Modelos de Optimizacion 2019-2020

Amanda Marrero, Manuel Fernández, Loraine Monteagudo

25 de noviembre de 2020

1. Introducción

Seguramente te has sentido estancado en algún momento, y no es por falta de ideas, sino por falta de tiempo. A todos les ha pasado al menos una vez. Te abrumas y no tienes tiempo para todo o para todos, y te preguntas: ¿Cómo organizar mi tiempo?

En ocasiones por desconocimiento podemos perder valiosas oportunidades, llegar tarde a una entrevista de trabajo o incluso no conocer del tiempo que disponemos para coordinar una salida con amigos. Para evitar alguno de los escenarios anteriores necesitas administrar tu tiempo de manera efectiva, sacando así mejor provecho del día. Una buena forma de hacerlo es usando un horario semanal.

En el siguiente artículo proponemos una solución a la organización de las actividades , con la cual esperamos que se vuelvan más productivos.

2. Modelando el Problema

Organizar la agenda de una persona particular, suena sumamente complicado, más aún , si durante el transcurso de los días se añaden nuevas actividades. El problema que nos proponemos resolver es encontrar una asignación óptima de un conjunto de tareas en una agenda semanal con actividades previamente prefijadas. Matemáticamente hablando:

$$Min\ z = \sum_{i \in Turnos} \sum_{j \in Actividades} C_{ij} * X_{ij}$$

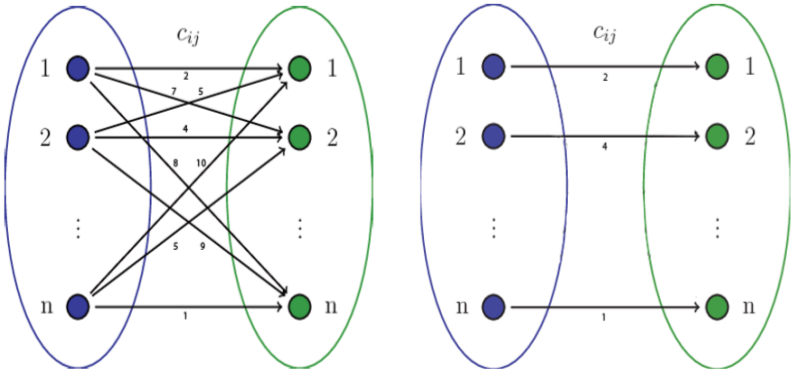
Donde:

- X_{ij} es una variable booleana que representa si la actividad "j" fue asignada al turno "i".
- C es una matriz de costos que se computa con un función que depende de la prioridad y el horario deseado de cada actividad .

Restricciones del problema:

- $\sum_{j \in Actividades} X_{ij} = 1 \ \ \forall i \in Turnos$. O sea que cada actividad es asignada a uno y solo un turno.
- $\sum_{i \in Turnos} X_{ij} = 1 \ \ \forall j \in Actividades$. O sea que cada turno es asignado a una y solo una actividad.

Podemos observar como el problema anterior es un caso particular del problema de asignación, el mismo consiste en encontrar la forma de asignar ciertos recursos disponibles para la realización de determinadas tareas al menor costo. Formalmente, consiste en encontrar un emparejamiento de peso óptimo en un grafo bipartito ponderado.



Este problema es tan antiguo como la revolución industrial y desde entonces varios métodos de solución han surgido. Normalmente es atacado con métodos heurísticos, pero también se ha intentado resolver con métodos deterministas como Simplex y el algoritmo húngaro.

2.1. Algoritmo Húngaro

El Algoritmo Húngaro es un algoritmo de optimización que resuelve problemas de asignación en un tiempo polinómico, O(n^3). La solución del problema se basa en los siguientes teoremas, que constituyen la base teórica y respaldarán la validez de dicho algoritmo.

Teorema 2.1 *Si las variables X_{ij} , i =1,. . . ,n, j =1 ,. . . ,n, son soluciones óptimas para un problema de asignación con función objetivo:*

$$z = \sum_{i=1}^{n}\sum_{j=1}^{n} C_{ij} * X_{ij}$$

entonces esos mismos valores son también soluciones óptimas para un problema cuya función objetivo es :

$$z' = \sum_{i=1}^{n}\sum_{j=1}^{n} C'_{ij} * X_{ij}$$

siendo C'_{ij} = C_{ij} - u_i - v_j, con u_i y v_j constantes.

Demostración

$$z' = \sum_{i=1}^{n}\sum_{j=1}^{n} C'_{ij} * X_{ij} = \sum_{i=1}^{n}\sum_{j=1}^{n} (C_{ij} - u_i - v_j) * X_{ij} =$$

$$= \sum_{i=1}^{n}\sum_{j=1}^{n} C_{ij} * X_{ij} - \sum_{i=1}^{n}\sum_{j=1}^{n} u_i * X_{ij} - \sum_{i=1}^{n}\sum_{j=1}^{n} v_j * X_{ij} =$$

$$= z - \sum_{i=1}^{n} u_i \sum_{j=1}^{n} X_{ij} - \sum_{j=1}^{n} v_j \sum_{i=1}^{n} X_{ij} =$$

$$= z - \sum_{i=1}^{n} u_i - \sum_{j=1}^{n} v_j = z - K$$

Las funciones z y z' se diferencian en la constante "K ", por tanto, alcanzan el óptimo en el mismo conjunto de valores de las variables.

