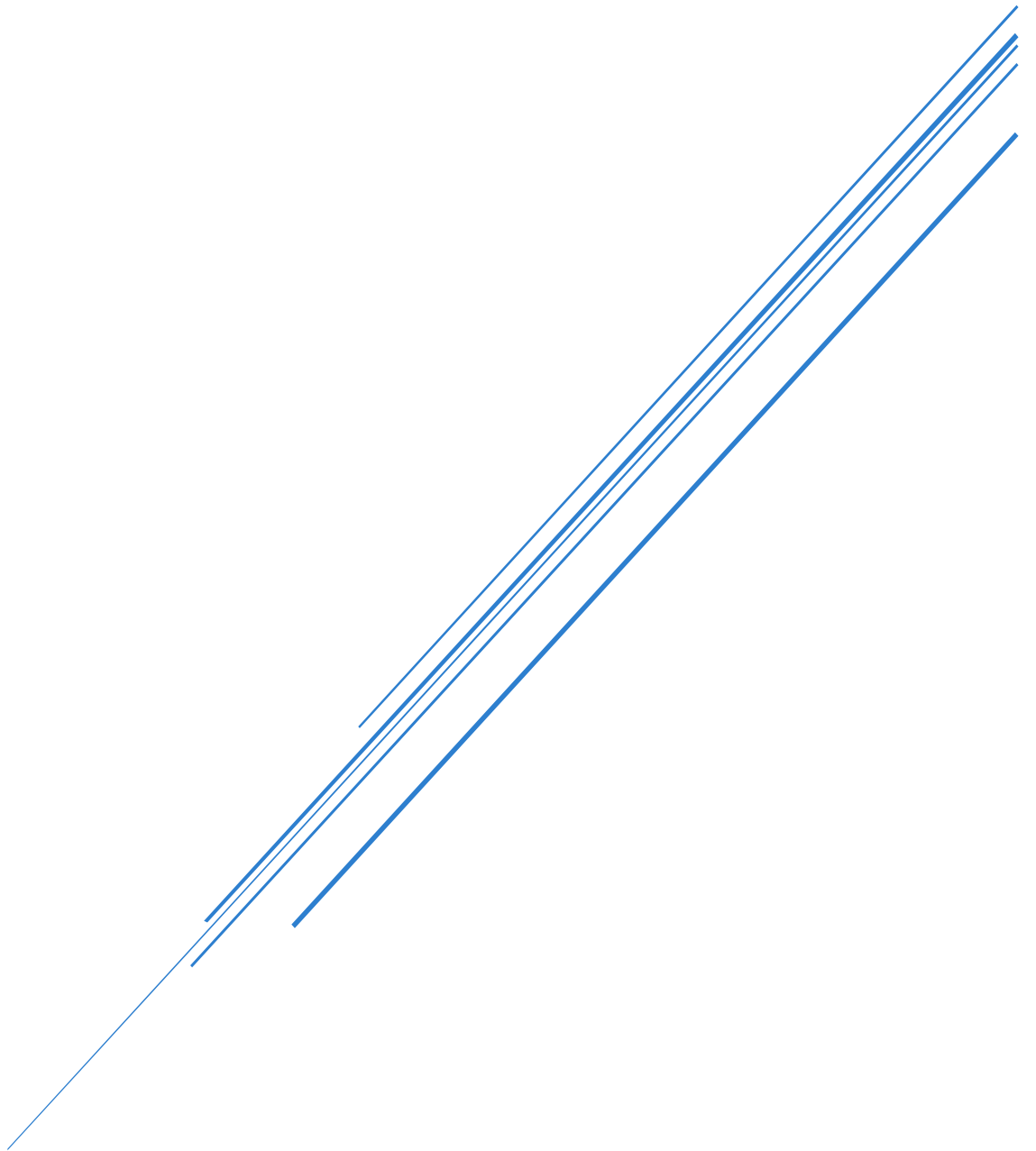


TRABAJO DE TÉCNICAS METAHEURISTICAS

Algoritmos genéticos y enfriamiento simulado



Santiago Millán Giner
MIARFID - UPV

Contenido

1.- Descripción del problema	2
2.- Parametrización del problema	3
2.1.- Individuo y gen	3
2.2.- Función objetivo	3
2.2.1.- Puntuación de ataque	4
2.2.2.- Puntuación de defensa	4
2.2.3.- Combinación de los objetivos individuales en la función objetivo	4
3.- Implementación	5
4.- Experimentos con algoritmos genéticos	6
4.1.- Clonación, mutación y supervivencia del mejor	6
Talla N=6	6
Talla N=20	8
Talla N=100	8
N=1000	9
4.2.- Precalculo de los puntuación individual	9
4.3.- Pareto	10
N = 100	10
N = 1000	11
5.- Experimentos con enfriamiento simulado	11
5.1.- Descenso exponencial de la temperatura sin Pareto	11
N=6	12
N=100	12
5.2.- Usando como punto de partida el resultado del algoritmo genético	13
5.3.- Pareto	14
N=100	14
N=1000	14
6.- Conclusiones	15

1.- Descripción del problema

El objetivo de este trabajo es hallar el equipo de **N** pokemons, teniendo en cuenta de que existen **D** especies de pokemons distintas (se llama así por la base de datos PokeDex, utilizada para resolver este problema). Para poder determinar cual es el mejor equipo, se describirá la métrica en el apartado de la función objetivo, pero primero es importante conocer algunas de las reglas básicas del juego.

En el juego cada pokemon tiene 5 estadísticas base, cada una de ellas pueden tomar un valor que va de 1 a 255. Esas estadísticas son:

- **HP:** Puntos de vida. Si llegan a 0 el pokemon muere
- **Ataque:** Daño físico
- **Defensa:** Evita que se reduzca el HP al recibir daño físico
- **Ataque especial:** Daño especial
- **Defensa especial:** Evita que se reduzca el HP al recibir daño especial
- **Velocidad:** Hace que tu pokemon ataque primero

