KhronoPol
Time Planning System for the Facultad Politécnica UNA

Rodrigo Fernández
Facultad Politécnica
Universidad Nacional de Asunción
Paraguay
rrf.marin@gmail.com

Marcelo Franco
Facultad Politécnica
Universidad Nacional de Asunción
Paraguay
marcelodf12@gmail.com

Diego P. Pinto-Roa
Facultad Politécnica
Universidad Nacional de Asunción
Paraguay
dpinto@pol.una.py

***Abstract*—This work presents an integral solution for the administration and automatic generation of class schedules and class assignment denominated KhronoPol. The system developed is based on a modular architecture, multiuser and accessible from the Internet with a high degree of maintainability. Given the complexity that the problem can reach KhronoPol has 2 phases: in the first phase a Genetic Algorithm seeks the optimal allocation of time and subjects, and the second phase a Backtracking Algorithm performs the assignment of classrooms to previously calculated subjects. It has been taken as a case study the schedule of the Facultad Politécnica - Universidad Nacional de Asunción (FPUNA) in which two real scenarios are posed: (a) initial scheduling, and (b) rescheduling minimizing changes considering new sections and changes of teachers. The FPUNA consists of 16 undergraduate degrees, 1.000 courses per semester, 60 classrooms of different capacities, and 300 teachers.**

1. Introducción

Cualquier institución educativa de nivel universitario, tiene la problemática de la planificación horaria, en especial cuando las carreras que ofrece son numerosas. La carga operativa del mantenimiento y modificación es elevada.

La Facultad Politécnica de la Universidad Nacional de Asunción (FPUNA) no es ajena a esta situación, actualmente cuenta con 16 carreras de grado, y posee más de 2000 asignaturas, que son impartidas en dos semestres.

Estas asignaturas deberán ser calendarizadas para dictarse semanalmente, respetando la restricción de turno, disponibilidad de profesores, y no solapamiento con asignaturas del mismo semestre de la misma carrera.

Además, el horario de clase puede ser dinámico al inicio de cada semestre, y se necesitan mecanismos para la actualización y publicación de los cambios en tiempo real. Actualmente la FPUNA no cuenta con una herramienta que satisfaga en plenitud estas necesidades, por lo cual, en este trabajo proponemos una solución para esta problemática.

1.1. La problemática, caso FPUNA

La Facultad Politécnica de la Universidad Nacional de Asunción ofrece 16 carreras de grado, cada una de ellas posee en promedio de 50 asignaturas, existen carreras que son impartidas en 3 turnos diferentes (mañana, tarde y noche) las cuales se dictan de lunes a viernes entre las 7:30 hs. y 22:15 hs. y sábados entre las 7:30 hs, y 17:00 hs. Las asignaturas tienen una duración medida en horas cátedras, cada hora cátedra equivale a 45 minutos reloj y los recesos entre asignaturas diferentes son de 15 minutos. Para un mejor aprovechamiento de tiempo y espacio, dividimos el tiempo en slots de 15 minutos. Por lo que una hora cátedra equivale a 3 slots. Las asignaturas son agrupadas por semestres. Las de semestre impar son dictadas durante el primer semestre del año, las de semestre par durante el segundo semestre del año y las asignaturas más demandantes son habilitadas en ambos semestres.

La infraestructura edilicia cuenta con 60 aulas de diferentes capacidades, y con diferentes requerimientos (aulas normales, laboratorios de informática, sala de dibujo), y una asignatura no puede ser asignada a un aula que no cuente con la capacidad para albergar a todos los alumnos inscriptos en dicha asignatura, tampoco puede impartirse más de una asignatura en una misma aula, en un mismo periodo de tiempo. El horario es elaborado antes del inicio de las inscripciones de cada periodo de clases, por eso no se posee información sobre la cantidad de alumnos por asignatura, por lo cual la asignación de aulas se realiza una vez terminada el periodo de inscripción.

Cada asignatura ya cuenta con un profesor asignado para impartir dicha asignatura, y es claro que un profesor no puede impartir más de una asignatura en el mismo periodo de tiempo, y debe considerarse que puede existir periodos de tiempo en el que el profesor no está disponible para enseñar. Una asignatura puede corresponderse a una o más carreras, y dentro de cada carrera a un semestre diferente, además de ello pueden existir más de una sección por cada asignatura. Las asignaturas del mismo semestre y la misma carrera no pueden impartirse durante el mismo periodo de tiempo, por lo que cada sección posee un conjunto-conflicto que corresponde a las secciones que no pueden impartir en

TABLE 1. TABLA DE SÍMBOLOS.

$\in$	Pertenece
$\{\}$	Conjunto
$\langle\rangle$	Tupla
$\mid$	Tal que
$\subset$	Sub conjunto
$\wedge$	Conjunción

el mismo periodo de tiempo que ella. Una asignatura que posea más de una sección puede requerir que sus secciones sean agrupadas en periodos de tiempo iguales, por lo cual, cada sección posee un conjunto-grupo que corresponde a las secciones que deben impartirse en el mismo periodo de tiempo que ella.

