ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

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.

[illegible]

Valentin Junet

[illegible]

Samuel Imfeld

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	[illegible]
	[illegible]

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:

1. In the original model every node knows all the other nodes of the opposite gender. In a setting like a database of some match-making site this may be true. But as soon as the number of nodes gets big, it costs a huge amount of computation time to consider all nodes. In reality, information about the nodes of opposite is never complete (this would mean knowing about 3.5 billion people).
2. It is also conceivable that at some point a node might change its opinion about other nodes and rearrange them in his preference rating

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 and 6.1 whereas Samuel concentrated on chapters 1, 5 and 6.2.

3 Introduction and Motivations

What is stability? How to find its optimality? -Well, actually, whats an optimal stability?-. Of course, answering those questions would be the work of a philosophical essay which is not our pretention. We will only be considering some very abstract and theoretical nodes which, for some practical and rhetorical reasons, will be called men or women. Although not in some philosophical sense stability and other related parameters (which were going to introduce soon enough) will be the center of our attention. We will see how they relate to our lovely nodes and interpret their correlations. The key point of this work is to generate random links or friendship between men and women, to classify those in a random order, to apply the algorithm of Gale and Shapley which match together in an optimal way men and women according to their preferences, to modify some parameters and then interpret the results. For a more precise description of the procedure, you may refer to chapter 4, Description of the Model.

The stable marriage problem is quite well known; it is originally an interesting mathematical problem, but it also has something catchy, everyone has its own idea of what is stability, how we think it is optimal, and how we would like to be matched, or how we think it should be done and there comes this emotionless algorithm who pretends -with the support of some mathematical proofs- to find all of this for us and to do it as least as good as we would. Of course, this is exaggerated but when we think about those internet sites who match people according to some theoretical criteria, isnt it what were doing? To let some numbers find an objective answer to those questions, to our subjective questions? Solets take one of the most fundamental of those matching algorithm and instead of letting it answer our questions, see what question we can extract from it.

4 Description of the Model

4.1 Gale and Shapleys algorithm

Lets first describe the algorithm of Gale and Shapley. First every man, resp. woman, classifies the women, resp. men, according to his, resp. her, preferences. Then the algorithm follows those steps:

1. Single men propose to their favorite women.
2.
   - Women accept their favorite suitor or their only suitor.
   - If a woman has a fianc and a suitor, she chooses the one she likes better.
   - If a woman rejects a man, then his new favorite woman is the next on his list.
3. If the single men have still women on their list of preferences, go to step 1.

So the algorithm terminates, when the single men have no women left on their list of preferences. Gale and Shapley proved that this algorithm always terminates with an optimal stability. According to their definitions, an unstable assignment occurs when there is a man or a woman likes someone else than his/her wife/husband better and this someone likes him/her better to than her/his husband/wife. If there isnt any unstable assignment, then the marriages are called stable. They also gave a precise definition of what optimality means:

Definition 1: *A stable assignment is called optimal if every applicant is at least as well* *off under it as under any other assignment.*[1]

Definition 2: *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.*[2]

For more precise information you may refer to the quoted paper of Gale and Shapley.

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 since, for example, when a man goes in a bar, he doesnt particularly make a list of the women he prefers and propose to each woman according to his list (or at least this is not systematically observed). However there is some modification we can make to make it more realistic. One of these is that every men dont 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 doesnt know a woman who lives in Switzerland, so he wont propose to her; but we didnt 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 doesnt 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 propose to a woman) with a certain probability which is fixed.

We also introduced some new concepts, like the number of dumps, or an optimality index. The number of dumps is the number of time the women rejected their fianc for another man, this can be seen has an indication of time, the more people get dumped, the longer it is to reach a final assignment. The optimality index gives us an information about how good is the method we are using with regards to the list of preferences of each man and woman. For example the optimality index decrease if a man is married to his fifth choice, rather than his second. It takes in account all the men and women, it contains therefore an information about the method.

A more precise description of how we did these modifications will be presented in the next chapter.

[1]Gale and Shapley [1962]
[2]Gale and Shapley [1962]

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 and 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.

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.

