Calendario NFL
Optimización con algoritmos genéticos

Fernando Márquez Pérez
420004482

Emiliano Domínguez Cruz
315335075

Fecha de entrega: 4 de Junio de 2023

Índice

1. Problema

1.1. Descripción

Año con año la *National Football League (NFL)* se enfrenta al reto de preparar el calendario con los juegos de la temporada. A lo largo de 18 semanas (que van de jueves a miércoles) entre septiembre y enero se juegan 272 partidos definidos con anterioridad en base a la división a la que pertenece cada equipo, el su desempeño en la temporada pasada y los equipos contra los que se ha enfrentado previamente. De forma que cada equipo juega 17 partidos durante la temporada y cuenta con una semana de descanso, o BYE.

Esto no es una tarea fácil ya que se enfrentan a 4 problemas principales

- El espacio de búsqueda es sumamente grande. La cantidad de ordenamientos posibles para los 272 partidos que se juegan es inmensa.
- Los partidos que se juegan son fijos por lo que no es posible cambiar alguno que este complicando las cosas. Además la distribución es compleja, nadie juega contra todos ni el mismo número de veces, lo que complica más los algoritmos.
- Se tienen muchas restricciones que afectan severamente los posibles ordenamientos de los partidos.

Además de esto, cada semana cuenta con *horarios estelares* que se televisan a nivel nacional los días lunes (MNF), jueves (TNF) y domingo (SNF). Como estos horarios son importantes se busca que se jueguen partidos con valor de interés para las televisoras. Tomar en cuenta estos horarios complica aún más el problema.

Además se deben de tomar en cuenta algunas anomalías: Las semanas 15 y 18 no tienen horarios predeterminados, estos se eligen al término de su semana anterior. Y la semanas de acción de gracias y de navidad. En acción de gracias (semana 11 o 12) se juegan tres estelares más en jueves que sustituyen el TNF, además este año uno de ellos se juega en blackday friday. En la semana de navidad (semana 16 o 17) se agregan estelares según el día de navidad: sábado, domingo y lunes agregan hasta tres estelares afectando los demás de la semana, los demás días no suelen afectar.

1.2. Codificación de ejemplares

Nuestros ejemplares constan de la información de la temporada y se almacenan como un txt con el formato:

- id_equipo, conferencia, división, bye pasado, 3 visitantes. Para los 32 equipos
- id_estadio, huso horario. Para los 30 estadios.
- id_partido, id local, id visitante, id estadio. Para cada uno de los 272 partidos.
- Información extra: semana thanksgiving, semana navidad, día navidad.

Tomemos como breve ejemplo el siguiente:

```
0 AFC N 7 FALSE  <-- Equipos
1 AFC N 11 TRUE
...

0 EST            <-- Estadios
1 EST
...

192 20 24 19 2   <-- Partidos
88 17 18 17 4

11               <-- Información extra
16 L
```

Cuyo significado en lenguaje natural seria algo como

```
Buffalo Bills, Americana, Norte, Bye semana 7, no triple visitante
Miami Dolphins, Americana, Norte, Bye semana 11, triple visitante
```

```
...

Highmark Stadium, Tiempo del Este
Hard Rock Stadium, Tiempo del Este
...

ID, Vikingos, Bucaneros, U.S. Bank Stadium, Calif.
ID, Gigantes, Vaqueros, MetLife Stadium, Calif.
...

Semana 11
Semana 16, Lunes
```

2. Solución

Una solución es una propuesta de calendario donde se reparten los 17 partidos y un descanso para cada uno de los 32 equipos en las 18 semanas de la temporada. Con información de si el horario en el que se juega el partido estelar o no y a qué estelar corresponde. La Figura 1 muestra el calendario para la temporada de este año.

