INSTITUT FÜR INFORMATIK
der Ludwig-Maximilians-Universität München

Simulation-Based Evaluation of Reputation Algorithms for Technology Enhanced Learning Platforms

Marco Hoffmann

Masterarbeit

Aufgabensteller	Prof. Dr. François Bry
Betreuer	Prof. Dr. François Bry, Niels Heller, Dr. Alexander Pohl
Abgabe am	26.07.2016

Acknowledgement

This thesis has benefited greatly from the support of various people, whom I would like to express my gratitude to.
I would like to thank Prof. Dr. François Bry for giving me the chance to investigate the interesting topic of reputation algorithm evaluation as well as for his inspiring ideas and comments. I would also like to thank him for his feedback in early stages of this thesis which have allowed me to further hone my writing.
I am also very grateful to my first advisor Dr. Alexander Pohl for his highly competent notes and suggestions and the extremely productive discussions we had. This work has greatly gained from them. His openness and insightful thoughts have often caused me to see things from another perspective. I am very glad he continued to support me even after he left the university and went into the industry.
I want to express my deep gratitude to my second advisor Niels Heller, who assumed Alexander's advisory position during the second half of this work. His profound mathematical knowledge often provided stepping stones that very much furthered this work.
A special thanks goes to my friends, who have supported me morally throughout the past eight months.
Last, but not least, I want to thank my parents for providing moral and financial support throughout my studies and for allowing me to enjoy the invaluable broad education that has helped me reach the point where I am now.

Declaration

I hereby state that I wrote this document without any other help and without making use of any other sources than explicitely mentioned.

München, 26.07.2016 Marco Hoffmann

Abstract

This thesis introduces a simulation-based testbed for reputation algorithms specifically for classroom scenarios. Agents with several personality traits that determine their behavior during the simulation can be set up. These agents may then post artifacts (representing questions or pieces of information, i.e. content in a digital backchannel), vote artifacts of other agents. The agents can also be configured to exhibit specific social phenomena such as the Matthew effect, or asocial behavior, which can significantly influence the outcomes of reputation systems. Tested reputation algorithms then try to determine a reputation score for each agent based on their behavior. Apart from the simulator, the testbed is bundled with tools to load and parse external data, aggregate negative and positive votes into a positive number range and output test results in visualized or textual form.
In order to quantify some qualities of reputation algorithms, three quality measures were devised: *Correctness* and *Inversion Quality* are measures used to determine the accuracy of the reputation algorithms' results, i.e. if the reputation scores are in accordance to what the test programmer expected. *Distinction* tries to measure the confidence of an algorithm in its assignment. It expresses how clearly differently behaving agents are separated from each other in terms of reputation score.
Finally, four algorithms (Weighted PageRank, Backstage Reputation Algorithm, EigenTrust and Aggregate) are tested to provide a first outlook about the simulator's capabilities.

Zusammenfassung

Diese Arbeit stellt eine simulationsbasierte Testumgebung für Reputationsalgorithmen speziell für digitale Lernumgebungen vor. Anhand verschiedener Persönlichkeitsattribute können Agenten für eine Simulation konfiguriert werden. Abhängig von dieser Konfiguration wird ein bestimmtes Verhalten ausgedrückt. Die Agenten können Artefakte einstellen (welche beispielsweise Fragen oder Informationen repräsentieren, also typischer Inhalt in einem digitalen Backchannel) und Artefakte anderer Agenten bewerten. Ausserdem können Agenten so konfiguriert werden, dass sie bestimmtes soziale Effekten unterliegen, beispielsweise dem Matthew effect. Zusätzlich können sich Agenten bösartig und schädigend verhalten.
Die für eine Simulation ausgewählten Reputationsalgorithmen bestimmen danach für jeden Agenten einen Reputationswert, basierend auf seinem Verhalten. Neben dem Simulator verfügt die Testumgebung über weitere Werkzeuge, mit denen man externe Daten laden und parsen, negative und positive Votes in einen positiven Zahlenraum konvertieren und Ausgabe textuell und visualisiert speichern kann.
Um einige Qualitätsaspekte von Reputationsalgorithmen qualifizieren zu können, wurden drei Qualitätsmaße entworfen: *Correctness* und *Inversion Quality* sind Maße mit denen man die Genauigkeit eines Algorithms bestimmen kann, also ob die berechneten Reputationswerte mit der Erwartung des Testprogrammierers bezüglich dieser Werte übereinstimmen. Mit dem Maß *Distinction* wird versucht, das Vertrauen eines Algorithmus in seine Ergebnisse zu bestimmen. Es drückt aus, wie stark sich verschieden verhaltende Agenten in ihrem Reputationswert unterscheiden.
Schlussendlich werden vier Algorithmen (Weighted PageRank, Backstage Reputation Algorithm, EigenTrust and Aggregate) mit Hilfe der Testumgebung getestet, um einen ersten Einblick in die Fähigkeiten des Simulators zu erreichen.

Contents

CHAPTER 1

Introduction

Measuring and displaying reputation of users in a system is an important task in many fields of social media and online commerce platforms nowadays. It is not a presumptuous claim that online reputation is an essential corner stone of social media in which user actions are supposed to be reciprocative. In P2P file sharing systems, users are file distributors that rate each other for quality of service or file integrity. The use of a collaborative reputation score can identify malicious agents that try to spread malicious data. These agents can then be excluded from the network [AD01]. In commercial areas, where buyers can neither assess the quality of an item they want to buy because they are physically in a distant location, nor judge the trustworthiness of the seller, a reputation score created by preceding buyers can help identify high-quality items or merchants [Gef00]. In classrooms, integrating a reputation system in digital services such as the digital backchannel Backstage [BGBP11] may also serve as an incentive mechanism. With Backstage, students can digitally attach notes, questions and answers to questions to lecture slides, which can then be rated positively, negatively, or marked as "off-topic" by peers. A reputation system might motivate participants to contribute. However, a reputation score in this field could also be used to identify students that might need more attention from a teacher: A very low reputation score might indicate a subject-specific or personal problem.

Given the number of available algorithms that determine a reputation score from aggregated votes (or some other form of 'expression of opinion', such as links or file quality observations), it is of interest to determine which algorithm is most suitable for a given application. In the current research, the quality of a reputation system is primarily judged by the algorithm's robustness against certain attacks by malicious users, who might try to gain an advantage (e.g., a higher reputation than they 'deserve', or lowering the reputation of others) by tricking the system. The most common types of attacks that are identified are the sybil attack [Dou02] (a single user controlling multiple identities, thus asserting more manipulative power), whitewashing attacks [FPCS04] (a user registering a new account when the old ac-

count's reputation has dropped under a threshold) or collusion attacks [HZNR09] (multiple agents following a mutual strategy to increase his or her own reputation or lower these of others). In the classroom, robustness against these kinds of attacks, while still relevant, do not appear to be the sole criterion for the suitability of an algorithm. Betraying the system is not rewarded financially and there is no benefit of betraying a peer. However, there may still be students trying to challenge the system just for fun, e.g. by deliberately posting plausible sounding yet wrong answers to questions asked by peers. In their paper, Hazard and Singh take a different approach on the evaluation of reputation algorithms with their Macau desiderata: they attempt to rate algorithms based on properties such as *Monotonicity* (better users should have better scores) and *Convergence* (how fast an algorithm converges against the final scores the users *should* have as the available data increases) [HS13]. While conveying interesting concepts, these desiderata are very abstract and many algorithms satisfy their requirements well enough. A method that allows for a more fine-grained quality assessment seems desirable.
In order to compare algorithms specifically in the context of digital classroom applications like Backstage, a more classroom-specific simulation approach is introduced. An evaluation testbed that simulates a classroom with many virtual students is developed and is documented in this thesis. A multitude of factors are considered to simulate various kinds of student behavior on Backstage. Students can post artifacts (in a real world scenario, artifacts would be questions and answers) of good or bad quality that can be rated negatively or positively by their peers. These votes are aggregated in one or several vote matrices, one for each kind of vote, where an entry a_{ij} of such a matrix A contains the number of votes of a specific type which i has given to j. These matrices are processed by reputation algorithms after a given count of simulation iterations. It is then possible to assess these algorithms: How many iteration steps were necessary to identify a group of maliciously acting students? After several iterations, how clearly do reputation scores of highly competent students differ from those of less competent students? For example, a good reputation algorithm should rank productive, competent students highest, followed by moderately productive, 'normal' students, and ranking malicious and very lazy students lowest. Certainly, a reputation system used in a classroom context should be informative, i.e. truthfully reflect the student behavior and usage characteristics, while not introducing further distraction and misplaced incentives. For example, it should not be possible for groups of (less competent) friends that overprice each other, completely disregarding an artifact's quality, to gain the highest scores.
An algorithm's time-wise performance will also be looked upon, as to find out if more complex, slower algorithms outperform their more simple rivals or just add, in the classroom-specific context, unnecessary complexity.

1.1. Overview

The outline of this thesis is as follows: The second chapter, *Related Work*, introduces some other testbeds for reputation algorithms with a focus on simulation-based frameworks. It also presents some other noteworthy reputation algorithms that could potentially be useful in a classroom setting.
The third chapter, *Simulation Model*, explains the components, parameters and capabilities of the simulator. It also elaborates how agents can be parameterized and how they determine which action should be taken each iteration step of the simulation.
The fourth chapter, *Implementation*, documents the usage of the simulator and highlights the extensibility and functionality of the most important components of the framework implementation.
The fifth chapter, *Evaluation*, first introduces and explains the algorithms that are used for a first evaluation. Then, it outlines the three quality measures proposed. Finally, it demonstrates the test framework's capabilities as well as the selected reputation algorithm's behavior with the help of seven tests. Time-wise performance is also a subject that is briefly explored.
In the final chapter *Conclusion*, the evaluation results are interpreted and proposals for the development of reputation algorithms are derived from the test results. Also, limitations and future development regarding the simulator is discussed.

CHAPTER 2

Related Work

Employing a simulator to assess the quality of a reputation algorithm from specific perspectives and in a given context is a common approach. In 2010, Kerr and Cohen introduces TREET, which focuses on discovering potential security vulnerabilities of reputation algorithms and the development of trust in the context of online market places [KC10]. Thus, it is restricted to a seller/buyer scenario in which goods are traded and untrustworthy sellers are avoided. A more recent approach by Nguyen et. al. with their simulator called EStarMom takes the same line of focus on marketplaces, but further introduces some quality metrics that are based on game theory [PNJd15]. They propose two categories of metrics, 'community-based', which includes metrics like misbehavior detection speed, and 'individual-based', such as the gap between expected satisfaction before a transaction and actual satisfaction regarding a received product, which reportedly reflects the truthfulness of public reputation scores. EStarMom was released in 2015. A rather specific context is chosen by one of the first simulators, which is the Agent and Reputation Testbed (ART) created by Fullam et. al. in 2005 [FKM+05]. The authors have chosen a scenario in which agents estimate the worth of paintings for clients, and consult other agents before they make a decision. Lastly, in 2010, West et. al. introduces a P2P trust testbed that simulates a file sharing network. It should be mentioned that there is also criticism against the use of simulators. In their article, Jelenc et. al. claim that the assumption that testing different algorithms against the same decision making process (i.e. the logic the simulator's agents act upon) is incorrect, with the way decisions are made has too much of an influence on the outcome [JHSMT13]. Another strategy is selected by Hazard and Singh, who do not use a simulator for evaluation, but rather devise mathematical desiderata that focuses on implications of behavior and convergence speed given a target reputation value [HS13].

Four algorithms are evaluated in this thesis: Aggregate, a self-devised algorithm, EigenTrust [KSGM03], Weighted PageRank [XG04] and the Backstage Reputation Algorithm [Poh15]. However, there are many more noteworthy and interesting

reputation algorithms that could not be included in this thesis. In 2002, Josang and Ismail proposed their Beta Reputation System (BRS) which makes use of beta probability density functions to calculate a trust value from negative and positive experiences. Interestingly, BRS does not only consider binary negative or positive observations, but they can have a 'level of positivity', respectively negativity. Similarly, TRAVOS [TPJL06] also relies on beta probability density functions for their calculations, but the authors chose to only allow binary negative or positive outcomes. Another interesting algorithm that focuses on social reputation and closeness of relationships is the REGRET system designed by Sabater and Sierra in 2001 [SS01]. This algorithm considers the time an impression of a person was perceived. Impressions that are further in the past are not taken into account as strongly, allowing the algorithm to adapt quickly to changing situations (e.g., a student has a change of motivation and starts contributing bad quality content). Finally, Abdul-Rahman and Hailes propose their ARH trust model, which divides experiences into four categories: very bad, bad, good and very good [ARH00]. Noteworthy about this algorithm is that it claims to have a mechanism that detects lying agents who constantly badmouth others. The opinions of detected liars are henceforth discounted in the reputation score calculation.
Even though attack resistance does not seem to be the primary field of concern for a classroom scenario, it was still important that malicious agents and thus attacks are supported by the simulator to some extent. In their article, Swamynathan et. al. provide a brief overview over the most common attack types [SAZ10]. They were used to shape this thesis' student model so that such attack patterns can be simulated as well. Their article is supplemented by the research of Hofman et. al., who reveal some more attack types, such as orchestrated attacks in which multiple attackers collaborate [HZNR09].
Last but not least, Vassileva's article provides a comprehensive overview about different incentive mechanisms in the context of E-Learning [Vas08]. Her insights about the positive effects of reputation systems used in social learning provide a foundation for the employment of such systems within technology-enhanced learning platforms. Apart from motivating students to actively contribute in order to gain a high reputation and be socially recognized, Vassileva claims that reputation systems help finding quality content and encourage the emergence of a like-minded community.

CHAPTER 3

Simulation Model

The core component of the evaluation testbed is the simulation part. The simulation consists of five components.

- **Artifacts** represent anything a student has produced and sent on the backchannel, e.g. a question or a post. Artifacts have a quality: some content contributed to a digital backchannel might be informative or enlightening, other might be off-topic or contain misleading or wrong information. To reduce complexity and increase the simulation's traceability, it can either be of good quality or bad quality, but nothing in between. An artifact has exactly one originator.
- **Votes** can be cast by students for an artifact. They can either be positive or negative, depending on whether the voter believes an artifact to be of good or bad quality. A vote can only be cast once for a single artifact. Votes determine the students' reputations: a Student A voting for an artifact by an originator B can be interpreted as a Student A voting for a Student B. A vote also has exactly one originator and exactly one target artifact.
- A **student** is an agent that expresses a certain behavior based on predefined personality traits and on a predefined set of feasible actions. Each turn, a decision tree is traversed with each node holding a probability for one decision branch or another. These probabilities are determined by the personality traits. The outcomes of the decision branches can be posting a good or bad artifact, casting a positive or negative votes fairly (giving good votes to good artifacts and bad votes to bad artifacts) or unfairly (giving good votes to bad content and vice versa), or acting maliciously, e.g. by trying to cast negative votes to positive artifacts only. Students also carry a freely selectable *group flag* that helps identifying their agenda easily in the evaluation phase.
- A **community** is the comprehensive data object that manages all students, artifacts and votes and provides methods for the student agents to access

relevant information for them, such as artifacts that can currently be rated, top students and so on. It also stores all information necessary for the simulation itself and keeps track of statistics.

- During one **iteration**, every student can execute exactly one action (e.g. posting, voting, or idling). The simulation ends after a specified number of iterations.

As the community object encapsulates everything necessary for one simulation, multiple simulations can be run parallel. Thus, the testbed greatly benefits from multicore processors.