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).

```
function [ engaged, stable ] = makeMatch( m, f )
%makeMatch finds engagements for preferences according to Gale–Shapley ...
    algorithm
...
freemen = [(1:n)',ones(n,1)];
engaged = zeros(n,2);
while ~isempty(find(freemen(:,2)==1,1))
    theman = find(freemen(:,2)==1,1);
    thegirl = m(theman,1);
    index = find(engaged(:,2)==thegirl,1);
    if(isempty(index) )
        engaged(theman,1) = theman;
        engaged(theman,2) = thegirl;
        freemen(theman,2) = 0;
    else
        fiance = engaged(index,1);
        girlprefers = f(thegirl,:);
        if(find(girlprefers==theman,1)<find(girlprefers==fiance,1))
            engaged(theman,1) = theman;
            engaged(theman,2) = thegirl;
            engaged(fiance,1) = 0;
            engaged(fiance,2) = 0;
            freemen(theman,2) = 0;
            freemen(fiance,2) = 1;
        else
            m(theman,:) = [m(theman,2:n) 0];
        end
    end
end
stable = checkEngagements(engaged,m,f);
...
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`.
  - 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.

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 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 $c \in [0, 1]$

The resulting output variables are:

- no. of instabilities
- no. of dumps
- no. of singles
- optimality index, as defined above

6 Simulation Results and Discussion

6.1 part valentin

6.2 part samuel

7 Summary and Outlook

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`.

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

9 Appendix: MATLAB Codes

generateRandom.m

```
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

```
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
%   they have a limited visibility radius
%   n: number of men and women
%   mode: visibility radius mode, optional argument
%     1 --> const, one constant radius for all nodes
%     2 --> random, a new random radius is generated in each iteration
%           value is between 0.1 and 0.5
%   default mode is const
%   mpref: mens preferences in nxn matrix
%   fpref: womens preferences in nxn matrix

global verbosity

if (nargin >= 2 && mode == 1)
    assert(nargin==3);
    r = radius;
end
if(nargin < 2)
    mode = 1;
    r = 0.2;%default value
end

% generate random coordinates
% and extend to torus
men = zeros(3,9*n);
rnd = rand(2,n);
men(:,(0*n)+1:1*n)=[(1:n);rnd];
men(:,(1*n)+1:2*n)=men(:,(0*n)+1:1*n)+[zeros(1,n);ones(1,n);zeros(1,n)];
men(:,(2*n)+1:3*n)=men(:,(0*n)+1:1*n)+[zeros(1,n);ones(1,n);ones(1,n)];
men(:,(3*n)+1:4*n)=men(:,(0*n)+1:1*n)+[zeros(1,n);zeros(1,n);ones(1,n)];
men(:,(4*n)+1:5*n)=men(:,(0*n)+1:1*n)+[zeros(1,n);-ones(1,n);ones(1,n)];
men(:,(5*n)+1:6*n)=men(:,(0*n)+1:1*n)+[zeros(1,n);-ones(1,n);zeros(1,n)];
men(:,(6*n)+1:7*n)=men(:,(0*n)+1:1*n)+[zeros(1,n);-ones(1,n);-ones(1,n)];
men(:,(7*n)+1:8*n)=men(:,(0*n)+1:1*n)+[zeros(1,n);zeros(1,n);-ones(1,n)];
men(:,(8*n)+1:9*n)=men(:,(0*n)+1:1*n)+[zeros(1,n);ones(1,n);-ones(1,n)];

women = zeros(3,9*n);
rnd = rand(2,n);
women(:,(0*n)+1:1*n)=[(1:n);rnd];
women(:,(1*n)+1:2*n)=women(:,(0*n)+1:1*n)+[zeros(1,n);ones(1,n);zeros(1,n)];
women(:,(2*n)+1:3*n)=women(:,(0*n)+1:1*n)+[zeros(1,n);ones(1,n);ones(1,n)];
women(:,(3*n)+1:4*n)=women(:,(0*n)+1:1*n)+[zeros(1,n);zeros(1,n);ones(1,n)];
women(:,(4*n)+1:5*n)=women(:,(0*n)+1:1*n)+[zeros(1,n);-ones(1,n);ones(1,n)];
women(:,(5*n)+1:6*n)=women(:,(0*n)+1:1*n)+[zeros(1,n);-ones(1,n);zeros(1,n)];
women(:,(6*n)+1:7*n)=women(:,(0*n)+1:1*n)+[zeros(1,n);-ones(1,n);-ones(1,n)];
women(:,(7*n)+1:8*n)=women(:,(0*n)+1:1*n)+[zeros(1,n);zeros(1,n);-ones(1,n)];
women(:,(8*n)+1:9*n)=women(:,(0*n)+1:1*n)+[zeros(1,n);ones(1,n);-ones(1,n)];

%plotting
% if verbosity~=0
%     plot(men(2,1:n),men(3,1:n),'o',women(2,1:n),women(3,1:n),'o');
%     label1 = cellstr( num2str(women(1,1:n)') );
%     label2 = cellstr( num2str(men(1,1:n)') );
%     text(women(2,1:n),women(3,1:n),label1);
%     text(men(2,1:n),men(3,1:n),label2);
%     title('nodes in plane');
%     legend('men','women');
% end

d = zeros(2,9*n);
mpref = zeros(n,n);
fpref = zeros(n,n);


for i=1:n
    man = men(:,i);
    for j=1:9*n
        woman = women(:,j);
        d(:,j) = [woman(1,1);norm(man(2:3)-woman(2:3),2)];
    end
    if mode==2
        r = rand*0.4+0.1;
    end
    index = find(d(2,:)<r);
    available = women(:,index);
    sz = size(available,2);
    if sz>n
        available = available(:,1:n);
        sz = n;
    end
    perm = randperm(sz);
    mpref(i,1:sz) = available(1,perm);
end
for i=1:n
    woman = women(:,i);
    for j=1:9*n
        man = men(:,j);
        d(:,j) = [man(1,1);norm(man(2:3)-woman(2:3),2)];
    end
    if mode==2
        r = rand*0.4+0.1;
    end
    index = find(d(2,:)<r);
    available = men(:,index);
    sz = size(available,2);
    if sz>n
        available = available(:,1:n);
        sz = n;
    end
    perm = randperm(sz);
    fpref(i,1:sz) = available(1,perm);
end
end
```

