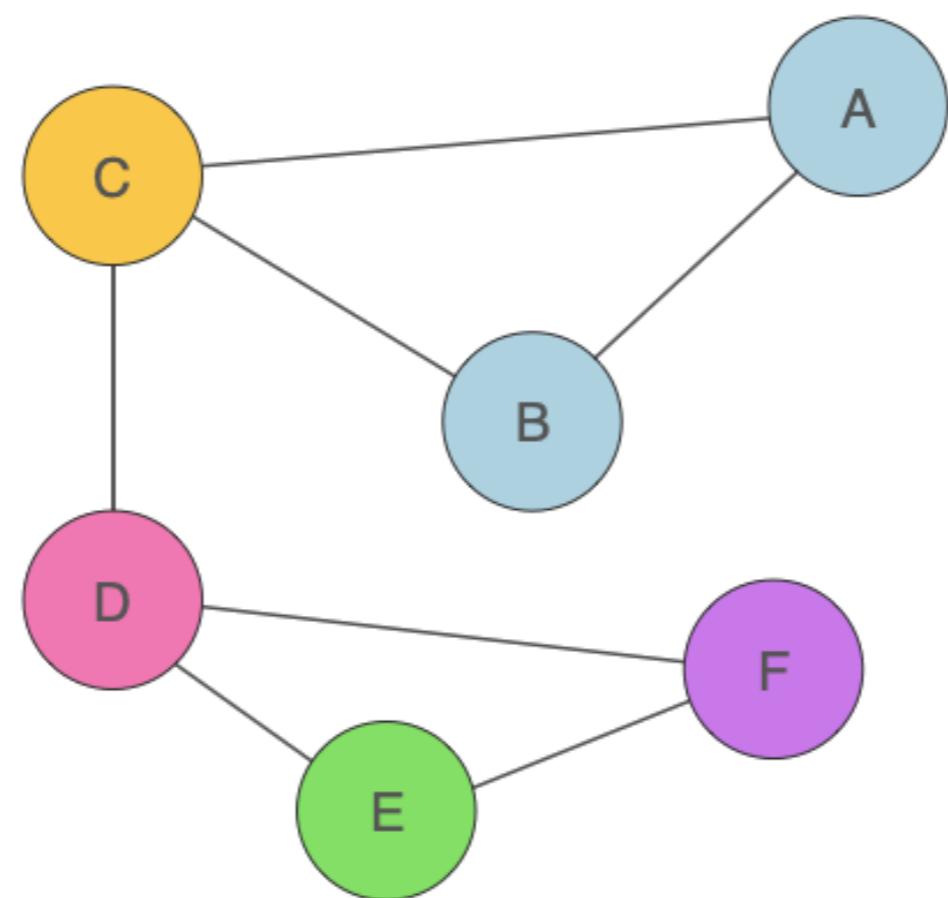
$$\frac{\text{Aのエッジ数} * \text{Cのエッジ数}}{\text{エッジの総数} * 2} = \frac{2 * 3}{14} = 0.429$$

