DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico Nº3

Darwin DT en el mundial 2018

20 de Julio de 2018 — Algoritmos y Estructuras de Datos III

Integrante	LU	Correo electrónico
Bonaccini, Adrián	207/04	abonaccini@dc.uba.ar
Catalano, Arístides	279/10	aristidescata@gmail.com
Musso, Bruno Martín	676/11	brunomusso91@hotmail.com
Romera, Joaquin	183/16	jromera@dc.uba.ar

Entrega	Carolina Gonzalez	I
Reentrega		

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2610 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (+54 11) 4576-3300
http://www.exactas.uba.ar

Índice

1. Introducción

En el presente trabajo práctico se propone modelar un partido de fútbol donde dos equipos de tres jugadores compiten entre ellos para ver cuál truinfa. Cada equipo deberá hacer la mayor cantidad de goles en el arco contrario en una cantidad de turnos dada. El equipo con más goles a favor será el ganador y, en caso de que ambos alcancen la misma cantidad, habrá un empate. El modelo computacional se describe de la siguiente forma:

- *Referí*: Encargado de armar el partido y velar por las reglas. Ubica a los jugadores en las posiciones iniciales elegidas por los equipos y le otorga la posesión de la pelota al equipo que le corresponda. En cada paso se encarga de comunicar el estado actual de la cancha a los equipos. Una vez que estos toman la decisión de cuáles movimientos hará cada uno de sus jugadores, se los comunican al *referí* y se aplican al estado actual de la cancha. Dicho procedimiento se repite hasta que se agotan los turnos establecidos como la duración del partido. Luego se comunica quién fue el ganador o si hubo un empate y se termina el partido.
- *Cancha*: Se define el ancho y el largo de la misma al momento de iniciar cada partido. Recibe los movimientos de los jugadores y devuelve el estado de la cancha luego de ser aplicados. Es responsabilidad de la cancha determinar la información precisa sobre la ubicación de la pelota y los jugadores en cada momento (la misma podría estar siendo disputada por dos jugadores, libre en la cancha o en posesión de un jugador). Además lleva la cuenta de los goles que son realizados por cada equipo y tiene la capacidad de reiniciar la cancha reescribiendo los valores originales respecto de las posiciones y tenencia de la pelota.
- *Equipo*: Es una colección de jugadores. Recibe el estado de la cancha y con dicha información decide qué movimientos va a realizar cada uno de sus jugadores. Cada jugador posee un identificador y un valor que se asocia con su capacidad de disputar la posesión de la pelota.

1.1. Reglas del Juego

El juego cumple una serie de reglas detalladas a continuación:

1. La cancha tiene M columnas y N filas.
2. $M \geq 3$ es impar, N es par y $N \geq 2M$.
3. Un casillero puede contener:
   - Un jugador solo.
   - Un jugador con pelota.
   - La pelota sola.
   - Dos jugadores de equipos contrarios sin pelota.
   - Dos jugadores de equipos contrarios donde uno de los dos tiene la pelota.
4. Los jugadores pueden desplazarse moviéndose de un casillero a otro siempre y cuando estos casilleros compartan un borde o una esquina.
5. En caso de que un jugador se encuentre en el mismo casillero que la pelota, pero no tiene la posesión de la misma, si no hay otro jugador en dicho casillero, entonces el jugador pasa automáticamente a poseer la pelota.
6. En caso de que dos jugadores se encuentren en el mismo casillero y uno de ellos tiene la pelota, automáticamente el jugador sin la pelota intenta quitarle la pelota al otro jugador y tiene la probabilidad mostrada en (1) de lograrlo.
7. El juego tiene una duración en cantidad de pasos. Cada equipo debe decidir en cada paso qué movimiento realiza cada uno de sus jugadores. La duración del partido está dada por la variable S.
8. Los movimientos posibles para un jugador son:
   - Moverse a un casillero vecino dentro del tablero, para lo cual, debe indicar una dirección de las indicadas en la figura 1.

- Tirar la pelota en una dirección con una cierta fuerza (medida en cantidad de pasos). La dirección de la pelota es una de las 8 direcciones mostradas en la figura 1 y la cantidad de pasos indica durante cuanto tiempo la pelota va a moverse, terminados esos pasos la pelota quedará quieta en el lugar donde se encuentre.
  Nota: La pelota se mueve en cada paso dos casilleros en la dirección indicada en vez de un solo casillero como los jugadores, además no está permitido indicar una cantidad de pasos mayor a $\frac{M}{2}$ ni una cantidad de pasos que dejen a la pelota fuera del tablero.

9. Un jugador intercepta una pelota libre sólo si en algún momento ambos (jugador y pelota) se encuentran en un mismo casillero siguiendo las reglas marcadas en (6) y (8) según corresponda. También intercepta la pelota si la misma va de un casillero A a un casillero B y el jugador se encuentra en el casillero intermedio tanto cuando la pelota está en el casillero A como cuando la pelota está en el casillero B. En otras palabras, el jugador está en el camino de la pelota y el mismo se queda quieto mientras la pelota pasa sobre él. Si hay dos jugadores en el casillero intermedio, se utiliza la formula 2 para ver quién se queda con la pelota.

10. El juego comienza con un marcador en 0 para ambos equipos y cada vez que la pelota entra en un arco, se suma 1 punto en el marcador del equipo contrario al arco. Cuando termina el partido, el equipo con más puntos gana. En caso de tener ambos equipos el mismo puntaje, el partido es un empate.

11. Los arcos se encuentran en las posiciones $(\lfloor\frac{M}{2}\rfloor - 1, -1)$, $(\lfloor\frac{M}{2}\rfloor, -1)$, $(\lfloor\frac{M}{2}\rfloor + 1, -1)$ para un equipo y $(\lfloor\frac{M}{2}\rfloor - 1, N)$, $(\lfloor\frac{M}{2}\rfloor, N)$, $(\lfloor\frac{M}{2}\rfloor + 1, N)$ para otro.

12. Los jugadores arrancan en la posición que el equipo quiera siempre que esté en su mitad de la cancha.

13. El equipo que comienza (aquel cuyo arco está en la columna -1) tiene un jugador en la posición $(\lfloor\frac{M}{2}\rfloor, \frac{N}{2} - 1)$ que tiene posesión de la pelota.

14. Cada vez que un equipo mete un gol, los jugadores vuelven a la posición inicial pero el equipo al que le metieron un gol tiene la pelota y por ende tiene un jugador en la posición $(\lfloor\frac{M}{2}, \frac{N}{2} - 1)$ si el gol fue en el arco de la columna N o $(\lfloor\frac{M}{2}, \frac{N}{2})$ si el gol fue en el arco de la columna -1.

1.1.1. Probabilidades de quitar y ganar una pelota

$$P_{quitar}(p_{pelota}, p_{quitador}) = X \geq \frac{P_{quite}(p_{quitador})}{P_{quite}(p_{quitador}) + (1 - P_{quite}(p_{pelota}))}, X\ U(0,1) \tag{1}$$

$$P_{ganar}(p_1, p_2) = X \geq \frac{P_{quite}(p_1)}{P_{quite}(p_1) + P_{quite}(p_2)}, X\ U(0,1) \tag{2}$$

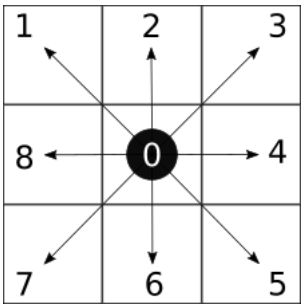


Figura 1: movimientos posibles

1.2. Objetivos

Como se menciona en la sección anterior, los equipos eligen por turno una selección de movimientos válidos para cada uno de sus jugadores que luego se lo comunican al referí. Se desarrolló un *equipo goloso* que, dado un estado de la cancha, elige para cada uno de sus jugadores el movimento que mayor cantidad de puntos le otorgue. Para poder ponderar dichos movimientos se implementó una *función puntuadora parametrizable*. La misma evalúa el estado de la cancha y asigna un puntaje de acuerdo a un conjunto de criterios que busquen y/o produzcan que el equipo haga un gol. El equipo goloso evaluará todas las canchas que se obtengan luego de aplicar todos los movimientos posibles y válidos de sus jugadores con el fin de elegir la combinación que de el mayor puntaje posible.

Con el objetivo de encontrar los movimientos que acerquen a los jugadores del equipo a la mejor estrategia, hay varios criterios que se utilizan para puntuar el estado de una cancha: la posición de los jugadores de ambos equipos, la posesión y la posición de la pelota y, en caso de que la misma esté en movimiento, en qué dirección y durante cuántos turnos seguirá moviéndose. Cada uno de estos criterios definido en la función puntuadora está multiplicado por un coeficiente que a su vez es parámetro de entrada de la función. Dichos parámetros serán los llamados *pesos* de la función. La idea es tener un control sobre la importancia que se le asigne a cada criterio y poder encontrar la combinación que mejore la estrategia del equipo.

Explorar metodologías para encontrar la combinación de pesos que permita al equipo tener la mejor estrategia será el objetivo en el presente trabajo. Sin embargo, la cantidad de combinaciones posibles hace que la exploración del espacio de búsqueda no pueda realizarse en tiempo polinomial. Tal es así que se realizará dicha exploración mediante heurísticas que provean resultados en tiempos acotados.

