DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

RTP3: Heurísticas

Darwin DT en el mundial 2018

Algoritmos y Estructura de Datos III

Integrante	LU	Correo electrónico
Bogetti, Gianfranco	693/15	gianbogetti7@hotmail.com
Destuet, Carolina	753/12	carolinadestuet@gmail.com
Murga, Christian Mariano	982/12	christianmmurga@gmail.com
Perez, Lucía Belén	865/13	luu.-18@hotmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

Resumen

En el presente trabajo vamos a modelar una simplificación de un juego de fútbol y apoyándonos en este modelo computacional, desarrollaremos distintas estrategias de juego para un equipo dado basándonos en algoritmos genéticos.

Palabras clave: Heurísticas, algoritmos genéticos *Grid-Search*, *Grasp Fitness function*, *Crossover*, *Mutation*.

Índice

1. Introducción

En este trabajo se introduce un juego deportivo inspirado fuertemente en el deporte de fútbol, más precisamente, siendo una simplificación del mismo: tenemos dos equipos de tres jugadores cada uno que durante un partido compiten entre ellos con el objetivo de obtener más goles que el equipo rival, el tiempo de duración del partido se fija de antemano. Hay dos resultados posibles del partido, puede haber un equipo ganador en caso de que uno obtenga más cantidad de goles que el otro al finalizar el partido, o bien se puede dar una situación de empate si es que ambos hicieron el mismo número de goles.

Es de nuestro interés hallar estrategias de juego para que un equipo juegue bien pero son numerosas las posibles situaciones de juego que se pueden presentar en un partido a la vez que no se puede predecir cuáles son las jugadas del equipo contrario, ambas son componentes fundamentales para determinar una estrategia de juego. Puesto que por estos motivos es muy complejo e incluso imposible hallar una solución exacta al problema de "ganar un partido", nos vamos a apoyar en el uso de ciertas técnicas algorítmicas que se conocen por el nombre de "**heurísticas**" y nos permitan hallar soluciones aproximadas, las heurísticas tienen como finalidad producir soluciones que en general son aproximaciones a la solución exacta al problema subyacente y son de utilidad cuando no se puede extraer toda la información necesaria para hallar una solución (en nuestro problema, dicha falta de información proviene de no saber qué jugada hará el equipo rival) o cuando el tiempo que lleva hallar y verificar la solución es muy alto para fines prácticos.

Sin meternos en detalle en la explicación de cada regla del juego dado que son extensas y creemos que no aportan a la comprensión del objetivo del trabajo (para más información, ver referencia [1]), un aspecto de este trabajo es modelar computacionalmente el juego brevemente descripto. Luego, apoyándose en tal modelo, otra instancia posterior consiste en dar forma a una estrategia de juego que sirva globalmente para cada jugada de un partido y, como ya dijimos, nos basaremos en técnicas heurísticas para definir una estrategia posible.

2. Desarrollo del problema

En el modelo computacional del juego, los aspectos a tener en cuenta son, por un lado, las condiciones estáticas del juego: los dos equipos que juegan el partido y sus atributos (cada jugador de un equipo tiene una probabilidad de quite de pelota asociada), una cancha con dimensiones de ancho M y alto N y dos arcos centrados en cada uno de los extremos verticales de la cancha, otra variable a considerar es un número S que determina en cuántos pasos finaliza el partido; por otro lado, las condiciones dinámicas del juego: en cada paso $1 \leq i \leq S$, cada equipo deberá hacer una jugada a través de sus jugadores, cada uno de ellos podrá quedarse en su lugar, desplazarse a alguno de los casilleros lindantes al que ocupa en la configuración actual del partido, o bien, si el jugador tiene posesión de la pelota, el mismo podrá patearla, estos posibles movimientos (quedarse quieto, desplazarse o patear pelota) están sujetos a y condicionados por las reglas expuestas en el enunciado. Todas estas facetas del juego, características dadas de antemano al partido, movimientos permitidos, la acción en sí de jugar que tiene cada equipo, es lo que se debe modelar computacionalmente como primera instancia del trabajo práctico.

Para poder definir una estrategia, la idea es parametrizar propiedades de esta y definir una función puntuadora que se provea de tales parámetros para darle un valor específico a cada jugada posible que pueda hacer el equipo dado un instante particular del partido. La jugada que esté asociada con el valor máximo será la elegida como el próximo movimiento a jugar por tal equipo. Dado un equipo y una configuración dada de un partido, donde una configuración de un partido es exactamente la ubicación de los jugadores y la pelota en la cancha asociada a dicho partido para un instante $1 \leq i \leq S$, se quiere evaluar cuál es un posible próximo mejor movimiento para que realice el equipo. Para el propósito recién explicitado, con los parámetros definidos, se considera a la función puntuadora de una configuración del partido como

$$f : cancha \times equipo1 \times equipo2 \times configuraciónJuego \to \mathbb{R}_{>0}$$

que tendrá la siguiente forma:

$$f(c, e1, e2, cJ) = \sum_{i=1}^{n} p_1 1 v_1 + ... + p_n v_n$$

donde $v_1, ...v_n$ representan características particulares que se le pueden atribuir a un estado de juego desde el punto de vista de un equipo. Para ilustrar, v_i puede ser por ejemplo, la distancia mínima del equipo al arco del equipo contrario en dicho momento del partido. Para cada v_i, el valor p_i es un peso al valor y se puede entender como una traducción numérica de qué importancia se le da a cierto aspecto de una estrategia. En el ejemplo de v_i que dimos recién, si dicho p_i tiene un valor mucho más alto que p_j para todo $j \neq i$, entonces esto quiere decir que en toda jugada se le está dando una alta prioridad a estar cerca de la cancha del rival, es decir que la estrategia se basa en una táctica ofensiva.

A continuación enumeramos las distintas técnicas heurísticas que utilizaremos en este trabajo para hallar estrategias de juego junto con una breve explicación de cada una, más adelante ahondaremos más detalladamente con un subconjunto de dichas técnicas:

- **grid search**: dado que la función puntuadora tiene varios hiperparámetros (los pesos $p_1, ..., p_n$ que se quieren optimizar) y el espacio de soluciones está conformado por todas las posibles combinaciones de valores que pueda tomar cada uno de los hiperparámetros, grid search consiste en una configuración del espacio de soluciones que se basa en, si cada p_i puede tomar c_i valores distintos, tomar el producto cartesiano $\prod_{i=1}^{n} c_i$ y de esta manera el espacio de soluciones toma la forma de una grilla. Una vez que le damos esta configuración al posible espacio de soluciones, dos formas de recorrerlo y de hallar soluciones aproximadas al problema son:

  1) **búsqueda local**: es un método metaheurístico de optimización, donde metaheurístico significa que es una técnica que no se apoya en las particularidades del problema a resolver, que empieza por una solución candidata a un problema dado e iterativamente la mejora haciendo modificaciones locales de la misma. Esencialmente comienza por una solución inicial y busca en su vecindario por una solución mejor. Si la encuentra, reemplaza la solución actual por la nueva.

  2) **grasp**: es un método iterativo que se compone de dos partes, una fase de construcción y otra de búsqueda local. Se obtiene una solución factible durante la primera etapa aplicando una heurística golosa mientras que en la segunda etapa se hace un proceso de búsqueda local y se compara la solución actual con la obtenida en la búsqueda, de ser mejor la obtenida se reemplaza la actual por esta nueva solución.

- **algoritmos genéticos**: son algoritmos que usan principios inspirados en la teoría de la evolución natural de Darwin. Este algoritmo simula el proceso de selección y consecuente supervivencia de los individuos de la población más aptos y mejor adaptados a su entorno entre generaciones consecutivas. Cada generación consiste en una población de individuos, donde, en nuestro problema, cada individuo representa un punto en el espacio de búsqueda de soluciones y una posible solución. Luego, a los individuos de la población se los hace atravesar un proceso de evolución.Los algoritmos genéticos hacen uso de los siguientes principios:

  -individuos en una población compiten por recursos y parejas de apareamiento.

  -los individuos más exitosos en cada competencia van a producir más descendencia que aquellos que tuvieron una mala performance. Genes de los individuos más aptos se propagan por la población de manera que dos padres competitivos en ocasiones producirán un individuo que sea mejor que los dos padres. Luego, en conjunto, cada generación sucesiva estará más adaptada a su ambiente.

  Luego de que una población inicial sea seleccionada aleatoramiente, el algoritmo simula el proceso de evolución natural a través de los siguientes pasos:

  1) **selección** equivale a supervivencia de los más aptos, esta selección se hace a través de una función de *fitness* que intenta establecer un puntaje de performance de un individuo capturando así la capacidad de adaptabilidad de los individuos.

  2) **crossover** representa el apareamiento entre individuos

  3) **mutación** en algunos individuos de la descendencia, sus genes pueden estar sujetos a mutaciones con una probabilidad muy baja

3. Pseudocódigo de algoritmos usados

Algoritmo 1: puntuarEstado

input : estado de un tablero
ouput: int puntaje asociado a ese estado

```
1  puntaje ← lista de enteros de tamaño 15                                  // O(1)
2  puntaje[0] ← distanciaMínimaAlArcoContrario(estado)                      // O(1)
3  puntaje[1] ← distanciaTotalAlArcoContrario(estado)                       // O(1)
4  puntaje[2] ← distanciaMáximaAlArcoContrario(estado)                      // O(1)
5  puntaje[3] ← distanciaALaPelotaDeUnJugador(estado)                       // O(1)
6  puntaje[4] ← esGOL(estado)                                               // O(1)
7  puntaje[5] ← puedoHacerGOL(estado)                                       // O(1)
8  puntaje[6] ← interceptéPelota(estado)                                    // O(M)
9  puntaje[7] ← puedoInterceptarLaPelotaAMiArco(estado)                     // O(M)
10 puntaje[8] ← disputaPorLaPelota(estado)                                  // O(1)
11 puntaje[9] ← puedoIrAQuitarLaPelota(estado)                              // O(1)
12 puntaje[10] ← distanciaMínimaAMiArco(estado)                             // O(1)
13 puntaje[11] ← distanciaTotalAMiArco(estado)                              // O(1)
14 puntaje[12] ← distanciaMáximaAMiArco(estado)                             // O(1)
15 puntaje[13] ← hiceUnPase(estado)                                         // O(1)
16 puntaje[14] ← distanciaALosContrarios(estado)                            // O(1)
17
18 sumaPuntaje ← suma de todos los elementos de la lista puntaje            // O(1)
19
20 return sumaPuntaje
```

La complejidad temporal de la función **puntuarEstado** es $O(M)$.