$$\text{モジュラリティ変化} = \frac{1}{\text{エッジの総数} * 2} * (1 - 0.429) \approx 0.041$$



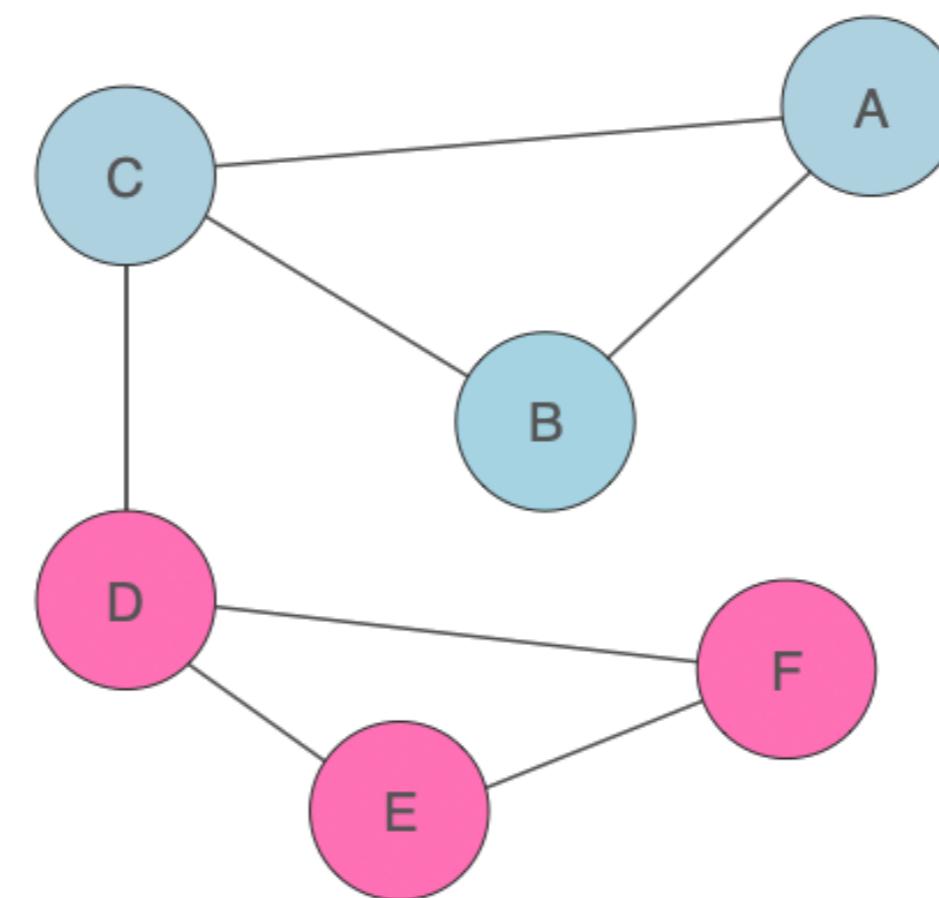
コミュニティ検出:Louvain 法

- AとBを同じコミュニティにしたほうがモジュラリティが高くなる
- AとBを同じコミュニティとして確定



コミュニティ検出:Louvain 法

- この操作を繰り返すと以下のコミュニティに分割される



実装

- 自分で実装しなくていい
- NetworkX が提供するメソッドをコールするだけ

NetworkX

- ネットワーク分析用パッケージ
- Python 製
- Rails から呼びたい



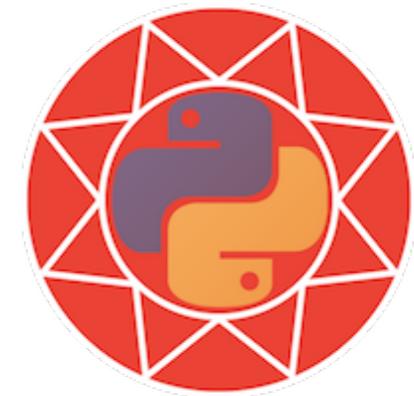
NetworkX <https://github.com/networkx/networkx>

PyCall.rb

- Ruby - Python ブリッジライブラリ
 - Python オブジェクトを Ruby から触れる
 - Ruby オブジェクトを Python 側に見せられる
- mrknさん作

PyCall <https://github.com/mrkn/pycall.rb>

PyCallがあれば Ruby で機械学習ができる <https://magazine.rubyist.net/articles/0055/0055-pycall.html>



固有ベクトル中心性

```
networkx = PyCall.import_module("networkx")
graph = networkx.Graph.new
# ... graphにノードやエッジを追加 ...
centralities = networkx.eigenvector_centrality(graph)
#=> {ノード名: 中心性} のハッシュが返る
```

- これだけのコードで
固有ベクトル中心性が得られる



Louvain 法

```
whole_communities = networkx_community
    .louvain_communities(whole_graph)
    .map { |communities| PyCall::List.new(communities).to_a }
```

すごい!!



