Instituto Politécnico Nacional
Escuela Superior de Cómputo

TLAMATINIME: Timetabling Problem, Prototipo de Optimización de Horarios en la ESCOM
2017-B092

Presentan:

Gómez Caballero Brenda
Larios Moguel Carlos Aníbal
Ricardo Flores José Antonio

Directores:

M. en C. José David Ortega Pacheco
M. en C. Mario Augusto Ramírez Morales

24 de octubre de 2018

Índice general

1. Módulo de Optimización **7**
1.1. Modelo de comportamiento del módulo: SistemaMódulo de Optimización 8
1.1.1. Prototipo1 del alogritmo de optimización 8
1.1.2. Prototipo2 del alogritmo de optimización 16
1.1.3. Prototipo3 del alogritmo de optimización 22
1.1.4. Pruebas de la versión: 1 30
1.1.5. Pruebas de la versión: 2 32
1.1.6. Pruebas de la versión: 3 33

2. Bibliografía **37**

3. Anexos **39**
3.1. Anexo 1. Información recabada 39

Índice de figuras

1.1. Determinación experimental del criterio de paro 31
1.2. Gráfica de mejora 31
1.3. Determinación experimental del criterio de paro 33
1.4. Gráfica de mejora 33
1.5. Determinación experimental del criterio de paro 35
1.6. Gráfica de mejora 35

3.1. Solicitud de información 40

CAPÍTULO 1

Módulo de Optimización

1.1. Modelo de comportamiento del módulo: SistemaMódulo de Optimización

En esta sección del documento se pretende explicar el comportamiento y funcionamiento de los distintos prototipos y versiones de los algoritmos usados para optimizar los horarios.

1.1.1. Prototipo1 del alogritmo de optimización

Tenemos que nuestras variables son Materia, Profesor, Grupo, Salón y Horario. Consideramos que no todos los profesores pueden impartir todas las materias por lo que se toma una variable MP la cuál es un arreglo de Tuplas Materia-Profesor que indica las materias que imparte cada profesor, de igual manera sabemos que los grupos son asignados a un salón de manera previa por lo que tenemos la variable GS la cuál lleva el Id de la tupla Grupo-Salón.

Teniendo en cuenta lo anterior y los conceptos de cómputo evolutivo expresados con anterioridad, tenemos que el cromosoma de los individuos que comprenden a nuestra población serán construidos con las tres variables mencionadas. De esta manera el cromosoma se puede representar de esta manera : (MP,GS,H).

Como se explicó anteriormente los algoritmos evolutivos tienen una estructura similar por lo que tenemos el siguiente pseudocódigo para representar nuestro algoritmo.

Algorithm 1: Algoritmo Principal Tlamantinime

```
1  inicializar los arreglos globales de valores MP, GS y H
2  inicializar el numero de iteraciones N
3  inicializar poblacionaux
4  poblacion = generarPoblacion()
5  y = 1
6  calificacion = 0
7  calificacionaux = 0
8  while y <= N do
9  |  calificacion = evalua(poblacion)
10 |  poblacionaux = poblacion
11 |  ran = random(0,1)
12 |  if ran == 1 then
13 |  |  poblacionaux = mutarHorarios(poblacionaux)
14 |  else
15 |  |  poblacionaux = mutarGrupos(poblacionaux)
16 |  end
17 |  calificacionaux = evalua(poblacionaux)
18 |  if calificacionaux <= calificacion then
19 |  |  poblacion = poblacionaux
20 |  end
21 |  incrementar y en 1
22 end
```

Siguiendo el pseudocódigo del algoritmo, después de inicializar las variables con la información que

utilizaremos para crear la estructura educativa, utilizamos la función generarPoblacion() para crear la población inicial con la cual trabajaremos. A continuación entramos a un ciclo que se detiene con el criterio de paro que hemos definido experimentalmente como un número de iteraciones, dentro del ciclo evaluamos la población inicial y creamos una copia de la misma, seleccionamos de manera aleatoria cualquiera de los dos operadores de mutación que hemos definido y operamos dicha mutación sobre la copia de la población inicial. Evaluamos la población mutada y comparamos contra la población inicial, la que sea determinada como mejor adaptada o, lo que es lo mismo, con la mejor calificación, será aquella con la que realizaremos la mutación en la siguiente iteración de forma que obtengamos el mejor resultado posible antes de alcanzar el número de iteraciones definido como criterio de paro.

Función: generar Poblacion

Esta función genera una población inicial, tradicionalmente la población inicial debe ser completamente aleatoria, sin embargo esto podría significar nunca encontrar una solución viable a partir de la misma. La función deifinida genera una población viable cuidando las restricciones escenciales y utilizando todas los atributos contenidos en el arreglo MP asegurando que todos los profesores impartan todas sus clases a

pesar de las mutaciones, mientras se verifica que los profesores no tengan traslapes.

Algorithm 2: generarPoblacion()

```
1  inicializar variable Tamanio con el número de individuos que debe tener la población
2  inicializar matrices binarias MG, PH, GH
3  inicializar variable poblacion y la variable individuo como arreglos
4  K = 1
5  pmi = [0,0]
6  t = 0
7  g = 0
8  profesor = 0
9  materia = 0
10 condicion = 0
11 while k <= Tamanio do
12     pmi ← individuo del arreglo MP sin repeticion
13     materia = pmi[0]
14     profesor = pmi[1]
15     while condicion! = 1 do
16         t ← individuo del arreglo H
17         g ← individuo del arreglo GS
18         if MG[materia][g] == 0 then
19             if PH[profesor][t] == 0 then
20                 if GH[g][t] == 0 then
21                     PH[profesor][t] = 1
22                     MG[materia][g] = 1
23                     GH[g][t] = 1
24                     condicion = 1
25                     individuo = [pmi,g,t]
26                     poblacion[k]= individuo
27                 end
28             end
29         end
30         incrementar k en 1
31     end
32     return poblacion
33 end
```

Función: evalua

Para evaluar a un individuo tomamos en cuenta las siguientes restricciones con su respectiva penalizacion. La penalizacion es mayor cuando se viola una restriccion escencial(hard constraint) y menor cuando se viola una restriccion no escencial(soft constraint).

- Un profesor no puede dar clase en dos grupos al mismo tiempo. Penalizacion = 30
- No se pueden impartir dos clases al mismo tiempo en un grupo. Penalizacion = 30
- No se debe impartir dos veces la misma materia en un grupo. Penalizacion = 10

- El horario de un grupo debe tener los menos huecos posibles. Penalizacion = 2

- El horario de un profesor debe tener los menos huecos posibles. Penalizacion = 2

- Se debe evitar que un profesor imparta dos materias distintas en el mismo grupo. Penalizacion = 1

