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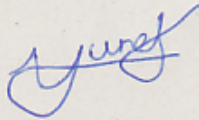Modelling and Simulating Social Systems with MATLAB

Project Report

Stable Marriage Problem
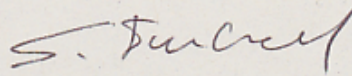
Valentin Junet & Samuel Imfeld

Zurich
December 2014

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.

Valentin Junet   Samuel Imfeld

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

___

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Stable Marriage Problem

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	First name(s):
Junet	Valentin
Imfeld	Samuel

With my signature I confirm that
- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	Signature(s)
Zurich, 10.12.2014	

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.

Contents

1   Abstract

Online dating or match-making websites are flourishing these days. More and more people rely on their algorithms when searching for their Mr. Right, or Mrs. Right, respectively. Algorithms for match-making are therefore of quite some interest.

An important result in this concern is the so-called Gale Shapley algorithm proposed by Gale and Shapley [1962]. In 2012, Shapley even received the Nobel Prize in Economics for his work. The goal of this paper is to discuss the original model described by Gale and Shapley [1962] and advance the model in some sense. Namely we are going to introduce two changes to the model: The incompleteness of information and the possibility of preference changes. The modifications will be discussed further in section 4.2.

It is however not our claim that these changes applied to the model will make it an exact description of reality. Our goal is to study the repercussions on the stability and other significant indicators that show up when applying the modifications.

2   Individual contributions

Code was written by both. In the report, Valentin focused on the chapters 3, 4, 6.1 and 6.2 whereas Samuel concentrated on chapters 1, 5 and 6.3.

3   Introduction and Motivations

The stable marriage problem is quite well known; it is originally an interesting mathematical problem, but it also has something catchy: Everyone has his own idea of what stability is, how we think it is optimal, and how we would like to be matched, or how we think it should be done. Then there comes this emotionless algorithm that pretends – with the support of some definitions and mathematical proofs – to find all of this for us and to do it as least as good as we would. Of course this is a little exaggerated. But when we think about those internet sites that match people according to some criteria, is't it what we're doing? Letting some numbers find an objective answer to those questions, to our subjective questions?

In order to prevent misunderstandings one must very precisely answer the following questions: What is stability? How can a stable situation be optimal? Of course answering those questions would result in a rather philosophical discussion which is not the goal of our paper. Nevertheless stability and other related indicators (which were going to introduce soon enough) will be the center of our attention. But in order to reduce complexity will only be considering some very abstract and theoretical nodes which, for some practical reasons, will be called men or women.

The key points of our work presented here are to generate random links (or "friendship") between men and women (which in the following discussions will be referred to by nodes), to classify those links in a random way in order to get a preference list and to apply the algorithm of Gale and Shapley which matches the nodes according to their preferences. Then we modify some parameters and interpret the results. For a more precise description of the procedure, you may refer to the next section.

3.1 Research Question

We want to analyze the influence that the modifications we are going to apply (see section 4.2) has on the simulation result, i. e. the indicators for stability and optimality (see sections 4.2 and 5.5 for detailed explanation). In particular the questions are:

For stability

- How is the stability influenced when the visibility radius becomes smaller (and therefore also the preference lists)?

- Are there more instabilities if the preference changes are more frequent?

- Does it matter whether the men or the women change preferences?

We expect that there are more instabilities for a higher preference change rate. We also think it will make a difference when women change preferences instead of men because the algorithm is not symmetric.

For optimality

- How is the optimality influenced when the visibility radius becomes smaller?

- Is the output less optimal if the preference changes are more frequent?

- Does the optimality depend on the input size?

Here we expect to see the following results: The result is less optimal for a smaller visibility radius because then it is more likely that nodes have no partner (which is considered as suboptimal, see section 4.2). It is also less optimal for a higher preference change rate.

4 Description of the Model

4.1 Gale and Shapley's algorithm

Let's first describe the algorithm of Gale and Shapley. The setting is of a very abstract nature. There is a certain number of nodes, half of them are men and the other half are

women. The first step is that every node makes a preference list, i. e. a ranking of all the nodes of oppposite sex. Then the algorithm proceeds as follows:

1. single men propose to their favorite women

2.
   - women accept their favorite suitor or their only suitor
   - if a woman has a fiance and a suitor, she chooses the one she likes better
   - if a woman rejects a man, then the next woman on the rejected man's list is his new favorite

3. if the single men still have women on their list of preferences, go to step 1

So the algorithm terminates when all nodes have a partner. This is obviously always achieved if the length of all the preference lists corresponds to the number of nodes of opposite sex (see Gale and Shapley [1962, p. 14, Theorem 1]). Gale and Shapley also proved that this algorithm always terminates with an optimal stability. For this purpose they gave precise definitions of stability and optimality:

Definition 1: *An assignment of applicants to colleges will be called unstable if there are two applicants α and β who are assigned to colleges A and B, respectively, although β prefers A to B and A prefers β to α.*[1]

Definition 2: *A stable assignment is called optimal if every applicant is at least as well off under it as under any other assignment.*[2]

These definitions are in terms of colleges and applicants because in their paper they first start with this setting and then go on to the special case of stable marriages. Translated to the words of match-making the definitions would be something like: "A set of marriages is called unstable if under it there are a man and a woman who are not married to each other but prefer each other to their actual mates"[3] and "A stable assignment is called optimal if every man is at least as well off under it as under any other assignment".

4.2 Modifications

We will now present in more detail the modifications we made to this algorithm. Our purpose was to modify and adapt it so that it represents more accurately the reality. Of course it is fundamentally impossible to make a perfect representation of reality through this: If for example a man goes into a bar, he doesn't particularly make a list of the

---

[1]Gale and Shapley [1962, p. 10]

[2]Gale and Shapley [1962, p. 10]

[3]Gale and Shapley [1962, p. 11]

women he prefers and propose to each woman according to his list (or at least this is not systematically observed). However, there are some modifications we can make to make it more realistic. One of these is that a man doesn't necessarily know every women (and vice versa). To improve this point, we modified the list of preferences so that men and women only know each other in a restricted area. For example, a man who lives in China doesn't know a woman who lives in Switzerland, so he won't propose to her (we didn't consider the case where the man in China could know the Swiss woman though internet or by traveling, he only knows the women "around" him). We also considered the case in which a man knows a woman in his area but she doesn't know him and then this man can propose to her and she might accept or not. The other modification we made is that people, obviously, can change their minds. We modified it so that during the process either only the men, only the women or both can change their mind. These changes of preferences are made each "round" (each time a single man proposes to a woman) with a certain probability which is fixed.

We also introduced some new concepts, namely the number of 'singles" and "'dumps" and the "optimality index". The number of singles tells us how many nodes didn't find a partner. Since we chose to do the simulation with the same number of men and women, there will be exactly the same number of single men and women. The number of "dumps" is the number of times a woman rejected her fiance for another man. This can be seen has an indication of time, the more people "get dumped", the longer it takes to reach a final assignment. The optimality index is defined as the sum over all ranks of the actual partners in the preference list of their respective partners, divided by two times the number of men/women squared (resulting in a normalization, e. g. it only takes values between zero and one). If a node has no partner, it counts as the maximal rank because having no partner is certainly not optimal. The optimality index should somehow give information about the optimality of the outcome, although it does not correspond to definition 2 in section 4.1. A low optimality index corresponds to a rather optimal assignment whereas a high optimality index indicates a not so optimal assignment.

5   Implementation

5.1   Conventions

The nodes (men resp. women) are referenced by integers from one to n where n is the input size (number of men/women).

Accordingly, a preference list is a permutation of the first n integers: The element $\sigma(1)$ is the most preferred node, $\sigma(2)$ the second most preferred and so on. In the case of non-complete information, there are $m < n$ distinct integers followed by zeros to complete the sequence.

4

5.2 Generating the Preference Lists

5.2.1 Random

For testing purposes we first coded a generator for random preference matrices which basically calls `randperm(n)` n times, resulting in a matrix whose rows represent the preference list of a single node. The generated matrix can be used to simulate the classic case with complete information.

5.2.2 The Plane Algorithm

The question that now arises is the following: How can I generate a preference matrix with non-complete information that represents reality in an appropriate manner? In particular there should be a mechanism for controlling the number of non-zero entries in the preference lists (i. e. controlling the degree of completeness of information). The number of non-zero elements should, however, not be the same for each node (this would mean that each node knows the same fixed number of people, which is clearly not a good representation of real situations). Additionally, situations like 'man x knows woman y but woman y does not know man x' should be considered. To achieve these characteristics we developed the 'generatePlane' algorithm: It is based on the idea that a node only knows his neighbours (e. g. the people in his town or community). Therefore his preference list should consist only of nodes that are closer than a certain distance, which we will call the visibility radius in the further discussions. We realized this by assigning to each node a random position in a two dimensional plane. For simplicity they are distributed in $[0, 1] \times [0, 1]$.

