



CENTRO DE CIENCIAS BÁSICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN
METAHEURÍSTICAS I
7° "A"

SEGUNDA EVALUACIÓN PARCIAL

Profesor: Francisco Javier Luna Rosas

Alumnos:

Almeida Ortega Andrea Melissa
Espinoza Sánchez Joel Alejandro
Flores Fernández Óscar Alonso
Gómez Garza Dariana
González Arenas Fernando Francisco
Orocio García Hiram Efraín

Fecha de Entrega: Aguascalientes, Ags., 31 de octubre de 2021

Índice

Explicación de la propuesta Cliente – Servidor del Equipo -----	1
El Servidor-----	3
El Cliente -----	3
¿Cómo funciona el Planificador de Carga? -----	4
Distribución de Carga Usando un Algoritmo Genético -----	8
Definición del algoritmo genético -----	9
El procedimiento del algoritmo genético -----	11
Generación de la población inicial -----	11
Evaluación y Función Objetivo -----	12
Etapa de Cruzamiento -----	14
Etapa de Mutación -----	16
Procedimiento de Lemantización -----	25
Resultados -----	29
Resultados de tiempo secuencial -----	30
Resultados de tiempo paralelo -----	32
Conclusiones -----	34
Referencias Bibliográficas -----	36
Anexos -----	38

Segunda Evaluación Parcial

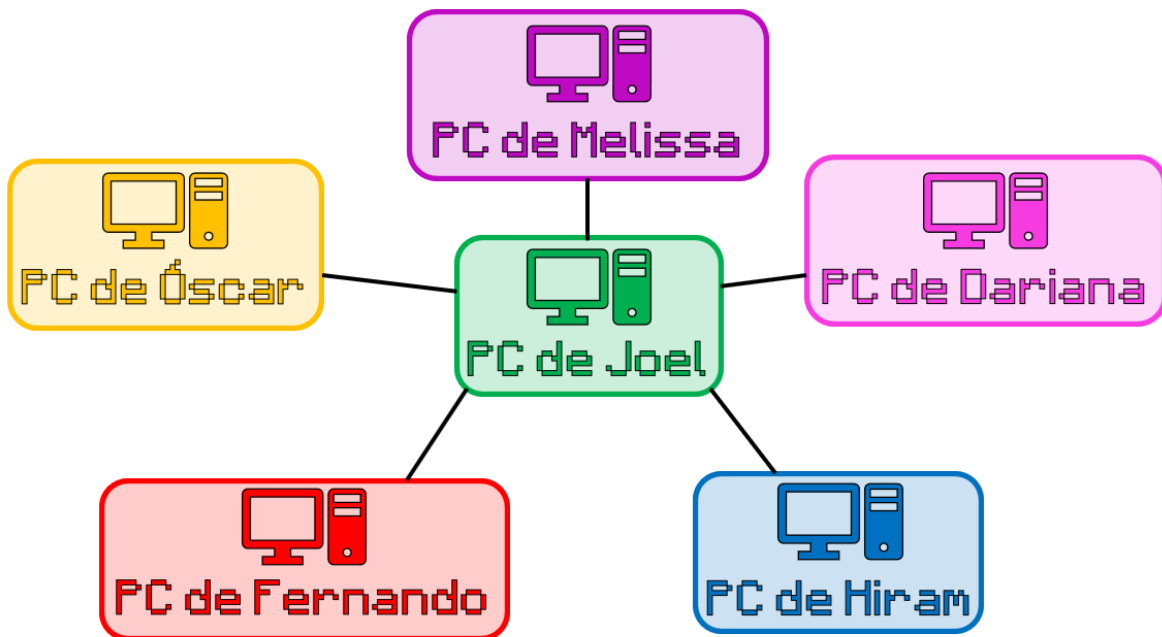
Explicación de la propuesta Cliente – Servidor del Equipo

La propuesta de arquitectura del equipo se sustenta en dos tipos de archivos base en código de Python: un archivo a nivel de servidor y otro a nivel de cliente.

A partir de ahora y para el resto del documento se denotarán las computadoras como:

- PC de Melissa.
- PC de Joel
- PC de Óscar.
- PC de Dariana.
- PC de Fernando.
- PC de Hiram

La arquitectura propuesta es la siguiente:



La idea de la arquitectura será principalmente un servidor central conectado a cinco clientes a modo de topología tipo estrella. El servidor también actuará como otro cliente más. En el sentido estricto de que para este trabajo, el cliente cumplirá la función de procesar los tweets y realizar el trabajo que regresará al servidor; para efectos de la actual práctica, se quiere decir, que el servidor también intentará ser un servidor que esté trabajando junto con los demás equipos, pues en términos rígidos del presente proyecto, el servidor tiene como única función realizar la distribución de carga a los demás equipos, es por tanto, que la PC de Joel realizará esta tarea junto con el procesamiento de información, en la que estarán los equipos de los seis integrantes trabajando. En este sentido, los archivos básicos serán dos: el archivo del servidor y el archivo del cliente.

Por la estandarización de nombres pasada, los equipos cumplen las siguientes funciones:

PC de Melissa	Cliente	
PC de Joel	Cliente	Servidor
PC de Óscar	Cliente	
PC de Dariana	Cliente	
PC de Fernando	Cliente	
PC de Hiram	Cliente	

Para ello, se desarrollaron dos archivos: el archivo en ejecución del servidor y el archivo del cliente.

El Servidor

El servidor tiene dos fases de código (véase anexo 1), donde la primera esperará la recepción de los otros cinco equipos:

```
### Conexión
import socket

s = socket.socket()
host = socket.gethostname()
port = 8080
s.bind((host,port))
s.listen(1)
print(host)
print("Esperando conexiones...")
conn, addr = s.accept()
print(addr + " se ha conectado")
```

Por este medio, los usuarios se conectarán al servidor, quien esperará su registro en la conexión y posteriormente será notificado el servidor.

Una vez que estén conectados los equipos, el servidor pasará a transmitir archivos de control a los usuarios. Más adelante se explicarán estos archivos de control.

```
### Transfiriendo archivos
filename = "Control.txt"
file = open(filename, 'rb')
file_data = file.read(1024)
conn.send(file_data)
print("Archivo enviado")
```

El Cliente

Por otra parte, el archivo del cliente (véase anexo 2), será quien espere información del servidor, que al ser proporcionada, la ingresará en el programa.

Así se podrá conectar en el servidor una vez que el servidor comience a ejecutarse:

```
### Conexión
import socket

s = socket.socket()
print("Ingresa la dirección del host")
host = input()
port = 8080
s.connect((host,port))
print("Conectado")
```

Después, el archivo contiene otro módulo de transferencia de datos. Es por este módulo por el que se enviarán datos del servidor al cliente y viceversa.

```
### Transfiriendo archivos
filename = "Control.txt"
file = open(filename, 'wb')
file_data = s.recv(1024)
file.write(file_data)
file.close()
print("Recibido")
```



Los archivos principales por enviar serán los tweets sin analizar del servidor al cliente y éstos mismos analizados del cliente al servidor, sin embargo, otros archivos de control serán enviados entre los equipos.


¿Cómo funciona el planificador de carga?

El planificador de carga en sí será todo documento compartido por medio de archivos de formato .txt entre el servidor y el cliente.



Una prueba de trabajo de ellos es el archivo prueba.txt ubicado en el equipo del servidor junto a su respectivo directorio. En este caso de entrega, el archivo no contiene ninguna información relevante de la planificación de tareas, sin

embargo, pueden mostrarse los dos directorios de servidor y cliente respectivamente a continuación:

 prueba.txt	06/10/2021 06:38 p. m.	Documento de te...	1 KB
 server.py	06/10/2021 11:30 p. m.	Archivo PY	1 KB

 clientJoul.py	06/10/2021 11:30 p. m.	Archivo PY	1 KB
---	------------------------	------------	------

Y al ejecutarse ambos programas y estar ambos equipos en contacto, el cliente obtiene este archivo de texto:

 clientJoul.py	06/10/2021 11:30 p. m.	Archivo PY	1 KB
 prueba.txt	06/10/2021 06:38 p. m.	Documento de te...	1 KB

Sin embargo, el verdadero orden de documentos en cliente y servidor será regularmente el siguiente:

- El servidor:
 - El archivo Python del servidor `server.py`.
 - Un archivo de control de planificación de carga y administración `Control-Servidor.txt`.
- El cliente:
 - El archivo Python del servidor `client.py`.
 - Un archivo de recepción de control de planificación de carga y administración `Recepcion-Nombre.txt`. El nombre del archivo variará según el integrante del equipo. Ejemplos: `Recepcion-Melissa.txt`, `Recepcion-Hiram.txt`.
 - Un archivo de emisión de control de planificación de carga y administración `Emission-Nombre.txt`. El nombre del archivo variará según el integrante del equipo. Ejemplos: `Emission-Dariana.txt`, `Emission-Oscar.txt`.

La función de los archivos de texto plano serán encargarse del plan de carga. Al realizarse, los números comenzarán a cambiar en los archivos. Primeramente el

archivo de control del servidor (véase anexo 3) tendrá la información general de los clientes.

El estado del servidor y de cada cliente. 0 significa que está apagado el servidor o los clientes no están en línea y 1 significa que el servidor está activo o los clientes están conectados a la red.

```
Estado: 0
Cliente Melissa: 0
Cliente Joel: 0
Cliente Oscar: 0
Cliente Dariana: 0
Cliente Fernando: 0
Cliente Hiram: 0
```

Por ejemplo, cuando el servidor esté activo, el primer renglón estará activo y los integrantes conectados en ese momento estarán señalados en el archivo también:

```
Estado: 1
Cliente Melissa: 1
Cliente Joel: 0
Cliente Oscar: 0
Cliente Dariana: 0
Cliente Fernando: 1
Cliente Hiram: 1
```

Posteriormente se encuentra un apartado para cada integrante en el que se reporta el progreso personal de cada integrante (véase anexo 4) y la asignación de cada uno a nivel cíclico dinámico como se muestra a continuación:

MELISSA

```
Tweets enviados a procesar: 0
Tweets ya procesados: 0
Tiempo: 0
```

Este formato se repite para los seis participantes del proyecto.

En un ejemplo por dar explicación a cada variable tomada en cuenta para cada usuario, se propone el siguiente ejemplo a explicar:

MELISSA

Tweets enviados a procesar: 400

Tweets ya procesados: 1000

Tiempo: 1.2302

La imagen anterior estaría proporcionando la información del PC de Melissa donde se habría indicado por el algoritmo genético habría calculado que lo óptimo es enviarle en el actual periodo de tiempo 400 tweets a procesar, mientras que ya tendría 1,000 tweets reportados al servidor como procesados en un tiempo de 1.2302 segundos.

Por parte del cliente, estos datos serían enviados al archivo de recepción del cliente (véase anexo 5) en el que se le indicaría primeramente si su conexión ya se realizó y posteriormente los tweets asignados para que trabaje su equipo, junto con un archivo CSV con los tweets indicados.

MELISSA

Conexion: 0

Tweets recibidos para procesar: 0

Cuando el cliente termina de trabajar en su procesamiento, escribirá sobre el archivo de emisión que será enviado al servidor con los datos que actualizará sobre el archivo de control, proporcionando los siguientes datos:

MELISSA

Tweets procesados: 0

Tiempo: 0

De modo que la información del plan de carga se mantendría en flujo en sintonía con los archivos CSV enviados a través de los usuarios de esta red y, donde si todo funciona bien, el planificador de carga mantendría una gestión de cada usuario correcta.

Distribución de Carga Usando un Algoritmo Genético

Se basó mucho en el artículo proporcionado en clases *“Observations on Using Genetic Algorithms for Dynamic Load-Balancing”* sin embargo, propone una arquitectura muy atractiva para la distribución de carga tanto dinámica como variable, pues en él se expone una idea con tareas de distinto peso.

En el caso propio, el equipo pensó en principio que no existiría esto en la presentación del algoritmo, ya que las tareas a procesar son el análisis de sentimientos de los tweets. Cada tweet deberá pasar por el mismo procedimiento y por ello se consideraba que se analizaría con el mismo peso, ya que se cree que las diferencias de tiempo pueden ser discriminadas. Esto debido a que primero se planteó una asignación de pesos a cada tweet bajo la fórmula:

$$Peso = \ln(Palabras)$$

Pero para la velocidad de los tiempos de ejecución, el comportamiento logarítmico era despreciable, ya que la cantidad máxima de palabras era de 200 en cada tweet, lo que hizo considerar más en despreciar estas diferencias.