La calificación que devuelve esta función es una sumatoria de todas las penalizaciones que se dan por infringir alguna restricción.

```
1  inicializar variable Tamanio con el número de individuos de la población que recibe
2  inicializar matrices binarias MGE, PHE, GHE,PGE
3  inicializar variable individuo como arreglo
4  calificacion = 0
5  profesor = 0
6  materia = 0
7  grupo = 0
8  horario = 0
9  k = 0
10 while k <= Tamanio do
11     individuo ← individuo de la población de individuos
12     materia = individuo[0][0]
13     profesor = individuo[0][1]
14     grupo = individuo[1]
15     horario = individuo[2]
16     incrementar phe[profesor][hora] en 1
17     incrementar mge[materia][grupo] en 1
18     incrementar ghe[grupo][hora] en 1
19     incrementar pge[profesor][grupo] en 1
20     if PHE[profesor][horario] > 1 then
21         incrementar calificacion en 30
22     if GHE[grupo][hora] > 1 then
23         incrementar calificacion en 30
24     if MGE[materia][grupo] > 1 then
25         incrementar calificacion en 10
26     if PGE[profesor][grupo] then
27         incrementar calificacion en 1
28     if PHE[profesor][1]==1 then
29         if PHE[profesor][2]==0 then
30             if PHE[profesor][3]==0 then
31                 incrementar calificacion en 2
32     if PHE[profesor][2]==1 then
33         if PHE[profesor][3]==0 then
34             if PHE[profesor][4]==0 then
35                 incrementar calificacion en 2
36     if PHE[profesor][3]==1 then
37         if PHE[profesor][4]==0 then
38             if PHE[profesor][5]==0 then
39                 incrementar calificacion en 2
40     if PHE[profesor][4]==1 then
41         if PHE[profesor][5]==0 then
42             if PHE[profesor][6]==0 then
43                 incrementar calificacion en 2
```

Algorithm 3: evalua(individuos)

```
45
46  if GHE[grupo][1]==1 then
47      if GHE[grupo][2]==0 then
48          if GHE[grupo][3]==0 then
49              incrementar calificacion en 2
50  if GHE[grupo][2]==1 then
51      if GHE[grupo][3]==0 then
52          if GHE[grupo][4]==0 then
53              incrementar calificacion en 2
54  if GHE[grupo][3]==1 then
55      if GHE[grupo][4]==0 then
56          if GHE[grupo][5]==0 then
57              incrementar calificacion en 2
58  if GHE[grupo][4]==1 then
59      if GHE[grupo][5]==0 then
60          if GHE[grupo][6]==0 then
61              incrementar calificacion en 2
62  return calificacion;
```

Funciones: mutación

Se definieron dos funciones de mutación, en ambos casos estamos considerando que dado el espacio limitado que tenemos para generar dichos cambios, aplicamos la mutación al mismo tiempo en dos cromosomas distintos de esta manera se busca mantener la viabilidad del resultado. La primera función de mutación actúa sobre el gen GS de manera que el profesor sigue impartiendo una materia en el mismo horario pero en un grupo distinto, lo cual mitiga los casos en que un profesor imparte dos materias distintas en el mismo grupo o los casos en que una materia se imparte dos veces en el mismo grupo.

Algorithm 4: mutacionGrupos(poblacion)

```
1  inicializar variable aux1 y aux2 como arreglos
2  grupo1 = 0
3  grupo2 = 0
4  aux1 ← individuo de la poblacion
5  aux2 ← individuo de la poblacion
6  grupo1 = aux1[1]
7  grupo2 = aux2[1]
8  aux1[1] = grupo2
9  aux2[1] = grupo1
10 poblacion[aux1] = aux1
11 poblacion[aux2] = aux2
12 return poblacion
```

La segunda función actua sobre el gen H manteniendo los profesores y las materias en el mismo grupo pero modificando de esta manera el horario en que la imparten, con esto se busca mitigar los traslapes en grupos y profesores a demás de reducir los huecos en los horarios de los profesores.

Algorithm 5: mutacionHorarios(poblacion)

```
1  inicializar variable aux1 y aux2 como arreglos
2  horario1 = 0
3  horario2 = 0
4  aux1 ← individuo de la poblacion
5  aux2 ← individuo de la poblacion
6  horario1 = aux1[2]
7  horario2 = aux2[2]
8  aux1[2] = horario2
9  aux2[2] = horario1
10 poblacion[aux1] = aux1
11 poblacion[aux2] = aux2
12 return poblacion
```

Resultado

Como resultado de este prototipo tenemos que las funciones crean un horario mejor que el actual de ESCOM para el espacio de prueba utilizado de acuerdo a nuestros criterios, el propósito de este prototipo fue generar las funciones que se van a utilizar y comprobar su correcto funcionamiento, aunado a esto pudimos determinar el número de iteraciones necesarias para llegar al mejor resultado posible de forma

que las mismas se puedan considerar un criterio de paro.

Como parte negativa tenemos que nunca se logra una calificación perfecta (igual a 0), teniendo como mejor calificación lograda en las pruebas el 42.

1.1.2. Prototipo2 del alogritmo de optimización

Teniendo en cuenta los resultados del protipo 1, tomamos las funciones ya definidas sin embargo, de acuedo a las pruebas del prototipo 1, se notó que el resultado nunca llegaba al óptimo y se detenía el progreso en al rededor del valor de 42, siendo esto sospechoso se notó un error en la función de evaluación, al analizar los llamados huecos se estaba cometiendo un error que penalizaba a los profesores con una sola clase.

Al corregir el error mencionado en la función de evaluación junto con algunos arreglos menores se logró llegar a una calificación de 0, lo cuál es la calificación óptima. Tener una calificación óptima se tomó en cuenta como uno de los criterios de paro de la función, por otro lado de acuerdo a la complejidad del problema, la probabilidad de no alcanzar esta calificación es alta por lo que alternativamente se utiliza como criterio de paro un número de iteraciones definido de forma experimental.

A continuación se muestra el pseudocódigo de las funciones utilizadas para el presente prototipo.

Algorithm 6: Algoritmo Principal Tlamantinime

```
1  inicializar los arreglos globales de valores MP, GS y H
2  inicializar el numero de iteraciones N
3  inicializar poblacionaux
4  poblacion = generarPoblacion()
5  y = 1
6  calificacion = 0
7  calificacionaux = 0
8  for y <= N do
9  |  calificacion = evalua(poblacion)
10 |  poblacionaux = poblacion
11 |  ran = random(0,1)
12 |  if ran == 1 then
13 |  |  poblacionaux = mutarHorarios(poblacionaux)
14 |  else
15 |  |  poblacionaux = mutarGrupos(poblacionaux)
16 |  end
17 |  calificacionaux = evalua(poblacionaux)
18 |  if calificacionaux <= calificacion then
19 |  |  poblacion = poblacionaux
20 |  end
21 |  if calificacion == 0 then
22 |  |  break
23 |  end
24 |  incrementar y en 1
25 end
```

Siguiendo el pseudocódigo del algoritmo, después de inicializar las variables con la información que utilizaremos para crear la estructura educativa, utilizamos la función generarPoblacion() para crear la población inicial con la cual trabajaremos. A continuación entramos a un ciclo que se detiene con alguno de los criterios de paro que hemos definido experimentalmente ya sea el número de iteraciones o la calificación óptima, dentro del ciclo evaluamos la población inicial y creamos una copia de la misma, seleccionamos de manera aleatoria cualquiera de los dos operadores de mutación que hemos definido y operamos dicha mutación sobre la copia de la población inicial. Evaluamos la población mutada y comparamos contra la población inicial, la que sea determinada como mejor adaptada o, lo que es lo mismo, con la mejor calificación, será aquella con la que realizaremos la mutación en la siguiente iteración de forma que obtengamos la población se mantenga o mejore con cada iteración.

