Performance Evaluation of Rating Aggregation Algorithms in Reputation Systems

Zhengqiang Liang and Weisong Shi
Wayne State University
{*sean,weisong*}*@wayne.edu*

Abstract

Ratings (also known as recommendations, referrals, and feedbacks) provide an efficient and effective way to build trust relationship amongst peers in open environments. The key to the success of ratings is the rating aggregation algorithm. Several rating aggregation algorithms have been proposed, however, all of them are evaluated independently so that it is difficult to compare the effect of these schemes. In this paper, we argue that what is missing is to evaluate different aggregation schemes in the same context. We first classify all state-of-the-art aggregating algorithms into five categories, and then comprehensively evaluate them in the context of a general decentralized trust inference model with respect to their resistance to different factors, such as dynamic behavior of peers and raters, dishonest ratings, and so on. The simulation results show that complicated algorithms are not always a good choice if we take the implementation cost and resistance to bad raters into consideration.

1 Introduction

Rating is a very powerful vehicle to find wanted resources and information in our human society. Hinted by the human social network, computer researchers are of great interests to employ ratings in the computing society [13], especially in the field of trust inference, where ratings are crucial to build the trust relationship. The key to the success of ratings is the rating aggregation algorithm, i.e., how to integrate the ratings from other into one peer's own trust view, including getting rid of bad raters and obtaining accurate ratings. In the rest of this paper, we will use the notion of *rater* to represent the peer who assigns her ratings for others.

Many interesting aggregating algorithms have been proposed. Basically, these algorithms can be classified into five categories. Though noting the potential advantages of ratings, Resnick *et al.* [14] challenge the feasibility of the distribution of feedbacks, from the point of the expensive cost for the feedback distribution. Holding the same view, Liang and Shi [8, 9, 10] suggest to treat the ratings from different raters equally considering the dynamics of P2P systems. They argue that simply averaging ratings is deserved considering the simplicity of the algorithm design, and the low cost in the system running. We call this approach *average* (or simply *Avg*). Different with [8, 9, 10], Wang *et al.* [18, 19] suggest that, averaging should be applied only for stranger raters, but for acquaintances, their ratings should be weighted. We call this approach *half weighted* (or simply *Half*) method in following sections. Yu *et al.* [16, 24, 25, 26] give another thought on this issue. They believe that only ratings from witnesses, who have interacted with the ratee (We call the peer which is recommended by raters as ratee), are useful. In their weighted majority algorithm (denoted as *WMA*), only the ratings from witnesses are aggregated, and the weight of witnesses is decreased if the rating is different from self own recognition. Different from *WMA*, Sriatsa *et al.* [17] argue that, the weight of ratings should be based on the similarity of the experience between the rater and the peer itself. We denote this approach as *personalized similarity measure (PSM)*. Finally, Jøsang *et al.* propose to aggregate the ratings and to update the weight of raters through deriving the expectation of the *Beta* distribution [2, 4, 5, 20]. The details of these algorithms are listed in Section 3.

It is natural to ask, which algorithm is the best? However, to our best knowledge, systematically evaluating different aggregation algorithms is still missing. Thus we take the first step to answer this question. We first abstract a basic decentralized trust inference model which integrates the rating into the trustworthiness value calculation. In this model, each peer will assign a trustworthiness value to a set of peers of interest. The trustworthiness value is derived from the interaction-derived information (first-hand information) and rating (second-hand information). Based on this model, we build a simulation platform using Netlogo, a popular multiple-agent simulation tool in the AI community [21], which allows us to tune and adjust all design parameters in a flexible way. We evaluate the effects of ratings aggregation from multiple angles, including the number of good services, the communication workload, the storage cost, and the running time. The simulation results show that, simple algorithms are considerable with their fairly good performance and less overload; in some cases the simple algorithm can even outperform the complicated algorithms.

The rest of the paper is organized as follows. Section 2 abstracts a basic decentralized trust inference model. Section 3 gives the details about the five state-of-the-art aggregating algorithms we will evaluate. The extensive comparison of these aggregation algorithms is depicted in Section 4. Related work and discussions are described in the following section. Finally, we conclude in Section 6.

2 The Basic Trust Model

To better compare different aggregating algorithms of ratings on the trust inference, we first abstract a generic decentralized trust model in open environments. To reduce the complexity of modeling, we make the following assumptions: (a) The quality of peers as service providers (*SP*s) and as raters is independent; (b) Each peer has a relative unique and stable ID. This will make reputation and trustworthiness make sense; and (c) Each peer holds a limited number of neighbors (six in our simulation).

Two types of basic information, interaction-derived information (or first-hand information) and rating (or second-hand information), are necessary to build a trust model. In order to make our evaluation general enough, we will employ a simple model in which the trustworthiness T is derived from two kinds of information mentioned above. We define $T = W \times I + (1-W) \times R$, where I is the value of the interaction-derived information, R is the value of rating, and W is the weight of I (correspondingly, $1-W$ is the weight of R). In our simulation, the value of W is fixed. But we will analyze two scenarios where W is high (0.8) and low (0.2) in the analysis. For a neighbor k, we calculate its first-hand information I as suggested in [26]. That is, I $= \sum_{i=1}^{h} s_i/h$, where h is the transaction number in a fixed interval, and s_i is the opinion about the transaction i with neighbor k. The update of R is implemented by different aggregation algorithms. Five algorithms corresponding to previous introduction are implemented. The details about these algorithms are listed in Section 3. For algorithms *Avg, Half,* and *Beta*, the possible raters can only be chosen from the neighbor list, i.e., only the peers with which the requester have interacted before can become a candidate. However, with the same limitation, the algorithms *PSM* and *WMA* will be restricted so that they can not show their special advantages. So for these two algorithms, we enlarge the rater candidate set to the two-hop neighbors, that is, neighbor's neighbor. Because of this, we can see that these two algorithms consume more storage in the analysis. In our evaluation, we try our best to keep our implementations in the original flavor. However, since these algorithms are just part of the approach from the original proposer, and we have to integrate all of them into the same platform with a little adjustment as listed in Section 3.