Algoritmo 2: Búsqueda Local

```
   input: int cantidadMejoras, int cantidadPartidos, float salto
 1 mejorGenomaActual ← genomaInicialRandom
 2 iteracionesAlgoritmo ← 0
 3 fitnessActual ← 0
 4 mejoroGenoma ← false
 5 tamGenoma ← tam(mejorGenomaActual)
 6
 7 while iteracionesAlgoritmo ≤ cantidadMejoras do
 8     iteracionesAlgoritmo ← iteracionesAlgoritmo - 1
 9     vecindario ← crearVector(tamGenoma*2,mejorGenomaActual)
10     para vecindario[0...15), vecino-jesimo[j] = mejorGenomaActual[j] + salto
11     para vecindario[15...30), vecino-jesimo[j] = mejorGenomaActual[j] - salto
12
13     fitnessVecinos ← crearVector(tam(vecindario),0)
14     for k ← 0 to tam(fitnessVecinos) do
15         resultado ← 0
16         fitnessGenomaActual ← 0
17
18         for l ← 0 to cantidadPartidos do
19             jugarPartido(mejorGenomaActual, vecindario[k])
20             if ganoPartido mejorGenomaActual then
21                 fitnessGenomaActual ← fitnessGenomaActual + 1
22                 resultado ← resultado - 1
23             else
24                 if ganoPartido vecindario[k] then
25                     fitnessGenomaActual ← fitnessGenomaActual - 1
26                     resultado ← resultado + 1
27                 end
28             end
29         end
30         fitnessVecinos[k] ← resultado
31         if fitnessActual ≤ fitnessGenomaActual then
32             fitnessActual ← fitnessGenomaActual
33         end
34     end
35     mejoro ← false
36     if fitnessActual < maximoFitness(fitnessVecinos) then
37         mejorGenomaActual ← vecindario[indiceMaxFitness(fitnessVecinos)]
38         mejoro ← true
39     end
40 end
```

El algoritmo de búsqueda local toma dos parámetros de entrada, "cantidadMejoras", "cantidadPartidos" y *salto*. Salto repressenta la distacia a la que estan las soluciones vecinas que generamos. CantidadMejoras indica la terminación del ciclo iterativo de búsqueda. Se parte de una solución inicial, un genoma generado de manera pseudoaleatoria. Luego, en cada paso del ciclo iterativo, en torno a la solución temporal s se genera un vecindario de treinta vecinos. Dicho vecindario se crea modificando de a una componente del genoma, si el genoma es de la forma $s = (g_1, ..., g_n)$, entonces, se tiene en cuenta un número $a > 0$ tal que para cada g_i, se consideran la soluciones $(g_1, ..., g_i + a, ..., g_n)$ y $(g_1, ..., g_i - a, ..., g_n)$ y considerando a de manera de estar en el rango de los valores alcanzables por cada componente g_i.

Luego de generar el vecindario, se establece el siguiente criterio para tener una noción de máximo y desplazarnos hacia él en el entorno elegido: se hace jugar "cantidadPartidos" veces a cada vecino contra s, dándole un puntaje a cada vecino de acuerdo al resultado total de los partidos jugados contra s. Si existe algún vecino v que obtiene un puntaje mayor al de s, entonces v será el nuevo máximo obtenido y nos desplazaremos ahora a un vecindario de v para obtener una próxima posible mejor solución local hasta terminar el ciclo.

Se creó una clase de **C++** llamada "genético" con todas las funciones necesarias para el diseño del algoritmo genético. Mencionamos los miembros de esta clase para mejor entendimiento del pseudocódigo. Uno de los miembros indica la cantidad de individuos de la población (**cantidadPoblacion**), otro la cantidad de genes (**cantidadGenes**), otros dos para la parte de mutar genomas: porcentajePoblacionMutada (cuántos individuos de la población mutan) y porcentajeMutacion (cuánto muta de un genoma) y, por último un miembro generacion (indica qué número de generación es), y poblacion (un vector con los individuos-genomas de la población).

Algoritmo 3: algoritmo Genético

```
   input : int fitness, int seleccion, int crossover
   ouput: genoma Óptimo
 1 genomaOptimo ← genomaInicial()
 2 genetico.poblacion ← generarPoblacionRandom()
 3 genetico.generacion ← 0
 4 evaluoPoblacion(genetico.poblacion, fitness)
 5 while genetico.generacion ≠ genetico.MaxGeneraciones do
 6 |   if seleccion = 1 then
 7 |   |   genetico.reproduccionSelectiva(crossover)
 8 |   else
 9 |   |   genetico.reproduccionRandom(crossover)
10 |   end
11 |   genetico.mutacion()
12 |   genetico.evaluarPoblacion(fitness)
13 |   genomaOptimo ← genomaMaxPuntaje(genetico.poblacion)
14 end
15 return genomaOptimo
```

El algoritmo genético toma como parámetros fitness, selección y crossover puesto que para cada uno de ellos hay dos posibles variaciones. En líneas generales, el algoritmo sigue los pasos ya explicados anteriormente del algoritmo genético: se crea una población de individuos de manera pseudoaleatoria con la función rand() de la librería algorithm de C++ (para cada genoma de la población se definen las componentes con rand() en el rango de valores de la componente del genoma).

Se evalúa a los genomas de la población de acuerdo a la función de fitness pasada por parámetro. Luego se hace la parte iterativa del algoritmo, que es en donde se hacer evolucionar a la población. Aquí es donde, en cada iteración, se crea la nueva generación con la función reproducciónSelectiva o reproduccionRandom de acuerdo a cuál sea el método de selección pasado por parámetro. Luego se aplica una mutación a parte de la población y se la evalúa con la función de fitness. Al final del procedimiento iterativo se selecciona el mejor genoma de la última generación creada.

Algoritmo 4: Crossover 1

```
   input: genoma padre, genoma madre, genoma hijo1, genoma hijo2
 1 hastaPadre ← 0
 2 if padre.puntaje() > madre.puntaje() then
 3 |   hastaPadre ← 3/4 ∗cantidadGenes
 4 else
 5 |   if padre.puntaje() < madre.puntaje() then
 6 |   |   hastaPadre ← 1/4 ∗cantidadGenes
 7 |   else
 8 |   |   hastaPadre ← 1/2 ∗cantidadGenes
 9 |   end
10 end
11 hijo1[0...hastaPadre) ← padre[0...hastaPadre]
12 hijo1[hastaPadre...cantidadGenes) ← madre[hastaPadre...cantidadGenes)
13 hijo2[0...hastaPadre) ← madre[0...hastaPadre]
14 hijo2[hastaPadre...cantidadGenes) ← padre[hastaPadre...cantidadGenes)
```

Algoritmo 5: Crossover 2

```
   input: genoma padre, genoma madre, genoma hijo1, genoma hijo2
 1 for i ← 0 to genetico.cantidadGenes do
 2 |   genRandomHijo1 ← randomEntre(0,1)
 3 |   genRandomHijo2 ← randomEntre(0,1)
 4 |   if genRandomHijo1 = 0 then
 5 |   |   hijo1-iesimo ← padre-iesimo
 6 |   else
 7 |   |   hijo1-iesimo ← madre-iesimo
 8 |   end
 9 |   if genRandomHijo2 = 0 then
10 |   |   hijo2-iesimo ← padre-iesimo
11 |   else
12 |   |   hijo2-iesimo ← madre-iesimo
13 |   end
14 end
```

Algoritmo 6: Fitness 1

```
   input :
 1 for j ← 0 to cantidadPblacion do
 2 |   for i ← indice+1 to cantidadPblacion do
 3 |   |   resultado ← genoma.play(poblacion[i])
 4 |   |   if resultado = 3 then
 5 |   |   |   poblacion[j].incGanados(0)
 6 |   |   |   poblacion[i].incPerdidos(1)
 7 |   |   end
 8 |   |   if resultado = 1 then
 9 |   |   |   poblacion[j].incEmpatados(0)
10 |   |   |   poblacion[i].incEmpatados(1)
11 |   |   end
12 |   |   if resultado = 0 then
13 |   |   |   poblacion[j].incPerdidos(0)
14 |   |   |   poblacion[i].incGanados(1)
15 |   |   end
16 |   end
17 end
```

La función Fitness1 va a puntuar a la Población actual. Cada genoma va a jugar un partido con cada otro genoma dentro de la población, funciona de manera análoga al sistema "todos contra todos"de competencias deportivas.