2023 NFL Schedule Grid

	WK 1	WK 2	WK 3	WK 4	WK 5	WK 6	WK 7	WK 8	WK 9	WK 10	WK 11	WK 12	WK 13	WK 14	WK 15	WK 16	WK 17	WK 18
ARI	at WAS	NYG	DAL	at SF	CIN	at LAR	at SEA	BAL	at CLE	ATL	at HOU	LAR	at PIT	BYE	SF	at CHI	at PHI	SEA
ATL	CAR	GB	at DET	JAX	HOU	WAS	at TB	at TEN	MIN	at ARI	BYE	NO	at NYJ	TB	at CAR	IND	at CHI	at NO
BAL	HOU	at CIN	IND	at CLE	at PIT	TEN	DET	at ARI	SEA	CLE	CIN	at LAC	BYE	LAR	at JAX	at SF	MIA	PIT
BUF	at NYJ	LV	at WAS	MIA	JAX	NYG	at NE	TB	at CIN	DEN	NYJ	at PHI	BYE	at KC	DAL	at LAC	NE	at MIA
CAR	at ATL	NO	at SEA	MIN	at DET	at MIA	BYE	HOU	IND	at CHI	DAL	at TEN	at TB	at NO	ATL	GB	at JAX	TB
CHI	GB	at TB	at KC	DEN	at WAS	MIN	LV	at LAC	at NO	CAR	at DET	at MIN	BYE	DET	at CLE	ARI	ATL	at GB
CIN	at CLE	BAL	LAR	at TEN	at ARI	SEA	BYE	at SF	BUF	HOU	at BAL	PIT	at JAX	IND	MIN	at PIT	at KC	CLE
CLE	CIN	at PIT	TEN	BAL	BYE	SF	at IND	at SEA	ARI	at BAL	PIT	at DEN	at LAR	JAX	CHI	at HOU	NYJ	at CIN
DAL	at NYG	NYJ	at ARI	NE	at SF	at LAC	BYE	LAR	at PHI	NYG	at CAR	WAS	SEA	PHI	at BUF	at MIA	DET	at WAS
DEN	LV	WAS	at MIA	at CHI	NYJ	at KC	GB	KC	BYE	at BUF	MIN	CLE	at HOU	at LAC	at DET	NE	LAC	at LV
DET	at KC	SEA	ATL	at GB	CAR	at TB	at BAL	LV	BYE	at LAC	CHI	GB	at NO	at CHI	DEN	at MIN	at DAL	MIN
GB	at CHI	at ATL	NO	DET	at LV	BYE	at DEN	MIN	LAR	at PIT	LAC	at DET	KC	at NYG	TB	at CAR	at MIN	CHI
HOU	at BAL	IND	at JAX	PIT	at ATL	NO	BYE	at CAR	TB	at CIN	ARI	JAX	DEN	at NYJ	at TEN	CLE	TEN	at IND
IND	JAX	at HOU	at BAL	LAR	TEN	at JAX	CLE	NO	at CAR	NE	BYE	TB	at TEN	at CIN	PIT	at ATL	LV	HOU
JAX	at IND	KC	HOU	ATL	BUF	IND	at NO	at PIT	BYE	SF	TEN	at HOU	CIN	at CLE	BAL	at TB	CAR	at TEN
KC	DET	at JAX	CHI	at NYJ	at MIN	DEN	LAC	at DEN	MIA	BYE	PHI	at LV	at GB	BUF	at NE	LV	CIN	at LAC
LAC	MIA	at TEN	at MIN	LV	BYE	DAL	at KC	CHI	at NYJ	DET	at GB	BAL	at NE	DEN	at LV	BUF	at DEN	KC
LAR	at SEA	SF	at CIN	at IND	PHI	ARI	PIT	at DAL	at GB	BYE	SEA	at ARI	CLE	at BAL	WAS	NO	at NYG	at SF
LV	at DEN	at BUF	PIT	at LAC	GB	NE	at CHI	at DET	NYG	NYJ	at MIA	KC	BYE	MIN	LAC	at KC	at IND	DEN
MIA	at LAC	at NE	DEN	at BUF	NYG	CAR	at PHI	NE	KC	BYE	LV	at NYJ	at WAS	TEN	NYJ	DAL	at BAL	BUF
MIN	TB	at PHI	LAC	at CAR	KC	at CHI	SF	at GB	at ATL	NO	at DEN	CHI	BYE	at LV	at CIN	DET	GB	at DET
NE	PHI	MIA	at NYJ	at DAL	NO	at LV	BUF	at MIA	WAS	IND	BYE	at NYG	LAC	at PIT	KC	at DEN	at BUF	NYJ
NO	TEN	at CAR	at GB	TB	at NE	at HOU	JAX	at IND	CHI	at MIN	BYE	at ATL	DET	CAR	NYG	at LAR	at TB	ATL
NYG	DAL	at ARI	at SF	SEA	at MIA	at BUF	WAS	NYJ	at LV	at DAL	at WAS	NE	BYE	GB	at NO	at PHI	LAR	PHI
NYJ	BUF	at DAL	NE	KC	at DEN	PHI	BYE	at NYG	LAC	at LV	at BUF	MIA	ATL	HOU	at MIA	WAS	at CLE	at NE
PHI	at NE	MIN	at TB	WAS	at LAR	at NYJ	MIA	at WAS	DAL	BYE	at KC	BUF	SF	at DAL	at SEA	NYG	ARI	at NYG
PIT	SF	CLE	at LV	at HOU	BAL	BYE	at LAR	JAX	TEN	GB	at CLE	at CIN	ARI	NE	at IND	CIN	at SEA	at BAL
SEA	LAR	at DET	CAR	at NYG	BYE	at CIN	ARI	CLE	at BAL	WAS	at LAR	SF	at DAL	at SF	PHI	at TEN	PIT	at ARI
SF	at PIT	at LAR	NYG	ARI	DAL	at CLE	at MIN	CIN	BYE	at JAX	TB	at SEA	at PHI	SEA	at ARI	BAL	at WAS	LAR
TB	at MIN	CHI	PHI	at NO	BYE	DET	ATL	at BUF	at HOU	TEN	at SF	at IND	CAR	at ATL	at GB	JAX	NO	at CAR
TEN	at NO	LAC	at CLE	CIN	at IND	BAL	BYE	ATL	at PIT	at TB	at JAX	CAR	IND	at MIA	HOU	SEA	at HOU	JAX
WAS	ARI	at DEN	BUF	at PHI	CHI	at ATL	at NYG	PHI	at NE	at SEA	NYG	at DAL	MIA	BYE	at LAR	at NYJ	SF	DAL
	SATURDAY		SUNDAY		SNF/NBC		MNF/ESPN/ABC		TNF	NFLN	THANKSGIVING		BLACK FRIDAY		CHRISTMAS DAY		INTERNATIONAL	

* - Sunday night games in Weeks 5-15 and Week 17 + MNF games in Weeks 12-15 subject to change; Week 16 NE-DEN game is Sunday 12/24 at 8:15 PM ET on NFL Network; Week 18 games TBD

Figura 1: Calendario oficial de la temporada 2023

Podemos pensarlo como una matriz tridimensional de tamaño *equipos* × *semana* × *horario*. Para la temporada 2023, por ejemplo, la matriz cuenta con $32 \cdot 18 \cdot 2 = 1152$ elementos.

2.1. Codificación

Encontrar una codificación adecuada es complicado debido a las restricciones impuestas por el hecho de ya tener los partidos que se jugarán. Decidimos entonces mantener una codificación *directa* basada en el calendario:

La solución codificada es una matriz con dimensiones *equipos* × *semanas* × 2 en donde la fila i representa al equipo con id i, la columna j representa a la semana $j+1$ y cada celda contiene el id del partido que se jugará y el horario.

Toda solución debe de cumplir con ciertas características:

- Cada fila debe de estar conformada alguna permutación de los partidos que juega el equipo correspondiente y su semana de descanso (al que representamos con el número de partidos, 272 para la temporada actual).

- Cada columna debe de contener los valores de los horarios correspondientes a su semana en pares (porque, idealmente, cada partido aparece dos veces en la semana, una en el local y otra en el visitante). Así entonces si en la semana 1 hay un partido de MNF, uno de TNF y uno de SNF y estos se codificación como 1, 2 y 3. Toda solución debe de tener en la columna 0: dos 1's, dos 2's, dos 3's y el resto ceros.

Ejemplo: Si tomamos en cuenta solo 6 equipos, tres semanas y un estelar por semana, una posible solución sería:

	WK 1	WK 2	WK 3
ARI	ATL	BAL	BUF
ATL	ARI	BUF	CAR
BAL	CAR	ARI	CHI
BUF	CHI	ATL	ARI
CAR	BAL	CHI	ATL
CHI	BUF	CAR	BAL

$\Longrightarrow$

0	1	2
0	3	4
5	1	6
7	3	2
5	8	4
7	8	6

Del lado izquierdo tenemos el fenotipo y del izquierdo el genotipo. En el ejemplo, el partido 0 es ARI contra ATL y se juega en el estelar de la semana, por eso está marcado en rojo. Al codificarlo queda en la ceda (0,0) y (1,0) porque los id de los equipo son 0 y 1. Si codificamos el estelar como 1 las celdas tendrían 1 en el horario por ser estelar, todas las demás celdas de la semana tendrán 0.