Definition 3: *The visibility radius defines the neighbourhood of a node in $[0, 1] \times [0, 1]$. A node can only know other nodes that have a distance smaller than the visibility radius.*

To avoid border effects, the edges are connected (one could view it as a torus). For the actual generation of the preference lists one just has to iterate through all nodes and determine all nodes of opposite sex that are in the disc of visibility radius. We then chose to use a random permutation of these neighbourhood nodes to keep it simple.

A constant visibility radius implies in particular that if node x knows node y, then also node y knows node x. But as we don't want this to be the case all the time we also made an option for a random visibility radius that is updated every step.

5.3 Making Matches

This is the part where the algorithm proposed by Gale and Shapley [1962] actually comes into play. An implementation in MATLAB is shown in listing 1 (The basic ideas for the implementation are taken from RosettaCode [2014] and adapted to MATLAB).

```
1  function [ engaged, stable ] = makeMatch( m, f )
2  %makeMatch finds engagements for preferences according to Gale-Shapley ...
       algorithm
3  ...
4  freemen = [(1:n)',ones(n,1)];
5  engaged = zeros(n,2);
6  while ˜isempty(find(freemen(:,2)==1,1))
7      theman = find(freemen(:,2)==1,1);
8      thegirl = m(theman,1);
9      index = find(engaged(:,2)==thegirl,1);
10     if(isempty(index) )
11         engaged(theman,1) = theman;
12         engaged(theman,2) = thegirl;
13         freemen(theman,2) = 0;
14     else
15         fiance = engaged(index,1);
16         girlprefers = f(thegirl,:);
17         if(find(girlprefers==theman,1)<find(girlprefers==fiance,1))
18             engaged(theman,1) = theman;
19             engaged(theman,2) = thegirl;
20             engaged(fiance,1) = 0;
21             engaged(fiance,2) = 0;
22             freemen(theman,2) = 0;
23             freemen(fiance,2) = 1;
24         else
25             m(theman,:) = [m(theman,2:n) 0];
26         end
27     end
28 end
29 stable = checkEngagements(engaged,m,f);
30 ...
31 end
```

Listing 1 :   makeMatch code skeleton

The main structures are the arrays freemen and engaged. The array freemen contains all free men (first column: indices of men, second column: 1 for free, 0 for not free) and engaged contains the engagements produced by the algorithm (first column: indices of men, second column: indices of women). The main loop starts in line 6 and continues until all men are in an engagement. Then in the loop the first free man is picked (line 7) and his most preferred girl is determined (line 8). Now one has to distinguish between two cases:

- The girl has no engagement: Everything is fine, the man and the girl are now engaged (lines 11 and 12). Also the man is not free anymore (line 13).

- The girl alread has a fiance: In this case one has to do another distinction:

    – The girl prefers the new man to her fiance: A new engagement is made and the old one is cancelled (lines 18-21). One also has to update freemen.

6

– The engagements remain unchanged, but the girl is removed from the man's preference list because she is not attainable for him.

In the end the engagements are checked with the `checkEngagements` algorithm described in section 5.4. However, when applying the modifications described in section 4.2 and still using this algorithm one runs into problems (as expected). Therefore we had to adapt the `makeMatch` algorithm to be able to handle the following situations:

- An unknown man proposes to a woman who is not engaged: We decided that in this case the proposing man should have a chance to succeed with his proposal, but this should not always be the case. Therefore we implemented a random decision with a certain probability for accepting (typically around 0.25). This can be seen as a simulation of the real-life situation 'the woman gets to know the unknown man and gets to like him (or not)'.

- An unknown node proposes to a woman who is engaged: We decided to apply the same procedure as above. One could argue that the probability for accepting should be lower because she already has a fiance, but we left that out for the sake of simplicity.

- The preference list of a man is empty but he is not engaged: In this case the man is just left with no partner.

The implementation of the above points will not be discussed here any further. The final algorithm is in the appendix for reference.

Finally we added the preference change functionality: In each iteration step a random decision is made between changing preferences or not, using the probability given in the additional parameter `changerate`. If the answer is positive, the preference list of one randomly chosen node is perturbed a little: A random node in the preference list is chosen and switched with the node one rank lower (for example `[4,2,3,1]` becomes `[4,3,2,1]` after switching nodes 2 and 3). There is also a parameter to determine whether only men, only women or both should change their preferences.

5.4 Checking the Engagements

An important indicator for the later discussion is the stability of the engagements. It can be checked using this algorithm. The main loop is shown in listing 2 (again the basic structure is inspired by RosettaCode [2014])

```
1  while he<=n
2      she = engaged(he,2);
```

```
3       % make sure that the man has a partner i. e. she is not 0
4       % otherwise continue because nothing to check
5       ...
6       hisindex = find(f(she,:)==he,1);
7       herindex = find(m(he,:)==she,1);
8       helikesbetter = m(he,1:herindex);
9       shelikesbetter = f(she,1:hisindex);
10      % check for her
11      for i=1:size(shelikesbetter)
12          guy = shelikesbetter(i);
13          guysgirl = engaged(guy,2);
14          guylikes = m(guy,:);
15          if (find(guylikes==she,1)<find(guylikes==guysgirl,1))
16              stable = false;
17          end
18      end
19      % check for him
20      ...
21      he=he+1;
22  end
```

Listing 2 : checkEngagements code fragment

The main loop is an iteration over all men (line 1), but only those who are engaged because having no partner is not considered as an instability (lines 3-5). After having retrieved all the indices and preference lists, one iterates over all nodes that appear before the actual partner in their respective preference lists and checks whether there is an instability (see definition 1 in section 4.1). This results in two for loops, one for the man and one for the woman he is engaged to.

5.5   Simulation

The actual simulation makes iterated calls to `generatePlane` and `makeMatch` to simulate the modified Gale-Shapley Algorithm for different parameter settings. The input parameters are:

- `n`: the input size $n \in \mathbb{N}$

- `mode, radius`: mode determines the choice of the visibility radius (either random or constant), and if the radius is constant then it can be set with the parameter radius $r \in [0, 0.5]$

- `changerate`: the rate at which preference changes are performed $c \in [0, 1]$

- `p`: determines whose preferences are changed $p \in \{0, 0.5, 1\}$

The resulting output variables are:

8

- no. of instabilities: number of instabilities according to definition 1 in section 4.1

- no. of dumps: number of fiances that have been dumped during match-making, see section 4.2

- no. of singles: number of single nodes that remain after match-making

- optimality index: as defined above, see section 4.2

6   Simulation Results and Discussion

Let's first make a general remark for the interpretation of the results. We labelled our nodes – like in the original algorithm – with man and woman, but a more accurate notation would be "the ones who choose" for the men and "the ones who are chosen" for the women. For the sake of interpretation, this indeed makes more sense since: In real life , when someone asks a person if he or she wants to go out with him/her, the one asking chooses someone that he/she prefers. The person who is asked makes his/her decision based on his/her preferences (at least according to this model). So we could do exactly the same by switching or even mixing both categories and find through our interpretation an even more realistic information about the difference in the results between choosing and being chosen.

It is also important to notice that every kind of interpretation with respect to reality is purely hypothetical. This is not a realistic model; it only shows us how this specific way of choosing and being chosen responds to the input parameters.

6.1   Instabilities

6.1.1   Parameters

For the considerations about the instability and the number of singles we did the following simulation: Generate a matrix of 2^n persons with n in `1:11` and either a random radius or a constant radius r in $\{0.1, 0.25, 0.4\}$. To make the matches, we chose a probability p – for the change of preferences – in `0:0.25:1`. We did this with the change of preferences only for men, only for women and for both. We iterated this simulation 11 times and then took the algebraic mean value as data.

6.1.2   Effects of preference changes

We will start by some considerations about the number of instable (as defined by Shapley and Gale) couples counted at the end of the process. We first note, that if the probability of changing preferences is 0, then there is like in the original algorithm no instability for

9

any kind of matrices. This would mean that the algorithm still holds with a restricted knowledge of the others. If we think of the plane as small or big cities, then the persons who follow this algorithm without changing their mind will be able to find a stable love regardless the size of their respective city.

We can also quickly mention that the bigger the probability that a change occurs is, the bigger is the number of instabilities (for both, men and women changing their mind), so when people are indecisive and change their mind, the marriages are less stable.

