A PROJECT REPORT
ON

"SOCIAL DYNAMICS OF SCIENCE"

BY

Kemal Akkoyun

UNDER THE GUIDANCE OF
Bulent Ozel

Department of Computer Science
Istanbul Bilgi University
Istanbul, Turkey
2012-2013

Acknowledgements

We are profoundly grateful to **Asst. Prof. Bulent Ozel** for his expert guidance and continuous encouragement throughout to see that this project rights its target since its commencement to its completion.

Kemal Akkoyun

ABSTRACT

This report prepared based on the agent-based model of “Social Dynamics of Science“ paper written by Xiaoling Sun, Jasleen Kaur, Staša Milojević, Alessandro Flammini, Filippo Menczer. The evaluation and result of the simulation overlap with the paper results and these results provide quantitive support for the key role of social communications in shaping the dynamics and diciplines of science.

Simulation models designed as explained in this paper with the facts about the complex socio-cognitive interactions of a changing group of scholars, publications, and scientic communities.

Keywords: science, diciplines, social interactions, society, agent-based simulation, knowledge diffusion.

Contents

List of Figures

Chapter 1

Introduction

1.1 Social Dynamics of Science Models

1.1.1 Origin of Study

The purpose that motivated us to do this project, ”Social Dynamics of Science” article[1]. First we tried to implement the simulation through the article. Models and simulation process implemented as explained in the article.

In the proposed model on this paper; Social Dynamics of Science, we build a social network of collaborations whose nodes are authors, linked by coauthored papers. Each author is represented by a list of disciplines indicating the scientic elds they have been working on, and every discipline has a list of papers. Similarly, each link is a paper describing the collaborations between these two authors.

At this stage, the current implementation does not cover all the simulation that expressed in the paper. In later stages of the project, implementation of simulation will be completed. In futher this project will develop and test with the real scholars data of Turkey.

1.1.2 Improvements

Models of the simulation will be develop and interactions between agents will be stronger than expressed in paper. The simulation iterations will be implement as one year cycle. In one iteration random quantity of author will be selected and walks are going to start from these nodes. With these developments, we aim to have more realistic simulation of knowledge distribution.

Chapter 2

Software Requirements Specification

The simulation implemented with Repast agent-based modelling and simulation toolkit. Repast has a variety of features, such as beign fully object-oriented, including a fully concurrent discrete event scheduler; both sequential and parallel, having social network modeling support tools. Repast models can be developed in many languages, in this simulation we prefered to implement models and interactions with Java. The simulation results will analyse with R.

For coding we used an intergrated development environment Eclipse. For documentation we used JavaDoc. All programs run on Debian GNU/Linux Operation System.

Chapter 3

System Design

3.1 Models

The network implemented as three main model; papers, authors, and disciplines. It represents a social network among authors. In this simulation the social network starts with one author writing one paper in one discipline. During the simulation the network evolves as new authors join, new papers are written, and new disciplines emerge over time.

Parameters will obtained from data sets to calculate related, portions. Simulation uses them as probability distribution seeds.

3.2 Parameters

In every step the simulation starts with choosing an author with uniformly distributed probability. Then starts to walk from this initial author, with creating a new paper. We modelled these behaviors through a biased random walk. The length of the random walk determines the number of co-authors of that related paper. At each step in the walk, the author visits a node (starting with itself) and decides to stop with probability pW or to continue the search for additional authors with probability 1-pW.

At every time step, with probability pN, the simulation also add a new author to the network with a new paper. The parameter pN regulates the ratio of papers to authors.

We introduce a novel mechanism to model the evolution of disciplines by splitting and merging communities in the social collaboration network. The idea, motivated

by the earlier observations from the APS data, is that the birth or decline of a discipline should correspond to an increase in the modularity of the network. Two such events may occur at each time step with probability pD.

