Lorem Ipsum Dolor

node2vec: Scalable Feature Learning for Networks

Abstract

- Problem:present feature learning approaches are not expressive enough to capture the diversity of connectivity patterns observed in networks.
- Node2vec:we learn a mapping of nodes to a low-dimensional space of features that maximizes the likelihood of preserving network neighborhoods of nodes. We define a flexible notion of a node's network neighborhood and design a biased random walk procedure, which efficiently explores diverse neighborhoods.

Intro

- we have to construct a feature vector representation for the nodes and edges.
- (1)hand-engineering domain-specific features based on expert knowledge
- (2)*learn* feature representations by solving an optimization problem

Related work

- word2vec : skip-gram
- DeepWalk

Optimization problem

$$\max_f \quad \sum_{u \in V} \log Pr(N_S(u)|f(u)).$$

条件独立性：

$$Pr(N_S(u)|f(u)) = \prod_{n_i \in N_S(u)} Pr(n_i|f(u)).$$

节点之间对称性：

$$Pr(n_i|f(u)) = \frac{\exp(f(n_i) \cdot f(u))}{\sum_{v \in V} \exp(f(v) \cdot f(u))}$$

最后目标函数：

$$\max_f \quad \sum_{u \in V} \left[-\log Z_u + \sum_{n_i \in N_S(u)} f(n_i) \cdot f(u) \right]$$

Search strategy

- BFS DFS

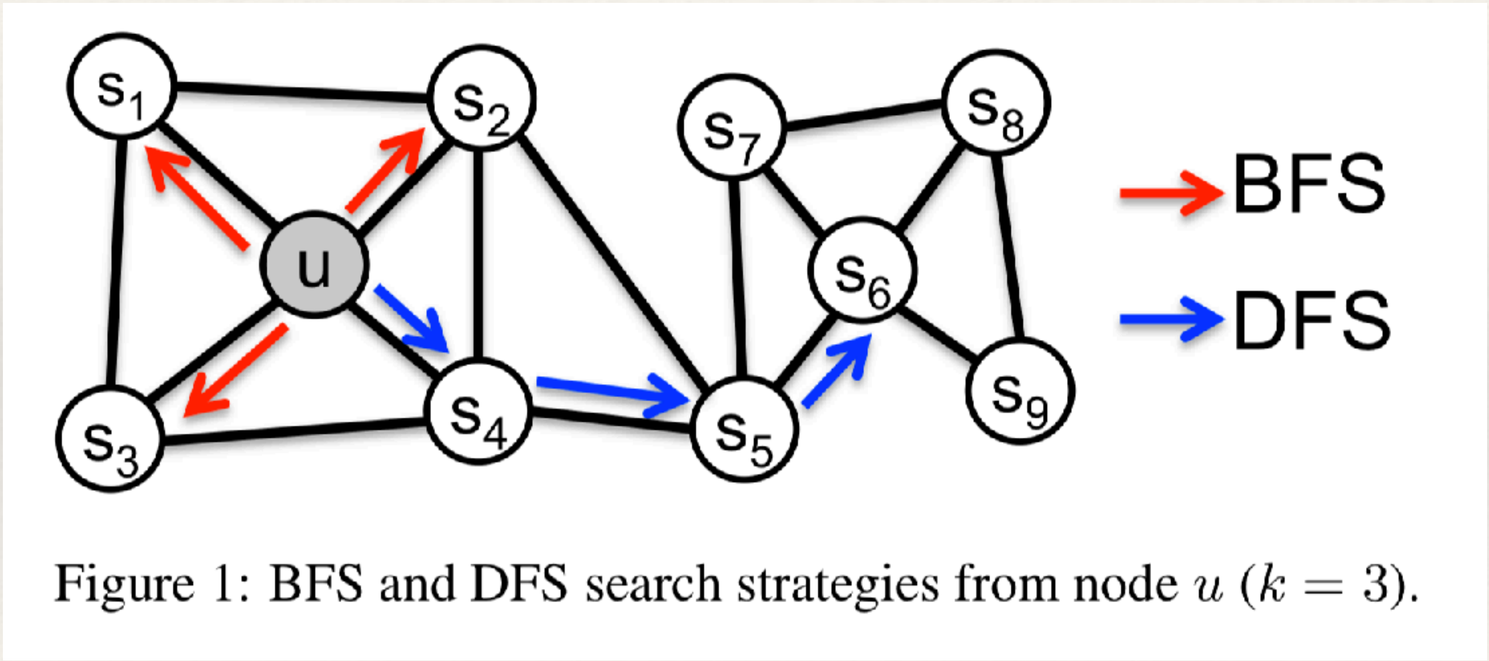


Figure 1: BFS and DFS search strategies from node u ($k = 3$).