All values of T, R and I are taken from range [0,1]. The model is an independent model, that is, each peer has its own trustworthiness values about other peers. Since in every trust model, ratings and interaction-derived information are necessary and enough to derive the trustworthiness value, our evaluation should be meaningful for all kinds of current approaches.

3 Aggregating Rating Algorithms

In the following we briefly describe the five algorithms evaluated in this paper. These five algorithms are the typical aggregating algorithms in the distributed trust inference where every peer holds personalized views on their potential raters. The key role of these algorithms is getting rid of bad raters and obtaining accurate ratings. Note that all these algorithms are proposed in the context of trust-independent assumption, where the the quality for a peer to be a *SP* is independent of the quality for a peer to be a rater. Some classical algorithms such as EigenTrust [6] are not included here because it is not able to be applied in such a context.

- **Average (Avg)** [8, 10]. In *Avg*, the rate aggregation R is defined as: $r_{ij} = \frac{\sum_{z=1}^{g} tr_{zj}}{g}$, where r_{ij} is the value of aggregated ratings towards peer j in peer i. tr_{zj} is z's rating towards j. g is the total number of ratings towards j. For the honest rater, tr_{zj} is the trustworthiness value of peer j in peer z, but for the bad rater, this value can be any value depending on what behavior pattern the rater acts with. Since all the weight of the rating are equal (to 1), there is no need to update the weight, thus there is no corresponding communication and storage cost for the weight update. This is the simplest algorithm among the five algorithms.

- **Half Weighted (Half)** [18, 19]. In HALF, the rating aggregation is defined as: $r_{ij} = w_t * \frac{\sum_{l=1}^{k} t_{il} * tr_{lj}}{\sum_{l=1}^{k} tr_{il}} + w_s * \frac{\sum_{z=1}^{g} tr_{zj}}{g}$, where w_t is the weight about the rating from the acquaintance, w_s is the weight about the rating from the stranger, and t_{il} is the weight (trustworthiness) about the rating from rater l. r_{ij}

and tr_{lj} are the same as *Avg*. For the weight update: $t_{il} = \alpha * t^o_{il} + (1-\alpha) * e_\alpha$, where t_{il} denotes the new trust value that the i^{th} peer has towards rater l; t^o_{il} denotes the old trust value. α is the learning rate–a real number in the interval [0,1]. e_α is the new evidence value. In the original paper [18], e_α can be -1 or 1. Authors suggest that, if the rating and the actual experience has considerable difference, e_α will be set to -1, otherwise, 1. However, if the value of α is large, it will make the value of t_{il} fluctuate very fast. So in the simulation, we change this formula a little bit with: $t_{il} = \alpha * t^o_{il} + (1-\alpha) * |s^j - t^o_{lj}|$ where, s^j is i's self-opinion towards transaction j, and α is set to be 0.51.

- **Personalized Similarity Measure (PSM)** [17]. In PSM, the rating aggregation is: $r_{ij} = \sum_{l=1}^{k}(tr_{lj} * \frac{S(il)}{\sum_{n=1}^{k} S(in)})$, where similarity $S(in)$ equals to $1 - \sqrt{\frac{\sum_{r \in IJS(i,n)}(\frac{\sum_{v \in I(i,r)} tr_{iv}}{|I(i,r)|} - \frac{\sum_{v \in I(n,r)} tr_{nv}}{|I(n,r)|})^2}{|IJS(i,n)|}}$. $S(in)$ is the personalized similarity between peer i and rater n, which is similar to the weight of the rater in *Half*. tr_{lj} has the same definition as that in *Avg*. Let IJS(i, n) denote the set of common peers with whom both peer i and n have interacted, and I(i, r) denotes the collection of interactions between peer i and r. The similarity between peer i and n is computed based on the root mean square of the differences in their opinion over the nodes in IJS(i, n). Note that in our simulation, what we use is actually a slight variant of the original algorithm proposed in [17]. However, our basic ideas are the same, i.e., based on the personalized similarity towards the same interaction.

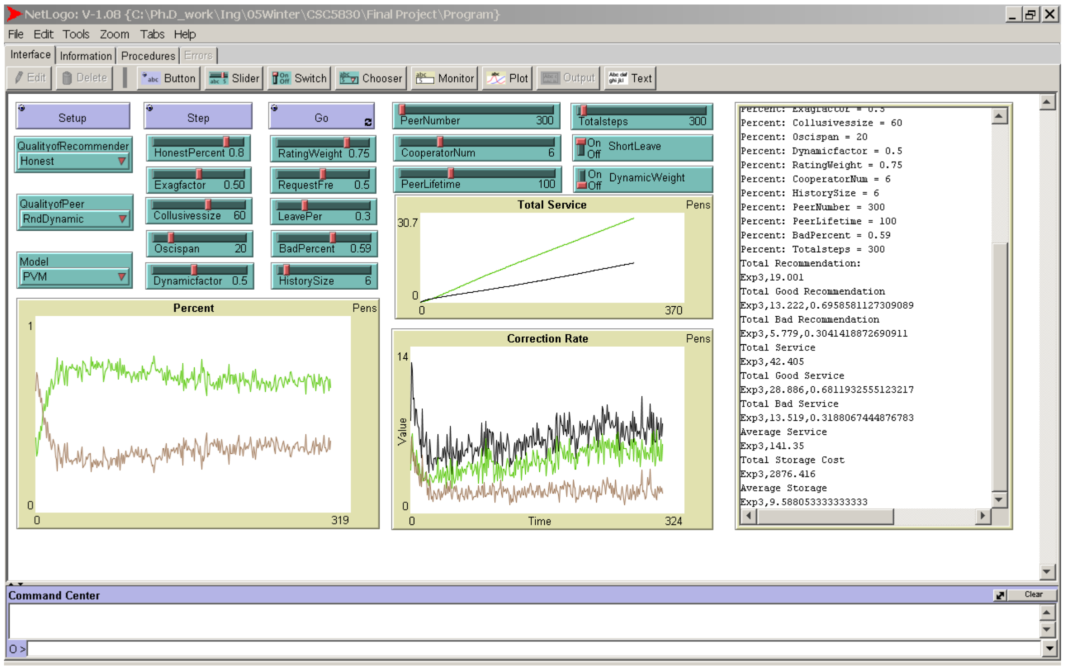
- **Weighted Majority Algorithm (WMA)** [26]. In WMA, the rating aggregation is defined as: $r_{ij} = \sum_{l=1}^{k}(lr_{lj} * \frac{t_{il}}{\sum_{n=1}^{k} t_{in}})$, where lr_{lj} is the firsthand information of peer l about j. For the weight update: $t_{in} = \theta t^o_{in}$, where t^o_{in} is the old weight of rater n in peer i. θ is defined as: $\theta = 1 - (1-\beta)|lr_{nj} - s|$.