Chapter 4

Implementation

4.1 Agents

4.1.1 Author

```
public class Author implements ISDoS{
private int id;
private ArrayList<Discipline> disciplines = new ArrayList<Discipline>();
private ArrayList<Paper> papers = new ArrayList<Paper>();
private Community community;

public Author() {}

public Author(int id) {
  this.id = id;
  this.setCommunity(new Community());
}

public void step(Double pW, Paper paper) {
  this.papers.add(paper); // Add paper to author's written papers.
  paper.getAuthors().add(this); // Add author to paper's co-author list.
  paper.getSharedDisciplines().addAll(disciplines); // Diffusion of knowledge,
      pass collective knowledge through paper.

  // Get environmental containers.
  Context<ISDoS> context = ContextUtils.getContext(this);
  // Get co-authership network.
  Network<ISDoS> net = (Network<ISDoS>)context.getProjection("coAuthorship
     network");

  if(RandomHelper.nextDoubleFromTo(0, 1) < pW){
    System.out.println("Next walk"); // A tracker for debugging.

    double[] pdf = new double[net.getDegree(this)]; // Probability
       distribution function among neighbors of author in network.

    double totalWeight = 0; // Total weight of edges in network.
    for(RepastEdge<ISDoS> edge : net.getEdges()){
      totalWeight += edge.getWeight();
    }
```

```
33        int index = 0;
34        ArrayList<Object> neighbours = new ArrayList<Object>();
35        for(RepastEdge<ISDoS> edge : net.getEdges(this)){
36          double weightOfedge = edge.getWeight();
37          double transitionProbability = weightOfedge / totalWeight; // Transition
                probability as explained above.
38          pdf[index] = transitionProbability;
39          index++;
40          neighbours.add(edge.getTarget());
41        }
42
43        RandomHelper.createEmpiricalWalker(pdf, 0); // A random distribution
            according to given pdf.
44        int indexOfneighbour = RandomHelper.getEmpiricalWalker().nextInt();
45        Author nextAuthor = (Author)neighbours.get(indexOfneighbour); // Get next
            possible co-author candidate.
46        nextAuthor.step(pW, paper); // Continue to walk through next candidate
            author.
47
48      } else {
49        paper.decideDiscipline(context); // Decide main discipline of paper.
50        paper.updateDisciplinesOfAuthors(); // Update disciplines of co-authors.
51        paper.updateCoAuthorNetwork(); // Update network according new knowledge.
52      }
53    }
```

4.1.2 Paper

```
1  public class Paper implements ISDoS{
2   private Integer id;
3   private ArrayList<Author> authors = new ArrayList<Author>();
4   private Discipline discipline;
5   private ArrayList<Discipline> unionOfSharedDisciplines = new ArrayList<
        Discipline>();
6
7   public Paper(Integer id) {
8     this.id = id;
9   }
10
11  public Paper(Integer id, ArrayList<Author> authors, Discipline discipline,
12      ArrayList<Discipline> sharedDisciplines) {
13    this.id = id;
14    this.authors = authors;
15    this.discipline = discipline;
16    this.unionOfSharedDisciplines = sharedDisciplines;
17  }
18
19  public void decideDiscipline(Context<ISDoS> context){
20    IndexedIterable<ISDoS> allDisciplines = context.getObjects(Discipline.class)
          ;
21    int[] disciplineCount = new int[allDisciplines.size()];
```

```
22      for(Discipline d : unionOfSharedDisciplines){
23        disciplineCount[d.getId()] ++;
24      }
25      // For this stage only one discipline is set for a paper.
26      int mostlySharedDisciplineID = 0;
27      for(int i = 0; i < disciplineCount.length; i++){
28        if(disciplineCount[i] > mostlySharedDisciplineID){
29          mostlySharedDisciplineID = i;
30        }
31      }
32       setDiscipline((Discipline)allDisciplines.get(mostlySharedDisciplineID));
33    }
34
35    public void updateDisciplinesOfAuthors(){
36      for(Author author : authors){
37        if(!author.getDisciplines().contains(discipline)){
38          author.getDisciplines().add(discipline);
39        }
40      }
41    }
42
43    public void updateCoAuthorNetwork(){
44      Context<ISourceContext> context = ContextUtils.getContext(this);
45      Network<ISDoS> net = (Network<ISDoS>)context.getProjection("coAuthorship
          network");
46      System.out.println("Authors:" + authors.size());
47      for(int i = 0; i < authors.size(); i++){
48        for(int j = i+1; j < authors.size(); j++){
49          System.out.println("Added "+authors.get(i).getId()+" "+authors.get(j).
              getId());
50          RepastEdge<ISDoS> edge = net.getEdge(authors.get(i), authors.get(j));
51          if(edge == null){
52            net.addEdge(authors.get(i), authors.get(j));
53          } else {
54            edge.setWeight(edge.getWeight()+1);
55          }
56        }
57      }
58    }
59  }
```