vprintf.m

```
function vprintf(varargin)
% VPRINTF controlled printing
%
global verbosity
if verbosity~=0
    fprintf(varargin{:});
end
```

makeMatch.m

```
function [ engaged, output ] = makeMatch( m, f, changerate, p )
%makeMatch finds engagements for preferences according to Gale-Shapley ...
    algorithm
%
%   men an women encoded as integers from 1 to n
%
%   input:
%   m: preference matrix of the men. Each row corresponds to a man and
%   the elements are the women listed according to his preferences.
%   f: preference matrix of the women. Each row corresponds to a woman and
%   the elements are the men listed according to her preferences.
%   changerate: rate at which preference changes are performed, e. g. if
%   changerate=0.2 then only in 20% of iterations preferences are changed
%   p: change preferences for men (p=1) / women (p=0.5) / both (p=0.5)
%
%   dimensions must be correct, m=nxn, f=nxn.
%
%   returns:
%   engaged: nx2 Matrix containing matches
%   output: output data -->
%   output(1,1): number of instabilities
%   output(1,2): number of singles
%   output(1,3): number of dumps
%   output(1,4): optimality index

% optional test prints
```

```
global verbosity
vprintf('mens preferences:\n');
if verbosity~=0 disp(m); end
vprintf('womens preferences:\n')
if verbosity~=0 disp(f); end
% assign local variables
initialm = m;
initialf = f;
n = size(m,1);
n2 = size(f,1);
% make sure dimensions agree
assert(n == size(m,2));
assert(n==n2);
if nargin > 2
    assert(nargin==4);
    assert(changerate<=1);
    assert(changerate>=0);
    assert(~isempty(find(p==[0,1,0.5],1)));
end
% more local variables
freemen = [(1:n)',ones(n,1)]; % column 1= men; column 2= 1 -> man is free, ...
    0 -> man isn't free
engaged = zeros(n,2);% column 1= men; column 2= women
dumped=0; % no of dumps
acceptrate = 0.75; % rate at which unknown nodes are accepted
% main loop
while ~isempty(find(freemen(:,2)==1,1)) % iterate as long as there are free men
    % preference changes
    if nargin > 2 % only if changerate and p are given
        if rand < changerate % change prefs?
            node = randi(n); % node whose prefs to change
            if rand < p % change for men or women
                pref = nonzeros(m(node,:))';
                len = size(pref,2);
                if len>1
                    k = randi([2,len]); % where in pref to change
                    girl1 = pref(k); % the girl to swap
                    i1 = find(initialm(node,:)==girl1,1); % index of girl1 ...
                        in initialm
                    girl2 = m(node, k-1); % girl to be swapped with
                    i2 = find(initialm(node,:)==girl2,1); % index of girl2 ...
                        in initialm
                    initialm(node, i2) = girl1;
                    initialm(node, i1) = girl2;
                    m(node, i1) = girl2;
                    m(node, i1-1) = girl1;
                end
            else
                pref = nonzeros(f(node,:))';
                len = size(pref,2);
                if len>1
                    k = randi([2,len]); % where in pref to change
                    man1 = pref(k); % the man to swap
                    i1 = find(initialf(node,:)==man1,1); % index of man1 in ...
                        initialf
                    man2 = f(node, k-1); % man to be swapped with
                    i2 = find(initialf(node,:)==man2,1); % index of man2 in ...
                        initialf
                    initialf(node, i2) = man1;
                    initialf(node, i1) = man2;
                    f(node, i1) = man2;
                    f(node, i1-1) = man1;
                end
            end %if_2
            vprintf('preferences changed\n');
        end %if_1
    end
    % +++
    theman = find(freemen(:,2)==1,1); % the first man free on the list
    thegirl = m(theman,1); % his first choice
        if thegirl==0; % theman doesn't know any free girls who want him, ...
            he'll be alone :(
            freemen(theman,2)=0;
            engaged(theman,:)=0;
        else
            index = find(engaged(:,2)==thegirl,1); % index of possible ...
                fiance of his first choice
            if(isempty(index) ) % thegirl is free -> theman will be engaged ...
                to thegirl
                if isempty(find(f(thegirl,:)==theman,1))
                    vprintf('man %d proposed to women %d, she does not know ...
                        him\n', theman, thegirl);
                    if rand>acceptrate % man accepts with a certain rate
