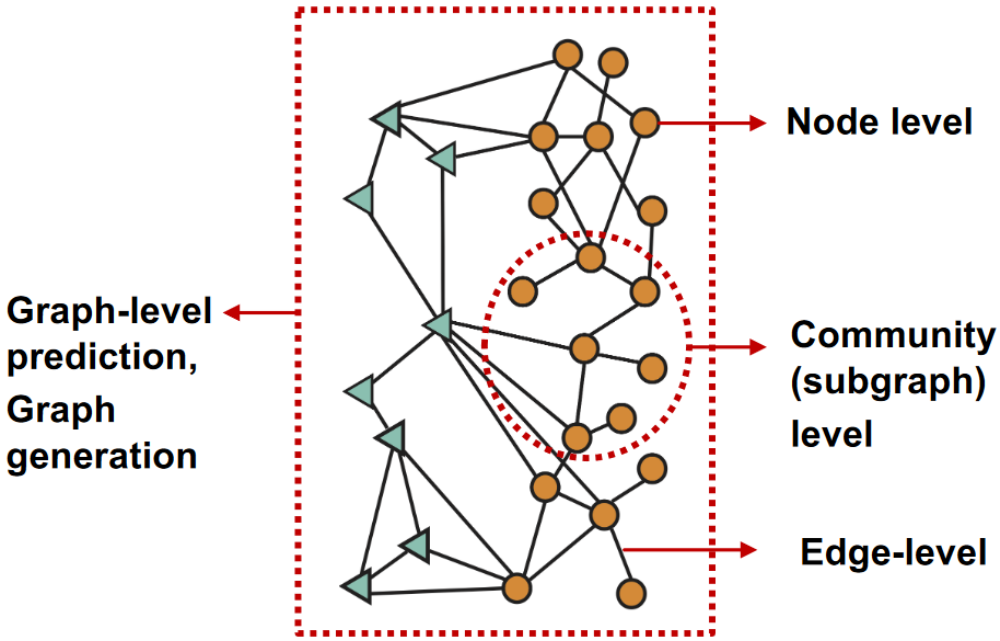
Traditional Method for ML in Graphs

图机器学习的任务可以分为三类：

- Node-level task 如蛋白质结构预测
- Edge-level task 如推荐系统中，$User \rightarrow item$的推荐预测
- Graph-level task 如分子毒性的预测



在传统的ML学习中，对于Graph而言，我们先对于Node/Edge/Graph进行对于的特征构造，之后输入ML模型进行预测。那么预测结果的好坏就与特征质量有直接的联系。传统ML的核心是构造出能够有效概括和总结图结构的特征，对于Node而言，构造的特征需要能够表示出Node在Graph中的位置和周围的图结构。

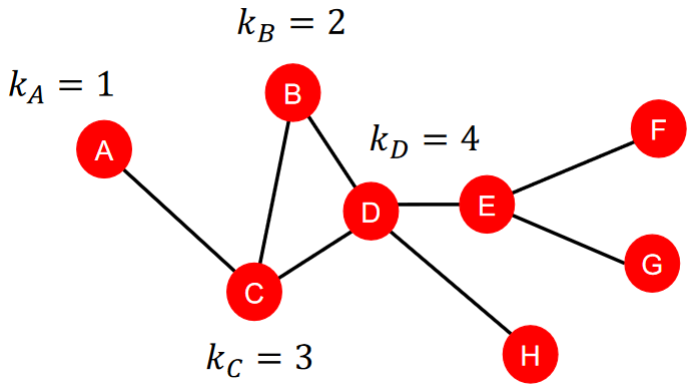
我们现在有$G = (V, E)$，目标是学得函数$f : V \rightarrow R$，那么我们应该怎么样学习f呢?

Node features

对于Node-level features，主要包括了Node degree，Node centrality，Clustering coefficient，Graphlets。

Node Degree

与节点相连的边的条数。对于有向图，Node Degree再细分为In Degree 和 Out Degree,使用k_v表示



Node Centrality

Node Degree将所有的边都看作是等价的，没有考虑Node Importance。Node Centrality c_v将Node Importance纳入了考虑范围。有如下几种Node Centrality的计算方式：Engienvector centrality，Betweenness centrality，Closeness centrality and many others...

Engienvector centrality认为如果一个节点v的邻居节点$u = N(v)$是重要的，那么这个节点自身也是重要的。我们将Centrality看作是一个节点其邻居节点的Centrality之和。

$$c_V = \frac{1}{\lambda} \sum_{u \in N(v)} c_u \qquad \lambda \text{ is some positive constant.}$$

上述的表达式求解是一个递归问题。我们可以将上式转化为矩阵形式：

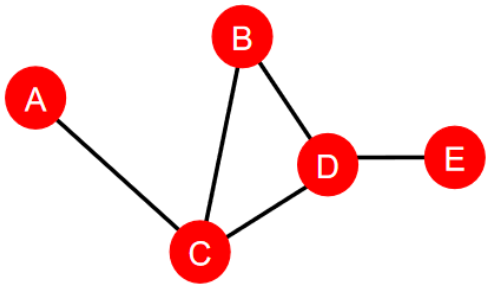
$$\lambda c = Ac \qquad \text{A为邻接矩阵，} A_{uv} = 1 \; if \; u \in N(v)$$