Función: generar Poblacion

Esta función genera una población inicial, tradicionalmente la población inicial debe ser completamente aleatoria, sin embargo esto podría significar nunca encontrar una solución viable a partir de la misma. La función deifinida genera una población viable cuidando las restricciones escenciales y utilizando todas los atributos contenidos en el arreglo MP asegurando que todos los profesores impartan todas sus clases a

pesar de las mutaciones, mientras se verifica que los profesores no tengan traslapes.

Algorithm 7: generarPoblacion()

```
1  inicializar variable Tamanio con el número de individuos que debe tener la población
2  inicializar matrices binarias MG, PH, GH
3  inicializar variable poblacion y la variable individuo como arreglos
4  K = 1
5  pmi = [0,0]
6  t = 0
7  g = 0
8  profesor = 0
9  materia = 0
10 condicion = 0
11 while k <= Tamanio do
12     pmi ← individuo del arreglo MP sin repeticion
13     materia = pmi[0]
14     profesor = pmi[1]
15     while condicion! = 1 do
16         t ← individuo del arreglo H
17         g ← individuo del arreglo GS
18         if MG[materia][g] == 0 then
19             if PH[profesor][t] == 0 then
20                 if GH[g][t] == 0 then
21                     PH[profesor][t] = 1
22                     MG[materia][g] = 1
23                     GH[g][t] = 1
24                     condicion = 1
25                     individuo = [pmi,g,t]
26                     poblacion[k]= individuo
27                 end
28             end
29         end
30         incrementar k en 1
31     end
32     return poblacion
33 end
```

Función: evalua

Para evaluar a un individuo tomamos en cuenta las siguientes restricciones con su respectiva penalizacion. La penalizacion es mayor cuando se viola una restriccion escencial(hard constraint) y menor cuando se viola una restriccion no escencial(soft constraint).

- Un profesor no puede dar clase en dos grupos al mismo tiempo. Penalizacion = 30
- No se pueden impartir dos clases al mismo tiempo en un grupo. Penalizacion = 30
- No se debe impartir dos veces la misma materia en un grupo. Penalizacion = 30

- El horario de un grupo debe tener los menos huecos posibles. Penalizacion = 2

- El horario de un profesor debe tener los menos huecos posibles. Penalizacion = 2

- Se debe evitar que un profesor imparta dos materias distintas en el mismo grupo. Penalizacion = 1

La calificación que devuelve esta función es una sumatoria de todas las penalizaciones que se dan por infringir alguna restricción.

Algorithm 8: evalua(individuos)

```
inicializar variable Tamanio con el número de individuos de la población que recibe
inicializar matrices binarias MGE, PHE, GHE,PGE
inicializar variable individuo como arreglo
calificacion = 0
profesor = 0
materia = 0
grupo = 0
horario = 0
k = 0
while k <= Tamanio do
    individuo ← individuo de la población de individuos
    materia = individuo[0][0]
    profesor = individuo[0][1]
    grupo = individuo[1]
    horario = individuo[2]
    incrementar phe[profesor][hora] en 1
    incrementar mge[materia][grupo] en 1
    incrementar ghe[grupo][hora] en 1
    incrementar pge[profesor][grupo] en 1
    if PHE[profesor][horario] > 1 then
        incrementar calificacion en 30
    if GHE[grupo][hora] > 1 then
        incrementar calificacion en 30
    if MGE[materia][grupo] > 1 then
        incrementar calificacion en 30
    if PGE[profesor][grupo] then
        incrementar calificacion en 1
    if PHE[profesor][1]==1 AND PHE[profesor][2]==0 AND PHE[profesor][3]==0 then
        if PHE[profesor][4] == 1 OR PHE[profesor][5] == 1 OR PHE[profesor][6] == 1 OR
          PHE[profesor][7] == 1 then
            incrementar calificacion en 2
    if PHE[profesor][2]==1 AND PHE[profesor][3]==0 AND PHE[profesor][4]==0 then
        if PHE[profesor][5] == 1 OR PHE[profesor][6] == 1 OR PHE[profesor][7] == 1 then
            incrementar calificacion en 2
    if PHE[profesor][3]==1 AND PHE[profesor][4]==0 AND PHE[profesor][5]==0 then
        if PHE[profesor][6] == 1 OR PHE[profesor][7] == 1 then
            incrementar calificacion en 2
    if PHE[grupo][4]==1 AND PHE[grupo][5]==0 AND PHE[grupo][6]==0 then
        if PHE[grupo][7] == 1 then
            incrementar calificacion en 2
```

Algorithm 9: evalua(individuos)

```
45
46  if GHE[grupo][1]==1 AND GHE[grupo][2]==0 AND GHE[grupo][3]==0 then
47      if GHE[grupo][4] == 1 OR GHE[grupo][5] == 1 OR GHE[grupo][6] == 1 OR
         GHE[grupo][7] == 1 then
48          incrementar calificacion en 2
49  if GHE[grupo][2]==1 AND GHE[grupo][3]==0 AND GHE[grupo][4]==0 then
50      if GHE[grupo][5] == 1 OR GHE[grupo][6] == 1 OR GHE[grupo][7] == 1 then
51          incrementar calificacion en 2
52  if GHE[grupo][3]==1 AND GHE[grupo][4]==0 AND GHE[grupo][5]==0 then
53      if GHE[grupo][6] == 1 OR GHE[grupo][7] == 1 then
54          incrementar calificacion en 2
55  if GHE[grupo][4]==1 AND GHE[grupo][5]==0 AND GHE[grupo][6]==0 then
56      if GHE[grupo][7] == 1 then
57          incrementar calificacion en 2
58  return calificacion;
```

Funciones: mutación

Se definieron dos funciones de mutación, en ambos casos estamos considerando que dado el espacio limitado que tenemos para generar dichos cambios, aplicamos la mutación al mismo tiempo en dos cromosomas distintos de esta manera se busca mantener la viabilidad del resultado. La primera función de mutación actúa sobre el gen GS de manera que el profesor sigue impartiendo una materia en el mismo horario pero en un grupo distinto, lo cual mitiga los casos en que un profesor imparte dos materias distintas en el mismo grupo o los casos en que una materia se imparte dos veces en el mismo grupo.

Algorithm 10: mutacionGrupos(poblacion)

```
1  inicializar variable aux1 y aux2 como arreglos
2  grupo1 = 0
3  grupo2 = 0
4  aux1 ← individuo de la poblacion
5  aux2 ← individuo de la poblacion
6  grupo1 = aux1[1]
7  grupo2 = aux2[1]
8  aux1[1] = grupo2
9  aux2[1] = grupo1
10 poblacion[aux1] = aux1
11 poblacion[aux2] = aux2
12 return poblacion
```

La segunda función actua sobre el gen H manteniendo los profesores y las materias en el mismo grupo pero modificando de esta manera el horario en que la imparten, con esto se busca mitigar los traslapes en grupos y profesores a demás de reducir los huecos en los horarios de los profesores.