Dichas heurísticas fueron implementadas utilizando las siguientes técnicas según fue requerido:

- **Grid-Search:** utilizando tanto *búsqueda local* como *grasp* se realiza una búsqueda de manera exhaustiva en vecindarios restringidos que permiten recorrer la grilla sistemáticamente.
- **Algoritmos genéticos:** la exploración de la grilla se traslada a decidir cuáles son los individuos más adaptados, que sobreviven, y cuáles los menos aptos, que son descartados, de manera de evolucionar una población de individuos.

1.3. Consideraciones previas

A lo largo de este informe se hace referencia a *movimientos posibles* o *movimientos válidos*. Con ello se refiere a movimientos (los pases son un tipo especial de movimientos) que un equipo selecciona para sus jugadores que están en conformidad con las reglas detalladas en la sección **Reglas del Juego**. Se tomó la decisión de implementar al *equipo* de forma tal que sólo realice movimientos válidos. De esa manera resulta más sencilla la lógica que modela la cancha.

Vale la pena hacer una aclaración también en cuestiones referidas a la nomenclatura utlizada. A lo largo del informe, al igual que se hace en el enunciado del trabajo práctico, se utilizan algunas palabras a modo de sinónimo.

- **Función Puntuadora:** los parámetros de la misma serán considerados pesos y/o cargas y un conjunto de estos valores representa la *estrategia* y es una *solución* del espacio de búsqueda.
- **Cancha:** la misma puede ser referida como tablero.
- **Jugador Goloso:** el jugador goloso representa un equipo en tanto toma decisiones acerca de los movimientos que cada uno de sus jugadores hará en cada turno. Sin embargo, está también asociado con un equipo de tres jugadores. El término *jugador* puede referirse a ambos modelos de representación y su desambiguación dependerá del contexto.
- **Turno:** en un partido, cada equipo decide qué movimientos realizarán sus jugadores para un turno dado. En algunos casos podrá ser referido como *paso*.

2. Función Puntuadora Parametrizable

2.1. Evaluar Cancha

Para que el equipo sea capaz de poder decidir cuál es el próximo movimiento a realizar se implementó una función que permite probar distintas combinaciones y ponderarlas. La misma evalúa el estado de una cancha de acuerdo a una serie de criterios establecidos y le otorga un puntaje correspondiente. Esta función a su vez es parametrizable, cada criterio tendrá así distinta participación en la conformidad total del puntaje otorgado. Los criterios considerados son los siguientes:

- La pelota está dentro del arco contrario.
- La pelota está dentro del arco propio.
- Distancia de los jugadores del equipo al arco contrario.
- Distancia de cada uno de los jugadores al oponente más cercano.
- Dispersión de los jugadores dentro de la cancha.
- Distancia de los jugadores del equipo a la pelota.

2.2. Pseudocódigo

```
1:  función EVALUAR CANCHA(estado_actual, movimientos, steps, pesos)
2:      estado_resultante ← aplicar movimientos a estado_actual                    ▷ O(1)
3:      res ← 0                                                                    ▷ O(1)
4:      si equipo metió un gol entonces
5:          res ← ∞
6:          retornar res
7:      si rival metió un gol entonces
8:          res ← 0
9:          retornar res
10:     si equipo tiene la pelota entonces
11:         para todo jugador del equipo hacer
12:             res ← res + (D_MAX - distancia del jugador al arco del rival) * peso_d_a   ▷ O(1)
13:             res ← res + (D_MAX - distancia del jugador al rival mas cercano) * peso_d_r   ▷ O(1)
14:         res ← res + dispersión de los jugadores * peso_disp_p                  ▷ O(1)
15:         res ← res/7
16:     si rival tiene la pelota entonces
17:         para todo jugador del equipo hacer
18:             res ← res + (D_MAX - distancia del jugador a la pelota) * peso_d_j_p   ▷ O(1)
19:             res ← res + (D_MAX - distancia del jugador al rival mas cercano) * peso_d_r_p   ▷ O(1)
20:         res ← res + dispersión de los jugadores * peso_disp_p_r                ▷ O(1)
21:         res ← res/7
22:     si la pelota está libre entonces
23:         para todo jugador del equipo hacer
24:             res ← res + (D_MAX - distancia del jugador a la pelota) * peso_d_p_l   ▷ O(1)
25:             res ← res + (D_MAX - distancia del jugador al arco del rival) * peso_d_a_r   ▷ O(1)
26:         para todo jugador del rival hacer
27:             res ← res + (D_MAX - distancia del jugador a la pelota) * peso_d_p_r   ▷ O(1)
28:         res ← res + dispersión de los jugadores * peso_disp_p_l                ▷ O(1)
29:         res ← res/10
30:     retornar res
                                                                                   ▷ Total: O(1)
```

La línea 2 del pseudocódigo corresponde a asignar a la variable el estado resultante de tomar la cancha en su *estado_actual* y aplicar los *movimientos* correspondientes. En caso de que algún jugador haya realizado un pase, se conoce, además de la dirección, la intensidad (*steps*) con la que se hizo. Todo esto es necesario para poder conocer el estado de la cancha luego de aplicar los movientos y proceder a puntuarla. La variable con el estado de la cancha es la que contiene la información sobre la ubicación de cada jugador, la de la pelota y, en caso de que se esté moviendo, en qué dirección y con qué fuerza. D_MAX es la distancia máxima entre dos objetos para una cancha determinada. El parámetro *pesos*, es una lista de los pesos que acompañan los criterios para cada caso determinado:

- El equipo goloso tiene la pelota:
  - *peso_d_a*: distancia del jugador al arco rival.
  - *peso_d_r*: distancia del jugador al rival más cercano.
  - *peso_disp_p*: dispersión de los jugadores.
- El equipo rival tiene la pelota:
  - *peso_d_j_p*: distancia del jugador a la pelota.
  - *peso_d_r_p*: distancia del jugador al rival más cercano.
  - *peso_disp_p_r*: dispersión de los jugadores.
- La pelota está libre:
  - *peso_d_p_l*: distancia del jugador a la pelota.
  - *peso_d_a_r*: distancia del jugador al arco rival.
  - *peso_d_p_r*: distancia del rival a la pelota.
  - *peso_disp_p_l*: dispersión de los jugadores.

Cabe destacar que los criterios **equipo metió un gol** y **rival metió un gol** no tienen ponderación variable. Esta decisión se basó en que el primer criterio corresponde al mejor escenario de todos los posibles mientras que el segundo es el peor. Es por eso que directamente se puntúa a dichos estados con el mejor y el peor valor respectivamente.

En las líneas 15, 21 y 29 se ve que al resultado se lo divide por 7 en los dos primeros casos y por 10 en el último. Esta modificación pretende normalizar el puntaje asignado al criterio que se evalúa. La idea es evitar los escenarios que otorguen un puntaje mayor a situaciones tales como: soltar la pelota y que quede libre, con respecto a mantenerla en posesión. Ese podría ser un caso por el hecho de que es mayor la cantidad de criterios que ponderan esto y no necesariamente el escenario sea mejor en términos de la estrategia para el equipo.

2.3. Análisis de complejidad

Para hacer el análisis de complejidad de la función **Evaluar Cancha**, se asume que todas la operaciones aritméticas realizadas tienen costo constante $O(1)$. Como todos los criterios utilizados se basan en la distancia euclídea entre dos objetos, las complejidades de cada una de estas funciones que realizan ese cálculo tienen costo constante también.

En cuanto a los ciclos realizados, siempre se itera sobre la cantidad de jugadores de cada equipo en el peor caso. Dicho caso se produce cuando la pelota está libre, en esta situación se calculan las distancias para cada jugador del equipo goloso y cada jugador del equipo rival.

La función *dispersión* que se utiliza en todos los casos, es la suma de las distancias entre los jugadores de un equipo, con lo cual también tiene costo $O(1)$.

La complejidad total de esta función es entonces $O(1)$.

3. Equipo Goloso

Éste recibe el estado actual de la cancha en cada turno y decide de acuerdo a la ponderación de la combinación de movimientos cuáles se realizarán. Para tomar una decisión cicla por todas las combinaciones de movimientos posibles y válidos de cada jugador del equipo. En caso de que algún jugador tenga la pelota, evalúa también hacer un pase (otro tipo de movimiento posible) en todas las direcciones válidas y con todos los valores de *steps* posibles (hasta $\frac{m}{2}$ y sin salir de la cancha).

Cada combinación que tiene recibe una ponderación de acuerdo al resultado obtenido por la función *Evaluar Cancha*. Le envía como parámetros el estado actual de la cancha, la combinación de movimientos que desea puntuar y los pesos de los criterios (fijos durante el partido). Luego de evaluar las combinaciones elige hacer el movimiento que más puntos obtenga de la función.