可以发现上式也是矩阵A的特征值λ对应的特征向量c的定义。我们将矩阵A对应的最大特征值λ_{max}对应的特征向量c_{max}作为centrality

Betweenness centrality认为如果一个节点在许多节点的最短路径上，那么这个节点是重要的，即

$$c_v = \sum_{s \neq v \neq t} \frac{\#(shortest\ paths\ bewteen\ s\ and\ t\ contain\ v)}{\#(shortest\ paths\ bewteen\ s\ and\ t)}$$

- Example:

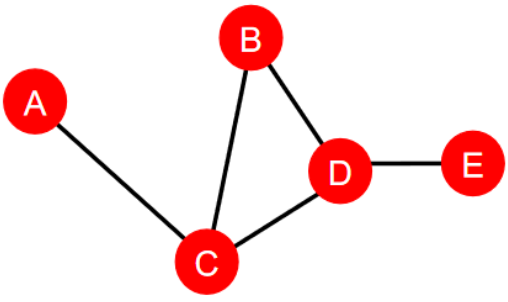


$$c_A = c_B = c_E = 0$$
$$c_C = 3$$
(A-C-B, A-C-D, A-C-D-E)
$$c_D = 3$$
(A-C-D-E, B-D-E, C-D-E)

Closeness centrality认为如果节点到其他节点的最短路径和很小，那么这个节点是重要的，即

$$c_v = \sum_{v \neq u} \frac{1}{shortest\ path\ length\ bewteen\ v\ and\ u}$$

Example:



$c_A = 1/(2+1+2+3) = 1/8$
(A-C-B, A-C, A-C-D, A-C-D-E)

$c_D = 1/(2+1+1+1) = 1/5$
(D-C-A, D-B, D-C, D-E)

