Algoritmos y Estructuras de Datos III

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Trabajo Práctico III

Darwin en línea

Integrante	LU	Correo electrónico
Dalmau, Franco	572/17	franco.dalmau@gmail.com
Huaier, Damián Fernando	229/17	damianhuaier@gmail.com
Marenco, Mateo	516/17	mateomarenco74@gmail.com
Salgado, Martín	705/16	cruz.sal97@gmail.com

Contents

1 Introducción

En este trabajo nos proponemos implementar un programa capaz de jugar exitosamente al juego *C en línea*, una generalización del *4 en línea*, utilizando heurísticas y algoritmos genéticos. Primero veamos en qué consiste dicho juego.

El juego *C en línea* consiste en un tablero de N columnas y M filas y dos jugadores que cuentan con p fichas cada uno, cada jugador tiene fichas de un mismo color y las fichas de un jugador son de distinto color a las fichas del otro jugador. Los jugadores se alternan en poner las fichas en el tablero. En cada turno un jugador elige una columna para colocar la ficha y esta se ubica en la casilla de fila más baja disponible. El objetivo entonces es conseguir tener un línea horizontal, vertical o diagonal, de C fichas de un mismo color, y el primer jugador en conseguirlo es el ganador.

Para resolver este problema diseñamos una heurística golosa para determinar, dados ciertos parámetros y un tablero cualquiera, qué movimiento se conviene realizar, que será aquel que genere un tablero más ventajoso. Los parámetros mencionados consisten en una serie de valores numéricos que no varían a lo largo del partido, pero cada jugador tiene sus propios parámetros. Es decir que en definitiva cada jugador está definido por dichos parámetros.

El interés principal de este trabajo consiste en determinar un conjunto de parámetros óptimo, por lo tanto nos encontramos frente a un problema de optimización donde el espacio de soluciones está formado por todas las configuraciones de parámetros posibles y se desea encontrar la configuración que defina al mejor jugador entre todas las configuraciones.

El problema es que *a priori* no se cuenta con una *función objetivo* a optimizar que esté bien definida, puesto que no hay manera de medir qué tan bueno es un jugador de manera absoluta. De hecho esto nos lleva a preguntarnos qué significa que una solución o jugador sea óptimo, y lo cierto es que no está definido y por lo tanto debemos conformarnos con el hecho de que un jugador puede ser mejor en relación a otro u otros. Pero más aún, tampoco está definido qué significa que un jugador sea mejor que otro, ya que hay varias variables que hay que considerar, que son el valor de N, M, c y quién empieza con la primera jugada. Sumado a que la cantidad de jugadores posibles es demasiado grande, nos valdremos del uso de métodos heurísticos para obtener los mejores resultados posibles en un tiempo razonable, aunque sin garantías de que sean efectivamente buenos (de acuerdo a lo que nosotros entendemos subjetivamente por "buenos").

Para ello, utilizaremos las siguientes dos técnicas:

- *Grid-search*: para cada parámetro, se define un conjunto de posibles valores que puede tomar, y se recorren de forma exhaustiva todas las posibles combinaciones de esos valores para cada parámetro, y se evalúa qué tan bueno es cada jugador generado. El objetivo entonces es acercarse a un buen jugador, aunque qué significa un buen jugador dependerá de algún criterio que elijamos.

- Algoritmos genéticos: probamos diversos algoritmos genéticos que se diferencian en las funciones de *fitness* y métodos de selección utilizados, pero todos ellos respetan un esquema general que consiste en comenzar con un conjunto inicial de soluciones, evaluarlas, seleccionar a las mejores, cruzarlas y de esta manera obtener nuevas soluciones para la siguiente generación, que habrán heredado las características de las mejores soluciones. El proceso se repite hasta que se cumpla algún criterio de terminación. Una motivación para considerar algoritmos genéticos radica en que estos son capaces de evitar quedarse estancados en un máximo local. En este caso particular no está definido qué significa máximo local, pero intuitivamente nos referimos a una solución, es decir, un jugador que "supera" a muchos jugadores similares a él, pero sin que esto signifique que no pueda haber otro más distinto que lo supere. Y para evitar encontrar estos jugadores y no otros potencialmente mejores, los algoritmos genéticos son una buena elección.

2 Desarrollo

2.1 Heurística

Nuestra heurística se basa simplemente en evaluar los N posibles movimientos que se pueden realizar en cada turno (cada movimiento viene definido por la columna en la cual se inserta la ficha) y quedarnos con el mejor de ellos. Es decir toma la mejor jugada a realizar sin tener en cuenta turnos posteriores, lo que la hace una heurística golosa.

Para ello se cuenta con una función parametrizable de *evaluación* del tablero para un jugador en particular, es decir una función que dada una configuración del tablero, a quién le toca jugar y un jugador, le asigna un puntaje que representa qué tan ventajoso es ese tablero para dicho jugador. La heurística simplemente evalúa todos los N tableros que resultan de los N movimientos posibles y realiza el movimiento que genere el tablero con el puntaje más alto.

Nuestro primer diseño de la función de evaluación del tablero partió de la idea de que cada jugador busca tener la mayor cantidad de líneas posibles que puedan llegar a ser líneas de longitud C. Sin embargo, es claro que las líneas de mayor longitud son "mejores" que las de menor longitud, por más que se tengan muchas de estas.

Comenzamos entonces con la siguiente definición: dada una línea L del jugador para el cual se está evaluando el tablero (que puede ser horizontal, vertical o diagonal), definimos su *peso* como $f(L) = a_1|L|^{p_1}$, donde $|L|$ denota la longitud de L, y a_1 y p_1 son dos parámetros. De esta forma con un exponente p_1 muy grande se le da más importancia a las líneas más grandes.

Análogamente, si L no es del jugador para el cual se está evaluando el tablero, definimos su peso como $f(L) = a_2|L|^{p_2}$, es decir, se utilizan otros parámetros, siendo a_2 negativo puesto que las líneas del oponente suponen una amenaza. De esta forma habría jugadores que le den más peso a sus propias líneas o a las del oponente según a_1 y a_2, mientras que la importancia de sus longitudes son reguladas por p_1 y p_2.

Con esta función de peso no se captura el hecho de que hay líneas que pueden ser más amenazantes que otras aunque tengan la misma longitud. Por ejemplo, si $C = 4$ y el oponente cuenta con una línea horizontal de longitud 3 en la primera fila del tablero y con la capacidad de extenderse a izquierda y a derecha, entonces será imposible evitar que gane, puesto que se podrá bloquear dicha línea de un lado pero luego se extenderá por el otro. Por el contrario, una línea horizontal de longitud 3 que no puede extenderse de manera inmediata a izquierda y a derecha resulta mucho menos importante.

Tomando como motivación esa misma situación, modificamos ligeramente la función f de esta manera, para una línea L del jugador para el cual se evalúa el tablero:

$$f(L) = \begin{cases} \alpha_1 a_1 |L|^{p_1} & \text{si } L \text{ puede extenderse de manera inmediata a ambos lados} \\ \beta_1 a_1 |L|^{p_1} & \text{si } L \text{ puede extenderse de manera inmediata a un solo lado} \\ a_1 |L|^{p_1} & \text{en caso contrario} \end{cases}$$

Que L pueda *extenderse de manera inmediata* hacia un lado significa que con sólo una jugada del jugador al cual pertenece L, L puede incrementar su longitud en uno, hacia dicho lado. Por lo tanto, α_1 y β_1 definen por cuánto multiplicamos el peso de la línea si puede extenderse a ambos lados o a uno sólo respectivamente.

Si L pertenece al jugador contrario se toma la misma definición pero con los parámetros a_2, p_2, α_2 y β_2.

Finalmente, el *puntaje* de un tablero T para un jugador J se define como:

$$\sum_{L \in noBloqueadas(T)} f(L)$$

donde $noBloquedas(T)$ es el conjunto de todas las líneas del tablero que podrían llegar a tener, potencialmente, longitud C.

A su vez añadimos dos parámetros más, que son el puntaje que se suma en caso de que haya una línea ganadora (de longitud C) y en caso de que haya una casilla libre donde el oponente pueda insertar una ficha que lo haga ganar.

Sin embargo, decidimos descartar esta manera de encarar el problema, pues el diseño de los parámetros implicó dificultades en la experimentación. En primer lugar, con sólo elegir parámetros manualmente los jugadores resultaron ser muy capaces, a tal punto que no resultaba fácil ganarles, para uno mismo.

Principalmente encontramos que los valores para estos parámetros que eran obtenidos en sucesivas iteraciones del algoritmo genético eran muy variados, y no se lograba converger a ninguna solución en concreto. Además, los jugadores solían empatar la mayor cantidad de veces, lo cual implicaba que la mayoría de ellos eran calificados igualmente, siendo difícil distinguir entre jugadores buenos y malos. Estas dificultades nos llevaron a cancelar cualquier experimentación seria de la cual se puedan extraer resultados para analizar.

Por último, el código utilizado para implementar la evaluación del tablero resultó ser muy complejo y difícil de seguir. Era particularmente difícil determinar si las líneas estaban bloqueadas o no sin dejar de cumplir la restricción de complejidad para evaluar el tablero, que es $O(NM)$.

Decidimos entonces cambiar la forma en que se evalúa el tablero por una más sencilla y predecible, donde distintos parámetros puedan realmente significar diferentes formas de jugar. Los parámetros $w_1 \dots w_k$ son usados como *pesos* asociados a ciertas características del tablero. En este sentido el puntaje de todo tablero es de la forma $\sum_{i=1}^{k} w_i c_i$ donde cada c_i es una característica del tablero expresada en forma numérica. Por lo tanto el objetivo es definir cuáles son dichas características.

En primer lugar, es claro que un tablero donde existe una línea de longitud C del jugador es claramente ventajoso, puesto que implica que el jugador resulta ganador. Esto ya nos da una primera característica a considerar, que es la cantidad de líneas de longitud C del jugador. Definimos entonces:

$$c_1 = \#\{L \ : \ L \text{ es una línea del jugador y } |L| \geq c\}$$

Análogamente, un tablero donde el oponente está a un solo paso de formar una línea de longitud C es perjudicial, puesto que en ese caso el oponente probablemente resulte ganador. Por lo tanto definimos c_2 como la cantidad de casillas (i, j) del tablero tales que:

- (i, j) está desocupada e $(i - 1, j)$ está ocupada o $i = 0$. En otras palabras, (i, j) puede ser ocupada en una sola jugada.
- La ocupación de (i, j) con una ficha del oponente provoca una nueva línea del oponente con longitud mayor o igual a c.

Para definir c_3 y c_4 tenemos en cuenta que, dado que el objetivo es formar una línea de c fichas, puede ser deseable tener líneas lo más largas posibles. Para el jugador puede ser ventajoso contar con una línea de longitud grande, y perjudicial que el oponente cuente con una línea larga. Por lo tanto, definimos:

$$c_3 = \max\{|L| \ : \ L \text{ es una línea del jugador no } \textit{bloqueada}\}$$
$$c_4 = \max\{|L| \ : \ L \text{ es una línea del oponente no } \textit{bloqueada}\}$$

Es decir, tomamos la máxima longitud entre todas las líneas del jugador que no están bloqueadas. Decimos que una línea se encuentra *bloqueda* si no es posible extenderla hacia ninguno de los dos lados (ya sea porque le sigue una línea del oponente o porque colisiona contra el borde del tablero). Análogamente, se define c_4 de la misma manera pero considerando las líneas del oponente.

