


# Chapter 3

## Design of SensorRank

In Chapter 3.1, the formulation of SensorRank is presented. The analysis of SensorRank is conducted in Chapter 3.2.

### 3.1 Formulate SensorRank



Given a correlation network  $G = (V, E)$  derived above, we should assign each sensor an appropriate SensorRank to reflect the importance of sensor. The value of SensorRank for each sensor is the degree to measure how many and how similar neighboring sensors the sensor has. SensorRank should consider not only one-hop neighbors but also  $k$ -hop neighbors when determining SensorRank value. Clearly, if one sensor and its many neighboring sensors are similar, this sensor will have higher SensorRank. On the other hand, if this sensor only has a few similar neighboring sensors, this sensor will have lower SensorRank. In a correlation network, the similarity of one-hop neighbors has already been defined on the edges. In fact, the similarity of  $k$ -hop neighbors is also shown in the correlation network indirectly. For example,  $s_j$  and  $s_k$  are an one-hop neighbor and a two-hop neighbor of  $s_i$  respectively, and  $s_j$  and  $s_k$  are one-hop neighbors. Assume that  $s_i$  and  $s_j$  as well as  $s_j$  and  $s_k$  are very similar. The similarity of  $s_i$  and  $s_k$  are also similar as well. Therefore, if a sensor has higher SensorRank, this sensor

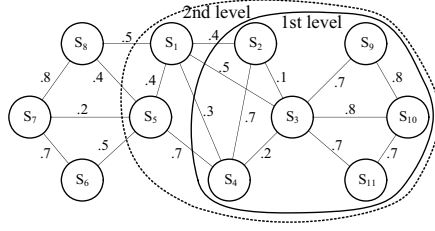


Figure 3.1: An example of SensorRank

should have many similar neighbors and neighbors' neighbors. In order to measure the value of SensorRank, we could imagine the random walks on the correlation network. Intuitively, one could randomly select a sensor. Since this sensor has some edges that link to other sensors, one should follow the edge with a higher value of similarity. As such, we could formulate the correlation network as a Markov chain, where each sensor  $s_i$  is viewed as the state  $i$ , and the transition probability of from state  $i$  (i.e., sensor  $s_i$ ) to state  $j$  (i.e., sensor  $s_j$ ) is defined as

$$p_{i,j} = \frac{tr_{i,j}}{\sum_{k \in nei(i)} tr_{i,k}}$$

SensorRank of  $s_i$  is denoted as  $rank_i$  which is referred the probability that one randomly walks to  $s_i$ . Thus, SensorRank of  $s_i$  is calculated as follows:

$$rank_i = \sum_{s_j \in nei(i)} rank_j \cdot p_{j,i}. \quad (3.1)$$

Due to the characteristic of random walks, SensorRank is a probability distribution to represent the likelihood that a person could randomly follow the links to any particular sensor. The computation of SensorRank requires several iterations to determine the steady state probability for each sensor. Specifically,  $rank_i^{(k)}$  is the value of SensorRank at the  $k$ -th iterations. In the beginning, the initial  $rank_i^{(0)}$  is set to 1. Note that  $rank_i^{(0)}$  can be set to any constant  $a$ , and the results will be  $a$  times the value generated when the initial SensorRank is set to 1. In the first round, each sensor  $s_i$  updates its SensorRank as  $rank_i^{(1)}$  using the initial SensorRanks of its neighbors (referred to as the first level neighbors). Now each sensor has considered the first level neighbors to calculate its SensorRank. In the second round, each sensor can obtain some information from the second level neighbors (the neighbors of

the first level neighbors) through its first level neighbors since its first level neighbors have explored their first level neighbors as well. Therefore, after the  $k$ -th round, sensor  $s_i$  has explored the  $k$ -th level neighbors and updated SensorRank as  $rank_i^{(k)}$ . Consider an example in Figure 3.1. In the first round,  $s_3$  has some similarity information from its first level neighbors  $\{s_2, s_4, s_9, s_{10}, s_{11}\}$ . Similarly, both  $s_2$  and  $s_4$  could exchange some information with their neighbors. In the second round,  $s_3$  can obtain similarity information from the second level neighbors  $\{s_1, s_5\}$  since its first level neighbors  $s_2$  and  $s_4$  have explored  $s_1$  and  $s_5$  during the first round. If  $k$  is larger, SensorRanks will be more accurate since every sensor can explore more neighbors. In sensor networks, the computation cost will be larger when the number of iterations is larger. Therefore, we can limit  $k$  to the upper bound  $\delta$ .