Algoritmo 7: Fitness 2

```
   input : cantJugadas
 1 clasificados ← ∅
 2 for i ← 0 to cantidadPblacion do
 3 |   clasificados ← i
 4 end
 5 for i ← 1 to cantidadPblacion do
 6 |   it ← clasificados.IteratorBegin()
 7 |   contador ← 0
 8 |   while contador ≠ i do
 9 |   |   it2 ← it
10 |   |   it2 ← it2 + 1
11 |   |   itGanados ← 0
12 |   |   it2Ganados ← 0
13 |   |   for j ← 0 to cantJugadas do
14 |   |   |   resultado ← genoma.play(poblacion[i])
15 |   |   |   if resultado = 3 then
16 |   |   |   |   poblacion[(*it)].incGanados(0)
17 |   |   |   |   poblacion[(*it2)].incPerdidos(1)
18 |   |   |   end
19 |   |   |   if resultado = 1 then
20 |   |   |   |   poblacion[(*it)].incEmpatados(0)
21 |   |   |   |   poblacion[(*it2)].incEmpatados(1)
22 |   |   |   end
23 |   |   |   if resultado = 0 then
24 |   |   |   |   poblacion[(*it)].incPerdidos(0)
25 |   |   |   |   poblacion[(*it2)].incGanados(1)
26 |   |   |   end
27 |   |   end
28 |   |   itAux ← it+2
29 |   |   if itGanados < it2Ganados then
30 |   |   |   clasificados[*it] ← erase
31 |   |   end
32 |   |   if itGanados > it2Ganados then
33 |   |   |   clasificados[*it2] ← erase
34 |   |   end
35 |   |   if itGanados == it2Ganados then
36 |   |   |   r ← Randomit, it2
37 |   |   |   clasificados[*r] ← erase
38 |   |   end
39 |   |   it ← itAux
40 |   |   contador ← contador + 2
41 |   end
42 end
```

Esta forma de evaluación consiste en un torneo donde el puntaje de un individuo es la suma de los puntajes en el total de equipos jugados. El sistema es análogo al de eliminación directa en competencias deportivas: el perdedor de un encuentro es inmediatamente eliminado. Se van jugando rondas y en cada una de ellas se elimina exactamente a la mitad de los individuos hasta dejar un único competidor.

Algoritmo 8: Método de selección 1: Reproducción Selectiva

```
   input: int crossover
 1 poblacionNueva ← vectorVacio
 2 sortPorFitness(genetico.poblacion)
 3 for l ← 0 to cantidadPoblacion*1/4 do
 4 |   poblacionNueva-iesimo ← genetico.poblacion-iesimo
 5 end
 6 for l ← 0 to cantidadPoblacion*1/4 do
 7 |   hijo1,hijo2,hijo3 ← vectorVacio
 8 |   madre ← numeroPseudoAleatorio(1,cantidadPoblacion*3/4)
 9 |   padre ← numeroPseudoAleatorio(1,cantidadPoblacion*3/4)
10 |   if crossover = 1 then
11 |   |   genetico.crossover1(madre,padre,hijo1,hijo2)
12 |   else
13 |   |   genetico.crossover2(madre,padre,hijo1,hijo2)
14 |   end
15 |   agregar(poblacionNueva,hijo1)
16 |   agregar(poblacionNueva,hijo2)
17 |   madre ← numeroPseudoAleatorio(1,cantidadPoblacion*3/4)
18 |   padre ← numeroPseudoAleatorio(1,cantidadPoblacion*3/4)
19 |   if crossover = 1 then
20 |   |   genetico.crossover1(madre,padre,hijo3,hijo2)
21 |   else
22 |   |   genetico.crossover2(madre,padre,hijo3,hijo2)
23 |   end
24 |   agregar(poblacionNueva,hijo3)
25 end
26 genetico.poblacion ← poblacionNueva
```

Algoritmo 9: Método de selección 2: Reproducción Random

```
   input: int crossover
 1 poblacionNueva ← vectorVacio
 2 for i ← 0 to cantidadPoblacion*1/2 do
 3 |   hijo1,hijo2 ← vectorVacio
 4 |   madre ← numeroPseudoAleatorio(1,cantidadPoblacion)
 5 |   padre ← numeroPseudoAleatorio(1,cantidadPoblacion)
 6 |   if crossover = 1 then
 7 |   |   genetico.crossover1(madre,padre,hijo1,hijo2)
 8 |   else
 9 |   |   genetico.crossover2(madre,padre,hijo1,hijo2)
10 |   end
11 |   agregar(poblacionNueva,hijo1)
12 |   agregar(poblacionNueva,hijo2)
13 end
14 genetico.poblacion ← poblacionNueva
```

Algoritmo 10: Mutación

```
 1 for i ← 0 to genetico.cantidadPoblacion do
 2 |   mutacion ← numeroPseudoAleatorio(1,100)
 3 |   if mutacion < genetico.porcentajePoblacionMutada then
 4 |   |   genetico.poblacion-iesimo ← genetico.mutar(poblacion-iesimo)
 5 |   end
 6 end
```

La función mutar utilizada en el algoritmo de mutación toma un genoma y a partir del miembro porcentajeMutacion, elige pseudoaleatoriamente (función rand() de C++) la cantidad de genes que represente ese porcentaje del total y los muta.

4. Algoritmos genéticos

4.1. ¿Qué son los algoritmos genéticos?

Los algoritmos genéticos son algoritmos de búsqueda para problemas computacionales de optimización que están íntimamente inspirados en el mecanismo de selección natural, un proceso biológico en el que los individuos más fuertes tienen más probabilidad de ser los ganadores en un entorno de competitivo. En este punto, los algoritmos genéticos usan una directa analogía de este proceso de evolución natural. Existen algunos puntos de este proceso respaldados por una fuerte evidencia experimental:

- La evolución es un proceso que opera sobre cromosomas más que sobre organismos. Aquellos son herramientas que codifican la estructura de un ser vivo.
- La selección natural es el mecanismo que relaciona los cromosomas con la eficiencia de la entidad a la que representan, permitiendo así que los organismos que están mejor adaptados al ambiente sean los que se reproduzcan más frecuente que aquellos que no.
- El proceso de evolución toma lugar durante la etapa de reproducción. Existen diversos mecanismos de reproducción en la naturaleza. Los más comunes son la mutación (que causan que los cromosomas de los descendientes sean distintos a los de los progenitores) y recombinación de los cromosomas de los padres para producir el descendiente.

Un algoritmo genético es un proceso iterativo que opera sobre un conjunto de individuos (la población). Cada individuo representa una solución potencial al problema que quiere ser resuelto. Inicialmente la población está aleatoriamente generada. A todo individuo de la población le es asignado, a través de una función de fitness, una medida de cuan bueno es respecto al problema bajo consideración. Este valor se usa para establecer un ranking entre los individuos dependiendo de su relativa idoneidad para el problema considerado y es la información cuantitativa que el algoritmo usa como única guía de búsqueda.

Cada individuo está codificado como una lista de componentes o variables de longitud finita en términos de algún alfabeto. Para continuar con la analogía genética, estos individuos están asociados a cromosomas y las variables son análogas a los genes. Luego, cada cromosoma (solución) está compuesta de varios genes (variables). El algoritmo genético usa una reproducción selectiva de las soluciones para producir descendencia mejor que los padres mediante la combinación de sus cromosomas.

A medida que los progenitores se reproducen y generan descendencia, debe hacerse espacio para estos nuevos individuos dado que la población siempre se mantiene con un tamaño estático. Individuos en la población son descartados y reemplazados por estas nuevas soluciones, eventualmente creando una nueva generación una vez que todas las posibilidades de reproducción de la vieja población han sido agotadas. Nuevas generaciones de soluciones son producidas conteniendo, en promedio, más buenos genes que una solución típica de una generación previa. Eventualmente, una vez que la población ha convergido, en el sentido de que no está produciendo descendencia notoriamente distinta a aquella de generaciones previas, el algoritmo en sí se dice que converge al conjunto de soluciones del problema en mano.

4.2. Componentes, estructura y detalles de implementación

Luego de que una población inicial se genera aleatoriamente, el algoritmo evoluciona a través de los siguientes tres operadores:

1) selección que equivale a supervivencia de los más aptos (se determina esto por la función de fitness);

2) crossover que representa el apareamiento entre individuos;

3) mutación que introduce modificaciones random.

1. Operador de **selección**

idea clave: dar preferencia a los mejores individuos, permitiéndoles pasar sus genes a la próxima generación.

2. Operador de **crossover** Se eligen dos individuos de la población a través del operador de selección.

Se elige una ubicación en de la cadena del cromosoma para realizar el crossover.

Los valores de los dos cromosomas se intercambian hasta este punto.

Si Cromosoma1=000000 y Cromosoma2=111111 y el punto de crossover es 2, entonces la descendencia será Descendencia1=110000 y Descendencia2=001111.

Los dos nuevos individuos creados de este apareamiento son incluidos en la próxima población.

Combinando porciones de individuos buenos de una población, es probable que este proceso de lugar a individuos mejorados con respecto a su ascendencia.

3. Operador de **mutación** Con una baja probabilidad, una porción de los individuos tendrá parte de sus genes alterados.

El propósito de esto es mantener la diversidad dentro de la población e inhibir una convergencia prematura a una solución subóptima.

4.3. Breve historia

Los principios básicos de la teoría de algoritmos genéticos fueron desarrollados por Holland, sus colegas y estudiantes de la universidad de Michigan [2]. Sus investigaciones en torno a esta teoría tuvieron una doble finalidad: 1) abstraer y poder explicar rigurosamente los procesos adaptativos de sistemas naturales, y 2) diseñar sistemas artificiales de software que capturen los mecanismos esenciales de los sistemas naturales para poder hallar soluciones óptimas a problemas computacionales de optimización.

El área de investigación de algoritmos genéticos rápidamente avanzó en las décadas de los 70 y 80. Parte de este avance se debe al progreso tecnológico de la época pero también debido a que los científicos dedicados a la investigación en computación se empezaron a dar cuenta de las limitaciones de la programación convencional y los métodos de optimización tradicionales para resolver problemas complejos. Investigadores descubrieron que los algoritmos genéticos eran una forma de hallar soluciones a problemas que otros métodos no podían resolver. Los algoritmos genéticos pueden simultáneamente testear muchos puntos de todo el espacio de solución, optimizar con parámetros discretos o continuos, proveer de varios parámetros óptimos en vez de una única solución y trabajar con diferente tipo de data [3].

4.4. Comparación con otros algoritmos de optimización

Cuando hablamos de métodos tradicionales de optimización, nos referimos principalmente a tres tipos: basados en análisis matemático, búsqueda exhaustiva y random [4].