- **Beta Model (Beta)** [2, 4, 5, 20]. In [5], Jøsang and Ismail propose the beta reputation system which use beta probability density functions to combine ratings and derive reputation towards the raters. In BETA, the rating aggregation is defined as: $r_{ij} = \frac{p+1}{p+n+2}$, where p is the number of positive ratings (> 0.5) towards peer j, and n is the number of negative ratings (< 0.5) towards peer j. Similar to *Avg*, in this algorithm we don't need to maintain the weight of raters. The raters in the upper and lower 10% will be excluded from the aggregation. This algorithm is also a simple algorithm without the weight update. Note that several variants have been proposed since they introduced the original idea; however, in our simulation, we just use a simple algorithm plus a simple filter to exclude the bad ratings.

4 Performance Evaluation

Now we are in a position to compare all state-of-the-art rating aggregation algorithms. Note that all these algorithms are proposed with the trust-independent assumption, that is the the quality for a peer to be a *SP* is independent of the quality for a peer to be a rater. The key role of these algorithms is to get rid of bad raters and obtaining accurate ratings. We build a simulation platform using NetLogo [21], a very popular multi-agent simulation tool in the AI community, which can easily model the parallel and independent agents, to simulate interactions among peers. With NetLogo we have developed a friendly GUI-based user interface to control the simulation, and we can easily tune different parameters to form different configurations. Figure 1 illustrates a snapshot of the simulation platform. With this platform, we can get different results for different parameter combinations.

As seen in Figure 1, there are many parameters involved in the platform design. Limited by the space, only the major parameters are described, as shown in Figure 2. We first define the type of qualities of both *SP*s and raters used in our evaluation. The behaviors of peers as raters B_R can be one of the three types: "malicious" (M), "exaggeration" (E), and "collusive" (C). Suppose s represents the real opinion of the rater. Malicious raters always give the complimentary opinion, that is, sending out $1 - s$ to others. The exaggerating raters will exaggerate their ratings by a exaggerating factor α, which is 0.5 in our simulation. For this type of raters, the rating $s + \alpha * (s - 0.5)$ will be sent out. For the final category, collusive raters will send out 1 for the peers in the collusive group, and 0 for the peers outside the group. In our simulation, the size of the collusive group is chosen as 20% (60 peers) of the total number of peers in the system. Three type of behavior patterns of *SP*s are studied: fixed (F), random (R), and oscillating (O). The type of "F" includes the fixed good and fixed bad. With the option F, *SP*s won't change their qualities once the simulation starts. With the option R, *SP*s will change their qualities randomly. While with the option O, *SP*s will change their qualities in a fixed oscillation span (20 steps in our simulation). Both the random and oscillating *SP*s are the dynamic *SP*s. The modes of aggregation are the five algorithms we present in Section 3, i.e., Avg (A), $Beta$ (B), $Half$ (H), PSM (P), and WMA (W). The percentage of the bad *SP*s (P_B) can be 20%, 40%, and 70% within the whole system. The percentage of the dynamic *SP*s (P_D) can be


NetLogo: V-1.08 {C:\Ph.D_work\Ing\05Winter\CSC5830\Final Project\Program}
File Edit Tools Zoom Tabs Help
Interface Information Procedures Errors
Edit Delete Button Slider Switch Chooser Monitor Plot Output Text
Setup
Step
Go
Total Service
Pens
30.7
0
0
370
Percent
Pens
1
0
0
319
Correction Rate
Pens
14
Value
0
0
Time
324
Command Center
Clear
O >

QualityofRecommender: Honest
QualityofPeer: RndDynamic
Model: PVM
HonestPercent 0.8
Exagfactor 0.50
Collusivessize 60
Oscispan 20
Dynamicfactor 0.5
RatingWeight 0.75
RequestFre 0.5
LeavePer 0.3
BadPercent 0.59
HistorySize 6
PeerNumber 300
CooperatorNum 6
PeerLifetime 100
Totalsteps 300
ShortLeave: On
DynamicWeight: Off

```
Percent: Exagfactor = 0.5
Percent: Collusivessize = 60
Percent: Oscispan = 20
Percent: Dynamicfactor = 0.5
Percent: RatingWeight = 0.75
Percent: CooperatorNum = 6
Percent: HistorySize = 6
Percent: PeerNumber = 300
Percent: PeerLifetime = 100
Percent: BadPercent = 0.59
Percent: Totalsteps = 300
Total Recommendation:
Exp3,19.001
Total Good Recommendation
Exp3,13.222,0.6958581127309089
Total Bad Recommendation
Exp3,5.779,0.3041418872690911
Total Service
Exp3,42.405
Total Good Service
Exp3,28.886,0.6811932555123217
Total Bad Service
Exp3,13.519,0.3188067444876783
Average Service
Exp3,141.35
Total Storage Cost
Exp3,2876.416
Average Storage
Exp3,9.588053333333333
```

Figure 1. A snapshot of the GUI interface of the simulation platform.

30% and 70%. For the percentage of honest raters, we make two choice available, 20% and 80%. Finally, the weight of rating can be 0.2 and 0.8.

	Description	*Possible Values*
B_R	Behavior of Bad raters	C, E, M
B_S	Behavior of Bad *SP*s	F, O, R
M	Mode of Aggregation	A, B, H, P, W
P_B	% of bad *SP*s	20%, 40%, 70%
P_D	% of dynamic *SP*s	30%, 70%
P_H	% of honest raters	20%, 80%
W	Weight of the rating	0.2, 0.8

Figure 2. Major parameters used in the simulation.