3.1. Pseudocódigo

```
1:  función BUSCAR MOVIMIENTOS(estado_actual)
2:      rank_máximo ← −∞                                              ▷ O(1)
3:      rank_actual ← −∞                                              ▷ O(1)
4:      resultado ← ∅                                                 ▷ O(1)
5:      para todo combinación de movimientos_posibles hacer           ▷ O(1)
6:          si los movimientos de esa combinación son válidos entonces
7:              para todo jugador del equipo hacer
8:                  si el jugador tiene la pelota entonces
9:                      para todo valor de steps válidos hacer        ▷ O(m)
10:                         rank_actual ← evaluar_cancha              ▷ O(1)
11:                         si rank_actual es mayor a rank_max entonces
12:                             rank_max ← rank_actual
13:                             resultado ← combinación de movimientos
14:             rank_actual ← evaluar_cancha
15:             si rank_actual es mayor a rank_max entonces
16:                 rank_max ← rank_actual
17:                 resultado ← combinación de movimientos
18:     retornar resultado                                            ▷ Total: O(m)
```

3.1.1. Análisis del pseudocódigo

- Las líneas 2, 3 y 4 corresponden a la inicialización de las variables.
- En la línea 5 comienza un ciclo que itera sobre todas las combinaciones de movimientos posibles (definidos en la sección *Reglas del Juego*). Para cada jugador pondera todos los movimientos posibles.
- En la línea 6 se chequea que dichos movimientos a puntuar sean válidos, en caso de que alguno no cumpla con la condición se pasa la siguiente combinación.
- Desde las líneas 7 a 13, si algún jugador del equipo tiene la pelota, evalúa qué valor retorna la función puntuadora probando todos los valores de *step* posibles en todas las direcciones. Esto es necesario porque un mismo movimiento de tipo *pase* puede ser hecho con distinta fuerza.
- En la línea 10 **evaluar_cancha**, es un llamado a la función descripta en la sección anterior. En los casos donde el movimiento que se evalúa es de tipo *pase* también se pasa como parámetro el valor de *steps*. De esa manera, se calcula el puntaje cuando el jugador con la pelota realiza el pase (la línea 8 alcanza como condición para asegurar que algún jugador tiene la pelota). Si el ranking obtenido es mayor al mejor ranking obtenido hasta el momento, se actualiza ese valor y se guarda esa **combinación** de movimientos (incluyendo *steps*) en la variable *resultado*.

- En la línea 14, se usa nuevamente la función **evaluar_cancha** sin el valor de *step*, ya que en este caso no se evalúan escenarios en que se realizan pases. Si se encuentra un puntaje mejor que el obtenido hasta el momento se actualiza el máximo y el *resultado* con la combinación de movimientos ganadora.

3.2. Análisis de complejidad

El pseudocódigo detallado en la sección anterior tiene tres ciclos:

- Linea 5: itera sobre todas la combinaciones de movimientos por cada jugador, en total son 9^3, $O(1)$.
- Línea 7: itera sobre los jugadores del equipo, $O(1)$.
- Línea 9: prueba todos los valores posibles para la variable *steps*. Potencialmente pueden ser $\frac{m}{2}$ por las restricciones mencionadas en la sección *Reglas del Juego*, $O(m)$

Los tres ciclos pretenden iterar por todas las combinaciones de movimientos posibles para el equipo. A medida que se construye la combinación, chequea que la elección de cada movimento se mantenga dentro de los rangos definidos por la cancha. Luego chequea que la combinación de movimientos obtenidos genere una jugada válida y, por cada jugador del equipo, en caso de estar en posesión de la pelota, la posibilidad de patear y con qué potencia (*steps*). Cada uno de los posibles estados de la cancha, generados al aplicar cada subconjunto de potenciales movimientos, son evaluados por la función puntuadora seleccionando el de valor máximo. Una vez verificadas todos los estados que posibilitan realizar un movimiento de tipo *PASE*, se hace un segundo cálculo para ponderar los estados donde cada jugador realiza un movimiento de tipo *MOVIMIENTO*.

Como los ciclos de movimientos y jugadores iteran por arreglos de tamaño constante *evaluar_cancha* tiene complejidad O(1). El único cálculo realizado cuyo costo es superior coincide con el cálculo de la máxima potencia del pase (función *steps válidos*). Dicha función, en base a la dirección del movimiento de tipo *PASE* y la posición del jugador, devuelve la potencia máxima válida para dicho movimiento (valor de *steps*).

En el peor de los casos, la pelota podría ir de arco a arco realizando $\frac{n}{2}$, pero es condición del enunciado que el valor máximo permitido sea $\frac{m}{2}$. Como máximo el algoritmo itera $\frac{m}{2}$ veces y se obtiene que el orden de complejidad del algoritmo es de $O(m)$.

4. Experimentación

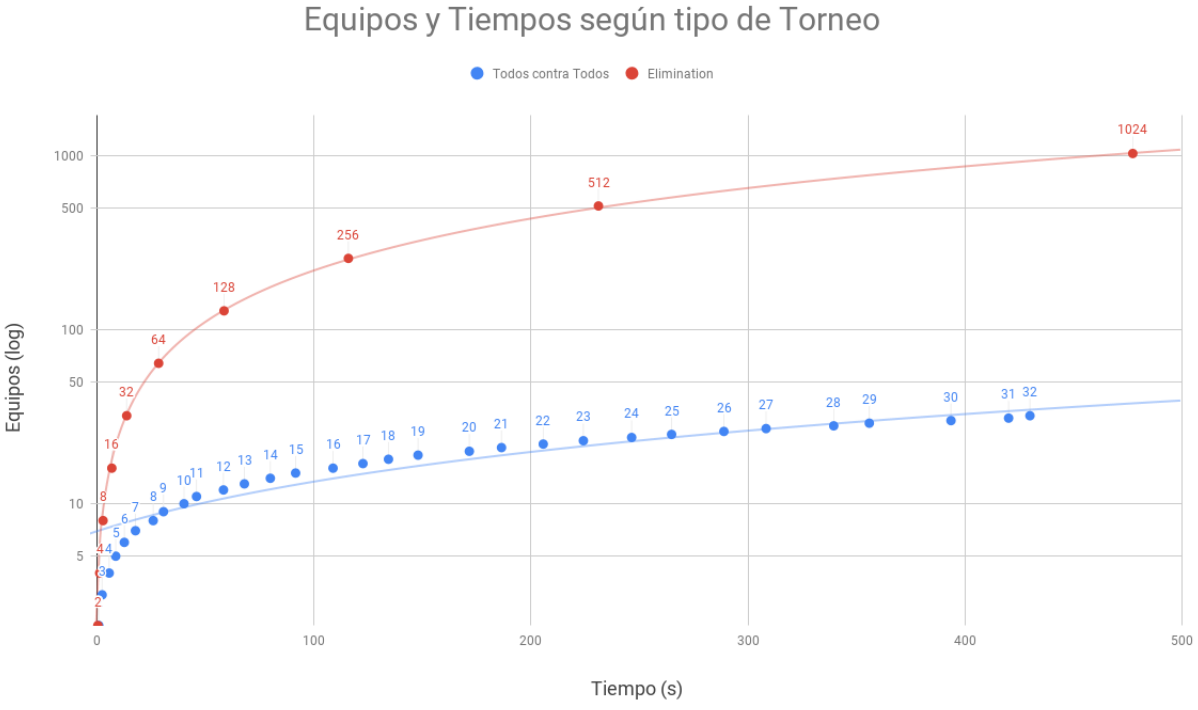
En esta sección se plantean los mecanismos implementados para conseguir valores para los pesos de los criterios definidos en la función **evaluar cancha** que caracterizan una buena estrategia. En el contexto del presente trabajo, una buena estrategia es el conjunto de pesos que harán que el equipo goloso obtenga mejores resultados en un partido que otros rivales. Dependiendo de la heurística utilizada, un mejor resultado puede ser una mayor cantidad de partidos ganados o una mayor cantidad goles marcados. Para obtener estos resultados se generará un conjunto de equipos y se los hará competir entre sí. Este proceso debe repetirse con la intención de ir mejorando los resultados hasta alcanzar algún criterio de corte.

4.1. Modalidad de los Torneos

Todos los equipos generados son **equipos golosos** con distintos pesos para cada uno de los criterios de la función evaluadora. Sin embargo no existe una forma de calificar un equipo de manera unívoca e individual para decidir si esos pesos representan una buena estrategia. En el contexto del partido de fútbol, el equipo debe siempre evaluarse en la competencia frente a otro, incluso no siempre es suficiente si se considera además la existencia de empates y el hecho de que uno de los equipos siempre arranca en posesión de la pelota a diferencia del otro.

Considerando la incapacidad para evaluar individualmente e intentando mitigar las desigualdades propias de los partidos, para realizar las competencias se han definido dos estrategias:

- *Torneo de eliminación directa*: se agrupan todos los equipos de a pares, se enfrentan en un partido de ida (empezando el de la izquierda) y vuelta (empezando el de la derecha) y se descarta el equipo que pierda los encuentros. En el caso de estar empatados en la cantidad de partidos ganados, se desempata por la cantidad de goles (valiendo el doble los goles que hace el equipo que empieza sin la pelota). Luego se repite el proceso con los equipos restantes hasta que haya solo un ganador.
- *Torneo de todos contra todos*: se arman partidos entre cada uno de los equipos involucrados, ida y vuelta. Se arma un ranking y luego se elige el que más partidos haya ganado.