                        engaged(theman,1) = theman; % make new engagement
                        engaged(theman,2) = thegirl;
                        vprintf('she accepts\nman %d is engaged to girl ...
                            %d\n', theman, thegirl);
                        freemen(theman,2) = 0; % man is not free anymore
                        f(thegirl,:) = [theman, f(thegirl,1:n-1)]; % update ...
                            preferences
                        initialf(thegirl,:) = [theman, ...
                            initialf(thegirl,1:n-1)]; % also in initial ...
                            matrix (will be used for checking)
                    else
                        vprintf('she declines\n');
                        m(theman,:) = [m(theman,2:n) 0]; % make pref list ...
                            of theman smaller
                    end % if_4
                else
                    engaged(theman,1) = theman; % make new engagement
```

```matlab
                    engaged(theman,2) = thegirl;
                    vprintf('man %d is engaged to girl %d\n', theman, thegirl);
                    freemen(theman,2) = 0; % man is not free anymore
                end % if_3
            else % thegirl is already engaged -> check if thegirl prefers ...
                theman to her fiance
                fiance = engaged(index,1); % her fiance
                girlprefers = f(thegirl,:); % pref list of thegirl
                howgirllikestheman=find(girlprefers==theman,1); % themans ...
                    index on thegirls preferences list
                howgirllikesfiance=find(girlprefers==fiance,1); % fiances ...
                    index on thegirls preferences list
                if(isempty(howgirllikestheman)) % thegirl doesn't know ...
                    theman -> thegirl accepts with a certain rate
                    if rand > 0.75
                        % thegirl prefers theman -> update pref list
                        f(thegirl,:) = [f(thegirl,1:howgirllikesfiance), ...
                            theman, f(thegirl,howgirllikesfiance+1:n-1)];
                        initialf(thegirl,:) = ...
                            [initialf(thegirl,1:howgirllikesfiance), ...
                            theman, ...
                            initialf(thegirl,howgirllikesfiance+1:n-1)]; % ...
                            also initial
                    end % if_4
                end % if_3
                if(find(girlprefers==theman,1)<find(girlprefers==fiance,1)) ...
                    % thegirl prefers theman ->change engagement
                    engaged(theman,1) = theman; % change fiance of the girl
                    engaged(theman,2) = thegirl;
                    engaged(fiance,1) = 0; % fiance is free again
                    engaged(fiance,2) = 0;
                    vprintf('girl %d dumped man %d for man %d\n', thegirl, ...
                        fiance, theman);
                    dumped=dumped+1;
                    freemen(theman,2) = 0;
                    freemen(fiance,2) = 1;
                else
                    m(theman,:) = [m(theman,2:n) 0]; % thegirl prefers her ...
                        fiance -> take thegirl out of themans preference list
                end % if_3
            end % if_2
        end % if_1
end % while
% result printing (suppressed if verbositiy set to 0)
if dumped==1
    vprintf('\n%d man has been dumped for another\n\n', dumped);
else
    vprintf('\n%d men have been dumped for others\n\n', dumped);
end % if
single = size(find(engaged(:,2)==0),1); % number of single nodes
```

```
if single==1
    vprintf('There is %d single man/woman\n\n', single);
else
    vprintf('There are %d single men/women\n\n', single);
end % if
[stable, counter] = checkEngagements(engaged,initialm,initialf); % check ...
    the engagements
if (stable)
    vprintf('marriages are stable\n');
else
    vprintf('marriages are unstable\n');
    if counter==1
        vprintf('there is %d unstable mariage\n', counter);
    else
       vprintf('there are %d unstable mariages\n', counter);
    end % if_2
end % if
% calculate optimality index
opt = 0;
for i = 1:n
    he = i;
    she = engaged(he,2);
    if she~=0
        hisindex = find(initialf(she,:)==he,1);
        herindex = find(initialm(he,:)==she,1);
    else
        hisindex = n;
        herindex = n;
    end
    opt = opt + hisindex + herindex;
end
opt = opt/(2*n*n);
vprintf('optimality index is %1.2f\n',opt);
% set output
output = zeros(1,4);
output(1,1) = counter;
output(1,2) = single;
output(1,3) = dumped;
output(1,4) = opt;
end
```