In the technical report version of this paper [9], we have proposed ten novel performance metrics for the evaluation of trust model and rating aggregation algorithms in a comprehensive way. However, we only select six most suitable performance metrics from the whole set because of the limit of the paper length, including the system workload ω, the accumulative correction rate ϕ_A, the total number of services η, the percentage of good service among all services η_g/η (η_g is the number of good services), the storage cost ζ, and the running time Υ, to conduct a complete comparison of the performance of different algorithms. Each figure includes a figure part and a table part. The figures are the interval plots on the data set corresponding to each peer. Two set of data are plotted. One is the number of good services η_g (the sub-figures on the left side), the other is the accumulative storage cost (the sub-figures on the right side, with unit of "10^3"). Each figure contains five to six vertical lines, which are corresponding to different algorithms. The random algorithm (denoted as *Rand*) is to simulate the case where no trust information is used to help to choose the *SP*s. So in *Rand*, peers randomly choose the *SP*s, and do not store the rating from raters. The value shown in the middle of the line is the corresponding mean. All plots are with 95% confident interval. In the table part, all the data corresponding to the metrics mentioned above are listed, and these data are from the view of the whole system. Note that since a large number of experiments with different parameters configurations have been conducted in our analysis, so we use experiment numbers to distinguish them. The configuration of parameters corresponding to each experiment is explained in the caption of the figure. The experiment numbers are listed in the x-axis of every figure beside the names of algorithms, and the first column in the table. For example, `Exp.#(6)` in Figure 4 means the *Beta* aggregation algorithm with the configuration that 20% peers are malicious raters.

Figure 3 shows the execution skeleton of service requesters and service providers in the simulation. To be worth noting that, peers only request raters to update their ratings once peers get one bad service. Besides getting one bad service, if since last update time **t** has passed, peers will update the weight of raters. In the simulation, the program runs 2,000 steps and then stops, and there are totally 300 peers are simulated.

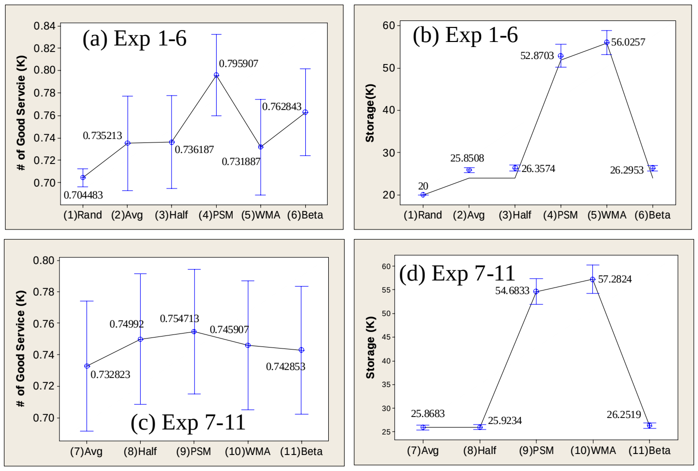
```
REQUESTER:

while TRUE do
    pick up one peer $k$ with the highest T;
    send the service request to $k$;
    after transaction, adjust $k$'s T, and update the history list;
    if $k$'s T < Threshold then
        put $k$ into the blacklist;
    end if
    if $k$'s service is bad then
        request all raters rating $k$ to send the new ratings;
        every $t$ steps, update the weight of raters;
    end if
end while
```

```
SERVICE PROVIDER:

while TRUE do
    get the service request from $i$;
    if $i$ is in the blacklist then
        Reject $i$'s request;
    else
        provide the service to $i$;
        if neighbor list is not full then
            add $i$ to the neighbor list;
        end if
    end if
end while
```

Figure 3. The skeleton of system execution in the simulation.



Exp #	ω	ϕ_A	η	η_g / η	ζ	Υ
1 (Rand)	0	0	278.914	0.758	6000.000	16.093
2 (Avg)	1.264	0.172	280.048	0.788	7755.226	22.362
3 (Half)	0.170	0.541	280.220	0.788	7907.216	19.158
4 (PSM)	49.048	0.743	280.604	0.851	15861.090	36.903
5 (WMA)	81.639	0.768	280.770	0.782	16807.712	41.38
6 (Beta)	0.947	0.946	279.558	0.819	7888.604	19.818
7 (Avg)	2.836	0.718	279.073	0.788	7760.482	22.132
8 (Half)	0.415	0.670	280.239	0.803	7777.020	20.069
9 (PSM)	72.619	0.779	279.540	0.810	16404.986	40.709
10 (WMA)	90.912	0.648	279.769	0.800	17184.726	40.247
11 (Beta)	0.707	0.098	280.224	0.795	7875.568	19.829

Figure 4. Baseline and the test of change of the percentage of honest raters: (a) and (b) (corresponding to Exp.#1 to Exp.#6) are the interval plots of baseline about η_g/η and the storage cost. In the baseline, only 20% peers are malicious raters. (c) and (d) (corresponding to Exp.#7 to Exp.#11) are the interval plots on the scenario with 80% malicious raters based on the baseline.

4.1 Baseline and Effect of Malicious Raters

Normally, we would expect most *SP*s and raters in the system are good, so we simulate an healthy environment as our baseline, where 80% *SP*s are good and 80% raters are honest. The left 20% *SP*s are fixed malicious *SP*s, and the left 20% raters are malicious. Since we want to study the effect of ratings in the trust inference, it is natural to set a high weight to the rating. In the baseline, the weight of rating W is set as 0.8, which is corresponding to figures (a) and (b), and Row 1 - Row 6 of the table in Figure 4. From Figure 4(a) and column "η_g/η" in the table, we can see that, from the viewpoint of η_g/η, *PSM* outperforms other algorithms obviously. Though *Beta* is a little bit worse than PSM, from Figure 4(b) and ζ, we can find that, the storage cost of *Beta* is just the half as that of*PSM*. The running time of *Beta* (19.818) is also far less than that of *PSM* (36.903). Despite *WMA* with the highest storage cost, the performance is not as good as *PSM*. It is even a little bit worse than *Avg* and *Half*. Because of the highest communication and storage cost, *WMA* is also the slowest algorithm. Comparing with *Rand*, all algorithms get considerable improvement, but in the cost of more storage and communication.