1.2. Modelo matemático general para la FPUNA.

1.2.1. Constantes.

- Tn: Cantidad total de slots times. El tiempo es dividido en 6 días de la semana, y cada día de la semana es dividido a su vez en intervalos de 15 minutos, desde las 7:00 hs hasta las 23:00 hs (64 slots por día), totalizando así 6x64 = 384 slots de tiempo. Por lo tanto, para este caso Tn = 384
- Cn: Cantidad total de características de aulas.
- An: Cantidad total de aulas.
- Pn: Cantidad total de profesores.
- Sn: Cantidad total de asignaturas.
- T: Conjunto de slots times, $T = \{t_1, \ldots, t_{Tn}\}$
- C: Conjunto de característica de aula, $C = \{c_1, \ldots, c_{Cn}\}$
- A: Conjunto de aulas, $A = \{a_1, \ldots, a_{An}\}$
- A': Conjunto de tuplas $A' = \{a'_1, \ldots, a'_{An}\}$ con $h_i > 0$ y $a'_i = \langle a_1, h_1, c_1\rangle$ donde h_i representa la capacidad del aula i y c_i la característica del aula.
- P: Conjunto de profesores, $P = \{p_1, \ldots, p_{Pn}\}$
- P': Conjunto de tuplas de $P' = \{p'_1, \ldots, p'_{Pn}\}$ con $p'_i = \langle p_i, D_i\rangle$ donde
- D_i: Conjunto de n slots time que el profesor i tiene disponible $D_i = \{t_{j_1}, \ldots, t_{j_n}\}$
- S: Conjunto de asignaturas, $S = \{s_1, \ldots, s_{Sn}\}$
- $S\prime$: Conjunto de tuplas, $S\prime = \{s'_1, \ldots, s'_{Sn}\}$ donde $s'_i = \langle s_i, p_i, c_i, q_i, d_i, D_i\rangle$ con
- $q_i > 0$ que representa la cantidad de alumnos que tiene la asignatura s_j y
- $d_i > 0$ que indica la cantidad slot time necesarios para la asignatura s_j
- Un conjunto excluyente de En tuplas $\langle s_i, s_j\rangle$ donde $s_i, s_j \in S$
- Un conjunto agrupador de Gn tuplas $\langle s_i, s_j\rangle$ donde $s_i, s_j \in S$.

1.2.2. Variables.

- Un conjunto horario de Sn tuplas $\langle s_j, t_j, a_j\rangle$.

1.2.3. Restricciones.

- **R-01**: Un profesor no puede impartir más de una clase de una asignatura por slot time.
- **R-02**: Una asignatura solo puede ser asignada a un aula con la misma característica requerida.
- **R-03**: Una asignatura debe ser asignada a N slots time consecutivos, donde N es la cantidad de slots time requeridas por la asignatura.
- **R-04**: Un profesor no puede ser asignado a un horario que no tiene disponible.
- **R-05**: Una asignatura no puede ser asignada a un horario que no sea de su turno.
- **R-06**: En un aula no pueden impartirse clases de dos asignaturas en el mismo slot time.
- **R-07**: Para cada tupla del conjunto excluyente, las asignaturas s_i y s_j no pueden ser asignadas al mismo slot time
- **R-08**: Para cada tupla del conjunto agrupador, las asignaturas s_i y s_j deben ser asignadas exactamente en los mismos slots de tiempo.

2. Herramientas Existentes

Reconocimos 16 requerimientos que deben ser cubiertos para resolver la problemática, y realizamos un análisis comparativo de las herramientas de software ya existentes. Clasificamos los requerimientos en dos: Los requerimientos básicos que están relacionados a la resolución del problema de calendarización en sí mismo y un segundo tipo de requerimientos que son los Complementarios, que están relacionados a la usabilidad, mantenimiento y comunicación.

En la Tabla 2 se presenta una comparativa entre las herramientas existentes: 3 herramientas de pagos (Untis [1], Kronowin [2] y GHC [3]), y 2 herramientas gratuitas y de código abierto (FET [4], TABLIX [5]). Como se puede apreciar las herramientas de pago cubren la mayoría de las necesidades complementarias ya que al ser un producto de pago se centran mucho en la experiencia del usuario. Sin embargo, están más bien pensados para problemas pequeños y por ende no cumplen al 100% con los requerimientos básicos.

Por otro lado, las herramientas OpenSource, pueden cubrir los requerimientos básicos, sin embargo, la experiencia del usuario es mala en dichas herramientas, y no tienen soporte para publicarlos como servicios y puedan ser consumidos desde Internet.

3. Trabajos Relacionados

En la literatura existente, varios trabajos ya han abordado el problema de calendarización. Algunos trabajos resuelven el problema con algoritmos deterministas como [6] y [7] que utilizan técnicas de búsqueda local, luego existen otros trabajos que atacan el problema con algoritmos no deterministas con metaheurísticas como: Algoritmos Genéticos [11] [12] y [13], Colonia de Hormigas [9] y [10], y Enjambre de

TABLE 2. COMPARACIÓN DE HERRAMIENTAS.

Software	Untis	Kronowin	GHC	FET	TABLIX
Requerimientos Basicos					
Planificación de horario	SI	SI	SI	SI	SI
Asignar aulas	SI	SI	SI	SI	SI
Reportes grupo de alumnos	SI	SI	SI	SI	SI
Reportes por profesores	SI	SI	SI	SI	SI
Reportes por aulas	SI	SI	SI	SI	SI
Soporte para mas de 1000 asignaturas	SIN INFO	SIN INFO	SIN INFO	SI	SIN INFO
Granularidad 15 minutos	NO	SIN INFO	SIN INFO	SI	NO
Requerimientos Complementarios					
Multiusuario	SI	NO	NO	NO	NO
Usabilidad	SI	NO	SI	NO	NO
Multiplataforma	NO	NO	NO	SI	NO
Autogestión de disponibilidad para profesores	SI	NO	SI	NO	NO
Optimización de recursos computacionales	SIN INFO	NO	SI	NO	NO
Disponibilidad en línea	SI	NO	SI	NO	NO
Notificaciones por correo u otro medio	SI	NO	NO	NO	NO
Migración Inicial Sencilla	NO	NO	NO	NO	NO

particulas (PSO) [14] [15] Según [7] se puede clasificar los problemas de calendarización en 3 grandes grupos:

- **CASO 1 - Horario de exámenes:** En este tipo de problema consiste en calendarizar los exámenes.
- **CASO 2 - Calendarización basado en matrículas:** Este tipo de problema intenta simular un caso de la vida real cuando los estudiantes eligen que materias toman, y el horario es construido según estas elecciones.
- **CASO 3 - Calendarización basada en curriculum:** Programación semanal de clases para universidades dentro de una número determinado de aulas y un periodo de tiempo. El conflicto entre los cursos está determinado por el currículum publicado por la Universidad, y no hay datos de matriculación.