Hay que notar que la definición de *bloqueada* en este caso es más sencilla que la que se había adoptado para la evaluación anterior, donde teníamos en cuenta si la línea podía llegar a tener longitud c o no. Con esta nueva definición puede haber una línea no bloqueda pero que en realidad nunca podría tener longitud c, y en tal caso sería deseable no considerarla. Sin embargo, decidimos dejar este problema de lado, aceptándolo como un defecto de nuestra función de evaluación, en favor de un código menos complejo.

Por último, también consideramos las cantidades de líneas no bloqueadas con las cuales cuenta el jugador y el oponente. Tener varias líneas puede significar una ventaja puesto que hay en cierto sentido más formas de llegar a una línea de c fichas. Entonces definimos:

$$c_5 = \#\{L \ : \ L \text{ es una línea del jugador no } \textit{bloqueada}\}$$
$$c_6 = \#\{L \ : \ L \text{ es una línea del oponente no } \textit{bloqueada}\}$$

Para mayor claridad adoptaremos la siguiente nomenclatura: sabiendo que cada w_i es el peso asociado a la característica c_i, llamaremos α_1 y α_2 a w_1 y w_2, β_1 y β_2 a w_3 y w_4, y γ_1 y γ_2 a w_5 y w_6.

Como conclusión los jugadores, las configuraciones de parámetros o las soluciones quedan definidas como las tuplas $\{(\alpha_1, \alpha_2, \beta_1, \beta_2, \gamma_1, \gamma_2) \in \mathbb{R}^6\}$

2.2 Tablero

Nuestra implementación del tablero consiste en mantener cuatro listas para todos los tipos de línea: horizontales, verticales y diagonales en ambas direcciones. Además, similarmente a una clase de Disjoint Set con path compression, cada casilla ocupada del tablero tiene un representante de linea para cada tipo desde donde se puede obtener rápidamente información como la longitud o si la línea puede extenderse hacia ambos lados. De esta forma, cuando se introduce una ficha nueva hace falta actualizar todas las líneas de forma tal que ésta se agregue a las potencialmente cuatro líneas afectadas (si no existen se crean). Las líneas viven en listas enlazadas, por lo que insertar y eliminar líneas son operaciones que cuestan tiempo constante, y además se pueden recorrer las líneas fácilmente con iteradores.

Gracias a esto se puede evaluar muy rápidamente un tablero dado, pues sólo hace falta recorrer las cuatro listas de líneas y consultar la información necesaria, que es la cantidad de líneas de longitud c del jugador, y la cantidad de líneas bloqueadas de ambos jugadores, e ir registrando las longitudes máximas. Esto se puede hacer en una única pasada por todas las líneas, dando una cota de complejidad de $O(NM)$.

Para contar la cantidad de casillas tales que si el oponente pone una ficha allí (suponiendo que puede hacerlo en una jugada) consigue una línea de longitud c, se realizan los siguientes pasos:

1. Recorrer las próximas casillas a ocuparse, que son a lo sumo N.

2. Por cada una de ellas ver si hay líneas en las direcciones correspondientes a 0 casillas de distancia, y consultar sus longitudes para ver si al poner una ficha en la casilla actual se forma una línea de c fichas. En nuestra implementación encontrar una línea a partir de una de sus casillas tiene un costo lineal en su longitud en el peor caso. La longitud de las líneas está acotada por c y c está acotado por M, por lo tanto en el peor caso se tiene un costo lineal en M por cada casilla visitada.

Como conclusión, dicha tarea no cuesta más que $O(NM)$ y por lo tanto la función de evaluación tiene complejidad $O(NM)$

2.3 Grid Search

Una vez que tenemos un programa que puede evaluar un tablero basándose en una serie de parámetros, nos interesa hallar valores para estos parámetros que optimicen el nivel de juego del programa.

Para encontrar parámetros que resulten en un buen jugador, primero hay que definir cuándo un jugador es bueno. Existen muchas formas de hacer esto, pero esencialmente todas se reducen a hacer que el jugador juegue contra alguien y ver cómo le va.

Lo que hicimos entonces fue definir un conjunto de jugadores variados, algunos de los cuales tienen estilos de juego bien definidos: algunos son muy agresivos y no les importa lo que hace el rival; otros son defensivos y buscan evitar la derrota, pero no se preocupan por ganar ellos mismos; también hay un jugador que en principio tiene parámetros sensatos, y otro que por el contrario juega "al revés": le da peso positivo a los atributos del rival y negativo a los propios, por lo que debería ser un jugador muy malo. También hay un jugador que fue sacado del algoritmo genético (explicado más adelante)[1].

En total tenemos un grupo de 9 rivales, contra los que un jugador deberá jugar para medir qué tan bueno es. Específicamente, definimos tres tableros de diferentes dimensiones y distinto valor de C. Cada jugador juega dos partidos (uno en el que empieza él, y otro en el que empieza el rival) contra cada rival en cada tablero, lo que da un total de 54 partidos. Cada victoria suma 1 punto por ganar, 0 puntos por empatar y -1 punto por perder. Por lo tanto los puntajes posibles para un jugador son todos los enteros entre -54 y 54 inclusive.

Los tableros que definimos fueron los siguientes:

- 4 en línea en un tablero de 7 columnas y 6 filas
- 5 en línea en un tablero de 9 columnas y 8 filas
- 7 en línea en un tablero de 14 columnas y 14 filas

[1] La lista completa de rivales está definida en el archivo resultados_gridsearch/rivales.txt

La razón por la elegimos que se jugara en estos tableros (y no por ejemplo en tableros al azar) fue que la performance de los jugadores puede variar mucho cuando se varían las dimensiones. Por lo tanto lo más justo es hacer que todos jueguen en exactamente las misma condiciones.

Las dimensiones elegidas nos parecieron razonables pero son arbitrarias, y el sistema de puntuación se basa en ellas. Podría existir un jugador que con este sistema se considera malo, pero que en tableros de otros tamaños obtendría mejores resultados. Cada vez que digamos que un jugador es bueno, o que un jugador es mejor que otro, siempre será en el contexto de este sistema.

Una objeción que podría hacerse a este sistema en el que cada jugador debe enfrentar a varios para ser calificado es que podría existir cierta rendundancia: si un jugador le gana al más fuerte de los rivales entonces seguro le ganará a todos los demás; tal vez sería mejor entonces tener uno o dos rivales solamente. Ante esto, decidimos hacer un torneo todos contra todos entre los rivales (en los mismos tableros). Si realmente existe un rival que es el mejor, entonces deberíamos ver uno que gana todas las partidas, o al menos que no pierde ninguna.

Cada jugador del conjunto de los rivales jugó 48 partidos, y resultó que no hubo ninguno que ganara todos. Lo que es más, todos perdieron al menos 10 partidos[2]. Esto nos indica que no existe un único mejor rival sino que los rivales son simplemente distintos entre sí. Tener una variedad de rivales puede aportar mucho al puntaje de un jugador: en caso de encontrar uno que obtenga un puntaje alto con nuestro sistema, podremos decir que es un jugador *completo*, capaz de ganar a jugadores con estilos de juego variados.

Una vez que tenemos una forma bien definida de puntuar a un jugador, vamos a buscar valores de los parámetros que resulten en un buen jugador. El primer método que vamos a usar para buscar estos valores es grid search.

La idea de grid search es definir un conjunto de valores que puede tomar cada parámetro, y simplemente probar todas las combinaciones de esos valores. Cada combinación se traduce en un jugador distinto, al cual vamos a calificar con nuestro sistema, y al finalizar vamos a ver qué jugadores tuvieron los mejores puntajes.

Una búsqueda exhaustiva como esta puede llevar mucho tiempo, por lo que la cantidad de posibles valores para cada parámetro no puede ser demasiado grande. Elegimos entonces 9 valores para cada uno: 4 positivos de variadas magnitudes , el 0, y 4 negativos, los opuestos de los positivos. De esta manera de define un total de $9^6 \approx 530.000$ jugadores. Los valores exactos que tomó cada parámetro se ven en la siguiente tabla.

Posibles valores de cada parámetro

	1	2	3	4	5	6	7	8	9
lineas_c_player	-1000	-500	-100	-10	0	10	100	500	1000
lineas_c_oponente	-1000	-500	-100	-10	0	10	100	500	1000
maxima_longitud_player	-150	-80	-10	-2	0	2	10	80	150
maxima_longitud_oponente	-150	-80	-10	-2	0	2	10	80	150
cant_lineas_player	-150	-80	-10	-2	0	2	10	80	150
cant_lineas_oponente	-150	-80	-10	-2	0	2	10	80	150

Figure 1

Si bien a priori tenemos algunas ideas de cuáles pueden ser algunos buenos valores para los parámetros (por ejemplo que dar un peso negativo a la cantidad de líneas de tamaño C que tiene el jugador probablemente sea una mala idea), quisimos dejar la libertad de elegir valores positivos y negativos de distintas magnitudes para cada parámetro.

Elegimos dar magnitudes generalmente más grandes para los parámetros *lineas_c* porque imaginamos que son los más "urgentes", es decir los que más deberían preocupar a un jugador (porque cuando la cantidad de líneas de tamaño C en un tablero no sea 0, quiere decir que la partida terminó) y que por lo tanto seguramente serán los de mayor peso. Sin embargo es perfectamente posible que nos equivoquemos, y en ese

[2]Resultados más detallados se encuentran en resultados_gridsearch/resultados-tct-rivales.txt

caso nos encontraríamos que el mejor jugador le da un peso chico (por ejemplo 10, 0 o -10) a esos parámetros, y que el parámetro *dominante* es otro.

2.4 Algoritmo Genético

Para el algoritmo genético iniciaremos con una población de n jugadores aleatorios, y calcularemos su *fitness* con un método de evaluación determinado. Luego, por cada iteración y mientras no se cumpla un criterio de terminación, realizaremos los siguientes pasos:

- Seleccionar $n/2$ jugadores mediante algún método de selección. Dependiendo del método un jugador podría ser seleccionado más de una vez.
- Cruzar de a pares a los jugadores seleccionados con una función de *crossover*, obteniendo n jugadores nuevos (dos por cada par).
- Mutar los genes de los nuevos jugadores.
- Definir a los nuevos jugadores como la nueva generación y evaluarlos (es decir, calcular sus fitness).