Entonces frente a un algoritmo de balanceo de carga de tareas donde cada tarea es un tweet y por lo tanto, cada tarea pesa lo mismo, la pregunta del equipo recaía en cómo realizar este procedimiento interesante, ya que se designó como valor 1 a cada tweet pero sin dar aún una propuesta atractiva en el proceso, lo más sencillo que podía hacerse, incluso sin un algoritmo genético, era repartir todas las tareas entre seis y que el procedimiento siga su curso.

Esta propuesta, aunque cierta, no realizaba ninguna tarea de evaluación heurística, por lo que el equipo discutió que este proyecto consiste en la distribución de carga de modo que un equipo no esté trabajando más de la cuenta mientras que otro equipo carece de trabajo. Para ello se propuso realizar el balance de carga con todas las tareas teniendo el mismo peso pero variando la potencia del procesador

de cada computadora, por lo que primeramente se midió el poder de procesamiento de cada equipo.

Primeramente se diseñó un problema computacionalmente fácil pero que requiriera de mucho tiempo para completarse (véase el anexo 5) y se reportaron los tiempos de ejecución de cada equipo con una serie de ejecuciones.

Después de realizar variadas ejecuciones del problema computacional anteriormente mencionado, se realizó una media de tiempo por PC. Los resultados fueron los siguientes (véase anexo 6 para evidencias de los resultados):

PC de Melissa	9.311
PC de Joel	14.8685
PC de Óscar	7.458
PC de Dariana	90.8248
PC de Fernando	16.1
PC de Hiram	23.006

Con estos datos conocidos, la idea era asignar la cantidad adecuada de tweets para que todos los procesadores, con su debida potencia, procesaran los tweets que les permitieran sus capacidades.

Definición del algoritmo genético

El algoritmo genético propuesto está planteado bajo los términos siguientes:

- Sea q_i (*Quantity of Individuals*) una variable entera positiva que representará la cantidad de cromosomas en la población del algoritmo genético.
- Sea *sample* una matriz computacional de tres columnas y q_i filas:
- Sea *qelitism* (*Quantity of Elitism*) una variable entera positiva que representará la cantidad de individuos que se heredarán por elitismo.

- Sea pc (*Probability of Crossing*) una variable entera dentro del intervalo $[0,100]$ que representará la probabilidad de que se active el cruzamiento.
- Sea pm (*Probability of Mutation*) una variable entera dentro del intervalo $[0,100]$ que representará la probabilidad de que se active la mutación.
- Sea qg (*Quantity of Generations*) una variable entera positiva que representará la cantidad de generaciones que se le indicarán al algoritmo genético para repetir el procedimiento.

sample tendrá la siguiente forma:

$$\begin{bmatrix} i_1 & f(i_1) & F(i_1) \\ i_2 & f(i_2) & F(i_2) \\ i_3 & f(i_3) & F(i_3) \\ \vdots & \vdots & \vdots \\ i_{qi} & f(i_{qi}) & F(i_{qi}) \end{bmatrix}$$

Donde i_n es el n-ésimo cromosoma de la población; $f(i_n)$ es el resultado de evaluar el n-ésimo cromosoma en la función objetivo – que más adelante se tratará y desarrollará – y $F(i_n)$ es la función acumulada objetivo desde el primer cromosoma hasta el n-ésimo cromosoma, es decir $F(i_n)$ está definida como:

$$F(i_n) = \sum_{k=1}^n f(i_k)$$

También cabe aclarar que un cromosoma general tiene una composición compleja, pues todos los cromosomas están definidos como una tupla de seis números de la siguiente forma:

$$i_n = (iM_n \quad iJ_n \quad iO_n \quad iD_n \quad iF_n \quad iH_n)$$

Donde:

- iM_n será la asignación de tweets para Melissa en la n-ésima solución.
- iJ_n será la asignación de tweets para Joel en la n-ésima solución.

- iO_n será la asignación de tweets para Óscar en la n-ésima solución.
- iD_n será la asignación de tweets para Dariana en la n-ésima solución.
- iF_n será la asignación de tweets para Fernando en la n-ésima solución.
- iH_n será la asignación de tweets para Hiram en la n-ésima solución.

La representación anterior está sujeta a la siguiente condición:

$$iM_n + iJ_n + iO_n + iD_n + iF_n + iH_n = total_tweets$$

Donde `total_tweets` es una variable computacional entera positiva que tendrá almacenado el valor de tweets total a procesar.

Y debido a que de ambas maneras se puede representar un cromosoma, la siguiente igualdad se satisface.

$$sample = \begin{bmatrix} i_1 & f(i_1) & F(i_1) \\ i_2 & f(i_2) & F(i_2) \\ i_3 & f(i_3) & F(i_3) \\ \vdots & \vdots & \vdots \\ i_{qi} & f(i_{qi}) & F(i_{qi}) \end{bmatrix} = \begin{bmatrix} (iM_1 & iJ_1 & iO_1 & iD_1 & iF_1 & iH_1) & f(i_1) & F(i_1) \\ (iM_2 & iJ_2 & iO_2 & iD_2 & iF_2 & iH_2) & f(i_2) & F(i_2) \\ (iM_3 & iJ_3 & iO_3 & iD_3 & iF_3 & iH_3) & f(i_3) & F(i_3) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ (iM_{qi} & iJ_{qi} & iO_{qi} & iD_{qi} & iF_{qi} & iH_{qi}) & f(i_{qi}) & F(i_{qi}) \end{bmatrix}$$

Es decir, es igual la representación de los cromosomas aunque se hablará con mayor profundidad usando la segunda representación.

El procedimiento del algoritmo genético

El procedimiento del algoritmo genético está definido de la siguiente manera:

Generación de la población inicial

Sea `remaining_tweets` una variable con características similares a `total_tweets`.

El procedimiento inicia estableciendo el valor de `total_tweets` en `remaining_tweets`.

Posteriormente se generará un número aleatorio entre 0 y `remaining_tweets` que se le asignará al primer elemento de la tupla del cromosoma. Asimismo se restará este valor a `remaining_tweets`, valor que se asignará a sí mismo.

Con el valor actualizado, se generará nuevamente otro número aleatorio entre 0 y el nuevo valor de `remaining_tweets` y se le asignará al segundo elemento de la tupla del cromosoma. Nuevamente este valor se restará este valor a `remaining_tweets`, valor que se asignará a sí mismo.

Este procedimiento se repite para los seis elementos de la tupla.

En caso de que `remaining_tweets` no sea cero para cuando se haya terminado de asignar valor a cada uno de los seis elementos, entonces se elegirá un elemento al azar para asignar el resto del valor de `remaining_tweets` a este elemento de la tupla.

Este procedimiento se repite para cada cromosoma dentro de la población.

Evaluación y Función Objetivo

Denotaremos para este punto los tiempos indicados por cada integrante de equipo como `timeM`, `timeJ`, `timeO`, `timeD`, `timeF` y `timeH` respectivamente.

Para plantear la función objetivo primeramente se pensó que la cantidad de tweets dividido entre el tiempo registrado de una persona deberá ser muy similar a la cantidad de tweets dividido entre el tiempo registrado de otra persona, por ejemplo:

$$\frac{iM_k}{timeM} = \frac{iJ_k}{timeJ} \quad \forall k \in \{1,2,3, \dots, qi\}$$

Esto debe ocurrir para todos los integrantes del equipo:

$$\frac{iM_k}{timeM} = \frac{iJ_k}{timeJ} = \frac{iO_k}{timeO} = \frac{iD_k}{timeD} = \frac{iF_k}{timeF} = \frac{iH_k}{timeH} \quad \forall k \in \{1,2,3, \dots, qi\}$$

La mejor forma para ajustar todos estos valores a uno similar podrá ser la herramienta estadística conocida como la varianza. Para ello primero se calculó la media de todos estos valores:

$$Media = \frac{\left(\frac{iM_k}{timeM} + \frac{iJ_k}{timeJ} + \frac{iO_k}{timeO} + \frac{iD_k}{timeD} + \frac{iF_k}{timeF} + \frac{iH_k}{timeH} \right)}{6}$$

Posteriormente se calculará un valor muy similar a la definición formal de varianza en Estadística Descriptiva:

$$\sigma^2 = \left| \frac{iM_k}{timeM} - Media \right| + \left| \frac{iJ_k}{timeJ} - Media \right| + \left| \frac{iO_k}{timeO} - Media \right| + \left| \frac{iD_k}{timeD} - Media \right| + \left| \frac{iF_k}{timeF} - Media \right| + \left| \frac{iH_k}{timeH} - Media \right|$$

Ahora, el valor más pequeño de σ^2 será el que describa mejor a un cromosoma, pero por fines prácticos del grupo, se conoce mejor la forma de evaluarlos cuando una función expresa a un cromosoma como “mejor” cuando ésta le asigna un valor mayor, para ello se invirtió la función:

$$Función\ objetivo = \frac{1}{\sigma^2}$$

Finalmente, se tuvieron pequeños inconvenientes en el desarrollo del código que llevaron a corregir la función debido a sus asíntotas, de modo que la función final será la siguiente:

$$Función\ objetivo = \frac{1}{(\sigma^2 \times 1,000) + 1} \times 100$$

Esta misma función califica a los cromosomas, por lo que con ella, pueden ordenarse los cromosomas y realizar elitismo. En tal caso, el algoritmo tomará los *qelitism* mejores que serán directamente insertados en la nueva población.

Etapa de Cruzamiento

De la misma forma como la función objetivo es útil para reportar a los mejores cromosomas y aplicar elitismo, ésta también funciona para realizar la selección por ruleta, que es una selección por ruleta tradicional.

Una vez que se seleccionan dos cromosomas, el siguiente operador a ejecutar será el cruzamiento, que será difícil ejecutar debido al modelo planteado.

Sean las tuplas $(iM_h \ iJ_h \ iO_h \ iD_h \ iF_h \ iH_h)$ y $(iM_k \ iJ_k \ iO_k \ iD_k \ iF_k \ iH_k)$ las elegidas por ruleta para realizar cruzamiento.

La primera idea era adaptar el cruzamiento tradicional del algoritmo genético simple. Esto implica realizar un punto de corte en ambas tuplas y cambiar los valores. Es decir, un posible resultado de las tuplas podía ser el siguiente:

- $(iM_h \ iJ_h \ iO_k \ iD_k \ iF_k \ iH_k)$
- $(iM_k \ iJ_k \ iO_h \ iD_h \ iF_h \ iH_h)$

El problema es que este cruzamiento no garantiza que la condición de suma (donde la suma de todos los elementos de una tupla resulte en el total de tweets) se satisfaga después de aplicar el operador, por lo que la idea se descartó.

Sin embargo, en lugar de realizar un punto de corte, la idea anterior le proporcionó al equipo una segunda propuesta que podría ser la definitiva, pues en lugar de realizar un punto de corte, quizá lo mejor sería sólo intercambiar un miembro de cada tupla, por ejemplo, el resultado sería el siguiente:

- $(iM_h \ iJ_h \ iO_k \ iD_h \ iF_h \ iH_h)$
- $(iM_k \ iJ_k \ iO_h \ iD_k \ iF_k \ iH_k)$

Nuevamente existe el problema que la condición de suma no se cumplirá después de aplicar el cruzamiento, puesto que, estrictamente deberán ser números distintos para que el cruzamiento tenga sentido, sin embargo, la diferencia de valor entre un elemento y otro causará que en una tupla exista un excedente de tweets a la cantidad total y en el otro elemento habrá una falta de tweets para alcanzar la cantidad total, por lo que la idea sería también descartada.