Aplicando el resultado del Teorema 2.1 a la tabla de costos se pueden hacer transformaciones sin que cambie la solución óptima. Concretamente las siguientes transformaciones: restar en filas y/o columnas una constante.

Teorema 2.2 *Si $C_{ij} \geq 0$, i,j = 1 , ..., n y el conjunto de valores de las variables es tal que:*

$$z = \sum_{i=1}^{n}\sum_{j=1}^{n} C_{ij} * X_{ij} = 0$$

entonces X_{ij} , i,j = 1 , ..., n, es la solución óptima.

Demostración

Si todos los costos C_{ij} , i,j = 1 , ..., n, son mayores o iguales que cero, el valor de la función objetivo será mayor o igual que cero. Por tanto, si los valores de las variables X_{ij} , i,j = 1 , ..., n, dan a z el valor 0, ese es el mínimo absoluto y al mismo tiempo la solución óptima para el problema.

El método de solución se basa en transformar la tabla de costos con las operaciones que permite el Teorema 2.1 para conseguir ceros. Si se consigue una asignación en la que la función objetivo tome el valor cero, teniendo en cuenta el Teorema 2.2, esa asignación ya es óptima.

Teorema 2.3 *En cualquier grafo bipartito, el número de aristas en un emparejamiento máximo es igual al número de vértices en un cubrimiento mínimo.(Teorema de Konig)*

Esta demostración supera de los objetivos del artículo, leer en:

1. `https://es.qaz.wiki/wiki/Konig%27s_theorem_(graph_theory)`
2. `https://mathworld.wolfram.com/Koenig-EgevaryTheorem.html`

El teorema de Konig asegura que el número de ceros que se pueden asignar independientemente en filas y/o columnas es igual al número al mínimo número de filas y/o columnas que cubren todos los ceros.

2.2. Algoritmo

1. Obtener ceros por filas. Restar a cada fila el mínimo

$$u_i = min_j \;\; c_{ij}$$

   Los nuevos elementos de la tabla serían: $c'_{ij} = c_{ij}$ - u_i i,j = 1,...,n

2. Obtener ceros por columnas. Restar a cada columna el mínimo

$$v_j = min_i \;\; c_{ij}$$

   Los nuevos elementos de la tabla serían : $c''_{ij} = c'_{ij}$ - u_i i,j = 1,...,n

∗ Notar que los pasos 1 y 2 están respaldados en el Teorema 2.1

3. Asignación de casillas que tengan ceros. Elegir la fila o la columna con menor número de ceros. Asignar uno y eliminar los ceros de la misma fila y columna. Repetir la asignación en filas y columnas continuando por aquella que tenga el mínimo número de ceros sin eliminar.

   a) Si al terminar la asignación en todas las filas hay un cero asignado, se tiene la solución óptima ⇒ Parar el algoritmo.

   b) Si existe alguna fila que no tenga un cero asignado al finalizar el procedimiento de asignación ⇒ Ir al paso 4.

∗ Como la matriz de costos cumple que : $C_{ij} \geq 0$, i,j = 1 , ..., n ⇒ podemos aplicar el Teorema 2.2

4. Elegir el mínimo numero de filas y/o columnas que cubren todos los ceros. Este número mínimo se consigue con el siguiente procedimiento:

   a) Marcar las filas que no tienen ceros asignados.

   b) Marcar las columnas que tienen ceros asignados en las filas marcadas en (a).

   c) Marcar las filas que tienen ceros asignados en las columnas marcadas en (b).

   Repetir (b) y (c) hasta que no se puedan marcar más filas y/o columnas. Las filas no marcadas y las columnas marcadas cubren todos los ceros. ⇒ Ir al paso 5.

∗ El paso 4 está respaldado por el resultado del Teorema de Konig.

5. Crear nuevos ceros. Elegir el elemento mínimo que no está cubierto. Restarlo a todos los elementos de las filas no cubiertas y sumarlo a los elementos de las columnas cubiertas . ⇒ Ir al paso 3.

Ejemplo 2.1. Supongamos que se quieren ubicar 4 actividades en los turnos libres del día" X", debiendo ser asignada cada actividad a un único turno. El grado de inconformidad para cada actividad ubicada en un turno dado se rige por la siguiente tabla. Calcular la asignación para la cual la suma total de la inconformidad sea mínima.

	1	2	3	4
A	58	58	60	54
B	66	70	70	78
C	106	104	100	95
D	52	54	64	54

1. Restamos en cada fila el mínimo , u = (54,66,95,52)

	1	2	3	4
A	4	4	6	0
B	0	4	4	12
C	11	9	5	0
D	0	2	12	2

2. Restamos en cada columna el mínimo , v = (0,2,4,0)

	1	2	3	4
A	4	2	2	0
B	0	2	0	12
C	11	7	1	0
D	0	0	8	2

3. Asignar ceros.

   a) La primera fila tiene un solo cero.Asignar(A,4) y Eliminar (C,4).En la segunda fila hay 2 ceros para asignar, en la tercera no hay ceros y en la cuarta hay 2 ceros.

