尊敬的吴老师，您好

我完成了基于no-overlapping community的diffusion minimization problem的仿真，仿真结果与原文基本一致。此处的原文是指Information Diffusion in Mobile Social Networks: The Speed Perspective（2014），与作者在2016年发表的Towards Information Diffusion in Mobile Social Networks的内容基本一致，不过2016的那篇是基于overlapping community的。下面先介绍论文内容，最后附上一些我的个人理解。

1. 问题描述

最快传播问题：在网络中选择一个大小为k的节点集合，使得从该集合开始传播的信息能够以最短的时间扩散到整个网络。

1.1 网络模型

记$G = (V, E)$为有权无向图，其中V为节点集合，E为边集合。对于两个相邻节点$u, v \in V$，w_{uv}表示两点连边的权重，同时$w_{uv} = w_{vu}$。边的权重反映了节点之间的互动频率。对于节点$u \in V$，d_u表示节点u的度，N_u表示节点u的相邻节点集合，并且$d_u = \sum_{v \in N_u} w_{uv}$。

1.2 传播概率模型

在信息传播模型中，节点的状态可分为激活的和未激活的。激活的节点可以向其未激活的邻居传播信息。具体的，当一个已激活的节点u向未激活节点v传递信息时，节点v被激活的概率为$\lambda_{uv} = \frac{w_{uv}}{d_u}$。这是因为信息从$u$传播到它的邻居$v$的概率应该正比于$v$在$d_u$的比例。换句话说，$u$与$v$的互动越频繁，$v$越有可能被$u$激活。从社会关系的角度来说，人们更喜欢向最好的朋友分享信息，而不是其他人。

1.3 问题描述

信息传播的过程可以这样描述：首先选择一个初始的已激活节点集合，随着已激活节点与未激活节点的互动交流，未激活节点以一定概率被激活，当所有节点都被激活时，传播过程结束。

记S为初始化节点集合，S的传播时间定义为传播过程开始至结束的时间间隔，记为$\tau(S, V)$。

在给定图$G = (V, E)$和一个整数k后，我们的目的是确定一个节点集合S，$|S| \leq k$且$S \subseteq V$，使得$\tau(S, V)$为最小。这个问题通常被称为最小传播问题（diffusion minimization）。

节点u到节点v的期望传播时间记为

$$t_{uv} = \frac{1}{\lambda_{uv}} \cdot \frac{1}{w_{uv}} = \frac{d_u}{w_{uv}} \cdot \frac{1}{w_{uv}} = \frac{d_u}{w_{uv}^2} \tag{1}$$

其中$\frac{1}{w_{uv}}$表示传播的平均时间间隔。对于任何节点对，例如节点u和v，u至v的最短期望传播时间记为$|(u, v)|$，为了简化描述，不妨称$|(u, v)|$为节点u至v的期望传播时间。

最小传播问题可以这样表述：寻找一个集合$S \subseteq V (|S| \leq k)$，使得期望传播时间$\tau(S, V)$：

$$\tau'(S, V) = \min \max_{v \in V} |(S, v)| \tag{2}$$

其中

$$|(S,v)| = \min_{u \in S} |(u,v)| \tag{3}$$

$|(S,v)|$是集合S至节点v的期望传播时间。

1.4 一个朴素算法

节点的紧密度（closeness）是该节点至所有其他节点的最短路径长度之和的倒数，例如节点u的紧密度为$1 \setminus \sum_{v \in V} |(u,v)|$。

紧密度可以看作是节点传播信息速度的度量，紧密度越大，传播速度越快。我们可以不断地从$V \setminus S$中选取最大紧密度的节点加入集合S，直至$|S| = k$。具体的，每次选取过程中节点u的紧密度这样计算

$$\frac{1}{\sum_{v \in V \setminus S} |(u,v)|}, u \notin S \tag{4}$$

2 社区化算法

相比于社区外部节点，社区内部的节点之间连接关系更多。社区结构清晰地表示了节点如何组织以及节点之间如何互动。在信息传播方面，社区有如下属性：

1) 在社区内部，节点互动频繁，信息可以快速传播。

2) 信息在社区之间的传播速度远比社区内部慢。

最基本的社区化算法是在每个社区内部至少选取一个传播节点。记$\mathfrak{C} = \{C_1, C_2, C_3, \cdots, C_l\}$为社区列表，其中$|\mathfrak{C}| = l$，$C_i \in \mathfrak{C}$表示一个社区。假设需要在$\mathfrak{C}$中确定$k$个节点，这里有两种情况：$k < l$和$k \geq l$。对于$k < l$，我们不能保证每个社区中都至少有一个初始传播节点，因此一些社区需要合并。对于$k \geq l$，我们要考虑如何在一个社区中选择多个初始传播节点。所以，接下来我们首先介绍如何合并社区，以确保$k \geq l$，然后介绍如何在一个社区中选择多个初始传播节点。

