Polytechnical University of Catalonia

Algorithms

Stable Roommates Problem

Authors:
César Adrián Mellado
Benjamí Parellada
Ioan Vlad Balteanu

Tutor:
Maria Blesa Aguilera

2020/2021 Q2

Contents

Notation

Here is some of the notation we have used. There are other symbols present, however these are the main ones needed to understand the project. There is some different notation between the problem generation and analysis of results that is due to trying to keep things consistent with the codes.

- n_{room}: the amount of rooms available.
- n: number of people that need to share a room. $2 \cdot n_{room} = n$.
- P_n: Probability that an arbitrary roommates instance of size n is solvable.
- $m = n_{sets}$: different number of room sizes.
- $n_{sample,k} = n_{problems}$: number of samples with room size $k \in 1 \ldots m$.
- $\hat{p}$: estimate of solvable problems of the population proportion.
- ϵ: sampling error.
- α: $(1 - \alpha)$ confidence level, (which we will take as 95%).
- $z_{\alpha/2}$: critical value from the $z - table$, which for a confidence of 95%, we can take 1.96.
- CI: confidence interval.
- n_{stable}: number of stable problems observed in the experimentation.
- $n_{unstable}$: number of unstable problems observed in the experimentation.
- $\bar{t}_{stable}$: average time to find stable solutions.
- $\bar{t}_{unstable}$: average time to find unstable solutions.
- $\bar{t}_{gen}$: average time to generate problems.

1 Introduction

This project studies the Stable Roommates Problem, in which n people pair up to share n_{room} rooms in such a way that all of them are happy. Concretely, we are trying to corroborate Mertens Conjecture on the asymptotic decay in the probability of a stable matching existing as n increases [8].

The project has been implemented in `C++` [7] and `R` [10]. `C++` has been used to create a prototype which will generate random instances of the problem, as well as the algorithm that searches for the pairings. However, due to time constraints and bugs, we have also done these two parts in `R`, to be able to start sooner with the experimentation. Once these results are obtained, `R` is used to analyze the results and to test if the Mertens conjecture holds or not.

This document first presents a more in depth definition of the Stable Roommates Problem. Second, it presents each of the 3 different algorithms: problem generation, Irving's algorithm and the program to analyze the results. Third, the design of experiments and sample size is presented and immediately following are the results of said experiments. Finally, the study is concluded exposing what each member has learned as well as the results seen in the study.

2 Problem description

The Stable Roommates Problem is a type of stable matching problem where agents act selfishly to optimize their own satisfaction, subject to mutually conflicting constraints.

A better known variant of the stable matching problem is the stable marriage problem [5], first introduced by David Gale and Lloyd Shapley [4], where n men and n women compete with each other to get a partner. Each woman ranks all men according to her individual preferences, and each man does the same.

All agents will want to get a partner that is on the top of their preferences list, however mutual attraction is not guaranteed. Gale and Shapley [4] proved that each instance of the stable marriage problem has at least one stable solution.

If we imagine the stable marriage problem as a graph, it is easy to see the bipartite structure of the graph, where the matches are only allowed between groups. However, in the case of the stable roommates problem this graph transforms into a complete graph, where every agent can match to any of the other agents. This problem, also first introduced by Gale and Shapley in [4], may have no valid solution.

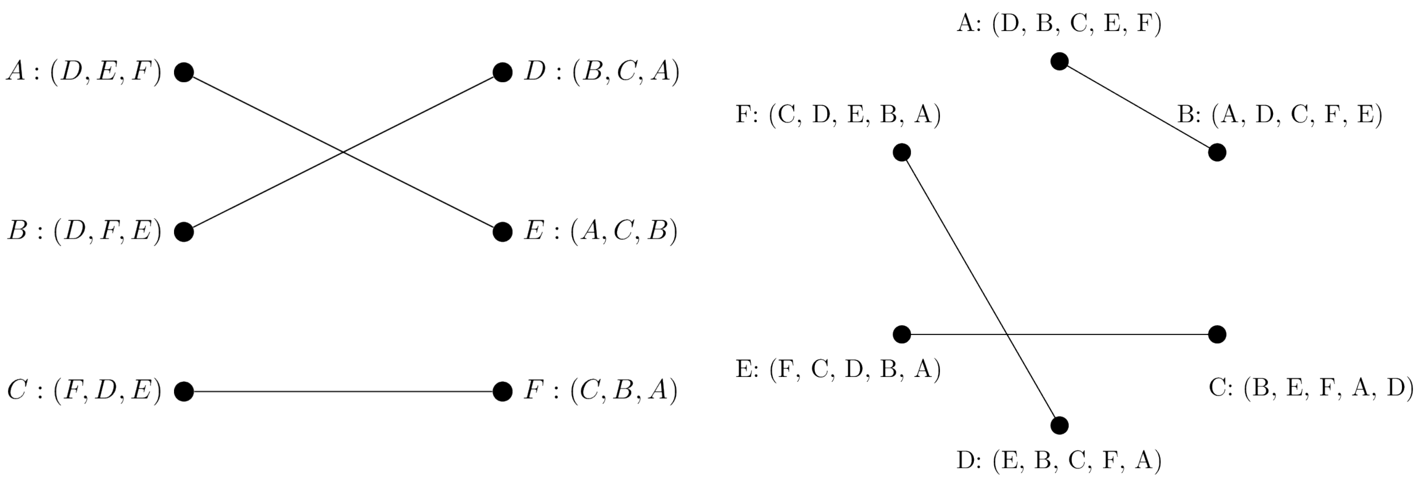

Figure 1: Example of stable matching, each vertex has a list of it's preferences in descending order. The edges represent pairings between 2 people. For the stable marriage problem (left), its possible to see the bipartite structure (each node only has preferences to the other part). For the stable roommates problem (right), the bipartite structure does not exist and it is a complete graph where each node can connect with any other. Image from the documentation of matchingR [10].