El tipo de Torneo *todos contra todos* parece realizar una búsqueda más exhaustiva con respecto al de *eliminación directa* ya que arma todos los partidos posibles entre dos equipos. Una ventaja que provee la *eliminación directa* por sobre el otro tipo de torneo es la reducción en los tiempos de ejecución. Esto es así dado que la cantidad de partidos es logarítmica con respecto a la cantidad de equipos analizados. Por otro lado, el tipo *todos contra todos* realiza $n \cdot (n-1) \cdot 2$ partidos.

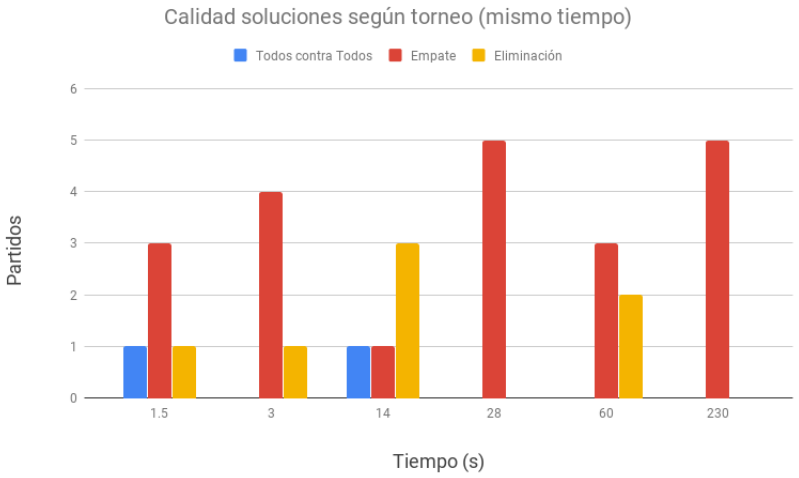
En el gráfico de la página anterior (**Equipos y Tiempos según tipo de torneo**), se muestran los tiempos de ejecución según el tipo de torneo. Puede verse que, para un conjunto grande de equipos, hacer competir todos contra todos resulta muy costoso mientras que con el torneo por eliminación se pueden evaluar conjuntos más grandes en un tiempo menor.

Teniendo en cuenta estos valores, se realizaron dos experimentos para determinar qué impacto tiene elegir un tipo de torneo u otro con respecto al equipo obtenido como ganador. El objetivo es determinar si un tipo de torneo es mejor que otro para evaluar los equipos, dado que un método es mucho más costoso temporalmente que el otro a pesar de ser más exhaustivo en la medida en que todos los equipos compiten entre sí. Así se pretende medir la *calidad* de solución obtenida de cada torneo.

En el primer experimento se eligió la cantidad de equipos para competir en cada tipo de torneo aproximando el tiempo de ejecución de ambos. Entonces la cantidad de equipos que compite en cada torneo quedó definida por los tiempos de ejecución de cada torneo. Por ejemplo: si para 4 equipos *todos contra todos* tarda 1 segundo, vemos la cantidad de equipos que pueden completar *eliminación directa* en aproximadamente 1 segundo. Se generan luego aleatoriamente los equipos necesarios para cada torneo y se los hace competir a los ganadores de ambos un partido de ida y otro de vuelta. Se realizaron cinco iteraciones del mismo experimento con nuevos equipos.

La cantidad de equipos generados por tipo de torneo y tiempo se muestra en la siguiente tabla:

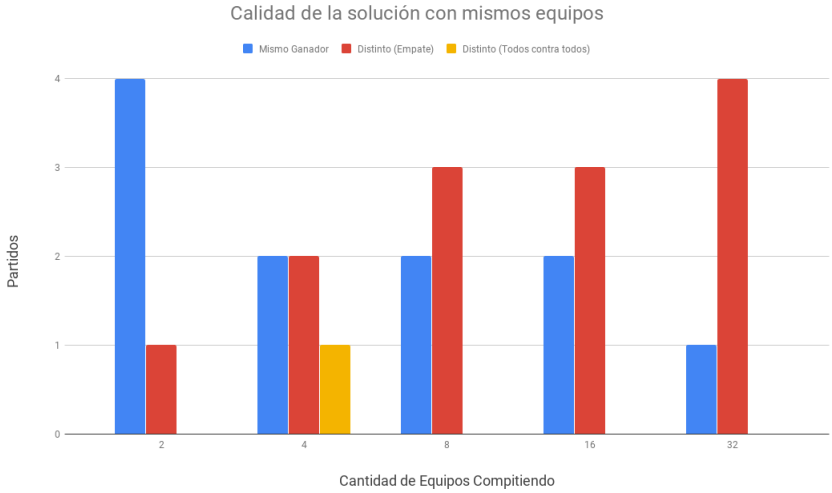
Tiempo (s)	Todos contra Todos	Eliminación Directa
1.5	2	4
3	3	8
14	6	32
28	9	64
60	12	128
230	24	512



En los resultados se observó que predominaron los empates. Si bien hacer a todos los equipos competir entre sí es más exhaustivo, el costo temporal no permite evaluar una gran cantidad de equipos en comparación con un torneo de eliminación directa. Este último evalúa los equipos en menor profundidad pero al poder tomar una mayor cantidad, se pueden abarcar más soluciones en la misma cantidad de tiempo.

Para el segundo experimento se hicieron competir los mismos equipos bajo los diferentes torneos. Se repitió el experimento para 2, 4, 8, 16 y 32 equipos. Al obtener el ganador de cada certamen se esperó encontrar al mismo equipo en ambos bajo la hipótesis de que el mejor debería prevalecer sin importar el tipo de competición. En caso de obtener resultados diferentes se los hizo jugar entre sí partidos de ida y vuelta y se registró el ganador. Se realizaron 5 iteraciones del mismo experimento con nuevos equipos.

Los resultados obtenidos muestran que en el 44 % de los casos se obtuvo el mismo ganador y que en el 52 % hubo distintos equipos que no se sacaron diferencia entre sí. Si bien el resultado no es contundente, se puede notar que en el 96 % de los casos no hubo diferencias cualitativas importantes.



Dados estos resultados iniciales se decidió utilizar el método de Eliminación Directa con partidos de ida y vuelta como torneo para evaluar los equipos en el resto de los experimentos. Si bien este método no asegura mejores soluciones que un torneo de todos contra todos, este último tampoco lo hace, y Eliminación Directa asegura poder realizar una mayor cantidad de experimentos en una menor cantidad de tiempo sin degradar la calidad de los resultados.

4.2. Espacio de búsqueda

El espacio de búsqueda donde habitan las estrategias queda definido por la cantidad de pesos a optimizar y por la cantidad de valores posibles que cada uno de ellos pueda tomar. Por ejemplo, si se quisiera optimizar una función con 3 parámetros y cada uno pudiera tomar 10, 10 y 5 valores posibles respectivamente, entonces la grilla o espacio de búsqueda generado estaría formado por $10^2 * 5$ soluciones posibles.

Con los criterios definidos en la *sección 2.1*, se establecen 10 parámetros, cada uno con valores que varían entre 0 y 1. Dependiendo de la granularidad con que se quiera experimentar, el espacio de búsqueda puede crecer considerablemente. Si cada parámetro pudiese variar en 0,01 su valor, por cada parámetro habría 10^2 posibilidades, dando un total de $(10^2)^{10} = 10^{20}$ variaciones distintas. Cada una de estas variaciones en los parámetros representan una estrategia distinta. Vemos entonces que aún utilizando el sistema de torneo por eliminación directa para comparar equipos, recorrer exhaustivamente este espacio de búsqueda resulta impracticable debido a la cantidad de potenciales combinaciones. Para hacer posible el recorrido de este espacio de soluciones, se implementan las distintas heurísticas que se presentan a continuación.

4.3. Grasp

Esta heurística también conocida como *Greedy Randomized Adaptive Search Procedure* es un algoritmo que consiste en iteraciones sobre sucesivas construcciones de soluciones golosas generadas al azar que se van mejorando iteración a iteración. Mientras no se verifique el criterio de parada, se ejecuta la siguiente secuencia:

- Solución = construir solución al azar.
- Mejor Solución = búsqueda local (Solución).
- Actualizar Solución (Solución, Mejor Solución).

El proceso de *búsqueda local* genera un vecindario al azar de 64 equipos y los enfrenta en un *torneo de eliminación directa* con partidos de ida y vuelta, desempatando por la cantidad de goles anotados. Para actualizar la solución luego juega un partido con las mismas características entre la nueva solución y la anterior. Este proceso se corresponde a la acción de conseguir el mejor equipo de un vecindario dado y luego hacerlo competir con el mejor actual. El criterio de corte utilizado será el número de iteraciones dadas las limitaciones temporales que presentan los largos tiempos de experimentación.

4.4. Búsqueda Local

El esquema general para aplicar búsqueda local es el siguiente:

- Elegir una solución inicial s.
- Elegir una solución v vecina a s que obtenga un puntaje superior de acuerdo a lo ponderado por la función evaluadora.
- Reemplazar v por s.
- Repetir hasta que v es mejor que todos sus vecinos o hasta llegar a x iteraciones.

La solución inicial en este caso es generada al azar. Para evaluar las soluciones vecinas a la inicial, los equipos golosos se enfrentan en el *torneo de eliminación directa* previamente descripto. Se exploraron dos formas de construir el vecindario de la solución inicial, las cuales son descriptas a continuación.

