Proyecto Algorísmia

Stable-roommates

Grado A

Documentación

Curso 2020/2021 2Q

Marc Belmonte
Jesús Benítez
Alex Urraco

UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Índice:

Introducción: 3

Elementos del problema: 3
- Generación de entradas aleatorias 3
- Pseudocódigo entradas aleatorias: 4
- Primeras aproximaciones 4
- Algoritmo 6
- Pseudocódigo (algoritmo de Irving): 7
- Justificación de la eficiencia del algoritmo propuesto: 8
- Problemas del algoritmo: 8

Desarrollo del proyecto: 8
- Método: 8

Conclusión: 16

Bibliografía: 16

Introducción:

El problema de los compañeros de habitación surge para resolver la asignación de pares de estudiantes en cuartos de una universidad, donde cada estudiante tiene una lista de preferencias estrictamente ordenada de los compañeros restantes. Se trata de una generalización del problema de los matrimonios estables, en el cual una instancia consiste en un conjunto de 2n personas de cardinalidad par, cada una incluyendo en su lista de preferencias a las 2n-1 personas restantes en orden estricto.

Un emparejamiento entre los miembros de este conjunto es una división del mismo en pares disjuntos. Consideraremos que un emparejamiento es estable si no se da el caso de tener dos personas x e y tales que x e y no forman pareja y x prefiere a y en vez de a su pareja e y prefiere a x en vez a su pareja. Esta es la noción de estabilidad que buscamos en nuestro problema.

Nuestro objetivo en este proyecto será llevar a cabo un experimento similar al de mertens. Dada una entrada aleatoria con 2n personas, buscaremos la probabilidad de que exista un emparejamiento estable. El número de emparejamientos posibles dada una n es: $\frac{(2n)!}{n!2^{2}}$.

Esta probabilidad(Pn) fue estudiada por Mertens, el cual da evidencias y argumenta en su artículo del 2005, que esta probabilidad tiende a cero para valores de n muy grandes.

Esta Pn es $\approx e\frac{1}{\sqrt{\pi}}(\frac{2}{n})^{\frac{1}{4}}$

Elementos del problema:

Generación de entradas aleatorias

Para generar las entradas aleatorias del problema hemos implementado una función que realiza permutaciones aleatorias para cada persona. Nuestro código itera sobre cada una de las personas, y por cada una de ellas llama a la función que crea la permutación. Esto lo hemos hecho utilizando la librería random de C++ que nos proporciona las funciones de rand() y srand(time(0)). La función srand(), crea una seed para generar los números aleatorios (si se utiliza una misma seed para cada ejecución los números random serían los mismos para cada una de las ejecuciones del código), pasando el parámetro time(0), nos aseguramos que esta seed sea distinta a la

anterior ejecución y la función rand() nos proporciona un valor aleatorio que pertenece a la seed en la que estamos.

Sea *i* la persona para la cual se va a crear la permutación, empezaremos creando una lista auxiliar que llamaremos *aux*, con todos las personas excepto *i.* Ahora generamos un número random x (0 <= x <= *aux*.size()), seleccionamos el elemento *aux*[x] y lo añadimos a una lista que llamaremos *def*. Este elemento *aux*[x], será borrado seguidamente de *aux*. Repetiremos este proceso hasta que en aux no queden elementos y entonces se devuelve la lista *def* con la permutación de *i*.

Pseudocódigo entradas aleatorias:

```
permutacion(int i):
        aux  // contiene todos los elementos menos i
        for j in range(0,size):
                index = rand(aux) //indice random de aux
                def.append(aux[index])
                aux.remove(index)
        return def

buildInput():
        for i in range(0,size):
                v = permutacion(i)
                preferencias.append(v) // lista con las diferentes permutaciones
```

Primeras aproximaciones

Antes de empezar con la explicación del algoritmo, debemos tener en cuenta algunas afirmaciones demostrables con ejemplos.
En primer lugar, podemos decir que el problema puede no tener solución. Gale and Shapley, que estudiaron el problema de stable-marriage, ya demostraron que el problema de stable-roommates no siempre tiene solución.

Persona	Lista de preferencia		
1	2	3	4
2	3	1	4
3	1	2	4
4	X	X	X

Es evidente que esta instancia del problema no tiene solución, puesto que cualquier emparejamiento con la persona 4, daría lugar a un emparejamiento inestable. Si emparejamos el 1 con el 4 y 2 con 3, como el 1 y el tres se prefieren antes que a las parejas que se les han asignado, romperían su actual pareja. Así que ahora nos quedan las parejas 2 - 4 y 1 - 3, pero estas parejas también se romperían puesto que el 1 y el 2 se prefieren antes que a sus actuales parejas y así sucesivamente.
Por otra parte, también podemos afirmar que los emparejamientos estables no son únicos dada una instancia del problema. Esto nos lo demuestra Knuth con un ejemplo algo más extenso.