Para el caso de estudio de este trabajo, nos encontramos con una Calendarización basada en curriculum. Pero con la particularidad de que en la FPUNA existen asignaturas que pertenecen a más de una carrera, a su vez pueden habilitarse más de una sección de una asignatura, y cada una de estas secciones no puede coicidir con ninguna otra sección de las asignaturas del semestre a la que corresponde en cada carrera. A esto se suma la complejidad que horario de clase no está estructurado en divisiones de tiempo uniformes. Por ejemplo: en una carrera, las clases pueden empezar a las 7:30 AM, pero en otra podrían empezar a las 8:00 AM y las asignaturas comunes entre estas carreras deben tener el mismo horario. Los trabajos citados no contemplan este tipo de complejidad. Para resolver esta particularidad, en este trabajo dividimos el tiempo en slots de 15 minutos, donde cada asignatura ocupa más de un slot, con esto también logramos optimizar el uso del tiempo.

4. Algoritmo Genético

Los Algoritmos Genéticos constituyen una técnica de búsqueda y optimización, altamente paralela, inspirada en el principio Darwiniano de selección natural y reproducción genética. [8]

Un Algoritmo Genético produce un conjunto de generaciones, en cada generación se tiene una población de N individuos. Los individuos son un conjunto de genes (la estructura de los individuos depende del problema). Cada individuo o cromosoma, representa a una posible solución del problema. Para avanzar a la siguiente generación, se efectúan las siguientes operaciones sobre la población:

- Selección: Se evalúa a cada individuo para determinar la cantidad de descendientes que podrá tener.
- Reproducción: Se toman los genes de 2 individuos de la población para generar 2 descendientes.
- Mutación: Se altera, de forma aleatoria, M genes de 1 descendiente.
- Fitness: Función que determina la calidad de una solución.
- Promoción: Función que determina si un individuo avanza o no a la siguiente generación.

Se mantienen N individuos con el mejor fitness. Y se repite el proceso.

5. Propuesta de Solución

Debido a que, en la FPUNA, no se posee información de la cantidad de alumnos inscriptos por asignatura, ya que el horario es elaborado y publicado semanas antes del periodo de inscriciones, por lo que decidimos dividir el problema en fases. En una fase cero realizamos un pre-procesamiento, luego en la fase uno se calcula el horario y en la fase dos se asignan las aulas.

Lunes	Martes	Miércoles	Jueves	Viernes	Sábado
1	65	129	193	257	321
2	66	130	194	258	322
3	67	131	195	259	323
4	68	132	196	260	324
5	69	133	197	261	325
6	70	134	198	262	326
7	71	135	199	263	327
8	72	136	200	264	328
9	73	137	201	265	329
10	74	138	202	266	230
11	75	139	203	267	131
12	76	140	204	268	32
13	77	141	205	269	-67
...	...	...	...	...	...
64	128	192	256	320	384

Slot posibles Slot prohibidos

Figure 1. Ejemplo de la configuración de slots posibles para una asignatura

Lunes	Martes	Miércoles	Jueves	Viernes	Sábado
1	65	129	193	257	321
2	66	130	194	258	322
3	67	131	195	259	323
4	68	132	196	260	324
5	69	133	197	261	325
6	70	134	198	262	326
7	71	135	199	263	327
8	72	136	200	264	328
9	73	137	201	265	329
10	74	138	202	266	230
11	75	139	203	267	131
12	76	140	204	268	32
13	77	141	205	269	-67
...	...	...	...	...	...
64	128	192	256	320	384

Slot posibles Slot prohibidos

Figure 2. Ejemplo de la configuración de slots disponibles de un profesor

Planteamos dos estrategias de Solución, en la fase uno utilizamos programación lineal entera mediante un algoritmo de ramificación y acotamiento [16] y la otra un algoritmo genético. Desde la interfaz gráfica se podrá escoger cual algoritmo utilizar.

Para la fase dos del problema. debido a que existen diferentes tipos de aulas (aulas normales, laboratorios de informática, laboratorio de electricidad, salas de dibujo) dividimos el problema en subproblemas independientes según el tipo de aula, que luego solucionamos con programación lineal entera.

6. Pre-procesamiento

En esta etapa, se realizan tres modificaciones al modelo general propuesto en la sección 1.2. Estas modificaciones se realizan para simplicar el modelo y son la siguientes:

6.1. División de Asignaturas

Esta división es necesaria debido a que cada asignatura puede ser dictada una, dos, tres o incluso cuatro veces por semana, por lo que para simplificar el modelo, consideramos cada división de la misma asignatura como una asignatura independiente que tienen restricción excluyentes E entre ellas y no pueden impartirse en el mismo horario. Para la situación particular de la Facultad Politécnica, se presentan mayormente asignaturas que deben impartirse al menos dos veces por semana, por lo que las $Sn = 1.000$ asignaturas se transforman en $Sn = 2.000$ asignaturas.

6.2. Eliminar restricciones de profesores

Esta simplificación se logra realizando una intersección de cada conjunto de slots de asignatura con el respectivo slot de disponibilidad del profesor que enseña dicha asignatura.

En la figura 1 se puede observar la configuración de slots de tiempos en las que una asignatura puede ser calendarizada, cada asignatura posee varios ”subconjuntos de slots de tiempo consecutivos posibles” según el turno. Para modelar N slots de tiempos consecutivos se necesitan $N+1$ varibales auxiliares, y $2 \cdot N+1$ restricciones. Para el ejemplo de la figura donde existe un solo subconjunto por día, se necesitan crear 6 variables auxiliares, y 13 restricciones más por cada asignatura.