A continuación presentamos el pseudocódigo del algoritmo:

Algorithm 1 *Algoritmo Genético*

```
1:  poblacion ← PoblacionAleatoria()
2:  evaluarJugadores(poblacion)
3:  while ¬criterioDeTerminación() do
4:      nueva ← PoblacionVacía()
5:      for i = 0 to poblacion.tamaño/2 do
6:          padre1 ← seleccionar(poblacion)
7:          padre2 ← seleccionar(poblacion)
8:          hijo1, hijo2 ← cruzar(padre1, padre2)
9:          mutar(hijo1), mutar(hijo2)
10:         nueva.añadir(hijo1, hijo2)
11:     evaluarJugadores(nueva)
12:     poblacion ← nueva
13: return mejorJugador(poblacion)
```

2.4.1 Funciones de *fitness*

Para medir qué tan adaptados a su ambiente están los individuos de una generación, implementamos dos funciones de *fitness*:

1. Torneo todos contra todos
2. Jugar contra mejores jugadores de la generación anterior.

La primera consiste en recorrer a toda la población, y por cada jugador hacerlo competir con el resto. Por cada enfrentamiento el jugador suma una cierta cantidad de puntos, y la cantidad total de puntos que haya sumado será exactamente su fitness.

Un enfrentamiento entre dos jugadores consiste en hacerlos jugar en tres tableros diferentes, generados aleatoriamente y con c aleatorio. A su vez, por cada tablero se juegan dos partidos, que se diferencian por qué jugador empieza jugando, ya que podría suceder que un jugador le gane a otro cuando empieza primero pero no cuando empieza segundo. El resultado de cada partido jugado define la cantidad de puntos que se suman a los jugadores:

- Si hay un ganador, el ganador suma dos puntos y el perdedor cero.

- Si hay empate, ambos suman un punto.

Este método de evaluación es bastante exhaustivo para evaluar a los jugadores, puesto que cada uno debe enfrentar a todos. No cuenta con la desventaja de los torneos por eliminación donde podría ocurrir que un jugador relativamente bueno sea eliminado en primera ronda por ejemplo, porque justo se enfrentó a uno de los pocos que eran mejores que él. Es decir, en los torneos por eliminación el resultado puede variar mucho de acuerdo a cómo se organizaron los partidos en la primera ronda, y más aún, resulta difícil definir un fitness de manera clara para cada jugador (la idea inmediata es que el fitness sea qué tan lejos llegó, pero esto implica que habrá muchos jugadores que tengan el mismo valor de fitness). Por estas razones optamos por el torneo todos contra todos.

La decisión de cuántos puntos sumar por perder, empatar y ganar puede dar lugar a diversas preferencias a la hora de evaluar a los jugadores. Por ejemplo, si se suman muchos puntos por empatar podemos obtener jugadores que prefieran bloquear al enemigo, mientras que si puntuamos a quienes ganen partidos mucho más que a quienes empaten habrá una mayor preferencia por jugadores agresivos que traten de extender sus líneas. En nuestro caso decidimos sumar dos puntos por ganar, uno por empatar y cero por perder, porque resulta una decisión equilibrada, sin generar grandes diferencias entre jugadores que tienden a empatar y que tienden a ganar.

Es muy difícil, o imposible, afirmar que jugadores que bloqueen más serán mejores que aquellos más agresivos o viceversa, de manera absoluta. Esto se debe a que no existe una forma de medir a un jugador independientemente de todos, si no que únicamente podemos limitarnos a afirmar que un jugador es bueno o malo respecto a otro u otros. En el algoritmo genético los fitness miden justamente qué tan buenos son los jugadores para esa población o ambiente. Un jugador agresivo por ejemplo puede obtener un fitness muy alto en una población particular, resultando ser uno de los mejores, pero quizás en otra población dicha estrategia le puede jugar en contra y no terminar ganando tantos puntos.

La clara desventaja del método de evaluación por torneo todos contra todos es que implica un mayor costo de procesamiento. La complejidad de evaluar a los n jugadores es cuadrática en función de n, sin contar el costo de cada enfrentamiento, que dependerá de N y M.

El segundo método de evaluación propuesto consiste en hacer que cada jugador tenga un enfrentamiento con cada uno de los $n/2$ mejores jugadores de la generación anterior, es decir, los $n/2$ de mayor fitness. Cada enfrentamiento se define exactamente igual que en el torneo todos contra todos: 6 partidos en 3 tableros, y los puntajes (que serán los valores de los fitness) se computan de la misma manera.

Con este método se plantea una variante interesante donde el fitness de un jugador ya no necesariamente indica qué tan bueno es en su propia generación, pues no se enfrenta a los jugadores que la componen, sino qué tan superior es a los de la generación anterior. De esta manera cuando miremos el jugador de fitness más alto de una generación en particular sabremos que, en cierto sentido, ese ha sido el que más ha evolucionado respecto de la generación anterior.

Este método puede ser criticado por el hecho de que el jugador de fitness más alto podría no ser el mejor de su generación. *A priori* nada impide que exista otro jugador que no sea tan bueno contra los $n/2$ anteriores pero que sí sea bueno en su población. Por otro lado, podría decirse que con este método se pone más esfuerzo en generar jugadores cada vez mejores a los anteriores, y en definitiva esto es lo que se espera con los algoritmos genéticos, que la soluciones vayan evolucionando. Pero nuevamente, como casi todas las afirmaciones que se hacen al enunciar ventajas y desventajas de los distintos aspectos de un algoritmo genético, esto no es algo que vaya a pasar en el 100% de los casos: quizás en una generación nueva ningún jugador supo ganarle a los 5 mejores anteriores, por ejemplo, y podría decirse que las nuevas soluciones no superaron a las de la generación anterior y que incluso empeoraron.

Para este método puede ser muy influyente contra cuántos jugadores de la generación anterior juega cada uno de la actual. Si cada uno juega contra unos pocos anteriores, se corre el riesgo de que la variedad de valores de fitness generados sea muy escasa y/o que sean muy parecidos, lo cual implica que estos valores no sean muy representativos de la calidad de los jugadores y por lo tanto se hagan selecciones que pueden ser inconvenientes. Por otro lado, a medida que se aumenta contra cuántos jugadores anteriores se enfrenta cada uno podría suceder que muchos de esos hayan obtenido valores de fitness muy bajos y por tanto estaríamos midiendo más bien qué tanto evolucionaron los de la generación actual respecto a un conjunto malo de jugadores de la anterior, lo cual no era el objetivo original del método. En nuestro caso elegimos que cada uno enfrente a los $n/2$ mejores anteriores esperando que este valor no favorezca ninguna de las dos situaciones

en particular.

2.4.2 Métodos de selección

Para determinar qué individuos de una generación dada pasan sus genes a la próxima generación utilizamos dos criterios:

1. Selección estricta.
2. Selección aleatoria ponderada por fitness.

La selección estricta consiste en seleccionar, estrictamente, a los $n/2$ jugadores con los fitness más altos. Es decir, podemos pensar que los jugadores se almacenan en una cola de prioridad por fitness y cada selección consiste en una operación de *pop*. Esto implica que un jugador nunca es seleccionado más de una vez en cada generación.

Por un lado, con este método no se le da oportunidad a las soluciones con menor fitness de pasar sus genes a nuevas soluciones de una generación siguiente. Esto puede ser visto como una desventaja, puesto que podrían existir jugadores con fitness bajos pero cuyos genes se podrían utilizar para crear soluciones que, quizás, resultan estar más adaptadas para la generación siguiente, efectivamente logrando transitar por conjuntos de soluciones diferentes de los que se venían considerando. En síntesis, excluir estas soluciones puede causar que el algoritmo se estanque en máximos locales.

Sin embargo, también hay que tener en cuenta que con este método las soluciones se seleccionan una única vez, y por lo tanto habrá $n/2$ soluciones distintas que estarán pasando sus genes a la nueva generación. Esta cantidad puede ser suficiente para asegurar que haya variedad en las generaciones. Además, la posibilidad de incluir una solución atípica, distinta al resto, en una nueva generación, ya viene dada por las operaciones de mutación que se realizan sobre los genes.

La selección ponderada consiste en seleccionar un jugador aleatorio, pero de tal manera que la probabilidad de que el jugador J sea elegido es:

$$p(J) = \frac{fitness(J)}{\sum_{J' \in poblacion} fitness(J')}$$

Notemos que es necesario que los fitness sean no negativos para que dicha probabilidad no carezca de sentido.

En este caso, a diferencia de la selección estricta, sí es posible que un jugador sea elegido más de una vez, lo cual aporta a la tendencia de transitar por máximos locales, porque podría pasar que siempre estemos pasando una reducida variedad de genes. Pero por otro lado las soluciones que no tienen buen fitness tienen posibilidades de pasar sus genes a la siguiente generación, y esto combate la tendencia a estancarse en máximos locales.

Para implementar el método de selección ponderada realizamos el siguiente preprocesamiento, donde f es la función de fitness (precomputada) y $J_0 \dots J_{n-1}$ son los n jugadores:

Algorithm 2 *Preprocesamiento para selección ponderada*

```
F ← Σ_{i=0}^{n-1} f(J_i)
S ← ArregloInt(n)
S[0] ← f(J_0)
for i = 1 to n − 1 do
    S[i] ← S[i − 1] + f(J_i)
```

Y luego la selección consiste en el siguiente algoritmo:

Algorithm 3 *Selección ponderada de un jugador*

```
r ← valor aleatorio entre 0 y F − 1 con distribución uniforme
for i = 0 to n − 1 do
    if r < S[i] then
        return J_i
```

En efecto lo que se hace es particionar el intervalo $[0, F-1]$ en subintervalos, uno por cada jugador, cuya longitud es el fitness del mismo. Entonces se genera un número al azar entre 0 y $F-1$, y el subintervalo al cual pertenezca determinará el jugador a elegir. A mayor fitness, mayor es el subintervalo del jugador y por lo tanto tiene mayor probabilidad de ser elegido.

2.4.3 Operación de *crossover*

Contamos con dos operaciones de *crossover*, es decir, con dos formas de cruzar dos jugadores y a partir de ellos crear dos jugadores nuevos.

La primera de ellas se trata de un crossover fijo, porque está predefinido cómo se intercambian los genes. Si $J_1 = (\alpha_1, \alpha_2, \beta_1, \beta_2, \gamma_1, \gamma_2)$ es un jugador y $J_2 = (\alpha'_1, \alpha'_2, \beta'_1, \beta'_2, \gamma'_1, \gamma'_2)$ es otro, entonces al cruzar a J_1 y J_2 se producen dos nuevos jugadores H_1 y H_2 de la forma:

$$H_1 = (\alpha_1, \alpha'_2, \beta_1, \beta'_2, \gamma_1, \gamma'_2), \quad H_2 = (\alpha'_1, \alpha_2, \beta'_1, \beta_2, \gamma'_1, \gamma_2)$$

La idea es que los parámetros de un jugador pueden clasificarse en dos categrías: aquellos asociados a características que miden datos del jugador (α_1, β_1 y γ_1), y los que están ligados a características que miden datos del oponente (α_2, β_2 y γ_2). Si tenemos dos jugadores relativamente buenos, entonces hacemos que en los hijos, los genes de una categoría hayan sido heredados de un padre, y los de la otra categoría de otro.

Esta elección tiene cierta coherencia pues se busca que un hijo sepa medir sus datos tan bien como uno de los padres, y medir los del oponente tan bien como el otro padre, efectivamente combinando ambas características para formar un jugador bueno en los dos aspectos. Por ejemplo, podemos contar con un jugador agresivo cuyos parámetros α_1, β_1 y γ_1 lo llevaron a ganar rápidamente en su ambiente y otro cuyos parámetros α_2, β_2 y γ_2 le permitieron evitar muchas derrotas, aunque no haya ganado muchos partidos. Por lo tanto idealmente un jugador que herede estos genes sabrá jugar agresivamente pero evitando perder en lo posible, por lo cual se espera que sea un jugador bueno para su ambiente.