Before we delve deeper into the stable roommates problem, what is a stable match for the roommates? Consider $2n$ people that need to share n rooms, a *pairing* will be a set of n pairs where each of the $2n$ people appear only once in all of the pairs. A pairing is said to be *stable* if two agents x and y, do **not** meet the following conditions:

1. x and y do not form a match/pair
2. - x prefers y over his current match
   - y prefers x over his current match

To prove there is no solution we present a counterexample, consider the following table of 4 people for 2 rooms:

1	3	2	4
2	1	3	4
3	2	1	4
4	1	2	3

Table 1: Table representing the preferences of each person, going from most preferred (left) to least preferred (right).

There are $\frac{(2n_{room})!}{n_{room}!2^{n_{room}}}$ different pairings possible. In this example, $n_{room} = 2$, $n = 4$ people will have to share 2 rooms, the total amount of different pairings are $\frac{(2\cdot2)!}{2!2^2} = 3$. The 3 different pairings are:

Pairing	**Room 1**	**Room 2**
1	1 4	2 3
2	2 4	1 3
3	3 4	1 2

Table 2: All 3 possible different pairings, the rooms are identical so switching rooms will not change the outcome.

- In pairing **1**, individual 1 will be very unhappy and ask individual 2 and 3 to change rooms. Individual 2 will agree since he prefers individual 1 over 3 and will improve his happiness. This will create pairing **3**.
- In pairing **2**, individual 2 will be very unhappy and ask individual 1 and 3 to change rooms. Individual 3 will agree since he prefers individual 2 over 1 and will improve his happiness. This will create pairing **1**.
- In pairing **3**, individual 3 will be very unhappy and ask individual 1 and 2 to change rooms. Individual 1 will agree since he prefers individual 3 over 2 and will improve his happiness. This will create pairing **2**.

Each of these parings is not stable, since in each one there will be a pair x and y that do not form a pair and where they prefer each other over their current match. We could endlessly loop over the pairings, however there will never be a pairing that satisfies all parties. Thus proving that there are certain cases with no possible solution for the stable roommates problem.

This problem was thought to be NP-complete, however Robert Irving presented a polynomial time algorithm to solve it [6]. Irving's algorithm either returns a stable matching or `false` if none exist. More on this algorithm will be presented on section 3.2.

While this was a major breakthrough, there still exists an open problem on the probability P_n that an arbitrary roommates instance of size n is solvable, i.e. has a stable solution. However, Mertens conjectures the asymptotic behavior of P_n [8] such that:

$$P_n \approx e\frac{1}{\sqrt{\pi}}\left(\frac{4}{n}\right)^{1/4} \tag{1}$$

Through experimentation, in the following parts we will try and give more evidence in favor of Mertens conjecture for the behavior of P_n.

3 Algorithms

In this next section, a concise description and the computational cost for the algorithms is given. Concretely, the algorithm to generate random instances, to solve an instance of the stable roommates problem (Irving's algorithm) and to analyze the results. It is important to note that while we have implemented the generation of test cases and Irving's algorithm to solve the stable roommates problem in `C++`, our solution fails when the room sizes are large. So in order to be able to complete the study without spending more time debugging we have decided to use `R` [10]. More concretely the library `matchingR` [12] which has an implementation in `C++` of Irving's algorithm, and uses `R` as a wrapper.

3.1 Problem generation

As stated in section 2, a problem has n_{room} available rooms and $2n_{room} = n$ people that need a room. For each person there also exists a list of their preferences for whom to be paired with, this list will have size $n-1$ since the person does not include himself. So, since there are n people and each has a list of $n-1$, to express a random instance we need a $n \times (n-1)$ matrix that contains all these preferences.

In the `C++` implementation we present two different problem generation types. The first, is able to generate a list of n_{sets} sets, each with $n_{problems}$. The sizes of these problems will increase linearly from a given minimum to a given maximum, so n will increase starting from the minimum one by one until reaching the maximum. The second, orientated to testing and debugging, generates only one set with fixed size.

To generate a random instance we first create a $n \times (n-1)$ table that will be constant with the following form:

0	1	2	3
1	0	2	3
2	3	0	1
3	2	1	0

Table 3: Table representing the preferences of each person, going from most preferred (left) to least preferred (right), for a $n_{room} = 2$.

If we call $2 \cdot n_{room} = n$, the number of people, this has an asymptotic cost of $O(n^2)$. To generate $n_{problems}$ problems of the same size we can use the operator $=$ to copy the constant table with cost $O(n^2 n_{problems})$

However, if we only copy, all the problems will be the same, we need to do permutations on the table. The `C++` standard library `<algorithm>` contains a method called `random_shuffle` that given a two pointers, start and end, rearranges the elements between them with linear complexity [1]. To generate the permutation, we execute this procedure for each of the n rows of the matrix, and if we have $n_{problems}$ the cost of this will be $O(n^2 n_{problems})$.

Once the permutations are done, we change the type of each row to a list since it will later help us on the implementation of the algorithm. Changing from a vector to a list with constructors has linear complexity, giving us, as before $O(n^2 n_{problems})$ cost.

Adding these cost together gives us the complexity of creating a set of $n_{problems}$ problems which is $O(n^2 n_{problems})$. However, we can create n_{sets} per execution, meaning we repeat n_{sets} times the process, bringing the total cost to $O(n^2 \cdot n_{problems} \cdot n_{sets})$.