También cada pokemon tiene uno o dos tipos elementales, dependiendo del tipo del ataque y los tipos del defensor, los HP que se restan al receptor del ataque se restan según los indicado en la tabla tipos.

Defender	Normal	Fire	Water	Grass	Electric	Ice	Fighting	Poison	Ground	Flying	Psychic	Bug	Rock	Ghost	Dragon	Dark	Steel	Fairy
Attacker																		
Normal													1/2	0			1/2	
Fire		1/2	1/2	2		2						2	1/2		1/2		2	
Water		2	1/2	1/2					2				2		1/2			
Grass		1/2	2	1/2				1/2	2	1/2		1/2	2		1/2		1/2	
Electric			2	1/2	1/2				0	2					1/2			
Ice		1/2	1/2	2		1/2			2	2					2		1/2	
Fighting	2					2		1/2		1/2	1/2	1/2	2	0		2	2	1/2
Poison				2				1/2	1/2				1/2	1/2			0	2
Ground		2		1/2	2			2		0		1/2	2				2	
Flying				2	1/2		2					2	1/2					1/2
Psychic							2	2			1/2					0	1/2	
Bug		1/2		2			1/2	1/2		1/2	2			1/2		2	1/2	1/2
Rock		2				2	1/2		1/2	2		2					1/2	
Ghost	0										2			2		1/2		
Dragon															2		1/2	0
Dark							1/2				2			2		1/2		1/2
Steel		1/2	1/2		1/2	2							2				1/2	2
Fairy		1/2					2	1/2							2	2	1/2	

Nota: En la tabla, se puede observar que hay inmunidades donde el multiplicador de daño es 0, esto puede de desarrollar la función objetivo, generando valores infinitos, es por ello que se ha optado por que los 0 pasen a ser 0,5.

Para que un equipo pokemon sea bueno, se recomienda que cubra ataque y defienda bien todos los tipos. Se valora muy positivamente tener varios pokemon que sean muy buenos atacando o defendiendo a determinados tipo, aunque sean débiles en otros aspectos, si sus debilidades son cubiertas por otro miembro. Ese criterio se le denomina **sinergia**, y hace que se prefiera un equipo donde todos los pokemon cumplan un rol determinado y lo haga muy bien, que tener miembros que hacen todo relativamente bien sin destacar.