Por supuesto que estas decisiones son heurísticas y no tienen por qué funcionar, más aún sabiendo que el desempeño de los jugadores depende mucho de la población en la que se encuentran. Nada impide que un jugador formado de esta manera resulte poco performante para su generación aún cuando sus padres fueron buenos jugadores en la anterior.

Nuestra segunda operación de crossover es aleatoria y fue ideada sin pensar en el significado de los genes. Simplemente se genera un punto de corte aleatorio en el genoma que define cuáles son los genes que serán intercambiados. Más en concreto, si $w_1 \dots w_6$ son los genes de J_1 y $w'_1 \dots w'_6$ los de J_2, se genera un i aleatorio generado uniformemente entre 1 y 6, y se procede a pasar los genes $w_1 \dots w_{i-1}$ y $w'_i \dots w'_6$ a H_1, y los genes $w'_1 \dots w'_{i-1}$ y $w_i \dots w_6$ a H_2.

Con este segundo método evitamos tener una forma de pasar genes de una generación a otra basada en nuestra intuición que podría funcionar bien para ciertas poblaciones pero no para otras. A pesar de que este método también puede resultar en jugadores que terminan siendo no muy eficientes, resulta más difícil que suceda continuamente debido al hecho de que los genes que se heredan de un padre y de otro van cambiando continuamente, de manera aleatoria, sin seguir ningún patrón en particular. Esta capacidad de evitar casos atípicos y difíciles de replicar en otras poblaciones puede ser visto como una ventaja.

Se podría dar un paso más, y que el punto de corte se haga sobre la representación binaria del genoma, considerando estos como simples cadenas de los símbolos 0 y 1 y por ende abstrayéndonos del hecho de que existen parámetros con valores determinados. Decidimos por una cuestión de simplicidad mantenernos con el método tal como fue explicado previamente.

2.4.4 Operación de mutación

La operación de mutación se aplica a cada uno de los genes del jugador, y consiste en, con probabilidad 0.1, realizar las siguientes operaciones:

- Generar un valor m aleatorio con distribución normal centrada en 0 y un desvío estándar específico para el gen o parámetro en cuestión.
- Sumar m al valor del gen.
- Con probabilidad 0.2, invertir el signo del gen.

Creemos que 0.1 es un valor razonable para mantener una frecuencia de mutaciones baja, que impida que las soluciones converjan y los jugadores siempre estén cambiando sin seguir ninguna tendencia en particular, pero lo suficientemente alto para evitar estancarse en un grupo de soluciones particular y poder transitar hacia otros conjuntos con características diferentes. Es decir, con este valor se lograría que las mutaciones cumplan su rol, el de evitar máximos locales sin perjudicar la evolución de las soluciones.

Para los parámetros diseñados, aparte de los valores en sí, resulta de particular interés el signo de los mismos, puesto que determinan si las características asociadas son beneficiosas o perjudiciales para el jugador. Podría suceder que en una población particular los mejores jugadores, por ejemplo, pesen positivamente la máxima longitud de las lineas del enemigo. Esto resultaría contrario a nuestra intuición, pues nos resulta obvio que dicha característica es negativa, sin embargo estos jugadores resultan ser los mejores en su ambiente si adoptan dicha estrategia. Para poder evolucionar a jugadores potencialmente mejores, sería interesante que puedan haber jugadores nuevos que, siguiendo el ejemplo, pesen negativamente la máxima longitud del enemigo, más allá de si terminan obteniendo mayor fitness o no.

Para dar lugar a esta posibilidad es que decidimos que haya mutaciones aún más infrecuentes que cambien el signo de los genes, ya que un cambio de signo resulta difícil si sólo se suma un valor con distribución normal y no se tiene un desvío estándar muy elevado.

3 Resultados y discusión

3.1 Grid search

3.1.1 Análisis de los jugadores

Después de considerar los 9^6 jugadores definidos anteriormente y de asignarles un puntaje a cada uno, encontramos a los que obtuvieron el máximo puntaje.

Recordemos que el máximo puntaje posible era de 54 puntos. Lo máximo obtenido fueron 48 puntos, y hubo 9 jugadores que obtuvieron ese puntaje, presentados a continuación:

Mejores jugadores del grid search

	lineas_c_player	lineas_c_oponente	maxima_longitud_player	maxima_longitud_oponente	cant_lineas_player	cant_lineas_oponente
1	500	-100	10	-2	2	10
2	1000	-500	10	-2	2	10
3	1000	-500	80	-10	2	80
4	1000	-500	80	-10	10	80
5	1000	-500	80	-2	2	80
6	1000	-500	80	-2	10	80
7	1000	-500	150	-10	2	80
8	1000	-500	150	-2	2	80
9	1000	-100	10	-2	2	10

Figure 2

Todos ellos tienen caracterísiticas en común:

- Pesos positivos en *lineas_c_player, maxima_longitud_player, cant_lineas_player* y *cant_lineas_oponente*.
- Pesos negativos en *lineas_c_oponente* y *maxima_longitud_oponente*.
- El parámetro de mayor magnitud siempre fue *lineas_c_player*.
- El segundo parámetro de mayor magnitud siempre fue *lineas_c_oponente*. Siempre tuvo valor absoluto estrictamente menor que *lineas_c_player*.
- *maxima_longitud_player* siempre fue mayor que *cant_lineas_player*.
- *maxima_longitud_player* siempre fue mayor en valor absoluto que *maxima_longitud_oponente*
- *cant_lineas_oponente* siempre fue mayor que *cant_lineas_player*
- Ningún valor fue 0.

Los jugadores convergieron en varios valores que eran esperables. El hecho de que el parámetro de mayor valor sea el de la cantidad de líneas propias de tamaño C puede traducirse en que si el jugador tiene la posibilidad de ganar inmediatamente, lo va a hacer.

Lo segundo más importante, en caso de no poder ganar, es impedir que el rival gane. Eso es lo que nos dice el hecho de que el parámetro de las líneas de longitud C del rival sea grande, pero no tan grande como el de las líneas propias. Estos resultados son muy sensatos y eran esperables.

Lo siguiente que vemos no era tan obvio: el peso de línea más larga es mayor que el de la cantidad de líneas. Esto nos dice que el jugador prefiere alargar una línea en lugar de ponerse a armar líneas nuevas.

Por lo que vemos después, es preferible para el jugador extender las líneas propias que impedir que el rival extienda las suyas.

Por último tenemos que los jugadores parecen considerar positivo el hecho de que el rival tenga muchas líneas. Esto inicialmente puede resultar ilógico, pero si lo pensamos nos damos cuenta de que esto, sumado a

que la máxima longitud del oponente tiene peso negativo, significa que el jugador prefiere que el rival tenga muchas líneas cortas en lugar de pocas líneas largas, que serían más amenazantes.

El último punto no es nada menor. Ningún jugador que usó el valor 0 en algún parámetro obtuvo el puntaje máximo (48). Esto nos dice que todos los parámetros aportan algo, y que si se quitara alguno entonces no se podría definir un jugador de igual calidad.

Podemos mirar con más detalle el impacto que tiene cada parámetro sobre el jugador. El siguiente mapa de calor muestra para cada parámetro y para cada valor que pudo tomar ese parámetro, el máximo puntaje obtenido por un jugador que usó ese valor en ese parámetro.

Por ejemplo, la casilla de arriba a la izquierda con el número 17 significa que de todos los jugadores (serían 9^5 jugadores) que usaron el valor -1000 en el parámetro *lineas_c_player*, el mayor puntaje obtenido fue 17. Considerando que muchos de los rivales no son muy buenos jugadores, este valor nos dice que probablemente sea una muy mala idea pesar muy negativamente nuestras líneas de tamaño C.

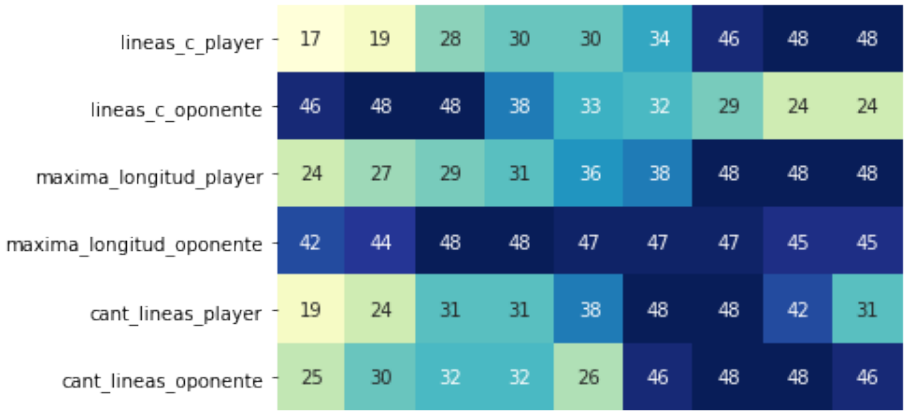

Figure 3: El valor de cada posición (i,j) corresponde al puntaje del mejor jugador que usó el j-ésimo valor del i-ésimo parámetro según fueron definidos en la tabla de la Figura 1.

Se pueden extraer muchas observaciones interesantes con este gráfico. Por ejemplo, la columna del medio representa el valor 0 en todos los parámetros. Por lo tanto los valores de esa columna nos hablan de qué tan importante es cada parámetro, porque esos son los casos en los que un parámetro "no se usó".

Vemos por ejemplo que el parámetro *maxima_longitud_oponente* fue en cierta forma el menos importante. Es decir, si tuviéramos que quitarle un parámetro al jugador, los mejores resultados se obtendrían quitando ese (en efecto el puntaje es apenas un punto menor que el máximo puntaje obtenido). Interesantemente, el valor menos prescidible fue *cant_lineas_oponente*. De hecho esa fila casi parece decir que es mejor tener ese parámetro con cualquier peso (positivo o negativo) a no tenerlo. No tenemos una buena explicación de por qué ocurrió eso.

También se observa que todas las filas tienen más de un valor con los que se obtuvo el puntaje 48. Recordando que los valores de los parámetros que consideramos en el grid search típicamente no eran demasiado cercanos uno del otro (por ejemplo para *lineas_c_player* consideramos los posibles valores 500 y 1000 sin nada en el medio), esto parece indicar que ningún parámetro necesita ser afinado con precisión milimétrica para que el jugador sea bueno, sino que existe bastante flexibilidad.

3.1.2 Resultados contra el jugador random

Vamos a elegir uno de los nueve jugadores: el jugador número 5, por ser el que tiene más características en común con los demás. Vamos a hacerlo jugar contra el jugador random y ver cómo le va.

La versión del tablero que más nos interesa es la clásica de 4 en línea, pero vamos a ir más lejos y hacerlos jugar en los tres tableros que definimos anteriormente, más uno en más grande, para ver si el jugador que consideramos bueno en los primeros tres tableros logra *adaptarse* a uno distinto. Se jugarán 2000 partidos en cada tablero, 1000 en los que empieza nuestro jugador y 1000 en los que empieza el random. En todos los casos la cantidad de fichas es ilimitada.