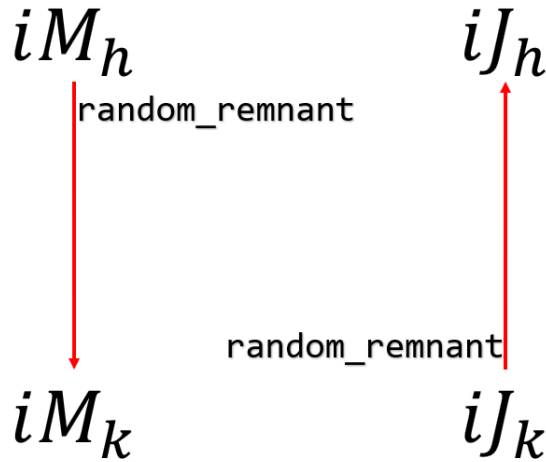
Sin embargo, esta idea anterior llevó a la propuesta final para el cruzamiento, pues esta diferencia podría recuperarse en otro elemento de la tupla, es entonces que se planteó la siguiente propuesta, que fue implementada en la realización del presente documento:

Supóngase que se eligen aleatoriamente los primeros dos elementos de la tupla para intercambiarse: iM_h , iJ_h , iM_k e iJ_k .

Se buscará el valor mínimo de estos valores:

$$top = \min(iM_h, iJ_h, iM_k, iJ_k)$$

Posteriormente se elegirá un número aleatorio entre 0 y top que se denotará como $random_remnant$. Este valor rotará entre los elementos elegidos:



Es decir, después del cruzamiento, las tuplas tendrán los siguientes valores:

- $(iM_h - random_remnant \quad iJ_h + random_remnant \quad iO_h \quad iD_h \quad iF_h \quad iH_h)$
- $(iM_k + random_remnant \quad iJ_k - random_remnant \quad iO_k \quad iD_k \quad iF_k \quad iH_k)$

Así, es claro que la cerradura de suma se mantiene para ambos cromosomas.

El modelo de cruzamiento está inspirado en una técnica Simplex de Investigación de Operaciones para balanceo de cargas en el método mencionado anteriormente. Todos los números de las tuplas se combinarán para crear dos hijos.

Este proceso se realiza tres veces (cada vez involucra a dos elementos de una tupla, de modo que hacerlo tres veces involucra a todos los elementos de la tupla) para cada cromosoma siempre que el cruzamiento se active.

Etapas de Mutación

La mutación realiza un procedimiento similar. En este caso se toma el número menor de la tupla y se generará un número aleatorio entre 0 y el menor de toda la tupla y se rotará este número que se denotará como *add*.

Sea $(iM_h \quad iJ_h \quad iO_h \quad iD_h \quad iF_h \quad iH_h)$ la tupla que activa el proceso de mutación.

Después de realizar la mutación, la tupla tendrá el siguiente aspecto:

$$(iM_h + add \quad iJ_h - add \quad iO_h + add \quad iD_h - add \quad iF_h + add \quad iH_h - add)$$

Al terminar el procedimiento de mutación, todos los operadores del algoritmo genético habrían sido aplicados, por lo que, si se especificó realizar otra generación, la iteración regresaría a repetir este proceso, de lo contrario, el algoritmo genético habría terminado.

Ahora se llevó lo anteriormente explicado a código Python (véase anexo 7) y lo primero que se realizó fue calibrar el algoritmo genético:

```
##% Calibrar variables
total_tweets = 6942021
qe = 6 # Inamovible
qi = 60 # Movible
pc = 75 # Movible
pm = 40 # Movible
qelitism = 5 # Movible
qg = 10
sample = [[], [], []]
power_process = [9.311, 14.8685, 7.458, 90.8248, 16.1, 23.006] # Orden alfabético: Melissa, Joel, Óscar, Dariana,
ig = 0
```

Posteriormente se crea la población inicial y se realizan las evaluaciones con respecto a la función objetivo:

```
##% AG: Creación de la población aleatoria
sample[0] = FirstSample(qe,qi,total_tweets)
PrintSample(qi, sample)

##% AG: Procedimiento de cada generación
while ig < qg:
    print("Población " + str(ig))
    print("")

    # Evaluamos a todas las muestras
    sample[1] = evaluate(qi,sample,power_process)

    print("Población " + str(ig) + " al ser evaluada")
    PrintSample(qi,sample)
    print("")

    # Evaluamos la suma acumulada
    sample[2] = evaluateA(qi,sample)

    print("Población " + str(ig) + " al ser evaluada con acumulación")
    PrintSample(qi,sample)
    print("")

    # Ordenamos por valor de f(x) es decir, por sample[1]
    sample = Sorting(sample, qi)

    print("Población " + str(ig) + " al ser ordenada")
    PrintSample(qi,sample)
    print("")

    # Reevaluamos la suma acumulada
    sample[2] = evaluateA(qi,sample)

    print("Población " + str(ig) + " al ser reevaluada")
    PrintSample(qi,sample)
    print("")
```

Luego se crea una nueva estructura para la nueva generación y se aprovecha para pasar por elitismo a los mejores individuos:

```
# Creamos una nueva muestra vacía
newSample = newSampleCreator(qi)

print("Nueva muestra " + str(ig + 1) + " vacía")
print(newSample)
print("")

print("Se aplicará elitismo")
x = input()

# Se aplica elitismo
ii = 0
while ii < qelitism:
    newSample[0][ii] = sample[0][ii]
    #newSample[1][ii] = sample[1][ii]
    #newSample[2][ii] = sample[2][ii]
    ii = ii + 1

print("Nueva muestra " + str(ig + 1) + " después de elitismo")
for i in range(qi):
    try:
        print(str(newSample[0][i]) + " = " + str(sum(newSample[0][i])))
    except:
        pass
print("")
print("Se aplicará cruzamiento")
x = input()
```

Después se realiza el cruzamiento primeramente realizando la selección de dos padres:

```
# Se aplicará cruzamiento
while ii < qi:
    # Seleccionamos por ruleta
    # Padre 1
    random_pick = random.uniform(0, sample[2][qi - 1])

    for ic in range(qi):
        if random_pick < sample[2][ic]:
            c1 = sample[0][ic]
            print("Se elige como padre 1 el individuo " + str(ic) + "(random_pick = " + str(random_pick) + ")")
            break

    # Padre 2
    random_pick = random.uniform(0, sample[2][qi - 1])

    for ic in range(qi):
        if random_pick < sample[2][ic]:
            c2 = sample[0][ic]
            print("Se elige como padre 2 el individuo " + str(ic) + "(random_pick = " + str(random_pick) + ")")
            break
```

Lo siguiente es evaluar si los padres elegidos realizarán cruzamiento o no:

```
ii = 0
while ii < qi:
    # Se evalúa si se aplicará mutación
    random_num = random.randint(0,99)
    if random_num < pm:
        print("El cromosoma " + str(ii) + " ha activado mutación (Probabilidad: " + str(random_num) +
        print("")

        processors = [0, 1, 2, 3, 4, 5]
        random.shuffle(processors) # Se mezclarán y se hará una sumas balanceadas cíclica

        print("Tupla mezclada: " + str(processors))
        print("")

        print("Valores antes de la mutación: ")
        print(newSample[0][ii])
        print("")

        min_value = min(newSample[0][ii])
        operation = random.randint(0, 1) # Elige la operación con la que empezará la suma equilibrada
        addition = random.randint(0, min_value)

        print("Tupla " + str(processors))
        print("min_value = " + str(min_value))
        print("addition = " + str(addition))
        print("")
```

Posteriormente se harán lo que en el equipo se le denominó “sumas equilibradas”, que es el concepto de equilibrio por medio de una técnica Simplex:

```
for i in [0, 2, 4]:
    # Sumas equilibradas
    min_value = min(c1[processors[i]], c1[processors[i + 1]], c2[processors[i]], c2[processors[i + 1]])
    operation = random.randint(0, 1) # Elige la operación con la que empezará la suma equilibrada
    addition = random.randint(0, min_value)

    print("Tuplas " + str(processors[i] + 1) + " y " + str(processors[i + 1] + 1))
    print("min_value = " + str(min_value))
    print("addition = " + str(addition))
    print("")

    if operation == 0:
        c2[processors[i + 1]] = c2[processors[i + 1]] - addition
        c1[processors[i + 1]] = c1[processors[i + 1]] + addition
        c1[processors[i]] = c1[processors[i]] - addition
        c2[processors[i]] = c2[processors[i]] + addition
    else:
        c2[processors[i + 1]] = c2[processors[i + 1]] + addition
        c1[processors[i + 1]] = c1[processors[i + 1]] - addition
        c1[processors[i]] = c1[processors[i]] + addition
        c2[processors[i]] = c2[processors[i]] - addition

    print("Valores después del cruzamiento: ")
    print(c1)
    print(c2)
    print("")
```

Si no se cruzan, el procedimiento contrario es heredarlos de igual manera.

Después se realiza la mutación:

```
# Se aplica mutación
ii = 0
while ii < qi:
    # Se evalúa si se aplicará mutación
    random_num = random.randint(0,99)
    if random_num < pm:
        print("El cromosoma " + str(ii) + " ha activado mutación (Probabilidad: " + str(random_num) + " < "
        print("")

        processors = [0, 1, 2, 3, 4, 5]
        random.shuffle(processors) # Se mezclarán y se hará una sumas balanceadas cíclica

        print("Tupla mezclada: " + str(processors))
        print("")

        print("Valores antes de la mutación: ")
        print(newSample[0][ii])
        print("")

        min_value = min(newSample[0][ii])
        operation = random.randint(0, 1) # Elige la operación con la que empezará la suma equilibrada
        addition = random.randint(0, min_value)

        print("Tupla " + str(processors))
        print("min_value = " + str(min_value))
        print("addition = " + str(addition))
        print("")

        if operation == 0:
            newSample[0][ii][processors[0]] = newSample[0][ii][processors[0]] - addition
            newSample[0][ii][processors[1]] = newSample[0][ii][processors[1]] + addition
            newSample[0][ii][processors[2]] = newSample[0][ii][processors[2]] - addition
            newSample[0][ii][processors[3]] = newSample[0][ii][processors[3]] + addition
            newSample[0][ii][processors[4]] = newSample[0][ii][processors[4]] - addition
            newSample[0][ii][processors[5]] = newSample[0][ii][processors[5]] + addition
```

El procedimiento se repetirá según la cantidad de generaciones deseado.

Un ejemplo de ejecución de código es el siguiente. A continuación se muestra la población inicial al ser evaluada y ordenada:

Población 0 al ser reevaluada		
[2969361, 1539170, 1815651, 18523, 412987, 186329] = 6942021	31622037	31622037
[1865628, 1340774, 3321758, 89961, 257220, 66680] = 6942021	31562903	63184940
[2199470, 2560613, 1966710, 93170, 32732, 89326] = 6942021	31546451	94731391
[758906, 3125166, 1812820, 347267, 419893, 477969] = 6942021	30975301	125706692
[668383, 1823891, 2580073, 1260065, 74070, 535539] = 6942021	30969025	156675717
[2547676, 1679639, 1001566, 977186, 302243, 433711] = 6942021	30870443	187546160
[875519, 2158420, 3016210, 586654, 191857, 113361] = 6942021	30774515	218320675
[1845586, 3211955, 444884, 202954, 1088166, 148476] = 6942021	30756221	249076896
[1680857, 1036057, 3977078, 218624, 2748, 26657] = 6942021	30399821	279476717
[2650913, 2797244, 147921, 126160, 717987, 501796] = 6942021	30170303	309647020
[227888, 2168853, 2913690, 981996, 641130, 8464] = 6942021	30168005	339815025
[2948873, 2369384, 816626, 341442, 283551, 182145] = 6942021	30106627	369921652
[3583567, 791277, 1486148, 492479, 398246, 190304] = 6942021	29768075	399689727
[568776, 723038, 3902812, 558178, 421514, 767703] = 6942021	29752731	429442458
[3441356, 688951, 2009969, 652667, 136069, 13009] = 6942021	29732121	459174579
[3793563, 313321, 2046431, 372089, 361754, 54863] = 6942021	29588863	488763442
[2551365, 3115234, 757508, 258417, 197677, 61820] = 6942021	29575993	518339435

[2482156, 3272432, 934092, 193415, 44168, 15758] = 6942021	29530019 547869454
[985930, 972146, 1833844, 2523668, 351808, 274625] = 6942021	29366973 577236427
[2650989, 2815668, 873007, 598450, 2346, 1561] = 6942021	29295811 606532238
[4078958, 1395460, 1084937, 340042, 507, 42117] = 6942021	29082167 635614405
[223062, 664049, 2854577, 1731647, 371117, 1097569] = 6942021	28842531 664456936
[4110636, 864439, 1453124, 461013, 42878, 9931] = 6942021	28776869 693233805
[4405407, 1578050, 554754, 168294, 221645, 13871] = 6942021	28749945 721983750
[3165453, 1296076, 764776, 1425381, 279518, 10817] = 6942021	28738499 750722249
[3678398, 28387, 1106706, 736053, 1260842, 131635] = 6942021	28597099 779319348
[4490396, 1031536, 1239910, 73570, 99624, 6985] = 6942021	28579967 807899315
[2606706, 1447308, 488702, 2090652, 26763, 281890] = 6942021	28525451 836424766
[4574472, 1273115, 925421, 9432, 104218, 55363] = 6942021	28411815 864836581
[880266, 4004944, 904777, 792508, 357802, 1724] = 6942021	28325263 893161844
[4617886, 656977, 1571440, 23067, 52409, 20242] = 6942021	28324987 921486831
[445283, 939017, 4049558, 1292680, 176507, 38976] = 6942021	28106749 949593580
[202769, 3178837, 1264093, 1763174, 124357, 408791] = 6942021	28036145 977629725
[3758510, 2760947, 317397, 22715, 40287, 42165] = 6942021	28027703 1005657428
[4841075, 1390739, 295531, 202586, 140523, 71567] = 6942021	27832845 1033490273
[4935908, 540620, 825644, 165224, 459982, 14643] = 6942021	27688943 1061179216
[3679508, 3014407, 108490, 35119, 12794, 91703] = 6942021	27678787 1088858003
[4331043, 1039278, 725498, 792389, 49251, 4562] = 6942021	27673303 1116531306
[88008, 5093751, 1741794, 1174, 7228, 10066] = 6942021	27373257 1143904563
[5349748, 144004, 1408908, 28851, 670, 9840] = 6942021	26861263 1170765826
[463477, 4867792, 253975, 698397, 257604, 400776] = 6942021	26787789 1197553615
[1322455, 5121094, 563, 480577, 3847, 13485] = 6942021	26716825 1224270440

Observemos que todas las tuplas suman 6,942,021. El número siguiente es su valor como función objetivo y el siguiente se trata del valor acumulado para realizar elitismo, que de hecho, al aplicar elitismo en esta población se obtiene lo siguiente:

```
Nueva muestra 1 vacía
[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]

Se aplicará elitismo

Nueva muestra 1 después de elitismo
[2969361, 1539170, 1815651, 18523, 412987, 186329] = 6942021
[1865628, 1340774, 3321758, 89961, 257220, 66680] = 6942021
[2199470, 2560613, 1966710, 93170, 32732, 89326] = 6942021
[758906, 3125166, 1812820, 347267, 419893, 477969] = 6942021
[668383, 1823891, 2580073, 1260065, 74070, 535539] = 6942021
```

Posteriormente en el cruzamiento, observemos cómo el algoritmo avisa que no se ha activado el cruzamiento:

```
Se elige como padre 1 el individuo 3(random_pick = 110888477.71224487)
Se elige como padre 2 el individuo 17(random_pick = 542848757.1963074)
Los cromosomas 49 y 50 no han activado cruzamiento (Probabilidad: !82 < 75!)
```

Asimismo, se aprecia a continuación la activación del cruzamiento:

```
Se elige como padre 1 el individuo 53(random_pick = 1515027406.0065634)
Se elige como padre 2 el individuo 43(random_pick = 1270469611.342023)
Los cromosomas 53 y 54 han activado cruzamiento (Probabilidad: 54 < 75)
```

Cuando esto sucede, se reparten en pares los valores de las tuplas para mezclarse como se muestra a continuación:

```
Tupla mezclada: [4, 2, 3, 5, 0, 1]

Valores antes del cruzamiento:
[445283, 939017, 4049558, 1292680, 176507, 38976]
[4394701, 1559395, 680532, 186949, 95867, 24577]

Tuplas 5 y 3
min_value = 95867
addition = 10059

Tuplas 4 y 6
min_value = 24577
addition = 13195

Tuplas 1 y 2
min_value = 445283
addition = 344965

Valores después del cruzamiento:
[100318, 1283982, 4039499, 1279485, 186566, 52171]
[4739666, 1214430, 690591, 200144, 85808, 11382]
```


Generando una población de individuos ya cruzados:

```
[2910289, 2355836, 817043, 345955, 322135, 190763] = 6942021
[1321644, 5129616, 50, 481090, 4658, 4963] = 6942021
[668843, 1826377, 2574342, 1270469, 71584, 530406] = 6942021
[668843, 1826377, 2574342, 1270469, 71584, 530406] = 6942021
[907762, 3985833, 877281, 793891, 376913, 341] = 6942021
[2446418, 3264221, 961149, 219354, 25496, 25383] = 6942021
[668843, 1826377, 2574342, 1270469, 71584, 530406] = 6942021
[6311967, 610862, 2571, 2131, 1055, 13435] = 6942021
[2456600, 3354140, 760853, 19511, 292442, 58475] = 6942021
[3536121, 450045, 2006624, 891573, 41304, 16354] = 6942021
[5559159, 725333, 625816, 3467, 11910, 16336] = 6942021
[5422971, 70781, 1408387, 28441, 1191, 10250] = 6942021
[6295358, 66474, 359857, 192163, 27896, 273] = 6942021
[3733768, 373116, 2115273, 303247, 361854, 54763] = 6942021
[2910289, 2355836, 817043, 345955, 322135, 190763] = 6942021
[3839643, 2758954, 275755, 14468, 44311, 8890] = 6942021
[3532215, 2699599, 297181, 225772, 138873, 48381] = 6942021
[4987467, 1698043, 97250, 19437, 15345, 124479] = 6942021
[5801053, 302214, 475984, 353544, 9219, 7] = 6942021
[2456600, 3354140, 760853, 19511, 292442, 58475] = 6942021
[3472949, 1287102, 457280, 1414743, 288492, 21455] = 6942021
[5280471, 121154, 731806, 368189, 128, 440273] = 6942021
[4739666, 1214430, 690591, 200144, 85808, 11382] = 6942021
[3689104, 47042, 980928, 717398, 1386620, 120929] = 6942021
[4987467, 1698043, 97250, 19437, 15345, 124479] = 6942021
[5280471, 121154, 731806, 368189, 128, 440273] = 6942021
[5752105, 1131588, 35261, 17545, 910, 4612] = 6942021
[2446418, 3264221, 961149, 219354, 25496, 25383] = 6942021
[6295358, 66474, 359857, 192163, 27896, 273] = 6942021
[784261, 2289257, 2885373, 631150, 213997, 137983] = 6942021
[6533076, 329256, 7074, 36297, 36281, 37] = 6942021
[1844933, 3214720, 442119, 187687, 1103433, 149129] = 6942021
[3839643, 2758954, 275755, 14468, 44311, 8890] = 6942021
[5801053, 302214, 475984, 353544, 9219, 7] = 6942021
[6295358, 66474, 359857, 192163, 27896, 273] = 6942021
[3733768, 373116, 2115273, 303247, 361854, 54763] = 6942021
[755218, 659469, 3578184, 581861, 408772, 958517] = 6942021
[6047208, 22261, 844701, 24559, 3287, 5] = 6942021
[758906, 3125166, 1812820, 347267, 419893, 477969] = 6942021
[2446418, 3264221, 961149, 219354, 25496, 25383] = 6942021
[5926199, 527918, 274824, 53085, 158711, 1284] = 6942021
[755218, 659469, 3578184, 581861, 408772, 958517] = 6942021
```

Para la mutación pasa algo similar, pues se puede apreciar el caso de cuando no se activa la mutación como cuando sí se activa a continuación:

Cuando se activa la mutación, el cromosoma se desordena y un valor será intercambiado entre todos los números de la tupla:

```
Tupla mezclada: [1, 3, 5, 0, 4, 2]

Valores antes de la mutación:
[947291, 467396, 3386111, 389788, 600845, 1150590]

Tupla [1, 3, 5, 0, 4, 2]
min_value = 389788
addition = 33411

Valores después de la mutación:
[980702, 433985, 3419522, 423199, 567434, 1117179]
```

Una vez que terminan estos procedimientos de ejecutarse, la nueva generación cambia de estructura para repetir y volver a heredar las veces que se hayan indicado.

El programa es muy inestable en sus ejecuciones, sin embargo en los mejores retornos se realizaron promedios y cálculos de estos resultados, obteniendo resultados que más adelante se expondrán.

Procedimiento de Lemantización

En contraste con la derivación, la lematización es mucho más poderosa, va más allá de la reducción de palabras y considera el vocabulario completo de un idioma para aplicar un análisis morfológico a las palabras, con el objetivo de eliminar solo las terminaciones flexivas y devolver la forma base o de diccionario de una palabra, que se conoce como lema.

En sí, de la palabra o frase original sacamos la función de la palabra raíz (lema), para agilizar el proceso se recomienda primero hacer una extracción de palabras centrales y reunir las, después hacer una conversión del tiempo y pasar todo a presente, y para terminar trasladamos de plural a singular todo.

Para realizar la lematización utilizamos el enfoque “WordNet(con etiqueta POS)”, Wordnet es una base de datos léxica disponible públicamente en más de 200 idiomas que proporciona relaciones semánticas entre sus palabras.

Está presente en la biblioteca nltk en Python, Wordnet vincula las palabras en relaciones semánticas por ejemplo sinónimos los cuales agrupa en forma de synsets (un grupo de elementos de datos que son semánticamente equivalentes).

Para utilizarlo se debe de descargar el paquete de nltk, posteriormente hay que descargar Wordnet desde nltk

```
import nltk
from nltk.stem import WordNetLemmatizer
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')
nltk.download('punkt')
from nltk.corpus import wordnet
```

Continuamos con el programa inicializando el lematizador, importando pandas y empezando a leer nuestra base de datos de tweets “Team10 Scrapping (Primera Limpieza).csv”.

```
8 lemmatizer = WordNetLemmatizer()
9 import pandas as pd
10 datos=pd.read_csv('Team10 Scrapping (Primera Limpieza).csv')
```

Para evitar que maneje todo como sustantivos utilizamos la etiquetas POS(Part of Speech), la cual sirve para definir el type de una palabra para ver si es un adjetivo, verbo, sustantivo o adverbio.

```
12 def pos_tagger(nltk_tag):
13     if nltk_tag.startswith('J'):
14         return wordnet.ADJ
15     elif nltk_tag.startswith('V'):
16         return wordnet.VERB
17     elif nltk_tag.startswith('N'):
18         return wordnet.NOUN
19     elif nltk_tag.startswith('R'):
20         return wordnet.ADV
21     else:
22         return None
```

Ahora seleccionamos y recorremos la columna que contiene los tweets, para extraerlos y guardarlos en “sentence” para posteriormente aplicar lo que es la lematización.

```
24 archivo="Lemantizacion.csv"
25 csv=open(archivo,"w")
26 for a in range(len(datos.iloc[:,7])):
27
28     sentence = datos.iloc[a,7]
29
30     pos_tagged = nltk.pos_tag(nltk.word_tokenize(sentence))
31
32     print(pos_tagged)
```

En esta parte es donde se hace la clasificación y se exporta en el nuevo archivo de Excel.

```
35     wordnet_tagged = list(map(lambda x:(x[0], pos_tagger(x[1])),
36                               pos_tagged))
37     print(wordnet_tagged)
38     lemmatized_sentence = []
39
40     for word, tag in wordnet_tagged:
41         if tag is None:
42
43             lemmatized_sentence.append(word)
44         else:
45
46             lemmatized_sentence.append(lemmatizer.lemmatize(word, tag))
47     lemmatized_sentence = " ".join(lemmatized_sentence)
48     lemmatized_sentence = lemmatized_sentence + "\n"
49     try:
50         csv.write(lemmatized_sentence)
51     except ValueError:
52         print("")
```


Ejemplo de ejecución:

```
[nltk_data] Downloading package wordnet to
[nltk_data] C:\Users\zente\AppData\Roaming\nltk_data...
[nltk_data] Package wordnet is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data] C:\Users\zente\AppData\Roaming\nltk_data...
[nltk_data] Package averaged_perceptron_tagger is already up-to-
[nltk_data] date!
[nltk_data] Downloading package punkt to
[nltk_data] C:\Users\zente\AppData\Roaming\nltk_data...
[nltk_data] Package punkt is already up-to-date!
Traceback (most recent call last):
```

Primero hace las descargas.

```
[('was', 'VBD'), ('Death', 'NNP'), ('And', 'CC'), ('the', 'DT'), ('word', 'NN'), ('was', 'VBD'), ('nigger', 'JJR'), ('And', 'CC'), ('the', 'DT'), ('word', 'NN'), ('was', 'VBD'), ('death', 'NN'), ('to', 'TO'), ('all', 'DT'), ('niggers', 'NNS'), ('And', 'CC'), ('the...', 'NN')]
[(('@', 'n'), ('@', 'n'), ('jatella', 'n'), (':', None), ('"', 'n'), ('In', None), ('the', None), ('beginning', 'n'), ('was', 'v'), ('the', None), ('word', 'n'), ('And', None), ('the', None), ('word', 'n'), ('was', 'v'), ('Death', 'n'), ('And', None), ('the', None), ('word', 'n'), ('was', 'v'), ('nigger', 'a'), ('And', None), ('the', None), ('word', 'n'), ('was', 'v'), ('death', 'n'), ('to', None), ('all', None), ('niggers', 'n'), ('And', None), ('the...', 'n')]
```

Después descompone todo por palabras y las analiza.

@ Mattimatikus @ Urbanistbhd @ DracheOhneLP Ich nenne andere nigger und werde nicht gesperrt
I be go to sue Delta chinese chink for not honor NSA who say I can fly . Life or death ! God famed Democrat leave Obama nigger human trafficking ! Law suit ! Europe already
Gon na take everything from Faye dad not to call Teddy a nigger
@ Big0047 Wakikaa kafanaa naaa ma nigger
@ matthew47138586 @ Oscaranking2 @ _littlehuman_ This nigger about to be a father http : //t.co/ydDVoNfwgA
@ akintonm especially si more than spelling .
@ CashWunner nigger mode well
If he can ' t help market us then fuck him market himself Typical nigger

Y los resultados son exportados a un nuevo archivo de Excel llamado "Lemantización.csv"

Resultados

El algoritmo genético estaba diseñado para poder configurar la cantidad de tweets sin realizar grandes cambios al código tan sólo con cambiar el valor de la variable `total_tweets`.

Debido a que el algoritmo genético presenta inestabilidades muy corregibles en el cruzamiento y mutación, se ejecutaron múltiples veces y los siguientes fueron los resultados de promedio que arrojó el algoritmo genético:

Tweets totales	Asignados a Melissa	Asignados a Joel	Asignados a Óscar	Asignados a Dariana	Asignados a Fer	Asignados a Hiram
10,000	2,315	1,791	3,209	203	1,234	1,248
20,000	4,992	3,223	6,248	573	2,851	2,113
30,000	7,479	4,841	9,477	758	4,320	3,125
40,000	9,946	6,479	12,638	1,008	5,742	4,187
50,000	12,689	7,843	15,751	1,305	7,364	5,048
60,000	15,363	9,275	19,037	1,431	8,809	6,085
70,000	17,678	11,067	22,198	1,681	10,116	7,260
80,000	20,211	12,640	25,030	2,261	11,840	8,018
90,000	22,770	14,187	28,425	2,277	13,227	9,114
100,000	25,260	15,804	31,722	2,392	14,667	10,155
200,000	50,639	31,489	63,143	5,084	29,186	20,459
300,000	75,594	47,598	94,727	7,613	43,845	30,623
400,000	100,890	63,365	126,235	10,219	58,713	40,578
500,000	126,154	79,165	157,647	12,920	72,799	51,315
600,000	151,388	94,995	189,144	15,535	87,651	61,287
700,000	176,708	110,738	220,767	18,028	102,024	71,735
800,000	201,880	126,630	252,286	20,622	117,026	81,556
900,000	227,448	142,126	283,753	23,267	131,309	92,097
1,000,000	252,667	157,971	315,164	25,970	146,190	102,038
2,000,000	505,063	316,212	630,578	51,690	292,122	204,335
3,000,000	757,480	474,432	945,761	77,645	438,003	306,679

4,000,000	1,009,975	632,475	1,260,855	103,782	584,042	408,871
5,000,000	1,262,458	790,730	1,576,360	129,311	730,179	510,962
6,000,000	1,515,276	948,548	1,891,532	155,274	876,172	613,198
6,942,021	1,752,929	1,097,724	2,188,458	179,704	1,013,759	709,447

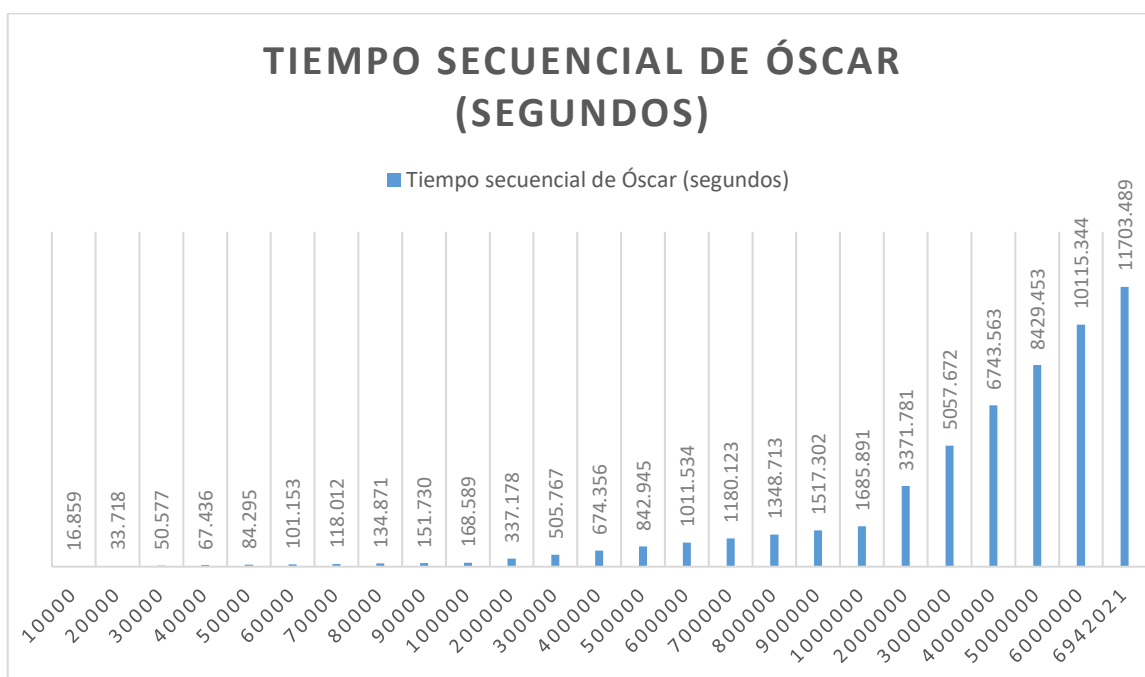
Resultados de tiempo secuencial

Para el reporte de tiempo secuencial se usará el mejor de los tiempos secuenciales, que en este caso será el tiempo establecido por Óscar (véase anexo 8 para un mayor reporte de tiempos por integrante) que es el siguiente:

Tweets	Tiempo de Óscar (segundos)
10,000	16.859
20,000	33.718
30,000	50.577
40,000	67.436
50,000	84.295
60,000	101.153
70,000	118.012
80,000	134.871
90,000	151.730
100,000	168.589
200,000	337.178
300,000	505.767
400,000	674.356
500,000	842.945
600,000	1011.534
700,000	1180.123
800,000	1348.713
900,000	1517.302

1,000,000	1685.891
2,000,000	3371.781
3,000,000	5057.672
4,000,000	6743.563
5,000,000	8429.453
6,000,000	10115.344
6,942,021	11703.489

Obsérvese la siguiente gráfica

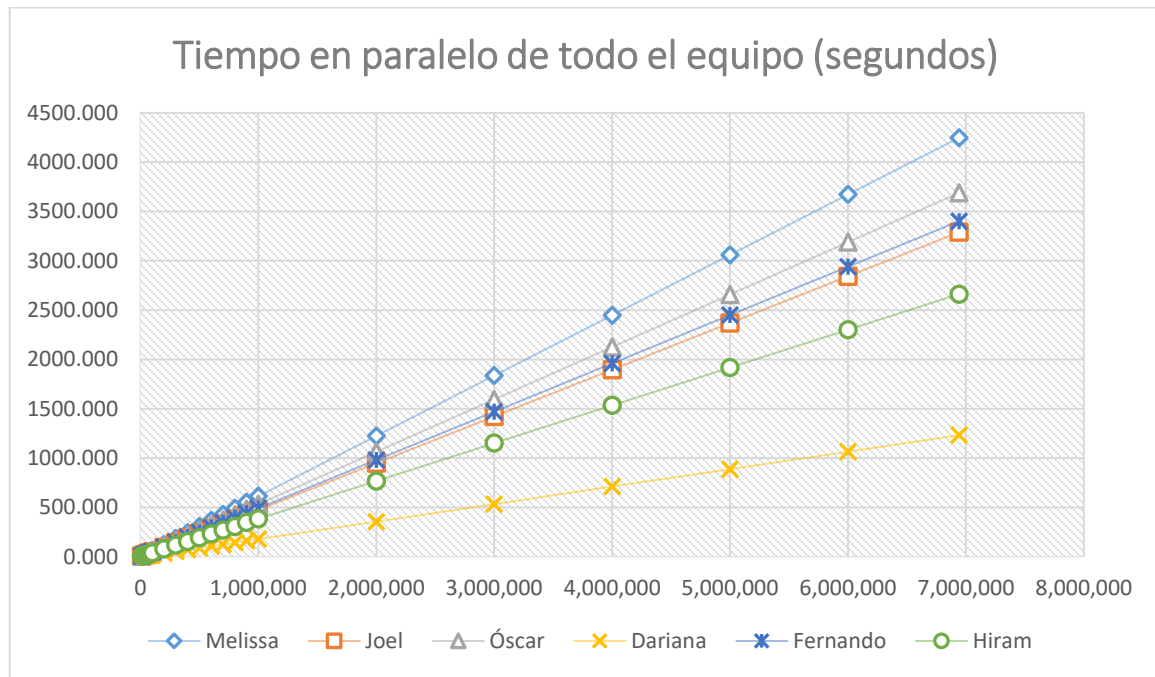


En la gráfica se muestra el crecimiento del tiempo según la cantidad de tweets asignada a nivel secuencial para Óscar. Entre otros datos que pueden observarse en el anexo 8, el PC de Óscar terminaría la tarea de procesar todos los tweets extraídos en tres horas y 15 minutos aproximadamente, si se trata de realizar un procedimiento secuencial, sin embargo, otros equipos como el PC de Melissa tomarían 4 horas y media en realizar toda la tarea secuencial; el PC de Hiram tardaría 7 horas y 10 minutos por sí solo. El peor de los casos sería en el

PC de Dariana, que tardaría 13 horas y 10 minutos terminar de procesar los más de seis millones de tweets que se almacenaron.