Persona	Lista de preferencia						
1	2	5	4	6	7	8	3
2	3	6	1	7	8	5	4
3	4	7	2	8	5	6	1
4	1	8	3	5	6	7	2
5	6	1	8	2	3	4	7
6	7	2	5	3	4	1	8
7	8	3	6	4	1	2	5
8	5	4	7	1	2	3	6

En esta entrada del problema podemos encontrar los siguientes emparejamientos estables { l/2, 3/4, 5/8, 6/7}, { l/4, 2/3, 5/6, 7/8} y { l/5, 2/6, 3/7, 4/8}. El razonamiento en este caso es algo más complejo, así que no vamos a incidir más en este aspecto, pero sí es importante tener en cuenta este aspecto al realizar la experimentación

Algoritmo

Para la implementación de nuestro código nos hemos basado en el algoritmo de Irving. Robert Irving mostró que es posible determinar que este tipo de problema tiene solución y propuso en 1985 un algoritmo eficiente estructurado en dos fases para encontrar un emparejamiento estable en el caso de que sea resoluble o responder negativamente cuando es irresoluble:

- Primera fase: Consiste en una secuencia de proposiciones de acuerdo a las siguientes reglas:
    - Si x recibe una propuesta de y, entonces:
        - ▪ Lo acepta si no tenía ninguna propuesta anterior.
        - ▪ Lo rechaza si ya tiene una propuesta mejor, es decir, si tiene una propuesta vigente de alguien mejor ubicado en su lista.
        - ▪ Lo reemplaza si es mejor que su propuesta vigente, rechazando la anterior y aceptando al nuevo pretendiente.

    - Un individuo propone siguiendo el orden de su lista de preferencias, deteniéndose cuando su propuesta es aceptada, y continuando si es rechazado en el futuro.

  Esta etapa finaliza en cualquiera de las siguientes situaciones:

    - Una o más personas fueron rechazadas por todas las demás. En este caso el problema es irresoluble.

    - Todas las personas tienen una proposición. En este caso se generan listas reducidas de preferencias teniendo en cuenta que las personas con menor prioridad que la propuesta vigente no pueden ser pareja en un emparejamiento estable.

  Así finaliza esta primera fase.

- Segunda fase: Trata los casos en los que existen listas reducidas con más de un elemento y se continúa su reducción mediante las siguientes reglas:

    - Buscar la primera persona X con más de un elemento en su lista reducida. o Buscar la segunda persona Y de la lista reducida de X.

    - Buscar la última persona de la lista reducida de Y.

- Repetir los dos pasos anteriores hasta que nos aparezca una X e Y que ya habían aparecido anteriormente. El conjunto de valores comprendidos entre la pareja X, Y repetida, los eliminaremos de las respectivas listas reducidas.

Esta etapa finaliza en cualquiera de las siguientes situaciones:

- Una o más personas se quedan sin proposiciones. En este caso el problema es irresoluble.

- Todas las personas tienen una única proposición que es la solución al problema

Pseudocódigo (algoritmo de Irving):

```
while there are unmatched people do
    let ai be first unmatched person
    ai proposes to their favourite roommate aj who has not rejected him
    if aj had received no proposal then
        aj accepts ai
    else if aj prefers ai over his current match ak then
        aj accepts ai
        reject symmetrically(ak,aj)
    else
        reject symmetrically(aj,ai)
    end

end

for all ai holding proposal from ai do
    reject symmetrically al(aj, ak) where aj prefers ai over ak
end

for all cycles in(p1,...pn) and (q1,...,qn) such that:
q is second preference of pi and pi+1 i slast preference of qi do:
    for i = 1..n-1 do
        reject symmetrically (qi, pi+1)
    end
end
```

Justificación de la eficiencia del algoritmo propuesto:

En el enunciado se comenta que existen algoritmos de coste cuadrático que consiguen encontrar un emparejamiento estable. El algoritmo que proponemos lo resuelve en coste cuadrático aunque la implementación que hemos hecho no es en coste cuadrático puesto que las estructuras de datos que usamos no son las óptimas. Nuestra implementación tiene un coste de:

- n^2*log n para la primera fase

y coste de:

- O (n^2 *(k + log n)) para la segunda fase,

donde k es la máxima extensión que puede tener un ciclo en la fase tres. En la descripción del algoritmo hemos comentado que se divide en dos fases, pero a nivel de código lo hemos dividido en tres bloques donde el primero y segundo pertenecen a la primera fase descrita por Irving.

Problemas del algoritmo:

Hemos implementado el algoritmo tal y como lo describe Irving, aun así, los resultados no son nada parecidos a los que llegó Irving con su algoritmo, los cuales se pueden consultar en un artículo suyo [4]. Para solucionar este problema, añadimos una extensión a la fase dos descrita por Irving que nos aportaba unos resultados parecidos a los que obtiene Irving pero que aún distan de la conjetura planteada por Martens.
La extensión que hemos añadido lo que hace es consultar si algunas de las listas de preferencias es de tamaño 1, si es así establecemos esa pareja como definitiva y por lo tanto eliminamos de las listas de la pareja, las otras personas respetando siempre el Lema 1 de Irving de su artículo [4].

Desarrollo del proyecto:

Método:

Con el fin de resolver el problema, hemos estructurado nuestro trabajo en varias fases.

Fase 1:
Esta fase fue básicamente una lluvia de ideas, tratando de organizar el trabajo y encontrar una base sobre la cual trabajar. De aquí sacamos en claro la necesidad de usar un lenguaje de programación que nos diese la facilidad de realizar experimentos y simulaciones. Teníamos las opciones de python o C++. En un primer momento,

elegimos C++ como el lenguaje a utilizar, así que empezamos a buscar información sobre el algoritmo a utilizar.