	1	2	3	4
A	4	2	2	$\boxed{0}$
B	0	2	0	12
C	11	7	1	$\not{0}$
D	0	0	8	2

   b) Seguir en las columnas en las columnas. En la primera hay dos ceros para asignar, en la segunda hay solo un cero, por tanto, Asignar(D,2) y Eliminar(D,1). En la tercera columna hay 1 cero $\Rrightarrow$ Asignar(B,3) y Eliminar (B,1).

	1	2	3	4
A	4	2	2	$\boxed{0}$
B	$\not{0}$	2	$\boxed{0}$	12
C	11	7	1	$\not{0}$
D	$\not{0}$	$\boxed{0}$	8	2

El número total de ceros asignados es 3. No se tiene la asignación óptima lo que implica que hay que continuar con el siguiente paso para conseguir más ceros.

4. Elegir el mínimo número de filas y/o columnas que cubren todos los ceros.

   a) Se marca la fila C porque no hay ceros asignados.

   b) En la fila marcada hay un cero en la cuarta columna ⇛ Marcamos la columna 4

   c) La cuarta columna tiene un cero asignado en la primera fila ⇛ Marcamos la fila 1 y repetimos el paso(b)

   (*b*) En la primera fila no hay ceros sin asignar.

   El proceso de marcar filas y columnas ha terminado. Cubrimos las filas no marcadas y las columnas marcadas.

	1	2	3	4	
A	4	2	2	$\boxed{0}$	X
B	$\not{0}$	2	$\boxed{0}$	12	
C	11	7	1	$\not{0}$	X
D	$\not{0}$	$\boxed{0}$	8	2	
				X	

5. Crear nuevos ceros. El mínimo de los elementos no cubiertos es 1. Entonces, restamos 1 a las filas no cubiertas y sumamos 1 a las columnas cubiertas. La tabla resultante de las operaciones anteriores es la siguiente:

	1	2	3	4
A	3	1	1	0
B	0	2	0	13
C	10	6	0	0
D	0	0	8	3

   ⇒ Volver al paso 3.

3 Asignar ceros.

	1	2	3	4
A	3	1	1	$\boxed{0}$
B	$\boxed{0}$	2	$\not{0}$	13
C	10	6	$\boxed{0}$	$\not{0}$
D	$\not{0}$	$\boxed{0}$	8	3

Los ceros asignados son 4 que coincide con la cantidad de actividades a ubicar.
⇒ Fin del Algoritmo.

* Solución óptima:

  A → 4 : La actividad 4 será ubicada en el turno A.

  B → 1 : La actividad 1 será ubicada en el turno B.

  C → 3 : La actividad 3 será ubicada en el turno C.

  D → 2 : La actividad 2 será ubicada en el turno D.

* Costo óptimo : $c_{A4} + c_{B1} + c_{C3} + c_{D2} = 54 + 66 + 100 + 54 = 274$.

3. Particularidades del Modelo

Retomando nuestro problema inicial, tenemos un par de particularidades para darle solución con el Algoritmo Húngaro descrito en la sección anterior.

* El algoritmo húngaro solo es aplicable cuando la matriz de costo C es una matriz cuadrada, por tanto, si en un determinado momento tenemos más turnos disponibles que actividades a asignar, tendremos que adicionar actividades ficticias donde $C_{ij} = 0$, para i = 1,...,n , $\forall j \in ActividadesFicticias$.

* Una pieza fundamental en los problemas de asignación es la matriz de costos. La pregunta sería entonces : ¿Cómo calcular nuestra matriz ?

Matriz de Costos

Para poder calcular nuestra matriz tenemos que hacer una serie de definiciones:

1. Una Actividad posee los siguientes campos: prioridad(p), hora de inicio(hI), día(D).
2. Sea M_j la matriz de costos para cada actividad en particular, es decir M_j[t,d] se encuentra el costo de la actividad j-ésima para el turno "t"del día "d".
3. Un Turno posee los siguientes campos: hora de inicio(hI), día(d).

* La prioridad p ∈ (1,2,3), la hora de inicio ∈ 8:00 - 16:00, los días ∈ (1,2,3,4,5,6,7).

4. Sea "Des" un arreglo de desplazamientos con los siguientes valores : 0,1,3,5,7,9,11,13,15,17.El arreglo "Des" será utlizado para calcular el grado de inconformidad cuando una actividad es ubicada fuera del momento deseado del usuario.

$$C_{ij} = M_j \quad i = 1, ..., n \ , \forall j \in Actividades$$

Nuestro calculo de "C" se reduce entonces de calcular M_j, $\forall j \in Actividades$.

Calcular M_j

Sea A_j(p,hI,D) una actividad.