Figures (c) and (d), and the Row 7 - Row 11 of the table in Figure 4 show the results when the percentage of malicious raters increases from 20% to 80%. Comparing these two scenarios, we can find that, η_g/η dose not decrease significantly. Among all algorithms, *PSM* drops most heavily, from 0.795 in subfigure (a) to 0.755 in subfigure (b) for the mean number of good services among all peers, and from 0.851 to 0.810 for η_g/η from the system view. Considering the significant change of the number of malicious raters, we can say that all algorithms still perform well. To this end, we argue that *when the environment is a healthy environment with a majority of good peers, all algorithms show quite a bit resistance against malicious raters.* From the view of the storage cost, only *PSM* and *WMA* increase a little bit, while other algorithms keep almost the same. The reason is, as explained in the Section 2, *PSM* and *WMA* will accept the raters from two-hops neighbors, thus more raters will be accepted and more ratings will be generated, which leads to more dependence on ratings for *PSM* and *WMA*. When the percentage of malicious raters increases, more ratings are needed to judge the quality of the rater. For other three algorithms, the number of ratings is far fewer than those of previous two algorithms, thus the change of the storage cost is not obvious.

4.2 Effect of Bad *SP*s

To study the resistance against the bad *SP*s, 80% bad peers are contained in this group of simulations, so that we can see the obvious effect brought by bad *SP*s. From Figure 5, we can see several similar results as the baseline: the costs of *PSM* and *WMA* are much higher than those of other three algorithms in terms of the storage cost and the running time, and *PSM* is still the best algorithm to get the largest η_g/η. However, comparing to the baseline, the significant increase of the percentage of bad *SP*s incurs the significant increase of the communication and storage for *PSM* and *WMA*, and sharply drop of η_g/η. When the quality of bad *SP*s are random or oscillating, η_g/η is basically higher than the case with fixed malicious quality. One amazing result is, in the case of *SP*s with random or oscillating quality, *Avg* is the second best algorithm to get the good services. Since the design of *Avg* is much simpler, and the cost of this algorithm is lower, it is worth considering *Avg* when an open system with most peers are oscillating *SP*s. We can also observe that the performance of the other simple algorithm *Beta* is close to the performance of *Avg*. Their performance is better than the more-complicated algorithms, such as *Half* and *WMA*.

4.3 Effect of Different Type of Bad Raters

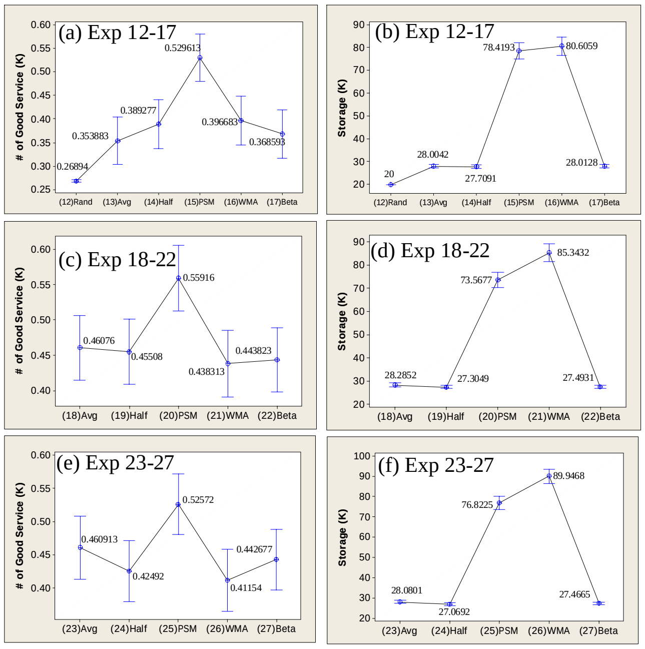
Next we compare the algorithms with respect to their resistance to the three kinds of bad raters, i.e., malicious raters, exaggerating raters, and collusive raters. In this group of experiments, we take the same configuration as the baseline except that the percentage of bad raters increases from 20% to 80%, and 20% fixed malicious *SP*s are changed to 40% oscillating *SP*s. Comparing Figure 4(a) and Figure 6(a), we can see that when the percentage of malicious raters increases significantly, the number of good services obtained by the *Avg* algorithm drops the least among all algorithms: from 0.735 to 0.666 (0.069), which is about 9.3% only. It shows that *Avg* is the strongest to resist the malicious raters. *PSM* also plays well when facing the bad raters. In Figure 6(c) when 80% raters are exaggerating, *WMA* shows a considerable weakness than other algorithms comparing to the previous cases, while in Figure 6(e) where 80% raters are collusive, *WMA* performs closely to *PSM* and is better than other four algorithms; for *Half*, the situation is opposite to *WMA*. This implies that *WMA* highly resists to collusive raters and weakly resists to exaggerating raters, while *Half* can highly resist to exaggerating raters, and weakly resists to collusive ratings. Another interesting fact we can see from Figure 6(a), (c), and (e) is, no matter facing what kind of bad raters, *Avg* performs similarly and is quite stable. This shows that *Avg* algorithm is robust against all the three kinds of bad raters. Figures (b), (d), and (f), and column "ω"and column "ζ" of the table in Figure 6 show the same result with all previous cases about the storage cost: *WMA* is with the highest communication and storage cost, and the cost of *WMA* and *PSM* are much higher than other cases from both views of the communication and storage.