6.1.3 Effects of visibility radius

We found out an interesting relation between the visibility radius and the number of instabilities. In general (meaning for every kind and probability in the change of preferences and every size of matrices) we found out that for the "big" radius 0.25 and 0.4, there is significantly less instability than for the 0.1 radius. Actually, for the two big radii, the mean number of instability is always less than two. For example for 2^{11} persons, the mean number of instability (w.r.t. all the probabilities) is 12.236 for r=0.1, 0.491 for r=0.25 and 0.218 for r=0.4. This would mean that the people with restricted possibilities who change their mind are more likely to end up in an unstable union than the people who have more choice.

6.1.4 Dependence on who changes preferences

Let's now consider the influence of whether only the men or only the women change their mind on the instability of the marriages. We noticed that for constants radius there is always more instability when the men change their preferences but, as mentioned in last paragraph, the number of instability for r=0.25 and 0.4 is so low that this difference, although present, is negligible. Except randomness, we cannot find an explanation for this in the way we made the changes in the algorithm since the change of preferences are made at the beginning of each main loop regardless if the change is made by a man or a woman. So this means that, up to randomness, the explanation should lay in the way of choosing and making matches. With this algorithm, once a man is matched, theres nothing we can do to change it, so if he changes his mind and the woman he now likes better likes him better too, theres nothing to do about it, it will be unstable, whereas if a woman changes her mind then the man she likes better might propose to her and even if she is engaged she can be with him. This means then that a person should make sure to be able to leave his partner if he changes his mind...not very surprising. However, what is very curious is that if we consider this relation not with a constant radius but rather with a random radius, then – from a certain amount of people (2^7) on – the opposite is true: there is more instability when the women change their preferences! With our previous considerations it is clear that, up to randomness, the explanation should be find in the construction of the
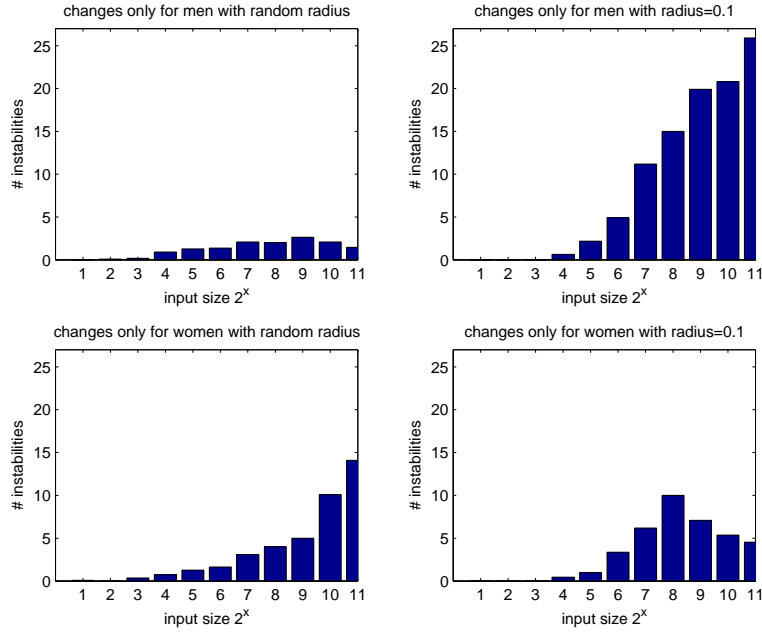
Figure 1: Number of instabilities in function of the number of persons, for a probability in the change of preferences fixed at 100% and for different radius.

plane. The fact that the radius are random influences the men and the women similarly, so the true difference (and the reason we introduced it) between constant and random radius is that, with the random one, a man can know someone who doesn't know him and propose to her and respectively a woman can be proposed to by someone she doesn't know (in this case, we arbitrarily decided that she accepts 25% of the time). As we can see in the graphs, in this case the undecided women don't produce particularly more instabilities, but the undecided however produce less instability than with a constant radius. This might either be caused by the fact that a random radius can be bigger than 0.1 and as we've seen in this case there is less instabilities, or because if they like better a woman who doesnt know him, then this wont be unstable. We can then suppose that with any radius the most influent factor of instability is the indecision of the men, the one who choses and can't break a union.

We also noticed that in the case in which both, the men and the women change their minds we observe that the results are approximately the mean between the ones in which only the men and the ones in which only the women change their mind.
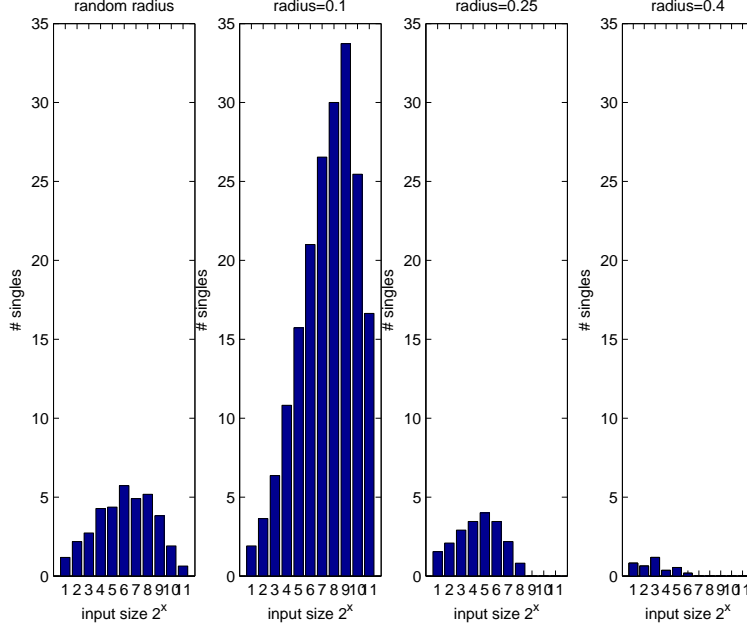
Figure 2: Number of singles in function of the number of person, for different radii.

6.2 Singles

We now want to consider how this variances of the algorithm influence the number of singles person. However let's first start by mentioning what does not have any effects on this number: The change of preferences, more precisely its rate and whether only men, women or both change them. Indeed, for any fixed radius and fixed number of persons in our simulation, we observe the same results by varying the rate of change and the group of people changing their minds.

The influent factors are then the number of people and the radius. Looking at figure 2 we can easily notice, that with the number of people increasing, the number of singles has a normal distribution. We performed a Kolmogorow-Smirnow-Test for all data on the number of singles in function of the number of persons. The test was always positive. We can observe that for the small radius 0.1, there is a lot more of singles men and women than for the others, actually for radius 0.4, it is even unusual to find a single pair. For radius 0.4, resp. radius 0.25 from 2^7 to 2^9 persons there isn't any single person left! So the more people know each other, the less singles there are. The random radius gives something intermediate, there is though more singles than for the radius 0.4 and 0.25 since the radius
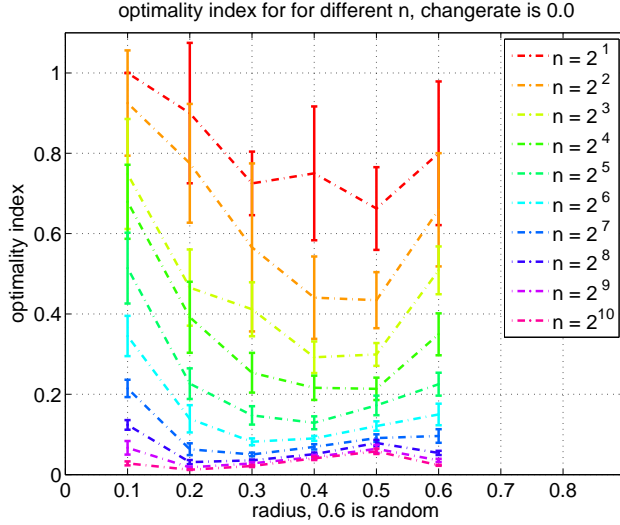
Figure 3: Errorbar plot for the optimality index in dependence of the visibility radius. The different lines correspond to different input sizes n. The preference change rate was set to 0.

0.1 has way more influence on the number of singles., resp. persons there isnt any single So the more people know each other, the less single there is. The random radius gives something intermediate, although there are more single persons than for the radius 0.4 and 0.25 since the radius 0.1 has way more influence on the number of singles.

6.3   Optimality Index

6.3.1   Parameters

The input parameters were chosen as follows: The input size is a power of 2. Due to the quadratic dependence of the generatePlane algorithm, the highest power evaluated is $2^{10} = 1024$. Originally we wanted to work with basis 10 but we had to change to basis 2 because of performance reasons as well. For the visibility radius we took the discrete values 0.1:0.1:0.5, and for the preference change rates values in 0:0.5:1. For each triple of parameters the simulation was performed 10 times.