1. $\forall t \in Turnos$ con t.d < A_j.D ⇒ M_j[t,d] = Q. Notar que Q sería un valor de penalización, por tanto, tiene que tomar un valor suficientemente grande.

   Cota superior para Q:
   Calculemos entonces el mayor valor que pudiera tomar una casilla en M_j para una actividad A_j determinada. Sea A_j(p = máximo valor admisible que sería 3, hI = 8:00, d = 1), es decir el primer turno del lunes. Como se explicará más adelante en el cálculo de M_j, es evidente que el mayor grado de inconformidad estará en el último turno del domingo. Supongamos que en dicho domingo el primer turno comienza a las 8:00 y el último turno a las 16:00 entonces aplicando el paso 3 del algoritmo para calcular M_j, tenemos que, denotando t_u como el último turno del domingo, se cumple que:

   $M_j[t_u, 5] = 3 + ((16\text{-}8) + 9 * (7\text{-}1)\ ) * 17 = 1057 \Rightarrow$ Q > 1057, por lo tanto tomaremos a Q = 1100.

2. $\forall t \in Turnos$ con t.d = A_j.D $\Rightarrow$ M_j[t,d] = A_j.p + abs(A_j.hI - t.hI) * k , donde k = Des[p]

3. $\forall t \in Turnos$ con t.d > A_j.D :
   Sea $d_z \in$ Días , $\forall d_z$ tal que $d_z > A_j$.D $\Rightarrow$

   a) M_j[t,d_z] = A_j.p + (t.hI - t_0.hI + (9 * (d_z - A_j.D))) * k. , siendo t_0 el primer turno $\in d_z$ y k = Des[A_j.p + (d_z - A_j.D)] .

¿Por qué calcular M_j de la manera anterior?
Por lo general, cuando una persona fija una actividad para un día específico se debe a que en los días anteriores no puede, por lo tanto en el paso 1 penalizamos todos aquellos turnos de los días anteriores para que sea baja la probabilidad de asignar dicha actividad a alguno de esos turnos. Con el mismo principio debemos penalizar, aunque en menor medida, a todos aquellos turnos que tengan lugar después del día deseado, teniendo en cuenta que mientras más lejos del día deseado esté más penalizado deba ser.Por último, todos aquellos turnos del día deseado tendrán el menor grado de inconformidad, pero de igual manera se debe tratar de ubicar dicha actividad lo más cerca posible a la hora indicada .Esto implica que a medida que los turnos se alejen de la hora establecida tendrán un mayor grado de incorformidad.

Ejemplo

Calculemos la matriz M_j para una actividad A_j con(p = 2,hI = 10:00, D = 3), con 5 turnos disponibles a lo largo de los 7 días de la semana, con hora inicial para el primer turno 8:00 y los demás turnos están a intervalos de una hora.

T/D	1	2	3	4	5	6	7
1, 8:00	1100	1100	8	47	128	245	398
2, 9:00	1100	1100	5	52	135	254	409
3, 10:00	1100	1100	2	57	142	263	420
4, 11:00	1100	1100	5	62	149	272	431
5, 12:00	1100	1100	8	67	156	281	442

La tabla anterior es la matriz de costos resultante para la actividad A_j, como se puede apreciar los valores en azul son los correpondientes al paso 1, los rojos al paso 2 y los de color piel fueron obtenidos con el paso 3.

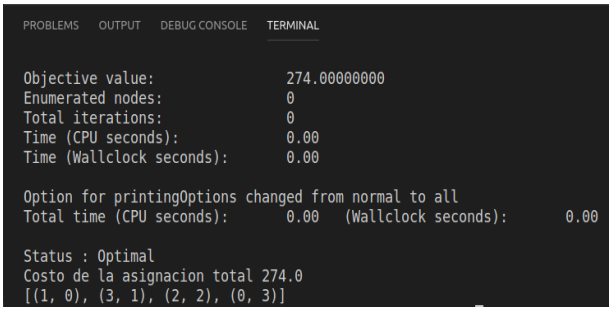
Optimización de la complejidad temporal

Al añadir cada nueva actividad lo esperado fuera correr todo el algoritmo con la matriz de costos "C" completa, pero dado que el algoritmo es O(n^3) y en el peor caso "C" puede tener 45 turnos disponibles cada inserción, serían 56^3 operaciones una cantidad nada pequeña. Por lo tanto, al añadir una actividad A_j intentaremos añadirla a su día de preferencia, en caso de no ser posible, intentamos con el día siguiente y así sucesivamente. De esta manera correríamos el algoritmo con una matriz C' que contendría los turnos asociados a un día determinado, lo que conllevaría en el peor de los casos a 7 * 8^3 operaciones por inserción, una mejora sustancial en cuanto a la complejidad temporal.

4. Correctitud del Algoritmo

Para comprobar la correctitud de nuestra versión propuesta del famoso Algoritomo Húngaro resolveremos el mismo problema usando PuLP. Pulp es una librería de Python utilizada para modelar y resolver problemas de optimización mediante programación lineal.Esta librería incluye soporte para problemas de maximización y minimización con restricciones.