- $t_x \geq lun_0 - M\sigma_1$
- $t_x \leq lun_f + M\sigma_1$
- $t_x \geq mar_0 - M\sigma_2$
- $t_x \leq mar_f + M\sigma_2$
- $t_x \geq mie_0 - M\sigma_3$
- $t_x \leq mie_f + M\sigma_3$
- $t_x \geq jue_0 - M\sigma_4$
- $t_x \leq jue_f + M\sigma_4$
- $t_x \geq vie_0 - M\sigma_5$
- $t_x \leq vie_f + M\sigma_5$
- $t_x \geq sab_0 - M\sigma_6$
- $t_x \leq sab_f + M\sigma_6$
- $\sum_{i=1}^{6} \sigma_i = 1$

Donde:

- $lun_0 = 1$
- $lun_f = 8$
- $mar_0 = 65$
- $mar_f = 72$
- $mie_0 = 129$
- $mie_f = 136$
- $jue_0 = 193$
- $jue_f = 200$
- $vie_0 = 257$
- $vie_f = 264$
- $sab_0 = 321$
- $sab_f = 328$

En la figura 2 se puede observar la configuración de slots de tiempos disponibles de un profesor. Del mismo modo para este ejemplo, como existen 3 subconjuntos de slots de tiempo consecutivos, se necesitan 4 variables y 9 restricciones.

Si consideramos estos dos grupos por separados, para el ejemplo tendríamos un total de 9 variables auxiliares y

Lunes	Martes	Miércoles	Jueves	Viernes	Sábado
1	65	129	193	257	321
2	66	130	194	258	322
3	67	131	195	259	323
4	68	132	196	260	324
5	69	133	197	261	325
6	70	134	198	262	326
7	71	135	199	263	327
8	72	136	200	264	328
9	73	137	201	265	329
10	74	138	202	266	230
11	75	139	203	267	131
12	76	140	204	268	32
13	77	141	205	269	-67
...	...	...	...	...	...
64	128	192	256	320	384

Slot posibles Slot prohibidos

Figure 3. Ejemplo de la intersección de slot de asignatura con slot profesor

22 restricciones. Sin embargo si realizamos una intersección de estos slots de tiempos, y lo consideramos como uno solo podemos ver en la figura 3 solo 3 subconjuntos de slots de tiempo consecutivos, lo que representa 4 variables y 9 restricciones frente a 9 variables auxiliares y 22 restricciones.

Ademas, en caso que un profesor dicte mas de una asignatura, se debe agregar una restricción excluyente E entre cada par de asignaturas dictadas por el mismo profesor, siempre y cuando no exista una restricción de grupo G en dichas asignturas.

6.3. Agrupamiento

Es posible agrupar las asignaturas que tengan una restricción de grupo G, y considerarlas como una sola asignatura. Al unir dos asignaturas, la asignatura resultante debe heredar los conflictos de ambas asignaturas padres. Por ejemplo:

Si tenemos las asignaturas A, B, C y D. La asignatura A tiene una restricción excluyente E con C y la asignatura B tiene una restricción excluyente E con D. Si las asignaturas A y B tienen una restricción de grupo G, entonces ambas se fusionan formando una asignatura AB y tiene restricción excluyente E con las asignatura C y D.

7. Fase 1: Cálculo de Horario

7.1. Algoritmo Genético para el cálculo de horario

Para el caso de la calendarización se tiene un conjunto Sn de asignaturas y a cada asignatura debe ser asignada a un slot de tiempo t. La representación que utilizamos para un cromosoma (solución) es un arreglo de enteros, donde el índice del arreglo representa la asignatura, y cada elemento del arreglo representa el primer slot de tiempo t asignado a dicha asignatura.

En la figura 4 se puede apreciar un ejemplo de cromosoma. Así esta solución, significa que la asignatura 0 se le asignó como slot inicial el 7, a la asignatura 1 el slot 1 y así sucesivamente.

Slot inicial →	7	1	1	3	...	5
Asignatura →	*0*	*1*	*2*	*3*	*…*	*N-1*

Figure 4. Estructura de un cromosoma

En Algorithm 1 se puede apreciar el algoritmo genético básico que utilizamos para el cálculo de horario.

Algorithm 1 Algoritmo Genético básico utilizado.

```
1:  procedure ALGORITMOGENETICO
2:      poblacion ← generarPoblacion()
3:      calcularFitness(poblacion)
4:      repeat
5:          seleccion(poblacion)
6:          cruzar(poblacion)
7:          mutar(poblacion)
8:          calcularFitness(poblacion)
9:          promocion(poblacion)
10:         generacion ← generacion + 1
11:         converge ← calcularConvergencia()
12:     until converge
13: end procedure
```

7.1.1. Función Fittness. Como se puede apreciar en la figura 5.b en esta solución se tienen 4 conflictos, y en la figura 5.a se tiene 8 conflictos, pero para esta segunda solución bastaría con reubicar la asignatura S1, para tener una solución sin conflictos, mientras que en la primera solución se tienen que reubicar las asignaturas S1, S2, S3 y S4. Esto nos indica que si utilizamos como función fitness solo la cantidad de conflictos una solución la solución a de 8 conflictos resultaría peor que la solución b de 4 conflictos. Por ello utilizamos como valor fitness la "Cantidad de asignaturas a reubicar para tener cero conflictos" (Algorithm 2). Donde la solución es mejor que otra si su valor fitness es menor.

El procedimiento para calcular la Cantidad de Asignaturas a reubicar consiste en: primero marcar todas las asignaturas como "Colocadas", luego buscar la asignatura más conflictiva, que es aquella que posee el mayor número de conflictos (Algorithm 3). Luego esta asignatura es marcada como "No colocada", se repite la búsqueda de la asignatura más conflictiva entre las colocadas hasta que la asignatura con mayor conflicto posea cero conflictos. Luego se retorna la cantidad de asignaturas "No colocadas". En la figura 6 se puede apreciar un ejemplo, con una solución de 8 asignaturas, el fitness de esta solución es 3, ya que son 3 la cantidad mínimas de asignaturas a reubicar para tener cero conflictos.