6.3.2 Effects of preference changes

Surprisingly the expected effect of the preference changes could not be observed at all. The optimality index is about the same for the extreme values 0 and 1 for the change rate, and also for 0.5. For small input sizes there is a small variation of the optimality index (that seems to be caused more by random effects than an actual correltation). For higher n the plot lines are at constant height (see also section 10.1 in the Appendix for plots).

6.3.3 Effects of visibility radius

The results of the visibility radius variation do also not look very promising. In figure 3 one can see that for low input sizes (n = 2 to 32) there seems to be a dependence on the visibility radius, namely the optimality index decreases for higher visibility radii (as we expected). But for higher n the effect disappears, and the optimality index even gets higher again for the visibility radius close to 0.5. Note that the points labeled with 0.6 on the x axis correspond to random radius and not visibility radius 0.6.

One way to explain this is that there might be several 'hidden' effects that cause this weird looking behavior. One of them could be that the visibility radius is chosen to be constant for all input sizes. It is now not the same when thousand instead of only ten nodes are placed in the plane, because for more nodes the number of nodes that happen to be in the neighborhood defined by a constant visibility radius is much higher.

6.3.4 Input size dependency

Looking at figure 4, one can clearly observe that the optimality index gets smaller for increasing input size, meaning that the result is more optimal for higher n. This is a bit counterintuitive at first thought. Why should the result be less optimal for smaller n? One could think that this effect is somehow produced by the modifications that were made to the Gale-Shapley algorithm. But it turns out that the effect can also be observed for randomly generated preference lists using `generateRandom`, see figure 5. For this type of preference lists none of the 'special treatments' in the modified algorithm (e. g. empty preference lists) apply, but the effect can still be observed. Onecan also observe that the higher the visibility radius gets, the closer is the curve to the one with randomly generated preferences.

So one might think that there is some power law relating the input size to the optimality index. Conducting an analysis (using `loglog` plot and `polyfit` for linear regression) yields

$$opt = c \cdot n^{-0.36}$$

But how can this relation be explained? We suspect that it has to do with the following: The optimality index measures the rank of the assigned partners. If now the number of
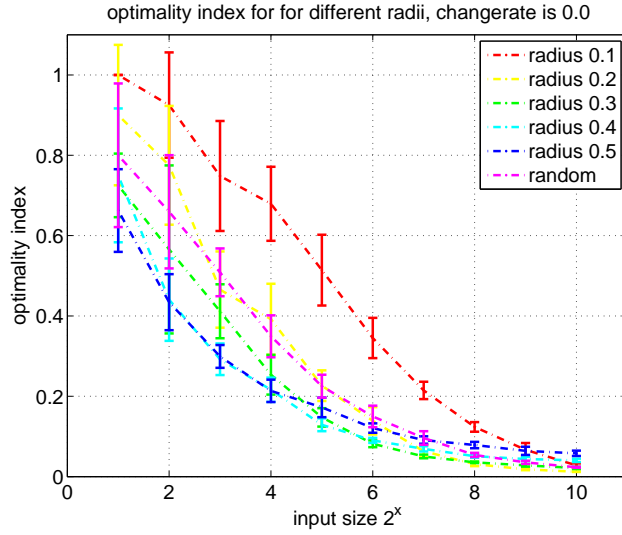
Figure 4: Optimality index errorbar plot in dependence of input size n. Preferences were generated by the `generatePlane` algorithm. Different lines represent different visibility radii.

nodes is high, the chance that for example two women choose the same man as their number one preference is higher than it would be for lower input sizes. But as only one of those women gets assigned to this man, the other women gets another man who is now at most second best on her list.
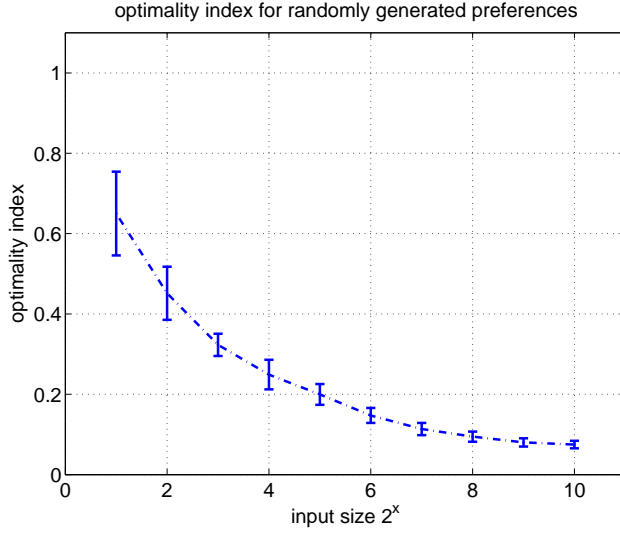
Figure 5: Optimality index errorbar plot in dependence of input size n. Preferences were generated randomly.

7 Summary and Outlook

The important points related to the stability of the modified model are the following:

- Without change of preferences stability still holds.

- For the "big" radii 0.25 and 0.4 there is practically no instability.

- The men who change their mind induce more instabilities than the women.

The important points related to the number of singles of the modified model are the following:

- The change of preferences has no influences on the number of single persons.

- The small radius 0.1 increases the number of single persons.

- When the input size is big enough, there are no single persons.

And for the optimality index:

- Neither change of preferences nor non-complete information through the visibility radius has an effect on optimality

16

- The optimality is related to the input size by a power law

8 References

D. Gale and L. S. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):pp. 9–15, 1962. ISSN 00029890. URL `http://www.jstor.org/stable/2312726`.

Youtube Numberphile. Stable marriage problem, 2014a. URL `https://www.youtube.com/watch?v=Qcv1IqHWAzg`.

Youtube Numberphile. Stable marriage problem, 2014b. URL `https://www.youtube.com/watch?v=LtTV6rIxhdo`.

RosettaCode. Stable marriage problem, 2014. URL `http://rosettacode.org/wiki/Stable_marriage_problem#Python`.

9 Appendix A: MATLAB Codes

generateRandom.m

```matlab
function [ m, f ] = generateRandom( n )
%GENERATERANDOM generates random preference matrices
m = zeros(n,n);
f = zeros(n,n);
for i=1:n
    m(i,:) = randperm(n,n);
    f(i,:) = randperm(n,n);

end
```