Juego	Tablero	Empieza	Ganados	Empatados	Perdidos
4 en línea	6 filas, 7 columnas	Jugador	987	0	13
4 en línea	6 filas, 7 columnas	Random	960	0	40
5 en línea	8 filas, 9 columnas	Jugador	991	0	9
5 en línea	8 filas, 9 columnas	Random	987	0	13
7 en línea	14 filas, 14 columnas	Jugador	1000	0	0
7 en línea	14 filas, 14 columnas	Random	999	0	1
9 en línea	17 filas, 18 columnas	Jugador	1000	0	0
9 en línea	17 filas, 18 columnas	Random	1000	0	1

Vemos que nuestro jugador ganó cómodamente en general. A medida que aumenta el tamaño del tablero y la cantidad de fichas para ganar, aumenta también el porcentaje de victorias del jugador, incluso para el tablero en el que el jugador nunca había jugado. Esto se explica por el hecho de que mientras más grande es c, menor es la probabilidad del random de armar una línea ganadora antes que su rival.

De hecho para ganar, al jugador random no le basta "intentar" armar una línea de tamaño c, porque en ese caso el jugador la defendería inmediatamente. Para ganar, necesita tener más de una amenaza al mismo tiempo, de manera que el jugador no pueda defenderlas juntas. Esto sin mencionar que el jugador random también necesita tener la suerte de efectivamente concretar una victoria una vez que tiene una doble amenaza.

También es interesante notar que no hubo empates, lo que nos hace pensar que las partidas generalmente tuvieron la siguiente forma: o bien nuestro jugador ganó rápido y sin problemas (porque el random necesitaría demasiada suerte para defender todas las amenazas), o bien el random tuvo la suerte de crear una amenaza imparable que nuestro jugador no previó, y además después tuvo la suerte de concretar la victoria.

3.2 Algoritmo genético

El principal objetivo de la experimentación con el algoritmo genético fue analizar, para cada variante del mismo, las características de las soluciones obtenidas, cómo evolucionaron los parámetros de las mismas en función del tiempo, y si tendieron a diverger o converger.

3.2.1 Consideraciones generales

Como se mencionó, los enfrentamientos entre dos jugadores consisten en seis partidos, dos por cada tablero, habiendo tres tableros. En concreto, cada tablero se genera aleatoriamente de la siguiente manera:

- N generado uniformemente entre 10 y 14.
- M generado uniformemente entre $3N/4 + 1$ y $3N/2$.
- c generado uniformemente entre $\min(N, M)/3 + 1$ y $2\min(N, M)/3 + 1$.

Esta forma de generarlos nos garantiza que los tableros no estén muy lejos de ser cuadrados, de manera de parecerse al tablero del *4 en línea*, y que los valores de c no sean ni demasiado chicos ni demasiado grandes en relación al tamaño del tablero. A su vez, los tamaños fueron elegidos teniendo en cuenta que con tableros muy grandes se alarga la duración de los partidos y por lo tanto lo tanto la experimentación termina llevando demasiado tiempo.

En todos los casos las poblaciones son de 20 jugadores, y este es un número que surgió al considerar por un lado la necesidad de contar con la suficiente variedad en los valores iniciales de los genes de manera que la cantidad de posibles generaciones a las que se pueda evolucionar sea lo más alta posible, y la necesidad de que las experimentaciones terminen en un tiempo razonable, puesto que con poblaciones grandes se tarda mucho tiempo en computar el fitness de los jugadores.

Cada jugador inicial se genera asignándole valores aleatorios generados uniformemente en los siguientes rangos:

- α_1 y α_2: $[-1000, 1000]$

- Resto de los parámetros: $[-100, 100]$

Ante la dificultad de determinar cuándo se está convergiendo a una cierta solución o si las soluciones obtenidas son buenas (puesto que sólo podemos decir que son buenas respecto a otras soluciones), no se adoptó ningún criterio de terminación en particular, que analice convergencia y/o calidad. Los criterios consistieron únicamente en un límite para el tiempo que se lleve iterando.

3.2.2 Evaluación: torneo todos contra todos

En primer lugar, consideramos el torneo todos contra todos (TCT) como método de evaluación, crossover aleatorio, y los dos métodos de selección explicados: estricta y ponderada.

A grandes rasgos esperábamos los siguientes resultados para los mejores jugadores de las generaciones en cada iteración (es decir aquellos con mayor fitness):

- α_1 positivo y α_2 negativo. Dado que estos parámetros básicamente indican qué tanto le importa a un jugador ganar y qué tanto evita perder, se espera que los jugadores de mayor fitness los tengan con los signos indicados, al menos luego de una cantidad razonable de iteraciones, puesto que en caso contrario sería difícil explicar cómo obtuvieron un fitness alto, excepto que el resto de los jugadores tampoco hayan tenido los signos indicados para ambos parámetros. Sin embargo esto es sumamente improbable que ocurra puesto que en alguna generación seguro que entrará un jugador con al menos uno de estos dos parámetros con el signo razonable, probablemente debido a una mutación, y tendrá muchas victorias y por lo tanto pasará sus genes a la generación siguiente, dando lugar a nuevos jugadores mejor preparados. También se espera que la magnitud de estos valores sea alta, dada la importancia de las características asociadas.

- β_1 positivo y β_2 negativo. Para un jugador sensato la máxima longitud entre todas las líneas debería ser importante, puesto que en cierto sentido mide qué tan cerca se está de ganar. Análogamente, la máxima longitud de las líneas del oponente debería ser un factor negativo puesto que indica qué tan cerca está el oponente de ganar. Las razones por las cuales creemos que esto debería ocurrir después de una cierta cantidad de iteraciones son análogas a las que se dieron para el caso de α_1 y α_2.

- Para γ_1 y γ_2 no es tan claro qué se espera. Tener muchas líneas puede ser considerado como una buena estrategia puesto que supone varios "caminos" para alcanzar la victoria. Sin embargo esto debe hacerse de manera inteligente, como lo haría un humano, en cambio un jugador con un γ_1 muy alto podría posiblemente tender a crear líneas en lugares dispersos sin crear conexiones útiles entre ellas, y de esta manera nunca se acercaría a una victoria. Un razonamiento análogo se puede hacer para γ_2: si este valor es muy negativo quizás el jugador termine bloqueándole líneas al oponente que no suponen una real amenaza.

En la Figura 4 mostramos un gráfico donde se observa, en función del tiempo, los valores de los parámetros del mejor jugador de la generación actual.

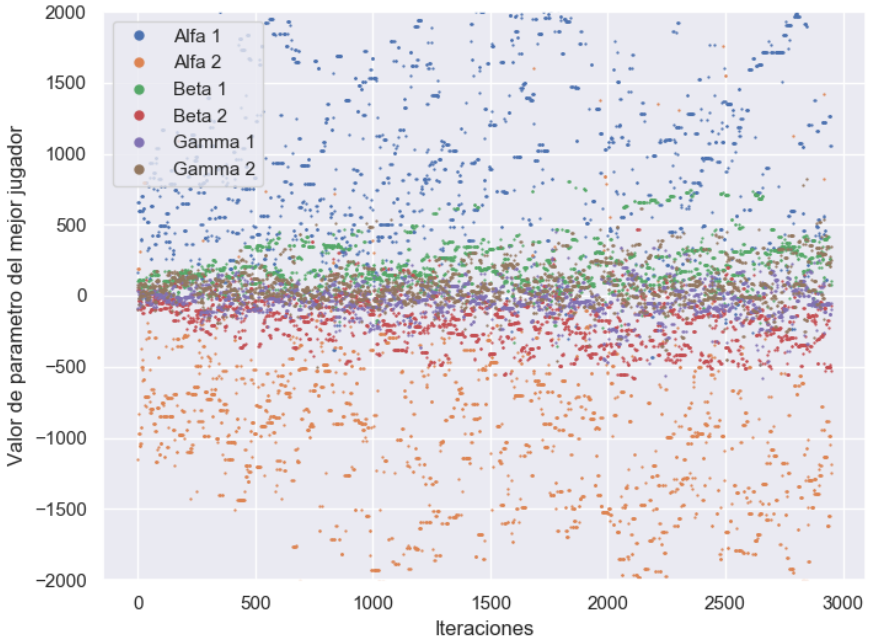

Figure 4: Valores de los parámetros de las soluciones de fitness más alto, por cada generación, obtenidos con evaluación con torneo TCT, selección estricta y crossover random. Se realizaron 2950 iteraciones.

Se puede observar que los signos de α_1 y α_2 fueron los esperados, pues hay una clara tendencia a que α_1 sea positivo y α_2 negativo. Si se observa con atención, hay ciertos casos excepcionales donde la mejor solución tiene a α_2 positivo o α_1 negativo. Esto probablemente se deba a una mutación que cambió el signo del gen en cuestión y eso no impidió que la solución con ese nuevo gen terminara siendo la mejor en su generación. Sin embargo rápidamente estos genes "se pierden", es decir, en algún momento las soluciones con esos genes mutados terminan obteniendo un fitness bajo, pues se imponen las soluciones con los valores más dominantes, y no llegan a ser considerados para la siguiente generación.

Lo mismo se puede decir de β_1 y β_2: los resultados fueron los esperados, aunque también se obtuvieron soluciones atípicas en alguna iteración aislada donde los signos eran contrarios.

Para γ_1 y γ_2 no se sigue ninguna tendencia en particular, y sus valores parecen oscilar aleatoriamente. Esto significa que para las generaciones transitadas el peso asignado a las cantidades de líneas del jugador y del oponente no parecieron marcar una diferencia en cuanto al rendimiento de los jugadores.

Sin embargo, y siendo el fenómeno que salta a primera vista, se observa que no sólo no hay convergencia a ninguna solución en absoluto, si no que además las mejores soluciones van cambiando constante y drásticamente. Esto es un claro defecto de la selección estricta. Habíamos mencionado que, por un lado, con selección estricta se descartan soluciones de bajo fitness a pesar de que podrían dar lugar a futuras soluciones potencialmente mejores, por lo tanto se tendería a transitar por menos soluciones. Por otro lado, se seleccionan $n/2$ jugadores distintos, que conforman la mitad de la población, y por lo tanto los genes pasados a la siguiente generación pueden ser muy variados (incluso muchos de estos podrían provenir de jugadores con fitness muy bajos aunque se encuentren entre los $n/2$ mejores), y esto aportaría a que se generen nuevas soluciones diferentes. Es posible que este segundo factor sea una realidad y que termine siendo mucho más significante que el primero, es decir, que el hecho de que la mitad de la población pase sus genes implique que las mejores soluciones en cada iteración vayan cambiando drásticamente. De todas formas no es más que una posible explicación, y para poder conocer las verdaderas razones se deberían hacer experimentaciones más detalladas mediante las cuales se trate de entender qué sucede realmente.