In the `R` implementation, the costs is similar however we change the notation to n_{sample} which will be the number of samples we take for problems of size n. Also, the way Irving's algorithm works on `R` [10] permits us to generate a random value between $[0, 1]$ and this will be translated directly to preferences, we also generate a $n \times n$ matrix as the implementation will just ignore the diagonal of the matrix. Such, we can use `runif` function [11] to generate the matrix of preferences. The computational cost to generate one problem will be then $O(n^2)$. However we generate m different problem sizes n, and for each of the problem sizes $n_i, i \in 1, \ldots, m$ we will generate $n_{samples}$ the overall complexity will be $O(n^2 \cdot n_{sample} \cdot m)$.

3.2 Main algorithm

In this section, the algorithm that given a random matrix with preferences will solve the "Stable Roommates Problem" is briefly explained. A more detailed explanation of the algorithm with proof and analysis of the cost can be found at [6]. We present some of the basic notions, but will not repeat the proofs nor present as much detail as that would be absurd.

Irving's algorithm [6] finds a stable solutions to instances of the stable roommates problem if one exists. Otherwise, it will return that there is no stable pairing.

The algorithm can be split into 2 different phases:

1. One-way proposals are made, and unpreferable pairs are forgotten. Concretely:
    (a) If x receives a proposal from y, then
        i. x rejects the proposal if he holds a better one already, (i.e. a proposal from someone higher than y in x's preference list)

ii. x accepts the proposal otherwise, rejecting any worse proposal x currently holds.

(b) An individual x proposes to the others in the order in which they appear in his preference list, stopping when the other party will consider him. If the other party later rejects x, then he will continue his sequence of proposals.

This phase can end in 2 different ways:

(a) Every person holds a proposal, we have reduced (more info at corollary 1.3 in [6]) the preferences list and we advance to phase 2.

(b) One person is rejected by everyone :'(. By corollary 1.2 in [6] this means no stable matching can exist. We return that no stable matching is possible and end the execution.

2. All-or-nothing cycles are located and removed from the preferences. Concretely, the cyclic sequence $a_1, \ldots, a_r$ of distinct persons is detected and removed such that

   (a) for $i = 1, \ldots, r-1$ the second person in a_i's current reduced preference list is the first person in a_{i+1}'s; we denote this person as b_{i+1}.

   (b) the second person in a_r's current reduced preference list is the first in a_1's, we denote this person as b_1.

   We call such a sequence $a_1, \ldots, a_r$ an all-or-nothing cycle. And the reduction involves forcing each b_i $i \in 1, \ldots, r$ to reject the proposal that he holds from a_i, thereby forcing each a_i to propose to $b_{i+1} mod(r)$, the second person in his current reduced list. As a result, all successors of a_i in b_{i+1} reduced list can be deleted, and b_{i+1} can be deleted from their lists. This phase can end in 2 different ways:

   (a) If in a reduced set of preference lists, every list only contains just one person, then the list specifies a stable matching (Lemma 4 in [6]). We return that there exists a stable matching and end the execution.

   (b) If one, or more, among the reduced set of preference lists is empty the instance does **not** admit a stable matching (Corollary 3.2 in [6]). We return that no stable matching is possible and end the execution.

The worst-case analysis of Irving's algorithm can be expressed largely in terms of "eliminations" from the preference lists. The total number of eliminations for any problem instance of size n cannot exceed n^2.

Phase 1: in the worst case scenario, i.e when a stable matching does not exist, one person (the one who is least preferred by all others in the set) will propose to all the members of the set. Hence, it might take $O(n)$ time at most. Finding the first unmatched item only costs as much n.

However, suppose that the function that seeks cycle is called k times, and that the i-th call yields tail of length t_i and a cycle of length r_i. Then the total number of operations involved in changing second choices, in a calls of the procedure, is $\mathrm{O}(n^2)$, since there are n elements in the array and none can be changed as many as n times. Hence, the total number in all calls of procedure of seek cycle function is:

$$O(n^2) + \sum_{i=1}^{k} \alpha_i + \beta(r_i + t_i - t_{i-1} + 1) + (\gamma \cdot r_i + \delta)$$

where α_i, β, γ, δ are constants and $t_0 = 0$. However, the i-th call results in at least $2r_i$ eliminations from the preference lists, and $r_i \geq$ for all i, so that:

$$k \leq 2\sum_{i=1}^{k} r_i \leq n^2$$

Therefore we can consider that the total number of operations is bounded by $\mathrm{O}(n^2)$.

Phase 2: Eliminating rotations might require traversing the entire list for all the members of the set. Hence, we might require $\mathrm{O}(n^2)$ time at most.

If the i-th call involves a cycle of length r_i, then the number of operations is bounded by a constant times r_i. Hence the total number of operations in all calls of the procedure is:

$$O\left(\sum_{i=1}^{k} r_i\right) = O(n^2). \tag{2}$$

So the resulting complexity is the sum of both parts which is $\mathrm{O}(n^2)$. So up until here, we the complexity of the initialization, plus the 2 phases of Irving's algorithm which all are $O(n^2)$.

More details and explanations on the previous steps can be found in section 4 of Irving's paper [6], this is just a brief summary.

3.3 Script that analyzes the test cases

The "algorithms" in `R`, that analyzes and returns the results consist of 3 different parts: reading the output from the main algorithm, analyzing the results and printing the results.

3.3.1 Input and Generation

First, we need to read the output the main algorithm returns us. Irving's algorithm will return the stable matching if it exists, and false if not. Since we are only interested if a solution exists or not, we change Irving's algorithm to return true instead of the stable matching.

So for a problem of size n, we generate n_{sample} different instances. Reading the results of these n_{sample} experiments, is linear time $O(n_{sample})$. Additionally, we do m experiments with different n, so the total will be $O(mn_{sample,k})$. Where $n_{sample,k}$ is the number of different experiments done with problems of size $k \in 1, \dots, m$.

So reading the output will be linear in the size of the output, since we cannot do better than reading one line at a time.

However, our implementation of the Irving's algorithm will output only a summary of the values. The real output will actually be a $m \times 7$ table which will have the columns n, $n_{sample,k}$, n_{stable}, $n_{unstable}$, t_{gen} t_{stable} and $t_{unstable}$. Where these last three are the time in seconds to generate the problem, and to solve it depending if the problem is stable or unstable. So instead of reading a file with $mn_{sample,k}$ different results, we will read only a table of size $7m$. Such, the complexity will be $O(m)$

3.3.2 Statistics

Second, we calculate the necessary statistics to analyse the output.

The first statistic we calculate is Pearson's chi-squared [2] test for the goodness of fit. Where the value of the test-statistic is

$$\chi^2 = \sum_{i=1}^{m} \frac{(O_i - E_i)^2}{E_i}$$

Where the O_i is the number of observations of type i. $E_i = np_i$ the expected (theoretical) count of type i, asserted by the null hypothesis that the fraction of type i in the population is P_i. And m is the different problem sizes we have experimented with.

So this statistic will be linear in the number of different problem sizes m. We need to calculate the sum of the square difference between the observed and expected divided by the expected. All these are constant time operations that we calculate m times, so the complexity will be $O(m)$. To calculate the p-value, we need to check this value against the χ^2 distribution of $m-1$ degrees of freedom, which can be done in constant time $O(1)$.

The second statistical method we will use is a generalized linear model. Concretely we are going to use a Binomial Regression Model with a logistic link. Not much information is found about the `R`'s function `glm` time complexity. However, the generalized linear model is similar to a linear regression with the application of a link function. The logistic link function is $\pi_i = \log(p_i/(1-p_i))$ and the normal linear model is basically a matrix multiplication $X^T X$, which in the naive calculation would be $O(n^3)$.

Other statistics are calculated such as the estimation of the proportion, variance, error and confidence intervals. All these operations are constant time, however we will have to do them for each value $k \in 1, \ldots, m$. So overall will be $O(m)$.

$$\hat{p} = \frac{frequency_{stable}}{n_{sample}} \qquad \hat{\sigma}^2 = \frac{\hat{p}(1-\hat{p})}{n_{sample}} \qquad err = |P_n - \hat{p}| \qquad CI = \hat{p} \pm z_{\alpha/2} \cdot \hat{\sigma}$$

3.3.3 Results

Finally, we need to present the results. Returning previously calculated values is a call to memory so it is constant time $O(1)$.

For the plots, we have 2 different complexities. Drawing the plot, which has $O(p_w \cdot p_h)$ where p_w is the number of pixels of width, and p_h are the number of pixels of height. The other complexity, comes from computing slopes (lines). As an example we consider the slope for P_n. To draw the slope, we have to evaluate the equation P_n for various values of n, and connect the points in such a way that seems a line. So to calculate slopes, we will need to generate many small t steps in the x-axis to evaluate P_n at each increment. Such an operation can be done in linear time $O(t)$.

4 Design of Experiments

The purpose of this project is to corroborate or dismiss Mertens conjecture of the probability P_n that an arbitrary roommates instance of size n has a stable matching. To do so, we have selected 11 instances for different amounts of people n.

The execution of the program has been done in a contained space such that no external factors can affect it. We have used Vultr [13] to provide us with multiple servers big enough to execute problems up to $n = 20000$. The specifications of the machines are: Debian 10 with Intel(R) Xeon (Cascadelake) CPU @ 2.92GHz with 16 cores and 64GB of RAM. We have used multiple of these machines to paralellize the execution of experiments, which took almost 9 days of continuous execution.

The experiment can be summarized in the following table:

Observation	It could be that P_n of the stable roommates problem has an asymptotic component.
Approach	We generate different experiments with a different range of n, we compare the observed frequencies with the expected frequencies if they were to follow equation (1).
Hypothesis	$H_0 : P_n = \hat{p}$ $H_1 : P_n \neq \hat{p}$
Method	• We select m different problem sizes n. • For each n we generate $n_{sample,k}$ problems. • For each $n_{sample,k}$ we use Irving's algorithm to solve it. • Measure the frequency of stable matchings for each experiment. • Compare the observed frequencies with the expected frequencies. • Fit to a binomial regression model trying to predict if $n^{-1/4}\beta$ can predict the probability of stable matching for problems of size n. • Evaluate if the difference and fit are significant with a Pearson's χ^2 test.

Table 4: Characteristics of the experiment.

4.1 Sampling

There are $\frac{(2n_{room})!}{n_{room}!2^{n_{room}}}$ possible pairings with rooms of size n_{room}. This number grows extremely fast, just with $n_{room} = 10$ we have 654729075 different possible pairings. It seems obvious that if we want to test with large rooms we cannot possibly enumerate all permutations and test if they are stable or not. We need to pick samples for each different problem size to estimate the proportion. Since the number of possibilities is so big, we can consider this an infinite population and we can sample with replacement.

Sample size is a statistical concept that involves determining the number of observations or replicates (the repetition of an experimental condition used to estimate variability of a phenomenon) that should be included in a statistical sample [9]. It is an important aspect of any empirical study requiring that inferences be made about a population based on a sample.

The difference between the sample statistic and population parameter is considered the sampling error. For example, if we measure the proportion from a thousand problems from an infinite population of all problems, the estimated proportion of the thousand is typically not the same as the true proportion for the population.

The goal of this study is to estimate the population proportion p so that the error is no larger than 0.05. That is, the goal is to calculate a 95% confidence interval such that:

$$\hat{p} \pm \epsilon = \hat{p} \pm 0.05$$

We know the formula for a $(1 - \alpha)100\%$ confidence interval for a population proportion is

$$\hat{p} \pm z_{\alpha/2}\sqrt{\frac{\hat{p}(1-\hat{p})}{n}}$$

We equate the terms appearing after each of the above $\pm$ signs and solve for n in the two previous equations.

$$\epsilon = z_{\alpha/2}\sqrt{\frac{\hat{p}(1-\hat{p})}{n}}$$

Such we get that for a confidence of 95% and error no larger than 0.05 the number of samples we need for each room size is:

$$n = ceiling\left(\frac{z_{\alpha/2} \cdot \hat{p}(1-\hat{p})}{\epsilon^2}\right) = ceiling\left(\frac{1.96^2 \cdot \hat{p}(1-\hat{p})}{0.05^2}\right)$$

The population proportion for each problem size will be estimated as $\hat{p} = e\frac{1}{\sqrt{\pi}}\left(\frac{4}{n}\right)^{1/4}$ from Mertens conjecture. The following table gives us the sample sizes for the different number of problem sizes sampled to give an error of 0.05 at a confidence interval of 95%. After 10000 we increase the sampling error to 0.1, since the execution time takes way to long. For 20000, we also increase the sampling error for the same reasons to 0.15. This is not ideal, however with limited time and resources there was no other alternative.

n	50	100	250	500	1000	2000	5000	7500	10000	15000	20000
P_n	0.816	0.686	0.545	0.459	0.386	0.324	0.258	0.233	0.217	0.196	0.182
n_{sample}	232	332	381	382	365	337	295	69	66	61	26
ϵ	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.10	0.10	0.10	0.15
Expected	43.65	189.23	227.71	175.21	140.78	109.30	76.09	16.08	14.31	11.95	4.74

Table 5: For each problem size, the probability P_n from Mertens conjecture, the sample size for each problem size to estimate the proportion with 95% confidence and maximum error of ϵ, and the expected stable solutions to be found.

5 Results

We present the results obtained with the actual sample sizes.

n	50	100	250	500	1000	2000	5000	7500	10000	15000	20000
n_{sample}	232	332	381	382	365	337	295	69	66	61	26
n_{stable}	159	209	202	184	122	110	73	16	13	13	4
$n_{unstable}$	73	123	179	198	243	227	222	53	53	48	22
$\hat{p}$	0.685	0.630	0.530	0.482	0.334	0.326	0.247	0.232	0.197	0.213	0.154
$\hat{\sigma}$	0.030	0.027	0.026	0.026	0.025	0.026	0.025	0.051	0.049	0.052	0.071
$\hat{p} - z_{\alpha/2}\hat{\sigma}$	0.626	0.578	0.480	0.432	0.286	0.276	0.198	0.132	0.101	0.110	0.015
$\hat{p} + z_{\alpha/2}\hat{\sigma}$	0.745	0.681	0.580	0.532	0.383	0.376	0.297	0.331	0.293	0.316	0.293
$\lvert P_n - \hat{p}\rvert$	0.13	0.056	0.015	0.023	0.051	0.002	0.01	0.001	0.020	0.017	0.029

Table 6: Summary of the results with the actual samples done.

We can see how the error with respect to Mertens conjecture is quite small after surpassing $n \geq 100$. Mertens gave an asymptotic approximation for P_n, and with $n^{-1/4}$ being quite a slow decay we expected the results to not be so close at $n \leq 100$. Additionally, Mertens conjecture returns a value probabilities of over 1 for values $n < 23$, which for a probability does not make sense. This is why we decided to start from $n = 50$ and try to experiment with values of n as high as possible. Next, we go in depth to compare our results with Mertens Conjecture.

5.1 Probability

Computing the Pearson's χ^2 goodness of fit test for the observed compared to the expected we get a p-value of 0.447, which is superior to the confidence level selected. Such, we can **not** reject the null hypothesis in favor of the alternative as the data does not show enough of a difference between the observed and expected to show a statistical difference. We can conclude that for our experiments, the probability of finding a stable matching in a problem of size n follow Mertens conjecture for P_n as shown in equation 1.

$\chi^2 =$	9.931	**p-value =**	0.447

Table 7: Summary of the Pearson's χ^2 goodness of fit test compared to the χ^2 with 10 degrees of freedom.

Following the Pearson's goodness of fit test, we will realize a binomial regression with a logit link. We will not go into much detail explaining the foundations of such a model as they would be a repeat of section 3.1 from [2]. However, suppose the response variable Y_i is binomially distributed $B(m_i, p_i)$ with $q = 1$ predictors and the following logit link:

$$\eta_i = g(\pi_i) = logit(\pi_i) = \log\left(\frac{\pi_i}{1-\pi_i}\right)$$

We will suppose that the logit is a linear function of the predictor for the underlying probability π_i. Such $g(\pi_i) = \mathbf{X_i}'\beta$. Where $\mathbf{X_i}$ is a vector of features and β is a vector of regression coefficients. The result of the fit on R gives us:

```
glm(formula = cbind(stable, samples_real - stable) ~ I(n^(-1/4)),
    family = binomial, data = df)
Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.5384  -0.8027  -0.3285   0.2163   2.1115
Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -1.9464     0.1208  -16.11   <2e-16 ***
I(n^(-1/4))   7.8326     0.5261   14.89   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
(Dispersion parameter for binomial family taken to be 1)
Null deviance: 255.815  on 10  degrees of freedom
Residual deviance:  11.403  on  9  degrees of freedom
AIC: 74.193
Number of Fisher Scoring iterations: 3
```

Provided that response is truly binomial and that the m_i are relatively large, the residual deviance is approximately χ^2 distributed with $n - q - 1 = 9$ degrees of freedom if the model is correct. Thus we can use the deviance to test whether the model is an adequate fit. For the logit model of the experimental data, we compute:

$\chi^2 =$	11.402	**p-value =**	0.249

Table 8: Summary of the Pearson's χ^2 goodness of fit test for the binomial regression to the χ^2 with 9 degrees of freedom.

Since the p-value is in excess of 0.05, we conclude that this model fits sufficiently well the data.

After we do the regression we can estimate probabilities as such:

$$\pi_i(\eta) = g(\pi)^{-1} = \frac{e^{\eta_i}}{1 + e^{\eta_i}}$$

Which is more interesting since we can use the coefficients given from the binomial regression to approximate the probability. For example, for $n = 10^4$, $logit(y) = -1.946443 + 7.832617 \cdot n^{-1/4}$, $logit(y) = -1.163181$, which reversing the logit we get the estimated probability of a random instance having a stable matching $e^{-1.163181})/(1 + e^{-1.163181}) = 0.2380$. Which is pretty close to the Mertens conjecture $P_n = 0.2168$, only an error of 0.0212 for the difference.

The next graphic represents the Mertens P_n value compared to our empiric testing results. The slope for the experimental data has been obtained interpolating for each n the value from the binomial regression. We can see that they are pretty similar.

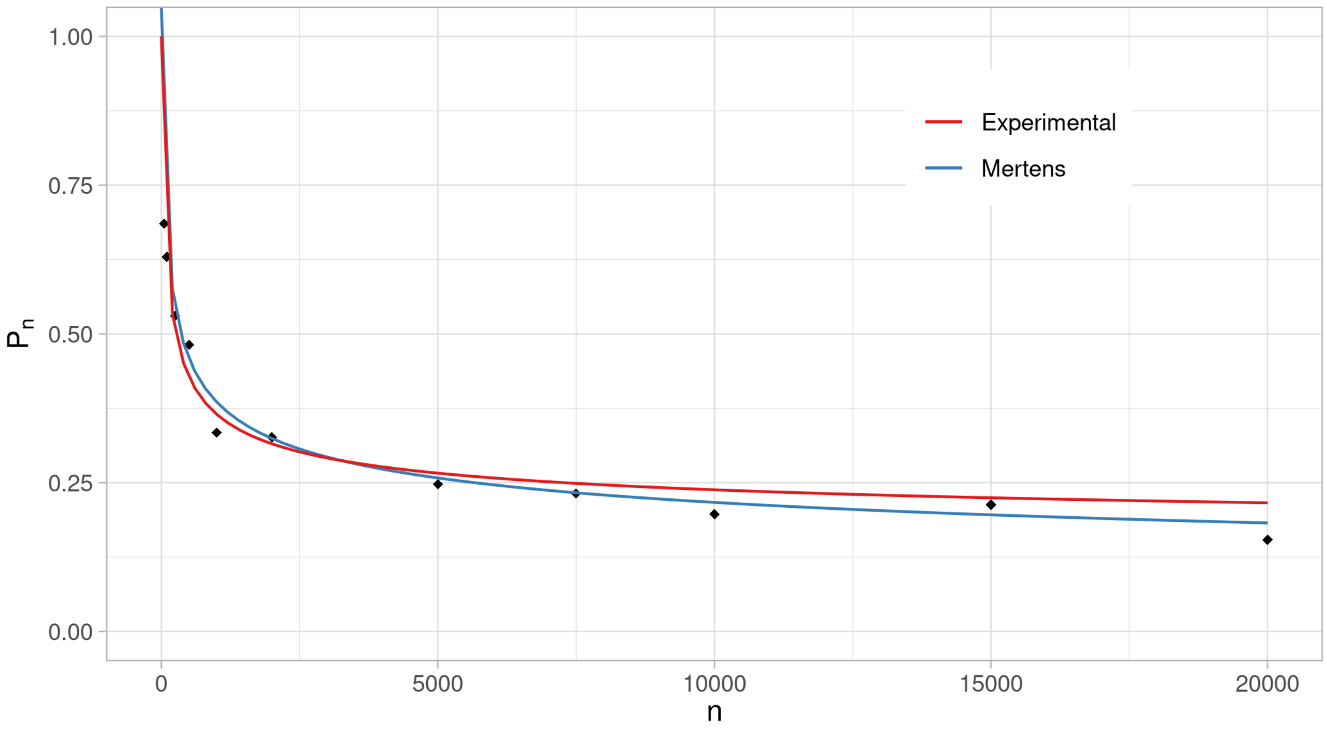

Figure 2: Probability P_n of admitting a stable matching in problems of size n. Each data point represents the average from the $n_{samples,k}$ done at that n.

This concludes the main part of the study, we present one last observation to show the quadratic nature of Irving's algorithm.

5.2 Time

We wish to study if Irving's algorithm is truly quadratic or not.

n_{room}	50	100	250	500	1000	2000	5000	7500	10000	15000	20000
n_{sample}	232	332	381	382	365	337	295	69	66	61	26
$\bar{t}_{stable}(s)$	0.005	0.014	0.166	0.777	4.444	24.373	475.339	2168.735	3832.616	12476.105	30086.440
$\bar{t}_{unstable}(s)$	0.005	0.014	0.166	0.777	4.437	24.404	477.269	1999.127	3836.895	12734.336	29369.467
$\bar{t}_{gen}(s)$	0	0	0	0	0	0	0.002	0.010	0.058	0.596	2.944
$t_{total}(h)$	0.000	0.001	0.018	0.082	0.450	2.284	39.071	39.071	70.329	214.854	212.931

Table 9: Summary of the times taken to experiment.