4.4.1. Vecindario Random

Este vecindario se construye a partir de un vector de parámetros dado (un equipo), y una cantidad fija de vecinos, por defecto la cantidad de vecinos generada es 64. El tamaño elegido es potencia de 2 para poder realizar *torneos de eliminación directa*. Recibe por parámetro la **variación** que se le va a aplicar a cada parámetro del equipo.

En cada iteración se crea un *equipo_nuevo* igual al *equipo_inicial*. Luego por cada *parámetro*, se suma o resta la *variación*, dependiendo del *número_random* obtenido. Si luego de la operación ese paŕametro tiene un valor mayor que 1 o menor que cero, se saturan en esos valores para que queden dentro del rango definido. Por último al terminar de calcular cada parámetro del *equipo_nuevo* este se agrega al vecindario.

4.4.2. Vecindario Grande

Para generar un vecindario de este tipo se realiza el siguiente procedimiento:

- Se crean dos equipos a partir del original, *equipo_1* y *equipo_2*.
- A cada parámetro del *equipo_1* se le **resta** la *variación* definida.
- A cada parámetro del *equipo_2* se le **suma** la *variación* definida.
- Se agregan al vecindario todas las combinaciones que hay entre parámetros que hay entre los dos equipos. Esto significa que, para cada parámetro tengo dos opciones posibles tomar el del *equipo_1* o el del *equipo_2*. Quedando un total de $2^{10} = 1024$ combinaciones posibles.

```
1: función VECINDARIO RANDOM(equipo_inicial, variación, cantidad_vecinos)  ▷ Total: O(cantidad_vecinos)
2:     i ← 0
3:     mientras i sea menor que cantidad_vecinos hacer                      ▷ O(cantidad_vecinos)
4:         equipo_nuevo ← equipo_inicial
5:         para cada parámetro del equipo_nuevo hacer
6:             numero_random ← numero entre 0 y 100
7:             si numero_random es menor que 50 entonces
8:                 parámetro ← parametro + variación
9:             sino
10:                parámetro ← parametro − variación
11:            si parámetro es menor a 0 entonces
12:                parámetro ← 0
13:            si parámetro es mayor a 1 entonces
14:                parámetro ← 1
15:        agregar el equipo_nuevo al vecindario
16:        i ← i + 1
```

Un ejemplo de un equipo con solo 3 parámetros, deja un vecindario de 8 equipos nuevos: Dado un equipo con parámetros: $[4, 7, 5]$ y una *variación* de 1, el vecindario definido sería el siguiente:

[3 6 4] [3 6 6] [3 8 4] [3 8 6] [5 6 4] [5 6 6] [5 8 4] [5 8 6]

4.4.3. Experimentos sobre el tipo de vecindario

Como se puede apreciar en el gráfico de tiempos según tipo de Torneo de la **sección 4.1**, hacer competir 1024 equipos demanda más tiempo que hacer competir 64. Los tiempos de esas mediciones son los siguientes:

Tiempo (s)	Cantidad de quipos
1.356	4
2.905	8
6.940	16
13.796	32
28.517	64
58.637	128
115.932	256
231.135	512
477.301	1024

Para intentar encontrar un balance entre la exhaustividad y la velocidad de la búsqueda se realizó el siguiente experimento: a partir de la misma solución inicial se realizó una búsqueda local utilizando los dos esquemas, luego se hizo competir las soluciones finales de ambos. Como eliminación directa sobre 1024 equipos llevó en promedio 8 minutos, se hizo una iteración de la búsqueda local que utiliza el vecindario grande y se hicieron 10 iteraciones de la que usa el vecindario random para compararlos en base a tiempos aproximados. El experimento se repitió 10 veces otorgando los siguientes resultados:

Iteración	Resultado	Tiempo vecindario random / vecindario grande
1	Empate	71,5 %
2	Empate	75,5 %
3	Empate	68,6 %
4	Empate	70 %
5	Empate	76,59 %
6	Empate	68,72 %
7	Empate	68,28 %
8	Empate	70,79 %
9	Empate	69,19 %
10	Vecindario Grande	68,98 %

De acuerdo a los resultados arrojados por estos experimentos decidimos utilizar el vecindario random en los experimentos siguientes. El costo extra del tiempo que consume hacer competir a 1024 equipos no se tradujo a los resultados. Una posible explicación es que quedarse con el mejor de un vecindario grande no tiene la robustez de moverse de un vecindario a otro repetidas veces.

4.4.4. Experimentos sobre el radio del vecindario

Uno de los problemas que surgen en las estrategias de búsqueda local es el estancarse en un máximo o mínimo local. Una solución puede ser mejor que todas sus vecinas dependiendo de cómo armamos el vecindario, si el radio del mismo es demasiado pequeño *escalar* de una solución a una mejor puede tardar mucho o incluso puede no encontrar soluciones mejores; mientras que un radio demasiado grande podría resultar en explorar soluciones muy distantes y la idea de explorar un vecindario perdería sentido -efectivamente esto es algo que aprovecha, aunque de distinta manera, el método Grasp-.

Se decidió comprobar el efecto de reducir, o afinar, el radio en cada iteración. Primero se probó con el vecindario grande y luego con el generado al azar. La hipótesis es que si la búsqueda es más exhaustiva entonces se debería ver una mejora en achicar el radio en cada iteración, mientras que en el otro caso al solo ver 64 vecinos por vez, afinar la búsqueda no debería introducir cambios notorios. El experimento consistió en arrancar con un radio de 0.16 y en cada iteración dividirlo por 2, realizando 4 iteraciones. Como siempre en cada iteración se juega un torneo de eliminación directa entre los vecinos y se va avanzando el vecindario de acuerdo al ganador. Luego se repitió el mismo proceso pero dejando fijo el radio en 0.16. Al finalizar se hicieron competir ambos ganadores. Para el vecindario de 1024 vecinos se repitió el experimento 5 veces por el costo temporal. Para el de 64 vecinos por iteración se realizaron 50 repeticiones del experimento.

	Radio fijo	Radio variable	Empates
Vecindario grande	2	3	0
Vecindario random	15	1	34

Dadas las pocas repeticiones del primer experimento no podemos sacar conclusiones definitivas sobre el impacto de ambas variantes. Para el segundo caso sí encontramos una tendencia a los empates, como era esperado. En base a estos resultados se decidió dejar un radio fijo. El valor del mismo será sujeto a análisis en la comparación con Grasp.

4.4.5. Búsqueda Local sin Grasp

En este experimento se quiere comprobar la robustez de Grasp, se espera que al construir una solución nueva al azar en cada iteración y hacer búsqueda local alrededor de la misma se puede evitar estancarse en máximos locales, a diferencia de hacer muchas iteraciones de búsqueda local partiendo de un mismo lugar. Para esto se realizó el experimento de la siguiente manera: se generó una solución al azar que serviría como punto de partida para ambos algoritmos, luego se realizaron 20 iteraciones -para búsqueda local implicó moverse por los vecinos de la solución inicial, para Grasp en cambio, resultó en generar 20 soluciones nuevas y ver el mejor del vecindario de cada una-. Para acentuar las diferencias entre ambos métodos se fijó un radio de 0.025 y el vecindario se armó en base a 16 vecinos. Por último, se hizo competir a los ganadores de ambos en partidos de ida y vuelta. El experimento se repitió 50 veces obteniendo los siguientes resultados:

	Grasp	Búsqueda Local	Empates
Partidos ganados	39	9	2

Habiendo ganado en el 78 % de los experimentos consideramos que Grasp ofrece una mejor alternativa para explorar el espacio de búsqueda de soluciones que hacer búsqueda local únicamente. El peligro de caer en un máximo o un mínimo local es contrarrestrado por la generación al azar de puntos de partida en Grasp, asimismo el salto de las soluciones iniciales entre iteraciones permite un recorrido más amplio del espacio de soluciones posibles.

4.4.6. Evolución de las soluciones de Grasp

El presente experimento se realiza para analizar el nivel de mejora de las soluciones de la heurística Grasp a lo largo del tiempo. Para el mismo se ejecutó el algoritmo durante 12 horas y se eligieron 10 soluciones intermedias para hacerlas competir entre sí (incluyendo la primera y la última). El tipo de torneo para el paso final fue *todos contra todos*, el mismo se repitió 10 veces.

Se esperó que las últimas soluciones encontradas fueran mejores. Sin embargo, como se puede observar en la siguiente tabla, si bien la última iteración fue la ganadora y la primera salió última, el resto de los equipos no cumplen con las expectativas y tampoco arrojan una tendencia definida.

Solución	Puntos	Goles
9	472	1198
2	456	1253
3	403	916
4	316	780
5	258	612
8	250	730
6	196	529
1	127	481
7	84	384
0	73	294

Podemos concluir dada la naturaleza del problema, que incluye elementos probabilísticos, que poder ganar un partido no es una propiedad transitiva de cada equipo. Debido a esto las heurísticas ayudarán a encontrar *buenas* soluciones pero no se puede esperar encontrar una *óptima* -habrá que considerar la existencia de tales soluciones únicamente bajo algún criterio estadístico-. Por otro lado, el inabarcable tamaño del espacio de soluciones posibles hace necesario invertir tiempo en explorarlo, independientemente de la heurística utilizada. Efectivamente, como trabajo futuro se espera contar con más tiempo para explorar exhaustivamente el espacio de búsqueda, experimentar con distintos criterios de corte, formas de evaluar las soluciones y sofisticar la función evaluadora parametrizable.

