ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Lecture with Computer Exercises:
Modelling and Simulating Social Systems with MATLAB

Project Report

A Generalized Model for Peer Review: Design and Implementation

Xiang Gao

Zurich
Dec 2011

Agreement for free-download

We hereby agree to make our source code for this project freely available for download from the web pages of the SOMS chair. Furthermore, we assure that all source code is written by ourselves and is not violating any copyright restrictions.

Xiang Gao

Contents

1 Introduction and Motivations

In this paper, we will investigate the modeling of peer review process. There are not many existing papers on the modeling of peer review. In 2010, Thurner [3] implemented a simple model and argued that a small fraction of rational referee will decrease the quality of publication obviously. Francisco [2] implemented a model called PR-1 which is a subset of their proposed model PR-M and their experiments showed that three reviewers are enough to perform a good selection.

The author of this paper intended to extend the existing model to a new one, but due to the time limitation and the author's ability, we only implemented a small prototype. The contribution of this paper is two fold, one is to define the six building blocks for modeling peer review, these core concepts could be the foundation for further research. Another contribution is to develop a small code framework based on the six building blocks, which could be reused, extended by other researchers so as to speed up the development of programs. We do not give any detail model or implementation about peer review process, so we call this model a generalized model. This decision is to give other people who want to extend existing model or idea more flexibility.

The rest of paper is organized as follows, we first define the six building blocks for peer review, next we describe in detail about the implementation, then we show an example model based on the building blocks and code framework, at last we conclude the paper.

2 Six Building Blocks for Modeling Peer Review

In the academic world, scientists, papers, and journals are three important entities. Scientists produce papers, submit papers to journals for review, after receiving the manuscript, the journal will arrange other scientists to review papers and decide if the papers are accepted, rejected, or revision. These three entities constitutes three building blocks in the peer review process.

The other three building blocks are not concrete entities, but three algorithms or methods. One is called producer, which defines the quality of the paper produced by a scientist, another one is called submitter, which decides which journal the scientist wants to submit his or her paper. The last one is named reviewer, this is the method journals use to review and accept papers.

These concepts are not new, but are extracted from existing models. Most of the existing research try to use agent based model to simulate the peer review process, when come to the implementation, they must decide the three algorithms we defined as building blocks. Different models employ different algorithms for the three algo-

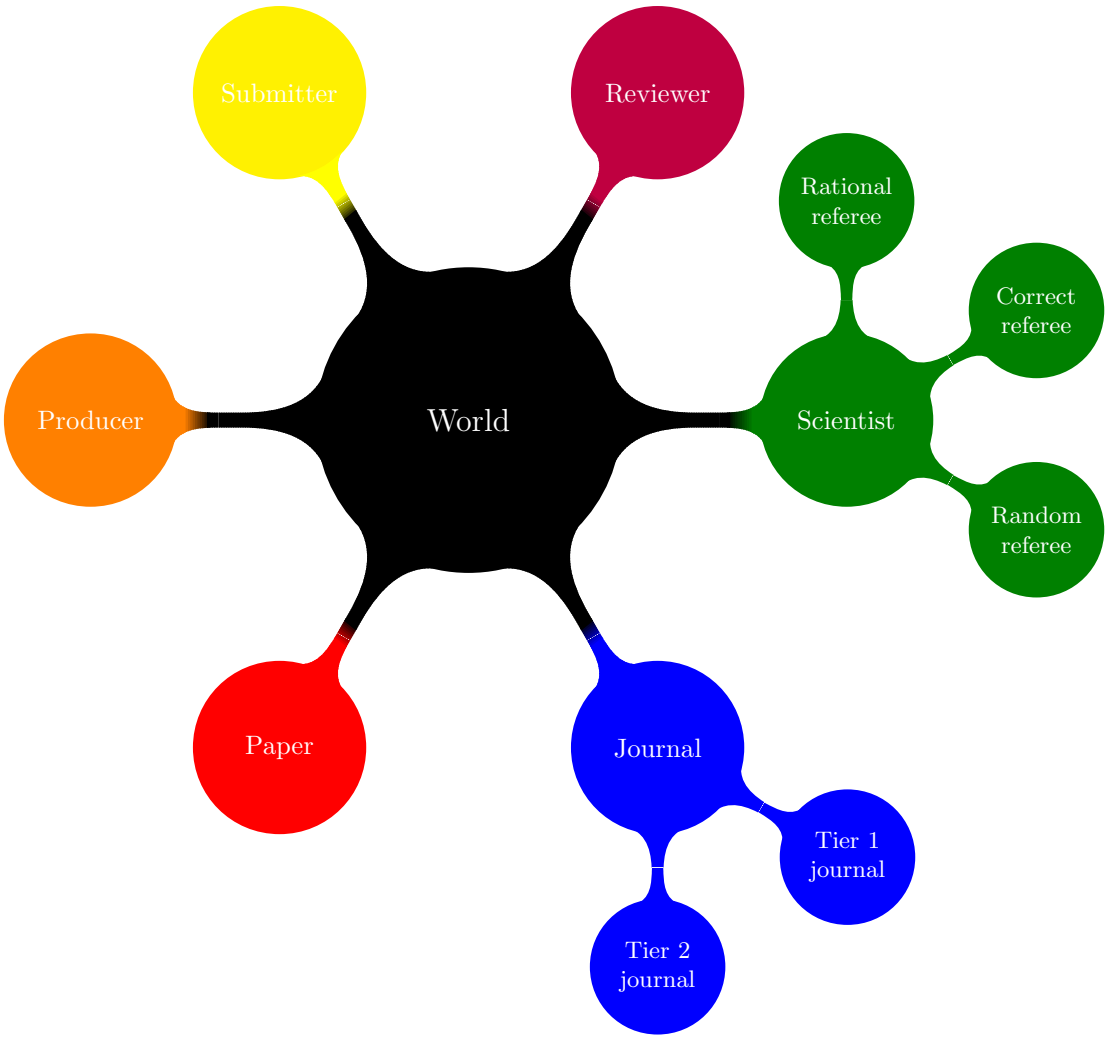


Figure 1: Six building blocks for modeling peer review.

rithm building blocks and use different attributes of the three entity building blocks. Each one of the building blocks can be designed separately, and you can run a lot of experiments with differently combinations. This will result in a lot of work to find the best method to implement the peer review (many researcher aim to find a good way to improve the peer review), which is far beyond the capability of the author. Therefore we focus to implement a small prototype to illustrate the framework we propose and show an example to the user how to extend the model. Hopefully, this work will help others save some efforts when design the model and implement it.