4.4 Effect of Oscillating *SP*s

One of the most challenging issues in open environments is to detect and handle dynamic behaviors, which attracts a lot of attentions of researchers. In this group of experiments we intend to study the effect of dynamics. Comparing to the baseline, Figure 6 increases the percentage of oscillating *SP*s from 30% to 70%. For better comparison we again include *Rand*. The result shows that, except *PSM*, all other four algorithms get fewer η_g/η than *Rand*. Even for *PSM*, the best algorithm we think until now, it can just get very limited improvement. Considering the cost of communication and storage, we can say that all the compared algorithms play a negative effect when the quality of most SPs are oscillating. Thus we conclude that *the system dynamics is a big threat and challenge for the trust inference in open environments, and more intelligent and agile detection mechanisms are needed for the future research.*

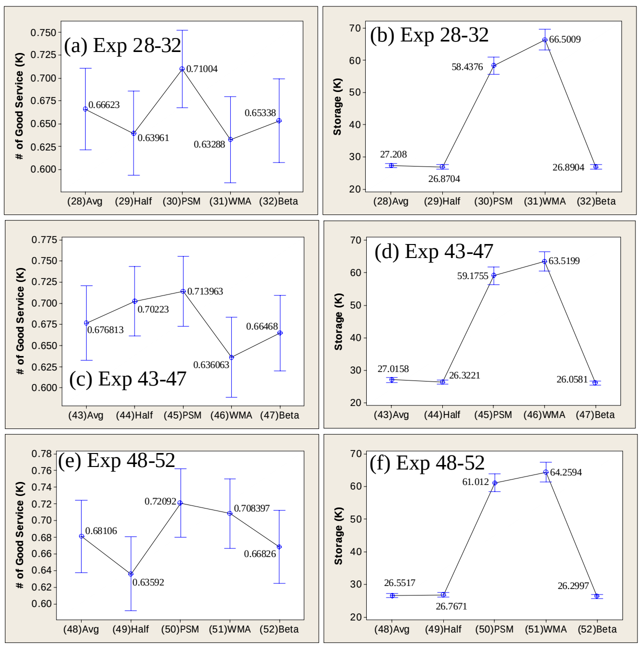
4.5 Effect of Rating Weight Change

In Section 4.2, we have known that increasing the percentage of bad *SP*s quite a bit can drop η_g/η significantly. In this group of experiments, we use the same percentage, 70% oscillating *SP*s, as the case in Figure 5(e). We also increase the percentage of malicious raters from 20% to 80%. If we keep the same weight of rating, that is $W = 0.8$, it can be imagined that the performance of the new scenario must be worse than the case in Figure 5(e), for there are much more malicious raters. Here we decrease the weight of the rating from 0.8 to 0.2. From Figure 8(a) and column "ζ" in the table, we find that η_g/η does not drop, but increases significantly instead. *Avg* and *Beta* even beat *PSM* considerably. However, from column "ζ" and column "Υ", we can see that the corresponding storage cost of *Avg* and *Beta* also increase significantly. These the running time of these two algorithms is increaseing as well, comparing with Figure 5(e). Thus, when the weight of rating is set to low and the environment is severe with many bad *SP*s and raters, the simple algorithm *Avg* and *Beta* can get same



Exp #	ω	ϕ_A	η	η_g / η	ζ	Υ
12 (Rand)	0	0	280.130	0.288	6000.000	9.464
13 (Avg)	1.178	0.080	280.661	0.378	8401.274	22.302
14 (Half)	3.250	0.034	280.039	0.417	8312.734	24.465
15 (PSM)	243.649	0.326	280.088	0.567	23525.796	59.346
16 (WMA)	287.646	0.497	279.814	0.425	24181.782	77.421
17 (Beta)	3.103	0.035	279.772	0.395	8403.846	23.203
18 (Avg)	3.054	0.531	280.540	0.493	8485.548	33.078
19 (Half)	2.481	0.504	280.171	0.487	8191.470	23.034
20 (PSM)	148.413	0.644	279.263	0.601	22070.306	57.002
21 (WMA)	285.491	0.566	279.779	0.470	25602.964	73.976
22 (Beta)	1.111	0.512	279.468	0.476	8247.928	22.392
23 (Avg)	3.710	0.629	279.931	0.494	8424.042	23.294
24 (Half)	1.292	0.540	279.405	0.456	8120.748	22.643
25 (PSM)	191.658	0.660	279.383	0.565	23046.754	56.251
26 (WMA)	302.745	0.518	279.512	0.442	26984.046	78.112
27 (Beta)	1.149	0.521	278.889	0.476	8239.952	24.135

Figure 5. Test of the quality change of *SP*s: (a) and (b) are the interval plots of the scenario where there are 70% fixed malicious *SP*s. (c) and (d) are the results about the scenario where 70% random *SP*s. (e) and (f) are the results about the scenario with 70% oscillating *SP*s.



Exp #	ω	ϕ_A	η	η_g / η	ζ	Υ
28 (Avg)	3.128	0.728	279.965	0.714	8162.394	25.026
29 (Half)	2.960	0.707	279.530	0.686	8061.128	25.056
30 (PSM)	81.060	0.806	280.012	0.761	17531.268	51.835
31 (WMA)	168.392	0.668	280.109	0.678	19950.274	66.295
32 (Beta)	0.928	0.931	279.389	0.702	8067.122	22.131
43 (Avg)	2.273	0.956	281.109	0.722	8104.726	23.744
44 (Half)	1.928	0.963	279.799	0.753	7896.626	21.451
45 (PSM)	81.416	0.842	279.578	0.766	17752.656	41.640
46 (WMA)	146.900	0.736	280.156	0.681	19055.976	48.830
47 (Beta)	0.191	0.560	280.106	0.712	7817.424	19.668
48 (Avg)	0.356	0.503	279.624	0.731	7965.508	22.202
49 (Half)	6.577	0.502	279.457	0.683	8030.142	20.209
50 (PSM)	82.381	0.672	281.032	0.770	18303.598	40.398
51 (WMA)	110.891	0.619	280.622	0.757	19277.826	39.907
52 (Beta)	1.943	0.776	279.939	0.716	7889.914	23.183

Figure 6. Test of different types of bad raters. In all these three scenarios, there are 40% oscillation *SP*s: (a) and (b) are the interval plots of the scenario where there are 80% malicious raters, (c) and (d) are the results about the scenario with 80% exaggerating raters, while (e) and (f) are the results about the scenario with 80% collusive raters.

level or even better performance than the complicated algorithm like *PSM*. The reason is in this situation, relying more on the self-experience which is more reliable than the rating, is more helpful to understand the surrounding environment.