checkEngagements.m

```
function [ stable,counter ] = checkEngagements( engaged, m, f )
%checkEngagements checks whether a set of engagements is stable
%
%   men an women encoded as integers from 1 to n
%
%   input:
```

```
%   engaged: engagement matrix
%   m,f: preference matrices
%
%   dimensions must be correct, m=nxn, f=nxn, engaged=nx2
%
%   returns:
%   stable: true for stable engagements, false otherwise
%   counter: the number of unstable mariages

n = size(m,1); % input size
% reverse the engaged matrix such that the new matrix has the index of the
% women on the column one and those of their respective husbands in row two
invengaged=zeros(n,2);
copy = engaged(:,[2,1]);
i=1;
while i~=n+1
    index=copy(i,1);
    while index==0 && i~=n % find first index that is nonzero
        i=i+1;
        index=copy(i,1);
    end % while
    if index==0 && i==n
        break;
    end % if
    invengaged(index,:)=copy(i,:);
    i=i+1;
end % while
% assign local variables
stable=true;
he=1;
counter=0;
inst = [0,0];
% main loop
while he<=n
    she = engaged(he,2); % she is engaged to he
    while (she==0 && he~=n) % he is not engaged, so there is no instability ...
        -> check the next man
        he = he+1;
        she = engaged(he,2);
    end % while
    if she==0 % -> he=n is not engaged, nothing to check.
        break;
    end %if
    % get indexes in pref lists
    hisindex = find(f(she,:)==he,1);
    herindex = find(m(he,:)==she,1);
    helikesbetter = m(he,1:herindex);
    shelikesbetter = f(she,1:hisindex);
    % check for her
    if ~isempty(shelikesbetter) % there is no one on earth she likes better
```

```
        for i=1:size(shelikesbetter) % loop to check if there is ...
            unstability for the girl
            guy = shelikesbetter(i); % all the guys she likes better
            guysgirl = engaged(guy,2); % the guy she is engaged to
            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.
               stable = false;
               vprintf('man %d and woman %d like each other better\n', guy, ...
                   she);
               inst = [guy,she;inst];

            else
                guylikes = m(guy,:); % the ordered preferences of guy
                if (find(guylikes==she,1)<find(guylikes==guysgirl,1)) % if ...
                    guy also likes she better than his wife -> unstable
                    stable = false;
                    vprintf('man %d and woman %d like each other better\n', ...
                        guy, she);
                    inst = [guy,she;inst];
                end % if_3
            end % if_2
        end % for
    end % if_1
    % now the other way round, check for him
    if ~isempty(helikesbetter) % there is no one on earth he likes better
        for i=1:size(helikesbetter) % loop to check if there is unstability ...
            for the man
            girl = helikesbetter(i); % all the girls he likes better
            girlsguy = invengaged(girl,2); % the girl he is engaged to
            if girlsguy == 0 && ~isempty(find(f(girl,:)== he,1))% if this ...
                girl isn't engaged, then she could be with her -> unstable
                stable = false;
                vprintf('man %d and woman %d like each other better\n', he, ...
                    girl);
                inst = [he,girl;inst];
            else
                girllikes = f(girl,:);% the ordered preferences of girl
                if (find(girllikes==he,1)<find(girllikes==girlsguy,1)) % if ...
                    guy also likes she better than his wife -> unstable
                    stable = false;
                    vprintf('man %d and woman %d like each other better\n', ...
                        he, girl);
                    inst = [he,girl;inst];
                end % if_3
            end % if_2
        end % for
    end % if_1

    he=he+1; % go to the next man
```

```
end % while
% delete duplicate instabilities
inst = unique(inst, 'rows');
counter = size(inst,1)-1;
end
```