Algorithm 11: mutacionHorarios(poblacion)

```
1  inicializar variable aux1 y aux2 como arreglos
2  horario1 = 0
3  horario2 = 0
4  aux1 ← individuo de la poblacion
5  aux2 ← individuo de la poblacion
6  horario1 = aux1[2]
7  horario2 = aux2[2]
8  aux1[2] = horario2
9  aux2[2] = horario1
10 poblacion[aux1] = aux1
11 poblacion[aux2] = aux2
12 return poblacion
```

Resultado

Como resultado de este prototipo tenemos que las funciones crean un horario mejor que el actual de ESCOM para el espacio de prueba utilizado de acuerdo a nuestros criterios, el propósito de este prototipo fue corregir los errores encontrados en el prototipo 1, aunado a esto pudimos determinar un nuevo criterio de paro teniendo en cuenta que utilizamos un número de iteraciones determinado de forma experimental y el momento en que la calificación llega a 0 la cuál es la mejor calificación alcanzable para este problema.

Como parte negativa tenemos que como tal aunque sirve para probar las funciones y los conceptos, este algoritmo sólo tiene un individuo en la población inicial y solo genera un resultado.

1.1.3. Prototipo3 del alogritmo de optimización

Tomando en cuenta los resultados del segundo prototipo, utilizamos las funciones anteriores para avanzar más, en este caso el prototipo 2 utilizaba las funciones para realizar un sólo individuo, llámese individuo a una estructura educativa. Sin embargo para esta versión se han realizado las adecuaciones necesarias para generar no uno si no varios individuos.

En este caso tenemos que la población inicial tiene un tamaño que hemos determinado de manera experimental, así mismo de acuerdo a los objetivos de el presente trabajo, tenemos que se puede elegir el número de resultados que arroja el algoritmo, entre 1 y 10 según lo acordado con el profesor Iván Giovanny Mosso.

Como se explicó anteriormente los algoritmos evolutivos tienen una estructura similar por lo que tenemos el siguiente pseudocódigo para representar nuestro algoritmo. Se reemplazó dentro de la función principal la función generaPoblacion por poblacionEvaluada. La diferencia es que la función poblacionEvaluada genera una matriz con un número de filas igual al número de individuos determinado y donde la primer columna es contiene una estructura educativa y su segunda columna contiene la evaluación correspondiente a dicho individuo.

Algorithm 12: principal(número iteraciones,tamañoo poblacion, resultados deseados)

```
1  inicializar los arreglos globales de valores MP, GS y H
2  inicializar el numero de iteraciones N
3  inicializar el tamaño de la población inicial IND
4  inicializar el número de resultados deseados DESEADOS
5  inicializar poblacionaux
6  poblacion = poblacionEvaluada(IND,MP,H,GS)
7  poblacionaux = poblacion
8  y = 1
9  i = 1
10 stop = 0
11 for y <= N do
12 |   for i <= IND do
13 |   |   if poblacionaux[i][1] != 0 then
14 |   |   |   a
15 |   |   end
16 |   |   ux = mutacionGrupos(mutacionHorarios(poblacionaux[i][0]))
17 |   |   poblacionaux[i][0] = aux
18 |   |   poblacionaux[i][1] = evalua(aux)
19 |   |   if poblacion[i][1] == 0 then
20 |   |   |   incrementar stop en 1
21 |   |   end
22 |   |   incrementar i en 1
23 |   end
24 |   if stop == DESEADOS then
25 |   |   break;
26 |   end
27 |   incrementar y en 1
28 end
29 return poblacionaux
```

.

Función: generar Poblacion

Esta función genera una población inicial, tradicionalmente la población inicial debe ser completamente aleatoria, sin embargo esto podría significar nunca encontrar una solución viable a partir de la misma. La función deifinida genera una población viable cuidando las restricciones escenciales y utilizando todas los atributos contenidos en el arreglo MP asegurando que todos los profesores impartan todas sus clases a

pesar de las mutaciones, mientras se verifica que los profesores no tengan traslapes. A de

Algorithm 13: generarPoblacion(MP,H,GS)

```
1  inicializar variable Tamanio con el número de individuos que debe tener la población
2  inicializar matrices binarias MG, PH, GH
3  inicializar variable poblacion y la variable individuo como arreglos
4  K = 1
5  pmi = [0,0]
6  t = 0
7  g = 0
8  profesor = 0
9  materia = 0
10 condicion = 0
11 while k <= Tamanio do
12     pmi ← individuo del arreglo MP sin repeticion
13     materia = pmi[0]
14     profesor = pmi[1]
15     while condicion! = 1 do
16         t ← individuo del arreglo H
17         g ← individuo del arreglo GS
18         if MG[materia][g] == 0 then
19             if PH[profesor][t] == 0 then
20                 if GH[g][t] == 0 then
21                     PH[profesor][t] = 1
22                     MG[materia][g] = 1
23                     GH[g][t] = 1
24                     condicion = 1
25                     individuo = [pmi,g,t]
26                     poblacion[k]= individuo
27                 end
28             end
29         end
30         incrementar k en 1
31     end
32     return poblacion
33 end
```

Función: evalua

Para evaluar a un individuo tomamos en cuenta las siguientes restricciones con su respectiva penalizacion. La penalizacion es mayor cuando se viola una restriccion escencial(hard constraint) y menor cuando se viola una restriccion no escencial(soft constraint).

- Un profesor no puede dar clase en dos grupos al mismo tiempo. Penalizacion = 30
- No se pueden impartir dos clases al mismo tiempo en un grupo. Penalizacion = 30
- No se debe impartir dos veces la misma materia en un grupo. Penalizacion = 30

- El horario de un grupo debe tener los menos huecos posibles. Penalizacion = 2

- El horario de un profesor debe tener los menos huecos posibles. Penalizacion = 2

- Se debe evitar que un profesor imparta dos materias distintas en el mismo grupo. Penalizacion = 1

La calificación que devuelve esta función es una sumatoria de todas las penalizaciones que se dan por infringir alguna restricción.

Algorithm 14: evalua(individuos)

```
1  inicializar variable Tamanio con el número de individuos de la población que recibe
2  inicializar matrices binarias MGE, PHE, GHE,PGE
3  inicializar variable individuo como arreglo
4  calificacion = 0
5  profesor = 0
6  materia = 0
7  grupo = 0
8  horario = 0
9  k = 0
10 while k <= Tamanio do
11     individuo ← individuo de la población de individuos
12     materia = individuo[0][0]
13     profesor = individuo[0][1]
14     grupo = individuo[1]
15     horario = individuo[2]
16     incrementar phe[profesor][hora] en 1
17     incrementar mge[materia][grupo] en 1
18     incrementar ghe[grupo][hora] en 1
19     incrementar pge[profesor][grupo] en 1
20     if PHE[profesor][horario] > 1 then
21         incrementar calificacion en 30
22     if GHE[grupo][hora] > 1 then
23         incrementar calificacion en 30
24     if MGE[materia][grupo] > 1 then
25         incrementar calificacion en 30
26     if PGE[profesor][grupo] then
27         incrementar calificacion en 1
28     if PHE[profesor][1]==1 AND PHE[profesor][2]==0 AND PHE[profesor][3]==0 then
29         if PHE[profesor][4] == 1 OR PHE[profesor][5] == 1 OR PHE[profesor][6] == 1 OR
             PHE[profesor][7] == 1 then
30             incrementar calificacion en 2
31     if PHE[profesor][2]==1 AND PHE[profesor][3]==0 AND PHE[profesor][4]==0 then
32         if PHE[profesor][5] == 1 OR PHE[profesor][6] == 1 OR PHE[profesor][7] == 1 then
33             incrementar calificacion en 2
34     if PHE[profesor][3]==1 AND PHE[profesor][4]==0 AND PHE[profesor][5]==0 then
35         if PHE[profesor][6] == 1 OR PHE[profesor][7] == 1 then
36             incrementar calificacion en 2
37     if PHE[grupo][4]==1 AND PHE[grupo][5]==0 AND PHE[grupo][6]==0 then
38         if PHE[grupo][7] == 1 then
39             incrementar calificacion en 2
```