5 Relate Work and Discussions

Our work is inspired by a large amount of previous work on reputation-based systems [3, 14, 22] and trust inference in P2P systems [1, 6, 7, 8, 10, 11, 12]. To the best of our knowledge, we are the first to systematically analyze the effect of ratings on trust inference in open environments. Next, we will list some related work on rating aggregating algorithms, trust models and reputation systems. Trust inference or reputation based systems has been a hot topic and studied in the literature [2, 10, 15, 16, 19, 23, 24, 25, 26]. Basically, many researchers are advocating the usage of ratings and prefer to complicated rating aggregation algorithms to try to filter out the bad ratings [2, 15, 25, 26, 17, 4, 5, 20]. The key to the success of ratings is the rating aggregation algorithm, i.e., how to integrate the ratings from other into one peer's own trust view.

Several recent proposals try to filter out the bad ratings through introducing another evaluation model dedicated to evaluate the credibility of the ratings [2, 15, 19, 23, 24, 25, 26], where they try to add the weight to the rating based on the evaluation of raters. At the same time for the evaluation, peers adjust the weight for different raters based on the correctness of ratings in the history. These work is close to the social network, so they are applicable to the relatively stable environment. In open environments such as P2P systems, however, these work will be degraded because they are not sensitive enough to catch the dynamics of the system.

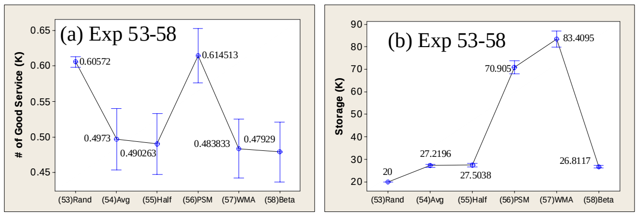
It is worth noting that the comparison of different rating aggregation algorithms is the objective of this paper, which distinguishes our work with previous work which focuses on proposing one specific aggregation algorithm. In these previous work, people proposed the aggregating model only focus on their own model. To the best of our knowledge, our work is the first effort in this field. Whitby *et al.* [20] conduct a similar analysis on the unfair ratings based on Bayesian reputation systems. However, [20] just focuses directly on the fair and unfair ratings, and doesn't not involve other factors, such as the dynamics of raters and *SP*s. Also, their evaluation of the effect in their work is just according to the average rating errors. Recently, Srivatsa *et al.* [17] gives another similar analysis in TrustGuard; however, their approach is pursuing from the angle of service providers, while our approach is from the angle of raters. Also, we consider more factors than their work. Thus, our work compliments to their work very well.

6 Summary

In this paper, we systematically evaluate the effect of different rating aggregating algorithms in the context of a simple distributed trust inference model. Figure 9 summarizes the evaluation results from the angles of algorithm complexity, system running cost, and system benefit. Based on the simulation results, we find that in most cases *PSM* outperforms other algorithms. However, when the number of bad raters and the number of dynamic peers increase, the performance of *PSM* degrades. In this case, considering the considerable cost and complexity of this method, we should turn to the simple algorithm like *Avg* and *Beta* combining with lower weight of ratings. Moreover there are some vulnerabilities residing in the design of *PSM* algorithm. *PSM* needs to calculate the personalized similarity based on the common neighbor set. However, it is very difficult to decide this common set in a real system, such as P2P systems. Basically peers decide this set based on the claim of raters. In the simulation, we make this common set accurate by abstracting it from the global information. But in real P2P systems, malicious raters can give malicious information to confuse the weight updating. Moreover, in a system with light workload, it is possible that peers can not find the raters with common neighbor set. Thus actually *PSM* is not as practical as other algorithms. From the results, we also observe that *WMA* has the largest cost, but with the worst performance, which is surprising. The possible reasons are: **(1)** In Yu's approach [26], the weight of rating is dynamic. While in this paper, the weight is fixed. **(2)** Studying carefully, we find that in Yu's approach, the weight update of raters always makes the weight decrease. Through comparing several major rating aggregating algorithms, we find that in several circumstances the simple algorithm outperforms the complicated algorithms. In particular, the *Avg* algorithm is more resistent to different type of bad raters. Actually, in our simulation, we might choose to

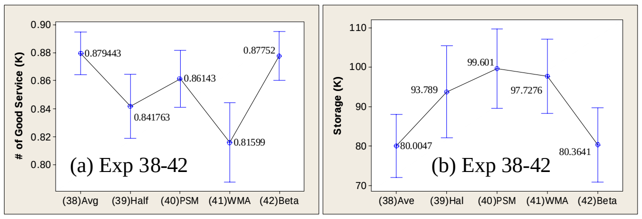
Algorithms	Complexity	Cost	Benefit
Avg	Lowest	Low	Better
Half	Medium	Low	Better
PSM	Highest	High	Best
WMA	Higher	Highest	Good
Beta	Lower	Low	Better

Figure 9. A brief summary for the compared algorithms.



Exp #	ω	ϕ_A	η	η_g/η	ζ	Υ
53 (Rand)	0.000	0.000	257.353	0.706	6000.000	8.081
54 (Avg)	2.204	0.901	256.782	0.581	8165.892	20.489
55 (Half)	0.657	0.598	257.099	0.572	8251.128	18.667
56 (PSM)	114.897	0.745	257.966	0.715	21271.490	46.416
57 (WMA)	239.884	0.543	257.365	0.564	25022.864	59.545
58 (Beta)	0.150	0.487	256.986	0.560	8043.514	17.976

Figure 7. Test when there are 70% dynamic *SP*s: (a) and (b) are the interval plots of η_g/η and the storage cost.