Resultados para el ejemplo 2.1 con Pulp



Como se puede apreciar se obtuvo el mismo resultado que con nuestro algoritmo.

Veamos cómo se comportan con un conjunto de tareas a asignar sobre una determinada Semana y un conjunto de actividades prefijadas:

Semana : Del Lunes 12 de Ocubre del 2020 al Domingo 18 de Ocubre del 2020

Actividades Prefijadas:

Nombre de la Actividad	Fecha	Hora de Ininio	Hora de Finalización
Encontrarme con Peter (EPT)	12/10/2020	-	-
Discusión sobre el cambio climático (DCC)	13/10/2020	8:00	-
Llevar a Sam al Parque (SP)	15/10/2020	12:30	13:00
Llevar a Sam al Parque (SP)	16/10/2020	12:00	13:00

Ejemplo 4.1

Nombre de la Actividad	Prioridad	Fecha	Hora
Encontrarme con Alina (EA)	1	14/10/2020	10:00
Éxamen de Condución (EC)	3	14/10/2020	10:00
Reunión con Ame (RA)	3	12/10/2020	13:00
Ver una Película (VP)	1	13/10/2020	11:00
Ir al Teatro (IT)	3	13/10/2020	11:00
Reunión con Russ (RR)	2	12/10/2020	13:00
Ir a Comer Pizza (CP)	2	13/10/2020	11:00

Resultados 4.1

Algoritmo Húngaro

Lunes	Martes	Miercoles	Jueves	Vierves	Sabado	Domingo
8:00-9:00/EPT	8:00-9:00/DDC	8:00-9:00/-	8:00-9:00/-	8:00-9:00/-	8:00-9:00/-	8:00-9:00/-
9:00-10:00/-	9:00-10:00/-	9:00-10:00/EA	9:00-10:00/-	9:00-10:00/-	9:00-10:00/-	9:00-10:00/-
10:00-11:00/-	10:00-11:00/VP	10:00-11:00/EC	10:00-11:00/-	10:00-11:00/-	10:00-11:00/-	10:00-11:00/-
11:00-12:00/-	11:00-12:00/IT	11:00-12:00/-	11:00-12:00/-	11:00-12:00/-	11:00-12:00/-	11:00-12:00/-
12:00-13:00/RR	12:00-13:00/CP	12:00-13:00/-	12:00-12:30/-	12:00-13:00/SP	12:00-13:00/-	12:00-13:00/-
13:00-14:00/RA	13:00-14:00/-	13:00-14:00/-	12:30-13:00/SP	13:00-14:00/-	13:00-14:00/-	13:00-14:00/-
14:00-15:00/-	14:00-15:00/-	14:00-15:00/-	13:00-14:00/-	14:00-15:00/-	14:00-15:00/-	14:00-15:00/-
1500-16:00/-	1500-16:00/-	1500-16:00/-	14:00-15:00/-	1500-16:00/-	1500-16:00/-	1500-16:00/-
			1500-16:00/-			

Algoritmo Pulp

Lunes	Martes	Miercoles	Jueves	Vierves	Sabado	Domingo
8:00-9:00/EPT	8:00-9:00/DCC	8:00-9:00/-	8:00-9:00/-	8:00-9:00/-	8:00-9:00/-	8:00-9:00/-
9:00-10:00/-	9:00-10:00/-	9:00-10:00/EA	9:00-10:00/-	9:00-10:00/-	9:00-10:00/-	9:00-10:00/-
10:00-11:00/-	10:00-11:00/CP	10:00-11:00/EC	10:00-11:00/-	10:00-11:00/-	10:00-11:00/-	10:00-11:00/-
11:00-12:00/-	11:00-12:00/IT	11:00-12:00/-	11:00-12:00/-	11:00-12:00/-	11:00-12:00/-	11:00-12:00/-
12:00-13:00/RR	12:00-13:00/VP	12:00-13:00/-	12:00-12:30/-	12:00-13:00/SP	12:00-13:00/-	12:00-13:00/-
13:00-14:00/RA	13:00-14:00/-	13:00-14:00/-	12:30-13:00/SP	13:00-14:00/-	13:00-14:00/-	13:00-14:00/-
14:00-15:00/-	14:00-15:00/-	14:00-15:00/-	13:00-14:00/-	14:00-15:00/-	14:00-15:00/-	14:00-15:00/-
1500-16:00/-	1500-16:00/-	1500-16:00/-	14:00-15:00/-	1500-16:00/-	1500-16:00/-	1500-16:00/-
			1500-16:00/-			

Como podemos observar en las tablas anteriores los resultados solo difieren en el Martes.Analicemos porque podemos decir que aunque difieren en la asignación son resultados equivalentes. La diferencia se encuentra en que nuestro algoritmo asigna la actividad "Ver una Película" al turno de las 10:00 y la actividad "Ir a comer Pizza" al turno de las 12:00 , mientras que el algoritmo PuLP los asigna en el orden inverso.¿Cúal de las dos asignaciones tiene una menor inconformidad? Para respodner la pregunta anterior mostremos el martes de la matriz de costos M_j asociada a cada activdad.