En general trabajamos con la versión tridimensional, pero para guardar las soluciones las aplanamos. Siguiendo el ejemplo, la solución aplana sería:

0	1	1	0	2	1	0	1	3	0	4	0	5	$\cdots$

2.2. Solución factibles e infactibles

Definimos a una solución **factible** como aquella cuyo calendario se puede implementar en la vida real, para lo que debe de cumplir algunas características extra a las dadas anteriormente: En cada semana deben de aparecer todos los partidos dos veces, en la fila del equipo local y la del visitante, además ambas apariciones deben de tener el mismo horario.

Como el problema al que nos enfrentamos es una generalidad del problema de calendarización, conseguir estas soluciones en tiempo polinomial no es posible por lo que debemos aceptar que tendremos soluciones infactibles en nuestro algoritmo y nos apoyaremos en la función de evaluación para tratar de llegar a soluciones factibles.

2.3. Espacio de búsqueda

Podemos partir el espacio en dos: El espacio para los equipos (dimensión $m \times n \times 0$) y el espacio para los horarios (dimensión $m \times n \times 1$).

En el caso de los equipos, cada fila es un permutación de 17 partidos y BYE y tenemos 32 filas, así que el tamaño del espacio es $18! * 32 \approx 2 \times 10^{17}$.

Para el caso de los horarios, hay hasta 16 partidos por semana y las semanas comunes tienen tres estelares (MNF, TNF y SNF), para esos casos hay $16!/13! = 3360$ posibles permutaciones. En acción de gracias hay $16!/11! = 524160$ posibilidades y en navidad hay, a lo más, $16!/10! = 5765760$. Esto da un total aproximado de

$$3360 \cdot 14 \cdot 524160 \cdot 5765760 \approx 1.42 \times 10^{17}$$

Por lo que el **tamaño estimado** del espacio de búsqueda es:

$$2 \times 10^{17} \cdot 1.42 \times 10^{17} = 2.82 \times 10^{34}$$

2.4. Ventajas y desventajas

Utilizar una codificación directa nos facilita evaluar e interpretar soluciones pero complica drásticamente trabajar con ella en términos de operadores. También notamos haber terminado con una representación muy sensible a cambios y con información repetida. Otras características que notamos fueron:

Ventajas

- Es una representación completa, se puede codificar toda solución posible en el espacio.
- Las reglas agregadas a filas y columnas reducen considerablemente el espacio de búsqueda.
  - El tamaño del espacio sin reglas es $(272 + 32)! = 304! \approx 2.56 \times 10^{625}$.

Desventajas

- Es una representación compleja y sensible a cambios.
- Es dependiente de los ejemplares (orden y tamaño).
- Depende de varias reglas para dar soluciones factibles, ni siquiera podemos asegurarlas.
- Complica cualquier operador que queramos implementar.
- Contiene información repetida.

3. Función de Optimización

Para la evaluación de soluciones optamos por un enfoque multi-objetivo basado en penalizaciones. La función de evaluación f se ve como un conjunto de reglas $R = \{r_1, r_2, \ldots, r_m\}$ en donde cada regla se puede pensar como una función que devuelve la penalización de la solución. Con esto la evaluación se ve como

$$f(s) = MAX\_VALOR - \sum_{r \in R} r(s)$$

MAX_VALOR nos permite ver a la función como de maximización y mantener todos los valores positivos. Una solución óptima tiene MAX_VALOR y la peor tiene un valor de 0.

3.1. Reglas

Dividimos a las reglas en dos grandes grupos: reglas **duras**, aquellas que la NFL busca asegurar en todos sus calendarios, y reglas **blandas**, que son deseables pero que no toman prioridad en la elección del calendario.

3.1.1. Reglas duras

Para diseñar el factor de penalización de estas reglas buscamos asegurarlos que tomen la prioridad, procurando que una solución que rompa una sola regla dura, aun cuando cumpla todas las blandas, sea peor que una que cumple con todas las duras.

Retomando lo que habíamos comentado sobre la factibilidad de las soluciones, agregamos como una regla dura verificarlo, esta regla es especial y cuenta con un peso extra, porque encontrar una que la cumpla es tarea fundamental.

Además de esta regla contamos con 6 más que la NFL utilizó para esta temporada (es importante mencionar que las reglas cambian año con año), la mayoría buscan asegurar factores de validez para la temporada y es sabido que siempre pueden cumplirse todas.

3.1.2. Reglas blandas

Estas reglas tienen menos pero porque la NFL las ve como reglas ideales pero que no siempre pueden cumplirse, y es que varias de estas reglas colisionan entre sí y no se pueden asegurar para todos los equipos, por lo que se buscan soluciones que las optimicen. Un ejemplo es minimizar la cantidad de equipos que juegan tres partidos consecutivos como visitante porque se considera una desventaja. De este tipo de reglas tenemos 7.

3.1.3. Listado de reglas

Como se mencionó, hay 7 reglas duras y 7 blandas. Debemos aclarar que la NFL utiliza muchas reglas más pero varias de ellas requieren información que es imposible de conseguir para analizar, esto nos afecta porque una solución perfecta puede no funcionar para la NFL por factores que no se tomaron en cuenta.

Las 14 reglas utilizadas son:

Reglas duras

- **HorarioFactible**. Verifica que los partidos aparezcan en ambos equipo que lo juegan. Su máxima evaluación es 2720000, 10000 por partido inválido.
- **PartidosTDAY**. Verifica que los partidos en el horario TDAY contengan a los equipos DAL (Vaqueros) y DET (Leones). Su máxima evaluación es 200, 100 por equipo no contenido.
- **NoMasDeSeisEstelares**. Verifica que ningún equipo juegue más de seis partidos en horario estelar. Su máxima evaluación es 3200, 100 por equipo que no lo cumpla.
- **NoCuatroJuegosFuera**. Verifica que ningún equipo juegue cuatro o más juegos consecutivos como visitante. Su máxima evaluación es 19200, con 100 por partido por encima de los 3 consecutivos.
- **ByeEnSemanasValidas**. Verifica que no haya descanso en las primeras 4 y las últimas 4 semanas de la temporada. Su máxima evaluación es de 3200, 100 por cada descanso en una semana invalida.
- **NoMasDeSeisByes**. Verifica que en una semana dada no haya mas de seis equipos descansando. Su máxima evaluación es de 1800, 100 por semana que no lo cumpla.
- **NoEstelaresEnBye**. Verifica que no se asigne un descanso en un horario estelar, esta regla es necesaria por las soluciones que no son factibles. Su máxima evaluación es de 6000, 100 por cada horario estelar que contenga un descanso.