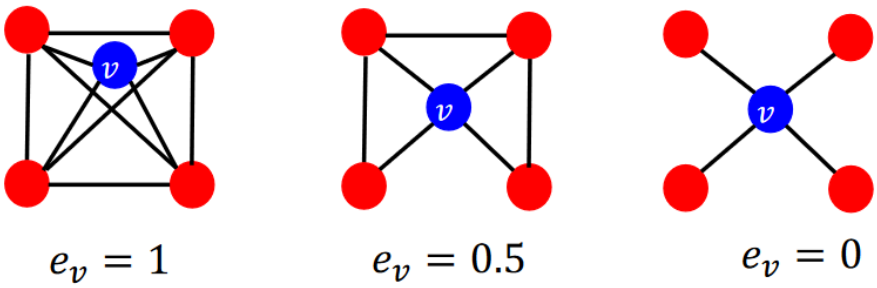
Clustering coefficient

Clustering coefficient衡量了一个节点的邻居节点的相互连接情况。即我的朋友们之间是否是互相认识。

$$e_v = \frac{\#(edges\ among\ neighbour\ nodes)}{C_{k_v}^2}$$

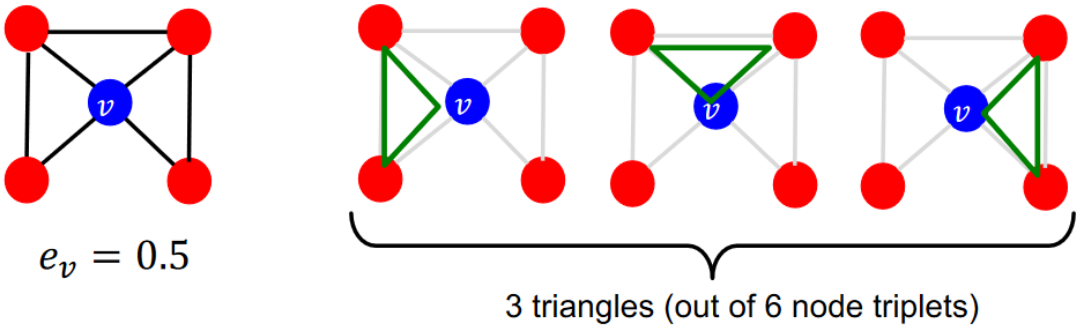
其中$C_{k_v}^2$表示邻居节点中任取两点的数量。

- **Examples:**



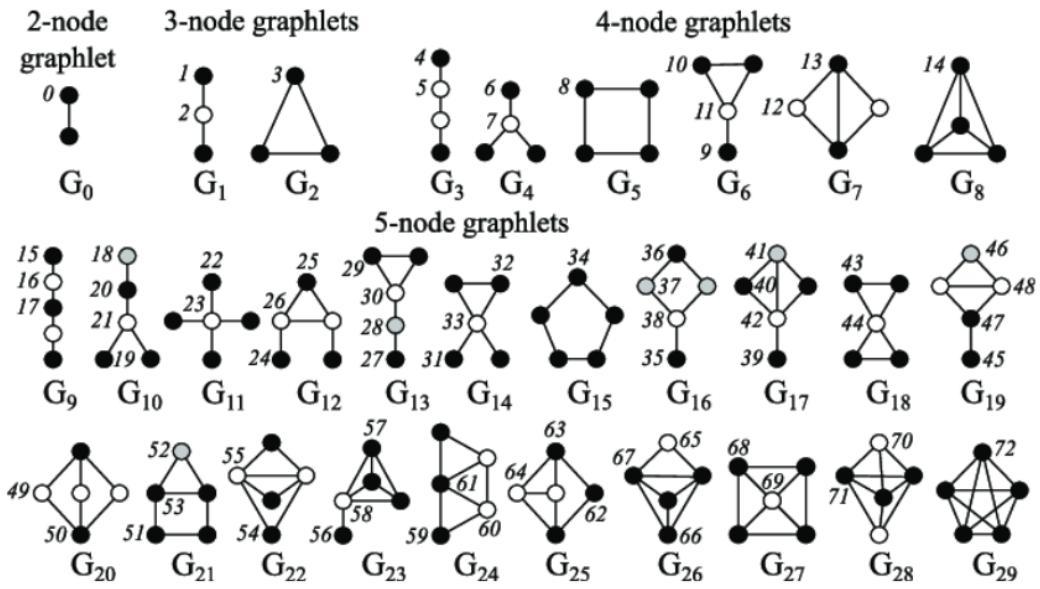
Graphlet 图元

- **Observation:** Clustering coefficient counts the #(triangles) in the ego-network



Clustering coefficient实际上对邻居节点和节点之间，构成的三角形进行计数。Graphlet对其进行了泛化，我们是否可以对Graph的某些特定子图结构进行计数，形成对应的计数向量呢?

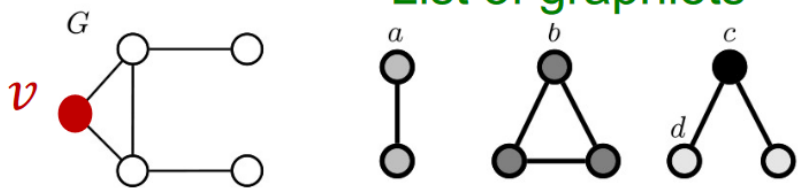
Graphlets: Rooted connected non-isomorphic subgraphs:

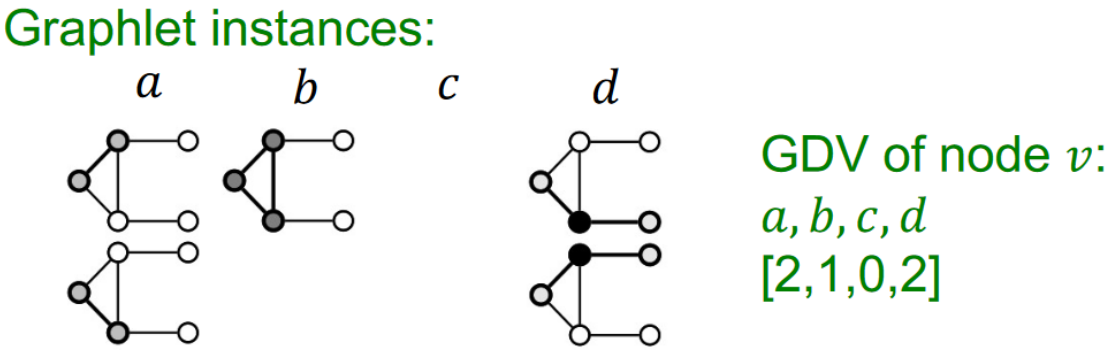


Graphlet对子图的形状进行了定义，并对这些子图的顶点位置进行了区分，如对于3顶点的graphlet而言，G1中端点的node实际上是一样的，相当于一个位置，中心的节点相当于第二个位置。对于G2而言，三角形的三个顶点都是等价的，相当于一个位置。

Graphlet Degree Vector (GDV)衡量了node接触到的图元graphlet个数（GDV counts #(graphlets) that a node touches)

- **Example:**





如上图所示，节点v在graph中可以满足三种graphlet，其中2 nodes的graphlet有两种可能性。三角形的3 nodes graphlet有一种可能性，折线形的的graphlet只能满足端点的两种可能，不满足中心点的可能性。所以得到的GDV为【2，1，0，2】

graphlet在5 nodes内共有73中可能性，那么对于一个顶点4 hops之内的特征而言，我们都可以用一个73维向量来表达。同时graphlet也是一种表达节点局部结构特征的方法，相较于之前的node feature，graphlet提供了关于node structure的更多信息。