Turno	Ver una Película	Ir a Comer Pizza
8:00-9:00	4	11
9:00-10:00	3	8
10:00-11:00	2	5
11:00-12:00	1	2
12:00-13:00	2	5
13:00-14:00	3	8
14:00-15:00	4	11
15:00-16:00	5	14

Con la tabla anterior tenemos que:

1. "Ver una Película " en turno de las 10:00 e "Ir a comer Pizza" en el turno de las 12:00 tiene un valor de 7.
2. "Ver una Película" en turno de las 12:00 e "Ir a comer Pizza" en el turno de las 10:00 tiene un valor de 7.

* Por tanto ambas asignaciones son equivalentes.

Ejemplo 4.2

Nombre de la Actividad	Prioridad	Fecha	Hora
Estudiar Italiano (EI)	1	15/10/2020	12:00
Reunión con Julls (RJ)	2	15/10/2020	8:00
Encontrarme con Pepe (EP)	3	12/10/2020	8:00
Escuchar Música (EM)	2	15/10/2020	12:00
Estudiar Italiano (EI)	3	16/10/2020	11:00
Comer Espaguetis (CE)	3	14/10/2020	9:00
Comer Dulce (CD)	2	14/10/2020	9:00
Llamar a mi madre (LLM)	3	15/10/2020	12:00

Resultados 4.2

Algoritmo Húngaro

Lunes	Martes	Miercoles	Jueves	Vierves	Sabado	Domingo
8:00-9:00/EPT	8:00-9:00/DDC	8:00-9:00/CD	8:00-9:00/RJ	8:00-9:00/-	8:00-9:00/-	8:00-9:00/-
9:00-10:00/EP	9:00-10:00/-	9:00-10:00/CE	9:00-10:00/-	9:00-10:00/-	9:00-10:00/-	9:00-10:00/-
10:00-11:00/-	10:00-11:00/-	10:00-11:00/-	10:00-11:00/-	10:00-11:00/-	10:00-11:00/-	10:00-11:00/-
11:00-12:00/-	11:00-12:00/-	11:00-12:00/-	11:00-12:00/EI	11:00-12:00/EI	11:00-12:00/-	11:00-12:00/-
12:00-13:00/-	12:00-13:00/-	12:00-13:00/-	12:00-12:30/LLM	12:00-13:00/SP	12:00-13:00/-	12:00-13:00/-
13:00-14:00/-	13:00-14:00/-	13:00-14:00/-	12:30-13:00/SP	13:00-14:00/-	13:00-14:00/-	13:00-14:00/-
14:00-15:00/-	14:00-15:00/-	14:00-15:00/-	13:00-14:00/EM	14:00-15:00/-	14:00-15:00/-	14:00-15:00/-
1500-16:00/-	1500-16:00/-	1500-16:00/-	14:00-15:00/-	1500-16:00/-	1500-16:00/-	1500-16:00/-
			1500-16:00/-			

Algoritmo Pulp

Lunes	Martes	Miercoles	Jueves	Vierves	Sabado	Domingo
8:00-9:00/EPT	8:00-9:00/DDC	8:00-9:00/CD	8:00-9:00/RJ	8:00-9:00/-	8:00-9:00/-	8:00-9:00/-
9:00-10:00/EP	9:00-10:00/-	9:00-10:00/CE	9:00-10:00/-	9:00-10:00/-	9:00-10:00/-	9:00-10:00/-
10:00-11:00/-	10:00-11:00/-	10:00-11:00/-	10:00-11:00/-	10:00-11:00/-	10:00-11:00/-	10:00-11:00/-
11:00-12:00/-	11:00-12:00/-	11:00-12:00/-	11:00-12:00/EI	11:00-12:00/EI	11:00-12:00/-	11:00-12:00/-
12:00-13:00/-	12:00-13:00/-	12:00-13:00/-	12:00-12:30/LLM	12:00-13:00/SP	12:00-13:00/-	12:00-13:00/-
13:00-14:00/-	13:00-14:00/-	13:00-14:00/-	12:30-13:00/SP	13:00-14:00/-	13:00-14:00/-	13:00-14:00/-
14:00-15:00/-	14:00-15:00/-	14:00-15:00/-	13:00-14:00/EM	14:00-15:00/-	14:00-15:00/-	14:00-15:00/-
1500-16:00/-	1500-16:00/-	1500-16:00/-	14:00-15:00/-	1500-16:00/-	1500-16:00/-	1500-16:00/-
			1500-16:00/-			

Para este ejemplo se obtuvieron las mismas asignaciones.

5. Manual de usuario

La experiencia de usuario se puede realizar mediante un sitio WEB o una aplicación móvil desarrollada para android. También estará a disposición del usuario un parser de archivos de Emacs para su integración con la aplicación.