Louvain 法

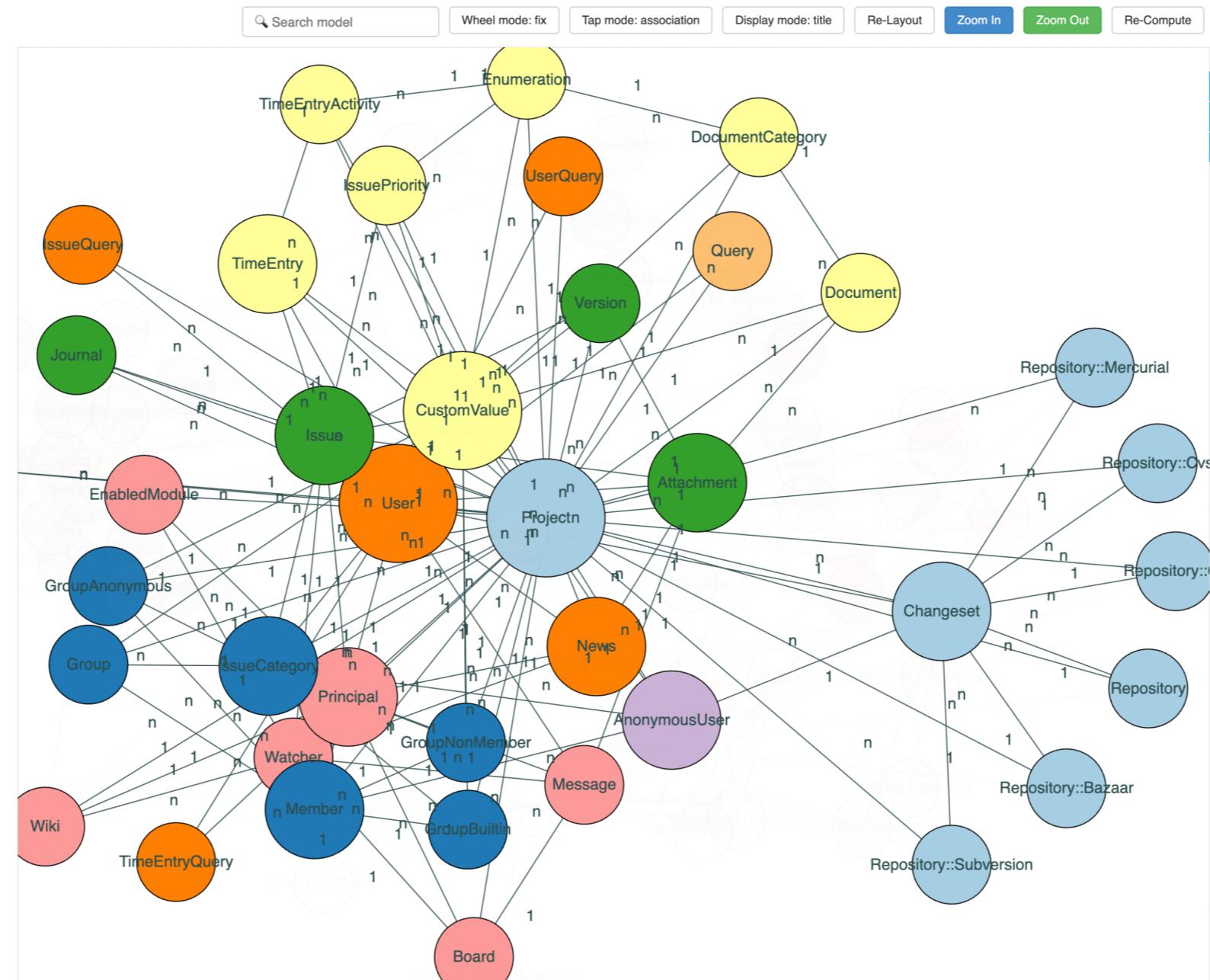
- 素の実装だとコミュニティが巨大になる場合がある
 - 再帰的に分割を試みる

```
def split_communities(graph, communities)
#...
  communities.each do |community|
    if community.size <= MAX_COMMUNITY_SIZE
      result << community
    else
      subgraph = graph.subgraph(community)
      sub_communities = networkx_community.louvain_communities(subgraph).map { |comm| PyCall::List.new(comm).to_a }
      #...
      splitted_sub = split_communities(subgraph, sub_communities)
      result.concat(splitted_sub)
    #...
  end
end
```

ErdMap



■ デモ (Redmine の ErdMap)



Mastdon <https://github.com/redmine/redmine>

Sample HTML https://github.com/makicamel/erd_map/blob/main/sample/redmine.html

ErdMap を使ってみて

ErdMap

- 初見の小規模アプリ（モデル数 100）の場合
 - 重要な概念がわかる
 - アプリの機能や構造・データフローのイメージが湧く
 - 読むべきモデルがわかる
 - ErdMap が索引になる

ErdMap

- 触り慣れている中規模アプリ（モデル数 1,400）の場合
 - 発見が多い
 - 最近こんなモデルが作られていたのか、とか
 - 気になっていたあのモデルはこういうものだったのか、とか
 - あのサービスの主要モデルの関連はこうなっているのか、とか

ErdMap

■ 楽

- 並列に並ぶ多くのファイルから概念や構造を見出すのは大変
- 重みづけされグルーピングされて可視化されているとかんたん
 - 最新の状態の簡易的な ER 図がいつでもそこにある

■ 可視化面白い

- 見えるようにすると何かが見える（かも）
 - 設計の改善点、ボトルネック...

Rails アプリケーションの「地図」

- 役に立つかかもしれない、立たないかもしれない
- 面白いなあと感じている
- ぜひ試してみてください

Special Thanks

@youchan

ご清聴ありがとうございました