3 Design and Implementation

3.1 Class Design

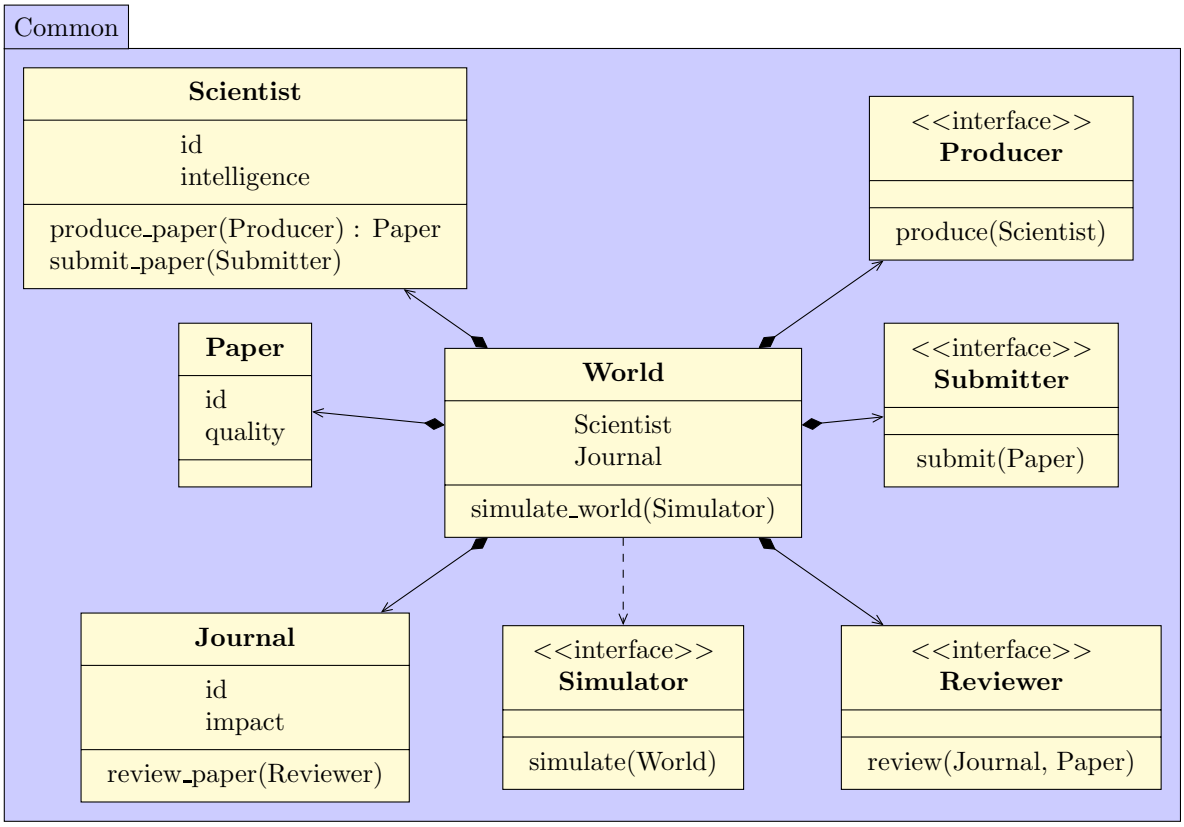


Figure 2: Class diagram of the whole "world".

This part details the implementation of our generalized model. The six building blocks are defined as classes in MATLAB. Scientists have attributes such as id and intelligence, they can produce paper, and submit paper. Paper has name, quality attributes, and other attributes if you think useful, i.e. a unique id to identify the paper for the journal, the citation number, references, co-authors. Journal has attributes like impact factors, editors, committees and so on.

Producer defines an interface called produce, submitter has an interface called submit, reviewer has an interface called review. The three algorithm are defined as abstract classes in MATLAB. Any specific model must implement their own algorithm for these three building blocks.

World is a mashup of all the components in the academic world, which is defined

as a separate class. Simulator as an interface, provides the the simulation algorithm to simulate the peer review process. All these class diagrams are shown in Figure 2.

3.2 Work Flow

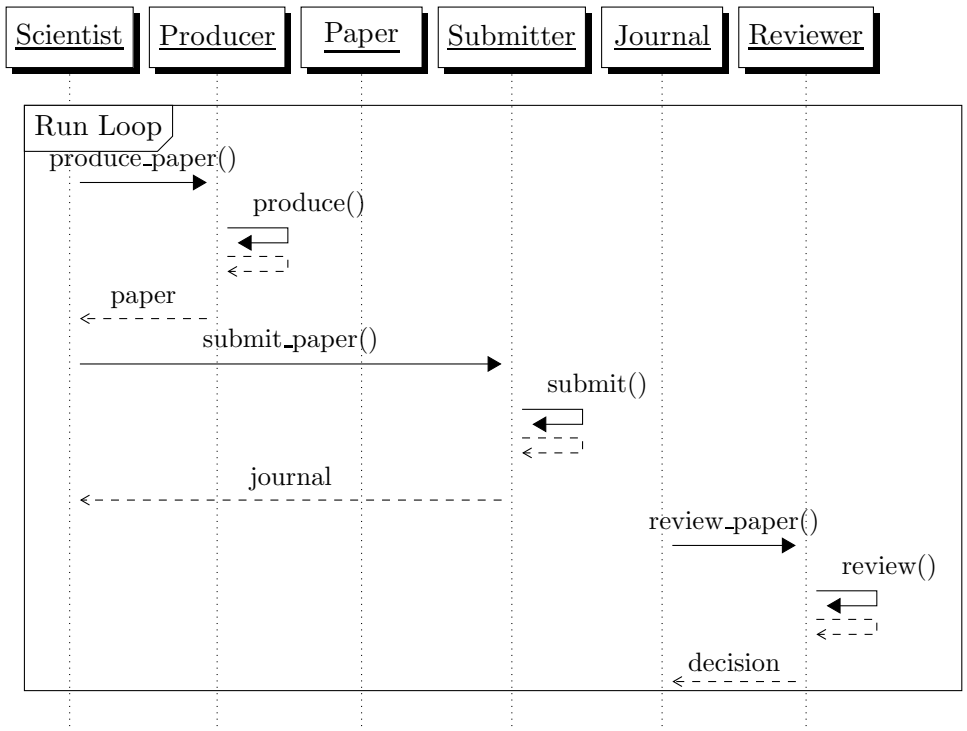


Figure 3: Work flow of simulation.

This section we briefly describe the flow of our simulation process. Although we do not intend to give any specific algorithm for the simulate process, however the framework is mainly designed for agent based model, we put it here for the sake of completeness. In every iteration, scientists first produce papers, and then submit papers to the journals. Journal will arrange reviewers to review the paper and make the final decision. This process loop until we reach the time limit. The detail process should be self-evident from the sequence diagram in Figure 3.