4.1.3 Discipline

```
1  public class Discipline implements ISDoS{
2    private Integer id;
3
4    public Discipline(Integer id) {
5      this.id = id;
6    }
7  }
```

4.1.4 Community

```
public class Community implements ISDoS {
  private int id;
  private ArrayList<Author> authors = new ArrayList<Author>();
  private ArrayList<Paper> papers = new ArrayList<Paper>();

  public Community(){}

  public Community(int id, ArrayList<Author> authors, ArrayList<Paper> papers) {
    this.id = id;
    this.authors = authors;
    this.papers = papers;
  }
}
```

4.2 Builder and Simulator

```
public class SocialDynamicsOfScienceBuilder extends DefaultContext<ISDoS>
implements ContextBuilder<ISDoS>  {
  private Double pW;
  private Double pN;
  private Double pD;

  public int getAuthorCount(){
    return getObjects(Author.class).size();
  }

  public int getPaperCount(){
    return getObjects(Paper.class).size();
  }

  public int getDisciplineCount(){
    return getObjects(Discipline.class).size();
  }

  public Context<ISDoS> build(Context<ISDoS> context) {

    // Set id for context.
    context.setId("SocialDynamicsOfScience");

    // Build network for context.
    NetworkBuilder<ISDoS> netBuilder = new NetworkBuilder<ISDoS>(
        "coAuthorship network",
        context, false);
    netBuilder.buildNetwork();

    // Get rates for simulation as parameters.
    Parameters params = RunEnvironment.getInstance().getParameters();
    setpW((Double) params.getValue("authors_per_paper"));
```

```
33      setpD((Double) params.getValue("papers_per_author"));
34      setpN((Double) params.getValue("papers_per_discipline"));
35
36
37      Network<ISDoS> net = (Network<ISDoS>)context.getProjection("coAuthorship
            network");
38
39      // Initialize a network for simulation.
40      // TODO: Get an initial data set from outside from a file.
41      Author A = new Author(0);
42      Discipline D = new Discipline(0);
43      Paper P = new Paper(0);
44
45      // Author our only agent but we need to keep track of other entities.
46      context.add(A);
47      context.add(D);
48      context.add(P);
49
50      net.addEdge(A, A);
51      A.getDisciplines().add(D);
52      A.getPapers().add(P);
53      P.getAuthors().add(A);
54      P.setDiscipline(D);
55
56      // Since in this stage, I do not implemented split/merge of disciplines,
57      // - I need more than one discipline.
58
59      Author A1 = new Author(1);
60      Author A2 = new Author(2);
61      Author A3 = new Author(3);
62      Author A4 = new Author(4);
63
64      Discipline CS = new Discipline(1);
65      Discipline MATH = new Discipline(2);
66      Discipline PHY = new Discipline(3);
67      Discipline PHIL = new Discipline(4);
68
69      Paper P1 = new Paper(1);
70      Paper P2 = new Paper(2);
71      Paper P3 = new Paper(3);
72      Paper P4 = new Paper(4);
73
74      context.add(A1);
75      context.add(A2);
76      context.add(A3);
77      context.add(A4);
78
79      context.add(P1);
80      context.add(P2);
81      context.add(P3);
82      context.add(P4);
83
84      context.add(CS);
85      context.add(MATH);
```

```
86      context.add(PHY);
87      context.add(PHIL);
88
89      net.addEdge(A1, A2);
90      net.addEdge(A2, A3);
91      net.addEdge(A1, A3);
92      A1.getDisciplines().add(CS);
93      A1.getPapers().add(P1);
94      P1.getAuthors().add(A1);
95      A2.getDisciplines().add(CS);
96      A2.getPapers().add(P1);
97      P1.getAuthors().add(A2);
98      A3.getDisciplines().add(CS);
99      A3.getPapers().add(P1);
100     P1.getAuthors().add(A3);
101     P1.setDiscipline(CS);
102
103     net.getEdge(A2, A3).setWeight(2);
104     A2.getDisciplines().add(MATH);
105     A2.getPapers().add(P2);
106     P2.getAuthors().add(A2);
107     A3.getDisciplines().add(MATH);
108     A3.getPapers().add(P2);
109     P2.getAuthors().add(A3);
110     P2.setDiscipline(MATH);
111
112     net.addEdge(A3, A3);
113     A3.getDisciplines().add(PHY);
114     A3.getPapers().add(P3);
115     P3.getAuthors().add(A3);
116     P3.setDiscipline(PHY);
117
118     net.addEdge(A4, A);
119     A4.getDisciplines().add(PHIL);
120     A4.getPapers().add(P4);
121     P4.getAuthors().add(A4);
122     A.getDisciplines().add(PHIL);
123     A.getPapers().add(P4);
124     P4.getAuthors().add(A);
125     P4.setDiscipline(PHIL);
126
127     // An example community.
128     Community community = new Community();
129     context.add(community);
130
131     SocialDynamicsOfScienceSimulator simulator = new
            SocialDynamicsOfScienceSimulator(pW, pN, pD);
132     context.add(simulator);
133
134     return context;
135   }
136 }
137
138 public class SocialDynamicsOfScienceSimulator implements ISDoS {
```