Reglas blandas

- **NoTresJuegosFuera**. Verifica que ningún equipo juegue tres juegos consecutivos como visitante. Su máxima evaluación es 192, 1 por secuencia de partidos consecutivos como visitante.
- **CalificacionHorarios**. Suma los valores que se le asignan a los partidos estelares. Su máxima evaluación es 240, con el valor asignado por partido variando dependiendo del interés que le consideramos.
- **TodosHorarioEstelar**. Verifica que todo equipo tenga al menos un partido en horario estelar. Su máxima evaluación es de 32, 1 por equipo que no tenga algún partido estelar.
- **NoByesTempranosConsecutivos**. Verifica que aquellos equipos que tuvieron un descanso en la primera mitad de la temporada pasada no vuelvan a tenerlo. Su máxima evaluación es de 32, 1 por cada equipo que tenga temporadas consecutivas con descansos tempranos.
- **JuegosDivisionalesAlFinal**. Verifica en que mitad de la temporada se encuentren los partidos entre dos equipos de una misma división, penalizando si ocurren en la primera mitad. Su máxima evaluación es de 144, 1 por partido divisional en la primera mitad de la temporada.
- **NoMasDeDosHusosParaTNF**. Verifica que para un partido de jueves por la noche, ninguno de los equipos involucrados deba recorrer más de dos husos horarios. Su máxima evaluación es de 234. 1 por equipo que deba de recorrer dos o más husos horarios para un partido de TNF.

4. Algoritmo genético

El algoritmo consta de distintas partes fundamentales: La selección de la población inicial, los operadores de cruza y mutación, la selección de padres para la nueva generación y la forma de reemplazo en la nueva generación. Además, para nuestro caso particular, nos interesa la fase de pseudo-reparación de soluciones.

4.1. Fase de reparación

Comenzamos hablando de esto porque todos los puntos que implican alterar soluciones lo tratan.

Sabiendo que no podemos generar soluciones factibles desde un inicio y que mantenerlas es complicado, decidimos considerar una fase de reparación que genera soluciones muy cercanas a ser factibles y que, de encontrar una factible, trata de mantener esta propiedad en sus descendientes.

Para corregir los partidos seguimos el Algoritmo 1 que daría soluciones factibles si no fuera por casos donde algunas filas sobreescriben la corrección de las anteriores, pero nos permite pasar de tener 100 o 200 partidos mal posicionados por solución a tan solo 20.

Un punto interesante es que aplicar el mismo algoritmo tres veces a la solución minimiza el número de partidos mal

posicionados a más de la mitad.

Algoritmo 1 Algoritmo para pseudo-reparar equipos de una solución

Input: Solución
Output: Solución modificada

```
1: for fila ∈ solucion do
2:     equipo ← Equipo correspondiente a la fila
3:     for columna ∈ solucion do
4:         semana ← Semana correspondiente a la columnass
5:         partido ← solucion[fila][columna][0]
6:         if partido ≠ BYE then
7:             contra ← EquipoContra(equipo, partido)          /* Contra el que juega */
8:             col2 ← Columna donde esté partido en la fila de contra.
9:             Intercambiar solucion[contra.id][columna] y sol[contra.id][col1].
10:        end if
11:    end for
12: end for
```

Para corregir horarios simplemente nos aseguramos de que los partidos tengan el mismo horario en sus dos apariciones, esto reduce considerablemente el espacio de búsqueda y asegura que la dimensión de los horarios siempre sea válida.

4.2. Población Inicial

La población inicial se genera con permutaciones aleatorias de los partidos de cada equipo y de los horarios de cada semana y después se pseudo-reparan. No verificamos que no se repitan soluciones en la población pero el espacio de búsqueda es tan grande que es poco probable que eso suceda y, más aún, que se mantenga entre generaciones.

4.3. Selección de padres

Para este paso utilizamos un método de selección por ruleta en el que la probabilidad de cada individuo s de ser elegido está determinado por la ecuación

$$P(s) = \frac{f(s)}{\sum_{s' \in POB} f(s')}$$

Con POB la población actual. La probabilidad de cada individuo entonces es equivalente a la proporción de su evaluación con respecto a la suma total de las evaluaciones de todos los individuos; A mejor evaluación, mayor la probabilidad de ser elegido.

La ventaja que obtenemos con este método es la alta probabilidad de selección que tienen los individuos más aptos, lamentablemente esto también presenta la desventaja de afectar la diversidad de la nueva generación al casi siempre tomar a los mejores individuos como padres, lo que puede provocar que el algoritmo converja demasiado rápido.

4.4. Estrategia de reemplazo

Utilizamos una estrategia reemplazo generacional en donde todos los padres son sustituidos por sus hijos. Tomamos también en cuenta el **elitismo** permitiendo que el individuo más apto de la población pase a la siguiente generación directamente.

4.5. Operadores

La codificación complicó mucho estos operadores pero tratamos de cargar la complejidad en las funciones de reparación.

4.5.1. Cruce

Inicialmente implementamos dos algoritmos distintos: Un **cruce por filas** que tomaba una sección horizontal de la solución de un padre y rellenaba el resto del otro utilizando OX1 para mantener las permutaciones válidas y un **cruce por columnas** que tomaba una sección horizontal y reparaba las soluciones utilizando PMX.

Estos algoritmos no tomaban en cuenta la fase de reparación y su desempeño fue muy pobre (89443 de un máximo de 2754720 para nuestra mejor ejecución), por lo que desechamos esta idea.

El operador utilizado al final fue un **cruce por filas** con fase de reparación, el Algoritmo 2 muestra el proceso para generar un hijo. Al inicio nos preocupaba que no se alterara lo suficiente a las soluciones en la dimensión de los horarios por copiar de un sólo padre pero notamos que su fase de reparación se ve afectada por el nuevo orden de los horarios, asegurando diversidad en los nuevos individuos.

Algoritmo 2 Algoritmo de cruza para un hijo

Input: Dos soluciones padres
Output: Solución hijo generada

1: $m \leftarrow \text{RANDOM}(0, |equipos| - 2)$
2: $n \leftarrow \text{RANDOM}(m + 1, |equipos| - 1)$ `/*` *Así* $0 \leq m < n < |equipos|$ `*/`
3: En la dimensión de los partidos, copiar las filas m a n de uno de los padres y el resto del otro.
4: La dimensión de los horarios se copia tal cual del primer padre.
5: Pseudo-repara la solución generada.

4.5.2. Mutación

Al igual que con los operadores de cruza, inicialmente teníamos dos operadores de mutación, por **filas** y por **columnas**, que no tomaban en cuenta la fase de reparación. Como también dieron malos resultados los cambiamos por un único operador de cruza simple: Seleccionar un equipo aleatorio e intercambiar dos partidos de ese equipo de lugar en la solución, un simple SWAP.