Algorithm 15: evalua(individuos)

```
45
46  if GHE[grupo][1]==1 AND GHE[grupo][2]==0 AND GHE[grupo][3]==0 then
47      if GHE[grupo][4] == 1 OR GHE[grupo][5] == 1 OR GHE[grupo][6] == 1 OR
          GHE[grupo][7] == 1 then
48          incrementar calificacion en 2
49  if GHE[grupo][2]==1 AND GHE[grupo][3]==0 AND GHE[grupo][4]==0 then
50      if GHE[grupo][5] == 1 OR GHE[grupo][6] == 1 OR GHE[grupo][7] == 1 then
51          incrementar calificacion en 2
52  if GHE[grupo][3]==1 AND GHE[grupo][4]==0 AND GHE[grupo][5]==0 then
53      if GHE[grupo][6] == 1 OR GHE[grupo][7] == 1 then
54          incrementar calificacion en 2
55  if GHE[grupo][4]==1 AND GHE[grupo][5]==0 AND GHE[grupo][6]==0 then
56      if GHE[grupo][7] == 1 then
57          incrementar calificacion en 2
58 return calificacion;
```

Función: Población evaluada

Para generar varios individuos utilizando las funciones definidas se creó esta función, la cuál utiliza la función generar Población así como la función evalua. De forma que recibe como parámetros el tamaño de la población deseada y los arreglos materia-profesor, horario y grupo-salón. Utilizando dicha información y las funciones mencionadas, devuelve un una matriz con un número de filas igual al número de individuos determinado y donde la primer columna es contiene una estructura educativa y su segunda columna contiene la evaluación correspondiente a dicho individuo. Dicha matriz es nuestra población inicial con sus respectivas evaluaciones de forma que sea más sencillo manejar los resultados.

Algorithm 16: poblacionEvaluada(IND,MP,H,GS)

```
1 inicializar las vairables aux y pcalif como arreglos
2 inicializar variables tt y eva i = 0
3 for i <= IND do
4     tt = generarPoblacion(MP,H,GS)
5     eva = evalua(tt)
6     aux = [tt,eva]
7     pcalif[i] = aux
8 end
9 return pcalif
```

Funciones: mutación

Se definieron dos funciones de mutación, en ambos casos estamos considerando que dado el espacio limitado que tenemos para generar dichos cambios, aplicamos la mutación al mismo tiempo en dos

cromosomas distintos de esta manera se busca mantener la viabilidad del resultado. La primera función de mutación actúa sobre el gen GS de manera que el profesor sigue impartiendo una materia en el mismo horario pero en un grupo distinto, lo cual mitiga los casos en que un profesor imparte dos materias distintas en el mismo grupo o los casos en que una materia se imparte dos veces en el mismo grupo.

Algorithm 17: mutacionGrupos(poblacion)

```
1  inicializar variable aux1 y aux2 como arreglos
2  grupo1 = 0
3  grupo2 = 0
4  aux1 ← individuo de la poblacion
5  aux2 ← individuo de la poblacion
6  grupo1 = aux1[1]
7  grupo2 = aux2[1]
8  aux1[1] = grupo2
9  aux2[1] = grupo1
10 poblacion[aux1] = aux1
11 poblacion[aux2] = aux2
12 return poblacion
```

La segunda función actua sobre el gen H manteniendo los profesores y las materias en el mismo grupo pero modificando de esta manera el horario en que la imparten, con esto se busca mitigar los traslapes en grupos y profesores a demás de reducir los huecos en los horarios de los profesores.

Algorithm 18: mutacionHorarios(poblacion)

```
1  inicializar variable aux1 y aux2 como arreglos
2  horario1 = 0
3  horario2 = 0
4  aux1 ← individuo de la poblacion
5  aux2 ← individuo de la poblacion
6  horario1 = aux1[2]
7  horario2 = aux2[2]
8  aux1[2] = horario2
9  aux2[2] = horario1
10 poblacion[aux1] = aux1
11 poblacion[aux2] = aux2
12 return poblacion
```

Resultado

Como resultado de este prototipo tenemos que las funciones crean un horario mejor que el actual de ESCOM para el espacio de prueba utilizado de acuerdo a nuestros criterios, el propósito de este prototipo fue generar las funciones que se van a utilizar y comprobar su correcto funcionamiento, aunado a esto pudimos determinar el número de iteraciones necesarias para llegar al mejor resultado posible de forma que las mismas se puedan considerar un criterio de paro.

Como parte negativa tenemos que nunca se logra una calificación perfecta (igual a 0), teniendo como mejor calificación lograda en las pruebas el 42.

1.1.4. Pruebas de la versión: 1

En esta sección se describen los resultadods de las pruebas realizadas de la primera versión del algoritmo de optimización de horarios.

Como consideración tenemos que las pruebas fueron realizadas en una laptop dell Inspiron 15 con procesador intel i5 5ta generación 4gb de memoria ram usando el sistema operativo debian 9.

Pruebas realizadas

Inicialmente las funciones que comprenden el algoritmo fueron probadas por separado.

- Se corroboró que la función de generar Población utilizara todas las opciones de Materia-Profesor disponibles y que el resultado tuviera la estructura que hemos definido.
- Se probó la función de evaluación, para corroborar que las restricciones ingresadas sean tomadas en cuenta al momento de asignar una calificación a un horario.
- Se probaron las funciones de mutación de forma que el resultado arrojado después de llevar a cabo dichos operadores sea en realidad distinto a la entrada del mismo así como se corroboró que se llevaran a cabo de la manera esperada.

Una vez que se corrigieron los errores arrojados por las pruebas de cada función por separado, las funciones fueron integradas para ser utilizadas en conjunto. Para corroborar que el funcionamiento fuera correcto, utilizamos la estructura educativa actual de la ESCOM para el nivel 2 y el turno matutino. Utilizando estos datos, se creó una población inicial aleatoria, una población inicial usando la función generar Población y se tomó como ejemplo la estructura tal cual se utiliza hoy en día. Haciendo uso de la función de evaluación determinamos que la calificacion de la población aleatoria fue de 532, la calificación de la población generada por la función generar Población fue de 160 y finalmente la calificación de la estructura de ejemplo fue de 116. Teniendo en cuenta que nuestro problema es de minimización y la calificación óptima es 0, el horario actual fue el mejor evaluado.

Una vez que se operaron las funciones de mutación sobre la población generada por nuestra función de generar Población, se logró disminuir la calificación hasta llegar a 42 lo cuál demuestra que obtuvimos un mejor resultado que el actual de ESCOM.

Finalmente se determinó que el criterio de paro al no alcanzar una calificación de 0(calificación óptima) debe ser un número determinado de iteraciones, el cúal se determinó de forma experimental de la siguiente manera.