generatePlane.m

```matlab
function [ mpref,fpref ] = generatePlane( n ,mode, radius)
%GENERATEPLANE generates preference lists for men and women
%    based on a plane where women and men are represented by points
```

```matlab
4   %    they have a limited visibility radius
5   %    n: number of men and women
6   %    mode: visibility radius mode, optional argument
7   %       1 ——> const, one constant radius for all nodes
8   %       2 ——> random, a new random radius is generated in each iteration
9   %               value is between 0.1 and 0.5
10  %    default mode is const
11  %    mpref: mens preferences in nxn matrix
12  %    fpref: womens preferences in nxn matrix
13
14  global verbosity
15
16  if (nargin >= 2 && mode == 1)
17      assert(nargin==3);
18      r = radius;
19  end
20  if(nargin < 2)
21      mode = 1;
22      r = 0.2;%default value
23  end
24
25  % generate random coordinates
26  % and extend to torus
27  men = zeros(3,9*n);
28  rnd = rand(2,n);
29  men(:,(0*n)+1:1*n)=[(1:n);rnd];
30  men(:,(1*n)+1:2*n)=men(:,(0*n)+1:1*n)+[zeros(1,n);ones(1,n);zeros(1,n)];
31  men(:,(2*n)+1:3*n)=men(:,(0*n)+1:1*n)+[zeros(1,n);ones(1,n);ones(1,n)];
32  men(:,(3*n)+1:4*n)=men(:,(0*n)+1:1*n)+[zeros(1,n);zeros(1,n);ones(1,n)];
33  men(:,(4*n)+1:5*n)=men(:,(0*n)+1:1*n)+[zeros(1,n);-ones(1,n);ones(1,n)];
34  men(:,(5*n)+1:6*n)=men(:,(0*n)+1:1*n)+[zeros(1,n);-ones(1,n);zeros(1,n)];
35  men(:,(6*n)+1:7*n)=men(:,(0*n)+1:1*n)+[zeros(1,n);-ones(1,n);-ones(1,n)];
36  men(:,(7*n)+1:8*n)=men(:,(0*n)+1:1*n)+[zeros(1,n);zeros(1,n);-ones(1,n)];
37  men(:,(8*n)+1:9*n)=men(:,(0*n)+1:1*n)+[zeros(1,n);ones(1,n);-ones(1,n)];
38
39  women = zeros(3,9*n);
40  rnd = rand(2,n);
41  women(:,(0*n)+1:1*n)=[(1:n);rnd];
42  women(:,(1*n)+1:2*n)=women(:,(0*n)+1:1*n)+[zeros(1,n);ones(1,n);zeros(1,n)];
43  women(:,(2*n)+1:3*n)=women(:,(0*n)+1:1*n)+[zeros(1,n);ones(1,n);ones(1,n)];
44  women(:,(3*n)+1:4*n)=women(:,(0*n)+1:1*n)+[zeros(1,n);zeros(1,n);ones(1,n)];
45  women(:,(4*n)+1:5*n)=women(:,(0*n)+1:1*n)+[zeros(1,n);-ones(1,n);ones(1,n)];
46  women(:,(5*n)+1:6*n)=women(:,(0*n)+1:1*n)+[zeros(1,n);-ones(1,n);zeros(1,n)];
47  women(:,(6*n)+1:7*n)=women(:,(0*n)+1:1*n)+[zeros(1,n);-ones(1,n);-ones(1,n)];
48  women(:,(7*n)+1:8*n)=women(:,(0*n)+1:1*n)+[zeros(1,n);zeros(1,n);-ones(1,n)];
49  women(:,(8*n)+1:9*n)=women(:,(0*n)+1:1*n)+[zeros(1,n);ones(1,n);-ones(1,n)];
50
51  %plotting
52  % if verbosity~=0
53  %     plot(men(2,1:n),men(3,1:n),'o',women(2,1:n),women(3,1:n),'o');
```

```matlab
54 %       label1 = cellstr( num2str(women(1,1:n)') );
55 %       label2 = cellstr( num2str(men(1,1:n)') );
56 %       text(women(2,1:n),women(3,1:n),label1);
57 %       text(men(2,1:n),men(3,1:n),label2);
58 %       title('nodes in plane');
59 %       legend('men','women');
60 % end
61
62 d = zeros(2,9*n);
63 mpref = zeros(n,n);
64 fpref = zeros(n,n);
65
66
67 for i=1:n
68     man = men(:,i);
69     for j=1:9*n
70         woman = women(:,j);
71         d(:,j) = [woman(1,1);norm(man(2:3)-woman(2:3),2)];
72     end
73     if mode==2
74         r = rand*0.4+0.1;
75     end
76     index = find(d(2,:)<r);
77     available = women(:,index);
78     sz = size(available,2);
79     if sz>n
80         available = available(:,1:n);
81         sz = n;
82     end
83     perm = randperm(sz);
84     mpref(i,1:sz) = available(1,perm);
85 end
86 for i=1:n
87     woman = women(:,i);
88     for j=1:9*n
89         man = men(:,j);
90         d(:,j) = [man(1,1);norm(man(2:3)-woman(2:3),2)];
91     end
92     if mode==2
93         r = rand*0.4+0.1;
94     end
95     index = find(d(2,:)<r);
96     available = men(:,index);
97     sz = size(available,2);
98     if sz>n
99         available = available(:,1:n);
100        sz = n;
101    end
102    perm = randperm(sz);
103    fpref(i,1:sz) = available(1,perm);
```

19

```
104  end
105  end
```

vprintf.m

```
1  function vprintf(varargin)
2  % VPRINTF controlled printing
3  %
4  global verbosity
5  if verbosity~=0
6      fprintf(varargin{:});
7  end
```

makeMatch.m

```
1   function [ engaged, output ] = makeMatch( m, f, changerate, p )
2   %makeMatch finds engagements for preferences according to Gale—Shapley ...
        algorithm
3   %
4   %    men an women encoded as integers from 1 to n
5   %
6   %    input:
7   %    m: preference matrix of the men. Each row corresponds to a man and
8   %    the elements are the women listed according to his preferences.
9   %    f: preference matrix of the women. Each row corresponds to a woman and
10  %    the elements are the men listed according to her preferences.
11  %    changerate: rate at which preference changes are performed, e. g. if
12  %    changerate=0.2 then only in 20% of iterations preferences are changed
13  %    p: change preferences for men (p=1) / women (p=0.5) / both (p=0.5)
14  %
15  %    dimensions must be correct, m=nxn, f=nxn.
16  %
17  %    returns:
18  %    engaged: nx2 Matrix containing matches
19  %    output: output data —>
20  %    output(1,1): number of instabilities
21  %    output(1,2): number of singles
22  %    output(1,3): number of dumps
23  %    output(1,4): optimality index
24
25  % optional test prints
26  global verbosity
27  vprintf('mens preferences:\n');
28  if verbosity~=0 disp(m); end
29  vprintf('womens preferences:\n')
30  if verbosity~=0 disp(f); end
31  % assign local variables
```

```matlab
32  initialm = m;
33  initialf = f;
34  n = size(m,1);
35  n2 = size(f,1);
36  % make sure dimensions agree
37  assert(n == size(m,2));
38  assert(n==n2);
39  if nargin > 2
40      assert(nargin==4);
41      assert(changerate<=1);
42      assert(changerate>=0);
43      assert(~isempty(find(p==[0,1,0.5],1)));
44  end
45  % more local variables
46  freemen = [(1:n)',ones(n,1)]; % column 1= men; column 2= 1 -> man is free, ...
        0 -> man isn't free
47  engaged = zeros(n,2);% column 1= men; column 2= women
48  dumped=0; % no of dumps
49  acceptrate = 0.75; % rate at which unknown nodes are accepted
50  % main loop
51  while ~isempty(find(freemen(:,2)==1,1)) % iterate as long as there are free men
52      % preference changes
53      if nargin > 2 % only if changerate and p are given
54          if rand < changerate % change prefs?
55              node = randi(n); % node whose prefs to change
56              if rand < p % change for men or women
57                  pref = nonzeros(m(node,:))';
58                  len = size(pref,2);
59                  if len>1
60                      k = randi([2,len]); % where in pref to change
61                      girl1 = pref(k); % the girl to swap
62                      i1 = find(initialm(node,:)==girl1,1); % index of girl1 ...
                            in initialm
63                      girl2 = m(node, k-1); % girl to be swapped with
64                      i2 = find(initialm(node,:)==girl2,1); % index of girl2 ...
                            in initialm
65                      initialm(node, i2) = girl1;
66                      initialm(node, i1) = girl2;
67                      m(node, i1) = girl2;
68                      m(node, i1-1) = girl1;
69                  end
70              else
71                  pref = nonzeros(f(node,:))';
72                  len = size(pref,2);
73                  if len>1
74                      k = randi([2,len]); % where in pref to change
75                      man1 = pref(k); % the man to swap
76                      i1 = find(initialf(node,:)==man1,1); % index of man1 in ...
                            initialf
77                      man2 = f(node, k-1); % man to be swapped with
```

```matlab
78                      i2 = find(initialf(node,:)==man2,1); % index of man2 in ...
                            initialf
79                      initialf(node, i2) = man1;
80                      initialf(node, i1) = man2;
81                      f(node, i1) = man2;
82                      f(node, i1-1) = man1;
83                  end
84              end %if_2
85              vprintf('preferences changed\n');
86          end %if_1
87      end
88      % +++
89      theman = find(freemen(:,2)==1,1); % the first man free on the list
90      thegirl = m(theman,1); % his first choice
91          if thegirl==0; % theman doesn't know any free girls who want him, ...
                he'll be alone :(
92              freemen(theman,2)=0;
93              engaged(theman,:)=0;
94          else
95              index = find(engaged(:,2)==thegirl,1); % index of possible ...
                    fiance of his first choice
96              if(isempty(index) ) % thegirl is free -> theman will be engaged ...
                    to thegirl
97                  if isempty(find(f(thegirl,:)==theman,1))
98                      vprintf('man %d proposed to women %d, she does not know ...
                            him\n', theman, thegirl);
99                      if rand>acceptrate % man accepts with a certain rate
100                         engaged(theman,1) = theman; % make new engagement
101                         engaged(theman,2) = thegirl;
102                         vprintf('she accepts\nman %d is engaged to girl ...
                                %d\n', theman, thegirl);
103                         freemen(theman,2) = 0; % man is not free anymore
104                         f(thegirl,:) = [theman, f(thegirl,1:n-1)]; % update ...
                                preferences
105                         initialf(thegirl,:) = [theman, ...
                                initialf(thegirl,1:n-1)]; % also in initial ...
                                matrix (will be used for checking)
106                     else
107                         vprintf('she declines\n');
108                         m(theman,:) = [m(theman,2:n) 0]; % make pref list ...
                                of theman smaller
109                     end % if_4
110                 else
111                     engaged(theman,1) = theman; % make new engagement
112                     engaged(theman,2) = thegirl;
113                     vprintf('man %d is engaged to girl %d\n', theman, thegirl);
114                     freemen(theman,2) = 0; % man is not free anymore
115                 end % if_3
116             else % thegirl is already engaged -> check if thegirl prefers ...
                    theman to her fiance
```

```matlab
117                    fiance = engaged(index,1); % her fiance
118                    girlprefers = f(thegirl,:); % pref list of thegirl
119                    howgirllikestheman=find(girlprefers==theman,1); % themans ...
                           index on thegirls preferences list
120                    howgirllikesfiance=find(girlprefers==fiance,1); % fiances ...
                           index on thegirls preferences list
121                    if(isempty(howgirllikestheman)) % thegirl doesn't know ...
                           theman -> thegirl accepts with a certain rate
122                        if rand > 0.75
123                            % thegirl prefers theman -> update pref list
124                            f(thegirl,:) = [f(thegirl,1:howgirllikesfiance), ...
                                   theman, f(thegirl,howgirllikesfiance+1:n-1)];
125                            initialf(thegirl,:) = ...
                                   [initialf(thegirl,1:howgirllikesfiance), ...
                                   theman, ...
                                   initialf(thegirl,howgirllikesfiance+1:n-1)]; % ...
                                   also initial
126                        end % if_4
127                    end % if_3
128                    if(find(girlprefers==theman,1)<find(girlprefers==fiance,1)) ...
                           % thegirl prefers theman ->change engagement
129                        engaged(theman,1) = theman; % change fiance of the girl
130                        engaged(theman,2) = thegirl;
131                        engaged(fiance,1) = 0; % fiance is free again
132                        engaged(fiance,2) = 0;
133                        vprintf('girl %d dumped man %d for man %d\n', thegirl, ...
                               fiance, theman);
134                        dumped=dumped+1;
135                        freemen(theman,2) = 0;
136                        freemen(fiance,2) = 1;
137                    else
138                        m(theman,:) = [m(theman,2:n) 0]; % thegirl prefers her ...
                               fiance -> take thegirl out of themans preference list
139                    end % if_3
140                end % if_2
141            end % if_1
142 end % while
143 % result printing (suppressed if verbositiy set to 0)
144 if dumped==1
145     vprintf('\n%d man has been dumped for another\n\n', dumped);
146 else
147     vprintf('\n%d men have been dumped for others\n\n', dumped);
148 end % if
149 single = size(find(engaged(:,2)==0),1); % number of single nodes
150 if single==1
151     vprintf('There is %d single man/woman\n\n', single);
152 else
153     vprintf('There are %d single men/women\n\n', single);
154 end % if
```

```matlab
155 [stable, counter] = checkEngagements(engaged,initialm,initialf); % check ...
        the engagements
156 if (stable)
157     vprintf('marriages are stable\n');
158 else
159     vprintf('marriages are unstable\n');
160     if counter==1
161         vprintf('there is %d unstable mariage\n', counter);
162     else
163         vprintf('there are %d unstable mariages\n', counter);
164     end % if_2
165 end % if
166 % calculate optimality index
167 opt = 0;
168 for i = 1:n
169     he = i;
170     she = engaged(he,2);
171     if she~=0
172         hisindex = find(initialf(she,:)==he,1);
173         herindex = find(initialm(he,:)==she,1);
174     else
175         hisindex = n;
176         herindex = n;
177     end
178     opt = opt + hisindex + herindex;
179 end
180 opt = opt/(2*n*n);
181 vprintf('optimality index is %1.2f\n',opt);
182 % set output
183 output = zeros(1,4);
184 output(1,1) = counter;
185 output(1,2) = single;
186 output(1,3) = dumped;
187 output(1,4) = opt;
188 end
```