7.1.2. Población. La población esta formada por N individuos, donde cada individuo representa una solución posible para el problema. La población se mantiene constante durante cada generación gracias a la función de promoción.

Algorithm 2 Calcular cantidad de asignaturas a reubicar para obtener cero conflictos

```
1:  procedure CALCULARFITNESS
2:      for all asig in asignaturas do
3:          asig.colocada ← true
4:      end for
5:      repeat
6:          maxConflicto ← 0
7:          for all asig in asignaturas do
8:              if asig.colocada then
9:                  asig.colocada ← false
10:                 conflictos ← nroConflictos(asig)
11:                 if conflictos > maxConflicto then
12:                     masConflictiva ← asig
13:                     maxConflicto ← conflictos
14:                 end if
15:                 asig.colocada ← true
16:             end if
17:         end for
18:         if maxConflicto > 0 then
19:             masConflictiva.colocada ← false
20:         end if
21:     until maxConflicto == 0
22:     aReubicar ← 0
23:     for all asig in asignaturas do
24:         if Not asig.colocada then
25:             aReubicar ← aReubicar + 1
26:         end if
27:     end for
        return aReubicar
28: end procedure
```

Algorithm 3 Contar la cantidad de conflictos de una asignatura

```
1:  procedure NROCONFLICTOS(asig)
2:      conflictos ← 0
3:      for all asig2 in asig.excluyentes do
4:          if asig2.colocado then
5:              if asig.slot == asig2.slot then
6:                  conflictos ← conflictos + 1
7:              end if
8:          end if
9:      end for
10:     for all asig2 in asig.agrupador do
11:         if asig2.colocado then
12:             if not asig.slot == asig2.slot then
13:                 conflictos ← conflictos + 1
14:             end if
15:         end if
16:     end for
        return conflictos
17: end procedure
```

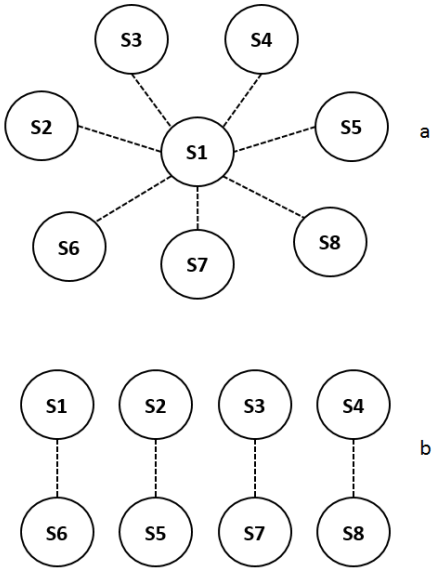

Figure 5. Grafo de conflictos de 2 soluciones.

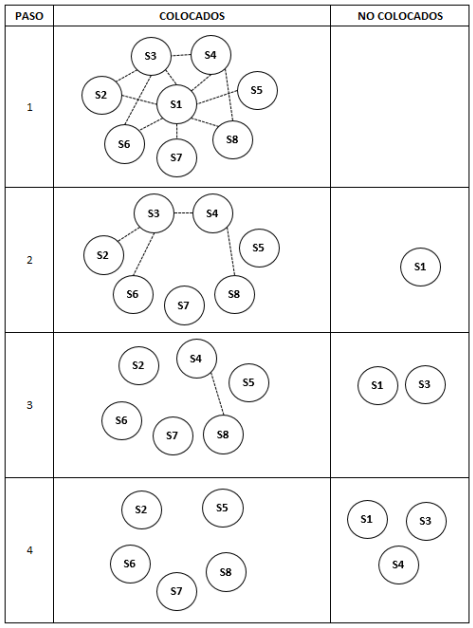

Figure 6. Ejemplo del Cálculo del fitness.

7.1.3. Selección. Para la operación de selección utilizamos el torneo binario [8]. En cada torneo se selecciona 2 individuos al azar, y resulta ganador del torneo aquel que tenga mejor (menor) fitness, con una probabilidad $P < 5\%$ a que la peor solución gane el torneo. El ganador del torneo tiene derecho a tener un descendiente. Se realizan N torneos (donde N es igual al tamaño de la población), generándose así N descendientes.

7.1.4. Reproducción. Se eligen 2 descendientes al azar, y se cruzan sus genes en forma aleatoria. Resultando 2 individuos

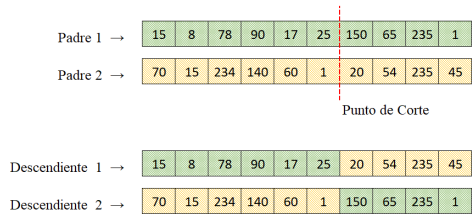

Figure 7. Reproducción mediante punto de corte

que comparten los genes de ambos padres. Este proceso se repite hasta que todos los descendientes se hayan cruzado con otro descendiente. El método que utilizamos es el punto de corte, el método se puede ver en la figura 7 en donde se elige un punto de corte aleatorio, se divide cada inviduo en dos partes y se cruzan sus genes.

7.1.5. Mutación. Luego de la reproducción se elige un individuo al azar, y se cambia un gen al azar con una probabilidad $P < 5\%$. Esto para minimizar la posibilidad de estancarse en máximo locales.

7.1.6. Promoción. Durante esta fase los descendientes se unen a la población, quedando un total de 2N individuos. Luego la población es ordenada según la calidad, y son promocionados los N mejores individuos.

7.1.7. Convergencia. Se dice que el problema converge cuando ya no existe diversidad en la población y los individuos son idénticos.