5.1. Parser de Emacs

Las tareas prefijadas tienen un papel fundamental a la hora de planificar un horario semanal. Con este programa se posibilita el procesamiento de un archivo de Emacs convirtiéndolo en varios archivos json, donde en cada archivo se agruparán actividades de una determinada semana. Los archivos json serán almacenados en la carpeta jsons con el nombre : {año}_{#semana} . Para la ejecuión del programa se necesita python3.7:

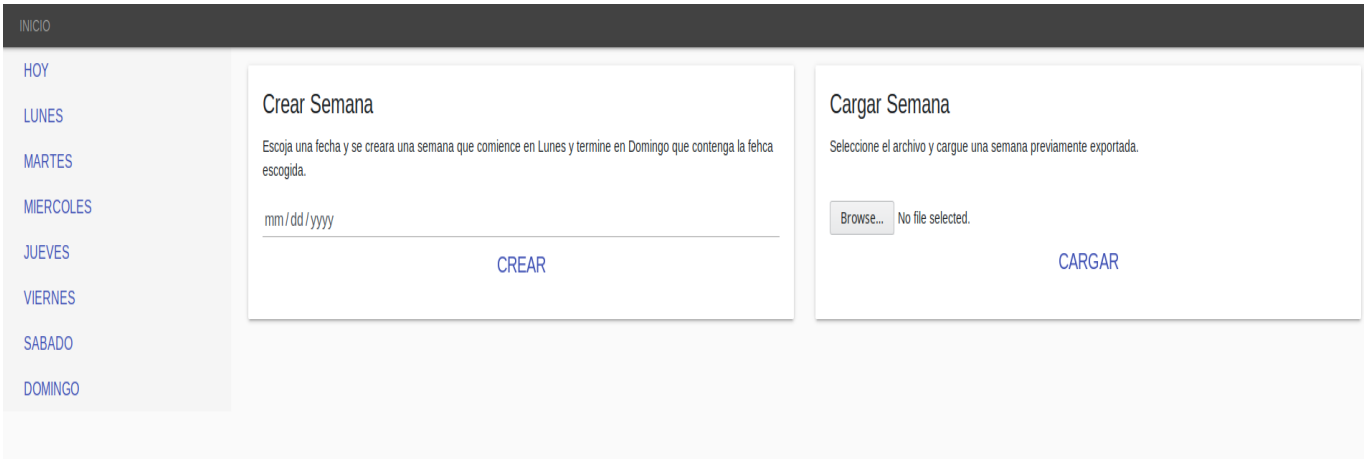
```
python3.7 act_parser.py
```

Una vez ejecutado el programa el usurio debe copiar la carpeta jsons generada tanto en la carpeta de la API como en la carpeta de la WEB, para su futura utilización con dichas herramientas visuales.

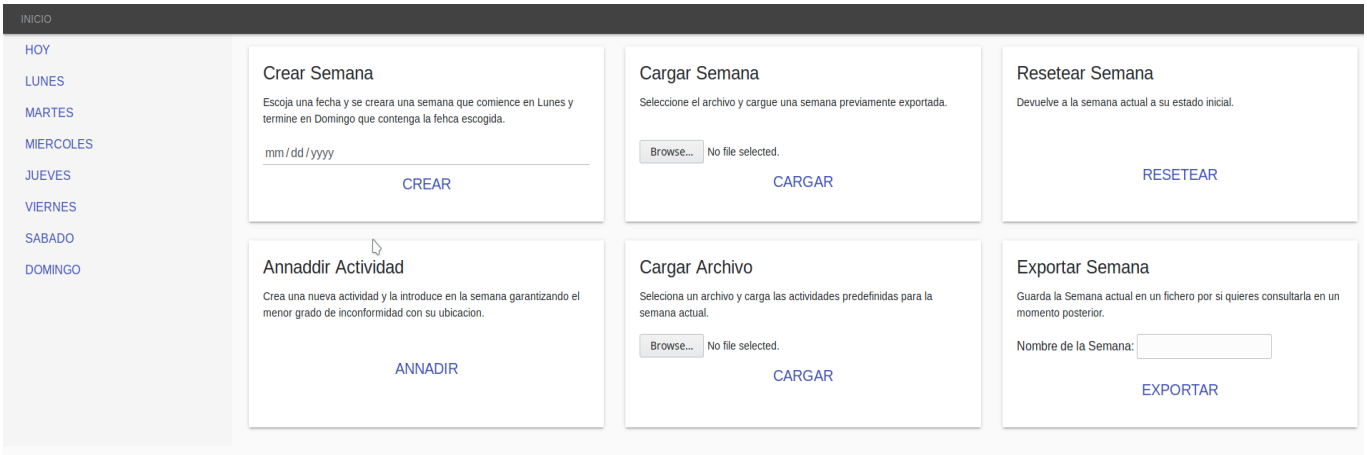
5.2. WEB

Pasos para montar el servicio Web de manera local:

1. Abrir una consola en la ubicación donde se encuentre almacenado el proyecto
2. python3 server.py (Requiere tener Flask instalado)
3. Abrir un navegador web y conectarse a la dirección : https://127.0.0.1:5000



Para Iniciar a organizar tu agenda primero necesitas crear o cargar una Semana.
Ya una vez creada la semana aparecen una gama de posibles acciones sobre esa semana.