2.1 社区合并

社区中心节点是在社区内部，对所有其他节点具有最小期望传播时间的节点。社区中心节点的期望传播时间称为社区传播半径。

记N_C和$R(C)$为社区中心节点和社区传播半径，并且

$$R(C) = \min_{u \in C} (\max_{v \in C} |(u,v)|) \tag{5}$$

通过社区合并，社区数量会从l减少至k。之后，整个网络的期望传播时间由具有最大传播半径的社区决定。所以在社区合并的过程中，我们应该最小化$\max\{R(C) : C \in (C)\}$。显然，$R(C_i \cup C_j) > \max\{R(C_i), R(C_j)\}$。因此被合并的社区的传播半径会扩大。我们每次需要合并两个社区，有$\binom{|\mathfrak{C}|}{2}$种选择，显然直接枚举行不通。

一种可行的社区合并方法是合并相邻的社区，使得合并后的社区的传播半径尽可能小。如果C_j满足公式(6)，则称C_j为C_i的相邻社区。C_i的相邻社区集合记为$\mathfrak{C}_{C_i}$

$$\mathfrak{C}_{C_i} = \left\{ C_j \in \mathfrak{C} \setminus \{C_i\} : \frac{\sum_{u \in C_i, v \in C_j} w_{uv}}{|C_j|} \geq \frac{\sum_{u \in C_i, v \in V \setminus C_i} w_{uv}}{|V \setminus C_j|} \right\} \tag{6}$$

社区合并过程如下：首先，我们在$\mathfrak{C}$选择具有最小传播半径的社区C_i；从C_i的相邻社区集$\mathfrak{C}_{C_i}$中选择C_j，使得$R(C_i \cup C_j)$最小，如果$R(C_i \cup C_j) \leq \max\{R(C) : C \in \mathfrak{C}\}$，那么合并$C_i$和$C_j$，否则，遍历所有其他的社区。如果找不到满足$R(C_i \cup C_j) \leq \max\{R(C) : C \in \mathfrak{C}\}$的社区，选择具有最小$R(C_i \cup C_j)$的社区$C_i$和$C_j$来进行合并。具体算法过程见Algorithm 1。

Algorithm 1 Community Merging

Input: $\mathfrak{C}$

Output: $\mathfrak{C}$

while $|\mathfrak{C}| > k$ **do**
  $\mathfrak{C}' = \mathfrak{C}$
  $C_i = C_j = \emptyset$
  while $|\mathfrak{C}'|$ **do**
    $C_i' = \arg\min_{C \in \mathfrak{C}'} R(C)$
    $C_j' = \arg\min_{C \in \mathfrak{C}'_{C_i}} R(C_i \cup C)$
    if $R(C_i' \cup C_j') < \max\{R(C) : C \in \mathfrak{C}\}$ **then**
      $C_i = C_i'$ and $C_j = C_j'$
      break
    else
      if $R(C_i \cup C_j) \leq R(C_i' \cup C_j')$ **then**
        $C_i = C_i'$ and $C_j = C_j'$
      end if
      $\mathfrak{C}' = \mathfrak{C}' \setminus \{C_i'\}$
    end if
  end while
  $\mathfrak{C} = \mathfrak{C} \setminus \{C_i, C_j\} \cup \{C_i \cup C_j\}$
end while

2.2 确定初始传播节点

经过社区合并后，$|\mathfrak{C}| \leq k$，因此我们需要在一个社区内确定多个传播节点。

记S_{C_i}为C_i中的初始传播集合，记$|(S_{C_i}, u)|$为S_{C_i}至节点$u \in C_i$的期望传播时间。为了在社区内确定多个初始传播节点，我们按照公式(7)来迭代选择节点u加入集合S_{C_i}

$$\arg\min_{u \in C_i \setminus S_{C_i}} \sum_{v \in C_i \setminus S_{C_i}} |(S_{C_i} \cup \{u\}, v)| \tag{7}$$

S_{C_i}选择完成后，社区C_i的期望传播时间记为$\tau'(S_{C_i}, C_i)$

2.3 算法设计

合理的选择初始传播节点可以有效地减少传播时间。一种最直接的方法是先为每个社区选择中心节点，然后迭代地在满足$R(C) : C \in (C)$的社区中选择初始传播节点，直至$|S| = k$。然而，这种方法并不总是有效的。举例来说，$\mathfrak{C}$中有原始社区C_i，已合并社区C_j和C_k，且$R(C_i) > \max\{R(C_j), R(C_k)\}$。那么显然在$C_i$中选择多个初始节点，在$C_j$和$C_k$中各选择一个初始节点更优。

社区化算法工作流程如下。首先我们先合并社区，直至$\forall C_i, C_j : R(C_i \cup C_j) > \max\{R(C) : C \in (C)\}$；然后，我们为每个选择中心节点作为初始传播节点，这样我们仍有$k-(C)$个初始传播节点待定；接下来，我们为具有最大传播半径的社区C_i选择额外的初始传播节点。如果C_i是已合并社区，那么将C_i分裂为社区C_j和C_k，用C_j和C_k的中心节点替代C_i的中心节点。如果C_i是原始社区，那么我们先根据公式(7)选取一个初始传播节点来代替N_{C_i}，然后在确定更多的初始传播节点，并加入集合S。算法细节见Algorithm 2。

