% Online Dating Decision Services in MiniZinc.
% ""
% Consider the following situation. You have been approached by an online dating
% service that wants to use a rules engine to improve its process for matching
% people. Below is a brief explanation of "business logic" behind their online
% dating services:
% 
% - Each person creates a profile defining their preferences
% - The rules check the profiles to determine all the possible matches for
%   a person
% - The matches are scored. Higher scores indicate a better match
% - Scoring (once the matching criteria are met) is based on the age
%   difference and the number of matching interests
%
% Each profile includes:
% 
% - Name
% - Gender
% - City
% - Age
% - List of interests
% - Minimum and Maximum acceptable age
% - Acceptable genders
% - Minimum number of matching interests.
% 
% And here are the rules (applied to both persons):
% 
% - Gender of the other person must be one of the acceptable genders
% - Age of the other person must be within the acceptable range
% - City must match exactly
% - Matching interests of the other person must match at least the
%   number specified
% 
% They even provided an example of a compatible match:
% 
% Jane (age 26, lives in Seattle, interests are skydiving, knitting, reading)
% is looking for a male age 28-32 with at least one of those interests
% Jim (age 29, lives in Seattle, interests are skydiving, soccer, knitting)
% is looking for a female age 24-29 with at least two of those interests
% 
% Now, you need to build a working prototype to win their business. Can you do it?
% If yes, send your solution to decisionmanagementcommunity@gmail.com.
% ""
% Note: the data is from page 3 of
% https://dmcommunity.files.wordpress.com/2017/05/online-dating-decision-intelliops.pdf
% ""
% name     age  gender minMatches acceptableSex city         minAge maxAge minInterests in
% interests
% _____________________________________________________________
% | Jane   | 26   | F     | 1   | M     | Seattle   | 28    | 32    | 1   | sky diving, knitting, reading |
% | Jim    | 29   | M     | 2   | F     | Seattle   | 24    | 29    | 2   | sky diving, soccer, knitting |
% | Natalie| 31   | F     | 2   | M     | Aliso Viejo| 35    | 45    | 3   | soccer, paddle boarding, kayaking |
% Ashley 29 F 1 M Aliso Viejo 29 40 2 soccer, paddle boarding, hiking
% Nathan 34 M 2 F Aliso Viejo 24 29 2 paddle boarding, kayaking, skiing
% Lisa 23 F 1 M Aliso Viejo 25 30 3 soccer, skiing, reading, kayaking
% Ken 29 M 1 F Aliso Viejo 21 25 1 soccer, kayaking, skiing, reading
% Flavio 41 M 1 F Aliso Viejo 21 40 1 soccer, skiing, reading
% Andrew 43 M 1 F Aliso Viejo 21 35 1 soccer, kayaking, paddle boarding, reading

""
%
%
% There is no solution where all people are matched just to a unique person, i.e.
% a perfect assignment. I.e. all_different/1 cannot be used here. This make sense
% since the number of people is odd (4 female and 5 male).
%
% The constraints might be reciprocal, e.g. for a match A’s age constraint is
% satisfied by B and also thatB’s age constraint must be satisfied by A,
% otherwise there is no match.
%
% With the boolean "reciprocal" variable set to true then there are two solutions
% with 4 perfect matches, otherwise it’s 36 solutions (with a variety of perfect
% matches).
%
% Here are two solutions:
% - The first table is the (best) matches, i.e. Jane’s best match is Jim (and Jim’s
%   best match is Jane)
% - The second table is the list of of whom a person has been matched, i.e. Natalie
%   has been matched to both Flavio and Andrew and Andrew has not been picked
%   by anyone.
% - The optimal solution is the second solution with a combined score (z) of 158.
% The score is 10*number of common interests - sum of age differences
% - The solutions below show that for all except one there are complete matches.
%   Jane <-> Jim
%   Natalie <-> Flavio
%   Ashley <-> Nathan
%   Ken <-> Lisa
%
% Jane -> Jim
% Jim -> Jane
% Natalie -> Flavio
% Ashley -> Nathan
% Nathan -> Ashley
% Lisa -> Ken
% Ken -> Lisa
% Flavio -> Natalie
% Andrew -> Natalie
%
% Jane is matched to {Jim}
% Jim is matched to {Jane}
% Natalie is matched to {Flavio, Andrew}
% Ashley is matched to {Nathan}
% Nathan is matched to {Ashley}
% Lisa is matched to {Ken}
% Ken is matched to {Lisa}
% Flavio is matched to {Natalie}
% Andrew is matched to { }
This MiniZinc model was created by Hakan Kjellerstrand, hakank@gmail.com
See also my MiniZinc page: http://www.hakank.org/minizinc/

include "globals.mzn";

int: n = card(name);
enum name = {Jane,Jim,Natalie,Ashley,Nathan,Lisa,Ken,Flavio,Andrew};
enum sex = {female,male};
enum city = {Seattle,AlisoViejo};
enum interest =
    {sky_diving,knitting,reading,soccer,paddle_boarding,kayaking,hiking,skiing};

int: Age = 1;
int: Sex = 2;
int: MinMatches = 3;
int: City = 4;
int: MinAge = 5;
int: MaxAge = 6;
int: MinInterests = 7;