上述的node feature可以总结为：

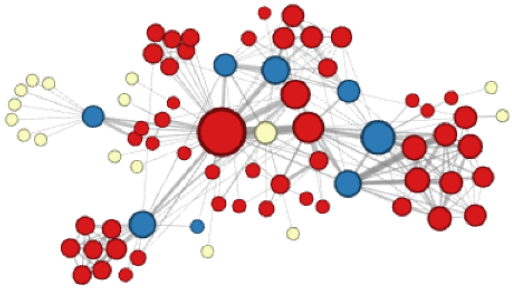
- Importance-based features:
  - Node degree
  - Different node centrality measures
- Structure-based features:
  - Node degree
  - Clustering coefficient
  - Graphlet count vector

Importance-based features捕捉了graph中node的importance。Node degree简单计数了邻居节点的个数，Node Centrality考虑了邻居节点的importance。这些特征通常在预测节点在图中的影响力时很有帮助，如预测社交网络中的大V用户。

Structure-based features捕捉了节点的局部邻居的性质。Node degree可以看做捕捉了邻居的数量，Cluster Coefficient捕捉了局部邻居的连接情况，GDV捕捉了局部邻居的结构特征。这些特征在预测特定节点在图中的角色是很有帮助，如Predicting protein functionality in a protein-protein interaction network.

基于这些特征，我们可以用不同的feature区分不同的node。

Different ways to label nodes of the network:

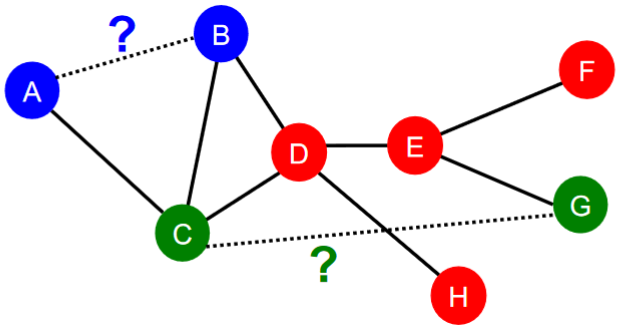


Node features defined so far would allow to distinguish nodes in the above example

However, the features defines so far would not allow for distinguishing the above node labelling

Link Prediction Task and Features

- The task is to predict **new links** based on existing links.
- At test time, all node pairs (no existing links) are ranked, and top K node pairs are predicted.
- The key is to design features for **a pair of nodes**.



Link Prediction Task的目的是通过已有的edge来预测新的edge，关键在于如何设计特征来表达一对node和他们的edge？我们有两种设计Link Prediction的模式

- Links missing at random 随机移除一些图中的edge并且预测他们
- Links over time 给出一张图从t_o到t_0'的egde，并且预测在t_1到t_1'时间内新出现的edge。在测试时，我们找到t_1到t_1'时间内真实出现的edge数目n，挑选预测集中最可能出现的n条edge，检验预测集edge和真实edge的重合数量。

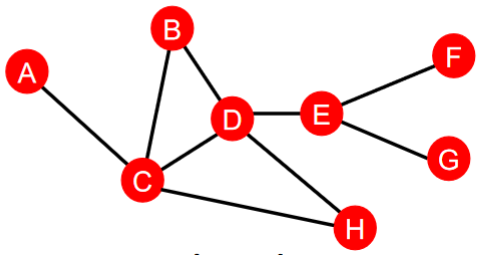
Link Prediction Features可以被分为以下三种类型。

- Distance-based feature
- Local neighborhood overlap
- Global neighborhood overlap

Distance-based feature

两点之间的最短距离

- Example:



$$S_{BH} = S_{BE} = S_{AB} = 2$$
$$S_{BG} = S_{BF} = 3$$

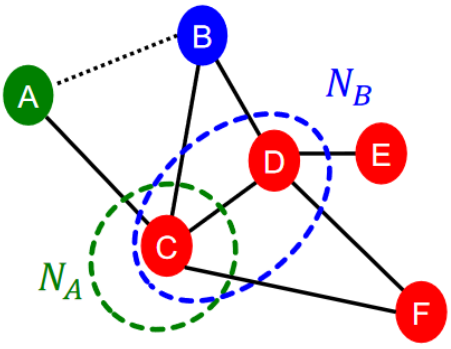
但是最短距离不能捕捉两点之间的公共neighbour，比如对于点对（B，H），他们的公共邻居为C,D。但是（B,E）的公共邻居只有D。但是他们的最短距离都为2.

Local neighborhood overlap

Captures # neighboring nodes shared between two nodes v_1 and v_2:

- **Common neighbors:** $|N(v_1) \cap N(v_2)|$
  - Example: $|N(A) \cap N(B)| = |\{C\}| = 1$
- **Jaccard's coefficient:** $\frac{|N(v_1) \cap N(v_2)|}{|N(v_1) \cup N(v_2)|}$
  - Example: $\frac{|N(A) \cap N(B)|}{|N(A) \cup N(B)|} = \frac{|\{C\}|}{|\{C,D\}|} = \frac{1}{2}$
- **Adamic-Adar index:**

  $\sum_{u \in N(v_1) \cap N(v_2)} \frac{1}{\log(k_u)}$
  - Example: $\frac{1}{\log(k_C)} = \frac{1}{\log 4}$