Exp #	ω	ϕ_A	η	η_g / η	ζ	Υ
38 (Avg)	0.525	0.486	278.047	0.949	24001.414	39.677
39 (Half)	0.550	0.518	271.053	0.932	28136.702	58.664
40 (PSM)	33.316	0.570	277.538	0.931	29880.310	47.689
41 (WMA)	90.162	0.645	278.668	0.878	29318.288	42.491
42 (Beta)	0.513	0.483	276.743	0.951	24109.230	40.889

Figure 8. Test when the weight of the rating is lower in a quite severe environment: (a) and (b) are the interval plots of η_g/η and the storage cost.

make the weight change dynamically in the trustworthiness derivation, as suggested in [26]. It is found that in this case, all algorithms perform better and *similar* in most cases (we don't list the result here because we think it is the issue related to the trust model design). Considering the simplicity of implementation, high resistent to malicious raters, and the acceptable performance, we argue that a simple algorithm like the averaging aggregating is good enough for the trust inference model. Thus, the design emphasis should focus on attacking other algorithm-independent factors, such as the dynamics of systems, rather than the rating aggregating algorithm itself. This is very crucial in resource-constraint collaborative systems, e.g., mobile ad hoc networks or sensor networks, where each peer has limited resources, e.g., memory or power.

Clearly, the work reported herein has not exhausted the problem area, and there is much more investigation needs to be done in this field. We hope this paper raises this problem to the community.

References

[1] K. Aberer and Z. Despotovic. Managing trust in a peer-to-peer inforamtion systems. *Proceedings of the 10th International Conference on Information and Knowledge Management (CIKM'01)*, 2001.

[2] S. Buchegger and J. L. Boudec. A robust reputation system for p2p and mobile ad-hoc networks. *Proceedings of the Second Workshop on the Economics of Peer-to-Peer Systems. (2004)*, 2004.

[3] F. Cornelli, E. Damiani, S. D. C. Vimercati, S. Paraboschi, and P. Samarati. Choosing reputable servents in a p2p network. *Proc. of the 11th International World Wide Web Conference (2002)*, May 2002.

[4] S. Ganeriwal and M. Srivastava. Reputation-based framework for high integrity sensor networks. *Proceedings of the 2nd ACM workshop on Security of ad hoc and sensor networks*, 2004.

[5] A. Jøsang and R. Ismail. The beta reputation system. *In Proceedings of the 15th Bled Electronic Commerce Conference*, June 2002.

[6] S. Kamvar, M. T. Schlosser, and H. Gacia-Molina. The eigentrust algorithm for reputation management in p2p networks. *Proc. of the 12th International World Wide Web Conference (2003)*, May 2003.

[7] S. Lee, R. Sherwood, and B. Bhattacharjee. Cooperative peer groups in nice. *Proc. of IEEE Conference on Computer Communications (INFOCOM'03)*, Mar. 2003.

[8] Z. Liang and W. Shi. Enforcing cooperative resource sharing in untrusted peer-to-peer environment. *accepted by ACM Journal of Mobile Networks and Applications (MONET) special issue on Non-cooperative Wireless networking and computing, to appear*, 2004.

[9] Z. Liang and W. Shi. Analysis of recommendations on trust inference in the open environment. Tech. Rep. MIST-TR-2005-002, Department of Computer Science, Wayne State University, Feb. 2005.

[10] Z. Liang and W. Shi. PET: A PErsonalized Trust model with reputation and risk evaluation for P2P resource sharing. *HICSS-38*, Jan. 2005.

[11] S. Marsh. *Formalising Trust as a Computational Concept.* Ph.D. thesis, University of Stirling, 1994.

[12] L. Mui, M. Mohtashemi, and A. Halberstadt. A computational model of trust and reputation. *Proceedings of the 35th Hawaii International Conference on System Sciences*, 2002.

[13] P. Resnick and H. R. Varian. Recommender systems. *Communications of the ACM* 40(3):56–58, Mar. 1997.

[14] P. Resnick, R. Zeckhauser, E. Friedman, and K. Kuwabara. Reputation systems. *Communications of the ACM* 43(12):45–48, 2001.

[15] J. Sabater and C. Sierra. Regret: Reputation in gregarious societies. *ACM SIGecom Exchanges* 3, 2002.

[16] M. P. Singh. Trustworthy service composition: Challenges and research questions. *Proceedings of the 1st International Joint Conference on Autonomous Agents and MultiAgent System (AAMAS)*, July 2002.

[17] M. Srivatsa, L. Xiong, and L. Liu. Trustguard: Countering vulnerabilities in reputation management for decentralized overlay networks. *To Appear in the Proceedings of 14th World Wide Web Conference (WWW 2005)*, 2005.

[18] Y. Wang and J. Vassileva. Bayesian network-based trust model. *Proc. of IEEE/WIC International Conference on Web Intelligence (WI 2003)*, Oct. 2003.

[19] Y. Wang and J. Vassileva. Trust and reputation model in peer-to-peer networks. *Third International Conference on Peer-to-Peer Computing (P2P'03)*, Sept. 2003.

[20] A. Whitby, A. Jøsang, and J.Indulska. Filtering out unfair ratings in bayesian reputation systems. *Accepted for the Autonomous Agents and Multi Agent Systems 2004 (AAMAS-04) Workshop on "Trust in Agent Societies"*, July 2004.

[21] U. Wilensky. Netlogo. `http://ccl.northwestern.edu/netlogo`

[22] L. Xiong and L. Liu. A reputation-based trust model for peer-to-peer ecommerce communities. *Proceedings of the IEEE Conference on E-Commerce*, June 2003.

[23] P. Yolum and M. P. Singh. Emergent properties of referral systems. *Proc on Autonomous Agents and Multiagent Systems*, 2003.

[24] B. Yu and M. P. Singh. A social mechanism of reputation management in electronic communities. *Proceedings of Fourth International Workshop on Cooperative Information Agents*, 2000.

[25] B. Yu and M. P. Singh. Searching social networks. *Proceedings of the 2nd International Joint Conference on Autonomous Agents and MultiAgent System (AAMAS)*, July 2003.

[26] B. Yu, M. P. Singh, and K. Sycara. Developing trust in large-scale peer-to-peer systems. *Proceedings of First IEEE Symposium on Multi-Agent Security and Survivability*, 2004.