8. Fase 1: Cálculo de horario con IPL

La segunda alternativa que proponemos para ubicar las asignaturas en el tiempo es resolverlo mediante un motor de Programación Lineal Entera (IPL), Si bien esta garantiza obtener la mejor solución, el tiempo de ejecución es mucho más alto que el algoritmo genético, por lo que fijamos un límite de tiempo de ejucución y retornamos la mejor solución encontrada hasta el momento. El modelo simplificado luego de aplicar el pre-procesamiento es el siguiente:

8.1. Constantes

- S : conjunto de asignturas
- $I = \{\langle start_{s,n}, end_{s,n}\rangle \mid s \in S \wedge n > 0\}$: subconjuntos de slots consecutivos
- $D = \{d_s \mid s \in S\}$: duración de cada asignatura
- $E = \{\langle e_{s1}, e_{s2}\rangle \mid s1 \in S \wedge s2 \in S \wedge s1 <> s2\}$: conjunto de restricciones excluyentes
- M : un número muy grande.

8.2. Variables

- $A = \{start_s \mid s \in S\}$: Asignación de horario

8.3. Función Objetivo

La función objetivo maximiza la cantidad de restricciones excluyentes E satisfechas.

Maximizar:

$\sum_{\langle s1,s2\rangle \in E, start \in A, d \in D}$
$(min(1, max(0, start_{s1} - start_{s2} + d_{s1} - 1)))+$
$(min(1, max(0, start_{s2} - start_{s1} + d_{s2} - 1)))$

8.4. Sujeto a

La primera restricción limita la asignación de horario de una asignatura al primer slot de cada subconjunto de slots consecutivos disponibles. Las segunda restricción es similar a la primera, pero esta limita la asignación al ultimo slot de cada subconjunto de slots disponibles. En ambas restricciones σ es la holgura, donde si esta es cero se cumple, y si es uno no se cumple la restricción. Esto debe ser así debido a que una asignatura puede ser asignada en cualquiera de los subconjutos de slots consecutivos disponibles. La tercera es una restricción auxiliar que se utiliza para asegurarse que una asignatura sea asignada a un y solo un subcojunto de slots consecutivos disponibles.

- $\forall_{s \in S}, \forall_{\langle start,end\rangle \in I} a_s \geq start_{s,n} - (M(1 - \sigma_{s,n}))$
- $\forall_{s \in S}, \forall_{\langle start,end\rangle \in I} a_s \leq end_{s,n} + (M(1 - \sigma_{s,n}))$
- $\forall_{s \in S} \sum_{n=0} \sigma_{s,n} = 1$

9. Fase 2: Asignación de Aulas

Para el problema de asignación de aulas, dividimos el problema en N problemas independientes según el tipo de aula requerida por cada asignatura y luego resolvemos cada subproblema mediante un motor de Programación Lineal Entera. El modelo es el siguiente:

9.1. Constantes

- S : conjunto de asignaturas
- A : conjunto de aulas
- $I = \{\langle start_{s,n}, end_{s,n}\rangle \mid s \in S \wedge n > 0 \wedge start_{s,n} > 0 \wedge end_{s,n} > 0\}$: conjunto de divisiones
- $C = \{c_s \mid c_s > 0 \wedge s <\in S\}$: cantidad de alumnos por asignatura
- $T = \{t_a \mid t_a > 0 \wedge a <\in A\}$: tamaño de aula

9.2. Variables

- $\Omega = \{a_s \mid s \in S \wedge (a \in A \vee a = 0)\}$: Asignación de aulas

9.3. Función Objetivo

Cada elemento a_s representa una asignación de aula (numero mayor a cero) a la asignatura s en caso que el aula esté disponible. En caso de no poder asignarse un aula a la asignatura este valor es cero. La función objetivo maxima la cantidad de asignaturas con aula asignada.

Maximizar:

$\sum_{a_s \in \Omega}(min(1, a_s))$

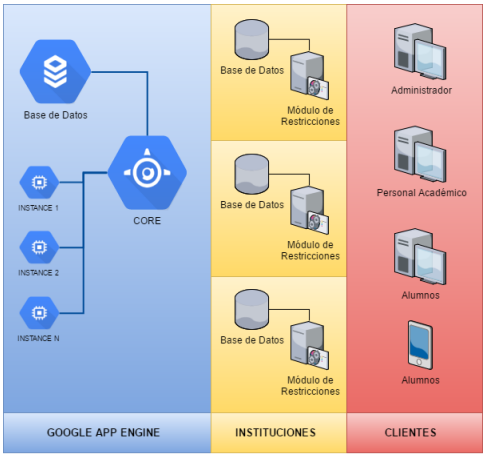

Figure 8. Arquitectura macro de la solución.

9.4. Sujeto a

La primera restricción indica que si dos asignaturas a_{s1} y a_{s2} tiene asignada un aula, y esta es la misma, entonces las asignaturas no deben coicidir en horario. La segunda restricción inidica que una asignatura solo puede asignarse a un aula con la capacidad suficiente.

- $\forall_{a_{s1} \in \Omega} \forall_{a_{s2} \in \Omega} \forall_{\langle start_{s1,n}, end_{s2,n} \rangle \in I}$ $a_{s1} = a_{s2} \wedge a_{s1} <> 0 \wedge a_{s2} <> 0 \rightarrow$ $start_{s1,n} > end_{s2,n} \vee start_{s2,n} > end_{s1,n}$
- $\forall_{t_a \in T} \forall_{c_s \in C} \forall_{a_s \in \Omega} t_a \geq c_s$

10. Arquitectura del Sistema

Una característica del problema de calendarización es que el consumo de recursos no es uniforme, ya que durante la generación del horario utilizaría muchos recursos de CPU y memoria RAM, pero una vez generado el horario, se libera todos los recursos, que en caso de ser dedicado quedaría todo el año ocioso. Por lo que tomando en cuenta la optimización decidimos diseñar la solución sobre una plataforma cloud, específicamente sobre Google App Engine. De manera que los recursos necesarios para el cálculo del horario sea solicitados on-demand, y liberados una vez finalizado el proceso. Si bien, la solución está pensada originalmente para la FPUNA, la arquitectura del sistema está diseñada para soportar varias instituciones. La solución se encuentra dividida en 3 capas, como se puede apreciar en la figura 8.