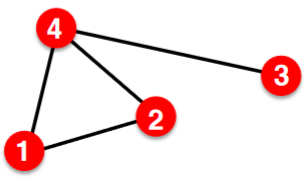
Local neighborhood overlap捕捉了两点之间公共的neighbor个数。

Global neighborhood overlap

Local neighborhood overlap的缺陷在于，如果两点之间不存在任何的公共邻居节点，那么Local neighborhood overlap的任何指标都将是0.但是在这两点仍然有可能会互相连接。Global neighborhood overlap从整张图的角度进行考虑，从而避免了这个问题。

Katz index计数了两点之间的所有可能路径数目。我们可以通过邻接矩阵A的幂函数来求解这个问题。

- **Computing #paths between two nodes**
  - **Recall:** $\boldsymbol{A_{uv}} = 1$ if $u \in N(v)$
  - Let $\boldsymbol{P_{uv}^{(K)}}$ = #paths of length $\boldsymbol{K}$ between $\boldsymbol{u}$ and $\boldsymbol{v}$
  - We will show $\boldsymbol{P^{(K)} = A^k}$
  - $\boldsymbol{P_{uv}^{(1)}}$ = #paths of length 1 (direct neighborhood) between u and $v = \boldsymbol{A_{uv}}$



$$\boldsymbol{P_{12}^{(1)} = A_{12}}$$

$$A = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

首先定义$P_{uv}^{(K)}$为节点uv之间所有路径长度为K的路径数目。可以看出$A_{uv} = P_{uv}^{(1)}$。我们接下来计算$P_{uv}^{(2)}$

- **How to compute $P_{uv}^{(2)}$?**
  - **Step 1:** Compute **#paths** of length 1 **between each of u's neighbor and v**
  - **Step 2**: **Sum up** these #paths across u's neighbors
  - $\boldsymbol{P_{uv}^{(2)} = \sum_i A_{ui} * P_{iv}^{(1)} = \sum_i A_{ui} * A_{iv} = A_{uv}^2}$

Node 1's neighbors

#paths of length 1 between Node 1's neighbors and Node 2

$P_{12}^{(2)} = A_{12}^2$

$$A^2 = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \times \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 3 \end{pmatrix}$$

Power of adjacency

计算$P_{uv}^{(2)}$时，我们首先计算所有从u的邻居节点到v的长度为1的路径数目，之后相加得到总的路径数目。上述的思路可以被写作$P_{uv}^{(2)} = \sum_i A_{ui} * P_{iv}^{(1)} = \sum_i A_{ui} * A_{iv} = A_{uv}^2$。我们可以得到$P_{uv}^{(2)}$就是A的平方。递归得到$P_{uv}^{(K)}$

- $\boldsymbol{A_{uv}}$ specifies #paths of length 1 (direct neighborhood) between u and v.
- $\boldsymbol{A_{uv}^2}$ specifies #paths of **length 2** (neighbor of neighbor) between u and v.
- And, $\boldsymbol{A_{uv}^l}$ specifies #paths of **length l**.

Katz index在计算是考虑了两节点之间所有长度的路径数目，对于越长的路径，施加了惩罚项β^l,l为路径长度。

- **Katz index** between v_1 and v_2 is calculated as

Sum over *all path lengths*

$$S_{v_1 v_2} = \sum_{l=1}^{\infty} \beta^l A_{v_1 v_2}^l$$

#paths of length l between v_1 and v_2

$0 < \beta < 1$: discount factor

- Katz index matrix is computed in closed-form:

$$\boldsymbol{S} = \sum_{i=1}^{\infty} \beta^i \boldsymbol{A}^i = \underbrace{(\boldsymbol{I} - \beta \boldsymbol{A})^{-1}}_{= \sum_{i=0}^{\infty} \beta^i \boldsymbol{A}^i} - \boldsymbol{I},$$

by geometric series of matrices

我们可以对三种Link Prediction Features做出如下的总结。

- **Distance-based features:**
  - Uses the shortest path length between two nodes but does not capture how neighborhood overlaps.
- **Local neighborhood overlap:**
  - Captures how many neighboring nodes are shared by two nodes.
  - Becomes zero when no neighbor nodes are shared.
- **Global neighborhood overlap:**
  - Uses global graph structure to score two nodes.
  - Katz index counts #paths of all lengths between two nodes.