---

**Algorithm** *SensorRank*

**Input:** a sensor  $s_i$ , and a threshold  $\delta$ .

**Output:**  $rank_i$  for  $s_i$ .

- 1:  $rank_i^{(0)} = 1$
- 2: **for**  $k = 1$  to  $\delta$  **do**
- 3:   **for all**  $s_j \in nei(s_i)$  **do**
- 4:      $p_{i,j} = \frac{tr_{i,j}}{\sum_{s_k \in nei(i)} tr_{i,k}}$
- 5:     send  $rank_i^{(k-1)} \cdot p_{i,j}$  to  $s_j$
- 6:   receive all  $rank_j^{(k-1)} \cdot p_{j,i}$  from every  $s_j \in nei(i)$
- 7:    $rank_i^{(k)} = \sum_{s_j \in nei(i)} rank_j^{(k-1)} \cdot p_{j,i}$




---

Given a correlation network shown in Figure 3.1, we now demonstrate how to calculate SensorRank. Initially, sensor  $s_i$  sets its SensorRank  $rank_i^{(0)}$  to 1. For sensor  $s_i$ ,  $s_i$  calculates the transition probability  $p_{i,j}$  to the corresponding neighbor  $s_j$  and sends  $rank_i^{(0)} \cdot p_{i,j}$  to  $s_j$ . For example,  $s_3$  sends  $rank_3^{(0)} \cdot p_{3,1} = 1 \cdot \frac{0.5}{3.4} = 0.147$  to  $s_1$ , 0.118 to  $s_2$ , 0.088 to  $s_4$ , and etc. At the same time,  $s_3$  receives SensorRanks from its neighbors. For example,  $s_3$  receives  $rank_2^{(0)} \cdot p_{3,2} = 1 \cdot \frac{0.4}{0.3+0.4} = 0.571$  from  $s_2$ . Upon receiving all the proportion of SensorRank

	$s_1$	$s_2$	$s_3$	$s_4$	$s_5$	$s_6$	$s_7$	$s_8$	$s_9$	$s_{10}$	$s_{11}$
$k = 0$	1	1	1	1	1	1	1	1	1	1	1
$k = 1$	1.55	0.24	2.34	0.42	1.99	0.67	0.94	0.51	0.55	1.27	0.51
$k = 2$	1.25	0.47	1.52	0.79	1.54	0.9	0.89	0.75	0.92	1.1	0.87

Table 3.1: SensorRank values for sensors in Figure 2.

from the neighbors,  $s_3$  can update its SensorRank to  $rank_3^{(1)}$ .

$$\begin{aligned}
rank_3^{(1)} &= \sum_{i \in \{1,2,4,9,10,11\}} rank_i^{(0)} \cdot p_{j,i} \\
&= 1 \cdot p_{1,3} + 1 \cdot p_{2,3} + 1 \cdot p_{9,3} + 1 \cdot p_{10,3} + 1 \cdot p_{11,3} \\
&= \frac{0.5}{2.4} + \frac{0.4}{0.7} + \frac{0.3}{1.2} + \frac{0.7}{1.5} + \frac{0.8}{2.3} + \frac{0.7}{1.4} \\
&= 2.34
\end{aligned}$$