Los métodos de optimización basados en cálculo se dividen en dos categorías: método directo e indirecto. El directo "salta" a la función objetivo y sigue la dirección del gradiente hacia un máximo o mínimo local, también se conoce como el método "hill-climbing". El método indirecto toma el gradiente de la función objetivo, lo iguala a cero y resuelve el conjunto de ecuaciones que se desprenden de tal igualdad. Estos dos métodos acarrean algunos problemas. En primer lugar, solamente buscan óptimos locales, lo cual los hace inservibles si se desconoce el vecindario del óptimo global o si hay otros óptimos locales cerca de él. En segundo lugar, estos métodos requieren la existencia de derivadas, y esto virtualmente nunca es el caso en aplicaciones prácticas.

La algoritmos de búsqueda exhaustiva realizan exactamente, como lo indica su nombre, una búsqueda completa, por todo el espacio de soluciones. Estos algoritmos requieren un espacio de búsqueda finito o discretizado al menos, y testean cada valor del espacio, uno por vez, para hallar el máximo o mínimo. Mientras que este método es simple, es el menos eficiente de todos los algoritmos de optimización. En problemas prácticos, los espacios de búsqueda pueden ser demasiado vastos para probar cada posibilidad.

Los algoritmos de búsqueda aleatorios aleatoriamente elige una muestra representativa del espacio de búsqueda y encuentra el valor óptimo de dicha muestra. Al usar este algoritmo dejamos librado al azar que estemos cerca de la solución óptima o muy lejos de ella.

Los algoritmos genéticos tienen muchas ventajas por sobre estos métodos tradicionales. A diferencia de los métodos basados en análisis matemático, los algoritmos genéticos van progresando partiendo de una población de soluciones candidatas en vez de un valor inicial. Esto reduce significativamente la probabilidad de hallar un óptimo local en vez de un óptimo global. Los algoritmos genéticos no requieren información extra, como derivadas, que no está relacionada con los valores de las soluciones posibles. El único mecanismo que guía su búsqueda es el valor numérico de la función de fitness de las soluciones candidatas basando esta guía en la definición que le da el que crea la función de fitness [5]. Por lo que acabamos de decir, los algoritmos genéticos son paralelos. La mayoría de los otros algoritmos sólo pueden explorar el espacio de soluciones hacia una solución en una dirección al mismo tiempo, y si la solución que descubren resulta subóptima, se debe abandonar todo el trabajo hecho y empezar de nuevo. En contraste, dado que los algoritmos genéticos tienen descendencia múltiple, pueden explorar el espacio de soluciones en múltiples direcciones a la vez.

Por último, una característica de los algoritmos genéticos que, a primera vista, parecería implicar una desventaja, resulta ser un beneficio: los algoritmos genéticos no poseen información intrínseca de los problemas que deben resolver. No utilizan información específica del problema, conocida de antemano, para guiar cada paso y realizar cambios en pos del mejoramiento; realizan cambios aleatorios en sus soluciones candidatas y posteriormente usan la función de fitness para determinar si esos cambios producen una mejora.

Como las decisiones de estos algoritmos están basadas en la aleatoriedad, los algoritmos genéticos consideran todos los caminos de búsqueda posibles; en contraste, cualquier estrategia de resolución de problemas que dependa de un conocimiento previo, debe inevitablemente comenzar descartando muchos caminos a priori, perdiendo así cualquier solución novedosa que pueda existir [6].

4.5. Aplicaciones

A continuación damos ejemplos de aplicaciones de algoritmos genéticos por fuera del contexto exclusivamente académico, es decir, aplicaciones que tengan un impacto más concreto y directo en el mundo fuera de la academia.

En el paper "Genetic algorithms and their applications in environmental sciences" de Sue Ellen Haupt y Randy Haupt, se enumeran muchísimos ejemplos de aplicaciones en el área de las ciencias ambientales. Muchos problemas en los cuales se usan los genéticos como herramienta involucran asociar algún tipo de modelo a datos observados por experimentos. El objetivo para esta modelización es hallar parámetros que optimicen la compatibilidad entre el modelo teórico posible y los datos observados. Un ejemplo de ajustar un modelo a los datos observados usando algoritmos genéticos es reportado por Mulligan y Brown (1998). Usan un algoritmo genético para calibrar un modelo de calidad de agua. Usaron regresión no lineal para hallar parámetros que minimizan el error de cuadrados mínimos entre el modelo teórico más ajustable y la data. Hallaron que el algoritmo genético funciona mejor que las técnicas tradicionales y además, notaron que aquel proveía información sobre el espacio de búsqueda, permitiéndoles desarrollar correlaciones entre los parámetros.

Entre otros trabajos relacionados a determinar la calidad de agua se incluyen el uso de algoritmos genéticos para determinar el camino del flujo del agua (Molian y Loucks 1995) y dimensionar la distribución de redes de agua (Simpson, et al. 1994).

Para manejar los suministros de aguas subterráneas también se halló útil el uso de algoritmos genéticos. Peralta y otros colaboradores han combinado algoritmos genéticos con redes neuronales y simularon distintas técnicas de tratamiento térmico para combinar las ventajas de cada una. Aly y Peralta (1999) usaron algoritmos genéticos para ajustar parámetros a un modelo de optimización de locaciones de bombeo para el tratamiento de aguas subterráneas.

Otro ejemplo de una aplicación exitosa de algoritmos genéticos es en la clasificación y predicción de ocurrencias de días lluviosos por Sen y Oztopal (2001). Usaron uno de tales algoritmos para estimar parámetros en una cadena de Markov.

Un ejemplo del mundo de la geofísica es determinar el tipo de capas de roca subterránea. Como no es práctico tomar muestras de suficiente resolución para crear mapas suficientemente buenos de las capas subterráneas, técnicas modernas hacen uso de información sísmica o aplican una corriente y miden el potencial de diferencia que genera una resistencia. Estos diversos métodos producen modelos subdeterminados de la Tierra. Ajustar parámetros que matcheen con los data es un proceso altamente no lineal. El uso de algoritmos genéticos ha sido exitoso para hallar soluciones reales para este problema [7].

4.6. Limitaciones

A pesar de que, como ya pudimos ilustrar, los algoritmos genéticos han demostrado ser rápidos y poderosos para la resolución de algunos problemas de optimización, también conllevan algunas limitaciones:

- Un obstáculo mayor de los algoritmos genéticos es la elección de una función de fitness (evaluación) para que se vayan generando mejores soluciones del problema en tratamiento. Una elección errónea de la función de fitness puede conducir a problemas críticos como no poder hallar una solución, o peor todavía, devolver una solución equivocada.

- Junto con una elección apropiada de fitness, los otros parámetros del algoritmo genético (tamaño de población, la proporción de mutación y crossover) también deben ser elegidos con cuidado. Por ejemplo, un tamaño de población muy pequeño no le dará oportunidad al algoritmo de recorrer un espacio de búsqueda lo suficientemente amplio para encontrar buenas soluciones.

- Convergencia prematura del algoritmo es otro factor a tener en cuenta. Puede pasar que un individuo sea mucho más apto que sus competidores. Luego este individuo quizás reproduzca mucha descendencia reduciendo así drásticamente la diversidad de la población y llevando al algoritmo a converger a un máximo local que representa ese individuo particular en vez de hacer una búsqueda más cercana al máximo global.

5. Experimentación

Nuestro principal objetivo a la hora de experimentar va a ser encontrar **buenos equipos** con las heurísticas que tenemos (Grasp y Genético). Cuando hablamos de *equipo* nos referimos a un genoma (pesos de la función puntuadora, que en sí representa a nuestro equipo) y al decir *buenos equipos* vamos a considerar ésto como equipos que puedan desarrollarse en una cancha dada y mantener una coherencia de juego con el equipo contrincante (no tener un equipo estático, moverse con la pelota hacia el arco contrario, hacer goles, perseguir al equipo contrario si éstos tienen la pelota, etc). También vamos a querer comparar dichas heurísticas y ver las ventajas y desventajas de cada una de ellas. Intentaremos contestar algunas de las siguientes preguntas: ¿es mejor la heurística del Algoritmo Genético que la de Grasp? ¿siempre es así? De los equipos resultantes por la heurística de Grasp, ¿cuál es el mejor? y si obtengo uno que era el mejor de los equipos resultantes, ¿es un buen equipo?

A su vez, analizaremos por separado las dos heurísticas tratadas en este trabajo.

Consideraciones importantes a tener en cuenta:

- "buen equipo" : no hay un comparador externo como para medir el rendimiento de nuestro equipo o la calidad de nuestro genoma, por ende decidimos que un buen equipo es relativo a la forma de entrenarse que tuvo. Por ejemplo si obtenemos un equipo (genoma) con el algoritmo Genético, ese equipo va a ser bueno en comparación a los equipos restantes de la última población resultante del algoritmo.
- Vamos a trabajar con un tamaño de cancha fijo (10x5) dado que aumentar las proporciones de cancha nos va a dar muchas más posibilidades de movimientos, pases, etc, que hace que nuestros costos temporales aumenten considerablemente, y por cuestiones de tiempo no las consideramos. Pero creemos que en caso de variar los tamaños de la cancha, los equipos que obtuvimos para nuestra cancha original también tendrían un comportamiento bueno en estas nuevas canchas ya que las estrategas de juego se mantienen.
- La cantidad de posibles movimientos dado un estado actual es muy alta, 729 (9*9*9) si no tenemos la pelota y mas de 17000 si algunos de nuestros jugadores tiene la posesión del balón. Debido al alto costo computacional asociado con la cantidad de posibles movimientos, decidimos restringirlos notablemente de la siguiente manera. Solo se va a considerar patear la pelota hacia adelante (3 direcciones posibles contando las diagonales) y con 3 niveles de fuerza (1, 2 y 4).
- Al momento de decidir el próximo movimiento de nuestros equipos no tenemos en cuenta el posible movimiento de los equipos contrarios, es decir, los suponemos estáticos; excepto para situaciones particulares, por ejemplo donde nuestro equipo quiere patear al arco y alguno de los contrarios puede llegar a interceptar la pelota. Esto lo hacemos por que considerar los movimientos del adversario representa un coste computacional muy grande y dado el limitado tiempo con el que disponemos decidimos evitarlo.
- Consideramos valores de pesos positivos y negativos en nuestra puntuadora para que los algoritmos puedan recompensar o penalizar determinadas estrategias de juego. Un peso positivo recompensa aumentando el valor de una puntuación dada, mientras que uno negativo lo penaliza reduciéndolo.
- Creamos un equipo "defensivo" para enfrentarlo contra distintos equipos. Lo llamamos de dicha manera porque intentamos modelar el comportamiento de un equipo que defiende, es decir, tratamos de que los jugadores se mantengan cerca de su arco (puntuamos con mayor peso éstas características) y les dimos altas probabilidades de quite de la pelota.
- A su vez, creamos un equipo "atacante" para también enfrentarlo contra diversos equipos. En grandes rasgos queremos que dicho equipo se concentre en llegar con la pelota al arco contrario e intentar hacer la mayor cantidad de goles posible. Para esto, generamos un genoma que valora más las características de hacer gol y acercarse al arco contrario.
- Vamos a usar el sistema de puntuaciones del Fútbol usado por la FIFA para decidir cuantos puntos se asignan luego de jugar un partido. Este es: Ganar un partido da 3 puntos, empatar da 1 perder da 0.
- Todos los experimentos se ejecutaron en las computadoras de los laboratorios del Departamento de Computación.