2.- Parametrización del problema

2.1.- Individuo y gen

Tanto para el algoritmo genético y para enfriamiento simulado se ha utilizado la misma representación del problema

En primer lugar, **el individuo es el equipo (P)**, representado por una lista de **N** pokemons. Cada pokemon es un gen y esta representado por su identificador en la base de datos PokeDex, dicho identificador es un entero puede tomar valores **1 a D**.

$$P = \{p_1, p_2, \dots, p_N\} \quad \text{donde} \quad \forall p \in P, p \in \{x \in \mathbb{Z} \mid 1 \leq x \leq D\}$$

2.2.- Función objetivo

La función objetivo, mide como de bueno es el equipo atacando y defendiendo a todos los tipos sumando también como de rápido es el equipo. Para implementar esta métrica, primero es necesario calcular otras funciones que miden el rendimiento individual. Dichas funciones dependen del tipo del contrincante y de las estadísticas base del pokemon.

Para representar cada **tipo** se ha utiliza el parámetro de entrada **t**. **El factor de efectividad** es representado con la función **m(t₁,t₂)**, donde el primer parámetro indica el tipo del ofensor y el segundo parámetro el tipo del defensor.

Cada una de las estadísticas base de cada pokemon, se representan con una función que recibe como entrada el identificador de este por ejemplo: **atk(p)**, **def(p)**. Lo mismo sucede con los tipos y la función **type₁** y **type₂**

2.2.1.- Puntuación de ataque

Para evaluar la **puntuación ofensiva (física y especial) de un pokemon respecto a un tipo**, suponemos que siempre con el tipo más efectivo contra el rival, por ello se utiliza el multiplicador máximo.

$$indScore_1(p, t) = atkScore_1(p, t) = atk(p) \cdot \max_{i \in \{1,2\}}(m(type_i(p), t))$$

$$indScore_2(p, t) = spAtkScore_1(p, t) = spAtk(p) \cdot \max_{i \in \{1,2\}}(m(type_i(p), t))$$

2.2.2.- Puntuación de defensa

La **puntuación defensiva (física y especial) de un pokemon respecto a un tipo**, esta directamente relacionada con la estadística correspondiente de defensa y los puntos de vida. Una estadística de defensa muy alta, con bajos puntos de vida (hp) y viceversa, resulta de poca utilidad. Por ello, se multiplica el hp por la defensa, y se saca la raíz cuadrada para mantener el orden de magnitud.

Cuando el pokemon solo tiene un tipo, el segundo multiplicador de tipo es 1, igual que en el juego original $m(t, NO_TYPE) = 1$

$$indScore_3(p, t) = defScore(p, t) = \frac{\sqrt{def(p) \cdot hp(p)}}{m(t, type_1(p)) \cdot m(t, type_2(p))}$$

$$indScore_4(p, t) = spDefScore(p, t) = \frac{\sqrt{spDef(p) \cdot hp(p)}}{m(t, type_1(p)) \cdot m(t, type_2(p))}$$

2.2.3.- Combinación de los objetivos individuales en la función objetivo

Aclaración: Notación n-ésimo máximo

Para describir esta función, se ha usado una notación para indicar los **n-ésimo máximo** una función. A continuación, se muestra un ejemplo de la notación utilizada para hallar el n-ésimo máximo de la función de velocidad. En este caso $P(n)$ representa la el pokemon contenido en el conjunto P , que tiene la n-ésima mejor velocidad.

$$vel_{(n)}(P_{(n)})$$

La **función objetivo**, se centra en el concepto de **sinergia** explicado anteriormente, que se obtiene sumando todos los puntajes individuales de cada pokemon, dividido entre el puesto en ese puntaje en dicho equipo. Así se premia a los equipos donde cada pokemon es lo mejor en lo suyo, teniendo un rol y cubriéndose las carencias entre sí. A continuación, se muestra cómo se sumaría una estadística al puntaje global del equipo.

$$\sum_{n=1}^N indScore_{s(n)}(P_{(n)}, t) / n$$

Por ejemplo, si un pokemon tenemos un equipo $P = \{p1, p2\}$

defScore(p1, AGUA) = 100

defScore(p2, AGUA) = 50

defScore(p1, PLANTA) = 40

defScore(p2, PLANTA) = 80

A la hora de aportar al puntaje grupal se sumaría **100/1 + 50/2 + 80/1 + 40/2**

La formula de la función objetivo o fitness, es el sumatorio de todos lo puntajes individuales (**indScore**) haciendo la media para cada uno de los tipos. **T** es el número de tipos existentes.

$$teamScore(P) = \sum_{s=1}^S \frac{\sum_{t=1}^T \sum_{n=1}^N indScore_{s(n)}(P_{(n)}, t) / n}{T} + \sum_{n=1}^N vel_{(n)}(P_{(n)}) / n^2$$

La estadística de velocidad se suma de forma independiente, pero la posición esta al cuadrado, porque sin este se premiaba demasiado esta estadística, obteniendo malos resultados.