Random walk

- Formally, given a source node u, we simulate a random walk of fixed length l. Let c_i denote the ith node in the walk, starting with c0=u. Nodes ci are generated by the following distribution:

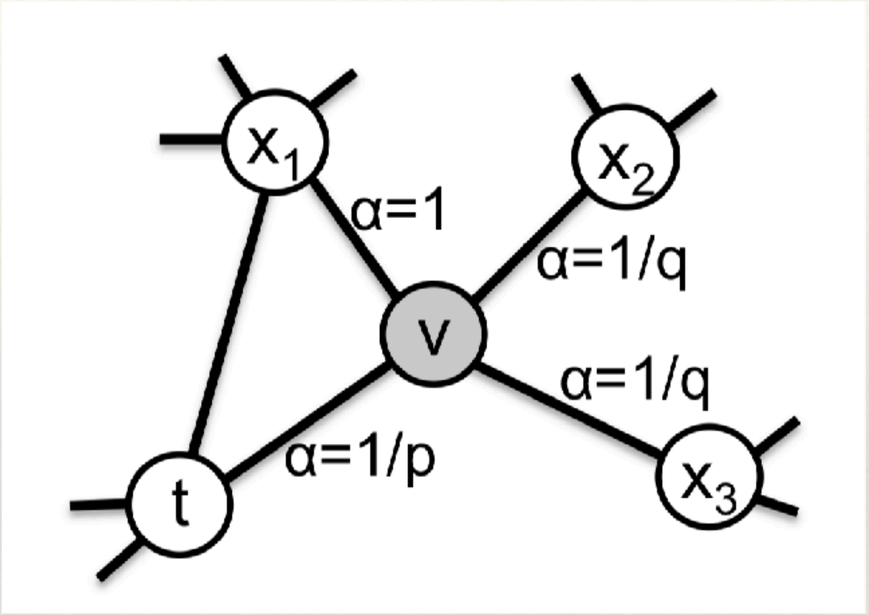
$$P(c_i = x \mid c_{i-1} = v) = \begin{cases} \frac{\pi_{vx}}{Z} & \text{if } (v, x) \in E \\ 0 & \text{otherwise} \end{cases}$$

- where π_{vx} is the unnormalized transition probability between nodes v and x, and Z is the normalizing constant.

Random walk

- $\pi_{vx}=\alpha_{pq}(t,x)\cdot w_{vx}$

$$\alpha_{pq}(t,x) = \begin{cases} \frac{1}{p} & \text{if } d_{tx} = 0 \\ 1 & \text{if } d_{tx} = 1 \\ \frac{1}{q} & \text{if } d_{tx} = 2 \end{cases}$$



Algorithm

Algorithm 1 The *node2vec* algorithm.

LearnFeatures (Graph $G = (V, E, W)$, Dimensions d, Walks per node r, Walk length l, Context size k, Return p, In-out q)
  π = PreprocessModifiedWeights(G, p, q)
  $G' = (V, E, \pi)$
  Initialize $walks$ to Empty
  for $iter = 1$ **to** r **do**
    for all nodes $u \in V$ **do**
      $walk$ = node2vecWalk(G', u, l)
      Append $walk$ to $walks$
  f = StochasticGradientDescent(k, d, $walks$)
  return f

node2vecWalk (Graph $G' = (V, E, \pi)$, Start node u, Length l)
  Inititalize $walk$ to $[u]$
  for $walk\_iter = 1$ **to** l **do**
    $curr = walk[-1]$
    V_{curr} = GetNeighbors($curr$, G')
    s = AliasSample(V_{curr}, π)
    Append s to $walk$
  return $walk$