# 1 Project proposal: Tindar

## 1.1 Motivation

I would like to work on this personal project to display my newly learned Machine Learning Engineering skills together with my Operations Research skills. Although I liked working with SageMaker during the course, I feel like too much functionality is abstracted away. To really sink in the deployment process, I would like to work on Tindar, an Operations Research that I came up with completely by myself. I really enjoy working on OR problems and having some creative freedom. I aim to advertise myself as someone skilled at OR, and this might be a good opportunity to showcase some of my thought-process and skills.

## 1.2 Domain Background

I call my project Tindar: the optimal matchmaker. Suppose you would have the swipe information of a dating app (Tindar), where the users express their interest in each other with a 1 (interested) or a 0 (not interested). Current dating apps are only focused on the individual, but Tindar tries to achieve the maximum happiness among the user community as a whole. Moreover, Tindar is more conservative: each user can only be paired up with a single other user.

## 1.3 Problem Statement

The goal of this project is to build an optimization model that pairs up as many users as possible. A match can only be made if both users have expressed mutual interest. As mentioned above, a user can only be coupled to one other user at most.

Once the model is defined mathematically, we will analyze its complexity and develop a fitting solution in Python. Furthermore, we investigate how the performance of our model is affected by the size of the Tindar community and its number of interest-edges:

1. Can we always find an optimal solution, even for large Tindar problems? If not, can we find a heuristic that provides a decent solution? How far is the heuristic's solution from the optimal solution (may be hard to anwer)?

2. How is the computation time affected by the size of the Tindar problem?

## 1.4 Datasets & Inputs

Because this problem is fictional, we do not know the interest matrix $A$. Therefore, we will randomly generate it. The data generation is thus part of the project.

The main concept is to generate matrix A based on a Bernouilli distribution (flipping a coin). This distribution is characterised by parameter $p$, the probability of one user expressing interest in another user.

Specifically, we will:

1. set $n$, the number of members in the community
2. set the Bernouilli parameter $p$
3. sample $n \times n$ elements from the Bernouilli(p) distribution
4. reshape the samples into an $n \times n$ matrix

In this way, we can generate as many Tindar datasets as we like, which supports our research goals.

## 1.5 Solution statement

Mathematically, the Tindar community can be seen as a graph, where each node represents a user, and a directed edge the interest of one user in another. The model tries to pick as many edges as possible without violating the constraints (mutual interest, max. 1 partner).

This maximum pairing problem can be formulated as a Binary Integer Linear Program. Let $(a_{i,j}) \in \mathbb{R}^{n \times n}$ be the interest matrix consisting of ones (if one user is interested in the other) and zeros (else). Then the model needs to decide which users to pair-up with the decision variables $x_{i,j}$:

$$x_{i,j} = \begin{cases} 1, & \text{if user i is coupled to user j} \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

### 1.5.1 Objective

The objective is simple: to pair up as many users as possible.

$$\max_{x_{i,j}} \sum_i \sum_j x_{i,j} \tag{2}$$

### 1.5.2 Constraints

There are a couple of constraints that the solution needs to obey.

The first is obvious: users can only be paired if they are mutually interested in each other.

$$x_{i,j} \leq a_{i,j} \quad \forall i, j \tag{3}$$

Next, pairing is obviously symmetric: if user i is paired to user j, then the reverse holds as well.

$$x_{i,j} = x_{j,i} \quad \forall i = 1, 2, ..., (n-1); j = (i+1), i, ..., n \tag{4}$$

We also have that a single user can only be assigned to 1 other person (one outgoing arc at most), and a user can be assigned to only a single other user (one incoming arc at most).

$$\sum_{j=1}^{n} x_{i,j} \leq 1 \quad \forall i = 1, 2, ..., n \tag{5}$$

$$\sum_{j=1}^{n} x_{i,j} \leq 1 \quad \forall i = 1, 2, ..., n \tag{6}$$

### 1.5.3 Full model

We thus have the Tindar model:

$$\begin{aligned}
\max_{x_{i,j}} \quad & \sum_i \sum_j x_{i,j} && && \text{(7a)} \\
\text{subject to} \quad & x_{i,j} \leq a_{i,j} && \forall i, j, && \text{(7b)} \\
& x_{i,j} = x_{j,i} \quad \forall i = 1, 2, ..., (n-1); j = (i+1), i, ..., n && && \text{(7c)} \\
& \sum_{j=1}^{n} x_{i,j} \leq 1 && \forall i = 1, 2, ..., n, && \text{(7d)} \\
& \sum_{i=1}^{n} x_{i,j} \leq 1 && \forall j = 1, 2, ..., n && \text{(7e)} \\
& x_{i,j} \in \{0, 1\} && \forall i = 1, 2, ..., n && \text{(7f)}
\end{aligned}$$

Note that all constraints are linear and the decision variables are binary. We are thus in a Binary Integer Linear Programming setting. Such problems can be tackled with algorithms like Branch & Bound, however for this project we will

use libraries that are ready to go. I have never properly experimented with the size of BiLP problems that todays software (CPLEX, Gurobi) is able to tackle, but my estimate is that the solvers should have no problem whatsoever with $n = 100$.

### 1.5.4 Python Implementation

I will use OOP to encapsulate the Tindar problem, its corresponding BiLP representation, and the methods for solving the problem. The model can be built and solved in Python using the PuLP library (https://coin-or.github.io/pulp/). We will use the default PuLP solver.

I will also write a small class to generate Tindar problems as described in Section 1.4. The Bernouilli draws can be done with Numpy.

## 1.6 Benchmark model

To benchmark the PuLP solution we will write a greedy heuristic, which will solve in $\mathcal{O}(n^2)$. This heuristic works as follows:

for i=1, ..., n

- get the i'th row of interest matrix A, which is the people that person i is interested in. Initialize match_made=False, j as small as possible with $j \neq i$. Then, while not match_made:
  1. if $a_{i,j} = 1$ and $a_{j,i} = 1$, set $x_{i,j} = x_{j,1} = 1$. set match_made=True
  2. else, $j = j + 1$ (skip $j == i$)

## 1.7 Evaluation metrics

The main evaluation metric is of course the optimisation objective. We will also look at computation time.

## 1.8 Project Design

I aim to build a minimalistic but powerful web-application with Flask, that:

- shows a visitor of the home page a little bit about of Tindar project and its concepts, and how to interact with the API endpoints
- allows a visitor to generate a Tindar problem and have its representation sent back to him
- allows a visitor to send a Tindar problem and have the solution together with other relevant information sent to back to him

Both API endpoints receive and send JSON data.

The Flask App will be deployed to AWS Lambda Functions. We will use AWS API Gateway to make the app available to anyone. This is very similar to the concepts explained during the course.