5.1. Grasp

Como dijimos anteriormente Grasp se divide en 2 partes, una de construcción de una solución inicial y la segunda de búsqueda local para mejorar la solución previamente construida.
Como a priori no sabemos como caracterizar a una buena solución, construir una buena solución inicial es muy complicado por lo que vamos a concentrar la experimentación en la parte de la búsqueda local.

5.1.1. Solución Inicial

Como solución inicial decidimos usar 15 pesos aleatorios entre -100 y 100 los valores que usa la función puntuadora y 3 valores aleatorios entre 0 y 100 que son las probabilidades de quite de cada jugador de la solución.

5.1.2. Vecindario

Consideramos parte de nuestro vecindario cualquier solución que solo difiera de nuestra solución actual en 1 peso en un valor que llamamos salto. Consideramos 3 tamaños de saltos diferentes para la creación de los vecindarios:

- Saltos de 10 unidades
- Saltos de 20 unidades
- Saltos de 40 unidades

Consideraremos a un vecino mejor que nuestra solución actual si luego de jugar 5 partidos entre ellos el vecino obtuvo 66 % de los puntos en juego (10 de los 15 puntos en juego). Un ejemplo de esto sería si el vecino gana 3 partidos, empata 1 y pierde 1.

5.1.3. Cantidad de Mejoras y punto de corte

Una iteración de la búsqueda local llamaremos a una mejora en nuestra solución actual (un vecino mejor que la solución actual se convierte en la nueva solución actual). A medida que comparamos nuestra solución actual con sus vecinos nos quedamos con el primero que consideremos mejor(sin comparar con el resto del vecindario no recorrido hasta el momento). Decidir cuando una solución es Buena es difícil ya que no hay manera de asegurar esto sin hacerla competir contra otras soluciones, algo que lleva consigo un alto costo temporal. Por lo tanto decidimos terminar la búsqueda local luego de 200 iteraciones (mejoras en la búsqueda local) o en el momento que no haya solución vecina que sea mejor que la actual.

5.1.4. Problemas en la Experimentación

Daremos una breve explicación de los principales problemas que surgieron durante la experimentación con Grasp.

- Randomización de los Pesos: Elegir los pesos totalmente aleatorios nos proporciono equipos “muy malos” que no lograban hacer puntos bajo ninguna circunstancia, evitando que la búsqueda local pueda mejorar la solución ya que compiten 2 equipos muy parecidos. Para evitar estos casos elegimos al equipo inicial con pesos positivos ya que obtuvimos así mejores resultados. Si permitimos que los vecinos tengan valores negativos en los pesos.
- Convergencia de Soluciones: En algunos experimentos notamos que la búsqueda local se “estanca” entre un número chico de soluciones evitando converger a un mejor Máximo Local. Imaginemos las siguientes soluciones A, B y C, A es peor que B, B es peor que C y C es peor que A. La búsqueda local se puede estancar en círculos entre estas 3 soluciones de forma permanente. $A \to B \to C \to A$.

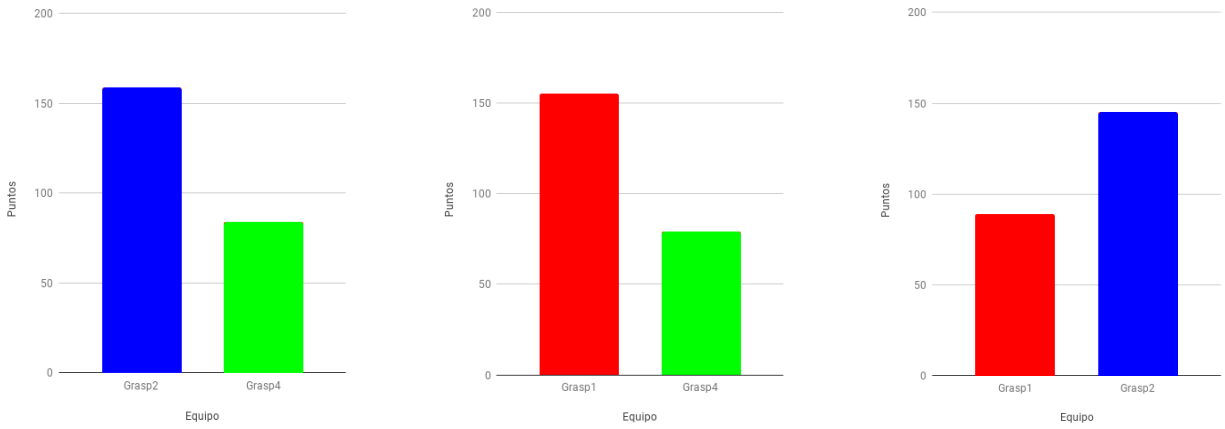

Figura 1: Competencias comparando diferentes métodos Grasp

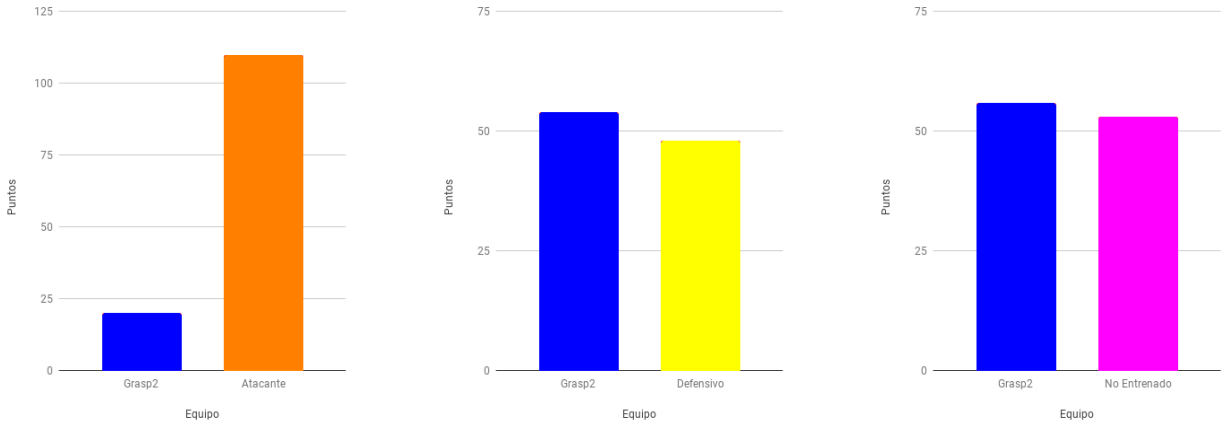

Figura 2: Competencias comparando Grasp2 contra equipos prefijados

5.1.5. Resultados

Luego de Generar 100 Soluciones con cada una de las 3 variables de Grasp que usamos nos quedamos con las 10 mejores de cada tipo y las hicimos competir entre ellas para analizar cual metodo nos produjo mejores equipos.
Para esto jugamos 10 partidos entre cada una de las soluciones de 1 variable contra cada una de las otras. Como podemos observar en los siguientes gráficos (1) el método grasp2 nos produjo equipos que al ponerlos a competir contra las otras 2 opciones son mejores ya que obtuvieron mejores puntajes que grasp1 y grasp4.
Sin embargo nos gustaría medir realmente que tan buenos son estos equipos contra adversarios diferentes, para esto los hicimos competir contra 3 equipos fijos, uno atacante, uno defensivo y uno sin entrenar.
Los resultados se observan en (2). Podemos ver que aunque nuestra solución le gana a las de Grasp no obtiene buenos resultados jugando contra las soluciones predefinidas lo que nos indica que probablemente no sea una "buena solución".

5.1.6. Mejoras en Futuros Experimentos

- Generar "buenas soluciones" para usar de base en la búsqueda local: partir la búsqueda local de soluciones pre-generadas con estrategias en mente (ejemplo: equipo ofensivo, defensivo).
- Convergencia de Soluciones: Evitar que se repitan soluciones en la búsqueda local.
- Modificación del Vecindario: Achicar y agrandar la diferencia entre la solución actual y sus vecinos según que tan cerca estemos de converger a una solución óptima.
- Búsqueda Local: Probar otras variaciones del método como por ejemplo moverse al mejor de todos los vecinos, no al primero que gane el 66 % de puntos.

5.2. Genético

Los tres grandes componentes que caracterizan al algoritmo genético son el fitness, el método de selección para la evolución entre generación y generación, y el crossover, es por ello que tomamos bajo consideración experimentos aislados en donde dejamos cada uno de estos parámetros variable y fijos los otros analizando así cuál nos brindaría mejores soluciones si es que se da tal hecho.