3.- Implementación

Para resolver este problema se ha optado por usar el lenguaje de programación **Python** por los siguientes motivos:

- Facilidad de instalar **añadir librerías usando pip**, como es el caso de pypokedex, que se utiliza para acceder a la base datos de pokemons.
- **Capacidad de integrar módulos en C++ o usar numpy**, para acelerar el programa para acelerar el programa si fuese necesario.

- **Integración con Jupyter**, permitiendo escribir en un documento código y formulas, pruebas y las gráficas de los resultados de esta.
- Permite la **redefinición de funciones** fácilmente, ya que **las funciones son tratadas como una variables**. Esta característica del lenguaje ha sido realmente útil, ya que me ha permitido ir intercambiando las funciones (como la de selección, la de reproducción o mutación) entre experimentos, sin cambiar el código fuente del algoritmo genético o enfriamiento simulado.

4.- Experimentos con algoritmos genéticos

4.1.- Clonación, mutación y supervivencia del mejor

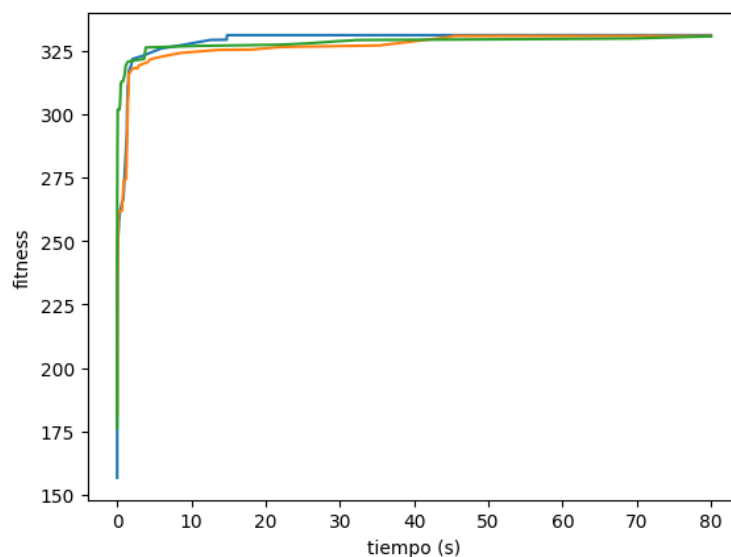
Talla N=6

Para el primer experimento se ha implementado la solución más sencilla, con idea de poder ir añadiendo más complejidad si fuese necesario para mejorar la solución. Inicialmente se va a probar con equipos de **N=6 miembros, D=1024 especies, T=18 tipos elementales**. Las tallas elegidas, se basan en las reglas del último juego sacado hasta el momento.







Este algoritmo está inspirado en el funcionamiento de los virus o seres unicelulares en la naturaleza. Primero, cada individuo de la población **se clona realizando siempre el mismo número de copias idénticas**.

Después toda cada gen de cada individuo de la población **muta con una probabilidad constante** de que un que cada gen sea reemplazado por otro pokemon al azar.

Al final de la iteración, únicamente pasan a la siguiente iteración un número constante de supervivientes, **los seleccionados son aquellos que tienen mejor fitness** (teamScore)



En 3 experimentos diferentes, haciendo **2 copias** de cada equipo, solo **quedando 4 supervivientes**, con una **tasa de mutación un 1/3**. Podemos ver como la gráfica converge en menos de 20 segundo a individuos con un **fitness de 330**. Para todos los experimentos se ha partido de una población con un solo equipo, formado por el mismo pokemon con bajas estadísticas, para ver cómo evoluciona la solución.