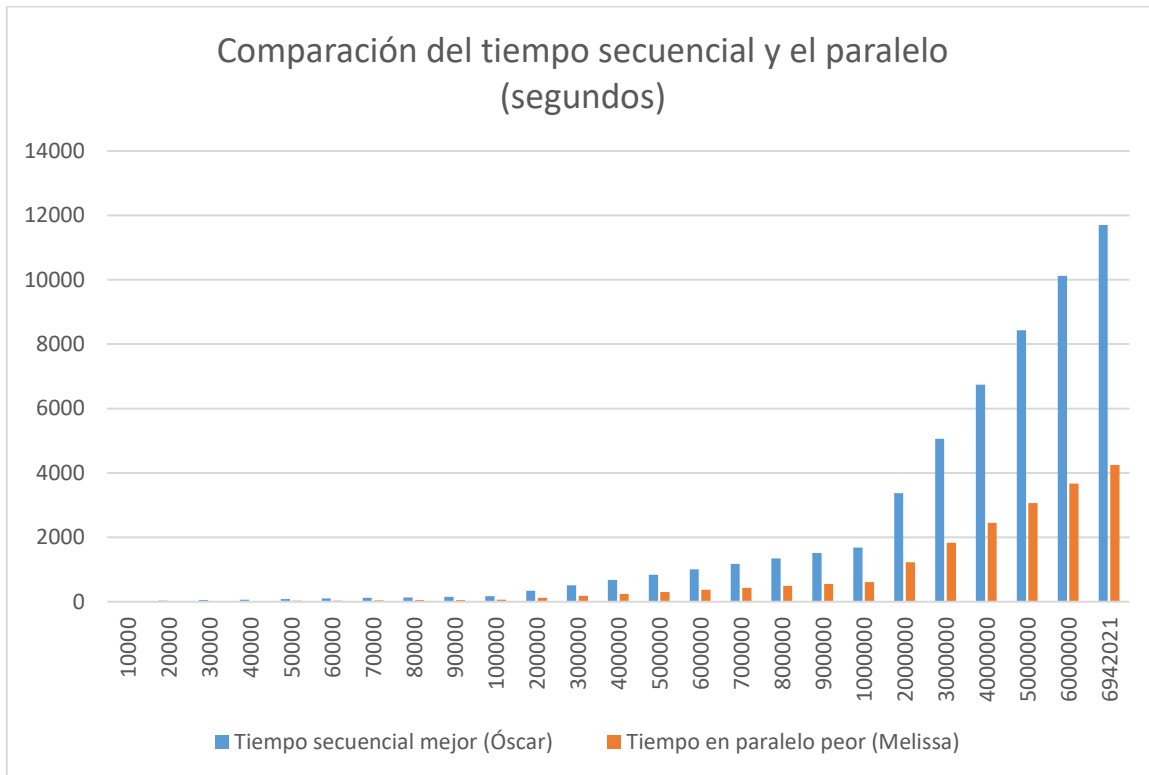
Resultados de tiempo paralelo

Para el reporte de tiempo primeramente se tomarán todos los valores (véase anexo 9 para un mayor reporte de tiempos por integrante) que se ve reflejado en el siguiente gráfico:



La interpretación que el equipo le dio a los resultados del tiempo paralelo es que pudieron ocurrir algunos errores a lo largo del procedimiento, por ejemplo, la obtención del tiempo de complejidad lineal, que probablemente no era un tiempo escalable a tanto tiempo; por ello, el tiempo muy largo que Dariana reportó, causó que el algoritmo genético, al momento de escalar tanto los tiempos, le asignara tan pocos tweets al elevar la cantidad de tweets. También puede ser causa del algoritmo genético, aunque realiza buen trabajo para reducir los tiempos, – como más adelante se expondrá – es claro que un algoritmo genético encuentra buenas soluciones aunque no necesariamente las mejores.

Puede observarse que debido a la asignación de tiempos y del algoritmo, los tiempos que peor escalabilidad tuvieron fueron los de Melissa y Óscar, especialmente Melissa que para procesar su parte en paralelo del total de tweets, tomó una hora y 10 minutos, siendo este el peor de los tiempos para esta categoría, pero por otro lado, se mejoró en un 285% el tiempo base del procesamiento secuencial.



Puede concluirse que, como anteriormente se mencionó, se mejoró el tiempo en un 285%, es decir, que el algoritmo genético consiguió que la mejoría en tiempos por distribución de carga realmente estuviera funcionando.

Conclusiones

Andrea Melissa Almeida Ortega:

Como se pudo observar ya en la práctica de algoritmo genético simple enfocada en nuestro proyecto, nos damos cuenta que la optimización es algo esencial para el software y no es simplemente priorizar que realice las acciones más rápido, si no de entender lo que está haciendo el programa para poder repartir las distintas tareas y acciones para que se puedan ejecutar de la manera más eficaz posible.

Joel Alejandro Espinoza Sánchez:

La implementación y uso del algoritmo genético fue muy importante para el proyecto de la materia y más importante con un planificador de carga y el procedimiento de lemantización. Con este programa se observó que la utilidad de un algoritmo genético puede ser muy importante en un procedimiento de optimización, y no necesariamente un algoritmo genético, pero para este proyecto fue fundamental.

Óscar Alonso Flores Fernández:

Al poder elaborar un algoritmo genético basándonos en lo que hemos estado elaborando podemos tener un esquema de trabajo mucho más directo y eficiente para poder llevar a cabo las distintas tareas, buscando siempre el poder cumplir con lo que el AG nos dicta y así optimizar el trabajo, la división de tareas, etc.

Dariana Gómez Garza:

En la elaboración de esta práctica implementamos para otro fin el algoritmo genético y otras herramientas útiles que nos ayudaron a observar las funciones de nuestro proyecto que tenemos en marcha de esta materia y lo útil que es para mejorar otros algoritmos. Las clases que tuvimos nos dieron una idea mejor de cómo realizar esta investigación. También nos dimos cuenta de cómo podríamos aplicar esto y que fuera funcional al mismo tiempo para nuestro proyecto en curso.

Fernando Francisco González Arenas:

Esta distribución de tweets permite utilidades en las que podemos agilizar el trabajo de las computadoras fácilmente. En este trabajo podemos observar cómo nuestras computadoras a partir de un programa en Python y un planificador de carga funcional podemos tener nuestras computadoras trabajando de manera acorde a su comportamiento.

Hiram Efraín Orocio García:

Un algoritmo genético es útil al momento de buscar soluciones por medio de exploración, así que aprender a implementar uno así sea en una aplicación básica, en un futuro lo podríamos implementar en problemas mayores que serán de gran ayuda.

Referencias bibliográficas

- C. Darwin (1859). *On the Origin of Species by Means of Natural Selection*, Murray, London.
- C. Reeves (1993). *Modern Heuristic Techniques for Combinatorial Problems*, Blackwell Scientific Publications.
- D. E. Goldberg, Jr. R. Lingle (1985). *Alleles, loci and the traveling salesman problem*, en Proceedings of the First International Conference on Genetic Algorithms and Their Applications, 154-159.
- Z. Michalewicz (1992). *Genetic Algorithms + Data Structures = Evolution Programs*, SpringerVerlag, Berlin Heidelberg.
- Brad Miller, David Ranum. (2006). *Problem solving with algorithms and data structures using python*, Fraklin, Beedle & Associates.
- Albert Y. Zomaya, Senior Member, IEEE, and Yee-Hwei Teh. (SEPTEMBER 2001). *Observations on Using Genetic Algorithms for Dynamic Load-Balancing*. IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, 12, 13. Octubre 10, 2021.
- Lima, A. (2021). Python: enfoques de lematización con ejemplos – Acervo Lima. Retrieved 12 October 2021, from <https://es.acervolima.com/2021/02/09/python-enfoques-de-lematizacion-con-ejemplos/>
- (2021). Retrieved 12 October 2021, from <https://www.youtube.com/watch?v=IhC01D6CbVU&list=PLgHCrivozIb0ULMKfJVV-rFdRG2OeEgfq&index=5>
- Análisis de sentimiento. Qué es y cómo realizarlo | QuestionPro. (2019). Retrieved 12 October 2021, from <https://www.questionpro.com/blog/es/herramienta-de-analisis-de-sentimientos/#:~:text=El%20an%C3%A1lisis%20de%20sentimiento%20es,sus%20actitudes%2C%20emociones%20y%20opiniones.>

- Análisis de sentimiento, ¿qué es, cómo funciona y para qué sirve?. (2017). Retrieved 12 October 2021, from <https://itelligent.es/es/analisis-de-sentimiento/>
- Lematización - Wikipedia, la enciclopedia libre. (2021). Retrieved 12 October 2021, from <https://es.wikipedia.org/wiki/Lematizaci%C3%B3n#:~:text=La%20lematizaci%C3%B3n%20es%20un%20proceso,flexionadas%20de%20una%20misma%20palabra.>
- (2021). Retrieved 12 October 2021, from <https://users.dcc.uchile.cl/~abassi/ecos/lema.html>
- Procesamiento del lenguaje natural - EcuRed. (2021). Retrieved 12 October 2021, from https://www.ecured.cu/Procesamiento_del_lenguaje_natural#:~:text=El%20%22Procesamiento%20del%20Language%20Natural,comprensi%C3%B3n%20autom%C3%A1tica%20del%20lenguaje%20natural.
- Procesamiento del lenguaje natural ¿qué es? - IIC. (2017). Retrieved 12 October 2021, from <https://www.iic.uam.es/inteligencia/que-es-procesamiento-del-lenguaje-natural/>
- Procesamiento del Lenguaje Natural con Machine Learning. (2021). Retrieved 12 October 2021, from <https://www.encora.com/es/blog/procesamiento-del-lenguaje-natural-con-machine-learning>

Anexos

Anexo 1: Código de server.py en Python.

```
### Conexión
import socket

s = socket.socket()
host = socket.gethostname()
port = 8080
s.bind((host,port))
s.listen(1)
print(host)
print("Esperando conexiones...")
conn, addr = s.accept()
print(addr + " se ha conectado")

### Transfiriendo archivos
filename = "Control.txt"
file = open(filename, 'rb')
file_data = file.read(1024)
conn.send(file_data)
print("Archivo enviado")
```


Anexo 2: Código de client.py en Python.

```
### Conexión
import socket

s = socket.socket()
print("Ingresa la dirección del host")
host = input()
port = 8080
s.connect((host,port))
print("Conectado")

### Transfiriendo archivos
filename = "Control.txt"
file = open(filename, 'wb')
file_data = s.recv(1024)
file.write(file_data)
file.close()
print("Recibido")
```

Anexo 3: Archivo de texto plano Control-Servidor.txt en su estado inicial.

Estado: 0
Cliente Melissa: 0
Cliente Joel: 0
Cliente Oscar: 0
Cliente Dariana: 0
Cliente Fernando: 0
Cliente Hiram: 0

MELISSA
Procesador (ultimo contacto): 0
Historial del procesador:

Tweets enviados a procesar: 0
Tweets ya procesados: 0
Tiempo: 0

JOEL
Procesador (ultimo contacto): 0
Historial del procesador:

Tweets enviados a procesar: 0
Tweets ya procesados: 0
Tiempo: 0

OSCAR
Procesador (ultimo contacto): 0
Historial del procesador:

Tweets enviados a procesar: 0
Tweets ya procesados: 0
Tiempo: 0

DARIANA
Procesador (ultimo contacto): 0
Historial del procesador:

Tweets enviados a procesar: 0
Tweets ya procesados: 0
Tiempo: 0

FERNANDO
Procesador (ultimo contacto): 0
Historial del procesador:

Tweets enviados a procesar: 0
Tweets ya procesados: 0
Tiempo: 0

HIRAM
Procesador (ultimo contacto): 0
Historial del procesador:

Tweets enviados a procesar: 0
Tweets ya procesados: 0
Tiempo: 0

Anexo 4: Archivo de texto plano Emision-Nombre.txt en su estado inicial (caso de ejemplo: Melissa).

MELISSA

Procesador: 0

Tweets procesados: 0

Tiempo: 0

Anexo 5: Código con complejidad computacional lineal estándar en Python.

```
import time

i = 0

inicio = time.time()
while i < 1000000:
    i = i + 0.01
final = time.time()
intervalo = final - inicio

print("Tiempo de ejecución: " + str(intervalo))
```

Anexo 6: Evidencias de los resultados individuales al ejecutar el código del anexo 5.