A partir de este resultado es razonable esperar que con selección ponderada dicho problema al menos se reduzca, dado que con dicho método de selección puede haber jugadores seleccionados más de una vez, y en

definitiva si hay unos pocos jugadores con mucho fitness serán estos los que principalmente determinarán las características de la generación siguiente, aunque sí se le da oportunidad a soluciones de menor fitness de pasar sus genes.

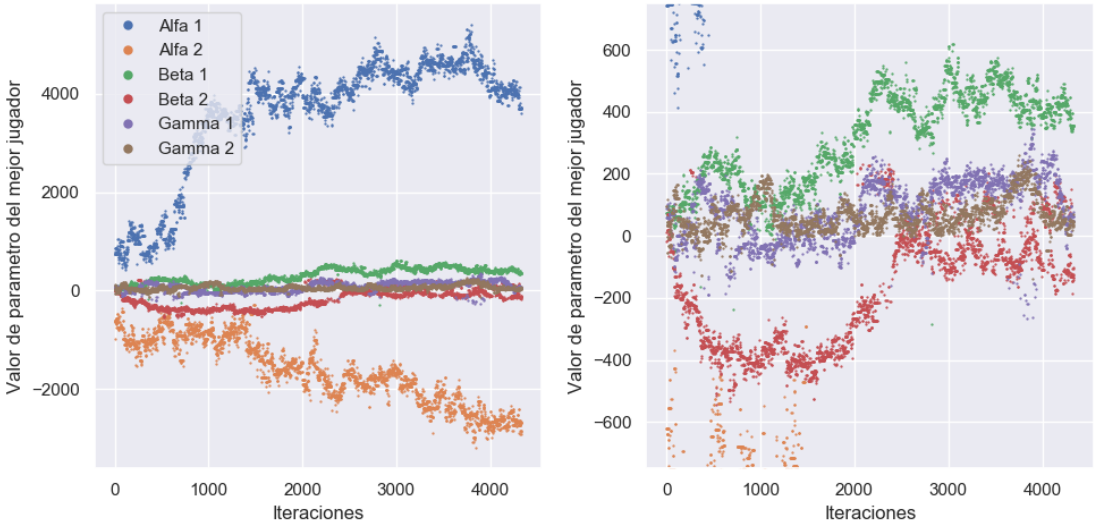

Figure 5: Valores de los parámetros de las soluciones de fitness más alto, por cada generación, obtenidos con evaluación con torneo TCT, selección ponderada y crossover random. A la derecha se muestra el mismo gráfico pero mostrando con más claridad los valores de β_1, β_2, γ_1 y γ_2. Se realizaron 4334 iteraciones.

En la Figura 5 vemos los resultados obtenidos con selección ponderada, y en efecto vemos que el problema prácticamente desapareció: las soluciones obtenidas parecen evolucionar de manera más continua y sin cambiar de manera drástica. Es razonable entonces pensar que con este método de selección se logra una mejor transmisión de los genes de una generación a otra, donde se asegura que en los jugadores de la siguiente generación estén más presentes los genes de los mejores jugadores anteriores, ya sea que provengan de dos jugadores con fitness muy alto comparado con el resto, o de 10 jugadores con fitness parecido. Por el contrario, con selección estricta, si hubieran dos jugadores muy buenos y el resto muy malos, los genes de estos jugadores malos estarían muy presentes en la siguiente generación.

Vemos que los resultados esperados en cuanto a los signos de los parámetros se mantienen, aunque luego de las 2500 iteraciones el valor de β_2 parece acercarse más a 0, y el de β_1 crece más, lo cual nos da a entender que se generaron jugadores más bien ofensivos. Nuevamente los valores de γ_1 y γ_2 parecen oscilar más o menos en torno a 0, cambiando de signo constantemente, lo cual parece indicar que no hay valores particulares para estos parámetros que determinen buenos o malos jugadores, al menos para las generaciones transitadas.

También es interesante el hecho que los jugadores más adaptados dieron más importancia a sus líneas de longitud c que a la posiblidad de que el oponente pueda ganar, ya que el valor de α_1 tiende a rondar por 4000 mientras que el de α_2 se encuentra por -2000. Esta es una estrategia que coincide con la de un jugador humano sensato: una jugada que genera una victoria es preferible a una que evita una victoria del oponente.

Hasta ahora se han mostrado resultados obtenidos con crossover aleatorio, pero resta observar las soluciones obtenidas con la operación de crossover fija que se describió en el desarrollo. El objetivo es ver si esta operación produce cambios respecto a lo que se observó con el crossover aleatorio, sin tener una hipótesis clara respecto a lo que sucederá.

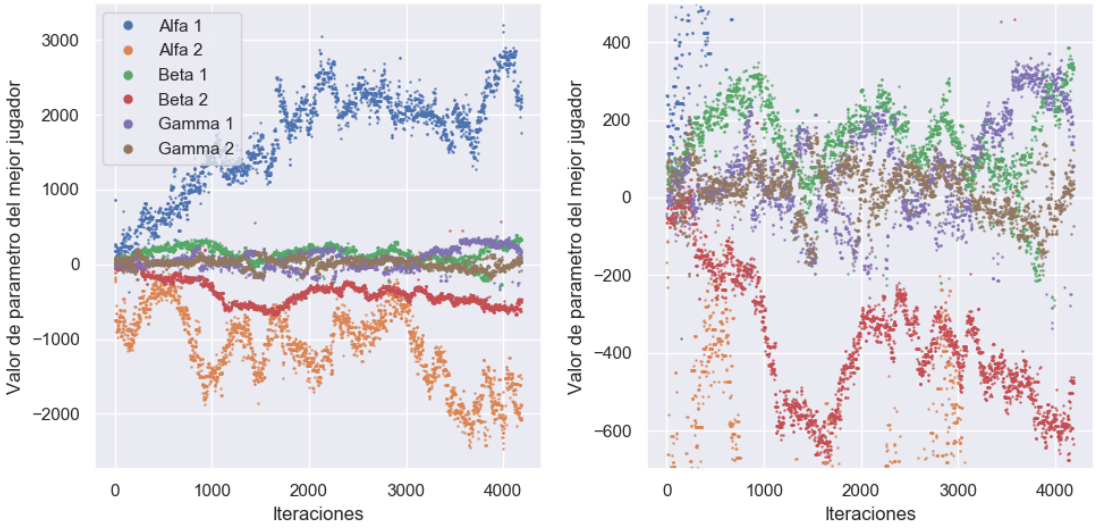

Figure 6: Valores de los parámetros de las soluciones de fitness más alto, por cada generación, obtenidos con evaluación con torneo TCT, selección ponderada y crossover fijo. A la derecha se muestra el mismo gráfico pero mostrando con más claridad los valores de β_1, β_2, γ_1 y γ_2. Se realizaron 4188 iteraciones.

En la Figura 6 vemos las soluciones obtenidas en función del tiempo. No se observa ningún fenómeno particular que llame nuestra atención de inmediato. Las diferencias con la Figura 5 son las siguientes:

- Se generaron soluciones con valores más bajos de β_1, que incluso han llegado a ser negativos. Esto implica que los mejores jugadores resultaron ser más defensivos.
- β_2 aumentó su magnitud bastante más que con TCT, lo cual también confirma que se generaron jugadores más defensivos.

Por otro lado, en general los signos de los parámetros coinciden con los que se obtuvieron en TCT, y α_1 también adquirió en promedio mayor magnitud que α_2, aunque en menor medida. El comportamiento de γ_1 y γ_2 resultó ser el mismo: no parece haber correlación entre las mejores soluciones y sus valores para estos parámetros.

Podemos concluir que el método de evaluación por torneo TCT cumplió en gran medida con lo que se esperaba, sobre todo con respecto a los signos de los parámetros. Además, los jugadores más adaptados a su ambiente resultaron tener magnitudes mayores para α_1, lo cual resulta sensato desde el punto de vista humano. Con crossover aleatorio se generaron jugadores más ofensivos y con crossover fijo, más defensivos, como pudo deducirse de los valores de β_1 y β_2. En ambos casos, los valores de γ_1 y γ_2 no resultaron decisivos para los jugadores puesto que se pudo observar que sus valores oscilaron alrededor de 0.

Con respecto a la convergencia, no se observa que se esté convergiendo a una solución clara, es decir, que los parámetros tiendan a acercarse a un cierto valor. A pesar de que los valores van dando saltos sensatos, de manera que parezca que las soluciones varían con continuidad (a diferencia de lo que se observó con selección estricta), no vemos que las soluciones dejen de variar en un momento. Resulta difícil suponer si en algún momento esto habría sucedido si se hubiera dejado que los procesos realicen muchas más iteraciones, y quizás nunca habría sucedido. Las mutaciones siempre introducen nuevos genes que pueden llegar a cambiar repentinamente hacia qué valores se tienden, y teniendo en cuenta que las mejores soluciones son simplemente las más adaptadas a su ambiente, y la complejidad del problema que se está resolviendo, donde pueden haber potencialmente muchas soluciones distintas igualmente "buenas" a las cuales se puede llegar si hay suficientes mutaciones, resulta dudoso que en algún momento se pueda converger a una solución específica.

3.2.3 Evaluación: jugar contra jugadores de la generación anterior

Ahora consideramos el segundo método de evaluación propuesto, que consiste en hacer que cada jugador juegue contra los 10 mejores jugadores de la generación anterior. No está claro qué se espera que suceda con los valores de los parámetros de las mejores soluciones en función del tiempo, más allá de que sean sensatos para nosotros, según se describió para el caso de TCT (es decir que los mejores jugadores contengan parámetros que en cierta medida se correspondan con las estrategias comunes de un ser humano).

Figure 7: Valores de los parámetros de las soluciones de fitness más alto, por cada generación, obtenidos con el segundo método de evaluación (jugar contra los $n/2$ mejores anteriores), selección estricta y crossover random. Se realizaron 2938 iteraciones.

En la Figura 7 vemos los resultados obtenidos con selección estricta y crossover random. Una vez más se confirma la inferioridad del método de selección estricta, puesto que las soluciones cambian constantemente y por lo tanto el algoritmo no recorre soluciones bien definidas. El hecho de que ocurra lo mismo con otro método de evaluación nos da un poco más de seguridad para creer que la selección estricta probablemente funcione de manera indeseada para la mayor parte de los métodos de evaluación, excepto quizás que se reduzcan las mutaciones a una frecuencia casi despreciable.