Habíamos considerado pseudo-reparar las soluciones después del SWAP pero notamos que la reparación llegaba a sobreescribir el cambio. Después de probar, vimos que mantener la solución sin reparar daba bueno resultados y permitía al algoritmo escapar de mínimos locales.

4.6. Pseudocódigo

En el Algoritmo 3 podemos ver el desarrollo de los puntos antes mencionados:

Comenzamos inicializando la población y descubriendo al mejor individuo para el paso elitista y para el criterio de término y, mientras no se llegue a la solución óptima ni se termine el tiempo, se obtiene la nueva generación con los hijos de los individuos de la generación actual y con el mejor de esta sin cambios.

Como el algoritmo de cruza requiere de dos padres, se seleccionan dos individuos de la población por ruleta, de los que se obtienen dos hijos (según la probabilidad de cruza, puede que sean los mismos padres). Y estos hijos se mutan con cierta probabilidad y se agregan a la nueva población.

Al final devolvemos al mejor individuo de la última generación, que puede o no ser óptimo.

5. Implementación

Todo el código se implementó utilizando Python3

5.1. Ejemplares

Los ejemplares se representan en la clase `TemporadaNFL` del archivo `temporada.py`. Un ejemplar contiene:

- El número de semanas y de equipos como `num_semana` y `num_equipos`.
- Semana correspondiente a `thanksgiving`.

Algoritmo 3 Algoritmo genético

Input: *num_individuos*, *timepo_limite*, *p_cruza*, *p_muta*
Output: Mejor solucion encontrada

```
1: poblacion ← PoblacionInicial(num_individuos)
2: mejor ← MejorIndividuo(poblacion)
3: while tiempo_transcurrido < tiempo_limite or mejor es optimo do
4:     nueva_poblacion ← Lista vacía
5:     Agregar mejor a nueva_poblacion                          /* Paso elitista */
6:     while Longitud(nueva_poblacion) ≠ num_individuos do
7:         padre1 ← SelecionaIndividuo(poblacion)
8:         padre2 ← SelecionaIndividuo(poblacion)
9:         hijos ← CruzaPadres(padre1, padre2, p_cruza)
10:        hijos ← Muta(hijos, p_muta)
11:        Agregar hijos a nueva_poblacion        /* Con cuidado de no pasarse del num. de individuos */
12:    end while
13:    poblacion ← nueva_poblacion
14:    mejor ← MejorIndividuo(nueva_poblacion)
15: end while
16: return mejor
```

- Tupla (día, semana) correspondiente a `navidad`.
- Diccionario con los posibles `horarios` (MNF, TNF, SNF, TDAY, XMAS, NONE) y sus respectivas codificaciones a enteros.
- Tupla con las semanas que no tienen horarios estelares como `semanas_sin_horarios`.
- Codificación del BYE (272 para la temporada actual).
- `max_calif_partido` para saber la mejor califiación que un partido puede tener.
- Una tupla con los `equipos`. Cada equipo es un diccionario con la llaves: acronimo, conferencia, division, bye_anterior (BYE de la temporada anterior), 3_consecutivos (booleano para saber si tuvo 3 partidos consecutivos como visitante la temporada anterior) y partidos (una lista con el id de los 17 partidos que juega). El id de cada equipo es su posición en la tupla.
- Una tupla con los `estadios`. Cada estadio es un diccionario con la llave huso, que representa su huso horario: PST, MST, CST o EST. Lo hacemos un diccionario apesar de sólo ser un valor para posible extensión del problema a casos con partidos internacionales. El id de del estadio es su posición en la tupla.
- Una tupla con los 272 `partidos` de la temporada. Cada partido es un diccionario con las llaves: local, visitante, estadio y calificacion (calificación del partido según la métrica mencionada en la codificación de ejemplares). El id del partido es su opsición en la tupla.

Además de esto, el ejemplar tiene ciertas funciones para poder utilizarlo: `leer_archivo` es una función de clase que permite generar instancias de temporadas codificadas desde el archivo txt. `leer_solucion` y `guardar_solucion` leen y guardan soluciones codificadas en archivos txt. `horarios_semana` devuelve una lista con los horarios se la semana requerida. `verifica_solucion` y `verifica_factibilidad` analizan soluciones a detalles (mayormente usada para debug). Y `equipo_contra` permite conocer rápidamente al contrincante de un equipo en algún partido.

5.2. Función de evaluación

Todo lo correspondiente a la evaluación está en la carpeta `evaluacion`.

Como las funciones de evaluación cambian año con año agregamos una clase abstracta `EvaluacionNFL` en el archivo `evaluacion.py` que contiene la función para evaluar una solución según las reglas que se decidan y otra para analizar una solución (evaluar con más detalle). Las clases que la heredan deben de implementar `carga_reglas` para saber las reglas que usará.

Todo lo relacionado a reglas está en la carpeta `reglas`. Toda regla debe de heredar de la clase `Regla` (en `regla.py`) e implementar `max_val`, para conocer su máxima penalización, y `evalua`, que es la encargada de penalizar a la solución que reciba. Las reglas duras además tienen un factor de penalización extra de 100.

Todas las reglas duras están en `duras.py` y todas las blandas en `blandas.py`.

5.3. Algoritmo genético

Todo el algoritmo quedó implementado en la clase `AlgoritmoGenetico` dentro de `genetico.py` que se inicializa con una instancia de un ejemplar de `TemporadaNFL` y de una función de evaluación `EvaluacionNFL` y que se puede ejecutar con el método `ejecutar`, al final devuelve un diccionario con los datos obtenidos durante la ejecución.

También agregamos el archivo `genetico` para ejecutar el algoritmo desde terminal y ver un resumen de los resultados. El script recibe, en orden: el ejemplar, el tamaño de la población, el tiempo límite en segundos, la probabilidad de cruza, la probabilidad de mutación y opcionalmente la semilla y el nombre de un archivo para guardar la solución.

También agregamos el script `evalua` para evaluar soluciones guardarlas, el script no sólo muestra la evaluación sino un analisis de la penalización de cada regla.

Ejemplo: `./genetico ../data/temporada2023.txt 100 30 0.9 0.01 42 sol.txt` ejecutaría el algoritmo para la temporada 2023 con población de 100 y tiempo de 30 segundos, con probabilidades para cruza y mutación de 0.9 y 0.01, respectivamente, y con semilla 42 y guardaría la solución en `sol.txt`. Si usamos `./evalua ../data/temporada2023 sol.txt` podemos ser la evaluación de nuevo.