Algorithm 2 Community based Algorithm

Input: $\mathfrak{C}$

Output: S

$\mathfrak{C}' = \mathfrak{C}$
while $\exists C_i, C_j : R(C_i \cup C_j) < \max_{C\in\mathfrak{C}'} R(C)$ **do**
  $\mathfrak{C}' = \mathfrak{C}' \setminus \{C_i, C_j\} \cup \{C_i \cup C_j\}$
end while
$S = \{N_C : C \in \mathfrak{C}'\}$
while $|S| < k$ **do**
  $C = \arg\max_{C\in\mathfrak{C}'} R(C)$
  if $C \in \mathfrak{C}$ **then**
    if $|S_C| = 1$ **then**
      $S = S \setminus \{N_C\}$
      $S = S \cup \left\{\arg\min_{u\in C\setminus S_C} \sum_{v\in C\setminus S_C} |(S_C \cup \{u\}, v)|\right\}$
    end if
    $S = S \cup \left\{\arg\min_{u\in C\setminus S_C} \sum_{v\in C\setminus S_C} |(S_C \cup \{u\}, v)|\right\}$
  else
    $S = S \cup \{N_C\} \cup \{N_{C_i}, N_{C_j}\}$
    $\mathfrak{C}' = \mathfrak{C}' \setminus \{C\} \cup \{C_i \cup C_j\}$
  end if
end while

3 仿真实验

我们将在人工合成数据集上对比社区化算法(Community)和朴素算法(Naive)，数据集来自于Benchmark程序，其参数列表如表1所示。其中n为节点数量，α为节点邻居数量，α_{max}为节点最大邻居数量，μ_w为边权重混合度，μ_t为图拓扑混合度，ε_1为节点度的最小幂，ε_2为社区大小分布的最小幂，β为权重分布最小幂。

Parameter	Value	Parameter	Value
n	500,1000,2000	ε_1	2
μ_w	0.1,0.3	ε_2	1
μ_t	0.1,0.3	α	15
β	2	α_{max}	20

表 1: benchmark 参数设置

图1展示了社区化算法和朴素算法在不同网络参数下的期望传播时间。在实际应用中，k通常是一个较小值，在这里我们选取的k值不会超过网络节点数的5%。

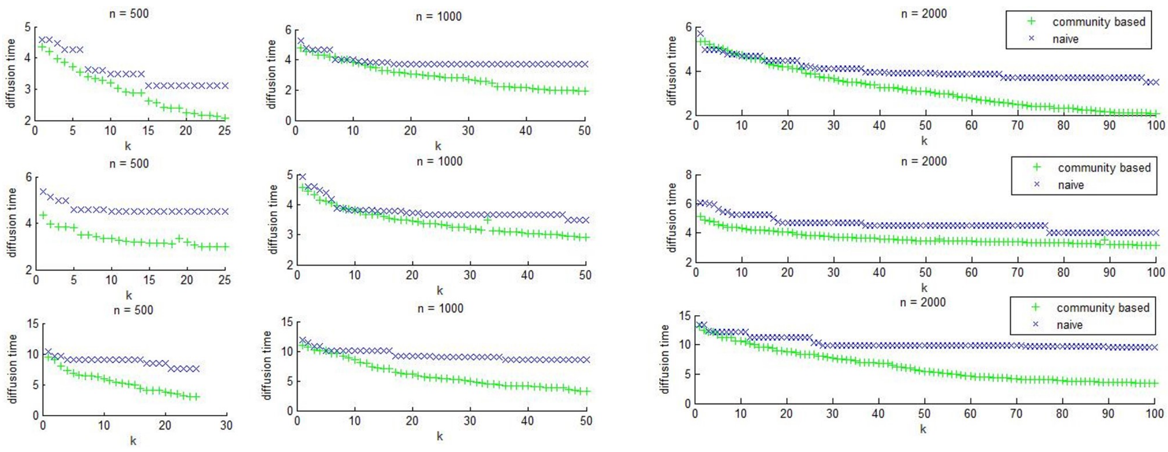

图 1: Community vs Naive

4 补充说明

作者在公式(1)中说明了如何计算两个相邻节点的期望传播时间，但关于如何计算任意两个节点的期望传播时间未做说明。我这里用了Dijkstra算法得到所有节点对的最短路径权值矩阵M，其中M_{uv}表示节点u到v的期望传播时间$|(u,v)|$。这样对于公式(3)而言，其实就是对矩阵M的某些行的所有元素求最大；对于公式(4)而言，则是对矩阵M的某一行求和取倒数；而公式(5)则是对矩阵M的某些行先求行最大，再对结果求最小；公式(7)是对矩阵M的某些行的所有元素求和。

我是用C++写的程序，对于规模为500，1000，2000的网络，运行时间大致为1s，1-5mins，20-60mins，并且时间都消耗在了Dijkstra算法上面。程序代码见`https://github.com/Chaomin702/Community`。

接下来我想去做

1) 基于overlapping community的最小传播问题仿真。

2) distributed set-cover algorithm仿真

学生王超民，2016年10月14日