Fase 2:
Como ya hemos introducido antes, en esta fase buscamos la base de nuestro código, un algoritmo capaz de resolver el problema de emparejamientos estables. Debido a que es un problema bastante conocido, no encontramos muchas dificultades a la hora de encontrar un algoritmo que cumpliese nuestros requisitos. Una vez elegido, nos dispusimos a implementarlo paso a paso. Cada vez que implementábamos un paso, debuggabamos y hacíamos distintas pruebas para asegurarnos que se comportaba de forma adecuada, esto lo repetimos para cada uno de los pasos en los que dividimos el algoritmo.
Como era de esperar, surgieron imprevistos y problemas a la hora de programar. Tuvimos que solucionar una serie de bugs que tenía el algoritmo de Irving, y que en un principio nos costó mucho arreglar. Fue en ese momento cuando nos planteamos la posibilidad de implementar el mismo código, pero ahora en python.
Una vez tomada la decisión, distribuimos el trabajo para intentar solucionar por un lado el código en C++, y por otro tratar de conseguir que también funcionase en python.
Finalmente conseguimos que nuestra primera opción funcionase y pasamos a la siguiente fase, la experimentación.

Fase 3:
Una vez hechas todas las comprobaciones y testear nuestra implementación, pasamos al último paso. Nos toca centrarnos en la abstracción de los datos y el análisis de estos. Con el primer código nos dimos cuenta que la implementación del algoritmo de Irving no es trivial. Así que con los resultados que obtenemos vamos comparando con los resultados que se obtienen en otras implementaciones del algoritmo encontrados por internet para comprobar que los resultados que obtenemos son los correctos.

Contratiempos:

A la hora de realizar los experimentos hemos tenido una serie de problemas respecto a los resultados esperados. Nuestras probabilidades sobre si un emparejamiento es estable o no, se alejan bastante respecto a la probabilidad obtenida mediante la fórmula de Mertens para n muy grandes.
Hemos hecho una explicación en profundidad sobre el funcionamiento de nuestro código, y justificado la implementación del algoritmo para generar entradas aleatorias.

Y es por esto que hemos decidido seguir adelante, analizar nuestros propios resultados (aunque no sabemos si son erróneos) y sacar conclusiones.

Experimentación:

Vamos a estudiar la probabilidad de obtener un emparejamiento estable para distintas entradas, con lo que empezaremos con n´s pequeñas e iremos aumentando hasta obtener una conclusión clara. Ejecutaremos para cada n, 100 veces el algoritmo para obtener un porcentaje lo más real posible.
Cada vez que ejecutemos el algoritmo de Irving, si éste reporta un emparejamiento estable, sumaremos uno a un contador, y al final de las 100 ejecuciones sacaremos el porcentaje de resultados positivos para la n que toque.

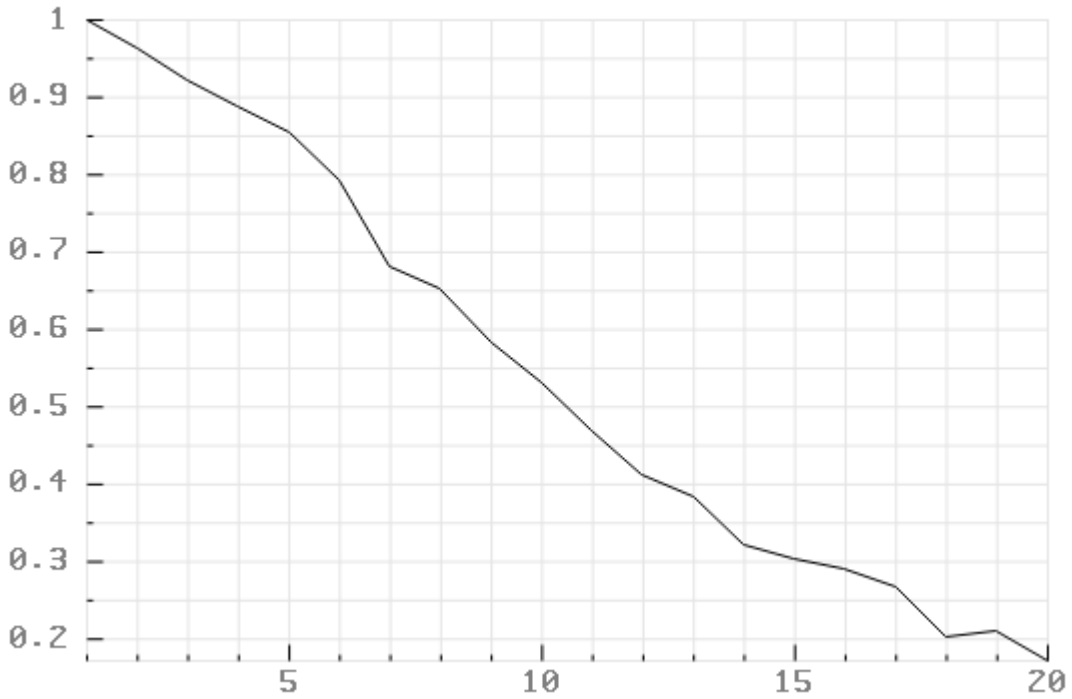

Gráfico 1. eje x: tamaño de la entrada n. eje y: probabilidad de ser estable.