4.5. Algoritmo Genético

A continuación se detallará cómo fue utilizada esta heurística para optimizar los pesos de los criterios. Este algortimo, recibe por parámetro el *tamaño de la población*, la cantidad máxima de *generaciones* que se van a crear y distintos valores booleanos (a modo de *flags*) que determinan qué tipo de variedad de las funciones son elegidas para la ejecución: *crossover* (aleatorio o agrupado), *fitness* (partidos ganados o goles realizados) y *selección* (competencia o selección ponderada). Cada combinación de estos parámetros determina una ejecución que hace al algoritmo genético distinto.

4.5.1. Pseudocódigo

Este algoritmo posee como criterio de corte dos podas, la primera es la cantidad máxima de generaciones permitidas y la segunda es la cantidad máxima de veces seguidas en las que una combinación puede permanecer como la mejor puntuada, esta condición cumple la función de evitar seguir buscando una mejor generación si luego de varias generaciones se sigue obteniendo el mismo resultado.

El parámetro de entrada *fitness* tiene como función determinar que función de fitness realizará el algoritmo. Dicha función puede dictaminar el puntaje de la combinación en base a:

- La cantidad de goles realizados en el torneo.
- El puntaje obtenido en el torneo.

```
1: función GENETICO(tam_población, cant_max_generación, cant_max_invicto, crossover, fitness, selección)
2:     poblacion ← Generar_población_aleatoria(tam_población)
3:     pesos_ganadores ← Realizar_torneo(población)
4:     pesos_anteriores ← ∅
5:     gen_actual ← 0
6:     gen_invicto ← 0
7:     mientras gen_actual < cant_max_generación & gen_invicto < cant_max_invicto hacer
8:         pesos_anteriores ← pesos_ganadores
9:         comb_sobrevivientes ← Selección(población, fitness, selección)
10:        poblacion ← Crossover(comb_sobrevivientes, crossover)
11:        pesos_ganadores ← Realizar_torneo(población)
12:        si pesos_ganadores = pesos_anteriores entonces
13:            gen_invicto ← gen_invicto + 1
14:        sino
15:            gen_invicto ← 0
16:        gen_actual ← gen_actual + 1
17:    retornar pesos_ganadores
```

Luego, para elegir las combinaciones que se utilizarán para crear nuevas con las que se conformarán la siguiente generación, existen dos métodos de selección determinado por el parámetro *selección*:

- Competencia: Elegir la mitad mejor puntuada.
- Selección Ponderada: Elegir (a lo sumo la mitad) de combinaciones cuyo puntaje supere una probabilidad determinada por el algoritmo.

Por último, el parámetro *crossover* decide cómo se realizarán las nuevas combinaciones para incorporar en la generación siguiente:

- Crossover Aleatorio: Se eligen aleatoriamente ciertos pesos de cada una de sus combinaciones padres.
- Crossover Especificado: Se le otorga los pesos de cuando la pelota esta en posesión del equipo propio o libre de una de sus combinaciones padre y los pesos de cuando la pelota esta en posesión del equipo rival de la otra combinación para una de las nuevas e invertimos la selección para la otra.

Si se realiza una selección ponderada y la cantidad de combinaciones elegidas es menor a la mitad de la población original, el crossover realizará más de dos hijos para algunas combinaciones con el fin de cubrir el faltante producido por dicha selección.

Durante el proceso de crossover, el algoritmo posee, a su vez, un proceso de mutación donde cada peso tiene una probabilidad del 5 % en conseguir un nuevo valor.

4.5.2. Experimentos sobre ganadores de generaciones

Al momento de comenzar las experimentaciones de los algoritmos genéticos, se estebleció utilizar 5 generaciones como límite para la cantidad máxima seguida permitida que una puede salir como ganadora para el criterio de corte (además del límite por generaciones totales). Los resultados obtenidos, mostraron que con este criterio, muchos de los algoritmos genéticos alcanzaron esta condición en pocas generaciones:

Generación	Combinación de pesos									
0	0.93	0.59	0.65	0.2	0.85	0.19	0.3	0.01	0.62	0.21
1	0.92	0.56	0.3	0.45	0.08	0.73	0.86	0.15	0.54	0.83
2	0.89	0.21	0.31	0.27	0.69	0.09	0.9	0.32	0.67	0.33
3	0.57	0.56	0.3	0.27	0.69	0.09	0.86	0.01	0.54	0.83
4	0.92	0.56	0.3	0.63	0.39	0.78	0.55	0.01	0.54	0.83
5	0.92	0.56	0.3	0.27	0.69	0.09	0.55	0.01	0.54	0.83
6	0.87	0.56	0.3	0.47	0.08	0.73	0.86	0.01	0.54	0.83
7	0.87	0.56	0.3	0.47	0.08	0.73	0.86	0.01	0.54	0.83
8	0.87	0.56	0.3	0.47	0.08	0.73	0.86	0.01	0.54	0.83
9	0.87	0.56	0.3	0.47	0.08	0.73	0.86	0.01	0.54	0.83
10	0.87	0.56	0.3	0.47	0.08	0.73	0.86	0.01	0.54	0.83

En esta experimentación se alcanzó la condición de corte en la sexta generación que, si se lo compara con el límite de 50 generaciones como tope establecido, resulta prematura. A partir de los resultados de este experimento, se decidió aumentar el límite de generaciones ganadoras iguales a 10.

La idea principal de los algoritmos genéticos se basa en que las poblaciones vayan evolucionando hacia una que el algoritmo considere mejor en base a los criterios que utilice. Para ello, el siguiente experimento tomará para cada una de las ocho variantes -generadas combinando los diferentes métodos de selección, fitness y crossover- una población inicial elegida aleatoriamente de 64 combinaciones de pesos a la que se le aplicará el algoritmo genético hasta un máximo de 50 generaciones donde se realicen torneos por eliminación directa.

Estos ganadores generacionales competirán en un nuevo torneo de todos contra todos bajo la hipótesis de que el ganador de cada generación debería ser capaz de ganarle a los ganadores de sus generaciones previas.

Se espera que, en ciertos casos, esta hipótesis no se cumpla debido a la gran cantidad de pasos probabilísticos que posee no solo el algoritmo genético, ya que no siempre garantiza por su forma de crear la siguiente generación que ella le gane a sus predecesoras, sino también la parte probabilística de cada partido, como son las probabilidades de que un jugador obtenga la pelota de un rival.

Dicha información se detallará con la siguiente tabla donde aparece el número de posición del ranking y el número de generación (desde la 0 hasta la 49) perteneciente a dicha posición para cada variante del genético.