3.3 Code Style

Now we say something about the coding style, although this should not appear in a formal academic paper. All the code are written in MATLAB, with object-oriented

programming style. Every class is defined in a single m file. Starting in the class file, is the comment describing the class. The first line of the comment is the class name, followed by a short description. Then comes the class properties and methods description. Users could find an example in the code appended in the end of this paper. This style will help MATLAB generate help documentation for you, when you type 'help' in the command line. Below every function name is the comment of this function, we avoid mix code and comments since it will make the code hard to follow. The core class files for the building blocks are organized in the folder called common, most of them are defined as abstract class, hinting the users to implement their models based on these building blocks. The data structures (Scientist, Paper, Journal) and algorithms (Producer, Submitter, Reviewer) are defined separately in the spirit of visitor pattern, in order to make the code more reusable and extendable.

Any specific model, for example, the Thurner's model we implemented, is organized in a separate folder. Their code will inherit the building blocks class, implement the missing abstract methods and extend the class attributes if needed.

Every model should have a script file to start the simulation. Often cases are you need to run experiments with different combinations of the parameters and every simulation is independent from each other. For efficiency, we employ the parallel computing ability of MATLAB to speed up the simulation.

4 Example

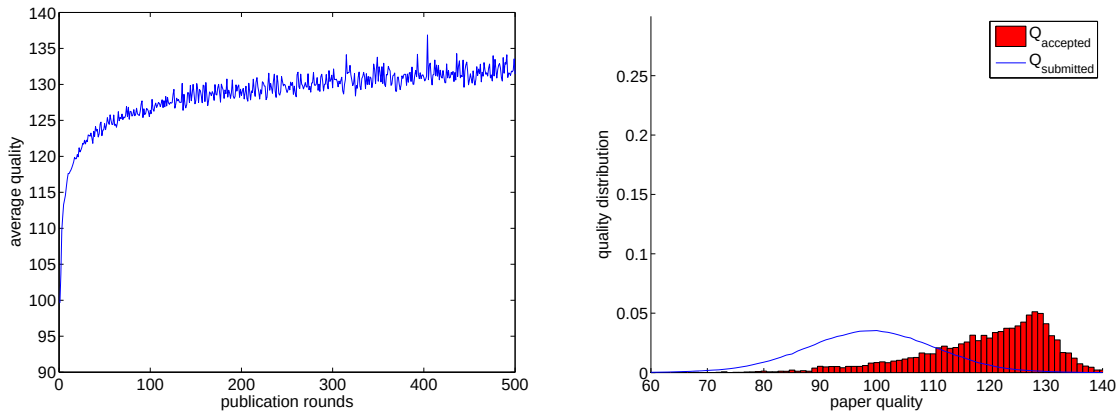
4.1 Thurner's Model: Introduction

Now we show an example based on the framework, specifically, we implement Thurner's model [3]. The detail description of their model can be found in Thurner's paper. Again for the sake of completeness, we briefly summarize it here. In Thurner's model, every scientist receives an IQ, drawn from a normal distribution, $Q_i^{author} \in N(100, \sigma^2)$, and the quality produced by the scientist is $Q_i^{submit} \in N(Q_i^{author}, \sigma^2_{quality})$. We defined a class called *GaussianProducer* to implenment the algorithm. At each time step, every scientist will produce one paper, send to the only journal, this is realized in the class *NaiveSubmitter*. The journal will select two reviewers from all the scientists except for the author. If two reviewers both accept the paper, this paper will be accepted, if both reject, the paper will be rejected, otherwise the paper is accepted randomly. For reviewers, the scientists are classified into three kinds, correct referees, rational referees and random referees. The random referees accept the paper randomly. The rational referees reject all paper with quality greater than his IQ and accept paper with quality below his IQ and above a minimum quality. The case for correct referees is a little more complicated. They accept paper above a minimum quality Q_{min}. It depends on the average quality of ac-

cepted papers. In order to calculate this value, they use a simple moving average, $M(t) = \lambda M(t-1) + (1-\lambda)\langle Q_i^{quality}(t-1)\rangle_i$, where $\langle\rangle_i$ means the average quality. And Q_{min} is calculated by $Q_{min} = M(t) + \alpha std[Q_i^{quality}(t-1)]$. A correct referee will accept paper with quality above this minimum standard. Thurner also introduced network relationship among the scientists. The scientists which are in the network will accept paper written by other scientists in the same network. There is only one network in their model, and size of can be configured as the percentage of the scientists among all the scientists.

4.2 Thurner's Model: Results and Discussion

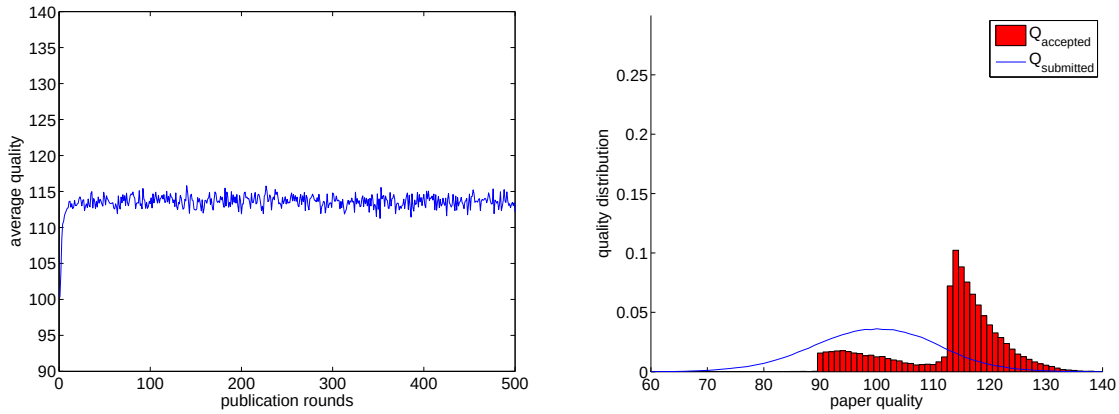
In the following we will reproduce all the results appeared in Thurner's paper. Most results are similar to that of Thurner's paper which more or less confirms the correctness of our implementation. We must point out that these results are not collected with the developed framework shown in this paper, but from the legacy code we have written, since we have not enough time to rerun all the experiments. But the algorithm is the same and we do not intend to reproduce the results exactly as in Thurner's paper.



(a) (b)

Figure 4: (a) The average paper quality when all the reviewers are correct ones. (b) The distribution of quality for submitted paper and accepted paper.

In Figure 4, we show the average quality of accepted paper during the simulation time steps, the scientists pool consists of 100 percent correct referees. It is obvious that the average quality grows as the time line moves.



(a) (b)

Figure 5: (a) The average paper quality when 90 percent the reviewers are correct ones and 10 percent are rational. (b) The distribution of quality for submitted paper and accepted paper.

In Figure 5, we change the scientists community by adding 10 percent rational referees, then average quality of accepted paper decreased compared with Figure 4.