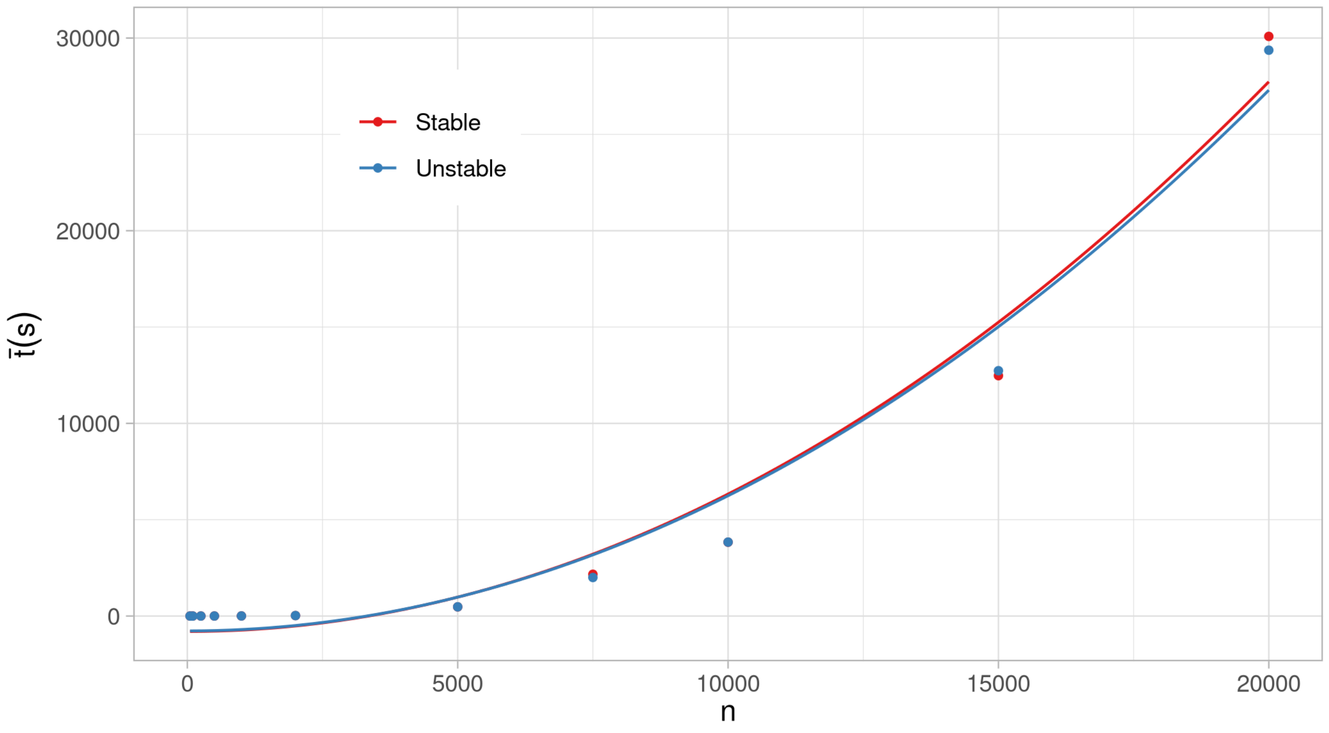

Figure 3: Average time, in seconds, for Irving's algorithm to finish grouped by if the problem is stable or not. The curve is a quadratic adjustment.

With this plot, it is pretty easy to see the asymptotic worst case time on Irving's algorithm is quadratic. We can see that the curve, independently if we have the problem presents a stable solution or not could be considered the same. To compare if there are differences between the time the algorithm takes depending if there exists a stable matching or not we can do the following linear model:

$$time(s) = \beta_0 + \beta_1 \cdot n + \beta_2 \cdot a + \beta_3 \cdot an$$

Where a is a dummy variable that represents if there was a solution or not. Contrasting this model, we will be able to see if there are significant differences between the time needed to find a solution if its stable or not. The output from the `R` code returns:

```
lm(formula = time ~ I(n^2) * type, data = dfPlot)
Residuals:
    Min      1Q  Median      3Q     Max
-2772.2  -907.6   721.3   782.1  2354.2
Coefficients:
                      Estimate Std. Error t value Pr(>|t|)
(Intercept)         -8.023e+02  5.521e+02  -1.453    0.163
I(n^2)               7.134e-05  3.865e-06  18.458 3.83e-13 ***
typeUnstable         2.946e+01  7.807e+02   0.038    0.970
I(n^2):typeUnstable -1.166e-06  5.466e-06  -0.213    0.833
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Residual standard error: 1568 on 18 degrees of freedom
Multiple R-squared:  0.9738,Adjusted R-squared:  0.9695
F-statistic: 223.4 on 3 and 18 DF,  p-value: 1.991e-14
```

In which we can clearly see that the overall model is significant, the p-value of the F-test is below the level of confidence. However, the interactions `typeUnstable` and `I(`n^2`):typeUnstable` have p-values of over 0.05, so we cannot consider them to be coming from different models. While the intercept is not significant, we cannot remove them since we would lose the ability to interpret the model [3].

In conclusion, even if the problem has a stable solution or not the time needed is modeled as:

$$time(s) = -802 + 0.000071336 \cdot n^2$$

One final observation, is that if we look at the average time to generate the problems, we can see it is almost negligible compared to the average time taken to find solutions. And while both were quadratic, the constants for Irving's algorithm are much larger making Irving's algorithm much slower than the generation.