After the first round,  $\{rank_i^{(1)} | i = 1, 2, 3, 4\} = \{1.55, 0.24, 2.34, 0.42\}$ . In the second round, sensors calculate the values of SensorRank with the updated values of SensorRank in the first round. For example,  $s_1$  now sends  $rank_1^{(1)} \cdot p_{1,3} = 1.552 \cdot \frac{0.5}{2.4} = 0.323$  to  $s_3$ . Similarly, when  $s_3$  receives all the values from its neighbors,  $s_3$  can update its SensorRank to  $rank_3^{(2)}$ . Assume that  $\delta = 2$ ,  $s_1$  will stop updating its SensorRank, and  $\{rank_i^{(2)} | i = 1, 2, 3, 4\} = \{1.25, 0.47, 1.52, 0.79\}$ . As expected,  $s_3$  has the highest SensorRank 1.521, since  $s_3$  has many similar neighbors. Because  $s_1$  has fewer similar neighbors than  $s_3$ ,  $s_1$  has fewer SensorRank than  $s_3$ . Note that  $s_2$  has the fewest SensorRank than other sensors since it has few neighbors and those neighbors are not similar to it. The values of SensorRank after the 3rd iteration are listed in Table 3.1. From Table 3.1, both  $s_3$  and  $s_5$  have higher SensorRank values since these two sensors have more similar neighbors than other sensors. This agrees with our requirement that the important sensors are those sensors who have more similar neighbors.

In the most situations, SensorRank of every sensor will be stable in a few iterations. That is, SensorRank will come to a steady state. But in some situations, it will not be stable. For example shown in Figure 3.2,  $s_1$  links to  $s_2$  and  $s_2$  links to  $s_3$ . In this case, we can find

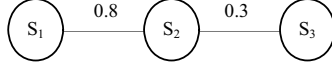


Figure 3.2: In this case, SensorRank of each sensor will not be converge.

that  $\{rank_i^{(k)} | i = 1, 2, 3\} = \{0.727, 2, 0.273\}$  and  $\{rank_i^{(2k+1)} | i = 1, 2, 3\} = \{1.455, 1, 0.545\}$  for  $k = 1, 2, \dots$ . Therefore,  $s_1$ 's SensorRank sometimes is larger than  $s_2$ 's but sometimes smaller than  $s_2$ 's SensorRank. In order to solve this problem, a simple way is that an extra edge linked from a sensor to itself is added for every sensor. The transition probability  $p_{i,i}$  of such link for each sensor  $s_i$  is assigned a very small value  $\varepsilon$ , and the transition probability  $p_{i,j}$  to every neighbor  $s_j$  will minus  $\frac{\varepsilon}{nei(i)}$ . In this way, the calculation of SensorRank will be affected slightly and the above problem can be solved easily. The proof will be discussed in the following chapter.

## 3.2 Analysis of SensorRank

Since the correlation network is modeled as a Markov chain, in this chapter we prove that the steady state probability of each sensor is possible, which means that SensorRank has the convergence property. To facilitate to formulate our network, we define the following terms:

**Definition 4.** Let  $P = \langle s_i = s_{i,1}, s_{i,2}, \dots, s_{i,k} = s_j \rangle$  be a path from  $s_i$  to  $s_j$ . Extended transition probability  $\tilde{p}_{i,j}(P)$  is defined as  $\prod_{i=1}^{k-1} p_{i,i+1}$ .

**Definition 5.** Tough path  $\hat{P}_{i,j}$  from  $s_i$  to  $s_j$  is defined as  $min_P \{\tilde{p}_{i,j}(P)\}$ .

The value of  $\tilde{p}_{i,j}(P)$  can be explained as a ranking process from  $s_i$  to  $s_j$ . The ranking process can be completed by only neighbors, but SensorRank of a sensor will effect another sensor far from it indirectly since the network is connected <sup>1</sup>. From the definition of  $\tilde{p}_{i,j}(P)$ , since  $p_{i,j} < 1$ , the effect is inversely proportional to the length of  $P$  from  $s_i$  to  $s_j$ . That

<sup>1</sup>The correlation network will be disconnected in the situation that some nodes with their trust relation to be 0. However, in an enough densed sensor network, due to the correlation property, the disconnected sensors can be recognized as faulty sensors because they are not similar to any their neighbors at all.

explains that the effect of ranking from nearby sensors is higher than far-away sensors. Since there are many path from  $s_i$  to  $s_j$ , the tough path can be explained as a path that  $s_i$  can effect the rank of  $s_j$  least among all paths.