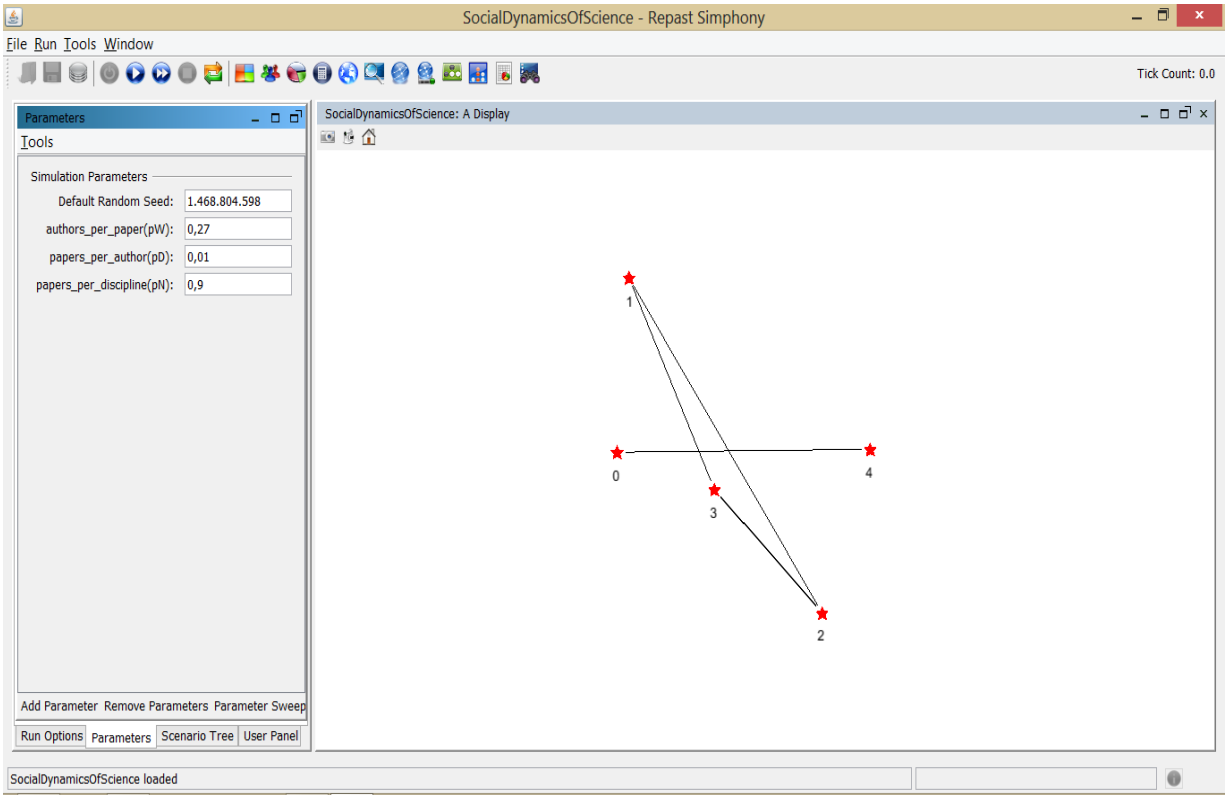
```
139   private Double pW;
140   private Double pN;
141   private Double pD;
142
143   public SocialDynamicsOfScienceSimulator(Double pW, Double pN, Double pD) {
144     this.pW = pW;
145     this.pN = pN;
146     this.pD = pD;
147   }
148
149   public void select() {
150     Context<ISDoS> context = ContextUtils.getContext(this);
151     System.out.println("stepper");
152     // Randomly get an author from context.
153     Iterable<ISDoS> authors = context.getRandomObjects(Author.class , 1); //
          Since this methods returns iterable object,
154     // Selects an author uniformly distributed randomly from context.
155     Author author = new Author();
156     // An implementation walk around, cused by usage of built-in context.
          getRandomObjects(Author.class , 1) method.
157     for(ISDoS a : authors){
158       author = (Author)a;
159     }
160
161     // Calls biased random walk by passing relavent data.
162     Paper paper = new Paper(context.getObjects(Paper.class).size());
163     context.add(paper);
164     author.step(pW, paper);
165
166     // With a probability seed that given as a parameter adds an author network
          or not.
167     if(RandomHelper.nextDoubleFromTo(0, 1) < this.pN){
168       Author newAuthor = new Author(context.getObjects(Author.class).size());
169       context.add(newAuthor);
170       Paper newPaper =  new Paper(context.getObjects(Paper.class).size());
171       context.add(newPaper);
172       newPaper.getAuthors().add(newAuthor);
173       newAuthor.getPapers().add(newPaper);
174
175       Iterable<ISDoS> coAuthors = context.getRandomObjects(Author.class , 1);
176       // Selects an author uniformly distributed randomly from context.
177       Author coAuthor = new Author();
178       for(ISDoS a : coAuthors){
179         coAuthor = (Author)a;
180       }
181       coAuthor.step(pW, newPaper);
182     }
183     if(RandomHelper.nextDoubleFromTo(0, 1) < this.pD){
184       evoluateDisciplines();
185     }
186
187   }
188 }
```