Graph-Level Features and Graph Kernels

我们希望构造能够捕捉整张图结构的特征。常见的方法是核方法（Kernel Method）

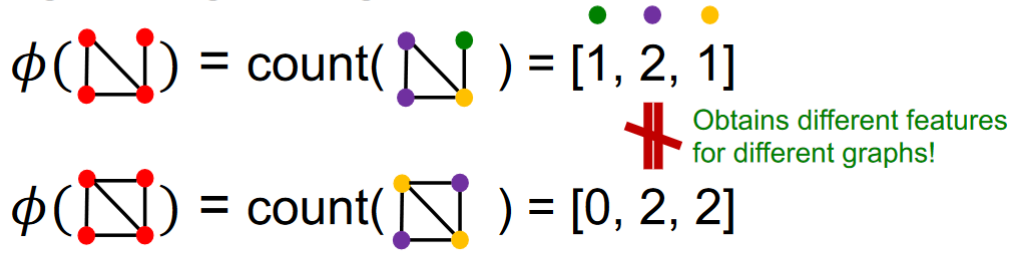
Graph Kernels: Measure similarity between two graphs:

- Graphlet Kernel [1]
- Weisfeiler-Lehman Kernel [2]
- Other kernels are also proposed in the literature (beyond the scope of this lecture)
  - Random-walk kernel
  - Shortest-path graph kernel
  - And many more…

与常见的Kernel Method相似，Graph Kernel实际上是给出了两张图之间的相似性度量。

What if we use Bag of **node degrees**?

Deg1: ● Deg2: ● Deg3: ●



- Both Graphlet Kernel and Weisfeiler-Lehman (WL) Kernel use **Bag-of-*** representation of graph, where * is more sophisticated than node degrees!

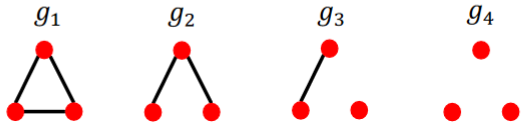
Graphlet Feature

Graphlet Feature 的想法是在一张图中出现的不同grahplet进行计数。但是Graph-level的特征和node level特征的graphlet定义略有差别。

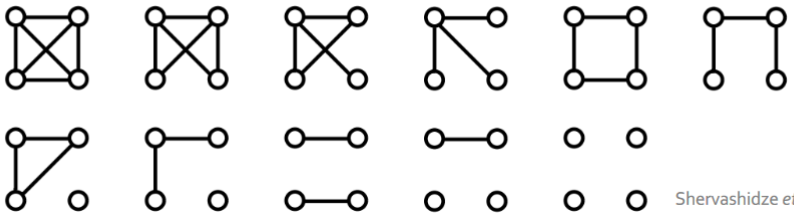
Graph-Level的graphlet中，不再要求节点都是连接的，即允许孤立点的存在。其次，The graphlets here are not rooted.即不在区分graphlet中点的位置。

Let $\mathcal{G}_k = (g_1, g_2, \ldots, g_{n_k})$ be a list of graphlets of size k.

- For $k = 3$, there are 4 graphlets.



- For $k = 4$, there are 11 graphlets.



Shervashidze *et al.*, AISTATS 2011

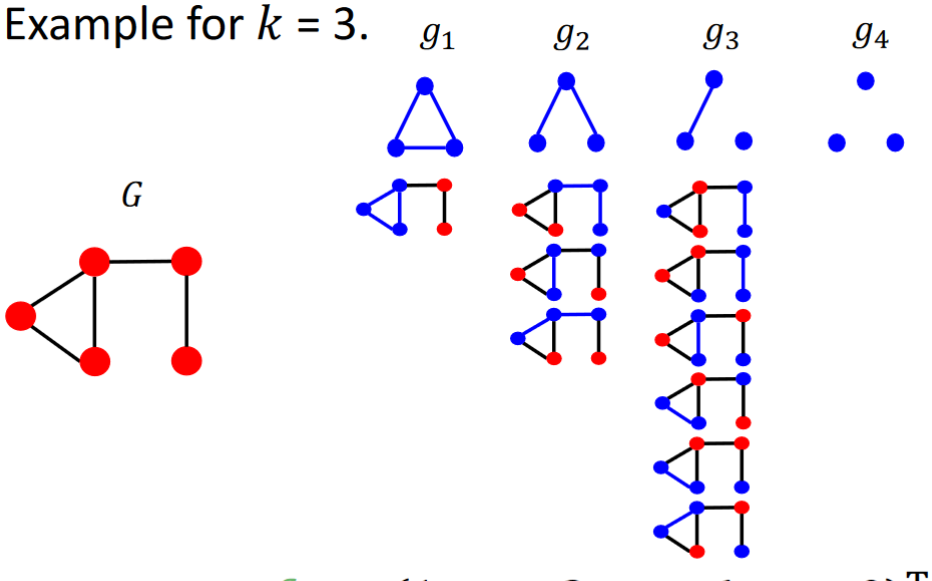
以上图为例，在k=3时，由于允许孤立点的存在，我们有四种graphlet。并且我们不再区分graphlet中点的位置区别（因为我们的feature针对的是Graph-Level）。

Given graph G, and a graphlet list $\mathcal{G}_k = (g_1, g_2, \ldots, g_{n_k})$, define the graphlet count vector $f_G \in \mathbb{R}^{n_k}$ as

$$(f_G)_i = \#(g_i \subseteq G) \text{ for } i = 1,2, \ldots, n_k.$$

对于给定的图G,对应的graphlet count vector的第i个分量对应graphletg_i在G中存在的数量。可以给出如下的例子：

- Example for $k = 3$.