The correlation network can be modelled as a Markov chain  $M$  with a sensor  $s_i$  as a state  $i$  and  $\tilde{p}_{i,j}$  as the transition probability from state  $i$  to state  $j$ . The corresponding transition matrix is  $P_{TR}$ .

We first prove the properties of  $M$ :

**Property 1.**  $M$  is ergodic.

**proof:**

Given a correlation network  $G = (V, E)$  and for any  $s_i \in V$ , we can conclude that  $s_i$  is a neighbor of itself since  $tr_{i,i} = 1$  and thus  $p_{i,i} = \frac{tr_{i,j}}{\sum_{k \in nei(i)} tr_{i,k}} = \frac{1}{\sum_{k \in nei(i)} tr_{i,k}}$ . Therefore,  $M$  is aperiodic. Moreover, since  $M$  is finite and reachable of any two states, we can conclude that  $M$  is ergodic.

Since ranks of sensors are relative to all sensors in the whole network, it is an important issue whether sensors' ranks will be stable or not. If there are sensors with never stable ranks, the system will become unreliable since we cannot decide how important these unstable sensors are. Fortunately, since  $M$  is ergodic, then the  $M$  has unique stationary distribution. That is, the ranks of all sensors will be stable. The coming question is when the ranks of sensors are stable. The question is equivalent to bound the mixing time of  $M$ . We apply the coupling technique for bounding the mixing time. The concept of coupling is to construct The principal idea of coupling is to introduce another another Markov chain  $M'$  that is stationary, being its stationary probabilities, has the same transition matrix as  $M$  and independent of  $M$ . Clearly, the distribution of  $M$  tends to be closed to  $M'$ . If the distribution of  $M$  and  $M'$  is closed to  $\epsilon$  after  $T$ , then we can estimate the rate of convergence in the limiting relation mentioned above. Interested readers can read [19] for further information.

Let the diameter of the network is  $diam(G)$ . Consider the Markov chain  $M^{diam(G)}$  given

by the transition matrix  $(P_{TR})^{diam(G)}$ . Suppose the mixing time of  $M$  (denote by  $\tau(\epsilon)$ ) is smaller than  $T$ , we can know that  $M^{diam(G)}$  cannot converge faster than  $k = \left\lfloor \frac{T}{diam(G)} \right\rfloor$  steps. Then, we design a coupling where two copies of the chain both move to state  $j$  together with probability at least  $m_j$  in every step, where  $m_j$  is the smallest element in the  $j$ -th column of  $(P_{TR})^{diam(G)}$ . The two chains can be made couple with probability at least  $\sum_j m_j$ . Hence,  $Pr(\text{Not Coupled after } k \text{ steps}) \leq (1 - m)^k$ . Note that  $m_j$  is the smallest element in the  $j$  column of  $(P_{TR})^{diam(G)}$ , then in one step of  $M^{diam(G)}$  reaches state  $j$  with probability at least  $m_j$  from every state. Therefore, we can get  $m_j$  is  $\tilde{p}_{i,j}(\hat{P})$  where  $|\hat{P}| = diam(G)$ .

From above,  $Pr(\text{Not coupled after } k \text{ steps}) \leq (1 - m)^k \leq \epsilon$  can imply that  $\tau(\epsilon) \leq \{diam(G) \ln \epsilon\} / \ln \left(1 - \sum_j \tilde{p}_{i,j}(\hat{P})\right)$  with  $|\hat{P}| = diam(G)$ . That represents that the convergence speed is effected by the size of network and the tough path. Since tough path from  $s_i$  to  $s_j$  is a merit how  $s_i$  can effect the rank of  $s_j$ , we can conclude that ranks can converge rapidly with little proportion between faulty and normal sensors in the network.