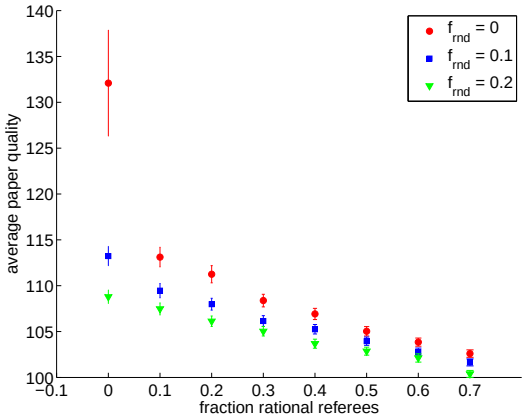


Figure 6: Comparison of average paper quality when varying the fraction of random reviewers from 0 to 0.2, the fractional of rational reviewers from 0 to 0.7.

In Figure 6, we vary the fraction of rational referees and add some random referees, detail parameters can be seen in the figure. The trend is that more rational referees will decrease the quality of accepted paper. Random referees will decrease the quality in the same effect as the rational referees.

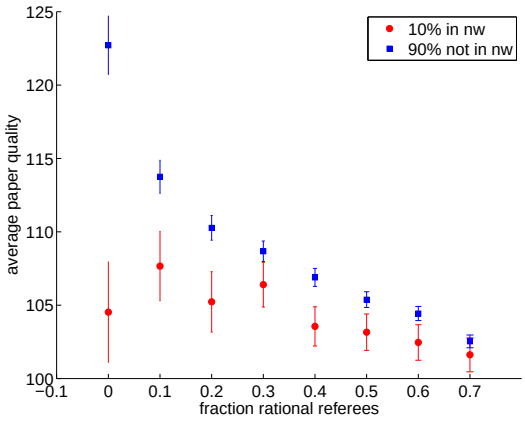


Figure 7: Comparison of average paper quality of when 10 percent scientists are in network and varying the fractional of rational reviewers from 0 to 0.7.

In Figure 7, we show the behavior of whole the peer review process with a small network among the scientists. The results compare the average quality of accepted paper between those in the network and those not. Obviously, scientists in the network have a lower publication quality due to the friendship-bias.

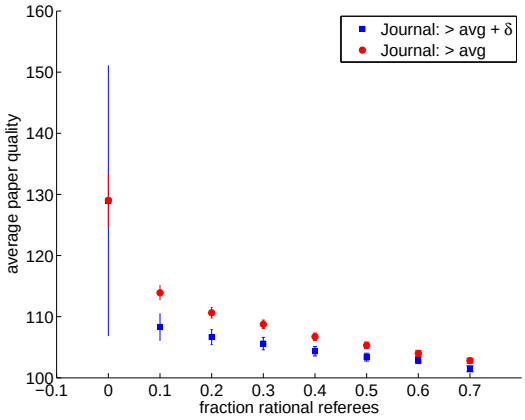


Figure 8: Effect of journal favors higher quality papers.

Before, all the experiments are run with parameter $\alpha = 0$. In Figure 8, we set $\alpha = 1$, which will increase the minimum quality of accepted papers slightly. From the results we see that the average quality decreases, it is explained in Thurner's paper that this phenomenon is due to the number of paper accepted by correct referees

decreases, but those accepted by rational reviewers remain the same.

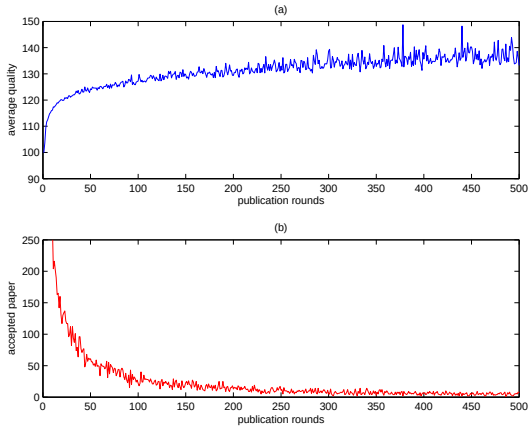


Figure 9: Average accepted quality vs accepted numbers. The reviewers are all correct ones.

Finally, we show some drawbacks of this model. In Figure 4, we see the average quality of accepted paper grows higher and higher. But that is not always the case, the average quality increases means in the next round the acceptance standard will increase, if the standard is very high, the accepted paper is very few. From Figure 9 (b), we can confirm this. And this case is not realistic, since journal always publish more or less the same number of papers in one issue. If we the model for a long time, there will even occur that no paper will be accepted, and the average quality will drop to zero. And this is fatal. Anyway, we do not have time to fix this bug in an interesting way, we put it here just to show some possible ways to extend the model.

5 Lessons and Experiences

Start early

Actually, we did not expect to spend too much time on this project. But this project turned out to be a lot of work. We started in December, just two weeks before the deadline, and at that time we did not even install MATLAB in the laptop. After one week development, we noticed that we did not have time to extend the existing model to a new one, so we can only implement an existing model and reproduce their work to show our framework works.

Plan to throw one away

The initial version of our code developed in the first week was very ugly and

hard to read and maintain. When we wanted to add new features to the existing code, it became a torture, that is the reason we decided to rewrite all the code. The lesson is that you must think carefully what you are going to do now and what you might do in the future. And make sure you have kept it simple. Otherwise, you will throw your code away, anyhow.

Don't repeat yourself

When the author developed this code, there were not any good examples to refer. So we decided to build the "world" from scratch. We hope this can help further students interested in this topic save some time if they want to extend existing model.

Summary and outlook

In conclusion, the goal of our framework is to be extendable, reusable, and parallelized. Currently we have Thurner's model and Allesina's model [1] (only implemented the six building blocks, call for coder to implement the world and simulator) in our github repository. We are happy to see more models added to this repository.

6 References

[1] S. Allesina. Accelerating the pace of discovery by changing the peer review algorithm. *Arxiv preprint arXiv:0911.0344*, 2009.

[2] F. Grimaldo, M. Paolucci, and R. Conte. A proposal for agent simulation of peer review.

[3] S. Thurner and R. Hanel. Peer-review in a world with rational scientists: Toward selection of the average. *Arxiv preprint arXiv:1008.4324*, 2010.