$$f_G = (1, \quad 3, \quad 6, \quad 0)^{\mathrm{T}}$$

现在我们得到了对于某一张图G的graphlet vectorf_G，那么对于两张图之间的相似度$Graph\ Kernel$我们可以定义为

$$K(G_1, G_2) = f_{G_1}^T f_{G_2}$$

为了避免Graph之间的大小差异造成的f_G差异过大，我们将f_G进行标准化后再做内积，即$h_G = \frac{f_G}{SUM(f_G)}$

$$K(G_1, G_2) = h_{G_1}^T h_{G_2}$$

Graphlet Kernel的缺点在于计算代价过大，如果要计算对应的count vector。我们需要遍历k次graph，n^k。Weisfeiler-Lehamn Kernel解决了这个问题。

Weisfeiler-Lehamn Kernel

Weisfeiler-Lehamn Kernel use neighborhood structure to iteratively enrich node vocabulary. Weisfeiler-Lehamn Kernel的思想可以被看做是Bag of Node 的拓展，node degree只考虑了1 hop的邻居节点信息，Weisfeiler-Lehamn Kernel 拓展到了多hops。

Weisfeiler-Lehamn Kernel 的实现算法为Color refinement。

- **Given:** A graph G with a set of nodes V.
  - Assign an initial color $c^{(0)}(v)$ to each node v.
  - Iteratively refine node colors by

$$c^{(k+1)}(v) = \text{HASH}\left(\left\{c^{(k)}(v), \left\{c^{(k)}(u)\right\}_{u \in N(v)}\right\}\right),$$

    where HASH maps different inputs to different colors.
  - After K steps of color refinement, $c^{(K)}(v)$ summarizes the structure of K-hop neighborhood

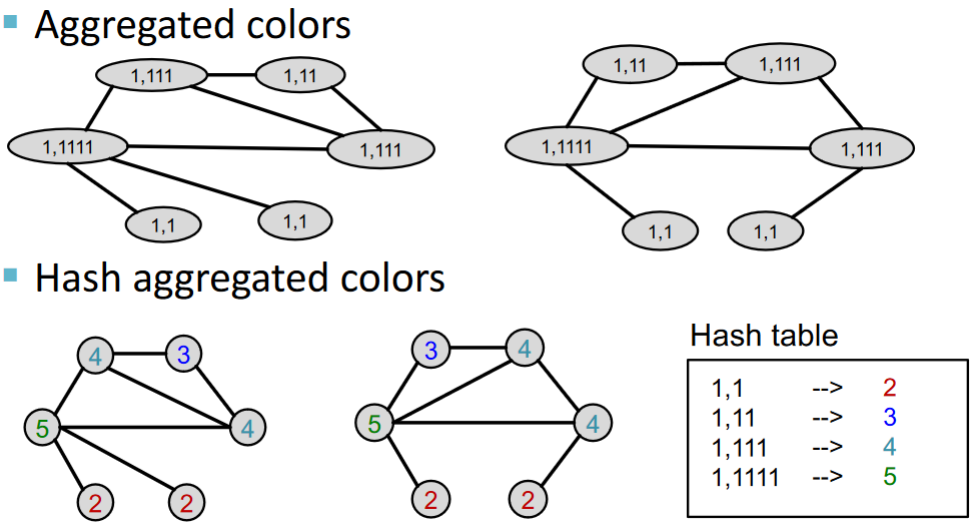
首先我们对图中的每一个node给定相同的颜色$c^{(0)}(v)$,之后根据第k轮时节点v和v的邻居节点的颜色，映射k+1轮节点v的颜色。映射的原则为不同的input必须映射到不同的颜色上。在经过K轮迭代后，$c^{(K)}(v)$可以概括节点v的k hops的结构特征。我们给出如下的例子：