Otro de nuestros parámetros de interés para variar en la experimentación es el tamaño de las poblaciones del algoritmo genético. Por un lado, una población numerosa tiene el beneficio de aportar variedad genética pero los tiempos de ejecución son muy altos. Por otro lado, una población chica, a pesar de no tener la misma variedad genética, es mucho más eficiente en tiempo pudiendo dar muchas más generaciones en la misma cantidad de tiempo que una población de tamaño grande.

Un componente del genético que todavía queda por mencionar es la mutación, es decir, la probabilidad de que el material genético de un individuo de la población mute. Como explicamos anteriormente en este trabajo, su objetivo es mantener e introducir diversidad genética en la población y en general, la probabilidad de que el gen de un genoma mute es baja, pues si no nos estaríamos acercando a una búsqueda totalmente random en el espacio de soluciones. Teniendo esto en cuenta, consideramos de interés ver qué relación hay entre el porcentaje de mutación de una población de acuerdo al tamaño de esta población.

Por último, como nuestro problema original tiene como objetivo generar un jugador "bueno" en ciertos contextos del juego expuesto en el problema, consideramos un último experimento que es contrastar todas las mejores soluciones obtenidas a través de los distintos posibles métodos del algoritmo genético.

Antes de comenzar con la descripción de la experimentación es importante hacer las siguientes aclaraciones con respecto a ella: en lo que se basó en analizar puntualmente una componente del algoritmo genético (por ejemplo, comparar los crossover entre sí), para aislar dicha componente y que los resultados **no** dependan de las otras partes que conforman al algoritmo, dejamos dichos parámetros fijos; ahora bien, como eran varias las combinaciones de dichos parámetros y por no disponer del tiempo en la instancia de este trabajo, para cada análisis de una componente aislada decidimos dejar una determinada combinación fija de los parámetros restantes, en la parte del experimento pernitente al análisis de dicha componente se aclara la elección de la combinación particular de los restantes parámetros.

Una última clarificación por hacer es que para los experimentos en dónde se decidió dejar fijos los parámetros de selección y/o crossover para poder analizar qué implicancias tenía variar cierto parámetro específico de genético, se tomó la decisión de elegir fijar selección con el método 1 y crossover también con su versión 1 por el siguiente motivo: tanto crossover 1 como selección 1 apuntan a intentar generar la mejor solución con cierto criterio específico, en contraste, crossover 2 y selección 2 son métodos que cruzan progenitores y seleccionan población de manera muy aleatoria, con lo cual los primeros permiten tener un mayor control de las soluciones que se van generando y esto conduce a poder analizar más aisladamente el parámetro que se quiere variar y analizar en un concreto experimento sin agregar ruido o resultados que están directamente relacionados con lo azaroso de selección 2 y crossover 2.

5.2.1. Fitness 1

En todos los análisis de un parámetro en particular del genético dejando al resto fijo, una vez conseguidos los mejores equipos para cada posible valor del parámetro (ejemplo, para crossover 1 y para crossover 2 y el resto de las componentes fijas), vamos a comparar cuán buenos son, es decir, los haremos jugar entre ellos, frente a un rival "random", un rival que "ataca"y un rival que "defiende", todo esto una cierta cantidad de veces.

- **Crossover1 VS Crossover2**

Para analizar crossover, decidimos dejar fijos fitness en 1, y selección en 1, dado que fitness 1 hace una evaluación más completa de los individuos que fitness 2, y selección 1 es un método de selección menos aleatorio que el método de selección 2. Además, para esta combinación particular del genético y variando crossover entre 1 y 2, consideramos dos poblaciones de 12 y 16 individuos con 10 generaciones cada una y un porcentaje de mutación del 10 %. Lo que quisimos evaluar haciendo esto es el comportamiento de las distintas formas de hacer *crossover*. Para lograr este cometido, corrimos el algoritmo Genético con las funciones de crossover 1 y crossover 2.

La tabla a continuación (1) se realizó con los máximos obtenidos de la última generación.

Consideremos primero los dos equipos "máximos" resultantes de la población con 12 individuos, es decir A y B. Los hicimos jugar 16 partidos entre ellos, todos en una cancha de dimensiones 10x5 y 50 steps de duración del partido, alternando los lados de la cancha donde juega cada uno y alternando quién comienza primero, es decir

Nombre	Crossover	#Población	#Generacion	Jugados	Ganados	Perdidos	Empatados	Puntaje
A	1	12	10	11	11	0	0	33
B	2	12	10	11	8	0	3	27
C	1	16	10	15	15	0	0	45
D	2	16	10	15	8	0	7	31

Tabla 1: Crossover1 VS Crossover2

4 partidos el equipo A estuvo del lado derecho de la cancha y empezaron ellos teniendo la pelota, 4 partidos el equipo B estuvo del lado derecho y empezaron ellos con la pelota y otros 8 partidos donde se ven reflejadas las posibilidades restantes. A continuación se muestran los resultados obtenidos:

Equipo	Jugados	Ganados	Perdidos	Empatados	Puntaje
A	16	4	0	12	24
B	16	0	4	12	12

Tabla 2: Equipo A VS Equipo B (población 12)

Resultados obtenidos enfrentando a los equipos A y B contra un equipo random, teniendo en cuenta las consideraciones anteriores:

Equipo	Jugados	Ganados	Perdidos	Empatados	Puntaje
A	16	3	0	13	22
B	16	0	1	15	15

Tabla 3: Equipo A y B VS Equipo Random

Repetimos exactamente el mismo experimento teniendo las mismas consideraciones que para A y B, para la población de 16 individuos obteniendo los resultados reflejados en las tablas de hacer jugar a C Y D enfrentados, contra un equipo random, defensivo y atacante respectivamente.

Equipo	Jugados	Ganados	Perdidos	Empatados	Puntaje
C	16	5	0	11	26
D	16	0	5	11	11

Tabla 4: Equipo C VS Equipo D (población 16)

A continuación se muestran los resultados de haber enfrentado a los equipos C y D contra un equipo random:

Equipo	Jugados	Ganados	Perdidos	Empatados	Puntaje
C	16	6	0	10	28
D	16	3	4	9	18

Tabla 5: Equipo C y D VS Equipo Random

Enfrentamos a los equipos A, B, C y D (cada uno por separado) contra el equipo "defensivo". Se mantuvieron las condiciones de juegos que mencionamos anteriormente, jugándose 64 partidos en total, donde cada equipo (A,B,C y D) jugó 16 partidos contra el equipo defensivo. Los 64 partidos jugados resultaron en empates, lo que no nos dice mucho sobre qué equipo (A, B, C o D) se comporta mejor frente a un oponente defensivo, ya que todos empataron todos los partidos que jugaron contra él.

También enfrentamos a los equipos A, B, C y D contra el equipo "atacante", con las mismas condiciones que se los enfrentó contra el equipo defensivo, y de 16 partidos jugados el equipo A perdió 12 y empató 4, el equipo B perdió los 16 partidos, el equipo C perdió 13 y empató 3 y el equipo D perdió 14 y empató 2.

Con lo que probamos hasta el momento no podríamos decidir qué método de crossover es mejor, dado que los equipos resultantes por haber utilizado cada uno de ellos, tuvieron comportamientos similares frente a los rivales con los cuales los enfrentamos, además de que sólo estamos comparando desde poblaciones iniciales de 12 y 16 individuos y con 10 generaciones.

Por otra parte, sí podemos afirmar que son equipos suficientemente "buenos" como para empatar y en algunos casos ganar (los equipos A, C y D) frente a un equipo aleatorio y empatar frente a un equipo defensivo pues todos empataron los partidos contra el equipo defensivo, y "malos" para enfrentar a otros jugadores muy singulares, pues frente al equipo "atacante" ninguno de los equipos pudo ganarle un partido.

Que los resultados de estos dos crossover sean similares frente a estos equipos externos no significa que ninguno de los dos crossover presente una ventaja o mejora con respecto al otro, son resultados absolutamente para nada conclusivos puesto que podría estar sucediendo que, pese a que alguno de los crossover sea mejor que el otro, el equipo atacante contra el que se los hizo jugar sea suficientemente bueno que ni aún la solución obtenida del mejor crossover le puede ganar.

- **Selección 1 vs Selección 2**

Para el análisis del método de selección, dejamos fijos fitness en 1, y crossover en 1 dado que fitness 1 hace una evaluación más íntegra de los individuos que fitness 2, y crossover 1 pretende hacer una combinación de progenitores lo mejor posible en contraste con crossover 2 que hace que el individuo herede los genes de uno u el otro progenitor de manera completamente aleatoria. Al igual que para el análisis de crossover 1, consideramos dos poblaciones de 12 y 16 individuos con 10 generaciones cada una y un porcentaje de mutación del 10 %.

La tabla a continuación (6) se realizó con los máximos obtenidos de la última generación.