10.1. Componentes

10.1.1. Clientes. El Front-End de la aplicación esta implementado con el framework de Google AngularJS [21] para la lógica de las vistas, y para los componentes visuales utilizamos Bootstrap [22]. Las acciones permitidas desde esta vista son:

- ABM de usuarios
- Manejo de Roles
- Administración de Horarios
- Configuración de Aulas
- Configuración de Asignaturas
- Configuración de Carreras
- Visualización del Horario

10.1.2. Módulo de Restricciones. Este módulo es el Back-end de la configuración del problema, está escrito en JavaEE [19], expone todas las operaciones mediante servicios RESTful [25], y debe ser desplegado en un servidor de aplicaciones con soporte para EJB.

Toda la configuración es persistida en una base de datos PostgreSQL [23], que es accedida mediante MyBatis [24].

Cada institución educativa deberá desplegar su propia instancia del módulo de Restricciones para acceder al motor de resolución de horarios (Core). Dependiendo de la cantidad de alumnos con que cuente, la institución podría desplegar el módulo de Restricciones en un servidor local en el caso de poseer pocos alumnos, y en caso contrario podría desplegar el módulo sobre alguna plataforma Cloud.

10.1.3. Core. El core es el módulo que se encarga de la resolución del horario. Esta construido en JavaEE [19], y es el que implementa los algoritmos descriptos en la sección anterior. También se encarga de la autenticación de las instituciones que envían sus restricciones para generar sus horarios. El algoritmo utiliza mucha CPU y un consumo moderado de memoria RAM cada vez que genera un horario, por ello, decidimos montar este módulo en Google App Engine [17].

10.2. Funcionalidades de KrhonoPol

KrhonoPol provee, para usuarios con permisos de acceso, un conjunto de módulos para su correspondiente administración y, una página de inicio acesible por cualquier usuario para la visualización del horario generado y publicado. Entre esos módulos se encuentran:

- Gestión Académica→Carreras: Gestión de Carreras de la Institución. Alta, Baja, Modificación y Gestión de Asignaturas en una Carrera.
- Gestión Académica→Profesores: Gestión de Profesores de la Institución. Alta, Baja, Modificación y configuración de datos personales y de su disponibilidad horaria para impartir clases en sus Asignaturas correspondientes.
- Gestión Académica→Asignaturas: Gestión de Asignaturas de la Institución. Alta, Baja, Modificación, Asignación de una Carrera, y un Profesor a la asignatura correspondiente. Configuración de énfasis, sección, nivel, periodo, cantidad de alumnos inscriptos (para la asignación de Aula), restricciones de Horario de las Asignaturas (Horarios en las que pueden ser impartidas), restricciones de conflictos (Asignaturas con las que está en conflicto, no pueden

coincidir en horario) y restricciones de grupo (Asignaturas con las que está agrupada, deben coincidir en horario).

- Gestión Académica→Aulas: Gestión de Aulas de la Institución. Alta, Baja, Modificación. Configuración de capacidad y tipo de Aulas (Normal, Laboratorios, Dibujo, etc.).
- Gestión de Horarios: Módulo en el cual el usuario podrá administrar todo lo que se refiere a un horario, entre ellos: calcular, editar, visualizar, publicar, descartar un horario.

11. Resultados Obtenidos

El hardware utilizado para realizar las pruebas fue una computadora con un procesador Core i7 de 8 núcleos y 32 GB de memoria RAM.

11.1. Cálculo de Horario

Las pruebas para la asignación de horarios fueron realizadas tomando como referencia el horario del primer periodo del año 2.017 de la FPUNA [18] En este periodo académico se habilitaron un total de 1.305 asignaturas, cada asignatura es dividida en 2 o 3 partes, de tal manera que cada división debe ser asignada preferentemente en días diferentes. Teniendo así un total de 2.478 divisiones. Sometimos al horario de referencia, a una evaluación para calcular el fitness del mismo y nos arrojó un valor de 146 divisiones a reubicar de 2.478 para obtener cero conflictos.
Implementamos el modelo de programación lineal entera para el cálculo de horario en CPLEX [26], y lo ejecutamos por 8 horas y nos retornó una solución con 118 divisiones a reubicar de 2.478 para obtener cero conflictos.
Para el caso del algoritmo genético realizamos un total de 30 ejecuciones del algoritmo genético, y escogimos la ejecución que nos arrojó la mejor solución, que tuvo un valor fitness de 96 divisiones a reubicar de 2.478 para obtener cero conflictos. Las ejecuciones tuvieron un tiempo promedio de 25 minutos, por lo que el tiempo total de ejecución fue de 12 horas aproximadamente.
Con esto podemos ver una notable mejora tanto en la calidad de la solución, como en tiempo de elaboración, ya que actualmente el horario es construido en un periodo variable de 10 a 15 días hábiles, si consideramos una jornada laboral de 8 horas, el tiempo sería de 120 horas. Por lo que tenemos una mejora de tiempo en un factor de 10 a 1 aproximadamente.

11.2. Asignación de Aulas

Luego de obtener el horario en la fase uno, procedimos a realizar la asignación de aulas para las asignaturas, teniendo en cuenta la capacidad del aula y la cantidad de inscriptos por asignatura. Las aulas disponibles son las siguientes:

- Normales = 52
- Laboratorios de Informática = 6
- Salas de Dibujo = 2

El tiempo de ejecución del algoritmo fue de 2 horas y logró ubicar todas las asignaturas a excepción de una. El tiempo necesario para realizar esta asignación en forma manual es de 2 días, de vuelta si consideramos una jornada laboral de 8 horas, el tiempo de mejora es de 10 a 1 aproximadamente.
En la figura 9 se puede apreciar como quedó la distribución de aulas del día miércoles, cada fila de la figura representa a un aula, y cada celda, en la fila, a una porción de tiempo de 15 minutos (slot time). Las celdas coloreadas indican el tiempo en el que las aulas están reservadas para impartir clases de alguna asignatura y las celdas no coloreadas indican la disponibilidad del aula.