Example of color refinement given two graphs

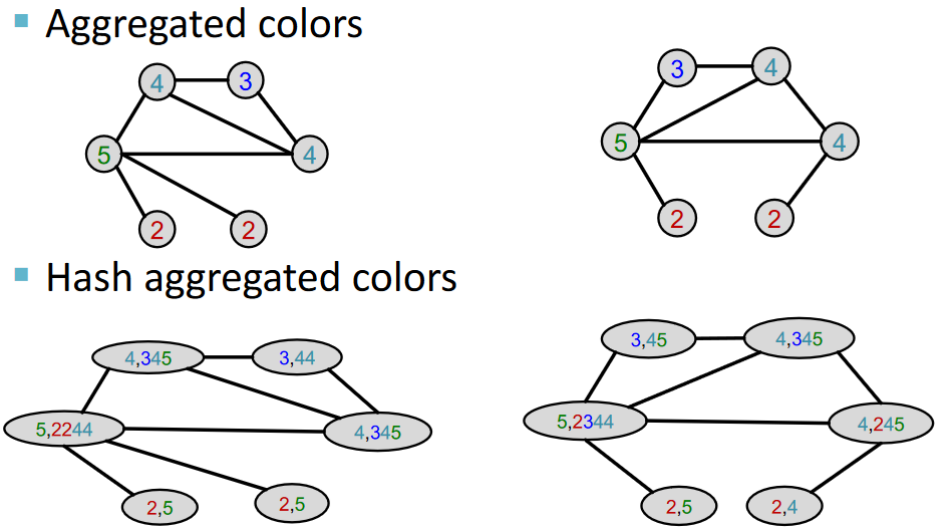
- Assign initial colors

- Aggregate neighboring colors

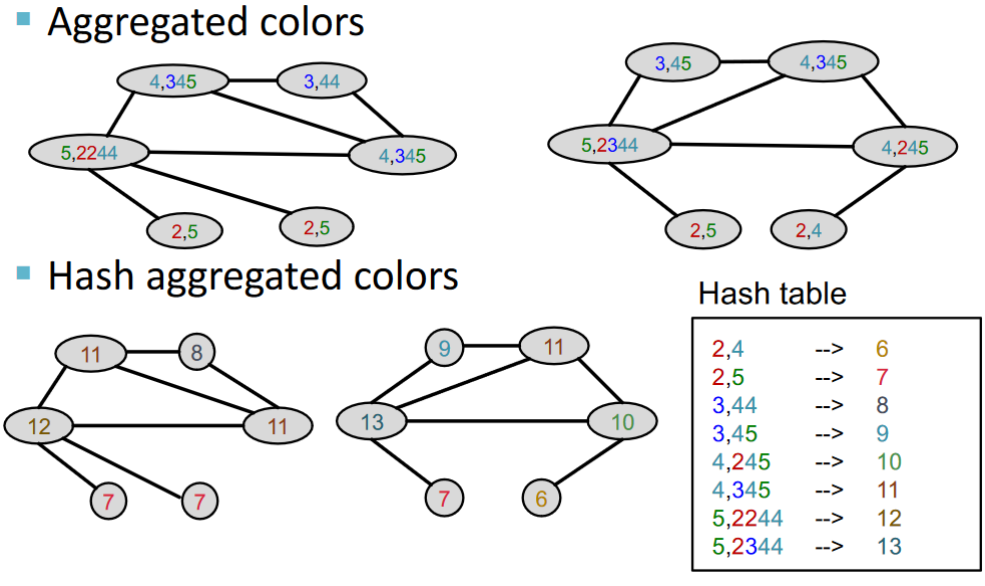
第一轮each node given the same color，我们汇总节点本身和邻居节点的color，得到了hash function的input。而不同的input对应着不同的新颜色。



以1,111为例，所有input为1,111的node都被映射成了浅蓝色。我们根据这一轮的颜色，继续进行加总。

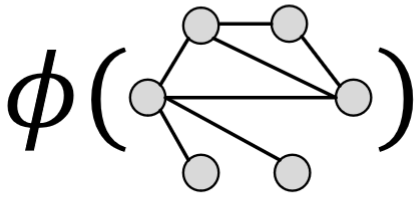


同样的将不同的input映射到不同的颜色上。

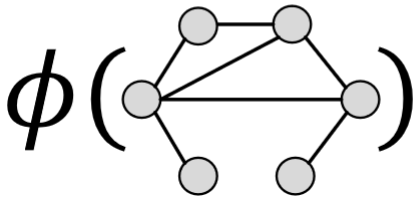


在两轮迭代后，得到的特征向量概括了2 hops的邻居节点信息。

After color refinement, WL kernel counts number of nodes with a given color.



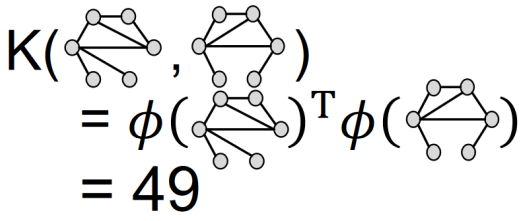
Colors
1,2,3,4,5,6,7,8,9,10,11,12,13
= [6,2,1,2,1,0,2,1,0, 0, 0, 2, 1]
Counts



1,2,3,4,5,6,7,8,9,10,11,12,13
= [6,2,1,2,1,1,1,0,1, 1, 1, 0, 1]

得到的特征向量进行内积就得到了Weisfeiler-Lehamn Kernel.

The WL kernel value is computed by the inner product of the color count vectors:



WL kernel在计算上更加有效，每一轮迭代的时间复杂度线性相关与graph中edge的数量(#edges)（两张图合计）。在计算kernel value时，我们只需要计算在两张图中都出现过的颜色的乘积即可（在任意一张图上没出现，乘积为0）。因此需要的颜色数目最多为node的总数（两张图合计）。也就是说对于颜色的计数时间复杂度线性相关与#（nodes）。总的来说，算法的时间复杂度线性相关于#（edges）

- **Graphlet Kernel**
  - Graph is represented as **Bag-of-graphlets**
  - **Computationally expensive**
- **Weisfeiler-Lehman Kernel**
  - Apply K-step color refinement algorithm to enrich node colors
    - Different colors capture different K-hop neighborhood structures
  - Graph is represented as **Bag-of-colors**
  - **Computationally efficient**
  - Closely related to Graph Neural Networks (as we will see!)