Nuestra primera implementación tenía como base un video de youtube que explicaba paso a paso cómo funciona el algoritmo. Pero este video no explica con detalle la tercera fase de Irving así que fuimos a consultar otras fuentes para corregir nuestro algoritmo [4] [8].
En el gráfico uno podemos observar un gráfico con la correcta (o eso creemos) implementación del algoritmo descrito por Irving donde la tendencia de la

probabilidad parece lineal, nada que ver con la conjetura de Mertens. Al ver unos resultados tan distantes de lo que esperábamos supusimos que el algoritmo fallaba por algún lado. Así que sacando instancias con el código que tenemos hasta el momento y comparando con los resultados que nos da la pagina web http://stablegroups.com [7]. Conseguimos encontrar un caso extraño.
Instancia completa:

0 - 22,16,9,3,15,6,2,28,29,1,14,35,23,5,10,8,13,32,34,21,30,12,4,33,36,17,38,26,39,19,27,7,20,24,18,31,11,25,37,
1 - 29,27,33,10,17,34,8,30,6,0,25,39,11,35,19,12,37,5,32,9,18,21,22,16,20,7,2,3,24,26,23,14,15,28,36,4,38,31,13,
2 - 20,38,4,5,25,19,34,1,27,14,3,23,7,28,15,37,29,12,11,35,31,10,21,13,33,9,0,39,22,30,18,36,24,8,32,16,26,17,6,
3 - 6,28,22,5,24,0,34,8,4,26,37,27,35,32,7,36,13,39,10,20,16,17,30,21,25,29,2,33,1,9,14,31,11,38,15,12,19,23,18,
4 - 37,38,26,13,21,33,15,23,6,11,10,8,36,9,22,39,35,20,14,29,31,18,17,3,2,12,28,19,0,34,32,25,5,30,16,24,7,1,27,
5 - 32,36,2,25,6,20,33,23,24,35,38,28,30,4,26,14,29,9,10,8,1,39,17,15,3,7,0,37,21,12,27,19,11,18,16,22,34,31,13,
6 - 23,33,27,0,37,32,38,28,5,35,7,4,14,10,9,12,11,3,1,2,31,8,16,25,15,36,29,39,18,17,19,24,26,21,30,13,34,20,22,
7 - 29,35,9,10,13,4,24,34,31,37,38,18,23,0,36,26,32,30,17,20,6,19,11,15,12,39,33,28,1,8,25,22,16,21,5,27,2,14,3,
8 - 13,30,23,35,3,9,34,29,18,12,2,5,24,39,6,33,14,25,10,19,22,15,38,1,26,32,4,16,7,28,17,36,0,31,20,37,11,21,27,
9 - 15,23,30,4,18,17,32,16,31,6,1,26,3,33,35,5,13,29,0,11,20,36,27,37,22,19,34,10,14,2,25,24,21,12,39,28,38,8,7,
10 - 24,38,3,15,1,9,19,23,32,30,37,2,20,11,5,28,17,14,33,39,8,36,34,25,6,29,35,0,31,16,26,12,18,4,21,22,7,27,13,
11 - 4,15,12,38,5,8,33,9,37,29,31,17,0,1,28,16,20,24,19,6,39,26,7,35,22,34,21,10,25,18,14,13,2,32,23,3,30,27,36,
12 - 35,4,13,7,10,23,0,36,20,34,37,1,17,39,31,21,24,2,26,3,33,28,14,11,8,16,5,19,38,32,29,9,25,6,27,18,30,15,22,
13 - 12,26,35,30,22,28,19,27,0,21,4,18,33,39,25,32,38,1,31,17,15,37,9,24,14,5,3,11,16,10,7,6,29,2,23,34,8,36,20,
14 - 5,22,32,16,34,17,15,25,26,0,37,7,39,27,19,8,18,33,4,38,29,30,28,12,13,35,3,11,21,23,24,36,9,6,31,2,1,10,20,
15 - 13,6,12,7,39,30,26,18,11,28,31,38,22,19,21,23,35,2,5,17,3,29,9,0,10,36,25,8,14,24,4,32,27,1,34,20,37,16,33,
16 - 11,3,35,31,19,28,34,0,29,2,26,1,4,36,5,8,15,20,23,32,9,10,37,6,14,27,38,25,7,12,18,24,21,13,33,17,30,22,39,
17 - 34,16,9,31,3,8,20,15,2,18,29,7,24,13,28,39,5,25,4,32,27,22,6,0,38,12,19,35,21,26,1,11,30,14,37,36,10,23,33,
18 - 21,29,5,12,39,28,34,22,10,20,9,27,0,37,16,3,13,31,6,8,26,11,7,2,14,33,23,24,15,38,4,36,1,30,25,19,35,17,32,
19 - 22,23,8,24,0,15,17,12,11,1,7,38,35,39,28,26,31,5,27,32,33,29,21,30,18,10,3,13,9,6,20,4,34,36,14,37,16,25,2,
20 - 3,34,36,28,8,17,39,12,23,16,15,6,0,4,32,33,10,24,27,9,19,22,29,30,37,25,2,35,14,21,26,1,5,13,38,7,11,31,18,
21 - 26,35,25,15,34,39,12,14,0,19,1,9,17,13,20,8,27,32,36,22,33,5,30,7,23,38,18,10,28,31,11,24,16,4,3,37,2,6,29,
22 - 21,10,23,36,8,38,31,11,28,9,13,1,4,39,35,32,6,17,24,34,14,20,30,37,16,2,7,0,29,3,33,27,19,25,15,26,5,12,18,
23 - 39,17,15,29,10,8,33,19,24,1,9,4,7,35,12,16,25,26,37,27,32,22,6,31,38,14,2,0,13,30,5,28,21,34,36,20,11,18,3,
24 - 33,18,19,10,6,17,0,37,29,26,31,12,36,23,21,11,7,4,22,34,15,25,14,39,20,9,3,35,2,30,1,32,5,8,16,13,27,28,38,
25 - 10,4,31,2,20,14,23,37,22,17,3,28,5,29,13,27,36,19,34,38,18,15,11,32,26,33,7,39,30,24,12,6,16,21,8,35,0,1,9,
26 - 24,25,27,37,12,34,7,35,2,5,32,19,39,16,9,29,0,21,18,13,38,17,1,31,36,23,15,10,8,11,3,20,33,6,22,30,4,14,28,
27 - 15,10,19,30,7,14,12,28,8,25,29,21,1,39,33,13,0,3,18,2,22,32,17,4,38,34,6,36,37,31,11,23,16,26,20,5,35,9,24,
28 - 5,35,22,27,4,8,25,31,3,17,21,30,39,23,6,38,10,37,33,13,26,14,11,29,24,15,16,2,12,1,36,9,7,19,0,32,18,20,34,
29 - 5,18,4,2,33,37,25,23,19,22,15,26,20,12,35,11,30,17,28,10,9,21,39,38,36,16,6,3,31,8,32,24,34,27,7,0,1,13,14,
30 - 18,0,5,4,21,37,36,17,33,28,34,31,24,29,14,6,1,27,35,38,20,22,25,16,13,7,3,9,23,11,10,32,39,19,8,26,2,15,12,
31 - 5,1,0,26,18,19,6,10,37,24,4,38,8,35,22,23,12,11,39,15,36,3,34,25,7,17,21,28,20,2,33,14,29,32,16,9,13,30,27,
32 - 25,9,30,28,33,24,27,20,36,31,38,6,10,19,5,0,21,4,37,35,11,14,3,26,29,15,2,12,1,39,7,34,23,17,18,13,16,8,22,
33 - 25,16,29,0,17,12,34,9,11,39,24,20,22,10,8,30,1,31,3,27,36,21,35,13,7,28,14,23,19,2,18,37,15,5,38,4,6,32,26,
34 - 19,13,2,4,32,33,37,38,9,20,22,5,28,11,31,14,39,12,36,35,18,25,10,6,17,24,27,30,0,15,8,3,1,16,7,26,23,21,29,
35 - 18,29,31,23,26,38,39,13,21,15,14,20,17,0,24,3,2,34,1,22,5,16,33,25,8,19,37,4,12,10,11,30,27,7,36,32,6,28,9,
36 - 29,19,21,16,17,5,8,28,35,32,25,10,27,39,12,11,13,31,7,24,34,0,23,30,22,9,26,1,6,33,38,14,4,2,37,3,15,18,20,
37 - 31,23,32,19,35,36,26,2,20,5,9,39,21,3,29,38,1,13,30,0,10,8,18,7,34,25,17,14,11,22,6,12,27,24,28,16,33,4,15,
38 - 22,35,26,27,4,3,23,10,2,36,15,14,39,28,0,34,11,17,21,7,24,32,20,29,19,30,6,12,37,8,9,31,33,16,13,18,5,1,25,
39 - 30,12,20,22,37,5,14,34,27,16,0,4,26,8,9,38,23,25,33,1,35,17,32,18,11,28,7,24,10,36,2,6,19,21,3,15,31,29,13,