6 Conclusions

In this project we have studied the conjecture given by Mertens about the probability that a random problem of size n has a stable matching [8]. After studying Mertens original article and studying more about the Stable Roommates Problem, we found Irving's algorithm that presented a way to solve this problem in $O(n^2)$ time.

We have implemented the algorithm in `C++`, however for large problem sizes we seemed to run into problems. To avoid debugging time, and since we are pressed on time, we decided to use the library `matchingR` [10] which contained wrappers in `R` for functions implemented in `C++`. After executing some experiments locally on our machines to make sure it works, we decided to rent a server on Vultr [13] which would run the experiments in controlled environment.

After less experiments than we would have hoped, we cannot refute the null hypothesis which was that the data follows Mertens conjecture. We tried two different methods, the first a simple Pearson's χ^2 goodness of fit test and a binomial regression model. Both gave us the same conclusion that the data fits the conjecture so for now, Mertens conjecture still holds.

We also analyzed the execution time of Irving's algorithm and compared if there were any differences in the time it took for stable problems versus unstable. After fitting a quadratic slope on the model, we found there was no difference between the two. We also found that quadratic model explains very well the time it takes to find a solution, which was an expected theoretical result. Regarding time, we also found a big disparity between the time to generate problems and Irving's algorithm. And while both are quadratic, their constants are not the same. Making us learn that sometimes asymptotic notation is not enough and to see the disparities we need to go look for the exact details.

7 Personal evaluation

In this section we expose what each of the members of the group has learned while doing this project.

All three of us were surprised by the amount of time it took to find solutions when n was large. We knew that $O(n^2)$ is supposedly slow, however we never experienced it like we have when trying to run the experiments for this project.

There was also an unexpected factor that had us question our sanity. And it had to do with this small sentence we overlooked from Irving's paper [6]: "...from a practical point of view, the memory requirements for the arrays, rather than the processing time of the procedure, are likely to be the limiting factor on the size of roommates instances to which the algorithm can be successfully applied.”

We scoffed at the sight of this and did not think about it anymore. However, 35 years after the published paper, we ran into this exact problem. When executing it with `R`, the execution seemed to randomly halt, saying it had no memory to store a vector of 3GB. After further analysis, we saw that the 32GB of RAM on the local PC we were using was not enough. Since the amount of storage needed for the algorithm overflowed the RAM. This is quite a lesson learned as well, since we usually do not take into consideration the cost in space of our algorithms. If we did take it into account, maybe we wouldn't have had to do trial and error with 4 different servers each with different sizes of RAM until one did not return an error.

7.1 César

This is something that teachers have repeated a lot during the career, but this project has confirmed that the efficiency of an algorithm it's almost more important than the algorithm itself, because a good implementation of an algorithm allows to execute programs with a lot of data, in addition to do more experiments in less time.

7.2 Benjamí

Apart from the obvious, learning about the problem and Irving's algorithm, I have also learned a bit more about statistics and sample sizes. Which i did not know beforehand. It seemed a useful knowledge to gain, since in a Computer Scientist career there will be many experiments to do, and it is impossible to enumerate them all. So learning how to select a good sample size to study a factor is definitely very important for me. I also learned to take into account the space costs for future experiments to not waste time doing trial and error with different servers.

7.3 Ioan

Diving again into `C++` programming as I was implementing the algorithm has been an interesting experience. I have also learned that a naive implementation of any kind of function is not only the best to do it and might be the reason for a bad complexity time. Furthermore I could investigate more on data structures and how being easier to implement does not mean easier to use. There is also the part of statistics that thanks to my colleagues I was able to understand better. In conclusion it was a pretty engaging project at it pushed us to work as a team, and try and find the best solution.

References

[1] *Cplusplus Random shuffle*. https://www.cplusplus.com/reference/algorithm/random_shuffle/. Accessed: 2021-03-25.

[2] Julian J Faraway. *Extending the linear model with R: generalized linear, mixed effects and nonparametric regression models*. CRC press, 2016.

[3] Julian J Faraway. *Linear models with R*. CRC press, 2014.

[4] David Gale and Lloyd S Shapley. "College admissions and the stability of marriage". In: *The American Mathematical Monthly* 69.1 (1962), pp. 9–15.

[5] Dan Gusfield and Robert W Irving. *The stable marriage problem: structure and algorithms*. MIT press, 1989.

[6] Robert W Irving. "An efficient algorithm for the "stable roommates" problem". In: *Journal of Algorithms* 6.4 (1985), pp. 577–595.

[7] ISO. *ISO/IEC 14882:1998: Programming languages — C++*. Available in electronic form for online purchase at http://webstore.ansi.org/ and http://www.cssinfo.com/. Sept. 1998, p. 732. URL: http://webstore.ansi.org/ansidocstore/product.asp?sku=ISO%2FIEC+14882%2D1998;%20http://webstore.ansi.org/ansidocstore/product.asp?sku=ISO%2FIEC+14882%3A1998;%20http://www.iso.ch/cate/d25845.html;%20https://webstore.ansi.org/.

[8] Stephan Mertens. "Random stable matchings". In: *Journal of Statistical Mechanics: Theory and Experiment* 2005.10 (2005), P10008.

[9] *Penn State Introduction to STAT 415*. https://online.stat.psu.edu/stat415/lesson/introduction-stat-415. Accessed: 2021-04-10.

[10] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2021. URL: https://www.R-project.org/.

[11] *Rdocumentation runif*. https://www.rdocumentation.org/packages/compositions/versions/2.0-1/topics/runif. Accessed: 2021-03-26.

[12] Jan Tilly, Nick Janetos, and Maintainer Jan Tilly. *Package 'matchingR'*. Tech. rep. Working Paper, 2018.

[13] *Vultr The Infrastructure Cloud*. vultr.com. Accessed: 2021-03-27.