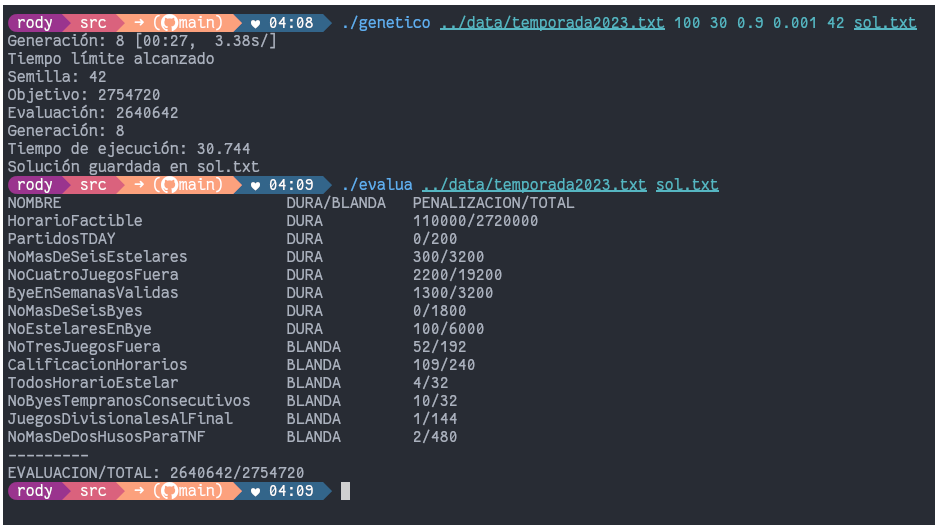

Figura 2: Ejemplo de uso desde terminal

5.3.1. Funciones de reparación

La reparación de partidos (filas) está en la función `repara_filas` y es una implementación del Algoritmo 1. Para optimizar la búsqueda de los partidos para los intercambio generamos al inicio un diccionario por equipo con sus partidos como llaves y su posición en la solución como valor.

La reparación de horarios (columnas) está en `repara_columnas` y sigue un algoritmo similar al de la reparación de filas pero, en lugar de utilizar un diccionario con un sólo valor para encontrar los índices de reemplazo, manejamos listas con los espacios libres de cada valor. Lo hacemos de esta manera para poder manejar los valores repetidos adecuadamente.

5.3.2. Inicialización de la población

Está en `inicializa_poblacion`. Los individuos, además de corresponder a la solución también tienen su evaluación consigo. Para cada equipo se obtienen sus partido del `ejemplar` y se permutan usando `np.random.shuffle` desde el generador de aleatorios. Se hace lo mismo para los horarios de cada semana.

5.3.3. Operador de cruza

Se encuentra en `cruza_filas` (su nombre quedó así por la implementación anterior). Es una implementación del Algoritmo 2. Para los índices generamos primero uno entre 0 y $|equipos| - 2$ y luego el otro entre el primero más 1 y $|equipos| - 1$, así aseguramos que el segundo siempre sea mayor.

Al final de la cruza, ambos hijos generados pasan por la fase de pseudo-reparación.

5.3.4. Operador de mutación

Implementado en `muta_filas`. Seleccionamos un equipos aleatorio usando el `rng` (generador de aleatorios), luego generamos dos semanas distintas aleatorias y las intercambiamos de lugar. Como habíamos mencionado, este operador no pasa por la fase de reparación.

5.3.5. Selección de padres

Para la sección calculamos la probabilidad de los individuos y la función de probabilidad acumulativa (la obtenemos con ayuda de `np.cumsum`) al actualizar la población y, en la selección por ruleta vemos a la ruleta como una recta de 0 a 1 y la probabilidad acumulativa nos dice cuanta sección de recta le corresponde a cada individuo. Obtenemos un número entre 0 y 1 aleatorio y con eso seleccionamos al individuo.

La función de ruleta está implementada en `selecciona_padre`, recibe además el número de padres a elegir y se asegura de devolver siempre dos padres en índices distintos de la población, esto no asegura que los padres no sean igual si la población permite repetidos, como es nuestro caso.

5.3.6. Reemplazo generación

`poblacion_generacional` es la función que implementa todo el proceso principal del algoritmo, es donde se seleccionan padres, se crean hijos, se mutan y se agregan a la nueva población. Esta función representa una nueva generación en el algoritmo.

Además, siguiendo el punto elitista, siempre agregamos al mejor individuo de la población actual a la nueva directamente.

5.4. Generación de tablas y gráficas

El archivo `ejecuciones.py` se encarga de realizar las ejecuciones del algoritmo necesarios y de guardar los datos en archivos json dentro de una carpeta `data`. `lectura.py` es la que se encarga de leer estos archivos y generar la gráfica y tabla de resultados.

5.5. Visualización de soluciones

`solucion-a-excel` recibe un ejemplar, una solución codificada y el nombre de la salida y genera un archivo excel con el mismo formato que utiliza la NFL (como el de la Figura 1).

6. Resultados

Lamentablemente no tuvimos tiempo de implementar resultados para otra temporada que no fuera la de este año, conseguir la información es complicado y mucho del trabajo hay que hacerlo a mano. Al final sólo realizamos 5 ejecuciones con semillas distintas sobre este ejemplar y obtuvimos el promedio de la evolución de óptimo contra el del mejor individuo. La Tabla 3 muestra los parámetros utilizados.

El resultado promedio obtenido de las 5 ejecuciones se resume en la Tabla 4

La Figura 5 muestra la evolución de la evaluación óptima en la mejor ejecución con respecto a la evolución promedio en las 5 ejecuciones. Las generación están en un factor de 100, así que la linea en el eje x con 17.5 es en realidad la generación 1750.

En general notamos un buen desempeño del algoritmo, de hecho la gráfica sugiere que los ejemplares podían seguir mejorando con más tiempo de ejecución.

Nombre	Descrición	Valor
ITERACIONES	Números de iteraciones	5
TAM_POB	Tamaño de la población	100
P_MUTACION	Probabilidad de mutación	0.01
P_CRUZA	Probabilidad de cruza	0.9
T_LIMITE	Tiempo límite	2400 seg.

Figura 3: Tabla de parámetros

Ejemplar	Promedio generaciones	Mejor eval	Promedio eval	Peor eval
temporada2023.txt	1839.4	2741542	2715138.2	2700226

Figura 4: Resumen de las ejecuciones

6.1. Evaluación de soluciones

Para tener una base, codificamos el calendario de la NFL (Figura 1) y lo evaluamos, los resultados están en la Figura 6. Como esperábamos, es una solución casi perfecta, aunque nos sorprendió ver que rompe una de sus reglas duras: Uno de los equipos tiene mas de seis estelares, investigando un poco parece ser que están buscando holgar reglas relacionadas a estelares.