checkEngagements.m

```matlab
1 function [ stable,counter ] = checkEngagements( engaged, m, f )
2 %checkEngagements checks whether a set of engagements is stable
3 %
4 %   men an women encoded as integers from 1 to n
5 %
6 %   input:
7 %   engaged: engagement matrix
8 %   m,f: preference matrices
9 %
10 %   dimensions must be correct, m=nxn, f=nxn, engaged=nx2
11 %
```

```matlab
12  %     returns:
13  %     stable: true for stable engagements, false otherwise
14  %     counter: the number of unstable mariages
15
16  n = size(m,1); % input size
17  % reverse the engaged matrix such that the new matrix has the index of the
18  % women on the column one and those of their respective husbands in row two
19  invengaged=zeros(n,2);
20  copy = engaged(:,[2,1]);
21  i=1;
22  while i~=n+1
23      index=copy(i,1);
24      while index==0 && i~=n % find first index that is nonzero
25          i=i+1;
26          index=copy(i,1);
27      end % while
28      if index==0 && i==n
29          break;
30      end % if
31      invengaged(index,:)=copy(i,:);
32      i=i+1;
33  end % while
34  % assign local variables
35  stable=true;
36  he=1;
37  counter=0;
38  inst = [0,0];
39  % main loop
40  while he<=n
41      she = engaged(he,2); % she is engaged to he
42      while (she==0 && he~=n) % he is not engaged, so there is no instability ...
              -> check the next man
43          he = he+1;
44          she = engaged(he,2);
45      end % while
46      if she==0 % -> he=n is not engaged, nothing to check.
47          break;
48      end %if
49      % get indexes in pref lists
50      hisindex = find(f(she,:)==he,1);
51      herindex = find(m(he,:)==she,1);
52      helikesbetter = m(he,1:herindex);
53      shelikesbetter = f(she,1:hisindex);
54      % check for her
55      if ~isempty(shelikesbetter) % there is no one on earth she likes better
56          for i=1:size(shelikesbetter) % loop to check if there is ...
                  unstability for the girl
57              guy = shelikesbetter(i); % all the guys she likes better
58              guysgirl = engaged(guy,2); % the guy she is engaged to
```

25

```matlab
59              if guysgirl == 0 && ~isempty(find(m(guy,:)== she,1)) % if this ...
                    guy isn't engaged, then she could be with him -> unstable, ...
                    unless he doesn't know her.
60                  stable = false;
61                  vprintf('man %d and woman %d like each other better\n', guy, ...
                        she);
62                  inst = [guy,she;inst];
63
64              else
65                  guylikes = m(guy,:); % the ordered preferences of guy
66                  if (find(guylikes==she,1)<find(guylikes==guysgirl,1)) % if ...
                        guy also likes she better than his wife -> unstable
67                      stable = false;
68                      vprintf('man %d and woman %d like each other better\n', ...
                            guy, she);
69                      inst = [guy,she;inst];
70                  end % if_3
71              end % if_2
72          end % for
73      end % if_1
74      % now the other way round, check for him
75      if ~isempty(helikesbetter) % there is no one on earth he likes better
76          for i=1:size(helikesbetter) % loop to check if there is unstability ...
                for the man
77              girl = helikesbetter(i); % all the girls he likes better
78              girlsguy = invengaged(girl,2); % the girl he is engaged to
79              if girlsguy == 0 && ~isempty(find(f(girl,:)== he,1))% if this ...
                    girl isn't engaged, then she could be with her -> unstable
80                  stable = false;
81                  vprintf('man %d and woman %d like each other better\n', he, ...
                        girl);
82                  inst = [he,girl;inst];
83              else
84                  girllikes = f(girl,:);% the ordered preferences of girl
85                  if (find(girllikes==he,1)<find(girllikes==girlsguy,1)) % if ...
                        guy also likes she better than his wife -> unstable
86                      stable = false;
87                      vprintf('man %d and woman %d like each other better\n', ...
                            he, girl);
88                      inst = [he,girl;inst];
89                  end % if_3
90              end % if_2
91          end % for
92      end % if_1
93
94      he=he+1; % go to the next man
95  end % while
96  % delete duplicate instabilities
97  inst = unique(inst, 'rows');
98  counter = size(inst,1)-1;
```

```
99 end
```