simulation.m

```
function [ data ] = simulation( saveit )
%simulation perform simulation
%
%   input:
%   saveit: if 1 data is saved, if 0 not
%
%   returns:
%   data: simulation data


%simulation

% simulate match making
% n is 2et, t from 1 to 6
% radius is either constant or random
%   when constant, in 0.1:0.05:0.5
% frequency

global verbosity
verbosity = 0;

assert(~isempty(find(saveit==[0,1],1)));
tmax = 6;
t = 2.^(1:tmax);
r = 0.1:0.05:0.5;
data = zeros(tmax,10,4);
seed = rng;
if saveit==1
    dirname = sprintf('data/%s',datestr(now,'yyyy_mm_dd_HH_MM_SS'));
    mkdir(dirname);
end

% radius random
for i=1:tmax
   n = t(i);
   [a,b] = generatePlane(n,2);
   [x,y] = makeMatch(a,b);
   data(i,10,:) = y;
end

% radius const
```

```
for i=1:tmax
    for j=1:9
        n = t(i);
        radius = r(j);
        [a,b] = generatePlane(n,1,radius);
        [x,y] = makeMatch(a,b);
        data(i,j,:) = y;
    end
end
% plot optimality index for each radius
hold on
handle = figure(1);
col = hsv(10);
%set(groot,'defaultAxesLineStyleOrder',{'-*',':','o'});
for i=1:10
    plot(1:tmax,data(:,i,4),'color', col(i,:), 'marker', '*','linestyle','--');
    title('optimality index for for different radiuses');

end
arr = ['r','a','n','d','o','m',' ',' ',' ',' ',' ',' '];
xlabel('input size 2^x');
ylabel('optimality index');
legend([num2str(r','radius %1.3f');arr]);
if saveit==1
    saveas(handle,sprintf('%s/figure_1.pdf', dirname));
end
hold off

% plot no of dumps for each radius
handle = figure(2);
for i=1:10
    subplot(3,4,i);
    bar(1:tmax,data(:,i,3));
    xlabel('input size 2^x');
    ylabel('number of dumps');
    ylim([0,100]);
    if i~=10
        title(sprintf('plotting #dumps for radius %1.3f',r(i)));
    else
        title('plotting #dumps for radius random');
    end

end
if saveit==1
    saveas(handle,sprintf('%s/figure_2.pdf', dirname));
end
% saving
if (saveit==1)
    save(sprintf('%s/data.mat',dirname),'data','seed');
end
```

```

end
```