Instancia después de pasar por la fase 1 de Irving:

0 - 16,3,6,
1 - 10,30,11,31,
2 - 25,
3 - 28,0,8,10,20,
4 - 6,11,
5 - 32,36,
6 - 0,32,7,4,
7 - 26,6,15,
8 - 23,3,14,19,

9 - 17,
10 - 24,3,1,
11 - 4,15,1,16,
12 - 13,
13 - 12,
14 - 39,27,8,30,
15 - 7,11,23,
16 - 11,0,
17 - 9,
18 - 29,
19 - 8,24,
20 - 3,39,
21 - 35,
22 - 38,
23 - 15,8,
24 - 19,10,
25 - 2,
26 - 37,7,
27 - 30,14,28,
28 - 27,3,
29 - 18,
30 - 14,1,27,
31 - 1,37,
32 - 36,6,5,
33 - 34,
34 - 33,
35 - 21,
36 - 5,32,
37 - 31,26,
38 - 22,
39 - 20,14,

Instancia rechazada después de la fase 2 de Irving:

0 - 16,
1 - 10,
2 - 25,
3 - 28,
4 - 11,
5 - 36,
6 - 32,7,
7 - 26,6,
8 - 23,14,19,
9 - 17,
10 - 24,1,
11 - 4,15,
12 - 13,
13 - 12,
14 - 27,8,
15 - 11,23,
16 - 0,
17 - 9,
18 - 29,
19 - 8,24,
20 - 39,
21 - 35,
22 - 38,
23 - 15,8,
24 - 19,10,
25 - 2,
26 - 7,
27 - 14,
28 - 3,
29 - 18,
30 -