Además de las 5 iteraciones mostradas, ejecutamos por un par de horas una ejecución individual para tratar de buscar una solución cercana a la óptima. La Figura 7 muestra la evolución del óptimo en esta ejecución y la Figura 8 muestra la solución encontrada, además la Figura 9 muestra su evaluación.

7. Conclusiones

El algoritmo es capaz de llegar a soluciones con buenas evaluaciones y no parece tener problemas graves de convergencia prematura pues todas las ejecuciones que hicimos siempre parecían poder mejorar.

Nuestra mejor solución está unos 3000 puntos abajo de la la de NFL. fue satisfactorio poder obtener una solución factible pero notamos que donde más problemas tiene el algoritmo es en acomodar los BYE fuera de las primeras y últimas 4 semanas de la temporada.

Creemos que el enfoque tomado es funcional pero probablemente algoritmos como búsqueda local iterada o recocido simulado puedan conseguir mejores resultados porque pudimos notar que sí hay variantes del operador de mutación que mantienen a las soluciones como factibles, así que una vez se alcanza una de estas se puede comenzar a optimizar sobre las reglas sin preocuparse por soluciones inválidas.

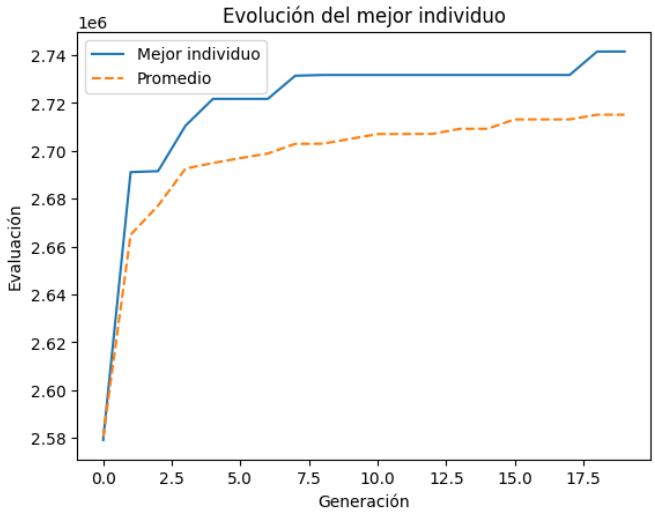

Figura 5: Evolución promedio

NOMBRE	DURA/BLANDA	PENALIZACION/TOTAL
HorarioFactible	DURA	0/2720000
PartidosTDAY	DURA	0/200
NoMasDeSeisEstelares	DURA	100/3200
NoCuatroJuegosFuera	DURA	0/19200
ByeEnSemanasValidas	DURA	0/3200
NoMasDeSeisByes	DURA	0/1800
NoEstelaresEnBye	DURA	0/6000
NoTresJuegosFuera	BLANDA	5/192
CalificacionHorarios	BLANDA	90/240
TodosHorarioEstelar	BLANDA	4/32
NoByesTempranosConsecutivos	BLANDA	10/32
JuegosDivisionalesAlFinal	BLANDA	3/144
NoMasDeDosHusosParaTNF	BLANDA	4/480

EVALUACION/TOTAL: 2754504/2754720

Figura 6: Evaluación del calendario NFL

Referencias

[1] Sharp Football Analysis. 2022 nfl regular season schedule grid & strength of schedule. `https://www.sharpfootballanalysis.com/analysis/2022-nfl-regular-season-schedule-grid-strength-of-schedule/`.

[2] Andrew Brian Goldberg. Highly constrained sports scheduling with genetic algotihms. *Amherst College*, 2003.

[3] Gurobi. Creating the nfl schedule with mathematical optimization. `https://www.gurobi.com/events/creating-the-nfl-schedule-with-mathematical-optimization/`. Accedido 2023-04-27.

[4] NFL Operations. Creating the nfl schedule. `https://operations.nfl.com/gameday/nfl-schedule/creating-the-nfl-schedule/`. Accedido 2023-04-27.

[5] Anant J Umbarkar and Pranali D Sheth. Crossover operators in genetic algorithms: a review. *ICTACT journal on soft computing*, 6(1), 2015.

[6] Amrith Deepak y Benjamin Teo y Yihao Yang. An introduction to the national football league scheduling problem using integer programming. *Carnegie Mellon University*, 2015.

[7] Bistra N. Dilkina y William S. Havens. The u.s. national football league scheduling problem. *Cornell University*, 2004.