Las opciones laterales nos mostraran las actividaddes que tenemos programadas para un dia determinado.

La siguiente imagen mostrará el menu para crear una nueva actividad:

5.3. Aplicación Movil

Pasos para recrear el entorno:

1. Crear y activar entorno virtual de python

```
python -m venv my-venv
source my-venv/bin/activate
```

2. Instalar las dependencias necesarias en el entorno:

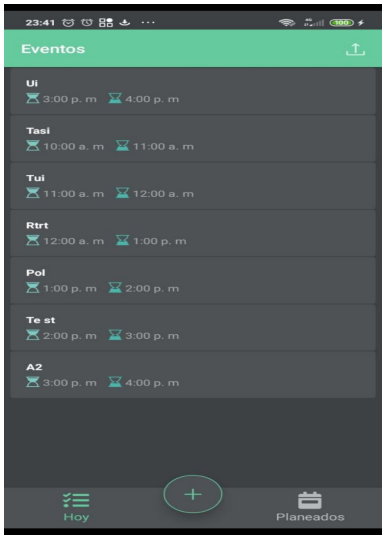
```
pip install -r requirements.txt
```

3. Crear hostpot con el móvil y conectar la PC. Determinar IP que asume la PC en la conexión

```
ipconfig en windows
ifconfig en linux
```

4. Ejecutar python app.py en el directorio raíz. (Levantar API, notar que lo hace en el puerto 5000 por defecto)
5. Abrir la apk previamente instalada e introducir el IP de la máquina seguida del puerto 5000. Esta será la dirección base de todas las peticiones requeridas para el correcto funcionamiento de la aplicación.

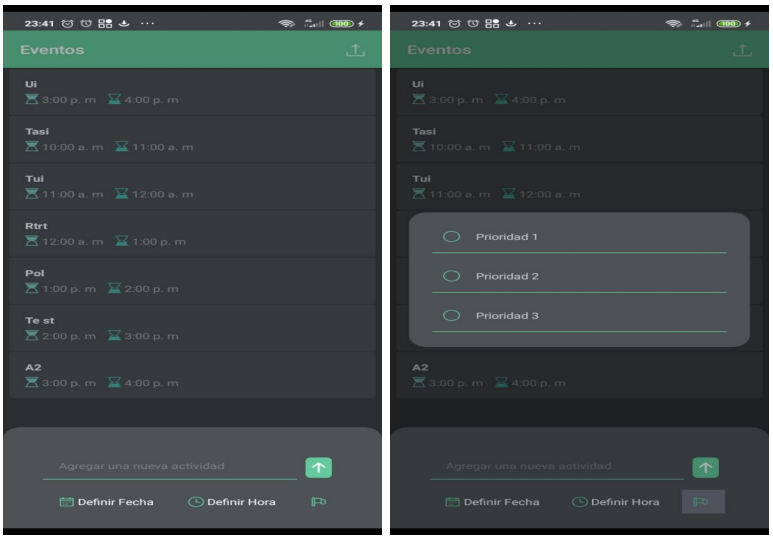
En la vista Inicial de la aplicación se tiene acceso a todas las funciones de la aplicación así como también muestra las actividades programadas para el dia de hoy.

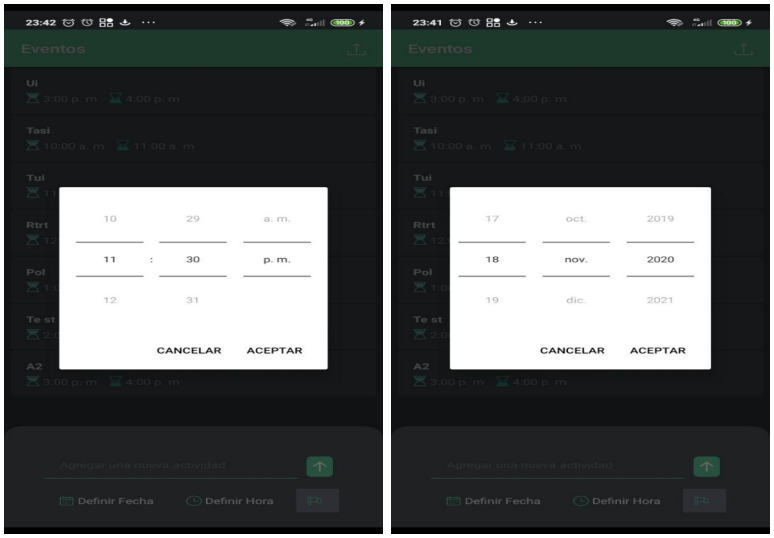


Funcionalidades:

1. Cargar Archivo de las actividades predefinidas (botón en la parte derecha Superior)
2. Eventos de Hoy ,muestra los eventos programados para el dia actual
3. Programados , nos lleva a una Vista donde podemos observar los eventos programados para cada día de la semana
4. Agregar un nuevo Evento , agrega un nuevo Evento a nuestra Semana (botón +)

Agregando un Evento:





Actividades Programadas: