Applications of Random walk

by wzylucky

2016-2-27
Applications of random walk

- Influence Maximization
- Random-walk domination
- PageRank
- Similarity Search
- Link-Prediction
Influence Maximization and Random Walk

• Definition of Influence Maximization
  Given a graph \( G \) and a constant \( K \). Try to find \( K \) nodes to influence the most number of nodes under predefined model.

• Formulation of Influence Maximization
  let \( I(S) \) be the expected spread of a node set \( S \) and \( h(j,S) \) be the influence from set \( S \) to \( j \). then:
  \[
  I(S) = \sum_{j \in V} h(j, S), \quad h(j, S) = \begin{cases} \sum_{i \in V} c p_{ji} h(i, S), & j \notin S, \\ 1, & j \in S, \end{cases}
  \]
  where \( p_{ji} \) is one-step transition probability and \( c \) is decay parameter. then continue selecting seed nodes by using the follow expression:
  \[
  \mu = \arg \max_{\mu \in (V-S)} I(S \cup \{\mu\}) - I(S)
  \]
Influence Maximization and Random Walk

• How to solve it with random walk?

  ➢ **Parallel Computation**

    run random walk \( R \) times for each node. And when performing \( R \) random walks from a particular node \( j \), measure the contribution of \( j \) to the marginal increment of every node. So that need only \( O(NR) \) random walks to derive the marginal increment of all nodes.

  ➢ **Walk Reuse**

    Record the total \( O(NR) \) random walks in memory, and apply the updates accordingly after one node is added into the result set.
Influence Maximization and Random Walk

• What information need to be stored?

1) To support walk reuse, the algorithm has to store each random walk. During this step, the space cost is \(O(NRL)\).

2) To support fast updates, maintain an indexed array of object is necessary. And this step of the space overhead is about \(O(NRL)\).

3) In the calculation process, it also needs to store two vectors: \(score[n]\) and \(P[n]\).
Random-walk domination

• Definition of Random-walk domination

Given a graph $G$ and a targeted set, if a node hitting any one targeted node can be regarded as that the targeted nodes dominate such a node by an $L$-length random walk.

• Two types of this problems

1. Given a graph $G$, how to target $K$ nodes such that the targeted nodes can be easily reached by the remaining nodes though an $L$-length random walk?

2. Given a graph $G$, how to find $K$ nodes so as to maximize the expected number of nodes that hit any one targeted node by the $L$-length random walk?
Random-walk domination

- Formulation of the two types problems

  let $T_{uS}^L$ denote the number of hops of that the $L$-length random walk starting at $u$ hits any one node in $S$ for the first time, then:

  $$T_{uS}^L \triangleq \min\{\min\{t : Z_u^t \in S, t \geq 0\}, L\}.$$ 

  let $h_{uS}^L \triangleq \mathbb{E}[T_{uS}^L]$ and we can get: 
  $$h_{uS}^L = \begin{cases} 0, u \in S \\ 1 + \sum_{w \in V \setminus S} p_{uw} h_{ws}^{L-1}, u \notin S \end{cases}.$$

  so the first problem is formulated as:

  $$\min_{|S| \leq k} \sum_{u \in V \setminus S} h_{uS}^L$$

  is equal to

  $$\max_{|S| \leq k} nL - \sum_{u \in V \setminus S} h_{uS}^L$$

  Similarly, the second problem can be formulated as:

  $$\max_{|S| \leq k} \mathbb{E}\left[ \sum_{u \in V} X_{uS}^L \right]$$

  where the $X_{uS}^L$ denote number of nodes which reached set $S$ in $L$-length random walk.
Random-walk domination

- How to solve the problems with random walk?
  - First, for each node, the algorithm independently runs $R$ $L$-length random walks.
  - Then, the algorithm materializes such samples and applies them to estimate the marginal gain $\sigma_u(S)$ for any given node $u$ and a given set $S$.

- What information needs to be stored?
  1) Holding $D[1:R][1:n]$ to estimator the hitting time $h_{us}$ based on the $i$-th $L$-length random walk.
  2) Holding $I[1:R][1:n]$ to index all the nodes that hit $v$ by the $i$-th $L$-length random walk.

Here, $\sigma_u(S) = \sum_{\omega \in V \setminus S_u} (h_{\omega S} - h_{\omega S_u}) + h_{us}$
PageRank and Random Walk

- **Formulation of PageRank**
  
  - if node $j$ point to node $i$, then $i$ get $\frac{r_j}{d_j}$ score from $j$. 
  
  Here $d_j$ is the out-degree of $j$.

  \[ r_i = \sum_{j \rightarrow i} \frac{r_j}{d_j} \Rightarrow r = r \cdot D^{-1}A = r \cdot P. \]

  Here $D = \text{diag}(d_1, d_2, \ldots, d_n)$. $A$ is the adjacent matrix of out-link.
PageRank and Random Walk

• how to solve it by using random walk?
  ➢ Run a random walk, at each node $i$, jump to one of its out-neighbors randomly with probability $(1-\alpha)$, or jump to any other nodes in the graph with probability $\alpha$. Then the stationary distribution of the random walk is $r$.
  ➢ The random walk converges if and only if the corresponding Markov chain is irreducible and aperiodic.

  **Note:** if $\alpha$ is large, the random walk converges to stationary distribution fast.

• what information needs to be stored?

  need One-dimensional array Score[1:n] to save score of each node.
Similarity Search and Random Walk

• Definition of Similarity Search
  Given a query vertex $u$, find top-$k$ vertices $v$ with the $k$ highest Sim-Rank scores $s(u, v)$ with respect to $u$.
  
  **Note:** Sim-Rank is a kind of general similarity measure, which is based on a graph-theoretic model that says “two objects are similar if they are related to similar objects.”

• how to solve it by using random walk?
  
  This question is divided into two steps:
  ① Preprocess phase
  ② Query phase
Similarity Search and Random Walk

- **Preprocess phase**
  - In order to precompute L1 bound and L2 bound, it needs to run R times T-length random walks with two times.
  - In order to enumerate “candidates” of highly similar vertices u, it needs to run NPQ times T-length random walks.

- **Query phase**
  First, Prune S by L1 and L2 bound; then, running a small number of T-length random walk to roughly estimate Sim-Rank scores for each candidate v; last, giving an accurate re-compute for these candidate which have a high scores.
Similarity Search and Random walk

- what information need to be stored?
  The random walk path should be retained.
link-prediction and random walk

• Definition of link-prediction?

  Given a snapshot of a social network at time $t$, we seek to accurately predict the edges that will be added to the network during the interval from time $t$ to a given future time $t'$. 

• How to solve it by using random walk?

  —Combine PageRank with supervised learning:

    ◆ PageRank is great to capture important nodes based on the structure of social network.
    ◆ Supervised learning is great at learning parameters with features
link-prediction and random walk

• How to combine PageRank with Supervised learning?

➢ Using Supervised learning to set edge strength.
  • Split the remain nodes into positive nodes and negative nodes.
  • Learn an edge strength function $f_\theta(x, y) = \exp(\theta^t, \varphi(x, y))$
    $-\varphi(x, y)$ is the feature vector of edge $e_{xy}$
    $-\theta^t$ is the parameter vector we want to learn.
  • Learn $\theta^t$ from the training date by the formula:
    $$\arg\min \sum_{p \in P} \sum_{n \in N} \delta(r_p < r_n) + \beta \|\theta\|^2$$
link-prediction and random walk

- Run PageRank on the weighted graph and assign an important score to each node.
- Recommend top-\(k\) nodes.

What information need to be stored?
- need one-dimensional array score[1:n] to store the score of each node.
Reference

- Pengpeng Zhao et al. “Impact of Online Activities on Influence Maximization: A Random Walk Approach”.
- Rong-Hua Li et al. “Random-walk Domination in Large Graphs”. *ICDE’14*
- Mitsuru Kusumoto et al. “Scalable Similarity Search for SimRank”. *SIGMOD’14*
- Glen Jeh et al. “SimRank: A Measure of Structural-Context Similarity”. *SIGKDD’02*
- Lars Backstrom et al. “Supervised Random Walks: Predicting and Recommending Links in Social Networks”. *WSDM’12*
- Xiaowei CHEN. PPT “Random Walk on Graphs: Theory and Applications”