31 - 37,
32 - 6,
33 - 34,
34 - 33,
35 - 21,
36 - 5,
37 - 31,
38 - 22

Referente al caso anterior, si nos fijamos en el último paso antes de rechazar la instancia tenemos la listas de la persona 1 - 10,30 y la lista de la persona 30 - 1. Claramente, la persona 30 solo puede ser emparejada con la 1 y si empezamos a ejecutar la segunda fase de Irving de la 1 nos manda al segundo de esta lista, el 30 que después nos manda al último de la lista de este, que es el mismo 1. Y así encontramos un ciclo que no debería considerarse, como tampoco se consideran los ciclos de las listas donde solo hay un elemento. Así que nuestra propuesta para solucionar este problema es asignar como parejas definitivas aquellas que solo les queda una opción a en su lista.

Como casos de estos se podrían dar muchísimos más puesto que como mayor sea la n mayor es la probabilidad que se de algún caso de estos. Lo veremos más adelante.

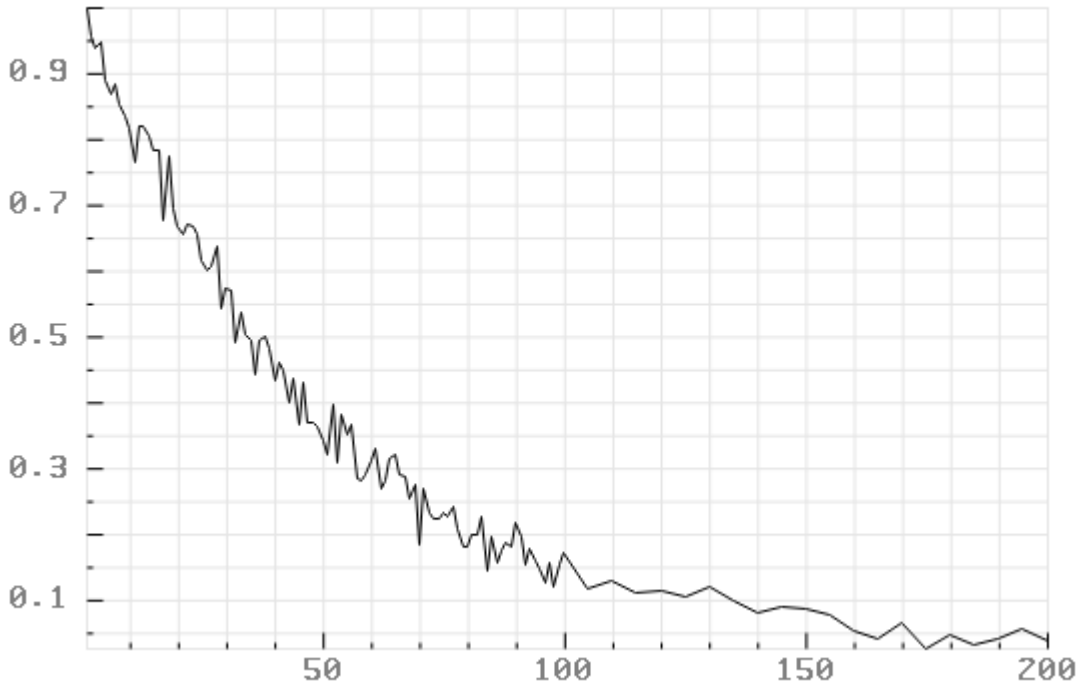

Gráfico 2. eje x: tamaño de la entrada n. eje y: probabilidad de ser estable.

Con el código corregido, hemos conseguido sacar un gráfico con un mejor aspecto a diferencia del anterior y con una forma parecida el que propone Mertens. Aun así

nuestra probabilidad es muy cercana a 0 con n = 200, cuando Martens nos dice que esta probabilidad debería ser 0,484975. No sabemos si Mertens está en lo cierto o no pero lo que sí podemos afirmar es que la probabilidad para cualquier n debería ser mayor o igual a la que nos acercamos nosotros, nuestro algoritmo aún le falta contemplar algunos casos que no se pueden apreciar con n pequeñas pero si son más abundantes en n muy grandes.

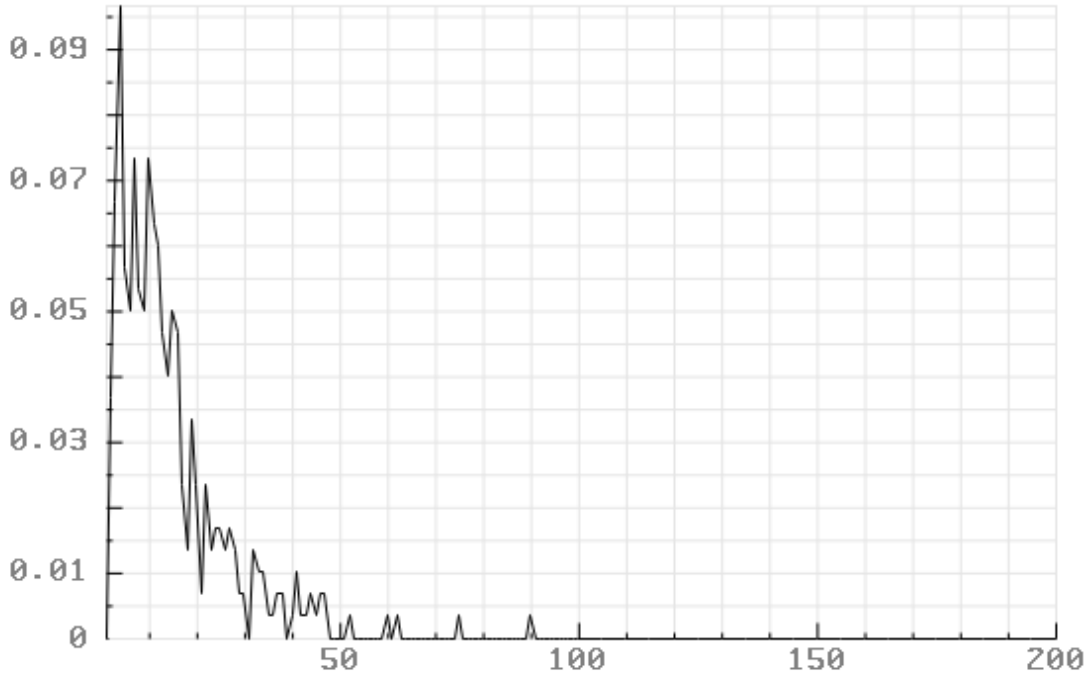

Gráfico 3. eje x: tamaño de la entrada n.
eje y: probabilidad de no ser estable en fase 1.

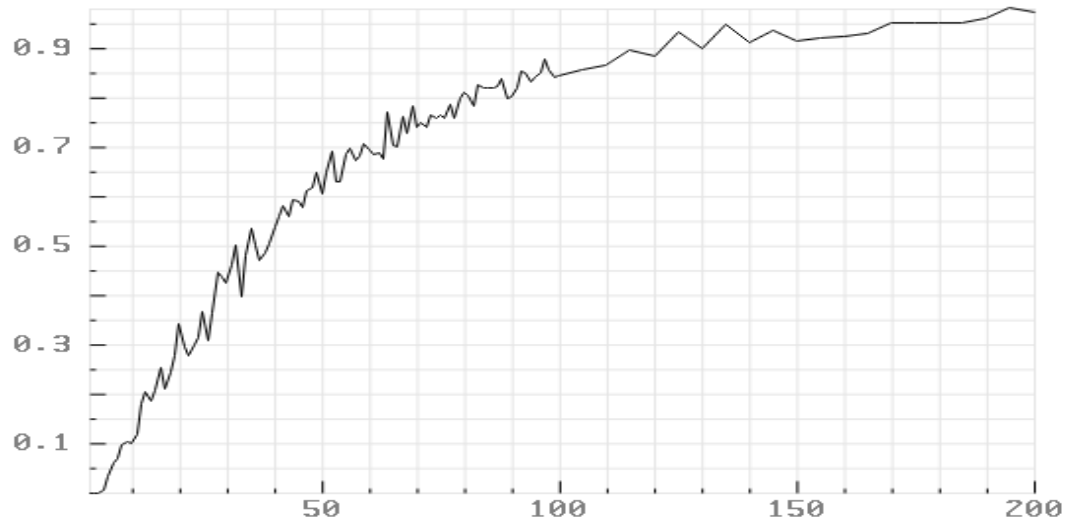

Gráfico 4. eje x: tamaño de la entrada n.
eje y: probabilidad de no ser estable en fase 2.

En la página anterior, tenemos los gráficos 3 y 4 que nos aproximan en qué momento se rechaza la entrada como una instancia estable del problema. Antes de todo notar la diferencia que hay entre el eje *y* del gráfico 3 donde el valor máximo es 0,095… mientras que en el gráfico 4 es 1.
También podemos observar que los rechazos con n muy pequeñas, de hasta 15 más o menos, el porcentaje de rechazos en fase 1 y en fase 2 no distan mucho. Pero si nos fijamos a partir de n = 50, los rechazos en fase 1 se podrían obviar casi puesto que no hay casi ni uno. Esto se debe a que el tamaño de las listas aumenta y por ende sus posibles permutaciones y con ello la probabilidad de que un mismo número se quede en las últimas posiciones de todas las listas. Cuando sucede eso hay más probabilidad que la instancia se rechace en la fase 1.
Nosotros hemos supuesto que si este tipo de patrón de rechazo está tan presente en las primeras iteraciones, su presencia no desaparece cuando las n son más grandes. Pero sí que aparecen nuevos patrones de rechazo que hacen a los otros tener menos presencia. También suponemos que en nuestro algoritmo se rechazan instancias que no deberían, puesto que hemos detectado que había un caso que no se contemplaba y lo hemos incluido, seguramente existen otros casos que aún no hemos tenido en cuenta y aun no hemos incluido.
Para finalizar la experimentación, hemos sacado un gráfico más preciso sobre nuestra probabilidad de estabilidad en las entradas para cada n ejecutando 2000 iteraciones por n.