Nombre	Selección	#Población	#Generación	Jugados	Ganados	Perdidos	Empatados	Puntaje
A	1	12	10	11	11	0	0	33
B	2	12	10	11	5	1	5	20
C	1	16	10	15	15	0	0	45
D	2	16	10	15	1	0	14	17

Tabla 6: Selección1 VS Selección2

Como en la sección anterior, hicimos competir a los equipos A y B entre sí, y a los equipos C y D entre sí, con las mismas consideraciones que tuvimos antes. A continuación se exponen las tablas 7 y 8 con los resultados:

Equipo	Jugados	Ganados	Perdidos	Empatados	Puntaje
A	16	4	0	12	24
B	16	0	4	12	12

Tabla 7: Equipo A VS Equipo B (población 12, steps = 50)

Equipo	Jugados	Ganados	Perdidos	Empatados	Puntaje
C	16	4	0	12	24
D	16	0	4	12	12

Tabla 8: Equipo C VS Equipo D (población 16, steps = 50)

Dado que en ambas competiciones (A vs B y C vs D) los resultados predominantes fueron los empates, se consideró volver a enfrentar a los equipos, hacerlos jugar entre ellos 16 partidos, pero ahora aumentando la cantidad de steps a **100** (recordar que en los cruces anteriores la cantidad de steps era **50**). Creemos que con este aumento en la cantidad de "tiempo del partido" los distintos equipos pueden desarrollar más estrategias que los lleven a poder realizar más goles. A continuación (en las tablas 9 y 10) se muestran los resultados obtenidos:

Equipo	Jugados	Ganados	Perdidos	Empatados	Puntaje
A	16	4	0	12	24
B	16	0	4	12	12

Tabla 9: Equipo A VS Equipo B (población 12, steps = 100)

Equipo	Jugados	Ganados	Perdidos	Empatados	Puntaje
C	16	5	0	11	26
D	16	0	5	11	11

Tabla 10: Equipo C VS Equipo D (población 16, steps = 100)

Al permitirles jugar más tiempo incrementando la cantidad de steps a 100, obtuvimos resultados muy similares al experimento anterior. Contrariamente a lo que esperábamos observar, que es que selección 1 obtenga mejores resultados que selección 2 pues selección 1 elige qué porción de la generación actual sobrevive y pasa a la generación siguiente en base a un puntaje de aptitud, en contraste con selección 2 que simplemente toma la porción de la generación actual que sobrevive de manera random, no hubo una significativa diferencia en el desempeño de ambos puesto que casi todos los partidos resultaron en empates. Una posible causa por la cual los resultados no se condicen con nuestra hipótesis es que estas poblaciones son muy chicas, por lo cual selección 1 puede no estar permitiendo que haya una variedad genética suficiente en la población y quedándose así en una solución muy local. Por imposibilidades de tiempo no proseguimos con esta experimentación viendo cómo contrastaba con nuestra hipótesis para poblaciones muy numerosas.

5.2.2. Fitness 2

- **Tamaño de la población**

Un análisis factible a tener en cuenta en la heurística basada en la teoría de la evolución es ver el impacto que tiene tamaños de población distintos en la performance del algoritmo genético. *Conjeturamos que una población muy grande incrementaría la diversidad de la población ayudando así al algoritmo genético a generar una solución óptima.*
Pongamos esta hipótesis a prueba en una configuración del algoritmo particular: **fitness 2, selección 1 y crossover 1**, y en donde la performance se mida a través de qué rendimiento tiene la solución generada por el algoritmo para jugar frente a los equipos random, atacante y defensivo utilizados para los experimentos de **fitness 1**. Para la mejor solución obtenida para dicha configuración y tamaños de población 16, 32, 64 y 128 respectivamente, evaluamos cuán buenas son estas soluciones para jugar contra los tres equipos mencionados, hacemos jugar cada una 16 partidos contra cada uno de los tres equipos y para una cancha de dimensiones 10x5 y 50 steps de duración del partido.

#Población	Jugados	Ganados	Perdidos	Empatados	Puntaje
16	16	0	9	7	7
32	16	0	11	5	5
64	16	0	10	6	6
128	16	0	13	3	3

Tabla 11: Diversas poblaciones de fitness2 VS Equipo Atacante

#Población	Jugados	Ganados	Perdidos	Empatados	Puntaje
16	16	0	0	16	16
32	16	1	0	15	18
64	16	0	0	16	16
128	16	0	0	16	16

Tabla 12: Diversas poblaciones de fitness2 VS Equipo Defensivo

#Población	Jugados	Ganados	Perdidos	Empatados	Puntaje
16	16	12	0	4	40
32	16	15	0	1	46
64	16	6	0	10	28
128	16	10	0	6	36

Tabla 13: Diversas poblaciones de fitness2 VS Equipo Random

Como se puede ver en los resultados de las tablas expuestas, al poner a prueba nuestra hipótesis con experimentos, no se cumplió la suposición hecha sobre la proporción directa de aumento de población y mejora de solución, al menos no según nuestra definición construida de solución buena para este contexto; en el mejor de los casos, todas las soluciones de fitness 2 con distinto número de población empataron frente al rival (esto se dio únicamente para el caso en que el adversario era un equipo defensivo).

Dado que había otros experimentos más prioritarios para llevar a cabo y disponíamos de un tiempo acotado, no pudimos seguir incursionando con este experimento, dado que fitness 2 evalúa a la población de una manera muy acotada (en una ronda hace jugar a todos los sujetos de la población de a pares, cada individuo juega únicamente con otro sujeto de la población, se elimina a la mitad de la población sacando del juego a los individuos que tuvieron un mal desempeño frente al rival con que les tocó jugar en tal ronda y se arma una siguiente ronda de juego de a pares con la mitad ganadora), hubiera sido razonable repetir el mismo experimento para fitness 1, que hace una evaluación más completa de la población que fitness 2.

5.2.3. Fitness 1 vs Fitness 2

Dado que Fitness 1 hace una calificación de aptitud de los individuos evaluando cómo es su performance en el juego contra todo el resto de los genomas de la población y Fitness 2 evalúa haciendo jugar de a pares y eliminado la mitad de los equipos hasta quedarse con el equipo que jugó y superó a la mitad de la población, ambas están evaluando individuos dentro del marco de la población del genético, fitness 2 tiene un rol importante en la experimentación dado que nos permitió poder realizar experimentos considerando poblaciones suficientemente más grandes que fitness 1 en tiempos considerablemente menores dado que fitness 1 es polinomial en la cantidad de individuos de la población a evaluar y fitness 2 es logarítmica.

Puesto que son dos métodos distintos de evaluación en el que fitness 1 parece ser más exhaustivo, **nos interesa ver si se da el caso de que este fitness 1 es más efectivo como evaluación que fitness 2.**

Con este objetivo en mente, una posible forma de contrastar esta hipotésis es tomar a la mejor y peor solución de fitness 1 que nos proporcionó el algoritmo genético en donde, como ya explicamos, mejor y peor es en relación a las definiciones que impusimos nosotros de buena solución para determinadas condiciones de juego y compararlas con la mejor y peor soluciones de fitness 2. Si por ejemplo, obtuviéramos que la mejor solución de fitness 1 le gana en una considerable proporción de partidos a la mejor solución de fitness 2 y, a su vez, que la peor solución de fitness 1 supera a la peor y a la mejor solución de fitness 2, esto sería un fuerte respaldo experimental a la hipótesis planteada. Para las mismas condiciones de cancha de antes, se hizo jugar mejor solución de fitness 1 contra mejor y peor solución de fitness 2 y exactamente lo mismo para la peor solución de fitness 1. Aclaramos qué parámetros comprenden mejor y peor solución de cada fitness :

- **Mejor solución de Fitness 1 y mejor solución de Fitness 2** método de selección = 1, crossover = 1
- **Peor solución de Fitness 1 y peor solución de Fitness 2** método de selección = 2, crossover = 2

Las condiciones de la cancha son exactamente las mismas que para los experimentos anteriores, la cantidad de steps fue 50 y la cantidad de partidos que se hizo jugar a las soluciones que se decidió enfrentar fueron 20 partidos para los cuatro enfrentamientos mencionados.

Equipo	Jugados	Ganados	Perdidos	Empatados	Puntaje
MejorFitness1	20	9	0	11	38
MejorFitness2	20	0	9	11	11

Tabla 14: Mejor solución Fitness1 VS mejor solución Fitness2

Equipo	Jugados	Ganados	Perdidos	Empatados	Puntaje
MejorFitness1	20	10	0	10	40
PeorFitness2	20	0	10	10	10

Tabla 15: Mejor solución Fitness1 VS peor solución Fitness2

Equipo	Jugados	Ganados	Perdidos	Empatados	Puntaje
PeorFitness1	20	2	0	18	24
MejorFitness2	20	0	2	18	18

Tabla 16: Peor solución Fitness1 VS mejor solución Fitness2

Equipo	Jugados	Ganados	Perdidos	Empatados	Puntaje
PeorFitness1	20	0	6	14	14
PeorFitness2	20	6	0	14	32

Tabla 17: Peor solución Fitness1 VS peor solución Fitness2

Frente a los 20 partidos jugados contra cada una, el mejor fitness 1 superó en términos de proporción de partidos ganados, empatados y perdidos del total tanto al mejor fitness 2 como al peor. Este resultado pareciera respaldar la suposición hecha previa al experimento. Por otro lado, la peor solución de fitness 1, que es con los métodos de selección y crossover ambos de valor 2, empató todos los partidos contra la peor solución de fitness 2 y únicamente ganó 2 contra la mejor solución de tal fitness; lo cual puede interpretarse como que una solución de fitness 2 no supera a una solución de fitness 1.

- **Mutación**

Dentro de los experimentos a considerar, dos experimentos de nuestro interés se centran en la influencia de la mutación en el comportamiento del algoritmo genético para determinados parámetros fijos del resto de las componentes de esta heurística. Ya explicamos en la introducción de esta experimentación porqué puede resultar de interés hacer un análisis en la relación entre el porcentaje de población mutada, y también el porcentaje de mutación de un genoma dado, y el tamaño de la población, dado que creemos que las tres variables son las que tienen una gran influencia en cuánta variedad genética tiene una población y consecuentemente, cuánto ayuda tal variedad a que haya la suficiente información para poder generar una solución lo suficientemente buena. Hay dos posibles experimentos para explorar esta relación:

- Porcentaje de población que muta

Dados fitness 1 con método de selección 1 y crossover 1 para poblaciones de tamaño $n = 12, 20, 28$ y dados fitness 2 con método de selección 1 y crossover 1 para poblaciones de tamaño $n = 16, 32, 64$, además de dejar fijo el porcentaje del genoma que muta en los valores $v = 10, 20$,decidimos variar el porcentaje de mutación a 10 %, 15 % y 20 % para dichos tamaños de población y hacer jugar a las soluciones obtenidas contra nuestros jugadores random, defensivo y atacante para luego hacer un análisis comparativo enfocándose en la pregunta ¿es inversamente proporcional el tamaño de la población y la cantidad de población mutada para generar una solución mejor o independientemente del tamaño de la población, aumentar la mutación ayuda a mejorar la solución?