Evidencia del tiempo de Melissa:

```
Tiempo de ejecución: 7.725000858306885

In [2]: runfile('C:/Users/mel_a/Documents/I.C.I/Python/untitled0.py',
wdir='C:/Users/mel_a/Documents/I.C.I/Python')
Tiempo de ejecución: 8.286704778671265

In [3]: runfile('C:/Users/mel_a/Documents/I.C.I/Python/untitled0.py',
wdir='C:/Users/mel_a/Documents/I.C.I/Python')
Tiempo de ejecución: 8.389048099517822

In [4]: runfile('C:/Users/mel_a/Documents/I.C.I/Python/untitled0.py',
wdir='C:/Users/mel_a/Documents/I.C.I/Python')
Tiempo de ejecución: 8.981545448303223

In [5]: runfile('C:/Users/mel_a/Documents/I.C.I/Python/untitled0.py',
wdir='C:/Users/mel_a/Documents/I.C.I/Python')
Tiempo de ejecución: 8.123982906341553

In [6]: runfile('C:/Users/mel_a/Documents/I.C.I/Python/untitled0.py',
wdir='C:/Users/mel_a/Documents/I.C.I/Python')
Tiempo de ejecución: 8.439027309417725

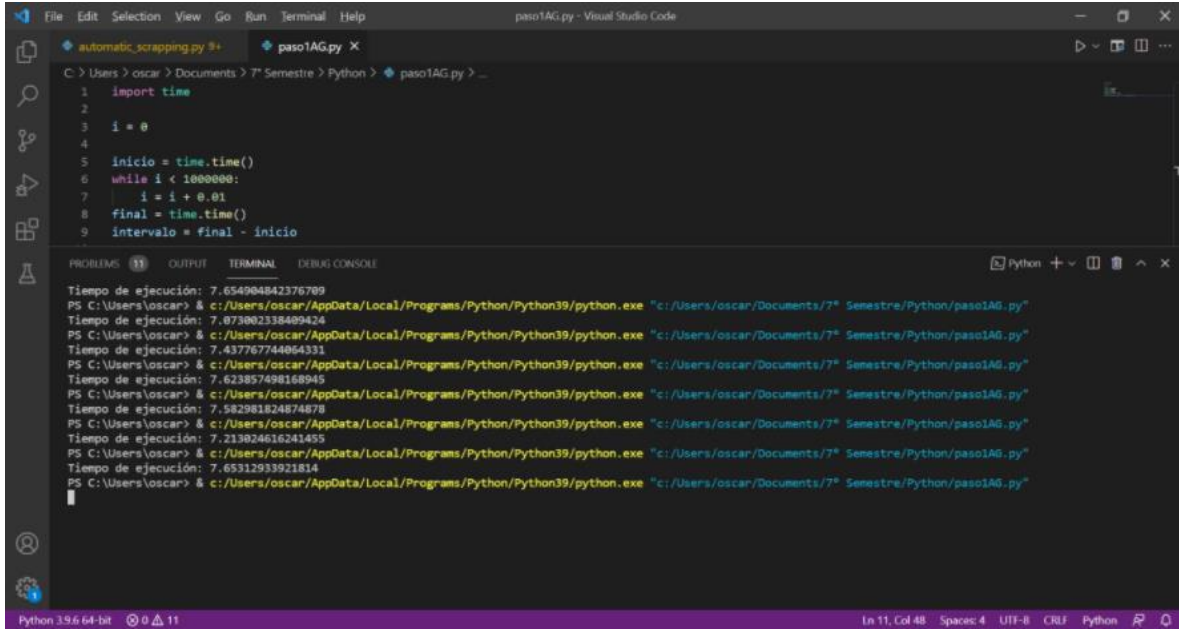
In [7]: runfile('C:/Users/mel_a/Documents/I.C.I/Python/untitled0.py',
wdir='C:/Users/mel_a/Documents/I.C.I/Python')
Tiempo de ejecución: 7.987653017044067

In [8]: runfile('C:/Users/mel_a/Documents/I.C.I/Python/untitled0.py',
wdir='C:/Users/mel_a/Documents/I.C.I/Python')
Tiempo de ejecución: 8.185094594955444

In [9]: runfile('C:/Users/mel_a/Documents/I.C.I/Python/untitled0.py',
wdir='C:/Users/mel_a/Documents/I.C.I/Python')
Tiempo de ejecución: 9.21445369720459

In [10]: runfile('C:/Users/mel_a/Documents/I.C.I/Python/untitled0.py',
wdir='C:/Users/mel_a/Documents/I.C.I/Python')
Tiempo de ejecución: 8.52103066444397
```

Evidencia del tiempo de Óscar:



```
File Edit Selection View Go Run Terminal Help
paso1AG.py - Visual Studio Code

automatic_scraping.py 9+
paso1AG.py X

C:\Users\oscar\Documents\7º Semestre\Python> python paso1AG.py
1 import time
2
3 i = 0
4
5 inicio = time.time()
6 while i < 1000000:
7     i = i + 0.01
8     final = time.time()
9     intervalo = final - inicio

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Python Python 3.9.6 64-bit

Tiempo de ejecución: 7.654904842376709
PS C:\Users\oscar> & c:/Users/oscar/AppData/Local/Programs/Python/Python39/python.exe "c:/Users/oscar/Documents/7º Semestre/Python/paso1AG.py"
Tiempo de ejecución: 7.073002338409424
PS C:\Users\oscar> & c:/Users/oscar/AppData/Local/Programs/Python/Python39/python.exe "c:/Users/oscar/Documents/7º Semestre/Python/paso1AG.py"
Tiempo de ejecución: 7.437767744064331
PS C:\Users\oscar> & c:/Users/oscar/AppData/Local/Programs/Python/Python39/python.exe "c:/Users/oscar/Documents/7º Semestre/Python/paso1AG.py"
Tiempo de ejecución: 7.623857498168945
PS C:\Users\oscar> & c:/Users/oscar/AppData/Local/Programs/Python/Python39/python.exe "c:/Users/oscar/Documents/7º Semestre/Python/paso1AG.py"
Tiempo de ejecución: 7.582981824874878
PS C:\Users\oscar> & c:/Users/oscar/AppData/Local/Programs/Python/Python39/python.exe "c:/Users/oscar/Documents/7º Semestre/Python/paso1AG.py"
Tiempo de ejecución: 7.213024616241455
PS C:\Users\oscar> & c:/Users/oscar/AppData/Local/Programs/Python/Python39/python.exe "c:/Users/oscar/Documents/7º Semestre/Python/paso1AG.py"
Tiempo de ejecución: 7.65312933921814
PS C:\Users\oscar> & c:/Users/oscar/AppData/Local/Programs/Python/Python39/python.exe "c:/Users/oscar/Documents/7º Semestre/Python/paso1AG.py"
```

Evidencia del tiempo de Dariana:

```
PS C:\Users\DARIANA\Desktop> c:: cd 'c:\Users\DARIANA\Desktop'; & 'C:\Python39\python.exe' 'c:\Users\DARIANA\.vscode\extensions
Tiempo de ejecución: 90.12797212600708
PS C:\Users\DARIANA\Desktop> c:: cd 'c:\Users\DARIANA\Desktop'; & 'C:\Python39\python.exe' 'c:\Users\DARIANA\.vscode\extensions
\ms-python.python-2021.10.1317843341\pythonFiles\lib\python\debugpy\launcher' '49857' '--' 'c:\Users\DARIANA\Desktop\prueba.py'

Tiempo de ejecución: 89.6790201663971
PS C:\Users\DARIANA\Desktop> c:: cd 'c:\Users\DARIANA\Desktop'; & 'C:\Python39\python.exe' 'c:\Users\DARIANA\.vscode\extensions
\ms-python.python-2021.10.1317843341\pythonFiles\lib\python\debugpy\launcher' '49862' '--' 'c:\Users\DARIANA\Desktop\prueba.py'

Tiempo de ejecución: 90.48065257072449
PS C:\Users\DARIANA\Desktop> c:: cd 'c:\Users\DARIANA\Desktop'; & 'C:\Python39\python.exe' 'c:\Users\DARIANA\.vscode\extensions
\ms-python.python-2021.10.1317843341\pythonFiles\lib\python\debugpy\launcher' '49867' '--' 'c:\Users\DARIANA\Desktop\prueba.py'

Tiempo de ejecución: 93.18430972099304
PS C:\Users\DARIANA\Desktop> c:: cd 'c:\Users\DARIANA\Desktop'; & 'C:\Python39\python.exe' 'c:\Users\DARIANA\.vscode\extensions
\ms-python.python-2021.10.1317843341\pythonFiles\lib\python\debugpy\launcher' '49872' '--' 'c:\Users\DARIANA\Desktop\prueba.py'

Tiempo de ejecución: 90.65469336509705
```

Evidencia del tiempo de Hiram:

```
== RESTART: C:/Users/zente/AppData/Local,
Tiempo de ejecución: 22.436562538146973
>>>
== RESTART: C:/Users/zente/AppData/Local,
Tiempo de ejecución: 23.764434337615967
>>>
== RESTART: C:/Users/zente/AppData/Local,
Tiempo de ejecución: 20.85911536216736
>>>
== RESTART: C:/Users/zente/AppData/Local,
Tiempo de ejecución: 23.063973665237427
>>>
== RESTART: C:/Users/zente/AppData/Local,
Tiempo de ejecución: 25.14137578010559
>>>
== RESTART: C:/Users/zente/AppData/Local,
Tiempo de ejecución: 23.372560024261475
>>>
== RESTART: C:/Users/zente/AppData/Local,
Tiempo de ejecución: 22.626899003982544
>>>
== RESTART: C:/Users/zente/AppData/Local,
Tiempo de ejecución: 22.254277229309082
>>>
== RESTART: C:/Users/zente/AppData/Local,
Tiempo de ejecución: 23.449257373809814
>>>
== RESTART: C:/Users/zente/AppData/Local,
Tiempo de ejecución: 23.12931776046753
>>> |
```


Anexo 7: Código del algoritmo genético en Python.

```
### Portada
'''
                Centro de Ciencias Básicas
    Departamento de Ciencias de la Computación
                Metaheurísticas I
                7º "A"

                Segunda Evaluación Parcial

                Profesor: Francisco Javier Luna Rosas

                Alumnos:
                    Almeida Ortega Andrea Melissa
                    Espinoza Sánchez Joel Alejandro
                    Flores Fernández Óscar Alonso
                    Gómez Garza Dariana
                    González Arenas Fernando Francisco
                    Orocio García Hiram Efraín

Fecha de Entrega: Aguascalientes, Ags., 31 de octubre de 2021
'''

### Descripción

### Importación de librerías
import numpy as np
import random

### Funciones de apoyo
# Función FirstSample
def FirstSample(qe,qi,total_tweets):
    sample = []
    for i in range(qi):
        sample.append([])
        remaining_tweets = total_tweets
        for j in range(qe):
            sample[i].append(random.randint(0, remaining_tweets))
            remaining_tweets = remaining_tweets - sample[i][j]
        if remaining_tweets != 0:
            assignment = random.randint(1,6)
            sample[i][assignment - 1] = sample[i][assignment - 1] + remaining_tweets
    return sample

# Función PrintSample
def PrintSample(qi,sample):
    for i in range(qi):
        try:
            print(str(sample[0][i]) + " = " + str(sum(sample[0][i])) + " \t\t " +
str(sample[1][i]) + " \t " + str(sample[2][i]))
        except:
            try:
                print(str(sample[0][i]) + " = " + str(sum(sample[0][i])) + " \t\t " +
str(sample[1][i]) + " \t " + str(sample[2]))
            except:
                print(str(sample[0][i]) + " = " + str(sum(sample[0][i])) + " \t\t " +
str(sample[1]) + " \t " + str(sample[2]))
    return
```

```

# Función evaluate
def evaluate(qi,sample,power_process):
    evaluation = []
    for i in range(qi):
        function = -(np.abs(sample[0][i][0] - 1752929) + np.abs(sample[0][i][1] - 1752929)
+ np.abs(sample[0][i][2] - 2188458) + np.abs(sample[0][i][3] - 179704) +
np.abs(sample[0][i][4] - 1013759) + np.abs(sample[0][i][5] - 709446)) + 34710105

        print("Función objetivo " + str(i))
        print(function)

        evaluation.append(function)
    return evaluation

# Función evaluateA
def evaluateA(qi,sample):
    evaluation = []
    evaluation.append(sample[1][0])
    i = 1
    while i < qi:
        evaluation.append(evaluation[i - 1] + sample[1][i])
        i = i + 1
    return evaluation

# Función Sorting
def Sorting(sample, qi):
    i = 0
    while i < qi - 1:
        if sample[1][i] < sample[1][i + 1]:
            for j in range(3):
                aux = sample[j][i]
                sample[j][i] = sample[j][i + 1]
                sample[j][i + 1] = aux
            i = 0
            continue
        i = i + 1
    return sample

# Función nullNewSample
def newSampleCreator(qi):
    nullCreator = []
    for i in range(qi):
        nullCreator.append(0)
    nullNewSample = []
    nullNewSample.append(nullCreator[:])
    nullNewSample.append(nullCreator[:])
    nullNewSample.append(nullCreator[:])
    return nullNewSample

%% Calibrar variables
total_tweets = 6942021
qe = 6 # Inamovible
qi = 60 # Movable
pc = 75 # Movable
pm = 40 # Movable
qelitism = 5 # Movable
qg = 10
sample = [[], [], []]
power_process = [9.311, 14.8685, 7.458, 90.8248, 16.1, 23.006] # Orden alfabético: Melissa,
Joel, Óscar, Dariana, Fernando, Hiram
ig = 0