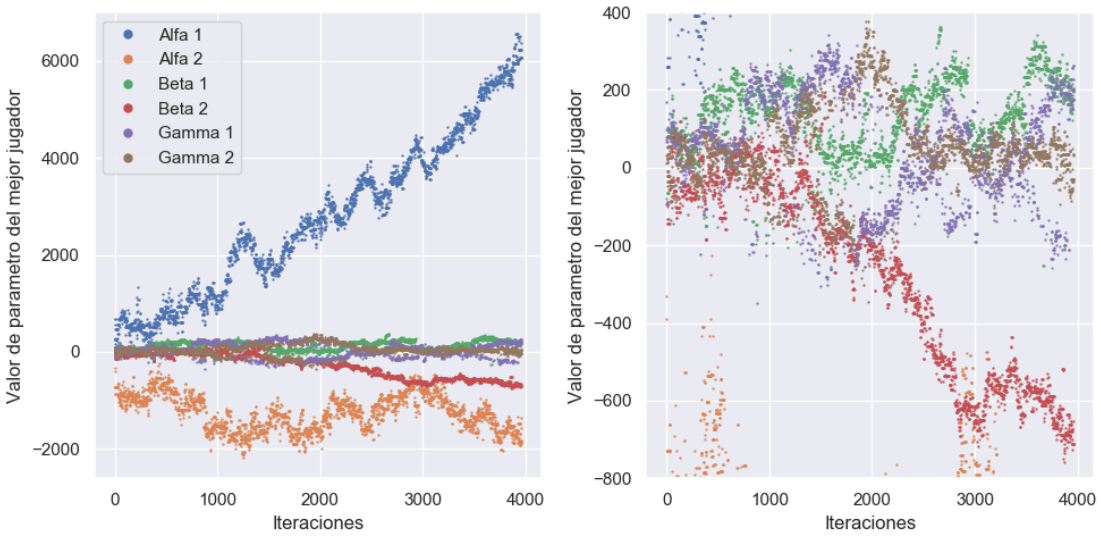

Figure 8: Valores de los parámetros de las soluciones de fitness más alto, por cada generación, obtenidos con el segundo método de evaluación (jugar contra los $n/2$ mejores anteriores), selección ponderada y crossover random. A la derecha se muestra el mismo gráfico pero mostrando con más claridad los valores de β_1, β_2, γ_1 y γ_2. Se realizaron 3962 iteraciones.

En la Figura 8 se observan los resultados al cambiar a la selección ponderada. Al igual que en el caso de TCT, la selección ponderada sí produce que se recorran soluciones mejor definidas a lo largo del tiempo, o en otras palabras que no haya cambios drásticos y de manera constante que impidan identificar con claridad qué soluciones nos está brindando el algoritmo.

Al igual que en TCT, los valores de α_1 y α_2 cumplen con lo previsto, aunque aquí se da la particularidad de que α_1 no sólo tiende a ser más grande que α_2 en magnitud (como en TCT) si no que además parecería crecer linealmente (aunque con oscilaciones) en función de la cantidad de iteraciones realizadas. Dado que α_1 mide qué tanta importancia tiene ganar el juego, podría suceder que este crecimiento en su valor a medida que pasa el tiempo nunca se corte, y de hecho, como seres humanos sensatos sabemos que el mejor valor para α_1 sería $+\infty$.

Otro fenómeno de interés es que el valor de β_2, a parte de ser negativo como esperábamos y como ocurre en TCT, decrece cada vez más (es decir aumenta en magnitud), y de hecho lo mismo puede observarse, aunque con cierta dificultad, en la Figura 7, con la selección estricta. Por lo cual parecería ser un comportamiento asociado al método de evaluación en sí, que consiste en que los jugadores que mayormente superan a la generación anterior son aquellos que le dan cada vez más importancia a bloquear las líneas de máxima longitud del enemigo. De todas formas no se cuenta con evidencia para hacer esta afirmación, y sería conveniente probar con diversas pobaciones iniciales e idealmente de mayor tamaño.

En cuanto a β_1, vemos que se mantiene positivo en general, como esperábamos, aunque adquiere valores menos altos que en TCT (en este caso la mayor parte de los valores se encuentran entre 0 y 200 y en TCT tienden a estar entre 400 y 600). Sumado al hecho de que β_2 es mayor en magnitud en comparación con TCT, podemos afirmar que con este método de evaluación y en esta ejecución en particular, por lo menos, se generaron jugadores más defensivos.

Nuevamente no hay una clara preferencia por determinados valores de γ_1 y γ_2, puesto que ambos son por momentos positivos y por momentos negativos.

En la Figura 9 mostramos los resultados obtenidos con crossover fijo. Aquí observamos algunas diferencias y similitudes:

- Los valores de β_2 crecieron en magnitud aún más que con crossover random, por lo que los jugadores resultaron aún más defensivos. Sin embargo también crecieron los de β_1, aunque no de manera tan marcada.

- Se mantuvo la tendencia de tener un α_2 menor en magnitud a α_1, pero luego de las 3000 iteraciones el valor de α_2 comenzó a aumentar en magnitud, llegando a ser mayor que α_1, y además no ocurrió que α_1 creciera arbitrariamente.
- Tampoco se prefirieron determinados valores para γ_1 y γ_2, aunque γ_2 fue mayoritariamente positivo. No está claro si dicho cambio se debe a usar crossover fijo, en cuyo caso sería difícil explicar por qué pues no hay una clara relación entre un hecho y otro, o fue una particularidad de la ejecución que se hizo.

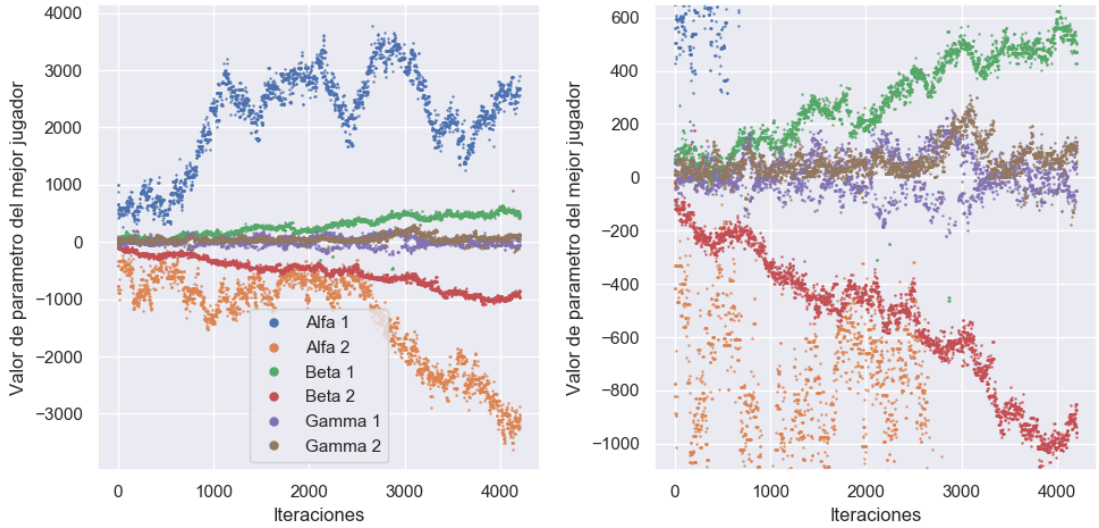

Figure 9: Valores de los parámetros de las soluciones de fitness más alto, por cada generación, obtenidos con el segundo método de evaluación (jugar contra los $n/2$ mejores anteriores), selección ponderada y crossover fijo. A la derecha se muestra el mismo gráfico pero mostrando con más claridad los valores de β_1, β_2, γ_1 y γ_2. Se realizaron 4204 iteraciones.

Sin embargo, con crossover fijo no se observan grandes diferencias, al igual que con TCT.

Como conclusión, el método de evaluación analizado resultó satisfactorio, puesto que en general se mantuvieron las mismas tendencias que en TCT con algunas leves diferencias, pero sin dejar de cumplir con lo que se esperaba principalmente, que era que los signos de los parámetros sean sensatos. Al igual que con TCT no se observó que alguna variante del algoritmo produjera soluciones que comienzan a converger a una en particular, puesto que los valores de los parámetros nunca dejan de variar, aunque lo hagan de manera no caótica como es el caso de selección ponderada. Resulta de mayor interés comparar las soluciones obtenidas con ambos métodos para ver cuál produjo la mejor solución, tarea que fue realizada y se explica en la sección de comparación final.

3.2.4 Experimentación adicional con mutación

Para dedicarnos a ver el efecto que tiene reducir la probabilidad de causar mutaciones, realizamos un sencillo experimento que consistió en correr el algoritmo con método de evaluación TCT, selección ponderada y crossover fijo, pero donde la probabilidad de causar una mutación en un gen es de 0.01 en vez de 0.1.

El resultado esperado es que las mejores soluciones tiendan a repetirse exactamente (es decir sin variaciones) a lo largo de las iteraciones y que los cambios en los valores de los parámetros sean poco frecuentes. De hecho, podría esperarse que se converja a una solución en particular.

Elegimos selección ponderada para evitar el comportamiento que se dio con selección estricta, que posiblemente hubiera dificultado esta experimentación, pero podríamos haber elegido el segundo método de evaluación y/o crossover random también.

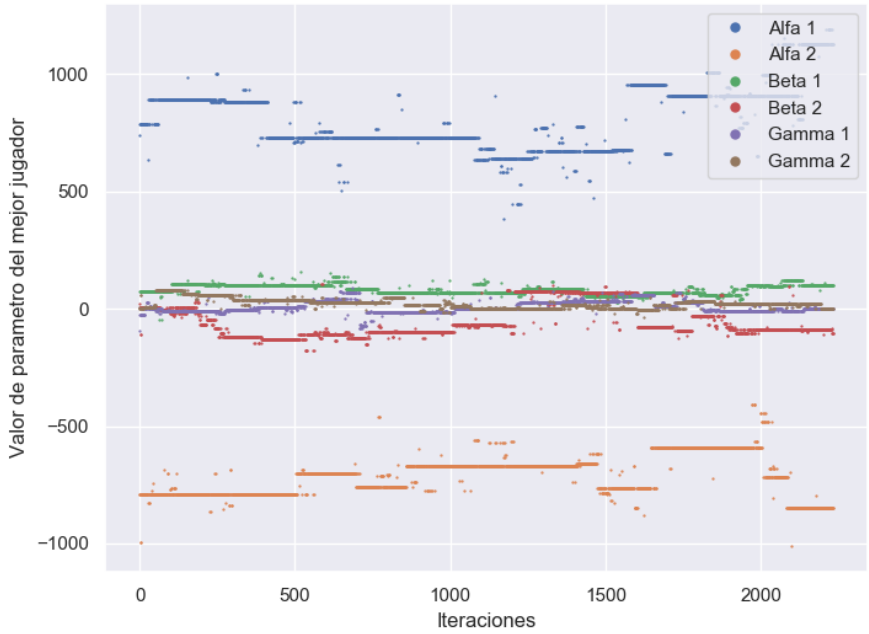

Figure 10: Valores de los parámetros de las soluciones de fitness más alto, por cada generación, obtenidos con evaluación con torneo TCT, selección ponderada, crossover fijo y probabilidad de mutaciones reducida a 0.01. Se realizaron 2233 iteraciones.

En la Figura 10 vemos los resultados, y podemos corroborar nuestra hipótesis: los cambios son mucho más infrecuentes y los valores de los parámetros persisten por varias iteraciones.

Es interesante notar que el valor de β_2 por momentos pasó a ser positivo, causado seguramente por una mutación que cambió al gen de signo. A partir de ese momento los mejores jugadores mantuvieron el signo positivo para β_2 hasta que nuevamente comienzan a ser dominantes las soluciones con β_2 negativo.