% data
% Note: I had to adjust the number of acceptable interests for some people.
array[name,1..7] of int: people =
array2d(name, 1..7,
[ 26,female,1, Seattle , 28, 32,1,
29,male,  ,2, Seattle , 24, 29,2,
31,female,2, AlisoViejo, 35, 45,2,
29,female,1, AlisoViejo, 29, 40,1,%2,
34,male,  ,2, AlisoViejo, 24, 29,1,%2,
23,female,1, AlisoViejo, 25, 30,1,
29,male,  ,1, AlisoViejo, 21, 25,1,
41,male,  ,1, AlisoViejo, 21, 40,1,
43,male,  ,1, AlisoViejo, 21, 35,1,
]);

% genders
array[name] of set of sex: gender =
[
  (male),
  (female),
  (male),
  (male),
  (female),
  (male),
  (female),
  (female),
  (female),
];

% interests
array[name] of set of interest: interests =
[
  {sky_diving,knitting,reading},                  % Jane
  {sky_diving,soccer,knitting},                   % Jim
  {soccer,paddle_boarding,kayaking},              % Natalie
  {soccer,paddle_boarding,hiking},                % Ashley
  {paddle_boarding,kayaking,skiing,reading},      % Nathan
  {soccer,skiing,reading,kayaking,reading},       % Lisa
  {soccer,kayaking,skiing,reading},               % Ken
  {soccer,skiing,reading,hiking,paddle_boarding}, % Flavio
  {soccer,kayaking,paddle_boarding,reading},      % Andrew
];

bool: reciprocal = true;

% decision variables
% best assignment
array[name] of var name: x;  % single assignment
array[name] of var set of name: y;  % multiple assignments
array[name] of var int: age_diff;
array[name] of var int: common_interests;
var int: z = 10*sum(common_interests) - sum(age_diff);
array[name] of var 0..1: complete_match;
var int: num_complete_matches = sum(complete_match);
% solve maximize z;
solve satisfy;
% solve :: int_search(x, first_fail, indomain_min, complete) satisfy;

constraint
  % all_different(x) /\
  
  % cannot be matched by oneself
  forall(p in name) (x[p] != p)
  ) /\
  forall(p in name) (
    % Note that these constraints might be reciprocal
    % (if "reciprocal" is set to true)
    
    % gender of the other person must be one of the
    % acceptable genders
    people[x[p],Sex] in gender[p] /\
    if reciprocal then people[p,Sex] in gender[x[p]] else true endif /\
    
    % age must be acceptable
    people[x[p],Age] in people[p,MinAge]..people[p,MaxAge] /\
    if reciprocal then people[p,Age] in people[x[p],MinAge]..people[x[p],MaxAge]
    else true
    endif /
    
    % city must match exactly
    people[x[p],City] = people[p,City] /\
    
    % Matching interests of the other person must match at least the
    % number specified
    common_interests[p] = card(interests[x[p]] intersect interests[p]) /\
    common_interests[p] >= people[p,MinInterests] /\
    
    % age difference (for the score)
    age_diff[p] = abs(people[p,Age]-people[x[p],Age]) /\
    complete_match[p] = if p = x[x[p]] then 1 else 0 endif
  )
  ;

% set of targets of the matches
constraint
  forall(p in name) (not(p in y[p]) /\
    forall(q in name) (
      x[q] = p <-> q in y[p]
    )
  )
  ;

output
["(p) -> \(x[p])\n"]
\[
| p \text{ in name} \\
| \text{++ } [
\text{"\n" } +
| \text{"(p) is matched to \(y[p]\)\n" } \\
| p \text{ in name} \\
| \text{++ }
| \text{"\nage_diff: \(age_diff\)\n", }
| \text{"common_interests: \(common_interests\)\n", }
| \text{"z: \(z\)\n", }
| \text{"complete_matches: \(complete_match\) num_complete_matches: \(num_complete_matches\):\n", }
| \text{++ }
| \text{if fix(complete_match[p]) = 1 } \text{\(\backslash p < \text{fix(x[p]) then}
| \text{" \(p) <-> \(x[p]\)\n" }
| \text{else}
| \text{"" }
| \text{endif}
| p \text{ in name} \\
| \text{;}
\]