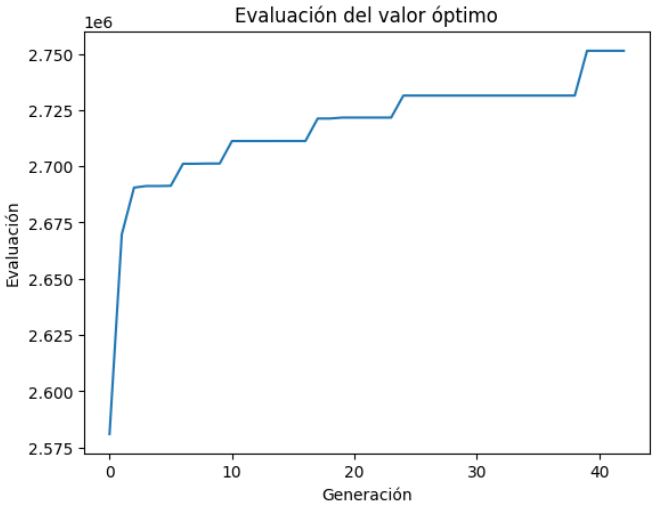

Figura 7: Evolución de la mejor solución encontrada

	WK 1	WK 2	WK 3	WK 4	WK 5	WK 6	WK 7	WK 8	WK 9	WK 10	WK 11	WK 12	WK 13	WK 14	WK 15	WK 16	WK 17	WK 18
ARI	LAR	at SF	at CLE	at HOU	CIN	at PIT	SF	at WAS	at LAR	ATL	at CHI	BYE	NYG	at PHI	at SEA	BAL	SEA	DAL
ATL	TB	at CAR	at TB	CAR	IND	at TEN	at DET	NO	at JAX	at ARI	BYE	GB	at NYJ	HOU	MIN	at NO	at CHI	WAS
BAL	DET	MIA	at PIT	SEA	at CLE	BYE	CLE	PIT	CIN	at TEN	at JAX	at SF	HOU	at CIN	LAR	at ARI	at LAC	IND
BUF	NYJ	TB	JAX	at PHI	at KC	BYE	LV	at LAC	at NE	DAL	at MIA	at WAS	DEN	NE	at CIN	MIA	NYG	at NYJ
CAR	at SEA	ATL	NO	at ATL	TB	MIN	DAL	HOU	at TB	at JAX	at NO	at CHI	at TEN	at DET	IND	GB	at MIA	BYE
CHI	LV	at GB	MIN	at TB	at LAC	BYE	at NO	GB	at MIN	at WAS	ARI	CAR	at CLE	at KC	at DET	DEN	ATL	DET
CIN	at KC	LAR	HOU	at TEN	at ARI	at SF	PIT	at CLE	at BAL	BYE	at PIT	at JAX	SEA	BAL	BUF	IND	MIN	CLE
CLE	BYE	TEN	ARI	SF	BAL	at IND	at BAL	CIN	at HOU	at PIT	at DEN	at SEA	CHI	at LAR	PIT	JAX	NYJ	at CIN
DAL	at MIA	at LAC	at PHI	NYJ	DET	WAS	at CAR	SEA	at NYG	at BUF	at SF	LAR	BYE	NYG	PHI	at WAS	NE	at ARI
DEN	GB	at KC	LV	MIN	at HOU	at LV	BYE	NYJ	WAS	KC	CLE	LAC	at BUF	at LAC	NE	at CHI	at DET	at MIA
DET	at BAL	BYE	GB	at LAC	at DAL	at GB	ATL	at MIN	at NO	LV	at TB	at KC	MIN	CAR	CHI	SEA	DEN	at CHI
GB	at DEN	CHI	at DET	NO	at NYG	DET	BYE	at CHI	at PIT	LAR	MIN	at ATL	LAC	TB	KC	at CAR	at LV	at MIN
HOU	NO	BYE	at CIN	ARI	DEN	at JAX	JAX	at CAR	CLE	IND	at IND	at TEN	at BAL	at ATL	at NYJ	PIT	TEN	TB
IND	JAX	at JAX	at NE	LV	at ATL	CLE	TB	at TEN	TEN	at HOU	HOU	BYE	PIT	NO	at CAR	at CIN	LAR	at BAL
JAX	at IND	IND	at BUF	KC	at PIT	HOU	at HOU	at TB	ATL	CAR	BAL	CIN	BYE	TEN	at TEN	at CLE	SF	at NO
KC	CIN	DEN	MIA	at JAX	BUF	at LAC	at MIN	LV	LAC	at DEN	at NYJ	DET	at LV	CHI	at GB	PHI	BYE	at NE
LAC	BYE	DAL	at NYJ	DET	CHI	KC	at NE	BUF	at KC	MIA	LV	at DEN	at GB	DEN	at LV	at MIN	BAL	at TEN
LAR	at ARI	at CIN	SF	PIT	BYE	NO	PHI	at NYG	ARI	at GB	SEA	at DAL	WAS	CLE	at BAL	at SF	at IND	at SEA
LV	at CHI	NYJ	at DEN	at IND	NE	DEN	at BUF	at KC	at MIA	at DET	at LAC	PIT	KC	MIN	LAC	NYG	GB	BYE
MIA	DAL	at BAL	at KC	NYG	at PHI	at NE	TEN	NE	LV	at LAC	BUF	NYJ	BYE	at NYJ	at WAS	at BUF	CAR	DEN
MIN	SF	NO	at CHI	at DEN	BYE	at CAR	KC	DET	CHI	at PHI	at GB	TB	at DET	at LV	at ATL	LAC	at CIN	GB
NE	at NYG	at PIT	IND	BYE	at LV	MIA	LAC	at MIA	BUF	at NYJ	WAS	NO	PHI	at BUF	at DEN	NYJ	at DAL	KC
NO	at HOU	at MIN	at CAR	at GB	TEN	at LAR	CHI	at ATL	DET	NYG	CAR	at NE	BYE	at IND	at TB	ATL	TB	JAX
NYG	NE	at WAS	WAS	at MIA	GB	SEA	NYJ	LAR	DAL	at NO	at PHI	PHI	at ARI	at DAL	at SF	at LV	at BUF	BYE
NYJ	at BUF	at LV	LAC	at DAL	WAS	BYE	at NYG	at DEN	PHI	NE	KC	at MIA	ATL	MIA	HOU	at NE	at CLE	BUF
PHI	WAS	at SEA	DAL	BUF	MIA	at TB	at LAR	SF	at NYJ	MIN	NYG	at NYG	at NE	ARI	at DAL	at KC	at WAS	BYE
PIT	TEN	NE	BAL	at LAR	JAX	ARI	at CIN	at BAL	GB	CLE	CIN	at LV	at IND	at SEA	at CLE	at HOU	BYE	SF
SEA	CAR	PHI	at TEN	at BAL	SF	at NYG	WAS	at DAL	at SF	BYE	at LAR	CLE	at CIN	PIT	ARI	at DET	at ARI	LAR
SF	at MIN	ARI	at LAR	at CLE	at SEA	CIN	at ARI	at PHI	SEA	BYE	DAL	BAL	TB	at WAS	NYG	LAR	at JAX	at PIT
TB	at ATL	at BUF	ATL	CHI	at CAR	PHI	at IND	JAX	CAR	BYE	DET	at MIN	at SF	at GB	NO	TEN	at NO	at HOU
TEN	at PIT	at CLE	SEA	CIN	at NO	ATL	at MIA	IND	at IND	BAL	BYE	HOU	CAR	at JAX	JAX	at TB	at HOU	LAC
WAS	at PHI	NYG	at NYG	BYE	at NYJ	at DAL	at SEA	ARI	at DEN	CHI	at NE	BUF	at LAR	SF	MIA	DAL	PHI	at ATL
	SATURDAY		SUNDAY		SNF/NBC		MNF/ESPN/ABC		TNF	NFLN	THANKSGIVING				CHRISTMAS DAY		INTERNATIONAL	

Figura 8: Mejor solución encontrada

```
NOMBRE                      DURA/BLANDA   PENALIZACION/TOTAL
HorarioFactible             DURA          0/2720000
PartidosTDAY                DURA          0/200
NoMasDeSeisEstelares        DURA          300/3200
NoCuatroJuegosFuera         DURA          1300/19200
ByeEnSemanasValidas         DURA          1200/3200
NoMasDeSeisByes             DURA          0/1800
NoEstelaresEnBye            DURA          0/6000
NoTresJuegosFuera           BLANDA        33/192
CalificacionHorarios        BLANDA        100/240
TodosHorarioEstelar         BLANDA        2/32
NoByesTempranosConsecutivos BLANDA        10/32
JuegosDivisionalesAlFinal   BLANDA        4/144
NoMasDeDosHusosParaTNF      BLANDA        6/480
---------
EVALUACION/TOTAL: 2751765/2754720
```

Figura 9: Evaluación de la mejor solución encontrada