7 Appendix

A Source Code

All the source code used developed during this project can be accessed from `https://github.com/gaox/Modeling-Peer-Review`. I put the code used in this paper below.

```
% World Class
%
% Description
% World is academic environment. Scientists live in the world, they write
% paper and submit to journals for publication.
%
% World Properties:
% scientists     - Scientists produce papers.
% journals       - Journal review papers and publish papers.
% time           - How long the simulation will last.
%
% World Methods:
% simulate_world - Simulate the peer review in the world.

% Author: Xiang Gao
% ETH Zurich, Dept. of Computer Science
% Email: gaox@ethz.ch
% Created: December 2011
% Last revision: 14-Dec-2011

%-------------- BEGIN CODE ----------------

classdef World < handle

    properties
        scientists;
        journals;

        time;
    end

    methods (Abstract)
        simulate_world(obj, simulator);
        % simulate_world: Simulate the peer review world.
        %
        % Input:
        %   simulator       - provide the algorithm for simulation.
    end

```

```
end

%————————————— END OF CODE —————————————
```

```
% Simulator Class
%
% Description
% Simulator provide the method to simulate the peer review process. The
% detailed implementation could be based on agent based model.
%
% Simulator Properties:
% None. Currently it only defines the interface needed to be implement.
%
% Simulator Methods:
% simulate       – Simulate the world, i.e. scientist produces paper, submit
%                  paper and review paper, journal accepts paper.

% Author: Xiang Gao
% ETH Zurich, Dept. of Computer Science
% Email: gaox@ethz.ch
% Created: December 2011
% Last revision: 14–Dec–2011

%————————————— BEGIN CODE —————————————

classdef Simulator < handle

    properties
    end

    methods (Abstract)
        simulate(world);
        % simulate: Provide the algorithm to simulate the peer review.
        %
        % Input:
        %   world          – the academic environment, including
        %                    scientists who produce paper, submit paper,
        %                    and review paper, and journals.
    end

end

%————————————— END OF CODE —————————————
```

```
% Scientist Class
%
% Description
```

```
4  % Scientist produce paper, submit paper and review papers.
5  % This class inherit from handle class in order to pass pointer instead
6  % of value.
7  %
8  % Scientist Properties:
9  % id            - The unique value to identify the scientist.
10 %
11 % Scientist Methods:
12 % produce_paper - The scientist finish writing a paper.
13 % submit_paper  - Submit the paper to a journal.
14
15 % Author: Xiang Gao
16 % ETH Zurich, Dept. of Computer Science
17 % Email: gaox@ethz.ch
18 % Created: December 2011
19 % Last revision: 13-Dec-2011
20
21 %-------------- BEGIN CODE ----------------
22
23 classdef Scientist < handle
24
25     properties
26         id;
27     end
28
29     methods
30         function obj = Scientist(id)
31         % Constructor: Construct the Scientist object, return the obj
32         %              handle.
33         %
34         % Input:
35         %   id              - the unique id of the scientist.
36         % Output:
37         %   obj             - the handle of the created scientist object.
38             if nargin > 0
39                 obj.id = id;
40             end
41         end
42
43         function paper = produce_paper(obj, producer)
44         % produce_paper: The scientist finish writing a paper.
45         %
46         % Input:
47         %   producer        - the producer provides the algorithm to
48         %                     produce a paper for the calling author.
49         % Output:
50         %   paper           - the paper produced for the scientist.
51             paper = producer.produce(obj);
52         end
53
```

```
54      end
55
56      methods (Static)
57          function journal = submit_paper(paper, submitter)
58          % submit_paper: Submit the paper to a journal.
59          %
60          % Input:
61          %   paper           - the paper to be submitted.
62          %   submitter       - the sutmitter provides the algorithm to
63          %                     decide to which journal the paper is ...
                submitted.
64          % Output:
65          %   journal         - the journal submitted to.
66              journal = submitter.submit(paper);
67          end
68      end
69  end
70
71  %--------------- END OF CODE ----------------
```

```
1   % Journal Class
2   %
3   % Description
4   % Journal receives paper, organize referees to review paper and decide
5   % which paper to accept for publication.
6   %
7   % Journal Properties:
8   % id          - The unique value to identify the journal.
9   %
10  % Journal Methods:
11  % review_paper - Arrange reviewers to review paper.
12
13  % Author: Xiang Gao
14  % ETH Zurich, Dept. of Computer Science
15  % Email: gaox@ethz.ch
16  % Created: December 2011
17  % Last revision: 13-Dec-2011
18
19  %--------------- BEGIN CODE ----------------
20
21  classdef Journal < handle
22
23      properties
24          id;
25      end
26
27      methods
28          function obj = Journal(id)
```

```
        % Constructor: Construct the Journal object, return the obj handle.
        %
        % Input:
        %   id              - id identifies the journal uniquely.
        %   impact          - impact of the journal, put here for ...
            further use.
        % Output:
        %   obj             - the handle of the created Journal object.
            if nargin > 0
                obj.id = id;
            end
        end

        function accept = review_paper(obj, paper, reviewer)
        % review_paper: Arrange reviewers to review paper.
        %
        % Input:
        %   paper           - the paper to be reviewed.
        %   reviewer        - reviewer provide the algorithm to review the
        %                     paper.
        % Output:
        %   accept          - 1 if the paper is accepted, 0 if rejected.
            accept = reviewer.review(obj, paper);
        end
    end

end

%---------------- END OF CODE ----------------
```

```
% Paper Class
%
% Description
% Paper represents the production of the scientist's research. The
% scientist will submit the paper object to journal for reviewing the
% publication.
%
% Paper Properties:
% author_id - The identification of the author.
% id        - A unique id to identify the paper.

% Author: Xiang Gao
% ETH Zurich, Dept. of Computer Science
% Email: gaox@ethz.ch
% Created: December 2011
% Last revision: 13-Dec-2011

%---------------- BEGIN CODE ----------------
```

```

classdef Paper < handle

    properties
        author_id;
        id;
    end

    methods
        function obj = Paper(author_id)
        % Constructor: Construct the Paper object, return the obj handle.
        %
        % Input:
        %   author_id       - author_id identifies the author of the paper.
        % Output:
        %   obj             - the handle of the created Paper object.
            if nargin > 0
                obj.author_id = author_id;
            end
        end
    end

end

%---------------- END OF CODE ----------------
```

```
% Producer Class
%
% Description
% Producer provide the method to produce papers. The design of this
% class is based on the visitor pattern.
% This class inherit from handle class in order to pass pointer instead
% of value.
%
% Producer Properties:
% None. This class only provides a method to generate paper for the
% scientist, so it only provides an interface called produce.
%
% Producer Methods:
% produce - Produce the a paper for the scientist.

% Author: Xiang Gao
% ETH Zurich, Dept. of Computer Science
% Email: gaox@ethz.ch
% Created: December 2011
% Last revision: 13-Dec-2011

%---------------- BEGIN CODE ----------------
```

```
23
24 classdef Producer < handle
25     properties
26     end
27
28     methods (Abstract)
29         paper = produce(obj, scientist)
30         % produce: An abstract method. Any algorithm to produce a paper
31         %          must implement the certain algorithm.
32         %
33         % Input:
34         %   scientist       - the handle of a scientist.
35         % Output:
36         %   paper           - the paper produced for the scientist.
37     end
38
39 end
40
41 %---------------- END OF CODE ----------------
```

```
1  % Submitter Class
2  %
3  % Description
4  % Submitter provide the method to submit papers. The design of this
5  % class is based on the visitor pattern.
6  % This class inherit from handle class in order to pass pointer instead
7  % of value.
8  %
9  % Submitter Properties:
10 % None. This class only provides a method to sutmit paper for the
11 % scientist, so it only provides an interface called submit.
12 %
13 % Submitter Methods:
14 % submit - Produce the a paper for the scientist.
15
16 % Author: Xiang Gao
17 % ETH Zurich, Dept. of Computer Science
18 % Email: gaox@ethz.ch
19 % Created: December 2011
20 % Last revision: 13-Dec-2011
21
22 %---------------- BEGIN CODE ----------------
23
24 classdef Submitter < handle
25
26     properties
27     end
28
```

```
29      methods (Abstract)
30          journal = submit(obj, paper)
31          % sutmit: An abstract method. Any algorithm to submit a paper
32          %         must implement the certain algorithm.
33          %
34          % Input:
35          %   paper           - the handle of a paper.
36          % Output:
37          %   journal         - the journal the paper submitted to.
38      end
39
40  end
41
42  %---------------- END OF CODE ----------------
```

```
1   % Reviewer Class
2   %
3   % Description
4   % Reviewer provide the method to choose reviewer for the journal to
5   % review papers and make the decision to accept or reject papers.
6   %
7   % Reviewer Properties:
8   % None. This class only provides a method to choose reviewers and make
9   % accept or reject decisions, so it only provides an interface called
10  % review.
11  %
12  % Reviewer Methods:
13  % review - Choose reviewer for the journal and review papers.
14
15  % Author: Xiang Gao
16  % ETH Zurich, Dept. of Computer Science
17  % Email: gaox@ethz.ch
18  % Created: December 2011
19  % Last revision: 13-Dec-2011
20
21  %---------------- BEGIN CODE ----------------
22
23  classdef Reviewer < handle
24
25      properties
26      end
27
28      methods (Abstract)
29          accept = review(obj, journal, paper)
30          % review: An abstract method. Any algorithm to submit a paper
31          %         must implement the certain algorithm.
32          %
33          % Input:
```

```
        %   journal         - the Reviewer choose referees for the journal
        %   paper           - the handle of the reviewed paper.
        % Output:
        %   accept          - 1 if the paper is accepted, 0 if rejected.
    end

end

%---------------- END OF CODE ----------------
```

```
% Run the simulation of Thurner's model.

% Author: Xiang Gao
% ETH Zurich, Dept. of Computer Science
% Email: gaox@ethz.ch
% Created: December 2011
% Last revision: 14-Dec-2011

f_random = 0;
f_rational = [0, 0.1];

lambda = 0;
alpha = 0;

MAX_TIMESTEP = 500;
SCIENTIST_NUM = 1000;
JOURNAL_NUM = 1;

tic;

distcomp.feature( 'LocalUseMpiexec', false );
matlabpool open 4;

for i = f_random
    parfor j = 1:length(f_rational)
        if (i + f_rational(j) > 1)
            continue;
        end
        world = ThurnerWorld(SCIENTIST_NUM, ...
                             JOURNAL_NUM, ...
                             MAX_TIMESTEP, ...
                             f_rational(j), ...
                             i, ...
                             1 - f_rational(j) - i, ...
                             lambda, ...
                             alpha);
        simulator = ThurnerSimulator();
        world.simulate_world();
```

```
    end
end

matlabpool close;

toc;
```

```
% ThurnerWorld Class
%
% Description
% ThurnerWorld is a mashup of all components in the peer review world
% described in Thurner's paper. The readers are high suggested to read the
% original paper for all the details.
%
% ThurnerWorld Properties:
% MEAN_IQ               - Average intelligence of scientist.
% STDDEV_IQ             - Standard deviation of intelligence of scientist.
% STDDEV_QUALITY        - Standard deviation of quality produced by
%                         scientist.
% REFEREE_PER_PAPER     - Number of reviewer for one paper.
% num_scientists        - Number of scientists in the world.
% num_journals          - Number of journals in the world.
% f_rational            - Fraction of rational scientists.
% f_random              - Fraction of random scientists.
% f_correct             - Fraction of correct scientists.
% lambda                - For computing the moving average.
% alpha                 - The parameter for changing the minimum
%                         requirement of accepted papers.
% producer              - Algorithm to produce paper.
% submitter             - Algorithm to submit paper.
% reviewer              - Algorithm to review paper.
%
% ThurnerWorld Methods:
% simulate_world - Simulate the peer review in the world.
%
% See also:
% http://arxiv.org/abs/1008.4324

% Author: Xiang Gao
% ETH Zurich, Dept. of Computer Science
% Email: gaox@ethz.ch
% Created: December 2011
% Last revision: 14-Dec-2011

%---------------- BEGIN CODE ----------------

classdef ThurnerWorld < World

```

```
42      properties (Constant)
43          MEAN_IQ = 100;
44          STDDEV_IQ = 10;
45
46          STDDEV_QUALITY = 5;
47
48          REFEREE_PER_PAPER = 2;
49      end
50
51      properties
52          num_scientists;
53          num_journals;
54
55          f_rational;
56          f_random;
57          f_correct;
58
59          lambda;
60          alpha;
61
62          producer;
63          submitter;
64          reviewer;
65      end
66
67      methods
68          function obj = ThurnerWorld(num_scientists, num_journals, time, ...
69                                      f_rational, f_random, f_correct, ...
70                                      lambda, alpha)
71          % Constructor: Construct the ThurnerWorld object, return the obj
72          %              handle.
73          %
74          % Output:
75          %   obj            - the handle of the created scientist object.
76              if nargin > 0
77                  obj.num_scientists = num_scientists;
78                  obj.num_journals = num_journals;
79                  obj.time = time;
80                  obj.f_rational = f_rational;
81                  obj.f_random = f_random;
82                  obj.f_correct = f_correct;
83                  obj.lambda = lambda;
84                  obj.alpha = alpha;
85                  obj.scientists = ThurnerScientist.empty(1, 0);
86                  obj.journals = Journal.empty(1, 0);
87                  for i = obj.num_scientists:-1:1
88                      intel = normrnd(ThurnerWorld.MEAN_IQ, ...
                            ThurnerWorld.STDDEV_IQ);
89                      if i <= f_correct * num_scientists
90                          obj.scientists(i) = ThurnerScientist(i, intel, ...
```

```
                                            ThurnerScientist.CORRECT_REFEREE);
                    elseif i ≤ (f_correct + f_random) * num_scientists
                        obj.scientists(i) = ThurnerScientist(i, intel, ...
                                            ThurnerScientist.RANDOM_REFEREE);
                    else
                        obj.scientists(i) = ThurnerScientist(i, intel, ...
                                            ThurnerScientist.RATIONAL_REFEREE);
                    end
                end
                for i = obj.num_journals:-1:1
                    obj.journals(i) = Journal(i);
                end
                obj.producer = ...
                    GaussianProducer(ThurnerWorld.STDDEV_QUALITY);
                obj.submitter = NaiveSubmitter(obj.journals);
                obj.reviewer = RandomReviewer(num_scientists, ...
                                              ThurnerWorld.REFEREE_PER_PAPER, ...
                                                  ...
                                              obj.scientists);
            end
        end

        function simulate_world(obj)
            ThurnerSimulator.simulate(obj);
        end
    end

end

%---------------- END OF CODE ----------------
```

```
% ThurnerSimulator Class
%
% Description
% ThurnerSimulator implement the simulation algorithm in Thurner's paper.
%
% ThurnerSimulator Properties:
% None. This class only provides a method to simulate the peer review model
% in the academic world, so it only provide a method called simulate.
%
% ThurnerSimulator Methods:
% simulate - Simulate the world, i.e. scientist produces paper, submit paper
%            and review paper, journal accepts paper.
%
% See also:
% http://arxiv.org/abs/1008.4324

% Author: Xiang Gao
```

```
% ETH Zurich, Dept. of Computer Science
% Email: gaox@ethz.ch
% Created: December 2011
% Last revision: 14-Dec-2011

%--------------- BEGIN CODE ----------------

classdef ThurnerSimulator < Simulator

    properties
    end

    methods (Static)
        function simulate(world)
        % simulate: Every timestep, each scientist produce a paper, the
        % quality is based on the intelligence of the author. Then they
        % submit the paper to the journal for review. The journal send the
        % paper to two reviewers, recall there are three kinds of reviewers
        % in the world. For details, please see Thurner's paper.
        %
        % Input:
        %   world           - the academic environment, including
        %                     scientists who produce paper, submit paper,
        %                     and review paper, and journals.
            avg_quality = zeros(1, world.time);
            std_quality = 0;
            moving_avg = zeros(1, 2);
            for t = 1:world.time
                accept_quality = zeros(1, world.num_scientists);
                accept_num = 0;
                if (t > 1)
                    moving_avg(2) = moving_avg(1) * world.lambda ...
                                + (1 - world.lambda) * ...
                                    mean(avg_quality(1:t-1));
                else
                    moving_avg(2) = 0;
                end
                world.reviewer.min_quality = moving_avg(2) + world.alpha ...
                    * std_quality;
                for i = 1:world.num_scientists
                    paper = ...
                        world.scientists(i).produce_paper(world.producer);
                    journal = world.scientists(i).submit_paper(paper, ...
                        world.submitter);
                    accept = journal.review_paper(paper, world.reviewer);
                    if accept
                        accept_num = accept_num + 1;
                        accept_quality(accept_num) = paper.quality;
                    end
                end
```

```
                if accept_num > 0
                    accept_quality = accept_quality(1, accept_num);
                    avg_quality(t) = mean(accept_quality);
                    std_quality = std(accept_quality);
                else
                    avg_quality(t) = 0;
                    std_quality = 0;
                end
                fprintf('year %d\n', t);
            end
            save('avg_quality', 'avg_quality');
            plot(1:world.time, avg_quality);
        end
    end

end

%---------------- END OF CODE ----------------
```

```
% ThurnerScientist Class
%
% Description
% ThurnerScientist is the scientists described in Thurner's paper. They are
% either rational reviewers, correct reviewers or random reviewers.
%
% ThurnerScientist Properties:
% CORRECT_REFEREE   - Identify the correct referee.
% RANDOM_REFEREE    - Identify the random referee.
% RATIONAL_REFEREE  - Identify the rational referee.
% type              - One of the value listed above.

% Author: Xiang Gao
% ETH Zurich, Dept. of Computer Science
% Email: gaox@ethz.ch
% Created: December 2011
% Last revision: 14-Dec-2011

%---------------- BEGIN CODE ----------------

classdef ThurnerScientist < Scientist

    properties (Constant)
        CORRECT_REFEREE = 1;
        RANDOM_REFEREE = 2;
        RATIONAL_REFEREE = 3;
    end

    properties
```

```
30          type;
31          intelligence;
32      end
33
34      methods
35          function obj = ThurnerScientist(id, intelligence, type)
36          % Constructor: Construct the ThurnerScientist object, return the obj
37          %              handle.
38          %
39          % Input:
40          %   id              – the unique id of the scientist.
41          %   intelligence    – the initial intelligence of the scientist.
42          %   type            – the type of the referee, correct, random or
43          %                     rational.
44          % Output:
45          %   obj             – the handle of the created ThurnerScientist ...
                object.
46              if nargin > 0
47                  obj.id = id;
48                  obj.intelligence = intelligence;
49                  obj.type = type;
50              end
51          end
52      end
53
54  end
55
56  %————————————— END OF CODE ————————————
```

```
1   % ThurnerPaper Class
2   %
3   % Description
4   % ThurnerPaper represents the production of the scientist's research. The
5   % scientist will submit the paper object to journal for reviewing the
6   % publication. The paper has a quality attribute defined in Thurner's
7   % paper.
8   %
9   % ThurnerPaper Properties:
10  % quality    – The quality of the paper.
11
12  % Author: Xiang Gao
13  % ETH Zurich, Dept. of Computer Science
14  % Email: gaox@ethz.ch
15  % Created: December 2011
16  % Last revision: 17–Dec–2011
17
18  %————————————— BEGIN CODE ————————————
19
```

```
classdef ThurnerPaper < Paper

    properties
        quality;
    end

    methods
        function obj = ThurnerPaper(author_id, quality)
        % Constructor: Construct the ThurnerPaper object, return the obj ...
            handle.
        %
        % Input:
        %   author_id       - author_id identifies the author of the paper.
        %   quality         - the quality of the paper.
        % Output:
        %   obj             - the handle of the created ThurnerPaper object.
            if nargin > 0