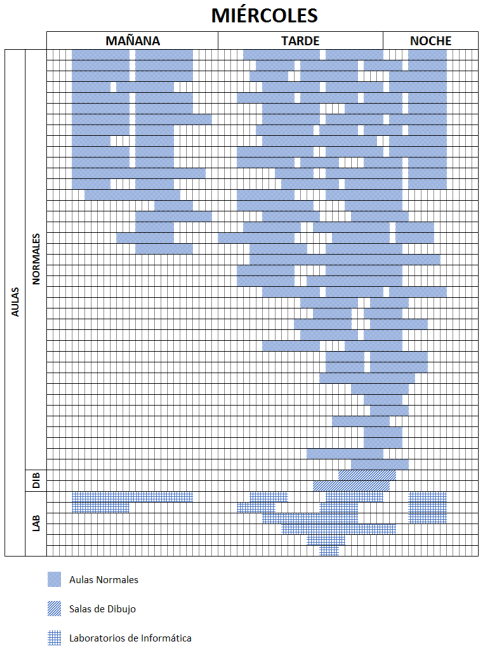

Figure 9. Distribución de aulas del día Miércoles.

12. Conclusión

En este trabajo presentamos una solución integral para la administración y generación automática de horarios de clases y asignación de aulas, para el caso de estudio: FPUNA. Si bien existen trabajos y herramientas que resuelven problemas de calendarización, ninguna satifacen el 100% de necesidades de la FPUNA.

Las pruebas realizadas arrojaron excelentes resultados, ya que logramos realizar una calendarización y asignación

de aulas satisfactoria en un periodo de tiempo considerablemente menor comparado con el método manual utilizado en la actualidad por la FPUNA. Se reduce el tiempo de varias semanas de trabajo a solo 12 horas de cómputo. Además de ello, como valor agregado, se reduce el esfuerzo para la gestión y mantenimiento del horario.

Otro punto a destacar es que con el horario actual, son los profesores quienes deben adaptar su disponiblidad con el horario, ya que para la construcción del mismo se utiliza siempre la del año anterior. Sin embargo con KhronoPol, esta situación se invierte y es posible construir un horario adpatado a la disponibildad de los profesores.

También dejamos abierta la posiblidad a desarrollos futuros para la integración de otras aplicaciones a KhronoPol, como por ejemplo aplicaciones móviles que consuman los servicios REST que disponibilizamos, integración con Google Calendar. Otro aporte importante sería la integración con el sistemas de inscripciones de tal forma que se puedan tener datos reales de la cantidad de alumnos inscriptos en cada asignatura para la apertura automática de nuevas secciones por asignaturas y asignación de aulas.

References

[1] Untis Expres, Generador de Horarios. http://www.programahorario.com/index.php?lang=SP.

[2] kronowin, Generador de Horarios. http://kronowin.es/0j/.

[3] GHC, Generador de Horarios para Centros Escolares. https://www.penalara.com/es/ES/quees/.

[4] FET, Generador de Horarios Libre. http://www.kdeblog.com/fet-el-generador-de-horarios-libre.html.

[5] TABLIX, Generador de Horarios. https://www.tablix.org/articles/about/.

[6] Tomás Müller (2005) Constraint-based Timetabling.

[7] Tomás Müller (2007) ITC2007 Solver Description: A Hybrid Approach.

[8] Goldberg, D. (1989) Genetics Algorithms in Search, Optimization and Machine Learning. Addison Wesley.

[9] Rakesh P. Badoni and D.K. Gupta (2015) A New Algorithm Based On Students Groupings For University Course Timetabling Problem.

[10] Krzysztof Socha, Michael Sampels, and Max Manfrin (2003) Ant Algorithms for the University Course Timetabling Problem with Regard to the State-of-the-Art.

[11] Noel Rodriguez, Jose Martinez, Juan J. Flores, Mario Graff (2014) Solving a Scholar Timetabling Problem Using a Genetic Algorithm - Study Case: Instituto Tecnologico de Zitacuaro.

[12] CARLA LENINCA PACHECO AGÜERO (2000) Distribución Óptima de Horarios de Clases utilizando la técnica de Algoritmos Genéticos.

[13] Hamdy M. Mousa and Ashraf B. El-Sisi (2013) Design and Implementation of Course Timetabling System Based on Genetic Algorithm.

[14] Souad Larabi Marie-Sainte1 (2015) A survey of Particle Swarm Optimization techniques for solving university Examination Timetabling Problem.

[15] Iosif V. Katsaragakis, Ioannis X. Tassopoulos and Grigorios N. Beligiannis (2015) A Comparative Study of Modern Heuristics on the School Timetabling Problem.

[16] Gurobi Optimization *Consultado el 02/02/2017*. http://www.gurobi.com/resources/getting-started/mip-basics

[17] Documentación oficial Google APP Engine, *Consultado el 15/04/2017*. https://cloud.google.com/appengine/.

[18] Horario de Clases FPUNA - Primer Periodo Académico 2017 *Consultado el 27/03/2017*. http://www.pol.una.py/archivos/Horario_clases_Primer_Periodo_31032017.xls

[19] Java Platform, Enterprise Edition (Java EE) 7 http://docs.oracle.com/javaee/7/index.html

[20] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein (2009) Introduction to Algorithms, Third Edition

[21] AngularJS API Docs https://docs.angularjs.org/api

[22] Bootstrap http://getbootstrap.com/components/

[23] PostgreSQL 9.4 https://www.postgresql.org/docs/

[24] MyBatis http://www.mybatis.org/mybatis-3/

[25] Leonard Richardson and Sam Ruby RESTful Web Service

[26] IBM ILOG CPLEX Optimization https://www.ibm.com/support/knowledgecenter/en/SSSA5P_12.6.1/ilog.odms.studio.help/Optimization_Studio/topics/COS_home.html