Sin embargo no notamos que hayan habido cambios de signo con α_1 y α_2, y esto posiblemente se deba a que o bien no hubo mutaciones de cambio de signo, lo cual podría ser improbable en 2233 iteraciones, o bien las hubo pero los jugadores obtenidos con esos nuevos valores cambiados de signo tuvieron un muy mal desempeño en el torneo TCT, sabiendo la importancia que tiene α_1 y α_2 a la hora de ganar y evitar derrotas.

Aquí también se observa que para γ_1 y γ_2 no hay valores bien definidos, y que los signos de β_1 y β_2 son los sensatos, salvo por el temporal cambio de signo en β_2.

Nuevamente no podemos decir que la solución esté convergiendo, es decir, que los valores de los parámetros se acerquen a valores particulares y no se muevan de ellos. Las mutaciones siempre logran generar cambios que persistan por varias iteraciones.

3.2.5 Comparación final

Luego de haber obtenido los mejores jugadores de cada variante del algoritmo genético, es decir, los de mayor fitness en la última generación, decidimos enfrentar a estos en un torneo todos contra todos como manera de evaluar su rendimiento relativo y ver cuál fue el mejor en relación al resto. Creemos que el torneo todos contra todos es una buena forma de evaluar a un grupo de jugadores por las mismas razones que se dieron al plantear éste como un método de evaluación para el algoritmo genético. Es una manera exhaustiva de comparar el rendimiento de cada jugador en relación al resto. Sin embargo nunca debemos olvidar que no existe una forma correcta e infalible de determinar cuál es el mejor jugador en un conjunto de jugadores, por las razones que mencionamos en la introducción del trabajo.

Cada enfrentamiento entre dos jugadores consistió de 1000 partidos: 2 por cada tablero (alternando quién

empieza), habiendo 20 tableros. Estos tableros se generaron exactamente como en el torneo todos contra todos usado como método de evaluación para computar los fitness de los jugadores.

Para aportar más claridad se indican cuáles son estos jugadores. Entendemos por "Método 1" al torneo TCT, y "Método 2" al que consiste en hacer jugar a cada jugador contra los $n/2$ mejores anteriores:

- Método 1, estricta, crossover random: (3506.37, -1184.18, 73.7608, -529.55, -47.8472, 342.921)
- Método 1, ponderada, crossover random: (3685.79, -2865.87, 356.439, -103.028, 60.7866, 43.7294)
- Método 1, ponderada, crossover fijo: (2210.19, -1515.14, 336.981, -633.743, 83.0344, 123.947)
- Método 1, ponderada, crossover fijo, pocas mutaciones: (1129.86, -846.567, 101.538, -105.688, 2.58489 0.575059)
- Método 2, estricta, crossover random: (3121.42, 1382.81, 307.857, -742.277, 82.2398, 182.659)
- Método 2, ponderada, crossover random: (6058.57, -1839.57, 203.012, -711.021, 228.761, -30.865)
- Método 2, ponderada, crossover fijo: (2571.43, -3047.06, 498.644, -958.882, 133.299, 109.88)

En la siguiente tabla se muestran los resultados:

	Ganados	Empatados	Perdidos
Método 1, estricta, crossover random	4819	3273	3908
Método 1, ponderada, crossover random	5455	4670	1875
Método 1, ponderada, crossover fijo	5360	4626	2014
Método 1, ponderada, crossover fijo, pocas mutaciones	5281	4673	2046
Método 2, estricta, crossover random	2300	2388	7312
Método 2, ponderada, crossover random	1204	1668	9128
Método 2, ponderada, crossover fijo	4329	5206	2465

Podemos observar que con TCT, selección ponderada y crossover random se obtuvo un jugador que resultó ganador entre el resto. Sin embargo, las diferencias son muy escasas con, por ejemplo, crossover fijo, donde se obtuvieron 5360 victorias, que son un poco menos que las 5455 obtenidas con crossover random, y además también son muy similares en cuanto a cantidad de empates y derrotas. En ambos jugadores vemos que $\alpha_1 > \alpha_2$, por lo cual priorizan ganar antes que evitar que gane el oponente, sin embargo con crossover fijo se produjo un jugador bastante más defensivo debido a su valor de β_2, que es bastante mayor en magnitud al de β_1.

Luego sigue el jugador obtenido con TCT y selección estricta. Sin embargo, recordemos que las soluciones de esta variante del algoritmo cambiaban abruptamente, y quizás si hubiéramos tomado un jugador correspondiente a algunas iteraciones menos su rendimiento en este torneo podría haber sido muy distinto. El jugador obtenido es muy defensivo debido a su valor de β_2.

Finalmente, observamos que los jugadores obtenidos con el segundo método de evaluación fueron los peores en el torneo, sobre todo los obtenidos con crossover random, y selección ponderada. Es notable sin embargo el hecho de que con crossover fijo el jugador correspondiente obtuvo pocas pérdidas y muchos empates, y si de hecho vemos sus parámetros, observamos que la magnitud de su α_2 es mayor a α_1, lo cual muy posiblemente haya significado que cuando tenía oportunidad de ganar pero a su vez el oponente estaba a punto de ganar, optó por bloquear al oponente. Además su valor de β_2 es muy negativo, más que β_1, lo cual lo define como un jugador defensivo y por ende explica su desempeño en el torneo, caracterizado por numerosos empates, aunque una cantidad de victorias que no es subestimable.

En cuanto al jugador obtenido con el segundo método de evaluación y selección estricta, su mal desempeño probablemente sea explicado por tener un α_2 de signo positivo, lo cual genera que siempre le de vía libre al oponente para que gane el partido si puede hacerlo.

Finalmente, para el jugador con la menor cantidad de victorias no vemos parámetros indudablemente malos. El parámetro que podríamos mirar con atención en este caso es γ_1, que vale 228.761 y es el más grande en comparación al resto de los jugadores. Es posible que explique en gran parte su mal desempeño,

puesto que un jugador con γ_1 muy alto puede tender a poner fichas aisladas en diversos lugares con el fin de maximizar su cantidad de líneas, sin formar líneas de mayor longitud. Este argumento cobra más fuerza si observamos su valor de β_1, que es muy similar, pero resulta insignificante porque la máxima longitud de las líneas típicamente será mucho menor a la cantidad de líneas no bloqueadas del jugador. Es decir, el aporte que hace la máxima longitud se pierde debido al elevado peso que tiene la cantidad de líneas no bloqueadas.

El mejor jugador (correspondiente a TCT, selección ponderada, crossover random) fue enfrentado al jugador random proporcionado por la cátedra en 10000 partidos, donde en todos ellos comenzó el random con la primera jugada. La cantidad de partidos ganados por el jugador del genético fue 9737, que representa un 97.7% de victorias. Se realizó el mismo proceso pero haciendo empezar al del genético con la primera jugada, ocasionando 9909 victorias, que constituyen un 99.09%. Esto nos dice que tanto el diseño de los parámetros como su valores obtenidos por el algoritmo genético sirvieron para formar un jugador decente, en el sentido de que es capaz de ganarle a un jugador sin estrategia, como lo es el random, de manera consistente.

3.3 Comparación entre ambas técnicas

Finalmente elegimos a uno de los mejores jugadores obtenidos en cada técnica. Los hicimos jugar en 4 tableros diferentes, 2 partidos en cada uno: uno donde empieza el jugador sacado del grid search, y otro donde empieza el sacado del genético. Notar que no tiene sentido hacerlos jugar más de una partida en las mismas condiciones porque estos partidos son determinísticos, por lo tanto todos los partidos serían iguales. Los jugadores que elegimos fueron:

- GridSearch = {1000, -500, 80, -2, 2, 80}
- Genético = {3685.79, -2865.87, 356.439, -103.028, 60.7866, 43.7294}

A continuación los resultados

Juego	Tablero	Empieza	Ganador
4 en línea	6 filas, 7 columnas	GridSearch	GridSearch
4 en línea	6 filas, 7 columnas	Genético	GridSearch
5 en línea	8 filas, 9 columnas	GridSearch	Empate
5 en línea	8 filas, 9 columnas	Genético	GridSearch
7 en línea	14 filas, 14 columnas	GridSearch	Genético
7 en línea	14 filas, 14 columnas	Genético	Empate
9 en línea	17 filas, 18 columnas	GridSearch	Empate
9 en línea	17 filas, 18 columnas	Genético	Empate

En definitiva ambos algoritmos parecieron haber producido jugadores de calidad similar, teniendo el grid search una ligera ventaja para tablero más pequeños. Esto puede deberse a que en el algoritmo genético los jugadores son entrenados para una selección más amplia de tableros y por consiguiente logran saldar las diferencias cualitativas que se produjeron en los primeros tableros con el otro algoritmo.

4 Conclusiones

Gracias al uso de heurísticas, logramos encontrar soluciones buenas para un problema de optimización que hubiera tomado demasiado tiempo resolver a la perfección.

El grid search buscó resolver un problema de optimización difícil con un enfoque simple: considerando solo algunos posibles valores para cada parámetro, sin tener garantías de que el mejor jugador posible esté entre ellas. A pesar de eso, consideramos que los mejores jugadores encontrados con esta técnica tienen un nivel de juego aceptable. Además, todos los mejores jugadores encontrados tienen muchos rasgos en común, lo cual parece indicar que el mejor jugador posible (basado en el método de evaluación del tablero usado) probablemente también comparta muchos de esos rasgos.

En cuanto al algoritmo genético, obtuvimos los mejores resultados con el método de evaluación por torneo todos contra todos, selección ponderada y crossover aleatorio.

Vimos que la selección estricta posee la clara desventaja de generar soluciones completamente diferentes a medida que avanza en las iteraciones, mientras que con selección ponderada los valores de los parámetros tienden a moverse con mayor continuidad, es decir se preservan en cierta medida las características de las mejores soluciones de las generaciones anteriores pero sí ocurren cambios en ellas.

No se notaron grandes diferencias entre los dos métodos de evaluación propuestos, puesto que ambos produjeron soluciones cuyos parámetros $\alpha_1, \alpha_2, \beta_1$ y β_2 contienen signos que para nosotros resultan sensatos, y además las magnitudes de α_1 y α_2 resultaron ser mucho mayores a las del resto de los parámetros, lo cual también es sensato. Sin embargo en la comparación final vimos que las soluciones obtenidas con TCT fueron superiores a las obtenidas con el segundo método, al enfrentarlas en un torneo todos contra todos.

En todas las variantes probadas no se pudieron apreciar tendencias marcadas en cuanto a los valores de γ_1 y γ_2: ambos valores oscilaron alrededor de 0 en casi todos los casos, lo cual nos lleva a pensar que estos parámetros no tienen real importancia para formar jugadores competentes.

No notamos grandes diferencias entre crossover random y crossover fijo que puedan ser tomadas como conclusiones del trabajo. En general con ambos métodos se obtuvieron soluciones similares.

Un aspecto importante con el cual no se experimentó es el tamaño de la población. Los algoritmos genéticos se ven beneficiado al tener una población diversa ya que esto permitirá que los jugadores, a medida que avanzan de generación en generación, puedan evolucionar en distintas direcciones, lo que daría lugar a que se pueda obtener un mejor jugador global. Además, tener mutaciones entre generaciones ayuda a esto dado que evita que se genere un estancamiento de los jugadores y empuja la convergencia hacia jugadores más hábiles.