simulation.m

```matlab
1  function [ data ] = simulation( saveit )
2  %simulation perform simulation
3  %
4  %   input:
5  %   saveit: if 1 data is saved, if 0 not
6  %
7  %   returns:
8  %   data: simulation data
9
10
11 %simulation
12
13 % simulate match making
14 % n is 2et, t from 1 to 6
15 % radius is either constant or random
16 %   when constant, in 0.1:0.05:0.5
17 % frequency
18
19 global verbosity
20 verbosity = 0;
21
22 assert(~isempty(find(saveit==[0,1],1)));
23 tmax = 6;
24 t = 2.^(1:tmax);
25 r = 0.1:0.05:0.5;
26 data = zeros(tmax,10,4);
27 seed = rng;
28 if saveit==1
29     dirname = sprintf('data/%s',datestr(now,'yyyy_mm_dd_HH_MM_SS'));
30     mkdir(dirname);
31 end
32
33 % radius random
34 for i=1:tmax
35     n = t(i);
36     [a,b] = generatePlane(n,2);
37     [x,y] = makeMatch(a,b);
38     data(i,10,:) = y;
39 end
40
41 % radius const
42 for i=1:tmax
43     for j=1:9
44         n = t(i);
45         radius = r(j);
```

```
46          [a,b] = generatePlane(n,1,radius);
47          [x,y] = makeMatch(a,b);
48          data(i,j,:) = y;
49      end
50 end
51 % plot optimality index for each radius
52 hold on
53 handle = figure(1);
54 col = hsv(10);
55 %set(groot,'defaultAxesLineStyleOrder',{'-*',':','o'});
56 for i=1:10
57     plot(1:tmax,data(:,i,4),'color', col(i,:), 'marker', '*','linestyle','--');
58     title('optimality index for for different radiuses');
59
60 end
61 arr = ['r','a','n','d','o','m',' ',' ',' ',' ',' ',' '];
62 xlabel('input size 2^x');
63 ylabel('optimality index');
64 legend([num2str(r','radius %1.3f');arr]);
65 if saveit==1
66     saveas(handle,sprintf('%s/figure_1.pdf', dirname));
67 end
68 hold off
69
70 % plot no of dumps for each radius
71 handle = figure(2);
72 for i=1:10
73     subplot(3,4,i);
74     bar(1:tmax,data(:,i,3));
75     xlabel('input size 2^x');
76     ylabel('number of dumps');
77     ylim([0,100]);
78     if i~=10
79         title(sprintf('plotting #dumps for radius %1.3f',r(i)));
80     else
81         title('plotting #dumps for radius random');
82     end
83
84 end
85 if saveit==1
86     saveas(handle,sprintf('%s/figure_2.pdf', dirname));
87 end
88 % saving
89 if (saveit==1)
90     save(sprintf('%s/data.mat',dirname),'data','seed');
91 end
92
93 end
```

simulation2.m

```matlab
1  function [ data ] = simulation2( saveit )
2  %simulation perform simulation
3  %
4  %   input:
5  %   saveit: if 1 data is saved, if 0 not
6  %
7  %   returns:
8  %   data: simulation data
9
10
11 %simulation
12
13 % simulate match making
14 % n is 2et, t from 1 to 6
15 % radius is either constant or random
16 %   when constant, r=0.4
17 %makeMatch with preference changes for men, women, men/women and
18 %probability of changing in p=0.1:0.1:0.9
19
20 global verbosity
21 verbosity = 0;
22
23 assert(~isempty(find(saveit==[0,1],1)));
24 tmax = 11;
25 t = 2.^(1:tmax);
26 p = 0.0:0.25:1.0;
27 pmax=size(p,2);
28 m=12; %number of different makeMatch called in simulation2
29 data = zeros(tmax,m*pmax,4);
30 r= 0.1:0.15:0.4;
31 seed = rng;
32 if saveit==1
33     dirname = sprintf('data/%s',datestr(now,'yyyy_mm_dd_HH_MM_SS'));
34     mkdir(dirname);
35 end
36
37 % radius random
38 for i=1:tmax
39     n = t(i);
40     [a,b] = generatePlane(n,2);
41     fprintf('Plane with rand radius, size %d is generated\n', n);
42     for j=1:pmax
43         [x1,y1] = makeMatch(a,b, p(j),1);% x: engagement matrix; y(1): ...
                #unstable mariage, y(2): #single men/women, y(3): #dumps, y(4): ...
                optimality index
44         [x2,y2] = makeMatch(a,b, p(j),0);
45         [x3,y3] = makeMatch(a,b, p(j),0.5);
46         data(i,j,:) = y1;
47         data(i,pmax+j,:)=y2;
48         data(i,2*pmax+j,:)=y3;
```

```matlab
49      end
50  end
51
52  %constant radius in r=[0.1,0.25,0.4]
53  for i=1:tmax
54      n = t(i);
55      [a,b] = generatePlane(n,1, r(1));
56      fprintf('Plane with radius r=0.1, size %d is generated\n', n);
57      for j=1:pmax
58          [v1,w1] = makeMatch(a,b, p(j),1);% x: engagement matrix; y(1): ...
                    #unstable mariage, y(2): #single men/women, y(3): #dumps, y(4): ...
                    optimality index
59          [v2,w2] = makeMatch(a,b, p(j),0);
60          [v3,w3] = makeMatch(a,b, p(j),0.5);
61          data(i,3*pmax+j,:) = w1;
62          data(i,4*pmax+j,:) = w2;
63          data(i,5*pmax+j,:) = w3;
64      end
65  end
66
67  for i=1:tmax
68      n = t(i);
69      [a,b] = generatePlane(n,1, r(2));
70      fprintf('Plane with radius r=0.25, size %d is generated\n', n);
71      for j=1:pmax
72          [v1,w1] = makeMatch(a,b, p(j),1);% x: engagement matrix; y(1): ...
                    #unstable mariage, y(2): #single men/women, y(3): #dumps, y(4): ...
                    optimality index
73          [v2,w2] = makeMatch(a,b, p(j),0);
74          [v3,w3] = makeMatch(a,b, p(j),0.5);
75          data(i,6*pmax+j,:) = w1;
76          data(i,7*pmax+j,:) = w2;
77          data(i,8*pmax+j,:) = w3;
78      end
79  end
80
81  for i=1:tmax
82      n = t(i);
83      [a,b] = generatePlane(n,1, r(3));
84      fprintf('Plane with radius r=0.4, size %d is generated\n', n);
85      for j=1:pmax
86          [v1,w1] = makeMatch(a,b, p(j),1);% x: engagement matrix; y(1): ...
                    #unstable mariage, y(2): #single men/women, y(3): #dumps, y(4): ...
                    optimality index
87          [v2,w2] = makeMatch(a,b, p(j),0);
88          [v3,w3] = makeMatch(a,b, p(j),0.5);
89          data(i,9*pmax+j,:) = w1;
90          data(i,10*pmax+j,:) = w2;
91          data(i,11*pmax+j,:) = w3;
92      end
```

```
93  end
94
95
96
97
98  % saving
99  if (saveit==1)
100     save(sprintf('%s/data.mat',dirname),'data','seed');
101 end
102
103 end
```

simulation3.m

```
1   function [ data ] = simulation3( saveit , tmax )
2   %SIMULATION3 perform simulation
3   %
4   %    simfeld
5   %
6   %    input:
7   %    saveit: if 1 data is saved, if 0 not
8   %    tmax: maximal exponent form input size (base 2)
9   %
10  %    returns:
11  %    data: simulation data
12
13  % simulate match making
14  % n is 2^t, t from 1 to tmax
15  % radius is either constant or random
16  %    when constant, in 0.1:rstep:0.5
17  % frequency changerate 0:fstep:1
18  % in makeMatch argument list p=0.5 -> change pref for both men and women
19
20  global verbosity
21  verbosity = 0;
22
23  assert(~isempty(find(saveit==[0,1],1)));
24  if nargin < 2
25      tmax = 6;
26  end
27  t = 2.^(1:tmax);
28  rstep = 0.1;
29  r = 0.1:rstep:0.5;
30  fstep = 0.5;
31  f = 0:fstep:1;
32  sizef = size(f,2);
33  sizer = size(r,2)+1;
34  m = 10; % number of iterations
35  data = zeros(tmax,sizer,sizef,m,4);
```

```matlab
36  % data dimensions:
37  % 1 input size n, tmax
38  % 2 radius, sizer
39  % 3 frequency, sizef
40  % 4 iterations, m
41  % 5 output values, 4
42  seed = rng;
43  disp(seed);
44  if saveit==1
45      dirname = sprintf('data/%s',datestr(now,'yyyy_mm_dd_HH_MM_SS'));
46      mkdir(dirname);
47  end
48  tic
49  fprintf('simulating for radius random\n');
50  % radius random
51  for i=1:tmax
52      n = t(i);
53      for k=1:sizef
54          freq = f(k);
55          for l=1:m
56              %[a,b] = generatePlane(n,2);
57              [a,b] = generateRandom(n);
58              fprintf('.');
59              [x,y] = makeMatch(a,b,freq,0.5);
60              data(i,sizer,k,l,:) = y;
61          end
62          fprintf('\n');
63      end
64      fprintf('n = %4d complete after %5.1f\n', n, toc);
65  end
66  if 1==0
67  fprintf('simuation for radius const\n')
68  % radius const
69  for i=1:tmax
70      n = t(i);
71      for j=1:sizer-1
72          radius = r(j);
73          for k=1:sizef
74              freq = f(k);
75              for l=1:m
76                  %[a,b] = generatePlane(n,1,radius);
77                  [a,b] = generateRandom(n);
78                  fprintf('.');
79                  [x,y] = makeMatch(a,b,freq,0.5);
80                  data(i,j,k,l,:) = y;
81              end
82              fprintf('\n');
83          end
84          fprintf('radius %1.1f complete\n', radius);
85      end
```

```
86        fprintf('n = %4d complete after %5.1f\n', n, toc);
87    end
88    end
89    % plotting
90    plot3(data);
91    % saving
92    if (saveit==1)
93        save(sprintf('%s/data.mat',dirname),'data','seed');
94    end
95
96    end
```