                obj.author_id = author_id;
                obj.quality = quality;
            end
        end
    end

end

%---------------- END OF CODE ----------------
```

```
% GaussianProducer Class
%
% Description
% GaussianProducer provide the method to produce papers. The design of
% this class is based on the visitor pattern.
% This class inherit from the abstract class Producer.
%
% GaussianProducer Properties:
% stddev    - The standard deviation of the gaussian distribution.
%
% GaussianProducer Methods:
% produce   - Produce the a paper for the scientist, the quality obeys
%             gaussian distribution.

% Author: Xiang Gao
% ETH Zurich, Dept. of Computer Science
% Email: gaox@ethz.ch
% Created: December 2011
% Last revision: 13-Dec-2011

%---------------- BEGIN CODE ----------------
```

```
22
23 classdef GaussianProducer < Producer
24
25     properties
26         stddev;
27     end
28
29     methods
30         function obj = GaussianProducer(stddev)
31         % Constructor: Construct the Scientist object, return the obj
32         %              handle.
33         %
34         % Input:
35         %   intelligence    – the initial intelligence of the scientist.
36         % Output:
37         %   obj             – the handle of the created scientist object.
38             if nargin > 0
39                 obj.stddev = stddev;
40             end
41         end
42
43         function paper = produce(obj, scientist)
44         % produce: Produce the paper, the quality of the paper is in
45         % gaussian distribution, the mean value is the intelligence of the
46         % scientist, with a small standard deviation. The paper
47         % also receives an author_id equal to the id of the scientist.
48         %
49         % Input:
50         %   scientist       – the handle of a scientist.
51         % Output:
52         %   paper           – the paper produced for the scientist.
53             quality = ceil(normrnd(scientist.intelligence, obj.stddev));
54             paper = ThurnerPaper(scientist.id, quality);
55         end
56     end
57
58 end
59
60 %————————————— END OF CODE ——————————————
```

```
1 % NaiveSubmitter Class
2 %
3 % Description
4 % NaiveSubmitter provide a naive method to submit papers. There is only
5 % one journal, therefore the NaiveSubmitter simply submit the paper to
6 % that journal.
7 %
8 % NaiveSubmitter Properties:
```

```
% journal   – The only journal in the world for publication.
%
% NaiveSubmitter Methods:
% submit    – Submit the paper to the only journal.

% Author: Xiang Gao
% ETH Zurich, Dept. of Computer Science
% Email: gaox@ethz.ch
% Created: December 2011
% Last revision: 18–Dec–2011

%————————————— BEGIN CODE ——————————————

classdef NaiveSubmitter < Submitter

    properties
        journal;
    end

    methods
        function obj = NaiveSubmitter(journal)
        % Constructor: Construct the NaiveSubmitter object, return the obj
        %              handle.
        %
        % Input:
        %   journal         – the only journal can be submitted to.
        % Output:
        %   obj             – the handle of the created submitter object.
            if nargin > 0
                obj.journal = journal;
            end
        end

        function journal = submit(obj, ¬)
        % submit: Sutmit the paper to the only journal.
        %
        % Input:
        %   paper           – the handle of a paper to be submitted.
        % Output:
        %   journal         – the journal to be submitted to.
            journal = obj.journal;
        end
    end

end

%————————————— END OF CODE ——————————————
```

```
% RandomReviewer Class
%
% Description
% RandomReviewer provide the method to choose reviewer for the journal to
% review papers and make the decision to accept or reject papers. The
% algorithm is used in Thurner's model.
%
% RandomReviewer Properties:
% num_reviewer_pool   - The total number of reviewers. All the scientists
%                       in the world can review paper.
% num_reviewer_chosen - The number of reviewers needed to review one
%                       paper.
% reviewers           - Reference to all available reviewers.
% min_threshold       - Minimum quality can be accepted by rational
%                       reviewers.
%
% RandomReviewer Methods:
% review              - Randomly choose (I must be kidding you) reviewer
%                       for the journal and review papers.
% choose_reviewer     - Choose the reviewers randomly.
%
% See also:
% http://arxiv.org/abs/1008.4324

% Author: Xiang Gao
% ETH Zurich, Dept. of Computer Science
% Email: gaox@ethz.ch
% Created: December 2011
% Last revision: 14-Dec-2011

%---------------- BEGIN CODE ----------------

classdef RandomReviewer < Reviewer

    properties
        num_reviewer_pool;
        num_reviewer_chosen;
        reviewers;

        min_quality;
        min_threshold = 90;
    end

    methods
        function obj = RandomReviewer(num_reviewer_pool, ...
                                      num_reviewer_chosen, ...
                                      reviewers)
        % Constructor: Construct the RandomReviewer object, return the obj
        %              handle.
        %
```

```
51          % Input:
52          %   num_reviewer_pool      - the total number of reviewers.
53          %   num_reviewer_chosen    - the number of reviewers needed to
54          %                            review one paper.
55          %   reviewers              - reference to all available reviewers.
56          %
57          % Output:
58          %   obj                    - the handle of the created reviewer ...
                object.
59              if nargin > 0
60                  obj.num_reviewer_pool = num_reviewer_pool;
61                  obj.num_reviewer_chosen = num_reviewer_chosen;
62                  obj.reviewers = reviewers;
63              end
64          end
65
66          function accept = review(obj, ~, paper)
67          % review: Randomly choose reviewers to review the paper. If the
68          %         reviewer's decision ties, accept it randomly.
69          %
70          % Input:
71          %   journal         - the Reviewer choose referees for the journal
72          %   paper           - the handle of the reviewed paper.
73          % Output:
74          %   accept          - 1 if the paper is accepted, 0 if rejected.
75              reviewers_chosen = obj.choose_reviewer(paper);
76              decision = 0;
77              for i = 1:obj.num_reviewer_chosen
78                  index = reviewers_chosen(i);
79                  if obj.reviewers(index).type == ...
                        ThurnerScientist.RATIONAL_REFEREE
80                      if paper.quality <= obj.reviewers(index).intelligence ...
                            && ...
81                              paper.quality >= obj.min_threshold
82                          decision = decision + 1;
83                      end
84                  elseif obj.reviewers(index).type == ...
                        ThurnerScientist.CORRECT_REFEREE
85                      if (paper.quality >= obj.min_quality)
86                          decision = decision + 1;
87                      end
88                  else
89                      decision = decision + randi(2, 1) - 1;
90                  end
91              end
92              if decision > obj.num_reviewer_chosen / 2
93                  accept = 1;
94              elseif decision < obj.num_reviewer_chosen / 2
95                  accept = 0;
96              else
```

```
97                  accept = randi(2, 1) - 1;
98              end
99          end
100
101         function reviewers_chosen = choose_reviewer(obj, paper)
102         % choose_reviewer: Randomly choose the reviewers to review the ...
                paper.
103         %                   The reviewer should not be the author himself.
104         %                   The algorithm is based on Knuth shuffle.
105         %
106         % Input:
107         %   paper             - the handle of the reviewed paper.
108         % Output:
109         %   reviewers_chosen - the chosen reviewers.
110             reviewers_pool = 1:obj.num_reviewer_pool;
111             pool_size = obj.num_reviewer_pool;
112             reviewers_chosen = zeros(1, obj.num_reviewer_chosen);
113             for i = 1:obj.num_reviewer_chosen
114                 index = randi(pool_size, 1);
115                 if reviewers_pool(index) == paper.author_id;
116                     reviewers_pool(index) = reviewers_pool(pool_size);
117                     pool_size = pool_size - 1;
118                     index = randi(pool_size, 1);
119                 end
120                 reviewers_chosen(i) = reviewers_pool(index);
121                 reviewers_pool(index) = reviewers_pool(pool_size);
122                 pool_size = pool_size - 1;
123             end
124         end
125     end
126
127 end
128
129 %--------------- END OF CODE ----------------
```