En cinco ocasiones se tomó la calificación arrojada por la función de evaluación para el resultado después de llevar a cabo la mutación un número determinado de veces el cúal fue aumentado gradualmente para determinar el momento en que la disminución en la calificación fuera despreciable. En la figura 1.1 se muestran los resultados del experimento y en la figura 1.2 se muestra la gráfica de los mismos donde se puede apreciar que a partir de las 60,000 iteraciones la mejora es mínima por lo cuál se determinó este punto como criterio de paro teniendo en cuenta que la calificación en este punto es muy superior a la del horario actual.

Repeticiones\Resultadc	Promedio	Calificación 1	Calificación 2	Calificación 3	Calificación 4	Calificacion 5	Tiempo(segundos)
0	160	160	160	160	160	160	0
5000	71,2	76	64	70	68	78	2.5
10000	58	60	56	50	60	64	4.9
15000	55,2	54	52	54	58	58	7.6
20000	53,6	52	58	52	50	56	9.5
25000	50,4	50	52	48	48	54	12.5
30000	50	50	50	52	50	48	14.3
35000	48,8	50	50	48	50	48	18.6
40000	48,4	50	48	48	48	48	21.9
45000	47,2	48	50	44	50	44	22.7
50000	46	46	48	44	46	46	23.8
55000	46	48	44	46	46	46	25.6
60000	46	46	46	46	46	46	29.3
65000	45,6	46	44	44	48	46	31.8
70000	45,6	44	46	44	48	46	33.5
75000	45,6	46	46	46	46	44	36.1
80000	44,8	44	46	46	42	46	38.1
85000	44,8	44	46	46	46	42	40
90000	43,6	44	44	44	42	44	43
95000	43,2	44	42	44	44	42	45.7
100000	43,2	42	44	44	44	42	48.65

Figura 1.1: Determinación experimental del criterio de paro

Figura 1.2: Gráfica de mejora

1.1.5. Pruebas de la versión: 2

En esta sección se describen los resultadods de las pruebas realizadas de la segunda versión del algoritmo de optimización de horarios.

Como consideración tenemos que las pruebas fueron realizadas en una laptop dell Inspiron 15 con procesador intel i5 5ta generación 4gb de memoria ram usando el sistema operativo debian 9.

Pruebas realizadas

Inicialmente las funciones que comprenden el algoritmo fueron probadas por separado.

- Se corroboró que la función de generar Población utilizara todas las opciones de Materia-Profesor disponibles y que el resultado tuviera la estructura que hemos definido.
- Se probó la función de evaluación, para corroborar que las restricciones ingresadas sean tomadas en cuenta al momento de asignar una calificación a un horario.
- Se probaron las funciones de mutación de forma que el resultado arrojado después de llevar a cabo dichos operadores sea en realidad distinto a la entrada del mismo así como se corroboró que se llevaran a cabo de la manera esperada.

Una vez que se corrigieron los errores arrojados por las pruebas de de la primera versión del algoritmo se utilizó el mismo esquema de pruebas. Para corroborar que el funcionamiento fuera correcto, utilizamos la estructura educativa actual de la ESCOM para el nivel 2 y el turno matutino. Utilizando estos datos, se creó una población inicial aleatdforia, una población inicial usando la función generar Población y se tomó como ejemplo la estructura tal cual se utiliza hoy en día.

Haciendo uso de la función de evaluación determinamos que la calificacion de la población aleatoria fue de 1315, la calificación de la población generada por la función generar Población fue de 20 y finalmente la calificación de la estructura de ejemplo fue de 36. Teniendo en cuenta que nuestro problema es de minimización y la calificación óptima es 0, el horario generado por la función de población inicial fue el mejor adaptado sin llegar a ser el óptimo.

Una vez que se operaron las funciones de mutación sobre la población generada por nuestra función de generar Población, se logró disminuir la calificación hasta llegar a 0 indicando así que nuestro algoritmo arroja soluciones óptimas de acuerdo a las restricciones planteadas.

Finalmente se determinó utilizar dos criterios de paro, si bien alcanzar una calificación óptima funciona como uno, la complejidad del problema implica una alta probabilidad de no encontrar soluciones óptimas en todas las ocasiones que se ejecute el algoritmo. Por lo tanto un criterio de paro alternativo debe ser un número determinado de iteraciones, el cúal se determinó de forma experimental de la siguiente manera.

En cinco ocasiones se tomó la calificación arrojada por la función de evaluación para el resultado después de llevar a cabo la mutación un número determinado de veces el cúal fue disminuyendo hasta llegar al 0 en cada uno de las 5 corridas. En la figura 1.3 se muestran los resultados del experimento y en la figura 1.2 se muestra la gráfica de los mismos donde se puede apreciar que a partir de las 20,000

iteraciones se alcanza la solución óptima en cada ocasión que se ejecuta el procedimiento, por lo tanto se define como criterio alternativo de paro alcanzar dicho número de iteraciones.

Repeticiones\Re	Promedio	Calificación 1	Calificación 2	Calificación 3	Calificación 4	Calificacion 5	Tiempo(segundos)
0	20	20	20	20	20	20	0.19
5000	2	2	0	2	2	4	2.9
10000	0,4	0	0	2	0	0	4.3
15000	0,4	0	2	0	0	0	8.3
20000	0	0	0	0	0	0	10.8
25000	0	0	0	0	0	0	13.2
30000	0	0	0	0	0	0	15.6
35000	0	0	0	0	0	0	18.6
40000	0	0	0	0	0	0	22,00
45000	0	0	0	0	0	0	25.1
50000	0	0	0	0	0	0	27.9

Figura 1.3: Determinación experimental del criterio de paro

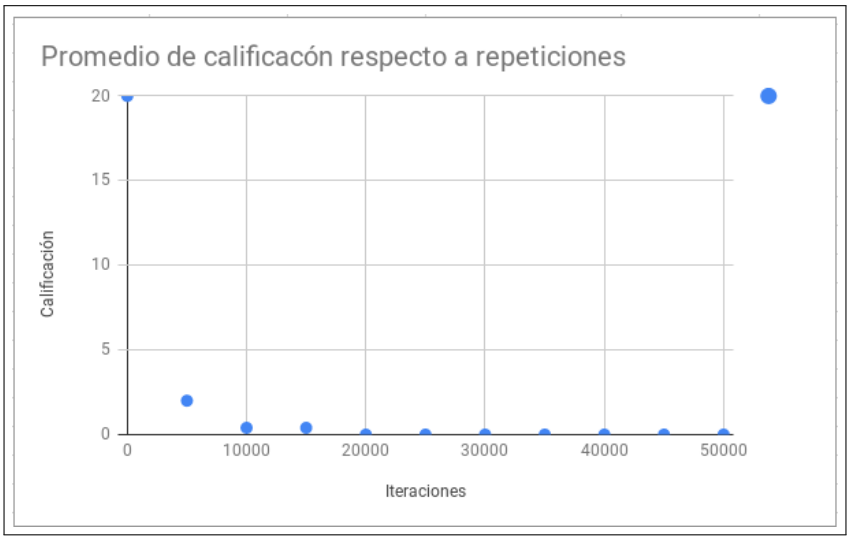


Figura 1.4: Gráfica de mejora

1.1.6. Pruebas de la versión: 3

En esta sección se describen los resultadods de las pruebas realizadas de la tercera versión del algoritmo de optimización de horarios.