iteratesimulation.m

```
1    function [data, dataplot] = iteratesimulation(saveit, n)
2        global verbosity
3        verbosity=0;
4
5        %!!!tmax, m, p, t, pmax must correspond to simulation2!!!
6        assert(~isempty(find(saveit==[0,1],1)));
7        seed = rng;
8        tmax=11;
9        m=12; %number of makeMatch called in simulation2
10       p = 0.0:0.25:1.0;
11       pmax=size(p,2);
12       t = 2.^(1:tmax);
13       data = zeros(tmax,m*pmax,4,n);
14       dataplot=zeros(tmax,m*pmax,4);
15       if saveit==1
16           dirname = sprintf('data/%s',datestr(now,'yyyy_mm_dd_HH_MM_SS'));
17           mkdir(dirname);
18       end
19       for i=1:n
20           data(:,:,:,i)=simulation2(0);
21           dataplot=dataplot+data(:,:,:,i);
22           fprintf('iteration %d is done\n', i);
23       end
24       dataplot=dataplot/(n*1.);
25
26
27
28    % saving
29    if (saveit==1)
30        save(sprintf('%s/data.mat',dirname),'data','dataplot','seed');
31    end
32
33    end
```

plot3.m

```matlab
1  function [  ] = plot3( data )
2  %PLOT3 plotting for optimality index analysis
3  %
4  %   simfeld
5  if 1==0
6  dirname = sprintf('data/%s',datestr(now,'yyyy_mm_dd_HH_MM_SS'));
7  mkdir(dirname);
8  % define arrays
9  sizer = size(data,2);
10 r=0.1:0.1:0.5;
11 f=0:0.5:1;
12 sizef=size(f,2);
13 tmax = size(data,1);
14 t=1:tmax;
15
16 % x-axis n, diff radius, plots freq
17 col = hsv(sizer);
18 for j=1:sizef
19     freq = f(j);
20     handle = figure(j);
21     set(gca,'FontSize',16);
22     hold on
23     for i=1:sizer
24         mm = squeeze(mean(data(:,i,j,:,4),4));
25         st = squeeze(std(data(:,i,j,:,4),0,4));
26         %plot(1:tmax,data(:,i,1,1,4),'color', col(i,:), 'marker', ...
                '*','linestyle','--');
27         errorbar(1:tmax,mm,st,'color', col(i,:), 'marker', ...
                '.','linestyle','-.', 'LineWidth', 2);
28     end
29     hold off
30     box on
31     grid on
32     title(sprintf('optimality index for for different radii, changerate is ...
            %1.1f',freq));
33     arr = ['r','a','n','d','o','m',' ',' ',' ',' '];
34     xlabel('input size 2^x');
35     ylabel('optimality index');
36     ylim([0,1.1]);
37     xlim([0,11]);
38     legend([num2str(r','radius %1.1f');arr]);
39     saveas(handle,sprintf('%s/figure_%d.pdf', dirname, j));
40 end
41
42 % x-axis radius, diff n, plots freq
43 col = hsv(tmax);
44 for j=1:sizef
```

```matlab
45      freq = f(j);
46      handle = figure(j+3);
47      set(gca,'FontSize',16);
48      hold on
49      for i=1:tmax
50          mm = squeeze(mean(data(i,:,j,:,4),4));
51          st = squeeze(std(data(i,:,j,:,4),0,4));
52          %plot(1:tmax,data(:,i,1,1,4),'color', col(i,:), 'marker', ...
                '*','linestyle','--');
53          errorbar(0.1:0.1:0.6,mm,st,'color', col(i,:), 'marker', ...
                '.','linestyle','-.', 'LineWidth', 2);
54      end
55      hold off
56      box on
57      grid on
58      title(sprintf('optimality index for for different n, changerate is ...
            %1.1f',freq));
59      %arr = ['r','a','n','d','o','m',' ',' ',' ',' ',' ',' '];
60      xlabel('radius, 0.6 is random');
61      ylabel('optimality index');
62      ylim([0,1.1]);
63      xlim([0,0.9])
64      legend(num2str(t','n = 2^{%2d}'));
65      saveas(handle,sprintf('%s/figure_%d.pdf', dirname, j+3));
66  end
67
68  % x-axis freq, diff n, plots radius
69  col = hsv(tmax);
70  for j=1:sizer
71      if j~= sizer radius = r(j); end
72      handle = figure(j+6);
73      set(gca,'FontSize',16);
74      hold on
75      for i=1:tmax
76          mm = squeeze(mean(data(i,j,:,:,4),4));
77          st = squeeze(std(data(i,j,:,:,4),0,4));
78          %plot(1:tmax,data(:,i,1,1,4),'color', col(i,:), 'marker', ...
                '*','linestyle','--');
79          errorbar(f,mm,st,'color', col(i,:), 'marker', '.','linestyle','-.', ...
                'LineWidth', 2);
80      end
81      hold off
82      box on
83      grid on
84      if j~= sizer
85          title(sprintf('optimality index for for different n, radius ...
                %1.1f',radius));
86      else
87          title(sprintf('optimality index for for different n, radius random'));
88      end
```

35

```matlab
89      %arr = ['r','a','n','d','o','m',' ',' ',' ',' ',' ',' ',' '];
90      xlabel('changerate');
91      ylabel('optimality index');
92      ylim([0,1.1]);
93      xlim([-0.1,1.4]);
94      legend(num2str(t','n = 2^{%2d}'));
95      saveas(handle,sprintf('%s/figure_%d.pdf', dirname, j+6));
96  end
97
98  % surfs
99  handle = figure(13);
100 [a,b]=meshgrid(1:10,0.1:0.1:0.5);
101 hold on;
102 mm = squeeze(mean(data(:,1:5,3,:,4),4));
103 view(0,90);
104 box on
105 grid on
106 surf(a,b,mm','EdgeColor','none');
107 colorbar;
108 set(gca,'FontSize',16);
109 xlim([1,10]);
110 xlabel('input size2^x');
111 ylabel('radius');
112 saveas(handle,sprintf('%s/figure_13.pdf', dirname));
113 end
114 %generaterandom
115 handle = figure(14);
116 set(gca,'FontSize',16);
117 box on
118 dd=zeros(10,20,4);
119 for i=1:10
120     n = 2^i;
121     for j=1:20
122         [a,b] = generateRandom(n);
123         [x,y] = makeMatch(a,b);
124         dd(i,j,:) = y;
125         fprintf('.');
126     end
127     fprintf('\n');
128 end
129 mm = squeeze(mean(dd(:,:,4),2));
130 st = squeeze(std(dd(:,:,4),0,2));
131 errorbar(1:10,mm,st, 'marker', '.','linestyle','-.', 'LineWidth', 2);
132 %plot(1:10,log2(mm), 'marker', '.','linestyle','-.', 'LineWidth', 2);
133 p = polyfit(1:10,log2(mm'),1);
134 % line = polyval(p,1:10);
135 hold on;
136 x=1:10;
137 plot(x,(2^p(2))*(x.^(p(1))));
138 p
```

```
139  xlabel('input size 2^x');
140  ylabel('optimality index');
141  title('optimality index for randomly generated preferences');
142  %ylim([0,1.1]);
143  xlim([0,11]);
144  box on
145  grid on
146  %saveas(handle,sprintf('%s/figure_14.pdf', dirname));
147  end
```

10 Appendix B: Plots

Additional Plots

10.1 Optimality Index

optimality index for for different radii, changerate is 0.0

optimality index for for different radii, changerate is 0.5

optimality index for for different radii, changerate is 1.0

optimality index for for different n, changerate is 0.0

optimality index for for different n, changerate is 0.5

42

optimality index for for different n, changerate is 1.0

optimality index for for different n, radius 0.1

optimality index for for different n, radius 0.2

optimality index for for different n, radius 0.3

optimality index for for different n, radius 0.4

optimality index for for different n, radius 0.5

optimality index for for different n, radius random

optimality index for randomly generated preferences