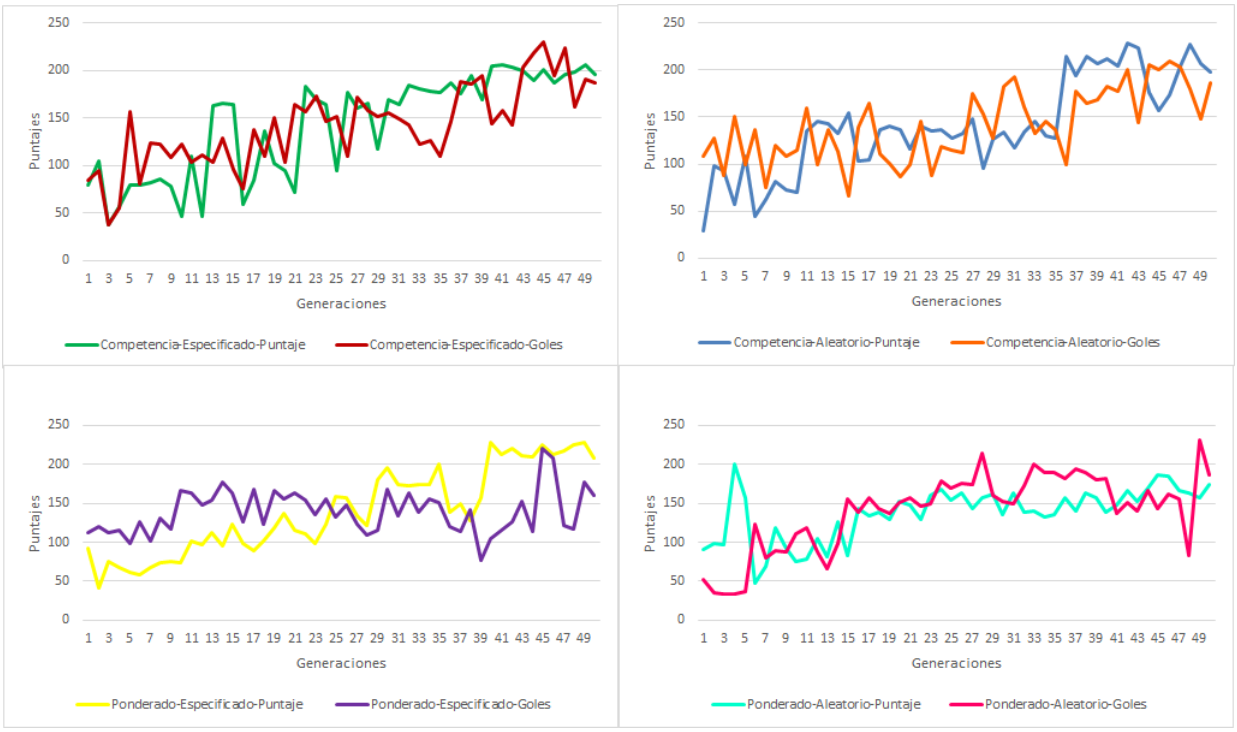
Ranking	Generaciones							
	Competencia-Especificado-Puntaje	Competencia-Especificado-Goles	Competencia-Aleatorio-Puntaje	Competencia-Aleatorio-Goles	Ponderado-Especificado-Puntaje	Ponderado-Especificado-Goles	Ponderado-Aleatorio-Puntaje	Ponderado-Aleatorio-Goles
1	48	44	41	45	39	44	3	48
2	40	46	47	43	48	45	44	27
3	39	43	42	46	47	13	45	32
4	41	42	37	41	44	48	49	36
5	44	45	35	44	41	16	43	34
6	42	38	39	30	46	29	23	33
7	47	48	48	49	40	9	46	37
8	49	36	38	29	45	18	41	49
9	46	49	40	39	42	20	47	35
10	37	37	46	47	43	31	30	39
11	43	22	49	40	49	14	25	38
12	35	26	36	36	34	10	37	23
13	45	20	43	26	29	49	28	25
14	31	47	45	38	28	33	22	26
15	21	27	44	16	32	23	38	31
16	32	40	14	37	33	19	48	24
17	33	21	26	31	30	12	4	43
18	34	4	32	10	31	21	35	45
19	25	29	11	27	24	42	27	28
20	36	24	12	3	38	34	24	20
21	22	28	21	48	25	11	42	16
22	29	18	18	21	36	25	19	14
23	38	30	23	33	35	37	20	46
24	27	23	17	42	19	32	40	29
25	13	35	19	15	26	22	26	41
26	30	39	22	12	37	30	15	19
27	14	31	10	5	23	24	36	22
28	23	41	31	34	14	7	32	30
29	12	16	29	32	27	41	17	21
30	26	13	13	28	18	15	31	17
31	17	33	25	1	20	5	39	44
32	28	6	33	7	12	17	34	42
33	10	9	34	23	21	26	29	15
34	1	7	24	24	10	46	16	18
35	18	32	28	9	17	35	33	40
36	19	11	30	13	22	1	21	5
37	24	25	20	25	15	8	18	10
38	7	34	4	17	11	47	13	9
39	16	17	16	0	13	28	7	13
40	6	8	15	8	0	3	11	7
41	0	12	1	18	16	40	1	11
42	5	10	27	35	2	43	2	8
43	4	19	2	11	8	36	8	47
44	8	14	7	20	7	0	0	6
45	20	1	8	4	9	2	14	12
46	15	0	9	2	3	27	12	0
47	3	5	6	22	6	39	10	4
48	9	15	3	19	4	6	9	1
49	11	3	5	6	5	4	6	3
50	2	2	0	14	1	38	5	2

Como se puede apreciar, si bien ocurre que algunas generaciones más viejas esten mejor posicionadas que otras posteriores, como en Ponderado-Aleatorio-Puntaje que la tercera generación salió como la mejor puntuada, predominan las últimas generaciones en el top 10 de cada variante, mientras que, en las peores 10 posiciones del ranking predominan las primeras generaciones. La experimentación realizada para Ponderado-Especificado-Goles es el caso que menos refleja dicha hipótesis ya que se encuentran más repartidas las generaciones. Las siguientes tablas ilustrarán un poco mejor dichas observaciones:

Rango Generaciones	Top 10							
	Competencia-Especificado-Puntaje	Competencia-Especificado-Goles	Competencia-Aleatorio-Puntaje	Competencia-Aleatorio-Goles	Ponderado-Especificado-Puntaje	Ponderado-Especificado-Goles	Ponderado-Aleatorio-Puntaje	Ponderado-Aleatorio-Goles
0-9	0	0	0	0	0	1	1	0
10-19	0	0	0	0	0	3	0	0
20-29	0	0	0	1	0	2	1	1
30-39	2	3	4	2	1	1	1	7
40-49	8	7	6	7	9	3	7	2

Bottom 10								
Rango Generaciones	Competencia-Especificado-Puntaje	Competencia-Especificado-Goles	Competencia-Aleatorio-Puntaje	Competencia-Aleatorio-Goles	Ponderado-Especificado-Puntaje	Ponderado-Especificado-Goles	Ponderado-Aleatorio-Puntaje	Ponderado-Aleatorio-Goles
0-9	7	5	9	3	9	4	7	7
10-19	2	5	0	4	1	0	3	2
20-29	1	0	1	2	0	1	0	0
30-39	0	0	0	1	0	3	0	0
40-49	0	0	0	0	0	2	0	1

Con el fin de brindar una mejor visualización de los datos, a continuación se muestran cuatro gráficos agrupados de a dos variantes con el objetivo de optimizar el espacio y no saturar los datos. Se espera ver con mayor claridad la forma en que las generaciones fueron mejorando con respecto a sus predecesoras en base a sus puntajes obtenidos del torneo respondiendo a la idea de *evolución* de una población.



Si bien los puntajes son un indicador de qué combinación resultó mejor, ellos fueron calculados para torneos internos de cada variante. Para poder determinar cuál variante brindó una mejor solución se decidió realizar *torneos de todos contra todos* entre el resultado final de cada variante del experimento previo. Se realizaron con dicha modalidad una cantidad de 20 torneos y se calculó qué combinación de pesos salió victoriosa la mayor cantidad de veces.

	Cantidad Victorias	Puntaje	Goles	Combinación de pesos utilizada									
Competencia-Crossover Especificado-Goles	4	30	52	0.86	0.37	0.1	0.62	0.02	0.63	0.16	0	1	0.18
Competencia-Crossover Especificado-Puntaje	0	21	42	0.68	0.35	0.53	0.51	0	0.16	0.67	0.09	0.52	0.4
Competencia-Crossover Aleatorio-Goles	2	18	35	0.92	0.28	0.11	0.53	0.21	0.69	0.97	0	0.4	0.64
Competencia-Crossover Aleatorio-Puntaje	0	16	43	0.97	0.41	0.06	0.33	0.66	0.91	0.96	0.05	0.08	0.87
Ponderado-Crossover Especificado-Goles	9	25	49	0.72	0.64	0.11	0.85	0.04	0.89	0.62	0.04	0.61	0.93
Ponderado-Crossover Especificado-Puntaje	1	14	37	0.68	0.35	0.5	1	0	0.34	1	0.08	0.51	0.59
Ponderado-Crossover Aleatorio-Goles	4	23	37	0.85	0.55	0.11	0.76	0.21	0.99	0.76	0.01	0.13	0.12
Ponderado-Crossover Aleatorio-Puntaje	0	14	32	0.76	0.5	0.03	0.67	0.12	0.2	0.79	0.02	0.73	0.96

Viendo la tabla anterior, la variante ganadora es la que utilizó el método de selección ponderada, un crossover especificado y una función de fitness por goles habiendo ganado 9 de 20 torneos, con los pesos, puntaje y cantidad de goles realizados detallados en la tabla.

4.6. Genético vs Grasp

En base al resultado del experimento de la sección anterior, se utilizará la variante mejor ponderada del algoritmo genético para realizar una comparación contra los resultados de Grasp. Con esta experimentación se espera que utilizar el algoritmo genético genere mejores soluciones que Grasp ya que este último busca mejorar la solución buscando nuevos vecindarios de manera aleatoría y, en cambio, el genético lo hace en base a evolucionar una poblacion inicial combinando las mejores soluciones obtenidas de ella.

Para ello se realizará un experimento previo para obtener el resultado que se comparará contra el de Grasp utilizando tiempos de ejecución similares. Dadas las diferencias entre ambos algoritmos, se utiliza el tiempo como criterio que los unifique. Se partirá de una población elegida aleatoriamente de tamaño 256 hasta un máximo de 65 generaciones, utilizando nuevamente *torneos por eliminación directa*.

Tanto el experimento hecho sobre el algoritmo genético como el que corresponde al de Grasp (ganador obtenido de la competición detallada previamente en la última tabla de la sección correspondiente) se requirieron 6 horas para finalizar su ejecución. A continuación se detallan los pesos del resultado de cada algoritmo:

Genético $\Longrightarrow$	0.89	0.35	0.04	0.62	0.03	0.82	0.80	0.02	0.38	0.85
Grasp $\Longrightarrow$	0.64	0.44	0.23	1.00	0.40	0.00	0.62	0.32	0.29	0.34

Para comparar ambos algoritmos se realizaron 100 partidos entre los pesos detallados arriba para calcular cuál de los dos gana una mayor cantidad de veces.

	Cantidad Victorias	Puntaje	Goles	Combinación de pesos utilizada									
Genético	88	421	937	0.89	0.35	0.04	0.62	0.03	0.82	0.8	0.02	0.38	0.85
Grasp	12	142	599	0.64	0.44	0.23	1	0.4	0	0.62	0.32	0.29	0.34

Del experimento se obtuvo que el algoritmo genético ganó 88 de 100 partidos contra Grasp, safistaciendo la hipótesis de que el genético generó una solución que brinda mejores resultados que Grasp en un tiempo similar.

5. Conclusión

Durante la elaboración de las consignas para el presente trabajo práctico, fue necesario reveer y adaptar metodologías de trabajo de etapas anteriores. Los casos con los que se trabajó en etapas anteriores estaban caracterizados por la elaboración de un código que fuera correcto y la correspondiente demostración de su correctitud junto con el análisis de su complejidad temporal y espacial. Sin embargo el trabajo con heurísticas parte de una base que, si bien sigue habiendo un código que se precie de ser correcto, ya no es posible conseguir un valor y saber que ese es el óptimo.

Los algoritmos heurísticos consiguen respuestas a problemas en tiempos relativamente accesibles cuando los espacios de soluciones que hay que explorar son demasiado grandes como para obtener esa respuesta de manera unívoca y polinomial. Es decir que la construcción de una solución tiene un caracter exploratorio dentro del espacio de soluciones. Dicha naturaleza de las heurísticas dio lugar a que gran parte de este trabajo fuera dedicado a la elaboración de experimentos que funcionaran como guías dentro del vasto espacio de candidatos. Si bien se sigue una guía teórica sobre cómo debe estructurarse tanto Grasp como el Genético, cada una de sus partes fueron construidas como porciones independientes. Resulta claro verlo en las distintas implementaciones de *crossover*, *selection*, y *fitness*. Dichas partes fueron puestas a prueba a lo largo de la elaboración del informe para refinar la idea acerca de la calidad de la respuesta obtenida.

Concluímos que dada la naturaleza del problema, la cual incluye elementos de tinte probabilístico, poder ganar un partido no es una propiedad transitiva de cada equipo. Es por esto que las heurísticas ayudarán a encontrar *buenas* soluciones pero no se puede esperar encontrar una *óptima*. Solo podrá considerarse la existencia de tales soluciones únicamente bajo algún criterio estadístico. El inabarcable tamaño del espacio de soluciones posibles hace necesario invertir tiempo en explorarlo independientemente de la heurística utilizada. De hecho, los últimos experimentos realizados, fueron aquellos en que se privilegió el tiempo para evaluar cuánto mejora la solución obtenida. Como trabajo a futuro se espera contar con más tiempo para explorar exhaustivamente el espacio de soluciones, experimentar con distintos criterios de corte, formas de evaluar las soluciones y sofisticar la función evaluadora parametrizable.

6. Estado del arte de los Algoritmos Genéticos

6.1. Definición

En las ciencias de la computación, los algoritmos genéticos son una familia de modelos computacionales inspirados en el proceso de selección natural donde se busca a partir de una población inicial ir obteniendo una nueva que se comporte mejor.

Cada algoritmo genético empieza con una población elegida, en la mayoría de los casos, de manera aleatoria. Esta población será conjunto de *individuos* con un grupo de propiedades a las que denominaremos *genes* o *cromosomas*. Luego, se les aplica una función que evalúa qué tan buenos son dichos cromosomas y se les da más oportunidades de *reproducirse* a los que obtengan un valor mayor (o menor si lo que interesan son las que den los números más chicos). Qué tan buena es una solución es dependiente del problema en cuestión.

Para obtener una buena función que evalúe la población lo recomendado es tener un conjunto de parámetros que optimicen el resultado. A veces no es posible tratar de forma independiente a cada variable, ya que puede haber interacciones entre ciertos parámetros cuyos efectos tienen que ser tenidos en cuenta. Esta interacción se la suele llamar *epistasis*.

Por otro lado, desarrollar dicha función suele involucrar realizar simulaciones para lograr dilucidar la mejor forma de optimizarla. También es importante que sea relativamente rápida para evaluar, ya que si, por ejemplo, evaluar una solución requiere 1 hora, tarda más de un año realizar 10 mil evaluaciones que serían aproximadamente 50 generaciones de una población de 200 strings. [1]

Es útil ver el proceso de ejecución de un algoritmo genético como un proceso de 2 partes. Primero selecciona de la población actual los más aptos para formar una población intermedia. Luego, procesos de mutación y recombinación (crossover en inglés) son aplicados en dicha población intermedia para obtener la siguiente población a evaluar. Este proceso de ir de la población actual a la siguiente constituye una generación en la ejecución del algoritmo genético.

El *crossover* se puede realizar de manera aleatoria de a pares de strings para formar 2 nuevos strings que se agregan a la siguiente población. Luego del crossover se puede aplicar el proceso de mutación con cierta probabilidad (normalmente baja) que modifica un string en alguno de sus valores.

Una vez ambos terminen, dicha población está lista para ser evaluada y volver a empezar el ciclo con esta nueva generación.

6.2. Áreas de aplicación

A continuación se detallarán algunas de las aplicaciones de esta estrategia para optimizar ciertos problemas.

- **Videojuegos** [2]:
  La idea para lograr que un videojuego sea entretenido viene de poder representar un desafío apropiado a los jugadores, un punto en el que no se sientan sobrepasados por la dificultad del juego ni aburridos por su facilidad.
  Esto se puede lograr permitiéndole al jugador elegir entre un conjunto de dificultades programadas, pero a medida que los jugadores avanzan, éstos van mejorando y aprendiendo como juega la AI. Tarde o temprano se terminan acostumbrando a reconocer los patrones programados hasta que ya no consiguen obtener un desafío por parte del juego.
  Una alternativa para éste problema es el uso de un algoritmo genético, permitiendo que una vez que el jugador se *adapte* a cómo funciona la AI, a partir de las operaciones convencionales de cualquier algoritmo genético se vayan optimizando las tácticas de juego de la AI y ya el juego no se basa en que tan rápido el jugador se acostumbre a como juega la AI sino a que tan rápido se puede adaptar a sus nuevas estrategias.
- **Robótica** [3]:
  La robótica es uno de los campos más importante de la industria computacional del momento y se utiliza

en numerosas industrias para incrementar su eficiencia y precisión, pero como el ambiente en el que los robots trabajan cambia con el tiempo se vuelve difícil programarlos para todos los posibles comportamientos para que el robot sobrelleve los cambios y se pueda adaptar a ellos.
Por este motivo se vuelve crucial el uso de algoritmos genéticos en la robótica, la idea detrás de esta técnica le permite al robot responder a los cambios readaptándose en base a calcular posibles escenarios que representen las distintas respuestas posibles y evaluar su performance. Luego de utilizar los operadores genéticos (crossover, mutation) buscamos obtener un conjunto de respuestas que mejoren las probabilidades de adaptación del robot.

- **Mercados Financieros** [4]:
  En el mercado de finanzas, los algoritmos genéticos son usualmente utilizados para encontrar la mejor combinación de parámetros en una regla de negociación financiera (trading rule) y así construir modelos para elegir las acciones que convienen comprar o vender.
  Como ejemplo de posibles parámetros a tener en cuenta al momento de modelar los cromosomas para este tipo de aplicación de los algoritmos genéticos pueden ser la media móvil de convergencia/divergencia (MACD), la media móvil exponencial (EMA) y las estocásticas. Otra cosa a tener en cuenta es el historial de las acciones en los últimos años para ayudar a optimizar los parámetros elegidos.

6.3. Limitaciones

Si bien los algoritmos genéticos son buenos para conseguir optimizaciones, tienen ciertas limitaciones. Por ejemplo [3]:

- La creación de la función de fitness tiene que ser lo suficientemente robusta como para poder soportar los cambios aleatorios de cada iteración.
- No alcanza solo con que la función sea robusta, si se elige una mala función también tendremos el problema de que haya casos favorables que se pierdan impidiendo así conseguir las mejores soluciones posibles.
- Si bien la función de fitness es importante, tampoco hay que descuidar los demás procesos como el crossover y las mutaciones, son ellos los que logran variar la población entre cada iteración para intentar obtener mejores resultados. Si fallan, podría no solo retrasar el tiempo hasta encontrar una solución eficiente, sino también lograr que nunca se llegue a una.
- Como las soluciones obtenidas son buenas en comparación a otras, tampoco podemos asegurar que lleguemos a las mejores posibles si no se varían bien las muestras entre cada iteración para obtener resultados nuevos y variados.
- Los algoritmos genéticos no funcionan muy bien para problemas de optimización donde los valores posibles de sus genes son solo valores de decisión (sí o no, verdadero o falso), en estos casos no hay forma de converger a una solución, una búsqueda aleatoria podría resultar más eficiente.

Referencias

[1] Darrell Whitley, *A Genetic Algorithm Tutorial*, http://www.cs.colostate.edu/~genitor/MiscPubs/tutorial.pdf

[2] Michael Martin, *Using a Genetic Algorithm to Create Adaptive Enemy AI*, gamasutra.com/blogs/MichaelMartin/20110830/90109/Using_a_Genetic_Algorithm_to_Create_Adaptive_Enemy_AI

[3] *Evolutionary Computation: Putting Nature to Work*, https://www.doc.ic.ac.uk/project/examples/2005/163/g0516312/Algorithms/Reality.html

[4] Justin Kuepper, *Using Genetic Algorithms to Forecast Financial Markets*, https://www.investopedia.com/articles/financial-theory/11/using-genetic-algorithms-forecast-financial-markets.asp