- Porcentaje de genoma que muta

Análogamente al experimento explicado anteriormente, dados fitness 1 con método de selección 1 y crossover 1 para poblaciones de tamaño $n = 12, 20, 28$ y dados fitness 2 con método de selección 1 y crossover 1 para poblaciones de tamaño $n = 16, 32, 64$ y fijando el porcentaje de población que muta en los valores $v = 10, 15$ decidimos variar el porcentaje de mutación de un genoma a 10 %, 15 % y 20 % para dichos tamaños de población y posteriormente hacer jugar a las soluciones obtenidas contra nuestros jugadores random, defensivo y atacante para luego hacer un análisis de los resultados obtenidos.

Lamentablemente, por falta de tiempo no se pudieron llegar a correr dichos experimentos para poder contrastar hipótesis y sacar conclusiones respecto a estas componentes del algoritmo genético.

5.2.4. Experimento global de mejores soluciones

Dadas todas las posibles combinaciones distintas que podemos obtener en el algoritmo de genético (combinación de fitness, método de selección y crossover), un último experimento que teníamos pensado llevar a cabo era someter a un proceso de comparación a las mejores soluciones obtenidas de cada combinación posible del genético para extraer conclusiones acerca de su calidad en cuanto a solución en comparación al resto de las soluciones, es decir, analizar si hay alguna de estas mejores soluciones obtenidas que sea una mejor solución en relación a las otras mejores bajo ciertas circunstancias. Con este fin, se procuraba hacer tal proceso de comparación de las siguientes formas:

Se quería considerar el conjunto de las mejores soluciones obtenidas bajo el algoritmo genético para una cantidad de población igual a dieciséis, y con todas las variaciones posibles de fitness 1, fitness 2, crossover

1, crossover 2, selección 1, selección 2. Nos pareció de interés considerar soluciones que provinieran del mejor resultado de genético para poblaciones de distinto tamaño por el hecho de que cuanta más población, habrá una mayor probabilidad de añadir diversidad genética, pudiendo esto añadir información para generar una solución más completa.

- todos contra todos

  Se hace jugar una cierta cantidad de veces enfrentados a x, y par de soluciones del conjunto de mejores soluciones obtenidas por genético, de esta forma se puede probar cuál es el rendimiento de cada solución de las mejores obtenidas en contraste con todo el resto del conjunto de las mejores

- todos contra jugador externo

  Se generan tres jugadores con cierto comportamiento particular (un jugador con pesos random, otro con pesos de valores de manera que simule un comportamiento defensivo y por último, un jugador con pesos de valores de forma tal de obtener una táctica ofensiva) de manera que nuestro conjunto de soluciones fuera puesto a prueba con un test externo y así lograr que el resultado de desempeño no estuviera sujeto a parámetros que también pretender ser evaluados, con estos jugadores creados por fuera del genético logramos crear ese marco externo independiente para testearlos.

Estamos inclinados a decir que de todas las soluciones tenidas en cuenta en este experimento, la que mejor desempeño puede llegar a tener es la solución de fitness 1, selección 1 y crossover 1.

Conjeturamos esta hipótesis basándonos en los siguientes puntos:

- Fitness 1 hace una evaluación más exhaustiva de la población que fitness 2, en efecto, en fitness 1 todos los jugadores se evalúan en base a su performance de juego frente al resto de la población mientras que en fitness 2 juegan de a pares quedando un equipo "eliminado" pasando de esta forma el "ganador" a la siguiente ronda y jugando de a dos de vuelta hasta que quede un único equipo vencedor de todos los pares contra los que jugó, que es exactamente la mitad de la población.
- Entre crossover 1 y crossover 2, crossover 1 puede llegar a contribuir a mejores soluciones dado que se fija de los dos progenitores cuál es el que mejor desempeño tuvo hasta el momento y crea al nuevo genoma con una mayor proporción de genes de este progenitor, en contraste, en crossover 2 se decide aleatoriamente para cada gen del genoma producto de dos antecesores de quién de ellos heredeará tal gen
- Finalmente, el método de selección 1 ordena a la población de acuerdo a cuán bueno se estima qué es cada genoma (el desempeño que tuvo relativo a los otros genomas en la evaluación), se elige a la mejor mitad de la población y se hace el crossover con estos individuos para generar a la mitad restante, por otro lado, en el método de selección 2 se elige aleatoriamente a la mitad de los individuos de la población y se genera la otra mitad de la nueva generación haciendo crossover entre ellos; lo que se puede observar tanto en crossover 1 como selección 1 en contraste con crossover 2 y selección 2 es que hay una intención en los primeros de elegir a los más aptos según el criterio de aptitud que se tomó, en los otros es totalmente arbitraria la elección permitiendo que se generen mejores soluciones por aleatoreidad.

Por estar limitados con el tiempo de experimentación no pudimos llevar a cabo este experimento y contrastarlo con la hipótesis expuesta.

5.3. Grasp VS Genético

Teniendo en cuenta que nuestro principal objetivo es encontrar "buenos equipos" de juego, y que tenemos dos heurísticas para encontrar equipos, vamos a intentar hacer una comparación entre los mejores equipos resultantes de la heurística de Grasp y los mejores equipos resultantes de la heurística de Genético.
Consideramos 10 de los mejores equipos resultantes de la heurística de Grasp y 10 de los mejores equipos resultantes de la heurística de Genético, y los enfrentamos entre sí. En total se jugaron 100 partidos de 50 steps cada uno, en una cancha de 10x5, donde cada uno de los equipos enfrentó a los otros 10 de la heurística contraria.
A continuación se exponen los resultados obtenidos:

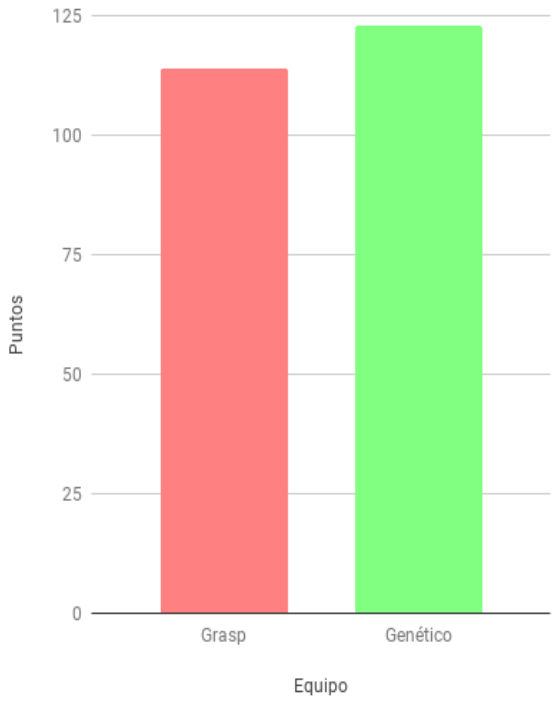

Figura 3: Mejores Grasp vs Mejores Genético

Como se puede observar en el gráfico (3) la suma de los puntajes que obtuvieron los equipos de Grasp es levemente inferior a la suma de los puntajes de los equipos de Genético. Habiendo llegado a estas instancias, no podríamos decir que las soluciones obtenidas con los algoritmos Genéticos son mejores que las obtenidas con Grasp.

5.4. Conclusiones

- A lo largo de este trabajo pudimos considerar 2 heurísticas para resolver problemas de optimización. Si bien nuestros resultados no fueron los esperados (no obtuvimos soluciones "buenas" para el problema del partido de fútbol) creemos que esto se debe principalmente a la falta de tiempo de experimentación, si tuviéramos mas tiempo para correr una mayor cantidad y variedad de experimentos creemos que obtendríamos mejores resultados que los actuales.

- Por otro lado, es difícil (y también costoso en términos de tiempo) determinar cuándo un equipo es "bueno". Dado que por más que hayamos probado a nuestro *equipo* en cuestión versus ciertos equipos particulares (como por ejemplo equipos atacantes, defensivos o random) y éste haya obtenido buenos resultados frente a dichos equipos, como existe una inmensa cantidad de equipos posibles, puede que nuestro *equipo* sea "bueno" o esté *bien entrenado* para jugar con algunos equipos, pero que sea muy "malo" al enfrentarse a otros.

- Una conclusión más a añadir es que, frente al problema puntual de querer generar un equipo en el menor tiempo posible para un torneo dado, en donde no se tiene a priori información del contrincante al cual

se enfrentará nuestro equipo o queremos que nuestro jugador tenga alguna característica particular para la cual se sabe qué aspecto tiene cierta componente de la solución, Grasp parece ser una buena opción en cuanto a costo temporal en contraste con la heurística Genética si se lo inicia en una búsqueda inicial golosa.

Referencias

[1] `https://campus.exactas.uba.ar/pluginfile.php/99684/mod_resource/content/2/tp3.pdf`

[2] J. H. Holland, Adaption in Natural and Artificial Systems. Cambridge,MA: MIT Press, 1975. 1

[3] Haupt, R. L., & Haupt, S. E. (1998). Practical Genetic Algorithms. New York: WileyInterscience

[4] Goldberg, D. E. (1989). Genetic Algorithms in Search, Optimization, and Machine Learning. Reading: Addison-Wesley

[5] Koza, J. R. (1994). Introduction to Genetic Programming. In K. E. Kinnear (Ed.),Advances in Genetic Programming (pp. 21-41). Cambridge: MIT Press

[6] Koza et al. 1999[41], p. 547

[7] Jervis and Stoffa 1993; Jervis, et al. 1996, Sen and Stoffa 1992a,b; Chunduru, et al. 1995, 1997; Boschetti, et al. 1995, 1996, 1997; Porsani, et al. 2000