```

```

%% AG: Creación de la población aleatoria
sample[0] = FirstSample(qe,qi,total_tweets)
PrintSample(qi, sample)

%% AG: Procedimiento de cada generación
while ig < qg:
    print("Población " + str(ig))
    print("")

    # Evaluamos a todas las muestras
    sample[1] = evaluate(qi,sample,power_process)

    print("Población " + str(ig) + " al ser evaluada")
    PrintSample(qi,sample)
    print("")

    # Evaluamos la suma acumulada
    sample[2] = evaluateA(qi,sample)

    print("Población " + str(ig) + " al ser evaluada con acumulación")
    PrintSample(qi,sample)
    print("")

    # Ordenamos por valor de f(x) es decir, por sample[1]
    sample = Sorting(sample, qi)

    print("Población " + str(ig) + " al ser ordenada")
    PrintSample(qi,sample)
    print("")

    # Reevaluamos la suma acumulada
    sample[2] = evaluateA(qi,sample)

    print("Población " + str(ig) + " al ser reevaluada")
    PrintSample(qi,sample)
    print("")

    # Creamos una nueva muestra vacía
    newSample = newSampleCreator(qi)

    print("Nueva muestra " + str(ig + 1) + " vacía")
    print(newSample)
    print("")

    print("Se aplicará elitismo")
    x = input()

    # Se aplica elitismo
    ii = 0
    while ii < qelitism:
        newSample[0][ii] = sample[0][ii]
        #newSample[1][ii] = sample[1][ii]
        #newSample[2][ii] = sample[2][ii]
        ii = ii + 1

    print("Nueva muestra " + str(ig + 1) + " después de elitismo")
    for i in range(qi):
        try:
            print(str(newSample[0][i]) + " = " + str(sum(newSample[0][i])))
        except:

```

```

        pass
    print("")
    print("Se aplicará cruzamiento")
    x = input()

    # Se aplicará cruzamiento
    while ii < qi:
        # Seleccionamos por ruleta
        # Padre 1
        random_pick = random.uniform(0, sample[2][qi - 1])

        for ic in range(qi):
            if random_pick < sample[2][ic]:
                c1 = sample[0][ic]
                print("Se elige como padre 1 el individuo " + str(ic) + "(random_pick = "
+ str(random_pick) + ")")
                break

        # Padre 2
        random_pick = random.uniform(0, sample[2][qi - 1])

        for ic in range(qi):
            if random_pick < sample[2][ic]:
                c2 = sample[0][ic]
                print("Se elige como padre 2 el individuo " + str(ic) + "(random_pick = "
+ str(random_pick) + ")")
                break

        # Se evalúa si se aplicará cruzamiento
        random_num = random.randint(0,99)
        if random_num < pc:
            print("Los cromosomas " + str(ii) + " y " + str(ii + 1) + " han activado
cruzamiento (Probabilidad: " + str(random_num) + " < " + str(pc) + ")")
            print("")

            # Mezclamos los 6 números de una tupla
            processors = [0, 1, 2, 3, 4, 5]
            random.shuffle(processors) # Se mezclarán y se harán sumas balanceadas de par
en par

            print("Tupla mezclada: " + str(processors))
            print("")

            print("Valores antes del cruzamiento: ")
            print(c1)
            print(c2)
            print("")

            for i in [0, 2, 4]:
                # Sumas equilibradas
                min_value = min(c1[processors[i]], c1[processors[i + 1]],
c2[processors[i]], c2[processors[i + 1]])
                operation = random.randint(0, 1) # Elige la operación con la que empezará
la suma equilibrada
                addition = random.randint(0, min_value)

                print("Tuplas " + str(processors[i] + 1) + " y " + str(processors[i + 1] +
1))

                print("min_value = " + str(min_value))
                print("addition = " + str(addition))
                print("")

```

```

        if operation == 0:
            c2[processors[i + 1]] = c2[processors[i + 1]] - addition
            c1[processors[i + 1]] = c1[processors[i + 1]] + addition
            c1[processors[i]] = c1[processors[i]] - addition
            c2[processors[i]] = c2[processors[i]] + addition
        else:
            c2[processors[i + 1]] = c2[processors[i + 1]] + addition
            c1[processors[i + 1]] = c1[processors[i + 1]] - addition
            c1[processors[i]] = c1[processors[i]] + addition
            c2[processors[i]] = c2[processors[i]] - addition

    print("Valores después del cruzamiento: ")
    print(c1)
    print(c2)
    print("")

    else:
        print("Los cromosomas " + str(ii) + " y " + str(ii + 1) + " no han activado
cruzamiento (Probabilidad: !" + str(random_num) + " < " + str(pc) + "!)")

        newSample[0][ii] = c1
        try:
            newSample[0][ii + 1] = c2
        except:
            pass

        ii = ii + 2

    print("Nueva muestra " + str(ig + 1) + " después del cruzamiento")
    for i in range(qi):
        try:
            print(str(newSample[0][i]) + " = " + str(sum(newSample[0][i])))
        except:
            pass
    print("")
    print("Se aplicará mutación")
    x = input()

    # Se aplica mutación
    ii = 0
    while ii < qi:
        # Se evalúa si se aplicará mutación
        random_num = random.randint(0,99)
        if random_num < pm:
            print("El cromosoma " + str(ii) + " ha activado mutación (Probabilidad: " +
str(random_num) + " < " + str(pm) + ")")
            print("")

            processors = [0, 1, 2, 3, 4, 5]
            random.shuffle(processors) # Se mezclarán y se hará una sumas balanceadas
cíclica

            print("Tupla mezclada: " + str(processors))
            print("")

            print("Valores antes de la mutación: ")
            print(newSample[0][ii])
            print("")

            min_value = min(newSample[0][ii])

```

```

        operation = random.randint(0, 1) # Elige la operación con la que empezará la
suma equilibrada
        addition = random.randint(0, min_value)

        print("Tupla " + str(processors))
        print("min_value = " + str(min_value))
        print("addition = " + str(addition))
        print("")

        if operation == 0:
            newSample[0][ii][processors[0]] = newSample[0][ii][processors[0]] -
addition
            newSample[0][ii][processors[1]] = newSample[0][ii][processors[1]] +
addition
            newSample[0][ii][processors[2]] = newSample[0][ii][processors[2]] -
addition
            newSample[0][ii][processors[3]] = newSample[0][ii][processors[3]] +
addition
            newSample[0][ii][processors[4]] = newSample[0][ii][processors[4]] -
addition
            newSample[0][ii][processors[5]] = newSample[0][ii][processors[5]] +
addition
        else:
            newSample[0][ii][processors[0]] = newSample[0][ii][processors[0]] +
addition
            newSample[0][ii][processors[1]] = newSample[0][ii][processors[1]] -
addition
            newSample[0][ii][processors[2]] = newSample[0][ii][processors[2]] +
addition
            newSample[0][ii][processors[3]] = newSample[0][ii][processors[3]] -
addition
            newSample[0][ii][processors[4]] = newSample[0][ii][processors[4]] +
addition
            newSample[0][ii][processors[5]] = newSample[0][ii][processors[5]] -
addition

        print("Valores después de la mutación: ")
        print(newSample[0][ii])
        print("")

        else:
            print("El cromosoma " + str(ii) + " no ha activado mutación (Probabilidad: !"
+ str(random_num) + " < " + str(pm) + "!)")
            print("")
            ii = ii + 1

        print("Nueva muestra " + str(ig + 1) + " después de la mutación")
        for i in range(qi):
            try:
                print(str(newSample[0][i]) + " = " + str(sum(newSample[0][i])))
            except:
                pass
        print("")
        print("Se tratará una nueva generación")
        x = input()

        sample = newSample
        ig = ig + 1

```

Anexo 8: Tabla de tiempo secuencial (en segundos) por integrante según la cantidad de tweets.

Tweets	Tiempo de Melissa	Tiempo de Joel	Tiempo de Óscar	Tiempo de Dariana	Tiempo de Fernando	Tiempo de Hiram
10,000	24.237	29.970	16.859	68.546	33.555	37.527
20,000	48.473	59.940	33.718	137.092	67.109	75.054
30,000	72.710	89.910	50.577	205.638	100.664	112.581
40,000	96.947	119.879	67.436	274.184	134.218	150.107
50,000	121.183	149.849	84.295	342.730	167.773	187.634
60,000	145.420	179.819	101.153	411.276	201.328	225.161
70,000	169.657	209.789	118.012	479.822	234.882	262.688
80,000	193.894	239.759	134.871	548.369	268.437	300.215
90,000	218.130	269.729	151.730	616.915	301.992	337.742
100,000	242.367	299.699	168.589	685.461	335.546	375.269
200,000	484.734	599.397	337.178	1370.921	671.092	750.537
300,000	727.101	899.096	505.767	2056.382	1006.638	1125.806
400,000	969.468	1198.794	674.356	2741.843	1342.185	1501.074
500,000	1211.835	1498.493	842.945	3427.303	1677.731	1876.343
600,000	1454.202	1798.191	1011.534	4112.764	2013.277	2251.612
700,000	1696.569	2097.890	1180.123	4798.225	2348.823	2626.880
800,000	1938.936	2397.589	1348.713	5483.685	2684.369	3002.149
900,000	2181.302	2697.287	1517.302	6169.146	3019.915	3377.417
1,000,000	2423.669	2996.986	1685.891	6854.607	3355.461	3752.686
2,000,000	4847.339	5993.971	3371.781	13709.213	6710.923	7505.372
3,000,000	7271.008	8990.957	5057.672	20563.820	10066.384	11258.058
4,000,000	9694.678	11987.943	6743.563	27418.426	13421.846	15010.744
5,000,000	12118.347	14984.928	8429.453	34273.033	16777.307	18763.430
6,000,000	14542.016	17981.914	10115.344	41127.639	20132.769	22516.116
6,942,021	16825.164	20805.137	11703.489	47584.823	23293.684	26051.225

Anexo 9: Tabla de tiempo en paralelo (en segundos) por integrante según la cantidad de tweets.

Es necesario tomar en cuenta la tabla 1 que aparece en el apartado de resultados donde se dividen los tweets, pues el tiempo reportado aquí es en función de los tweets obtenidos en dicha tabla.

Tweets	Tiempo de Melissa	Tiempo de Joel	Tiempo de Óscar	Tiempo de Dariana	Tiempo de Fernando	Tiempo de Hiram
10,000	5.611	5.368	5.410	1.391	4.141	4.683
20,000	12.099	9.659	10.533	3.928	9.566	7.929
30,000	18.127	14.508	15.977	5.196	14.496	11.727
40,000	24.106	19.417	21.306	6.909	19.267	15.712
50,000	30.754	23.505	26.554	8.945	24.710	18.944
60,000	37.235	27.797	32.094	9.809	29.558	22.835
70,000	42.846	33.168	37.423	11.523	33.944	27.245
80,000	48.985	37.882	42.198	15.498	39.729	30.089
90,000	55.187	42.518	47.921	15.608	44.383	34.202
100,000	61.222	47.364	53.480	16.396	49.215	38.109
200,000	122.732	94.372	106.452	34.849	97.932	76.776
300,000	183.215	142.651	159.699	52.184	147.120	114.919
400,000	244.524	189.904	212.818	70.047	197.009	152.276
500,000	305.756	237.256	265.776	88.562	244.274	192.569
600,000	366.914	284.699	318.876	106.486	294.110	229.991
700,000	428.282	331.880	372.189	123.575	342.338	269.199
800,000	489.290	379.508	425.327	141.356	392.676	306.054
900,000	551.259	425.950	478.377	159.486	440.602	345.611
1,000,000	612.381	473.437	531.332	178.014	490.535	382.917
2,000,000	1224.106	947.683	1063.086	354.315	980.204	766.805
3,000,000	1835.881	1421.866	1594.450	532.226	1469.702	1150.870
4,000,000	2447.846	1895.518	2125.664	711.385	1959.730	1534.364
5,000,000	3059.781	2369.806	2657.571	886.376	2450.087	1917.480
6,000,000	3672.528	2842.785	3188.916	1064.342	2939.961	2301.140
6,942,021	4248.520	3289.863	3689.501	1231.800	3401.629	2662.332