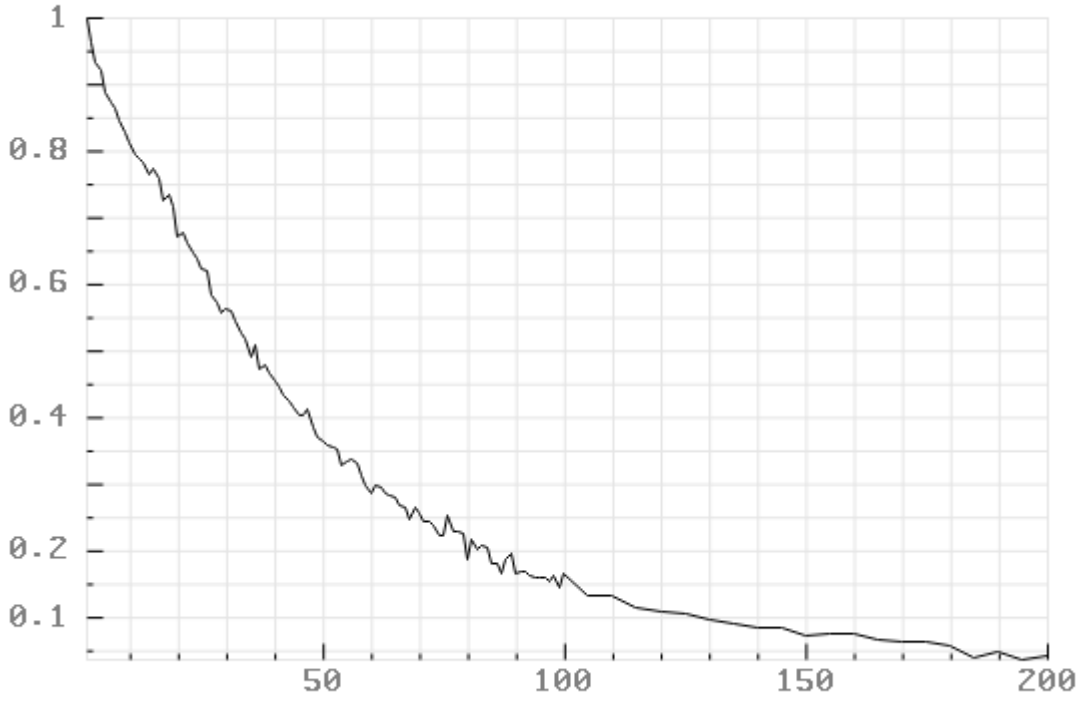

Gráfico 5. eje x: tamaño de la entrada n. eje y: probabilidad de ser estable.

Conclusión:

Tras la abstracción de los datos, y realizar un profundo análisis nos quedamos con la sensación de no haber llegado a una conclusión clara. Hemos visto que nuestra implementación del algoritmo no tiene en cuenta ciertos aspectos y casos especiales, que hacen que la probabilidad calculada se aleje mucho de los datos reportados por Mertens. Aún así, consideraremos los datos recogidos como "buenos" y haremos nuestra propia interpretación.

En primer lugar, si nos fijamos en n´s pequeñas aún podemos ver probabilidades más o menos aceptables en torno a 0,50, pero en el momento que nos acercamos a n = 100 esta probabilidad es prácticamente despreciable. Si lo relacionamos con la conjetura de Mertens, estamos viendo el mismo fenómeno a otras escalas de n´s, pero con el mismo trasfondo. A medida que la entrada crece, la probabilidad de encontrar un emparejamiento estable baja desmesuradamente.

En conclusión, considerando nuestros datos recogidos y después de haber comparado resultados en proporción a mertens, vemos que la tendencia decreciente de la conjetura, también la observamos en nuestra experimentación pero aun así no podemos corroborar del todo la conjetura de Mertens.

Bibliografía:

1. *Irving's Algorithm and Stable Roommates Problem. (2014, 5 diciembre). Irving's Algorithm and Stable Roommates Problem.* https://www.youtube.com/watch?v=5QLxAp8mRKo&t=402s
2. *Irving's Algorithm - The Stable Roommates Problem. (2014, 5 diciembre). Irving's Algorithm - The Stable Roommates Problem.* https://www.youtube.com/watch?v=mq1aYnXnCNI
3. *Irving's Algorithm - The Stable Roommates Problem. (2014, 5 diciembre). Irving's Algorithm - The Stable Roommates Problem.* https://www.youtube.com/watch?v=mq1aYnXnCNI
4. *Department of Mathematics, University of Saijord Salford MS 4WT, United Kingdom. (1984, 1 mayo). An Efficient Algorithm fo the «Stable Roommates» Problem. An Efficient Algorithm fo the «Stable Roommates» Problem.* https://uvacs2102.github.io/docs/roomates.pdf
5. *StackOverflow. (s. f.). StackOverflow.* https://stackoverflow.com/questions/28656004/c-random-doesnt-workreturns-same-value-always
6. *Rand. C++ . Rand.* http://www.cplusplus.com/reference/cstdlib/srand/

7. *Stable Groups. (s. f.). Stable Groups* *http://stablegroups.com/preferences/spreadsheet.php*
8. *Stable roommates problem. (s. f.). Wikipedia/Stable_roommates_problem.* *https://en.wikipedia.org/wiki/Stable_roommates_problem*