Como consideración tenemos que las pruebas fueron realizadas en una laptop dell Inspiron 15 con procesador intel i5 5ta generación 4gb de memoria ram usando el sistema operativo debian 9.

Pruebas realizadas

Inicialmente las funciones que comprenden el algoritmo fueron probadas por separado.

- Se corroboró que la función de generar Población utilizara todas las opciones de Materia-Profesor disponibles y que el resultado tuviera la estructura que hemos definido.
- Se probó la función de evaluación, para corroborar que las restricciones ingresadas sean tomadas en cuenta al momento de asignar una calificación a un horario.
- Se probaron las funciones de mutación de forma que el resultado arrojado después de llevar a cabo dichos operadores sea en realidad distinto a la entrada del mismo así como se corroboró que se llevaran a cabo de la manera esperada.

Una vez que se corrigieron los errores arrojados por las pruebas de la versión 2 del algoritmo, se adaptó el segundo prototipo de manera que la función principal reciba como parámetros, el número de iteraciones que se van a realizar, el número de individuos que conforman la población inicial y el número de estructuras educativas que el actor desea como resultado del algoritmo. Para corroborar que el funcionamiento fuera correcto, utilizamos la estructura educativa actual de la ESCOM para el nivel 2 y el turno matutino. Utilizando estos datos y teniendo en cuenta los resultados de las pruebas del prototipo 2, se realizó una experimentación para determinar el tamaño óptimo de la población inicial.

En una junta con el profesro Iván Giovanny Mosso, se le cuestionó sobre el número de estructuras educativas que le gustaría tener disponibles para analizar cuál es la mejor opción a lo que respondió 5, de esta manera para asegurarnos de que este requerimiento se satisfaga y siga siendo sostenible a futuro, determinamos permitir que se solicite al sistema arrojar un máximo de 10 estructuras educativas, de esta manera las pruebas fueron llevadas a cabo considerando el número máximo posible de estructuras educativas deseadas como resultado.

El tamaño de la población inicial influye de manera directa en el aumento tiempo que tarda en ejecutarse el algoritmo, sin embargo de igual manera influye en la disminución del número de iteraciones requeridas para lograr 10 estrucutras educativas. En cinco ocasiones se tomó el número de iteraciones que toma al algoritmo encontrar 10 posibilidades de estructura educativa con calificación óptima para cada uno de los tamaños probados para la población inicial. En la figura 1.5 se muestran los resultados del experimento y en la figura 1.6 se muestra la gráfica de los mismos donde se puede apreciar que el número iteraciones disminuye conforme el tamaño de la población inicial aumenta, sin embargo también se puede observar en la tabla el aumento en el tiempo. De esta manera se determinó que el punto medio entre el aumento de tiempo y la disminución de las iteraciones debe ser tomado como el mejor para nuesto algoritmo por lo que se tomó un tamaño de la población inicial de 30 individuos.

Tamaño de la población inicial	Promedio	Iteraciones 1	Iteraciones 2	Iteraciones 3	Iteraciones 4	Iteraciones 5	Tiempo(segundos)
10	6822	7406	9429	6548	4551	6176	25.3
15	3870	2820	3994	3984	4117	4435	32.5
20	3431,8	3995	3262	3351	4049	2502	35.4
25	2950	2497	3195	2909	3047	3102	47.1
30	2696,8	2513	2496	3032	2722	2721	47.1
35	2694,6	2754	2257	2820	2844	2798	64
40	2269,4	2266	2181	2093	2391	2416	65
45	2266,6	2190	2271	2371	2281	2220	68
50	1972,8	2111	2184	1632	1877	2060	71

Figura 1.5: Determinación experimental del criterio de paro

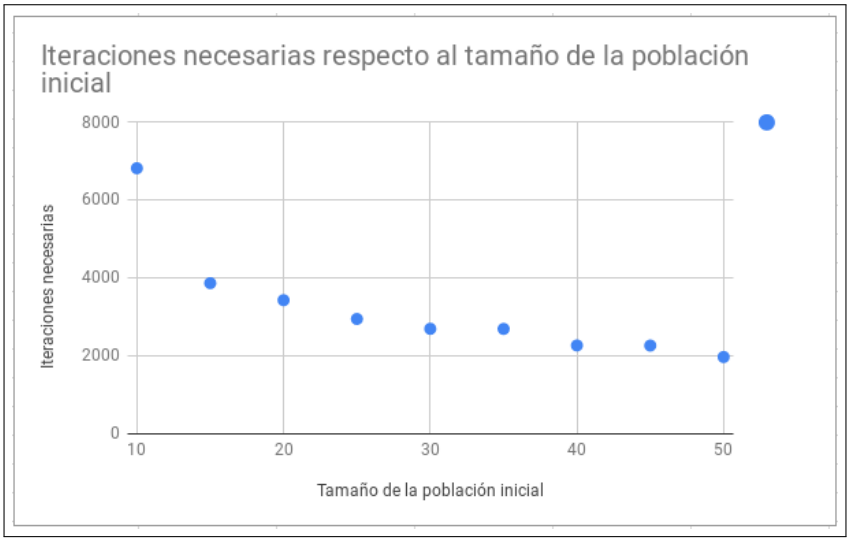


Figura 1.6: Gráfica de mejora

CAPÍTULO 2

Bibliografía

[1] https://www.python.org/ [2] https://www.latex-project.org/ [3] https://en.wikipedia.org/wiki/Balsamiq
[4] https://www.applesfera.com/aplicaciones-os-x-1/balsamiq-mockup-una-muy-buen-herramienta-para-esbozar-tus-futuras-apps [5] https://prezi.com/lxqgnl0h5m/que-es-staruml/ [6] https://www.djangoproject.com/
[7] https://developer.mozilla.org/es/docs/HTML/HTML5 [8] https://developer.mozilla.org/es/docs/Web/CSS/CSS3
[9] http://gs.statcounter.com/browser-market-share/all/mexico [10] http://gs.statcounter.com/os-market-share/desktop/worldwide [11] https://www.javascript.com/ [12]http://desarrollowebydesarrolloweb.blogspot.mx/2015/02/ta
comparativa-de-los-lenguajes-de.html [13]http://noticias.universia.com.ar/consejos-profesionales/noticia/2016/02/22/1136
cuales-lenguajes-programacion-populares.html

[14]C. A. C. Coello, "Introducción a la computación evolutiva," Notas del curso. Departamento de Ingeniería Eléctrica, Sección de Computación, Instituto Politécnico Nacional, México, 2004.

[15]Macías Duarte Carlos Antonio, .Análisis comparativo del desempeño de Técnicas Evolutivas aplicadas a la predicción de distribución de robos", Instituto Politécnico Nacional, México, 2016.

[16]Gregorio Toscano Pulido, .Optimización Multiobjetivo Usando un Micro Algoritmo Genético", Universidad Veracruzana, México, 2001.

[17]L. Araujo and C. Cervignón, .Algoritmos evolutivos: un enfoque práctico", Alfaomega, 2009.