3.1. Community

In order to take part in the simulation, virtual students need to be added to a *Community* that keeps tracks of votes and handles simulation execution, constraints and parameters. A *Community* assumes that there is only one 'lecture' that everyone attends: Every student within the *Community* has access to every posted artifact and may rate it. Every simulation iteration, each student is called so that they can perform their action. As to avoid biases based on execution order, the execution sequence is randomized every iteration. The following parameters are handled by the *Community* objects:

- **Maximum number of best students** respectively **maximum number of worst students**, a number controlling how many students at most can be selected for the best students and worst students list that are 'known' in the classroom for their achievements. Keeping these lists is important for simulating the *Matthew effect*: This denotes the phenomenon that excellent students receive better votes because it is commonly trusted that their produce is of good quality [Rig13]. Correspondingly, a negative effect, resulting in bad students receiving bad votes because it is assumed that their work is of inferior quality, might also exist as findings from Jin et al. suggests [JJLU13].

- A number that controls how many iteration steps must have passed until it becomes possible for students to act socially influenced, that is, affected by the Matthew effect. In the first few rounds, it is not very clear and mostly due to chance who is in the top student list. As the Matthew effect reinforces the top position of students (and vice versa for the reverse Matthew effect), a random student would be pushed to the top. Also, in real life, it takes some time for excellent students' reputations to spread and become common knowledge. It is thus not recommended to set this parameter to a very low number.

- **Points for good votes** respectively **points for bad votes** determine how a *good vote* and a *bad vote* translate to score. This is normally set to **1** for good votes and **-1** for bad votes, however, a bias can be introduced, e.g. a setup where bad votes have a more drastic effect (**scoreForBadVote=-3**) or none at all (**scoreForBadVote=0**).

Additionally, the *Community* maintains information that is important or interesting to the simulation.

- **Artifacts**, a list of artifacts (an abstract representation of posts such as comments or questions) that have already been posted. Student agents can only rate an artifact once and have to do so through the methods provided by the community object.
- **Current votings**, a matrix containing all votes
- A statistics module that keeps track of..
  - the total post count, additionally the ratio of good to bad quality posts
  - the total vote count, additionally the ratio of positive to negative votes
  - the student behavior regarding posts and votes per student group flag, e.g. student type 'motivated students' posted 50 times, 45 good quality posts, 5 bad quality posts

3.2. Student Model

Students are modeled by agents that perform exactly one action in every simulation iteration. Being idle in a round is also considered an action.
As previously mentioned, agent behavior can be adjusted by setting 'personality traits'. In an initial attempt that has been discarded in favor of the one described here, the probability for each possible action was calculated at once using formulas with many variables, as some actions are impacted by many traits. However, to increase understandability of agent's behavior, this approach was forsaken in favor of a decision tree approach, with each decision node probability only being influenced by one trait.
In addition to traits, every student agent can be part of social groups, and have groups whose members they dislike (and thus discriminate against when voting) or favor and vote positively, disregarding their actual performance. This way, friendship groups or rivaling groups can be introduced in the simulation.
Finally, each student carries a *group flag* used for later identification. This is best explained using an example: Given a simulation with 10 excellent students and 10 malicious students (Student 1 to Student 20), each of them are initialized with the respective label 'excellent' or 'malicious'. After the simulation, the output does not only contain student numbers and reputation values, but also the predefined label. This allows for an easier interpretation of the results.

3.2.1. Traits

The behavior of an agent can be adjusted using nine personality traits. The lowest a trait can be set to is 0.0 (Never do it), the highest is 1.0 (Always do it). Setting a trait to x translates to a chance of x in the corresponding decision node for the positive decision branch and $1 - x$ for the negative branch. Upon initialization of

an agent, a **variance** can also be specified — every trait will then be added a value within $[-variance, +variance]$ taken from an uniform distribution.

- **Activity** determines the chance that a student is not idle in an iteration.
- **Productivity** decides whether a student is actively contributing, thus producing artifacts, or being more passive in his or her activity, thus only voting other artifacts.
- **Positivity** defines whether a student is biased towards positive action — casting preferably positive votes or trying to gain trust, thus forgoing malicious behavior for the moment — or negative action, casting preferably negative votes or commencing disruptive behavior, such as spamming bad quality posts.
- High **Competence** enables an agent to post good quality artifacts or determine which of it's peers' artifacts are of good quality. If set to 0.0, a student will always produce bad quality artifacts. When voting, a student will have to guess the quality of an artifact: The detection rate is thus 50%.
- **Maliciousness** controls whether the student behaves reasonably in the context of a classroom, or is ready to make disruptive contributions or unfair or random votes.
- **Hostility** affects malicious behavior: A low value leads to random, more general malicious behavior, while a high value makes an agent target specific groups of students (peers belonging to groups that are in the agent's **repulsions** set) with its actions.
- **Loyalty** determines whether an agent behaves differently towards friends (peers beloging to groups within the agent's **affiliations** set). A very loyal agent gives friends good votes, and spares them of malicious action.
- **Influenceability** influences the agent's susceptibility to the Matthew effect (see section 3.1).
- Finally, **Social Positivity** decides whether an agent acting on the Matthew effect is praising a good student or punishing a bad student (reverse Matthew effect). This trait has no effect when m

3.2.2. Decision Tree

An agent's action is determined by traversing down a decision tree until an action node (a node outlined in blue in Figure 3.2) is reached. The start node in Figure 3.2 is the one furthest to the left. In every node, it is determined whether the next decision path is positive or a negative based on **one** agent's attribute, which corresponds to the chance of the *positive* outcome. Considering the first node, 'Do something?', the determining property is **Activity** as reflected in the figure. If it is set to, for example, **0.8**, each time the node is traversed, a coin is tossed that has

an 80% chance of returning 'Yes', which is the positive outcome. Thus, if traversed often enough, the agent will have taken the 'Yes' route about 80% of the time.
In three cases, the outcome of a decision node does not result in another decision node or a final action node, but a filter step is added that manipulates the list of artifacts that is available to the student in this iteration. A filter step is denoted by the yellow node background.

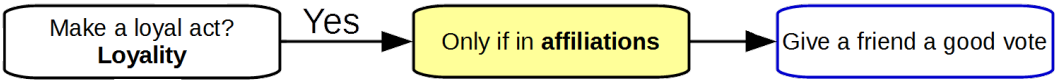


Figure 3.1.: In this case, only artifacts of friends remain, so that a situation is simulated in which a friend is explicitly upvoted. If there are no artifacts by friends, no action is taken and the round is skipped.

Apart from the situation in Figure 3.1, there are two more situations where filter steps are applied:

- When an agent performs a malicious vote, but the dice roll determined that loyal behavior is still expressed, all artifacts by peers that belong to a social group the agent likes are removed. They do not become target of the malicious action. If *Loyality* is set to **1.0**, colluding agents can be simulated that do not target each other while collaborating.
- An even more targeting behavior can be set by furthermore increasing the *Hostility* value, which removes every artifact that is not created by peers that are not in the agent's repulsion list. In this case, when an agent acts maliciously and in a hostile manner, only members of groups the agent specifically dislikes are targeted.

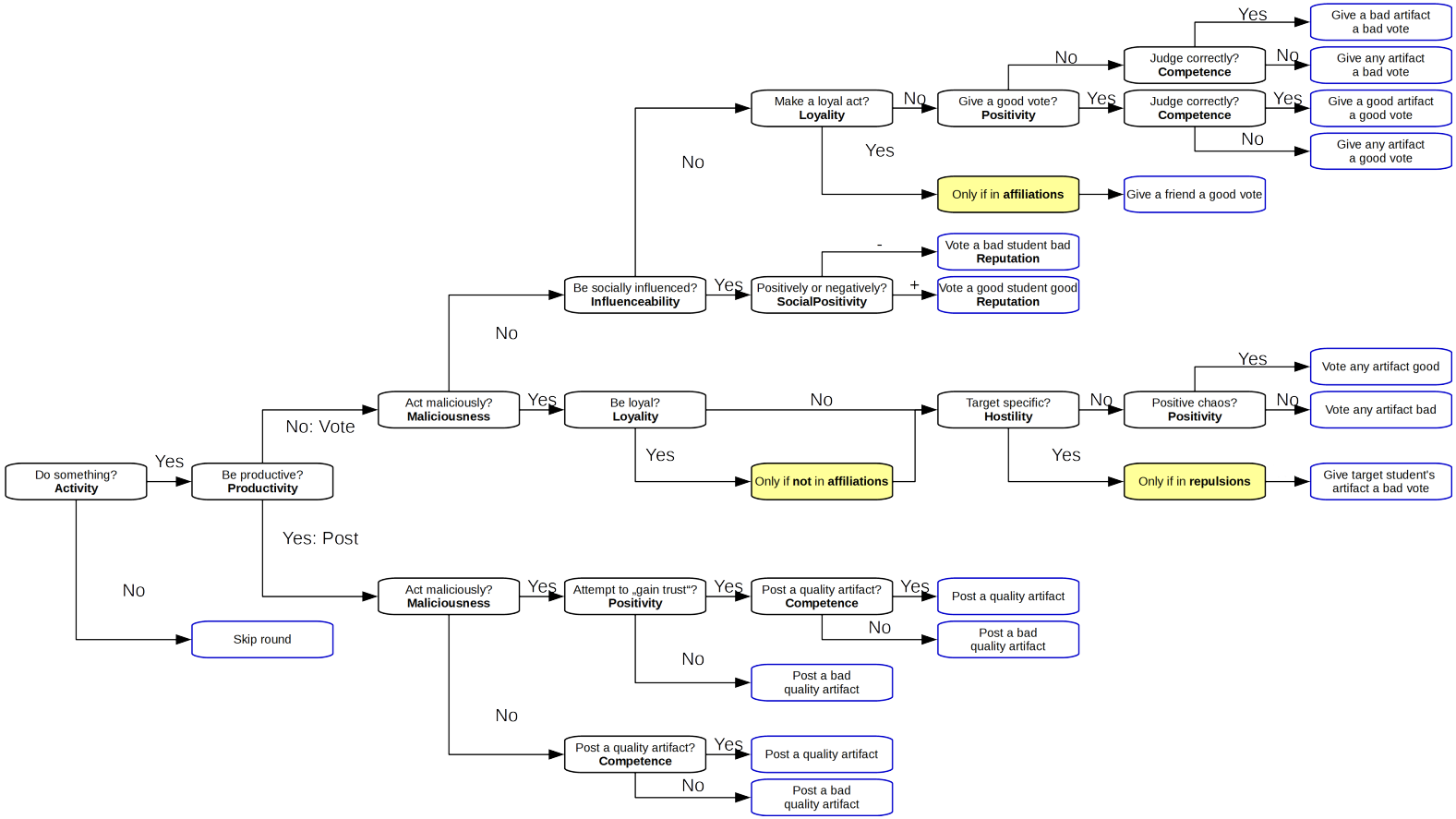


Figure 3.2.: An agent's decision tree.

CHAPTER 4

Implementation

The evaluation framework is written in Java 8 because previous versions of Java do not support Lambda Expressions. In a future version of the simulator, they could be used to instantiate agents that change their behavior over time by passing a Lambda Expression that manipulates an agent's parameters depending on the iteration count. During implementation, the focus was on easy extensibility, thus most components have been designed to be modular. The framework is bundled with the EJML Matrix Library for Java [EJM16], as many reputation algorithms make use of matrix operations. This chapter gives an overview over the most important system components beside the simulation model of the framework.

4.1. Input

The framework can run in two modes. Either the data for the reputation algorithms is generated by the simulator, or it is loaded from external sources. Backstage uses MongoDB [Mon16] to save its data, which supports data export into comma-separated value format. An input parser for Backstage's data export, `BackstageCSVParser`, is already available.
In Backstage, there are actually three voting types: positive, negative and off-topic. For the framework, these types have to be converted into numbers. By default, positive is treated as 1, negative and off-topic are both treated as -1. These interpretations can be reconfigured.
The recommended way to parse data is to first create a list of `LooseVotes`. A LooseVote is an object that contains three pieces of information:

- The caster of the vote c
- The receiver of the vote j
- The numerical interpretation of the vote, s, henceforth called 'score'. If this value is positive, the `LooseVote` is considered positive and vice versa. This

value can be set for each vote individually. Typically, a set of votes of one type has the same score.

As some algorithms, such as Weighted PageRank [XG04], operate solely on positive values, input parsers should pass the list of `LooseVotes` to the class `NegativeVoteProcessor` with a mode that specifies how negative votes are handled. Then, an object of type `ReputationData` is returned. If there are n unique students, this object contains a matrix $N \in (\mathbb{R}_0^+)^{n \times n}$ (negative votes) and a matrix $P \in (\mathbb{R}_0^+)^{n \times n}$ (positive votes). Four options for handling votes are already available. Depending on which mode is chosen, the matrices in the `ReputationData` object are constructed differently. N and P are initialized so that every entry has the value 0.

- **None**: negative votes are not converted anyhow. $P_{ij} = s^+_{i,j}$, where the $s^+_{i,j}$ are the sums of the scores of the positive votes of i for j. Similarly, but using the absolute value: $N_{ij} = |s^-_{i,j}|$, where the $s^-_{i,j}$ are the sums of the scores of the negative votes of i for j.
- **Ignore**: In this setting, P is defined as in the 'None' setting, N is set to zero in every entry.
- **Elevate**: a linear transformation into a positive range. The values of P are computed as sums of scores, similar to the previous settings, but without distinguishing between positive and negative weights. A leverage step in then used to assert that P only contains positive values, and the smallest value is 0. V is a $n \times n$ matrix where

  $V_{ij} = s^+_{i,j} + s^-_{i,j}$.

  If m is the smallest (possibly negative) entry of V, then set

  $P_{ij} = V_{ij} - m$.

  N is defined as in the 'Ignore' Setting.
- **Ratio**: Positive `Loosevotes` and negative `Loosevotes` are processed as in the 'None' setting , yielding values $c_{+,i,j}$ (corresponding to P_{ij} in 'None') and $c_{-,i,j}$ (corresponding to N_{ij} in 'None'). P is then defined with

  $P_{ij} = \frac{c_{+,i,j}}{c_{+,i,j}+c_{-,i,j}} \cdot c_{+,i,j}$. N is defined as in the 'Ignore' Setting. This way, positive votes given to j will be discounted if there are also negative votes.

The object of type `ReputationData`, which contains the aggregated data, is processable by the implemented algorithms (and all subsequently added algorithms that follow the framework's specifications).

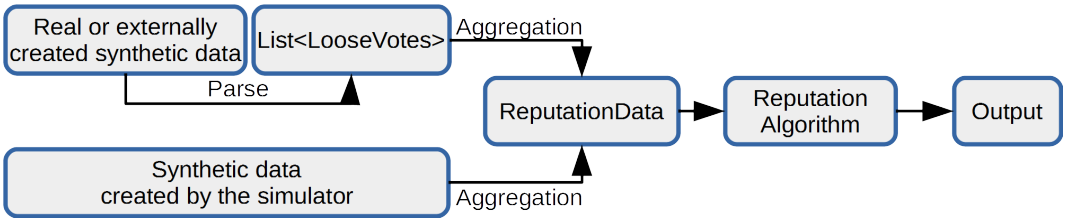


Figure 4.1.: The simulator's data pipeline

Additional input parsers can be added by extending the abstract `InputParser` class, which forces the implementation of `public void parseFile(String path, NegativeVoteTreatment nvt)`. The first argument specifies the file path that needs to be parsed, the second one refers to the negative vote treatment modes described above.

4.2. Output

In order to allow for easy interpretation of the results, multiple output modules can be assigned to a test. Custom modules can easily be implemented by extending the abstract `Outputter` class. Three output modules have been implemented already. In case further statistical analysis is desired with the help of external tools, the module `CSVOutputter` creates a comma-separated value file containing a list of sorted reputation values alongside an identifier for the corresponding agent and its group label. These files can be imported by most statistical tools, e.g. R [R16] or SPSS [SPS16]. Below is an example of the output created by the `CSVOutputter`.

```
...
"Student 3";"low";0.03335643230467201
"Student 18";"average";0.0334063999868436
"Student 22";"high";0.03346493363697268
"Student 28";"high";0.03346693164264181
...
```

The second output module, `QualityMeasureOutputter`, calculates the quality measure scores that are proposed in this thesis (cf. Section 5.2) and saves them to a text file. Also, the time that was required for the test to complete is added. The output file looks like this:

```
###Distinction###
zoneDiff=8.131255720790487E-5
averageDiff=6.414352571995097E-5
distinction=1.2676658524028357
###Correctness###
correctness=0.6
###Inversions###
inversions=58.0
inversionQuality=0.8066666666666666
###Time###
306.434s
```

Lastly, the module `ChartOutputter` generates a visualization of the results.

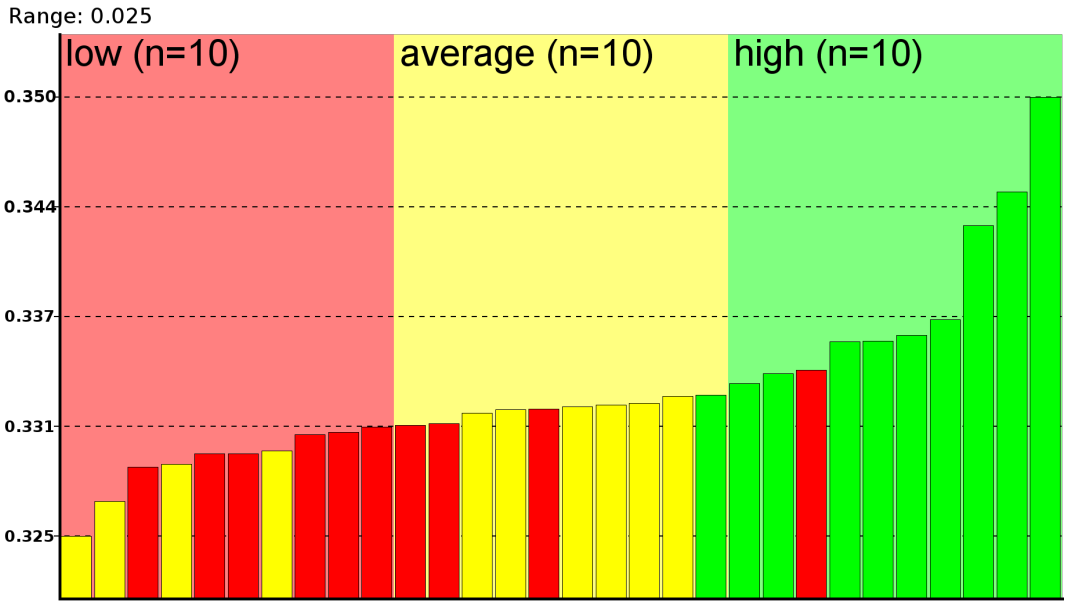


Figure 4.2.: A chart automatically generated by the `ChartOutputter` module.

As elaborated in Section 5.2, every agent in the simulation must belong to a group which are ordered by their expected ranking position. To visualize this, `ChartOutputter` outputs a special bar chart that was developed in this thesis. Upon initialization, the `ChartOutputter` must be informed of these groups, their name and the color they should be represented with. In the chart, every bar represents an agent — the further to the right the bar is, the higher in the reputation ranking. The height of the bar codes that agent's reputation score. The color of the bar identifies the group the agent belongs to. As the groups are to be ordered by the test programmer according to their expected ranking, the background color in the chart represents the expected ranking zone for a member of the corresponding group. For example, in Figure 4.2, it can easily be concluded that the red bar furthest to the right is mispositioned. It should be further to the left and ideally align with the red background.

To increase contrast, for the background color, the `ChartOutputter` module automatically decreases color saturation. On the top left corner of the expected ranking zone, the corresponding group's name as well as their group size is drawn. Optionally, in the top left corner of the output image, the difference between the highest reputation value and the lowest one is drawn, denoted as 'range'.

As can be seen, the `ChartOutputter` module can handle arbitrary group size ratios, colors and scales up to a high number of agents without losing lucidity.

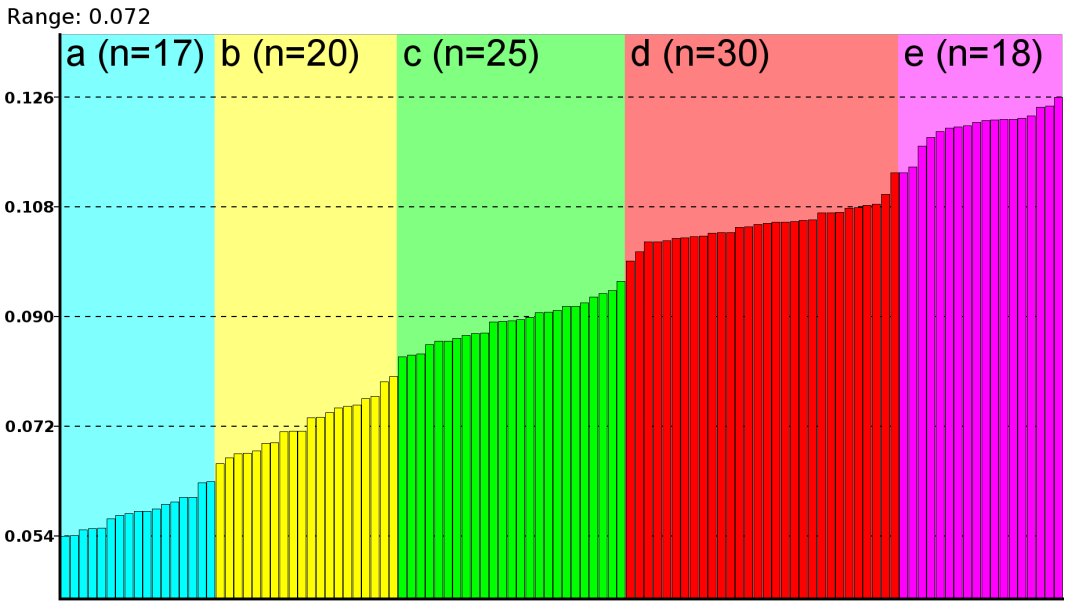


Figure 4.3.: The `ChartOutputter` module can handle arbitrary colors and scales up to many agents.

4.3. Algorithms

As the framework presents itself as an evaluation testbed, it was designed so it is simple as possible to add additional reputation algorithms. An algorithm must extend the abstract class `ReputationAlgorithm`, which enforces the implementation of two methods.

```
public SimpleMatrix getRawReputation(ReputationData rep)
public String getName()
```

The framework's scheduler is then able to autodetect the new algorithm. The parent class also provides methods for automatically sorting and adding identifiers and group to the reputation score vector. `getName()` should simply return the name of the algorithm. The method `getRawReputation()` takes the object `ReputationData`, which contains the student count and separate vote matrices for positive votes and negative votes. Each vote matrix's row i contains the sum of votes from a student i to a student j, stored in the corresponding column j. At the end of the calculation, an algorithm should return a `SimpleMatrix` (specified by the EJML library), which should have exactly one column and as many rows as there are students. Each row i should contain the calculated reputation score for a student i.
The framework was initially devised to test additions and parameters for the Backstage Reputation Algorithm BRA [Poh15] (also cf. Section 5.1.2). Thus, this algorithm was implemented especially modularly. BRA uses different matrices in its calculation: The *Activity Matrix*, *Backchannel Praise Probability Matrix* and an *Identity Matrix*. In the implementation, each of them is a module that encapsulates the

calculations required to construct the corresponding matrix. That way, they can be exchanged easily, and new modules only have to implement the corresponding abstract classes. They are (using parameters as proposed by Pohl) brought together in the module *SimpleEstimatedPraiseProbability*, which can also be exchanged, as long as the new module extends the abstract class `EstimatedPraiseProbability-Matrix`.

4.4. Scheduler

The scheduler is the module that handles test execution and provides helper functions for various tasks: It generates file names for the output modules that reveal which kind of algorithm was run and the corresponding parameters that were used, if desired. It also takes care that no previous test results are accidentally overwritten, adding unique numbers to the file name if necessary. Additionally, it automatically generates identifiers for unnamed agents with the help of modules extending `MappingCreator`, which assigns enumerated identifiers to agents. Two simple `MappingCreator`s are already available: `StudentMapping` gives the agents names in the form of the string 'Student' plus a number (e.g. 'Student5'), whereas `ShortStudentMapping` abbreviates the 'Student' to an 'S', e.g. 'S5'.
The scheduler also keeps a list of output modules it should use when a test result needs to be outputted. It is possible to add several output modules to the scheduler, so tests do not have to run multiple times for multiple outputs.
There are two ways to operate the scheduler. Either, a `ReputationData` object is passed onto the scheduler read from an external source, or a **Community** object containing agents and parameters as described in Chapter 3. In both cases, it is possible to either pass a list of reputation algorithms that have been set up using specific parameters, or let the framework autodetect all available algorithms and use them with their default parameters.
When the scheduler runs in simulation mode (i.e. a **Community** object has been passed), there are additional parameters that may be required by the simulator:

- **steps**: a number controlling how many iterations each simulation should run before termination
- **recordAfterEvery**: this number controls after how many iteration steps output result files should be produced. If steps=6 and recordAfterEvery=2, then the result will be outputted after round 2, 4 and 6.
- **rounds**: in case the tests should be run multiple times with the results being averaged across all test runs, this number controls how many times the test should be run.

For some tests, it might be interesting to check in what manner the algorithms converge. Indicator for the current state of the reputation ranking are given by the quality measures as described in Section 5.2. It would be possible to set *recordAfterEvery* accordingly and process the outputted files externally, however, the sched-

uler can be configured to perform a convergence test. In this case, a `QualityMeasureOutputter` is required, and the quality measures will be averaged after every *recordAfterEvery* round instead of the reputation scores.

Finally, after receiving the test, the scheduler automatically detects the number of cpu cores, performs a deep copy of the *Community* object for each algorithm that is to be tested (so that the simulations do not accidentally interfere with each other) and starts a thread per test.

4.5. Statistics

A statistics module can be added to a *Community* that keeps track actions and prints them to the console after a test has ended. This is often useful to get a first idea if the parameters of agents indeed lead to the desired behavior. The following sample output is taken from the *Coordinated Friends Test* (cf. Section 5.3.4).

```
Backstage done. Took 109.254s
1798 votes
--920 positive votes
--878 negative votes (51,17% positive)
823 posts
--15 quality posts
--108 bad posts (86,88% positive)
#### Posts by label ####
unaffiliated:
--698 quality posts
--62 bad posts (91,84% positive)
friends:
--17 quality posts
--46 bad posts (26,98% positive)
#### Votes by label ####
unaffiliated:
--True Good: 848, False Good: 9
--True Bad: 799, False Bad: 79
friends:
--True Good: 17, False Good: 46
--True Bad: 0, False Bad: 0
```

The test evaluates whether significantly less competent friends can outrank their more competent peers if all they do is exclusively vote each other up. The statistics output reveals that the agents are well configured. The friends group produced a lot more bad quality content than good quality content, and has a high number of false good votes.

4.6. Sample Test Setup

Finally, setting up an exemplary test is demonstrated. The test needs to be written in a class that extends the abstract class `Test`. Again, the *Coordinated Friends Test* (cf. Section 5.3.4) serves as an example.

```
name = "CoordinatedFriendsTest_2";
```

First of all, the test is named. The name is later used to inform the output modules that a new folder named like the test should be generated. The test results are stored in that folder.

```
ArrayList<LabelColorPair> lcp = new ArrayList<>();
lcp.add(new LabelColorPair("friends", Color.blue));
lcp.add(new LabelColorPair("unaffiliated", Color.cyan));
```

Both of the desired output modules, `ChartOutputter` and `QualityMeasureOutputter`, need to be informed about what kind of group labels are used and their corresponding colors for the chart visualization. Although `QualityMeasureOutputter` does not need the color, it also accepts the list of `LabelColorPairs` because it still contains all necessary information. In this case, there should be two types of desired agents: friends and agents without any social affilations — 'unaffiliated'.

```
ChartOutputter cout = new ChartOutputter(lcp);
QualityMeasureOutputter qmo = new QualityMeasureOutputter(lcp);
cout.setPrePath(getPrePath());
qmo.setPrePath(getPrePath());
```

The output modules are initialized with the required information. The `setPrePath` informs the output module that a specific folder is desired in which the results are stored. The `getPrePath` method is provided by the parent class `Test`, which generates a path based on the test name.

```
Scheduler scheduler = new Scheduler();
scheduler.addOutputter(qmo);
scheduler.addOutputter(cout);
```

A new scheduler is created and the configured output modules are added. The scheduler will now use both modules to output results.

```
Community c = new Community();

c.addUser("unaffiliated", //group label
1, //Activity
0.3, //Productivity
0, //Maliciousness
0, //Hostility
0, //Influenceability
0.5, //Positivity, 0.5 -> unbiased
0.9, //Competence
0, //Loyality
0, //Social Positivity
0, //Variance, 0 -> all values should be taken exactly as specified
new ArrayList<>(), //affiliations --- empty list -> none
```

```
new ArrayList<>(), //repulsions --- empty list -> none
new ArrayList<>(), //ownGroups --- empty list -> none
25); //number of agents that should be created with these settings
```

A new *Community* object needs to be created before agents can be added. In this case, first, the 25 normal peers are added with a high *Competence* of **0.9** and the previously used group label 'unaffiliated'.

```
ArrayList<String> friendGroup = new ArrayList<>();
friendGroup.add("bestFriends");

c.addUser("friends", 1, 0.3, 0,
        0, 0, 0.5,
        0.2, 1.0, 0, 0, friendGroup, new ArrayList<>(), friendGroup, 2);
```

After that, the two significantly less competent (*Competence* = **0.2**) friends are added. They should recognize each other as friends, so a list containing the group identifier 'friends' is added to the agent's `ownGroup` and `affiliations` field, meaning the agents belong to the social group called 'bestFriends', and also like members of the social group 'bestFriends'. The argument is passed as a list because agents can belong to more than one social group, respectively like or hate more than one social group.

```
scheduler.scheduleSynthethicAveragedAll(c, 100, 100, 2000);
```

Finally the scheduler is told to schedule a

- ...synthetic, so based on a *Community* object and not on external data
- ...averaged, so multiple tests should be run and the result should be averaged
- ...'all', so all available algorithms should be detected automatically and used

test run using `Community c`. The simulations should have 100 iterations, the results should be saved after every 100th round (corresponding to the end of the simulation in this case), and the results should be averaged over 2000 test runs.

CHAPTER 5

Evaluation

5.1. Tested Reputation Algorithms

Four different algorithms are implemented and evaluated in this thesis. Weighted PageRank [XG04] is an extension of PageRank [PBMW99], which is selected because it is a very popular, interesting and straight-forward algorithm. Its idea about transitive trust has inspired many other algorithms, such as EigenTrust [KSGM03], which has been selected as well, because it is often evaluated in other papers (e.g. [WKLS10],[SAZ10],[JHSMT13]). The Backstage Reputation Algorithm (abbreviated BRA) [Poh15] is of special interest because it is an adaption of eigenvector-based reputation similar to PageRank tailored for classroom communities. Finally, Aggregate is a simple reputation algorithm designed to satisfy the expectations from a reputation algorithm: positive votes toward an agent raise its reputation, negative votes lower it. This algorithm is employed in order to evaluate whether adding more complexity to an algorithm actually delivers better results.

5.1.1. Weighted PageRank

PageRank [PBMW99] is a well-known ranking algorithm by Larry Page, Sergey Brin et al. used with the search engine *Google*. It has been designed to determine the relevance of websites. Given a set of websites that link to each other, the premise is that websites that have many links pointing to it are more relevant or important than websites with less links. Additionally, a website that is considered relevant has a greater positive impact on the relevance of websites that it links to. As an explanation, Page et. al. introduce the concept of a random imaginary web surfer that starts at a website, and then randomly clicks links on this website that brings the surfer to another website. Furthermore, with a small chance, instead of following a random link, the surfer enters a new website address and restarts the process at a completely different website. This probability is called *leap factor*. The process is also re-initiated when a website has no outgoing links. Such a behavior can be

modeled as a Markov chain: The PageRank scores for websites are the probabilities of the surfer finding itself on the corresponding website after a large number of iterations. These scores can also be achieved by calculating the dominant eigenvector of the stochastic adjacency matrix that is yielded by accumulating the websites' links. Therefore, this algorithm is based on eigenvector computation.
Students can vote multiple artifacts of a single student. The original PageRank does however not consider the number of links from a specific website to another: It only considers if there is at least one link or not. Thus, in this case, an extension of PageRank by Xing et. al. was used that also considers the 'link count': Weighted PageRank [XG04].
For the web, PageRank is about the reputation of websites. This concept can be transferred to a classroom reputation context. The websites are students and a link from Student A to Student B is equivalent to a positive vote from Student A to an artifact of Student B. The leap factor is used to model random voting noise. It is slightly modified: It is not possible for a student to vote for themselves under any circumstance. Thus, it is also not possible in the random leap to 'jump' to oneself. The PageRank score for a student, originally meant as relevance, is interpreted as his or her reputation. As there are no such thing as a 'negative' link on the web, this algorithm does not support edges that are labeled with negative values.
Given four students A, B, C and D that have cast votes onto each other, cf. Figure 5.1.

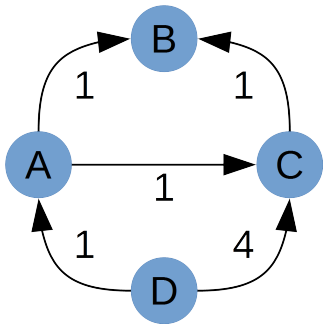


Figure 5.1.: A prototypical classroom voting scenario.

This graph is internally represented by an adjacency matrix.

Table 5.1.

↗	A	B	C	D
A	0	1	1	0
B	0	0	0	0
C	0	1	0	0
D	1	0	5	0

First, the adjacency matrix is converted into a row stochastic matrix. B has not

voted for anyone, so there will be a random jump with uniform probabilities for everyone else except B.

Table 5.2.

↗	A	B	C	D
A	0	0.5	0.5	0
B	0.333	0	0.333	0.333
C	0	1.0	0	0
D	0.2	0	0.8	0

At last, the random leap factor is incorporated by adding universally distributed probabilities while retaining the right stochastic matrix properties. In this case, a leap factor of 0.85 was used.

Table 5.3.

↗	A	B	C	D
A	0	0.481	0.481	0.039
B	0.333	0	0.333	0.333
C	0.039	0.922	0	0.039
D	0.216	0.390	0.745	0

Then, the left eigenvector is computed and normalized to magnitude 1.

Table 5.4.: Reputation scores as calculated by Weighted PageRank

A	0.168
B	0.376
C	0.312
D	0.144

In this example, Weighted PageRank assigns B a better score than C although B receives less good votes in total. This can be explained because C focuses its votes on B, while itself receiving good votes. Although D casts a lot of votes, D itself is not relevant because no one votes for D.

5.1.2. Backstage Reputation Algorithm (BRA)

The Backstage Reputation Algorithm (BRA [Poh15] is a prototype reputation algorithm which is based on eigenvector computation. As Backstage is a digital classroom backchannel, this algorithm has been tailored to a classroom scenario. It introduces the concept of a *rating activity coefficient* that reinforces the impact of

a student's opinion on his or her peers if that student has voted many times rather than a few times.
Given three students A, B and C. Let A give C exactly one positive vote, while B gives C a total of thirty positive votes. This yields the following adjacency matrix.

Table 5.5.

↗	A	B	C
A	0	0	1
B	0	0	30
C	0	0	0

(a) Adjacency Matrix

↗	A	B	C
A	0	0	1
B	0	0	1
C	0.333	0.333	0.333

(b) ..converted to a row stochastic matrix

Converting this matrix to a row stochastic matrix causes a loss of information. *Smoothing* tries to alleviate this by adding data points for every voting type. Let m be the number of data points added (the more artificial points are added, the stronger is the smoothing), d be the number of voting types, and u_i the total count of positive votes a student i has given. In the following example, $m = 1$ because it is the default value suggested by the author of BRA [Poh15] and $d = 2$ because there are two voting types: positive and negative votes. For each student, an activity coefficient a_i is then calculated:

$$a_i = \frac{u_i + m}{u_i + (m \cdot d)}$$

This yields the following rating activity coefficients.

$$\begin{aligned} a_A &= \frac{1+1}{1+(1\cdot 2)} &&= \frac{2}{3} &&= 0.666 \\ a_B &= \frac{30+1}{30+(1\cdot 2)} &&= \frac{31}{32} &&= 0.968 \\ a_C &= \frac{0+1}{0+(1\cdot 2)} &&= \frac{1}{2} &&= 0.500 \end{aligned}$$

These coefficients are then applied to the adjacency matrix. Let $b_{i,j}$ be the entries of the adjacency matrix and n be the number of students, then each entry of the new *estimated praise probability distribution matrix* $r_{i,j}$ is calculated using the following equation:

$$r_{i,j} = a_i \cdot b_{i,j} + (1 - a_i) \cdot \frac{1}{n}$$

The resulting matrix takes the total rating count of each student into account by uniformly adding stronger probabilities when few votes have been cast. Therefore, B has a higher adjacency value towards C than A.

Table 5.6.: Estimated Praise Probability Matrix

↗	A	B	C
A	0.111	0.111	0.777
B	0.011	0.011	0.979
C	0.333	0.333	0.333

BRA then constructs an *Estimated Praise Probability Matrix*. Let $B \in (0;1)^{n\times n}$ be the stochastic adjacency matrix as seen in 5.2, $A \in (0;1)^{n\times n}$ be a matrix with the activity coefficients arranged diagonally, $\mathbb{1}$ be the $n \times n$ identity matrix and

$$I \in (0;1)^{n\times n} = \begin{pmatrix} \frac{1}{n} & \cdots & \frac{1}{n} \\ \vdots & \ddots & \vdots \\ \frac{1}{n} & \cdots & \frac{1}{n} \end{pmatrix}$$

then the *Estimated Praise Probability Matrix* $R \in (0;1)^{n\times n}$ is formulated as

$$R = A \times B + (\mathbb{1} - A) \times I$$

Like Weighted PageRank, the dominant left eigenvector of R (containing the values $r_{i,j}$) yields the reputation scores.
Given the example from 5.1, B would be equal to the adjacency matrix given in 5.2. The rating activity coefficients contained in A (using $m = 1$ and $d = 2$), arranged diagonally, are as follows.

Table 5.7.: Rating Activity Coefficient Matrix

↗	A	B	C	D
A	0.75	0	0	0
B	0	0.5	0	0
C	0	0	0.667	0
D	0	0	0	0.857

The calculated *Estimated Praise Probability Matrix* differs slightly from Weighted PageRank's final matrix as seen in 5.3.

Table 5.8.: Estimated Praise Probability Matrix

↗	A	B	C	D
A	0.063	0.438	0.438	0.063
B	0.292	0.125	0.292	0.292
C	0.083	0.750	0.083	0.083
D	0.207	0.036	0.721	0.036

The normalized reputation scores calculated by BRA, while slightly different to Weighted PageRank's results, yield the same order. As expected, B's reputation drops a little bit, while C's reputation gains a little bit. This is easily explained by

the fact that C received a lot of votes from D. Those were reinforced by the rating activity coefficient. However, it was not enough to change the overall ranking. Even when D's vote count for C was increased drastically, the ranking is not changed. In chapter 5.3, more complex evaluation tests will show whether these differences have a visible impact on the ranking.

Table 5.9.: Reputation scores as calculated by the BRA

A	0.174
B	0.363
C	0.315
D	0.148

Although BRA seems to be designed for multiple voting types, as the author sets $d = 2$ corresponding to positive and negative votes, they also claim that their *estimated praise probability matrix* must be strictly positive. As the name suggests, it is a matrix containing the possibilities for **praise**. Thus, it can be said that BRA currently makes no use of negative votes apart from using it in the rating coefficient, where the actual number of negative votes a student has given a peer is not taken into account.

5.1.3. EigenTrust

EigenTrust is a reputation algorithm devised by Kamvar et al., designed to identify untrustworthy peers in P2P file sharing networks [KSGM03]. It is very popular in research, being cited more than 4000 times according to Google Scholar as of June 2016 [Goo16]. Its popularity among reputation algorithm testbed approaches (e.g. [JHSMT13],[Lag12]) qualifies EigenTrust as an interesting evaluation candidate. Although also not tailored for classroom application, the notion of trust in P2P networks can be regarded as similar to reputation in a classroom. A trusted peer provides high quality, legit files, a reputable student provides high quality posts and correct votes.
In EigenTrust, every peer i stores a list of *experiences* with other peers j that can be satisfactory (positive vote) or unsatisfactory (negative vote). In the first step, these experiences are aggregated. The number of unsatisfactory experiences is subtracted from the number of satisfactory experiences.

$$s_{ij} = satisfactoryCount(i,j) - unsatisfactoryCount(i,j)$$

These values are then normalized. Although negative votes are supported, an entry can not be negative.

$$c_{ij} = \frac{max(s_{ij},0)}{\sum_j max(s_{ij},0)}$$

If a peer has not made any experiences yet, EigenTrust assumes that there is a list of peers provided by the network that can be trusted by default which would make

up for the missing experiences. There are a few options to handle this in a classroom scenario: Some of the top performing students could be listed as trustworthy by default. However, at the beginning of a course, it is not likely that such a 'meta-reputation' has already been established. Thus, in our case, a student trusts everyone equally if no experiences have been made yet.
The values c_{ij} can be inserted in a matrix C at position ij, yielding the same stochastic right matrix as seen in the starting point matrix for Weighted PageRank (cf. 5.2). Now, a vector t_0 is initialized with uniform probabilities e for every peer. The basic EigenTrust algorithm multiplies C^T with t until $||t^{k+1} - t^k||$ is smaller than a predefined error threshold.

$t_0 = e$
do

$$t_{n+1} = C^T t_n \tag{5.1}$$
$$\delta = ||t_{n+1} - t_n||$$

until $(\delta < error)$

With an error value of 0.2, EigenTrust yields the following reputation scores for example 5.1:

Table 5.10.: Reputation scores as calculated by EigenTrust

A	0.141
B	0.421
C	0.328
D	0.110

5.1.4. Aggregate

In order to compare the more complex eigenvector-based algorithms with a very simple approach, an algorithm was devised that will be referred to as *Aggregate*. Aggregate simply sums up all votes a student's artifacts have received. The resulting aggregation value can be positive or negative. Then, the values of all students are normalized so that they are within the range of $[0,1]$. These values are the reputation scores.
Given example 5.1, the matrix seen in 5.1 is first transposed, then the sum of each row is calculated:

$$\begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 4 & 0 \end{pmatrix}^T \times \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 4 \\ 0 & 0 & 0 & 0 \end{pmatrix} \times \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 5 \\ 0 \end{pmatrix}$$

The normalized value is the final reputation score:

Table 5.11.: Reputation scores as calculated by EigenTrust

A	0.125
B	0.250
C	0.625
D	0.000

Because all eigenvector-based algorithms consider the 'weight' of a vote based on the caster's reputation and Aggregate does not, Aggregate is the only algorithm that ranks C higher than B in this example.

Table 5.12.: Ranking Comparison

	W-PageRank	BRA	EigenTrust	Aggregate
1st	B (0.376)	B (0.363)	B (0.421)	C (0.625)
2nd	C (0.312)	C (0.315)	C (0.328)	B (0.250)
3rd	A (0.168)	A (0.174)	A (0.141)	A (0.125)
4th	D (0.144)	D (0.148)	D (0.110)	D (0.000)

5.2. Quality Measures

In order to quantify the performance of each algorithm in a test and enable comparability, three quality measures with different properties were devised.
Before a test starts, every student s is assigned to a specific group $\mathbf{G}_n$ by the test programmer according to his or her expected ranking position. These groups are numbered consecutively and members of groups with a higher index are expected to have better ranking positions than those with lower indexes.
As a result, every student of a group has a *minimum expected ranking position* $min_{rp}(s)$ and a *maximum expected ranking position* $max_{rp}(s)$.

Definition 5.2.1 (Minimum Expected Ranking Position)**.** Let $s \in G_n$, then

$$min_{rp}(s) = \sum_{i=0}^{n-1} |G_i|$$

Definition 5.2.2 (Maximum Expected Ranking Position)**.** Let $s \in G_n$, then

$$max_{rp}(s) = \sum_{i=0}^{n} |G_i|$$

Given three groups of *malicious students* G_0, *average students* G_1 and *top performing students* G_2, containing respectively 2, 4 and 2 students. An intuitive result is given when the malicious students rank worst, the top performing students rank best and the average students rank in between.
$min_{rp}(s \in G_0) = 0$ and $max_{rp}(s \in G_0) = 2 - 1 = 1$, so the expected position for a malicious student is within $[0, 1]$.
$min_{rp}(s \in G_1) = 2$ and $max_{rp}(s \in G_1) = 2 + 4 - 1 = 5$, so an average student can expect to occupy a position within $[2, 5]$.

Definition 5.2.3. (Correctness) An ordering of n students s_1 to s_n produced by an algorithm $\mathbb{A}$ has a high correctness, when many students have an actual reputation ranking position within his or her expected position range $[min_{rp}(s), max_{rp}(s)]$. Let $actualPosition(s)$ be the actual position of a student s in a ranking.

$$correct(s) = \begin{cases} 1 & min_{rp}(s) \leq actualPosition(s) \leq max_{rp}(s) \\ 0 & otherwise \end{cases}$$

$$correctness(\mathbb{A}) = \frac{\sum_{i=1}^{n} correct(s_i)}{n}$$

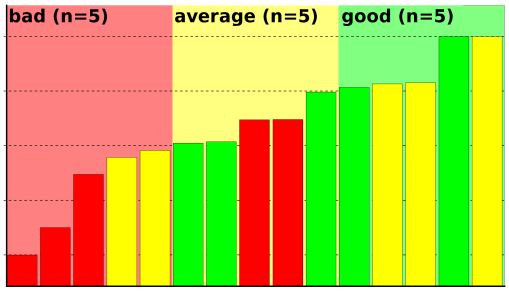


Figure 5.2.: A low correctness with many unfitting reputation values. Correctness = 0.33

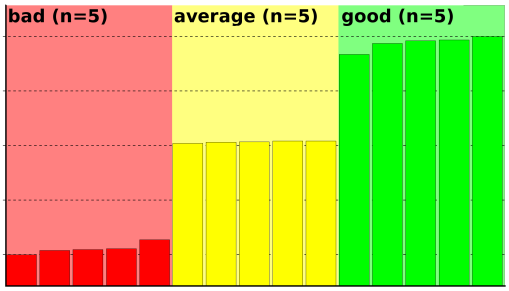


Figure 5.3.: High correctness representing the actual scenario. Correctness = 1.00

A perfect reputation algorithm would rank every student in a way that his or her reputation score represents his or her behavior in a reasonable way, i.e. every bar of a color is positioned within the area of corresponding color. This leads to a correctness of 1.0. The downside of this measurement is that there are situations in which correctness is always close to 1.0, reducing it's usefulness. An example would be a scenario with 99 good students and one bad student. In worst case, correctness would be $\frac{99}{100}$, but an algorithm that ranks said bad student second-lowest would still be better than an algorithm ranking the bad student as the top student. This is however not reflected by correctness.
Thus, a second quality measure borrowed from discrete mathematics is introduced to compensate for this shortcoming, based on the concept of *inversions* [VF90]. In contrast to the original definition, the numbers do not have to be distinct as to incorporate the concept of equal group affiliations.

Definition 5.2.4. (Inversion) Let $(A(1),...,A(n))$ be a sequence of n numbers. If $i > j$ and $A(i) < A(j)$, then the pair (i, j) is called inversion of A.

Definition 5.2.5. (Inversion Number) The inversion number is the count of inversions in a sequence. [BMJ04]

$$inv(A) = |\{(A(i), A(j)) | i < j \quad and \quad A(i) > A(j)\}|$$

Based on the reputation ranking order, a sequence $\mathbb{G}$ can be filled with the corresponding indexes i of the groups G_i a student belongs to. Assume a ranking

(s_1, s_3, s_2, s_4) with $s_1, s_2 \in G_1$ and $s_3, s_4 \in G_2$, the constructed sequence would be $(1,2,1,2)$, so two inversions. A low inversion number indicates a good reputation algorithm: the right reputation ranking would yield a sequence like $(1,1,2,2)$, having 0 inversions.

Definition 5.2.6. (Inversion Quality) An ordering of n students produced by an algorithm $\mathbb{A}$ has a high inversion quality when few students belonging to a worse performing group are ranked higher than students belonging to a better performing group and vice versa. If there are x groups, then

$$inversionQuality(\mathbb{A}) = 1 - \frac{inv(\mathbb{G})}{\frac{n\cdot(n-1)}{2} - \sum_{i=0}^{x-1} \frac{|G_i|\cdot(|G_i|-1)}{2}}$$

To eliminate group size as a factor, the inversion number is divided by the total number of possible permutations while considering that permutations within a group can be disregarded.

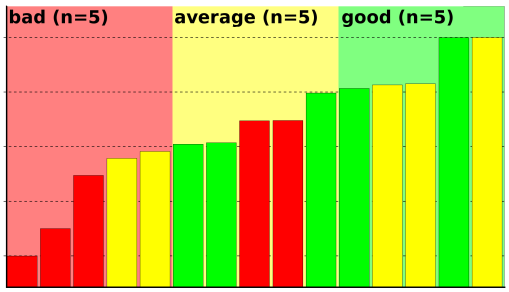


Figure 5.4.: Many lower performing students outrank the top performers. *inversionQuality* = 0.77

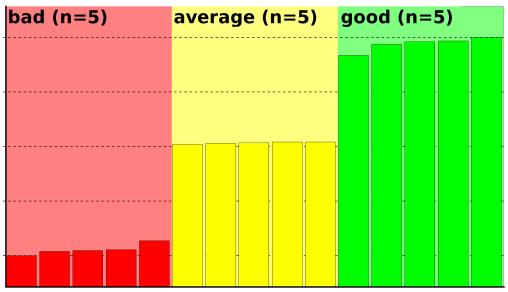


Figure 5.5.: High inversionQuality representing the actual scenario. *inversionQuality* = 1.00

The third quality measure *Distinction* concerns itself with how clearly each group of student types is distinguished from other groups. Strongly separated groups are interesting because they can be clustered easily and automatically. Identifying groups of students adds further merit to a reputation algorithm, as it can be used to counter attacks (e.g. by automatically blocking malicious agents) or, in context of a classroom, identify social structures such as friendship groups. A high distinction between groups could also be an indicator of the certainty of an algorithm about its results.

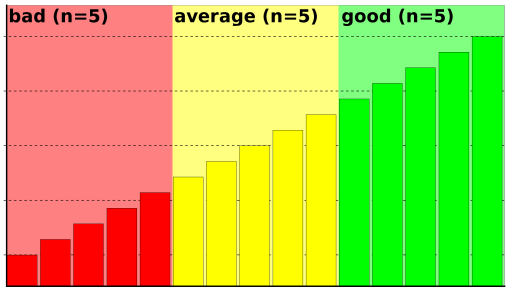


Figure 5.6.: Groups not distinct, can not be identified afterwards. *Distinction* = 1.0

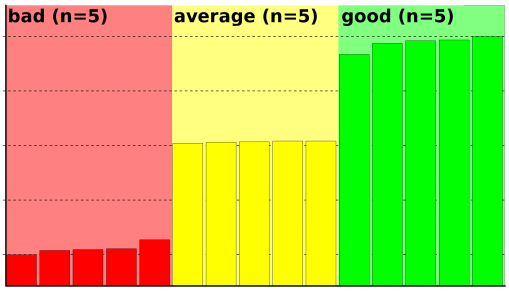


Figure 5.7.: Groups are highly distinct, clustering would be easy. *Distinction* = 6.1

Definition 5.2.7. (Distinction) The mean difference between adjacent scores within the ranking is calculated. Between group 'borders', the score difference is divided by the mean difference. The resulting factor is called *Distinction*.

A *Distinction* value that is lower or equal to 1 makes clustering very hard, as there are no distinct borders between groups (cf. Section 5.6).

Table 5.13.: Quality Measure Comparison

	Correctness	**Inversion Quality**	**Distinction**
Range	$[0,1]$	$[0,1]$	$[0,+\infty[$
Measures	Correct zone assignment	Correct ranking order	Confidence of assignment
Drawback	Uninformative when groups' sizes are extremely uneven	Less informative as groups' sizes grow	Only sensible when $correctness = 1.0$

5.3. Tests

In order to demonstrate the simulator's capabilities and functionality, as well as get an apprehension of the quality of the four selected reputation algorithms, seven tests have been designed. The results are evaluated in this section. To reduce reputation score noise produced by the stochastic behavior of the agents in the simulator, each test, except the Competence Convergence Test, ran 2000 times. The reputation scores were aggregated from every test round, and divided by the test round count afterwards. The test round count of 2000 was experimentally determined: None of the quality measures changed greatly upon further increase of the test round count. Apart from the Matthew Test and the Competence Convergence Test, every test allowed the simulation to run for 100 iterations. As the Matthew Test requires the establishment of commonly known top students, it was allowed to run for 200 iterations. Unless specified otherwise, *Activity* was always set to **1.0**, so the agents are never idle, and *Positivity* was set to 0.5, so negative and positive votes occur equally often. The rest of the parameters are, by default, set to zero if not explicitly specified.

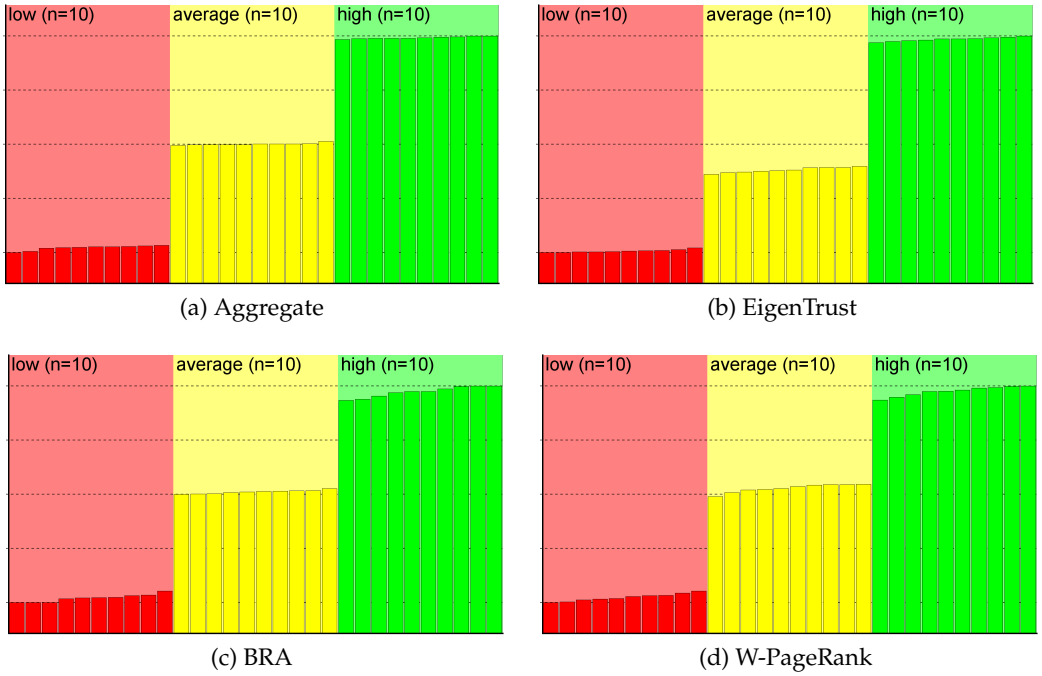
5.3.1. Competence Test

The first test examines whether students that are more likely to produce quality artifacts as well as vote more correctly receive a higher reputation score, which is common sense. The test specifies three student groups with ten students each that only differ by their competence value. Every student had the same *Productivity*, which was set to **0.2**, meaning approximately every fifth action results in the production of an artifact

The most competent student group 'high', depicted in green color, had *Competence* set to **0.9**, the average performers 'average' (orange) had their *Competence* set to **0.7** and the least competent group 'low' (red) received a *Competence* value of **0.5**.

Table 5.14.: Competence Test — Agent Settings

Name	low	average	high
Color	red	yellow	green
Group Size	10	10	10
Activity	1.0	1.0	1.0
Productivity	0.2	0.2	0.2
Positivity	0.5	0.5	0.5
Competence	0.5	0.7	0.9



(a) Aggregate (b) EigenTrust

(c) BRA (d) W-PageRank

Figure 5.8.: Competence Test Results

As can be seen, every algorithm managed to ace the test with not a single error of judgement.

Table 5.15.: Competence Test Quality Measures

	Aggregate	EigenTrust	BRA	W-PageRank
Correctness	1.0	1.0	1.0	1.0
Inversion Quality	1.0	1.0	1.0	1.0
Distinction	14.0	13.7	12.6	12.4
Time	81.2s	88.9s	99.4s	93.3s

However, although the distinction is very high for all algorithms, it can be seen that the more an algorithm takes negative votes into account, the higher its distinction value is. While Weighted PageRank does not incorporate for negative votes and BRA uses them only to smooth the rating coefficient, EigenTrust at least subtracts negative votes from the positives, however, it sets all negative entries in its vote matrix to zero before reputation score calculation. Finally, Aggregate allows for negative vote aggregations, thus it benefits a little in this basic scenario.

5.3.2. Activity Test

In this test, it is investigated whether equally competent and productive students are ranked higher when they contribute more actively, i.e. vote and post more than their less active peers. Although less intuitive than for the competence test,

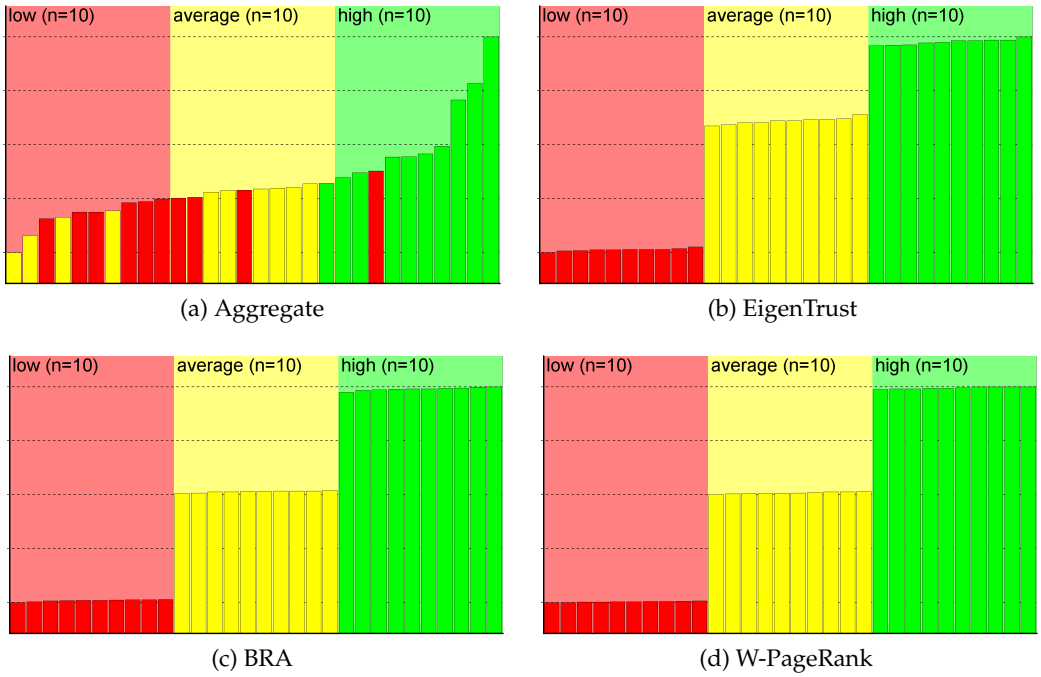
it is expected that more avid contributors will be ranked higher by a reputation algorithm. Again, three groups of ten students were set up for the simulation.
The following attributes were the same for every student:

- *Competence* was set to **0.7**, describing well-informed students that sometimes make mistakes
- *Productivity* was set to **0.4**, resulting in the production of an artifact about every fifth action

The *Activity* value for the three groups 'high', 'average' and 'low', representing highly active, average and inactive students, was set respectively to **1.0**, **0.6** and **0.2**.

Table 5.16.: Activity Test — Agent Settings

Name	low	average	high
Color	red	yellow	green
Group Size	10	10	10
Activity	0.2	0.6	1.0
Productivity	0.2	0.2	0.2
Positivity	0.5	0.5	0.5
Competence	0.7	0.7	0.7



(a) Aggregate (b) EigenTrust

(c) BRA (d) W-PageRank

Figure 5.9.: Activity Test Results

Apart from Aggregate, which only manages a correctness of 0.7 (and hence, a non-descript distinction value), the eigenvector-based algorithms produce a result that

strongly reflects the initial intuition. In this test, BRA and Weighted PageRank actually deliver a stronger distinction than EigenTrust. This might be because in an environment with rather competent students, there might be less reason and fewer occasions to vote negatively and thus a focus on positivity and praise leads to less noise. Also, BRA's activity-based reinforcement seems to have a small yet existent effect on the distinguishability of the results.

Table 5.17.: Activity Test Quality Measures

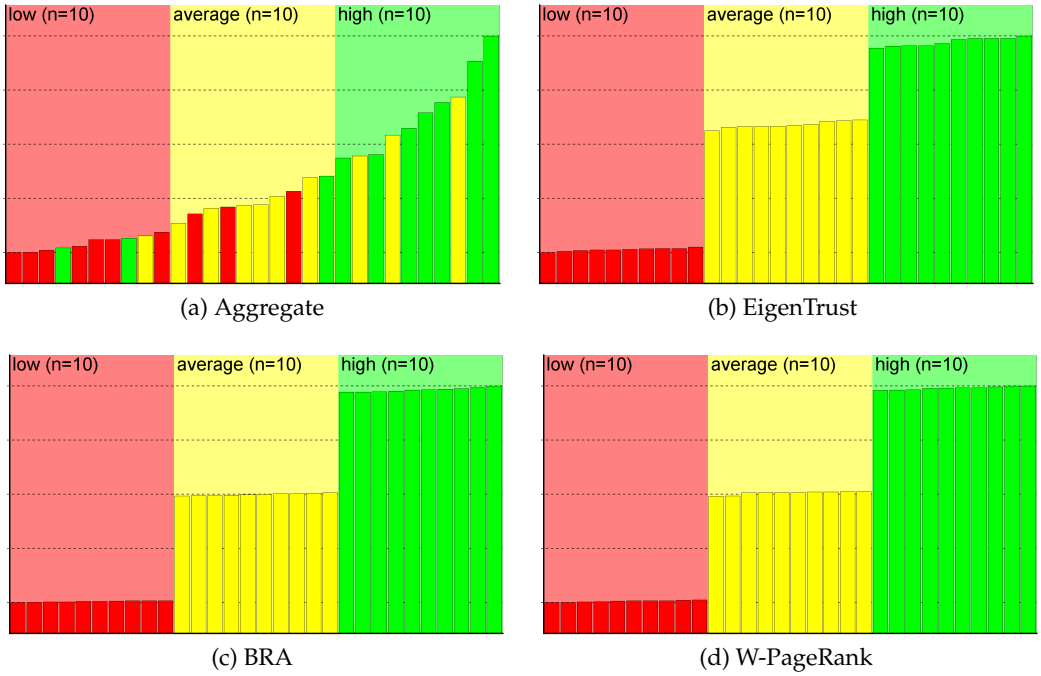
	Aggregate	EigenTrust	BRA	W-PageRank
Correctness	0.7	1.0	1.0	1.0
Inversion Quality	0.84	1.0	1.0	1.0
Distinction	0.46	12.6	14.1	14.0
Time	32.1s	39.5s	48.8s	49.1s

5.3.3. Productivity Test

The more basic tests are concluded with the Productivity Test, which investigates what kind of effect different levels of productivity, all other parameters being equal, have on the reputation score. Intuitively, given competent students, a higher productivity (more good posts and questions, but less opportunities to vote for others) seems preferable and thus should have a higher reputation. Although voting and thus assessing quality might very well be very important, generating (quality) content is the basis for votes.

As with the other tests, three student groups of highly productive students ('high'), average and unproductive students (respectively 'average' and 'low'), ten students each, were set up. The competence value was the same for every student. It was set to **0.95**, portraying a very excellent community which members rarely make mistakes.

Productivity was set to **0.8** for members of the 'high' group, **0.5** for members of the 'average' group and **0.2** for members of the 'low' group.



(a) Aggregate (b) EigenTrust

(c) BRA (d) W-PageRank

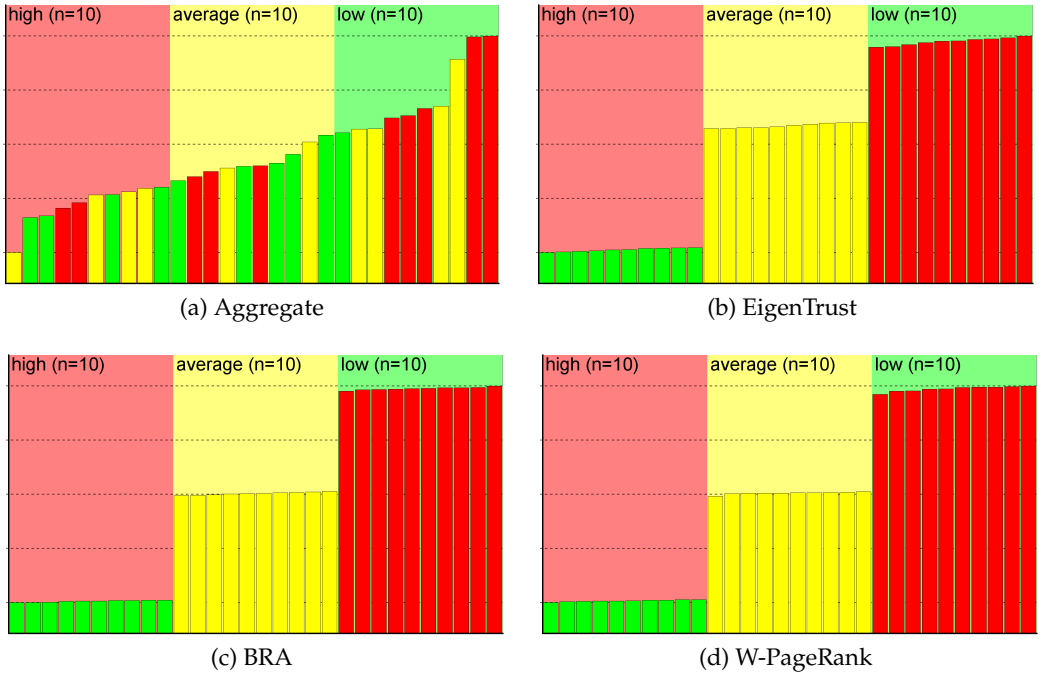
Figure 5.10.: Productivity Test Results, high competence

Again, Aggregate has some difficulties making a distinguished prediction, whereas its more complex eigenvector counterparts deliver results with high confidence. BRA's and Weighted PageRank's distinction advantage could, again, be due to the high overall competence in the classroom, making negative votes more of a noise than a necessity.

Table 5.18.: Productivity Test, high *Competence* — Quality Measures

	Aggregate	EigenTrust	BRA	W-PageRank
Correctness	0.66	1.0	1.0	1.0
Inversion Quality	0.82	1.0	1.0	1.0
Distinction	1.8	13.0	14.2	14.2
Time	108.6s	115.8s	125.6s	120.2s

Reputation can only be assessed correctly when the data it works on has at least a certain quality. For this second test, the *Competence* value of every student is set to **0.1**, representing a classroom with either very unmotivated or inexperienced students. Intuitively, producing more unmotivated artifacts of bad quality should result in a lower ranking. The ranking should thus be reversed, with the highly active 'high' group being ranked lowest and so on.



(a) Aggregate (b) EigenTrust

(c) BRA (d) W-PageRank

Figure 5.11.: Productivity Test Results, low competence. Expected ranking zones are reversed: the red group 'high' produces a lot of artifacts of bad quality and should thus be ranked lowest and so on.

Table 5.19.: Productivity Test — Agent Settings

	High Competence			Low Competence		
Name	low	average	high	low	average	high
Color	red	yellow	green	green	yellow	red
Group Size	10	10	10	10	10	10
Activity	1.0	1.0	1.0	1.0	1.0	1.0
Productivity	0.2	0.5	0.8	0.2	0.5	0.8
Positivity	0.5	0.5	0.5	0.5	0.5	0.5
Competence	0.95	0.95	0.95	0.1	0.1	0.1

As can be seen, the algorithms' results remain unchanged compared to when *Competence* was set to a high value. In a community where no member can assess peers' content correctly, the results will not reflect the real scenario truthfully. This second test shows that some ranking results can easily be misinterpreted. It might be necessary to determine if there is at least a certain amount of goodwill and competence existent in a user base, e.g. by having a competent person (a teacher) look through posted content occasionally. Additionally, having tutors and teachers take active part in voting and posting content might alleviate this problem.

Table 5.20.: Productivity Test, low *Competence* — Quality Measures

	Aggregate	EigenTrust	BRA	W-PageRank
Correctness	0.16	0.33	0.33	0.33
Inversion Quality	0.36	0	0	0
Distinction	0.6	13.5	14.1	13.9
Time	105.3s	112.4s	122.4s	116.7s

However, it could be argued that in a classroom context, even with low competence, high activity might still be good, e.g. when someone is asking a lot of (seemingly dumb) questions, he or she is still learning and contributing to the learning process of others, who might profit of the asked questions themselves. This is actually a limitation of the simulator, which does not differentiate between questions and posts. Thus, a 'bad artifact' in this test should rather be interpreted as a misleading post or a piece of misinformation. It could also be reasoned that asking good questions also requires competence.

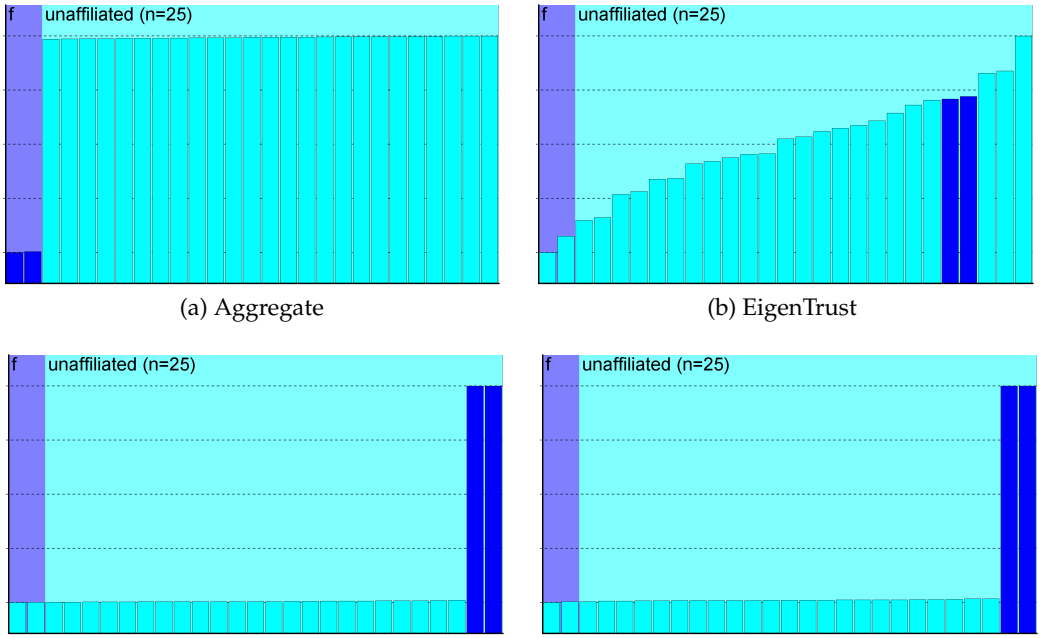
5.3.4. Coordinated Friends Test

In this test, it is evaluated how many friends who are significantly less competent than their peers are required to trick the system so that they reach top position in the reputation ranking. The members of the group 'f' (friends, depicted by a dark blue color) try to achieve this by voting each other up no matter the quality of their artifacts. The test starts with group 'f' having two members, and group 'unaffiliated' having 28 members (cyan color). *Productivity* was set to **0.3** for all students.

The group of unaffiliated students without any ties to each other had a *Competence* of **0.9**, whereas the friendship group's members had a competence of only **0.2**. The friends had their group label added to their list of groups they like, as well as to the list of groups they belong to themselves. Their *Loyality* was set to 1.0, so every time they decide to vote, they vote a friend's artifact positively.

Table 5.21.: Coordinated Friends Test — Agent Settings

Name	f	unaffiliated
Color	cyan	dark blue
Group Size	2	25
Activity	1.0	1.0
Productivity	0.3	0.3
Positivity	0.5	0.5
Competence	0.9	0.2
Loyality	1.0	0
Affiliations	['f']	[]
ownGroups	['f']	[]



(a) Aggregate (b) EigenTrust

(c) BRA (d) W-PageRank

Figure 5.12.: Coordinated Friends Test Results, 2 friends. Dark blue represents the friendship group, the unaffiliated group of students is represented with a cyan color.

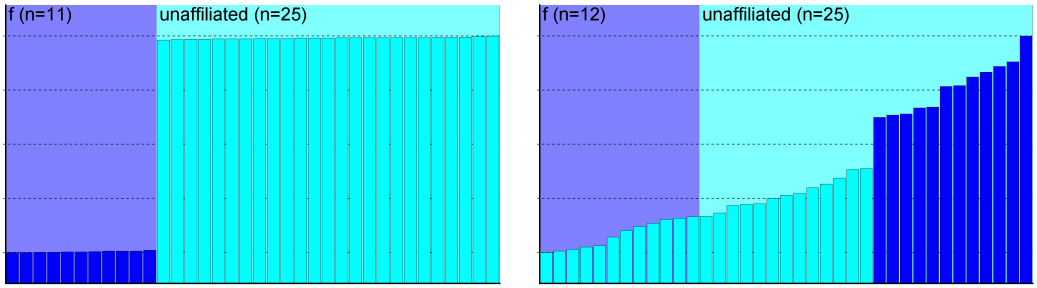
As can be seen, even with only two friends upvoting each other, the eigenvector-based algorithms are tricked and the friends get the highest scores by a great margin. This is probably reinforced by the insufficient negative vote consideration of Weighted PageRank and BRA — even when the friend group's members produce artifacts of bad quality, the peers' bad votings are not taken into account. EigenTrust however, which includes negative votes to a certain threshold, has a less clear result. The only algorithm that assesses the situation correctly is Aggregate. This can most likely be explained because it fully considers negative votes, thus the unaffiliated peers that can recognize the bad quality of the friends' artifacts have a strong impact. Furthermore, the total vote count is pivotal. With only two people, the friend group members might simply not be able to cast enough votes to surpass the unaffiliated students.

This test also demonstrates the weakness of the correctness quality measure, which is extremely uneven group sizes. Even though the eigenvector-based algorithms have a worst-case correctness, it still seems rather high with a value of 0.85. In this case, the inversion quality measure reveals much more information.

Table 5.22.: Coordinated Friends, Quality Measures

	Aggregate	EigenTrust	BRA	W-PageRank
Correctness	1.0	0.85	0.85	0.85
Inversion Quality	1.0	0.12	0	0
Distinction	26.5	1.9	0.005	0.003
Time	59.3s	65.8s	71.9s	68.6s

As for Aggregate, the friend group size can be increased up to eleven before it fails to give a correct assessment. This strong resistance against collaboration 'attacks' further backs the recommendation to employ an algorithm that takes negative votes fully into account when it comes to classroom scenarios, especially if the number of students is relatively small.



(a) Aggregate, 11 friends

(b) Aggregate, 12 friends

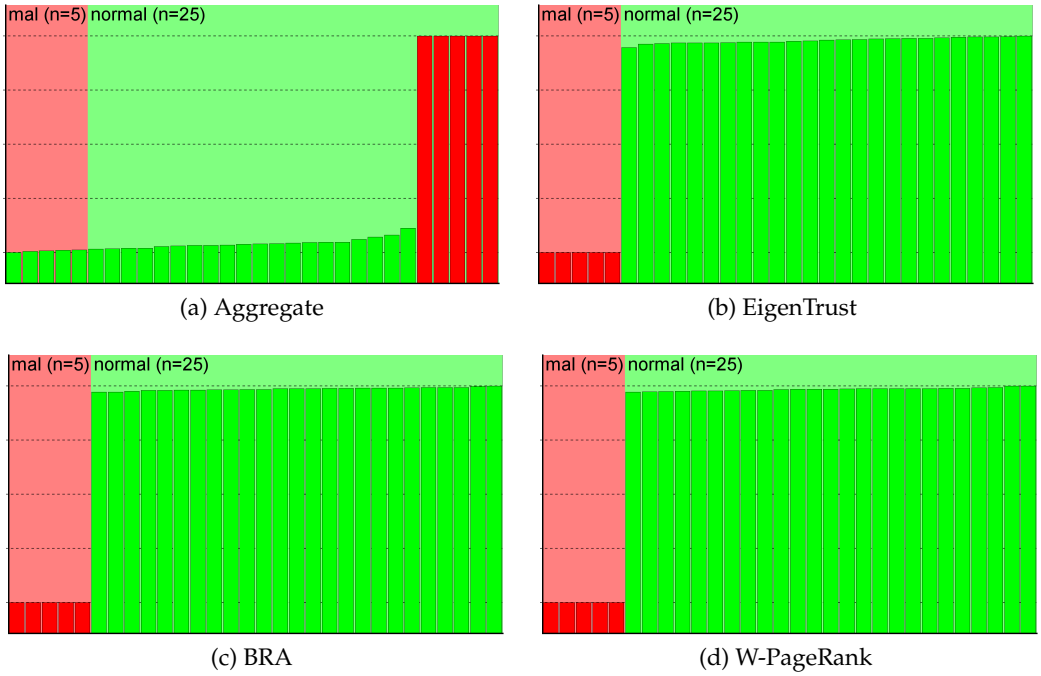
Figure 5.13.: Aggregate suddenly fails correct assignment when increasing from 11 to 12 friends.

Even though this result may convey the idea that especially Weighted PageRank can be tricked too easily, it is important to keep in mind that PageRank was developed for the web, which has a vast amount of websites. In this scenario, even just two friends already made up almost 7% of the community. The number of websites required to collaborate to trick PageRank is probably impractically high, given the total amount of websites. According to De Kunder, the internet is estimated to have at least 4 billion pages as of June 2016 [DK16].

5.3.5. Collusion Test

The implemented simulator is also able to recreate classic attack scenarios, such as the collusion attack, which is attackers following a mutual malicious strategy in order to trick the system. A typical case is that these malicious agents vote other peers negatively without contributing themselves, so that they gain a better reputation score relative to their peers. In the classroom, it is possible that there might be students trying to challenge the system for fun. These attackers should still be ranked lower than normal contributors.
The first scenario introduces five attackers in the group 'mal' (red) and 25 normal students that are added in the group 'normal' (normal). A *Competence* value of **0.8**

was applied to all students, representing a well-informed classroom.
For the normal students, *Productivity* was set to **0.3** and *Positivity* to **0.5** for no voting bias. The attackers' *Productivity* was set to zero, so they never contribute anything. Their maliciousness was set to 1.0, so they always choose to irrationally vote someone disregarding of his or her artifact quality. *Positivity* was set to zero, so they always cast a bad vote.



(a) Aggregate (b) EigenTrust

(c) BRA (d) W-PageRank

Figure 5.14.: Collusion Test, 5 attackers

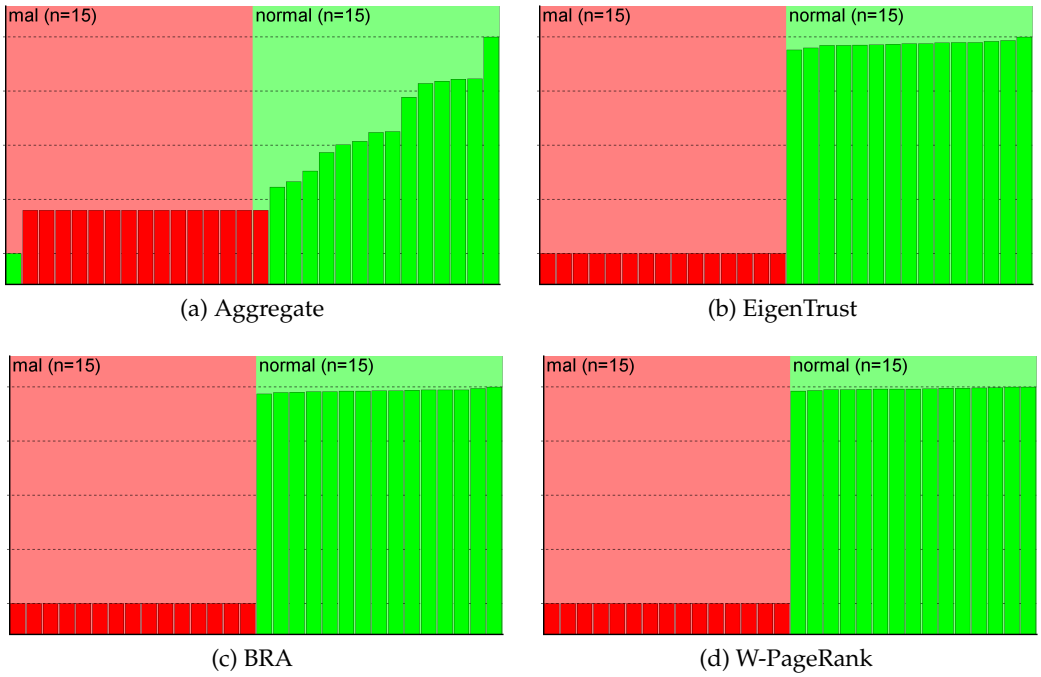
Every tested algorithm except Aggregate shows a very high resistance against the attack. Two factors might explain this: Firstly, the non-existent to limited consideration of negative votes actually provides an inherent resistance against this attack, which is based on negative votes. This could also explain why Weighted PageRank and BRA have a slightly higher distinction value than EigenTrust. Secondly, the property of transitivity of trust that is an inherent feature of the eigenvector-based algorithms might reinforce their good position. As the attackers never contribute an artifact, they cannot receive any ratings and therefore can not build reputation.

Table 5.23.: Collusion Test with 5 attackers: Quality Measures

	Aggregate	EigenTrust	BRA	W-PageRank
Correctness	0.66	1.0	1.0	1.0
Inversion Quality	0	1.0	1.0	1.0
Distinction	0.11	28.4	29.1	29.1
Time	59.1s	67.0s	76.5s	71.5s

Interestingly, an increase of attackers actually weakens their position in the ranking.

When there are as many attackers as normal students, Aggregate can detect them, and partially ranks them lower than the normal students. The inversion quality increases from 0 to 0.93 (cf. Figure 5.24). However, separation between normal and malicious students is less clear. Also, the other algorithms are not overwhelmed by the increase of attackers and assess the situation correctly and with high confidence.



(a) Aggregate (b) EigenTrust

(c) BRA (d) W-PageRank

Figure 5.15.: Collusion Test, 15 attackers

Table 5.24.: Collusion Test with 15 attackers: Quality Measures

	Aggregate	EigenTrust	BRA	W-PageRank
Correctness	0.93	1.0	1.0	1.0
Inversion Quality	0.93	1.0	1.0	1.0
Distinction	0	28.2	29.1	29.4
Time	41.6s	49.7s	58.1s	53.2s

Table 5.25.: Collusion Test — Agent Settings

Name	normal	mal
Color	green	red
Group Size	25 | 15	5 | 15
Activity	1.0	1.0
Productivity	0.3	0
Positivity	0.5	0
Competence	0.8	0.8
Maliciousness	0	1.0

5.3.6. Matthew effect

With this test, the impact of the Matthew effect on the ranking is investigated. The Matthew effect describes the phenomenon that that students who have been top-performing in the past generally receive better votes because their peers trust that their contributions are of good quality [Rig13]. The vote is then cast without a thorough assessment of the actual quality of the artifact. Intuitively, enabling the Matthew effect should reinforce the top students position even further. The distinction quality measure should thus increase visibly.
For this test, 20 normal students in the group 'average' (yellow) were set up along with three top-performing students in the group 'top'. All students had a *Productivity* of **0.4** and a *Social Positivity* of **1.0**, so the reverse Matthew effect (downvoting commonly known bad students' artifacts because it is assumed their contributions are always of bad quality) was disabled. The members of the 'average' group received a *Competence* value of **0.5**, while the 'top' members' was set to **0.9**. *Maximum number of best students* was set to **3**, because there were three members in the 'top' group. One test was run with the Matthew effect disabled (*Influenceability* set to zero), while the other one ran with activation of the Matthew effect after 100 simulation iterations. After said iterations, *Influenceability* was set to **0.5**, so every second positive vote by a member of the 'average' group was given explicitly to one of the best three students in the ranking yielded by the preceding simulation iteration.

Table 5.26.: ☒= Matthew effect disabled, ☑= Matthew effect enabled

	Aggregate		EigenTrust		BRA		W-PageRank	
	☒	☑	☒	☑	☒	☑	☒	☑
Correctness	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Inv. Quality	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Distinction	21.4	22.3	20.4	22.1	18.7	22.1	18.8	22.0
Time	293.6s	261.4	325.7s	296.0	335.9s	305.2	329.0s	301.0s

For the test with Matthew effect enabled, by end of simulation iteration 100, the simulator reports that there is a significant increase of false positives (good votes for bad artifacts), proving that the Matthew effect works as expected (cf. Figure

5.16). As expected, the top students gain an additional ranking score boost, and the distinction increases. Interestingly, with the Matthew effect enabled, the distinction values do not differ much between the algorithms, while there is more of a difference when the effect is disabled. This might possibly be because after the activation of the Matthew effect, the fraction of positive votes increases, reducing the effect of full negative vote consideration.

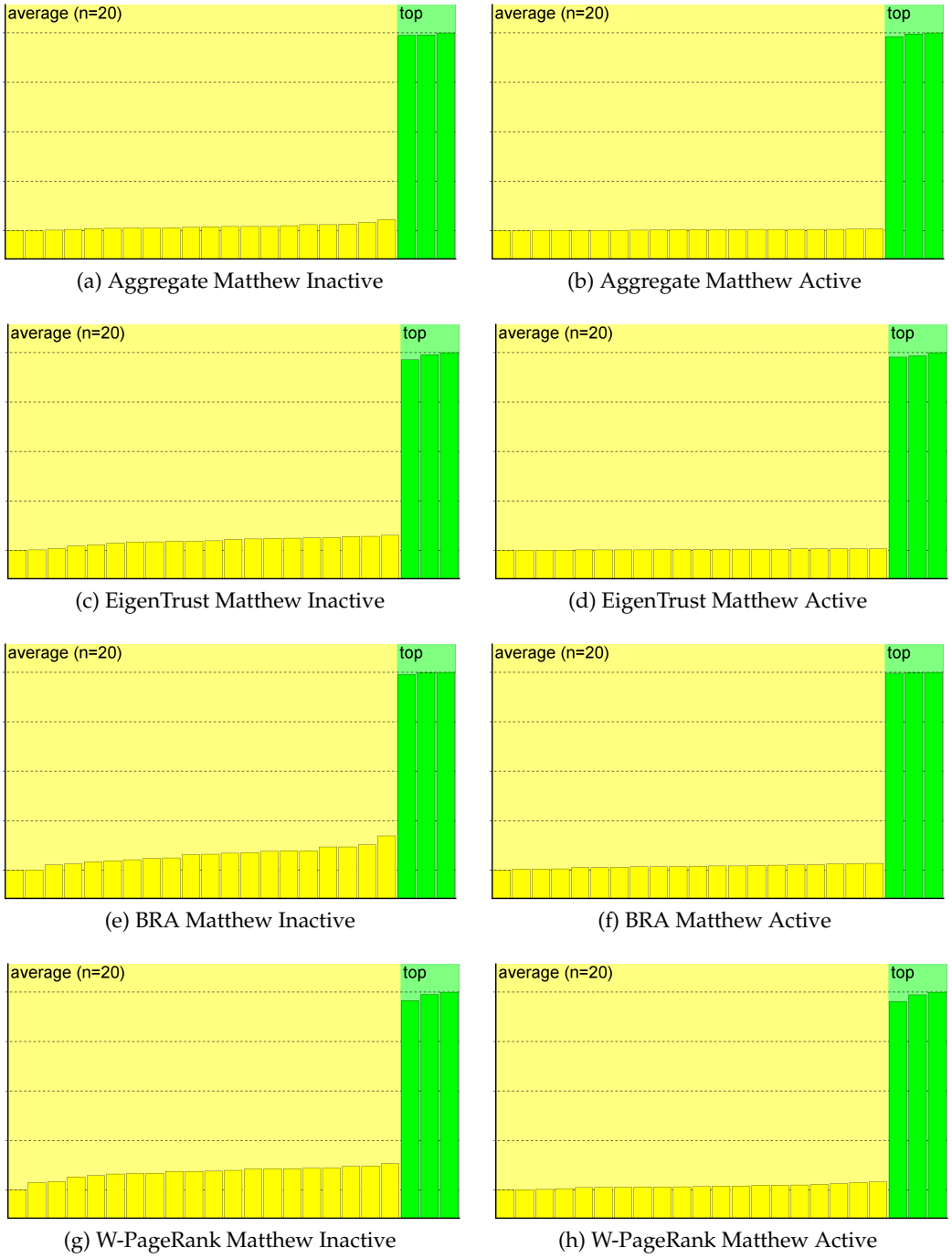
Reducing the *Competence* gap between 'top' and 'average' students drastically diminishes the distinction (cf. Table 5.27). When the test was rerun with a *Competence* value of **0.7** for an 'average' student and **0.8** for a 'top' student, distinction with Matthew effect enabled was worse for all algorithms compared to with Matthew effect disabled. A possible explanation could be that, given the small *Competence* difference of only **0.1**, the algorithms converge too slowly and thus, students belonging to the 'average' group get selected as top students, and are then pushed to the top. In the averaged result, this phenomenon could result in more noise and thus less distinction. This is supported by the fact that the faster converging algorithms (cf. Section 5.3.7) Aggregate and EigenTrust suffer from a less grave distinction reduction.

Table 5.27.: ☒= Matthew effect disabled, ☑= Matthew effect enabled. Distinction.

Aggregate		EigenTrust		BRA		W-PageRank	
☒	☑	☒	☑	☒	☑	☒	☑
21.4	18.6	20.4	18.1	18.7	15.1	18.8	15.1

Table 5.28.: Matthew Test — Agent Settings

	Active		Inactive	
Name	average	top	average	top
Color	yellow	green	yellow	green
Group Size	20	3	20	3
Activity	1.0	1.0	1.0	1.0
Productivity	0.4	0.4	0.4	0.4
Positivity	0.5	0.5	0.5	0.5
Competence	0.7	0.8	0.7	0.8
Social Positivity	0	0	1.0	0
Influenceability	0	0	0.5	0



(a) Aggregate Matthew Inactive

(b) Aggregate Matthew Active

(c) EigenTrust Matthew Inactive

(d) EigenTrust Matthew Active

(e) BRA Matthew Inactive

(f) BRA Matthew Active

(g) W-PageRank Matthew Inactive

(h) W-PageRank Matthew Active

Figure 5.16.: Matthew Test Results

5.3.7. Simulation Iteration Count Test

With this final test, the convergence behavior of the four tested algorithm is studied. The underlying scenario is that of the *Competence Test* (cf. Section 5.3.1), which was

selected because all tested algorithms delivered a clear and perfectly correct result in the standard test. When a convergence test is run, the scheduler does not average the reputation scores and thus, does not calculate the quality measures over an averaged reputation. Instead, it records the quality measure after each iteration step of a single test run. The test is repeated several times and the recorded quality measure values are averaged afterwards.

As the reputation values are not averaged, this test requires far more iteration steps than the basic *Competence Test*. Whereas the *Competence Test* showed perfect results after 100 iteration steps (averaged over 2000 rounds), none of the tested algorithms in the *Simulation Iteration Count Test* reached full correctness or inversion quality even after 600 iteration steps.

The test was run with the same student settings as in the *Competence Test*, with 600 iteration steps, and the quality measure values were averaged over 300 test rounds.

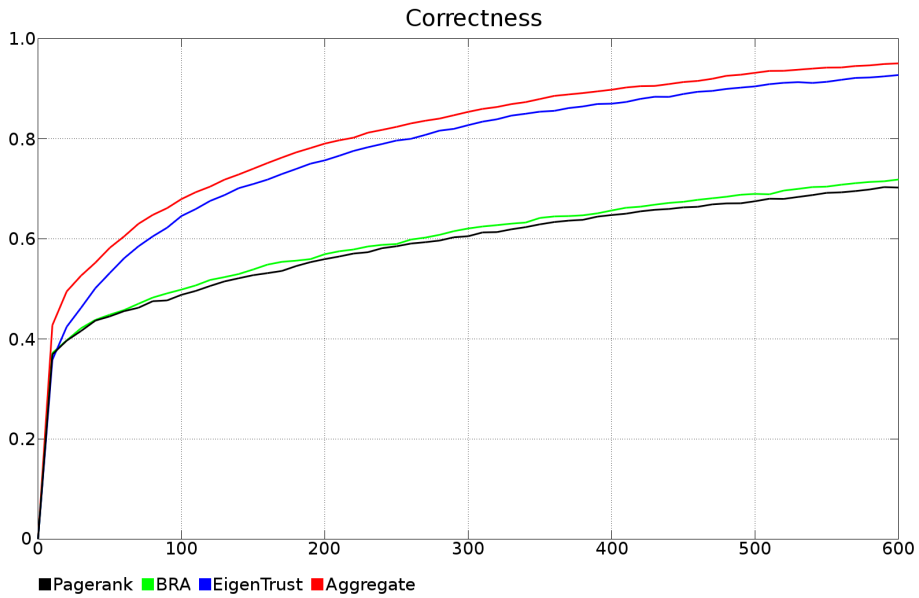


Figure 5.17.: Convergence of Correctness.

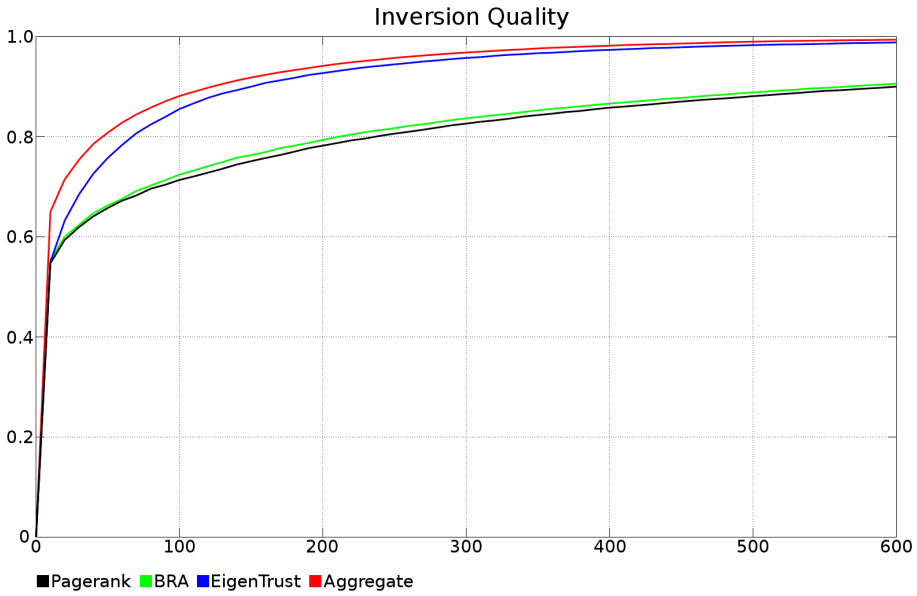


Figure 5.18.: Convergence of Inversion Quality.

For both correctness and inversion quality, Aggregate converges most quickly, followed closely by EigenTrust. BRA and Weighted PageRank converge far slower. This is probably again due to the lack of full consideration of negative votes. Inversion quality converges quicker than correctness at first. This is because inversion quality is biased towards higher values given the group sizes used in the *Competence Test*.

Figure 5.19.: Convergence of Distinction.

In Figure 5.19, both BRA and Weighted PageRank do not seem to converge at all.

This is because distinction only begins to make sense when a minimum correctness or inversion quality value has been reached. Because both algorithms converge less quickly than EigenTrust and Aggregate, they have not reached the threshold at end of iteration step 600. Surprisingly, while EigenTrust converges a bit slower when it comes to correctness and inversion quality, it has a higher overall confidence in its assignment in this scenario compared to Aggregate.

5.4. Performance

Time-wise performance is an ambivalent topic as it greatly depends on optimization and hardware capabilities. Thus, these results should be taken with a grain of salt. However, when it comes to university lectures, there might very well be over a thousand students simultaneously using a technology enhanced learning platform running on one server. If such a software is used to support MOOCs, even more data has to be processed. In such a scenario, it does no harm to use an algorithm that performs equally well, but requires less computational power.
All tested algorithms' computations are predominantly done by the matrix library EJML [EJM16]. As there is not much optimization that can be done outside the library, the time measurements should be fairly comparable. To counteract hardware capabilities, the test time measurements are expressed in percentage relative to Aggregate's performance. Every test was run on a dedicated cpu core with no other thread interfering.

Table 5.29.

	Aggregate	EigenTrust	BRA	W-PageRank
Competence Test	100%	109%	122%	114%
Activity Test	100%	123%	152%	153%
Productivity Test I	100%	106%	115%	110%
Productivity Test II	100%	106%	116%	110%
Coordinated Friends Test	100%	110%	121%	115%
Collusion Test I	100%	113%	129%	120%
Collusion Test II	100%	119%	139%	127%
Matthew effect disabled	100%	110%	114%	112%
Matthew effect enabled	100%	113%	116%	115%
Rounded Total Average	100%	112%	124%	119%

CHAPTER 6

Conclusion

In this thesis, a simulation-based testbed for reputation algorithms is introduced. In the current stage of implementation, it is already able to simulate many different situations, including social and malicious exceptional situations. If required, every part of the framework can easily be modified or extended. New algorithms can be added that are automatically detected, so test setup is very simple.
Currently, an artifact can only have a binary quality. In future versions, it may be reasonable to extend the simulator so that more levels of quality are supported. The simulator currently assumes that a certain level of competence is required to determine whether an artifact is of good or bad quality. That might not be a realistic reflection of the real world. Consider the following example: A question is posted in which the result to a formula is requested. In one case, a simple number is given as the result. It might be correct, but only someone who can see the calculation that was done can verify that it really is correct. This is what the simulator currently assumes. However, in another case, the resulting number could be given plus some explanations on how the result is reached along with secondary literature that elaborates the topic even more. In such a case, even an uninformed student can realize that the given answer is of a high quality.
This concept of level of quality could additionally be extended to social groups. Currently, the affiliation and repulsion lists do not distinguish a level of sympathy or dislike. More complex social constellations could be constructed if the concept of different strengths of bonds between the student agents was elaborated.
Also, agents' personality traits are currently fixed. These parameters are set before the simulation starts and can not change during a simulation run. It would be interesting to elaborate on a way that makes agents more dynamic and implements a kind of feedback mechanism, e.g. that allows agents to consume high quality posts of others, thus increasing their own competence. As a step towards this, a more simple option would be to pass a Lambda Expression to the agent that manipulates traits based on the iteration count of the simulation.
There is also room for improvement regarding the simulator's performance. As the

list of posted artifacts (i.e. the simulation iteration count) and student count grows, applying filter steps to the list of artifacts (filter out artifacts that have already been voted for or are from a disliked group, or prioritize friends' artifacts) have a significant effect on the overall simulation speed. This is mainly because the simulator currently uses linear searches. Applying more appropriate data structures, such as binary trees, might significantly increase performance and thus enable more iterations or a higher student count. Also, which is also interesting because it might also apply in real life situations, a kind of lifespan could be added to artifacts. In digital backchannels like Backstage, very old lecture slides are very likely less regarded than brand new ones. Content posted on old slides thus likely receive less votes. After some iteration counts, 'old' artifacts could be moved to another data structure, and only a few of them sampled when it occurs that old content should receive a vote. This would reduce the search space and thus increase performance.
In the evaluation chapter, four algorithms were selected which competed in eight tests in order to give a first insight of the testbed's capabilities. These tests suggested interesting implications. First, a reputation algorithm's ranking can only reflect the scenario truthfully when there is at least a portion of competent and benevolent agents. If this condition is not met, results can be extremely noisy or misleading. In case of a very uninformed community, supervision and external feedback from competent persons might be required.
Also, selecting an algorithm that makes full use of the information a technology-enhanced learning platform provides seems to improve convergence speed and accuracy. If the platform allows negative and positive votes, selecting an algorithm that considers both voting types likely delivers a more intuitive ranking. For future research, it might prove useful to investigate different aggregation methods that convert negative and positive votes into a positive number range, as required by many eigenvector-based reputation systems. An idea would be to split positive and negative votes, then calculate a 'good' and a 'bad' reputation (the latter based on absolute values of negative votes) separately. The resulting two types of ranking scores could then be aggregated to yield a final 'overall reputation score'.
Moreover, it should be investigated how to warrant resistance against friends voting each other up, especially in classrooms with few students. In this scenario, Aggregate proved to be much more resilient than its eigenvector-based counterparts. It could also prove interesting to combine multiple reputation algorithms into a hybrid algorithm that aggregates the scores of its base algorithms. The aggregated opinion of multiple reputation algorithms might be more correct than that of a single one.
Finally, although some interesting results were gained, its' important to note that the tests were primarily used to showcase the simulator's functionality. They are not sufficient to determine the quality of an algorithm. All algorithms were used with default parameters in very specific, and also few, scenarios. Choosing other parameters could yield very different results. This also applies to the selection and setup of tests. A thorough evaluation remains an interesting research question. The simulator presented in this thesis is a significant contribution to facilitating these experiments.
Currently and within the selected tests, the Backstage Reputation Algorithm be-

haves very similar to Weighted PageRank. However, BRA is still a prototype with many planned extensions yet to be implemented and ideal parameters yet to be determined. For example, in Backstage, lecturers can launch different types of quizzes which are answered by students. In future revisions, BRA is planned to use quiz results to calculate a reputation score for students. Due to the notion of transitive trust, students with a good quiz performance (and thus probably higher competence) can exert more influence via votes. Given a technology enhanced platform context, such tailored extensions could improve ranking accuracy compared to other algorithms. As this framework implements BRA especially modularly, it provides the ideal environment for further development and testing.

APPENDIX A

Source Files

The source files for the simulator and the LaTeX files for this thesis are available in a Git repository which can be found here: `git@gitlab.pms.ifi.lmu.de:Abschlussarbeiten/master_thesis_marco_hoffmann.git`.
The folder structure of this thesis' repository is as follows:

```
/
├── Misc (vector graphics files and conceptual notes)
├── Software (testbed source files)
└── Report (LaTeX files)
    ├── graphics (figures used in the thesis)
    └── bib (bibliography)
```

Bibliography

[AD01] Karl Aberer and Zoran Despotovic. Managing trust in a peer-2-peer information system. In *Proceedings of the tenth international conference on Information and knowledge management*, pages 310–317. ACM, 2001.

[ARH00] Alfarez Abdul-Rahman and Stephen Hailes. Supporting trust in virtual communities. In *System Sciences, 2000. Proceedings of the 33rd Annual Hawaii International Conference on*. IEEE, 2000.

[BGBP11] François Bry, Vera Gehlen-Baum, and Alexander Pohl. Promoting awareness and participation in large class lectures: The digital backchannel backstage. *Proc. IADIS*, pages 27–34, 2011.

[BMJ04] Wilhelm Barth, Petra Mutzel, and Michael Jünger. Simple and efficient bilayer cross counting. *J. Graph Algorithms Appl.*, 8(2):179–194, 2004.

[DK16] Maurice De Kunder. The size of the world wide web. *WorldWideWebSize*, 2016. `http://www.worldwidewebsize.com/`, *Last visited 24 July 2016*.

[Dou02] John R Douceur. The sybil attack. In *Peer-to-peer Systems*, pages 251–260. Springer, 2002.

[EJM16] EJML. Efficient java matrix library, 2016. `http://ejml.org/`, *Last visited 24 July 2016*.

[FKM+05] Karen K Fullam, Tomas B Klos, Guillaume Muller, Jordi Sabater, Andreas Schlosser, Zvi Topol, K Suzanne Barber, Jeffrey S Rosenschein, Laurent Vercouter, and Marco Voss. A specification of the agent reputation and trust (art) testbed: experimentation and competition for trust in agent societies. In *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 512–518. ACM, 2005.

[FPCS04] Michal Feldman, Christos Papadimitriou, John Chuang, and Ion Stoica. Free-riding and whitewashing in peer-to-peer systems. In *Proceedings of the ACM SIGCOMM workshop on Practice and theory of incentives in networked systems*, pages 228–236. ACM, 2004.

[Gef00] David Gefen. E-commerce: the role of familiarity and trust. *Omega*, 28(6):725–737, 2000.

[Goo16] Google. The eigentrust algorithm for reputation management in p2p networks - number of citations, 2016. `https://scholar.google.de/scholar?q=The+eigentrust+algorithm+for+reputation+management+in+p2p+networks`, *Last visited 24 July 2016*.

[HS13] Christopher J Hazard and Munindar P Singh. Macau: A basis for evaluating reputation systems. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 191–197. AAAI Press, 2013.

[HZNR09] Kevin Hoffman, David Zage, and Cristina Nita-Rotaru. A survey of attack and defense techniques for reputation systems. *ACM Computing Surveys (CSUR)*, 42(1):1, 2009.

[JHSMT13] David Jelenc, Ramón Hermoso, Jordi Sabater-Mir, and Denis Trček. Decision making matters: A better way to evaluate trust models. *Knowledge-Based Systems*, 52:147–164, 2013.

[JJLU13] Ginger Zhe Jin, Benjamin Jones, Susan Feng Lu, and Brian Uzzi. The reverse matthew effect: catastrophe and consequence in scientific teams. Technical report, National Bureau of Economic Research, 2013.

[KC10] Reid Kerr and Robin Cohen. Treet: The trust and reputation experimentation and evaluation testbed. *Electronic Commerce Research*, 10(3-4):271–290, 2010.

[KSGM03] Sepandar D Kamvar, Mario T Schlosser, and Hector Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *Proceedings of the 12th international conference on World Wide Web*, pages 640–651. ACM, 2003.

[Lag12] Brent Lagesse. Analytical evaluation of p2p reputation systems. *International Journal of Communication Networks and Distributed Systems*, 9(1-2):82–96, 2012.

[Mon16] MongoDB. Mongodb, 2016. `https://www.mongodb.com`, *Last visited 24 July 2016*.

[PBMW99] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: bringing order to the web. Technical report, Stanford InfoLab, 1999.

[PNJd15] Thao P Nguyen and Brian J d'Auriol. Estarmom: Extendable simulator for trust and reputation management in online marketplaces. Technical report, Academy of Science and Engineering, 2015.

[Poh15] Alexander Pohl. *Fostering awareness and collaboration in large-class lectures: principles and evaluation of the Backchannel Backstage*. Dissertation, München, Ludwig-Maximilians-Universität, 2015.

[R16] R. R project, 2016. `https://www.r-project.org/`, *Last visited 24 July 2016*.

[Rig13] Daniel Rigney. *The Matthew effect: How advantage begets further advantage*. Columbia University Press, 2013.

[SAZ10] Gayatri Swamynathan, Kevin C Almeroth, and Ben Y Zhao. The design of a reliable reputation system. *Electronic Commerce Research*, 10(3-4):239–270, 2010.

[SPS16] SPSS. Spss predictive analytics, 2016. `https://www-01.ibm.com/software/de/analytics/spss/`, *Last visited 24 July 2016*.

[SS01] J Sabater and C Sierra. A reputation model for cregarious societies. In *4th Workshop on Deception, Fraud, and Trust in Agent Societies*, pages 194–195, 2001.

[TPJL06] WT Luke Teacy, Jigar Patel, Nicholas R Jennings, and Michael Luck. Travos: Trust and reputation in the context of inaccurate information sources. *Autonomous Agents and Multi-Agent Systems*, 12(2):183–198, 2006.

[Vas08] Julita Vassileva. Toward social learning environments. *IEEE transactions on learning technologies*, 1(4):199–214, 2008.

[VF90] Jeffrey Scott Vitter and Philippe Flajolet. Average-case analysis of algorithms and data structures, 1990.

[WKLS10] Andrew G West, Sampath Kannan, Insup Lee, and Oleg Sokolsky. An evaluation framework for reputation management systems. *Trust Modeling and Management in Digital Environments: From Social Concept to System Development, Z. Yan, Ed. IGI Global*, pages 282–308, 2010.

[XG04] Wenpu Xing and Ali Ghorbani. Weighted pagerank algorithm. In *Communication Networks and Services Research, 2004. Proceedings. Second Annual Conference on*, pages 305–314. IEEE, 2004.