4.3 Documentation

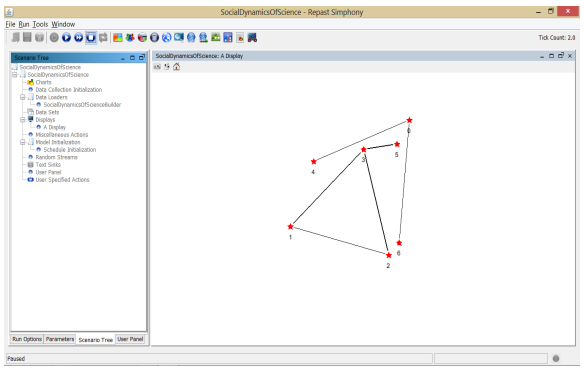
The codes are documented properly, you can see them from the Documentation/index.html file.

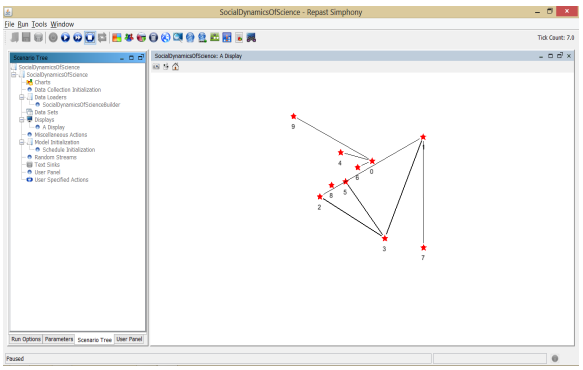
Chapter 5

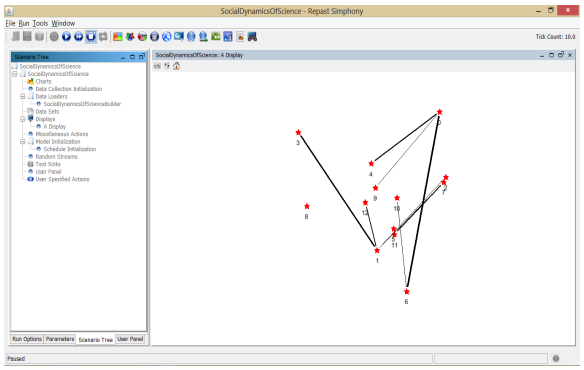
Simulation

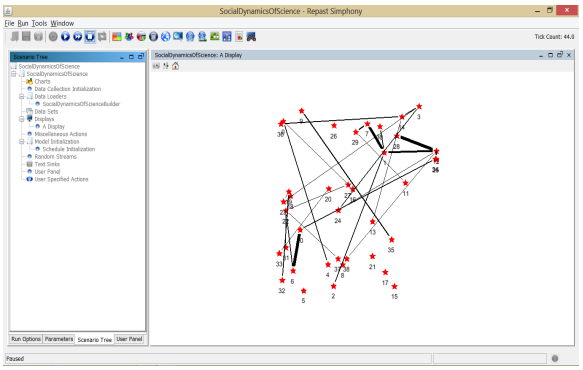


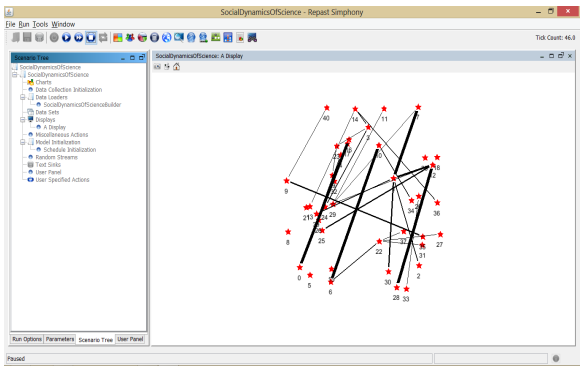
Initial Screenshot and Parameters

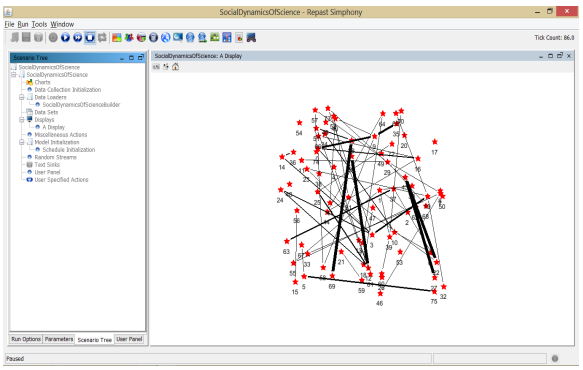

SocialDynamicsOfScience - Repast Simphony
File Run Tools Window
Tick Count: 2.0
Paused


SocialDynamicsOfScience - Repast Simphony
File Run Tools Window
Tick Count: 7.0
Paused


SocialDynamicsOfScience - Repast Simphony
File Run Tools Window
Tick Count: 10.0
Paused


SocialDynamicsOfScience - Repast Simphony
File Run Tools Window
Tick Count: 44.0
Paused


SocialDynamicsOfScience - Repast Simphony
File Run Tools Window
Tick Count: 46.0
Paused


SocialDynamicsOfScience - Repast Simphony
File Run Tools Window
Tick Count: 86.0
Paused

Chapter 6

Future Scope

In future development we can complete the modelize the evalution of diciplines by splitting and merging communities in this network. These events can be implement with these ideas,

- For a split event we select a random discipline with its coauthor network and decide whether a new discipline should emerge from a subset of this community. We partition the coauthor network into two clusters. If the modularity of the partition is higher than that of the single discipline, there are more collaborations within each cluster than across the two. We then split the smaller community as a new discipline. In this case the papers whose authors are all in the new community are relabeled to reect the emergent discipline. Borderline papers with authors in both old and new disciplines are labeled according to the discipline of the majority of authors. Some authors may as a result belong to both old and new discipline.
- For a merge event we randomly select two disciplines with at least one common author. If the modularity obtained by merging the two groups is higher than that of the partitioned groups, the collaborations across the two communities are stronger than those within each one. The two are then merged into a single new discipline. In this case all the papers in the two old disciplines are relabeled to reect the new one.

In this simulation, when it decides to continue to bias random walk, we just calculate the probability of co-authors with authors who have collaborated before are likely to do so again. We can develop this probabilistic selection on the bias random walk also with considering these facts,

- Authors who have collaborated before are likely to do so again
- Authors with common collaborators are likely to collaborate with each other
- It is easier to choose collaborators with similar than dissimilar background
- Authors with many collaborations have higher probability to gain additional ones

References

[1] *Social Dynamics of Science*; Xiaoling Sun, Jasleen Kaur, Staša Milojević, Alessandro Flammini, Filippo Menczer