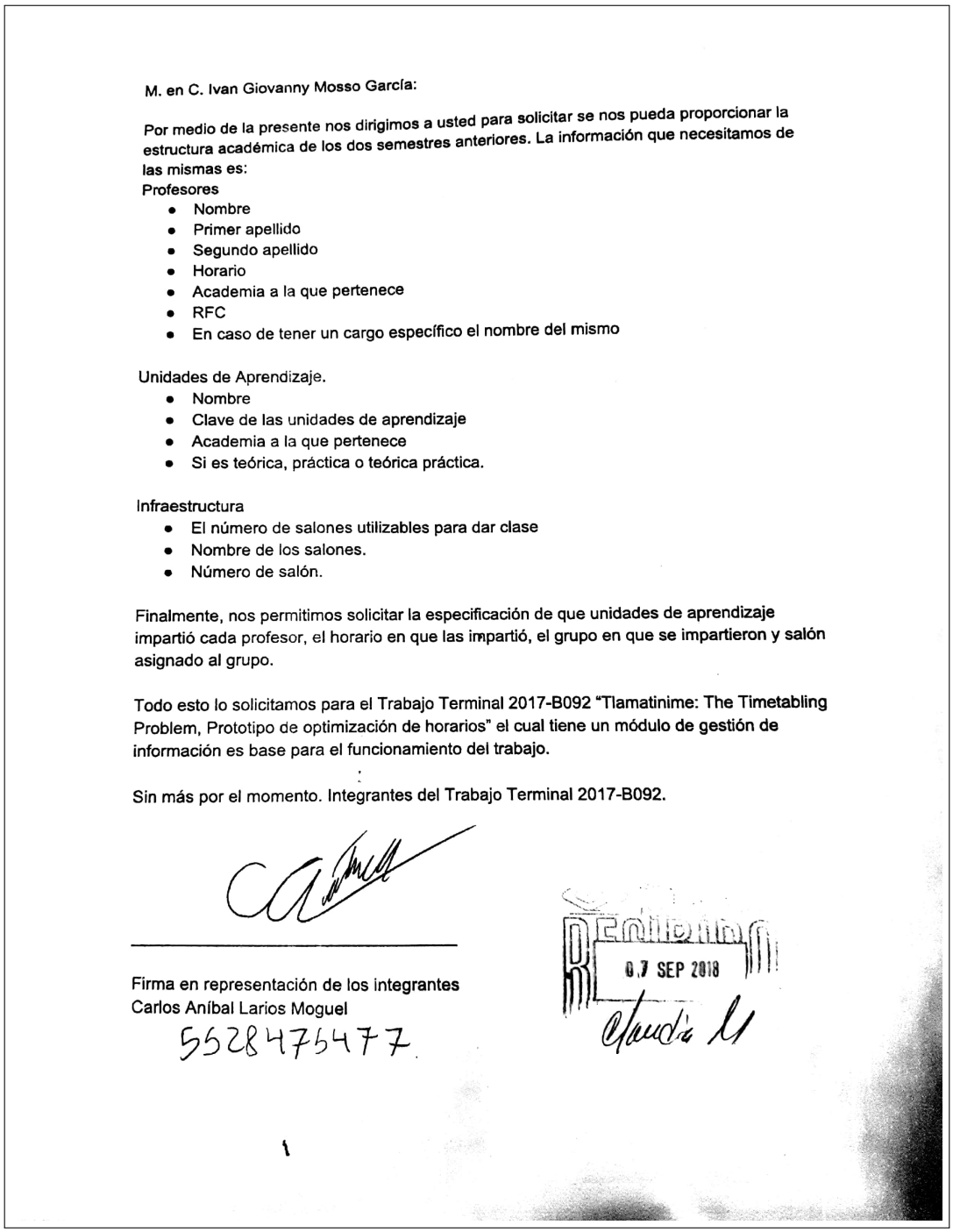
CAPÍTULO 3

Anexos

3.1. Anexo 1. Información recabada

Por medio de el documento mostrado en la imagen 3.1 se solicitó al subdirector académico el Maestro en Ciencias Iván Giovanny Mosso García la información de la estructura académica del semestre anterior y del semestre actual, así como los siguientes datos:

- Profesores:
  - Nombre
  - Primer Apellido
  - Segundo Apellido
  - Horario
  - Academia a la que pertenece
  - RFC
  - Nombre del cargo(en caso de tener alguno)
- Unidades de aprendizaje:
  - Nombre
  - Clave
  - Academia a la que pertenece
  - Tipo de unidad de aprendizaje(Teórica, Práctica, Teórica-Práctica)
- Infraestructura:
  - Número de salones utilizables
  - Nombre de los salones
  - Número de salón



M. en C. Ivan Giovanny Mosso García:

Por medio de la presente nos dirigimos a usted para solicitar se nos pueda proporcionar la estructura académica de los dos semestres anteriores. La información que necesitamos de las mismas es:

Profesores

- Nombre
- Primer apellido
- Segundo apellido
- Horario
- Academia a la que pertenece
- RFC
- En caso de tener un cargo específico el nombre del mismo

Unidades de Aprendizaje.

- Nombre
- Clave de las unidades de aprendizaje
- Academia a la que pertenece
- Si es teórica, práctica o teórica práctica.

Infraestructura

- El número de salones utilizables para dar clase
- Nombre de los salones.
- Número de salón.

Finalmente, nos permitimos solicitar la especificación de que unidades de aprendizaje impartió cada profesor, el horario en que las impartió, el grupo en que se impartieron y salón asignado al grupo.

Todo esto lo solicitamos para el Trabajo Terminal 2017-B092 "Tlamatinime: The Timetabling Problem, Prototipo de optimización de horarios" el cual tiene un módulo de gestión de información es base para el funcionamiento del trabajo.

Sin más por el momento. Integrantes del Trabajo Terminal 2017-B092.

Firma en representación de los integrantes
Carlos Aníbal Larios Moguel
5528475477

07 SEP 2018

Figura 3.1: Solicitud de información

En respuesta el profesor Iván nos proporcionó dos archivos en formato excel. El primero contiene las siguientes columnas con información referente a la estructura educativa:

- Profesor
- Departamento
- Grupo
- Unidad de aprendizaje
- Academia
- Salón
- Laboratorio
- Lunes
- Martes
- Miércoles
- Jueves
- Viernes

De este formato se cuenta con dos páginas, una para la estructura del semestre 2018/2 que cuenta con 459 filas y otra para la estructura del semestre 2019/1 que cuenta con 474 filas. Cada fila de este formato representa la combinación de la información de todas las columnas señaladas previamente.

De este archivo la única columna que no utilizamos es la información del departamento al que pertenece el profesor y la información de los laboratorios en que se imparte la unidad de aprendizaje debido a que esta gestión en particular fue dejada fuera del alcanze del presente trabajo.

El segundo archivo que se nos proporcionó contiene la información de 306 unidades de aprendizaje. Las columnas de este archivo son:

- Clave
- Nombre de la unidad de aprendizaje
- Abreviatura de la unidad de aprendizaje
- Nivel
- Academia a la que pertenece
- Horas teoría
- Horas prácticas
- Plan al que pertenece
- Carrera a la que pertenece

De este archivo tenemos que las horas prácticas y teóricas debemos sumarlas para obtener las horas totales de una unidad de aprendizaje que es lo que realmente necesitamos. Sin embargo no nos compete la información de el plan y la carrera a la que pertenecen ya que sólo contemplamos el plan de estudios actual de la ESCOM para la carrera de Ingeniería en Sistemas Computacionales.