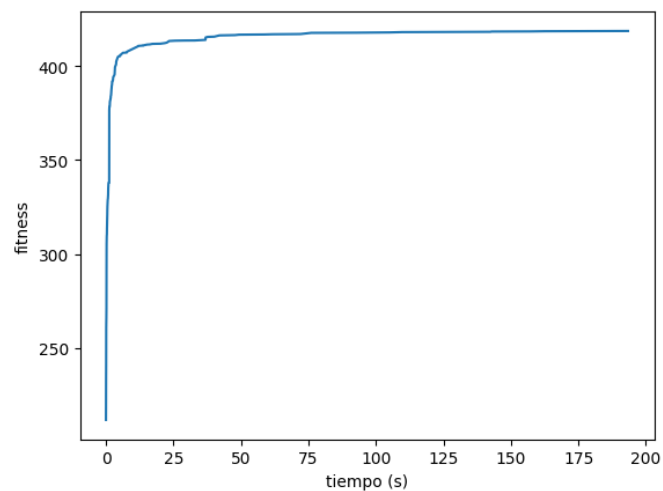
	Fitness	Segundos	Iter
	330.8227314229026	45.415054	3874
			
	330.64355637642507	79.592888	6649
			
	331.14501928004677	16.245724	1394
			

Los 3 experimentos comparten bastantes genes entre sí y podemos observar como hay diversos patrones, como poner 2 veces el pokemon que amarillo y una vez el rojo.

Se ha probado con otros parámetros y se consiguen tablas muy similares, en que el fitness mejora por uno o dos puntos.

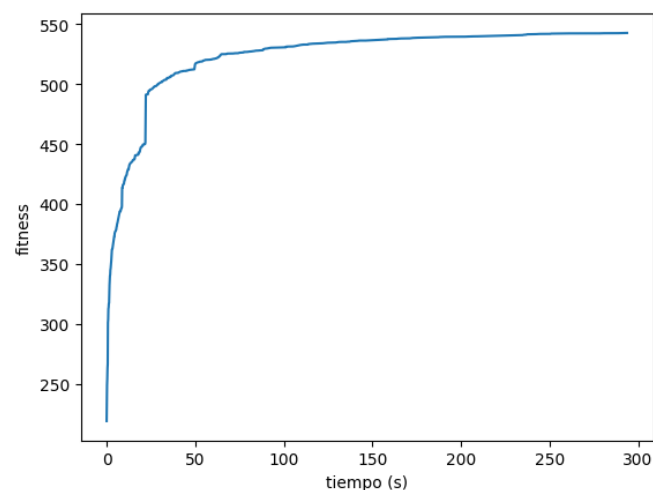
Talla N=20

Al ver que esta primera solución ha funcionado realmente bien, se ha probado a incrementar la talla del problema a **N=20 miembros**. Se ha optado por cambiar la tasa de mutación a **1/17**, ya para que de media cambien 2 o 3 genes por iteración



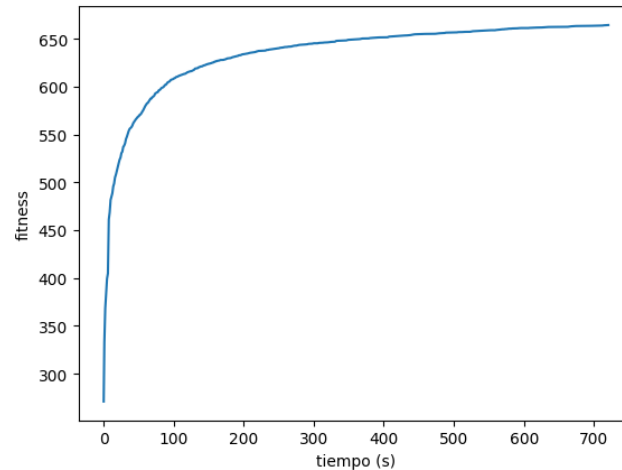
Al ver los resultados, parece converger rápidamente a un fitness de alrededor de 418 en menos de 50 segundos. Podría aumentar más la talla del problema.

Talla N=100



Con $N=100$ y ratio de mutación= $1/95$, tarda casi 5 minutos en converger a 540 de fitness. Los resultados siguen siendo bastante razonables, pero se nota que la curva es mucho más pronunciada.

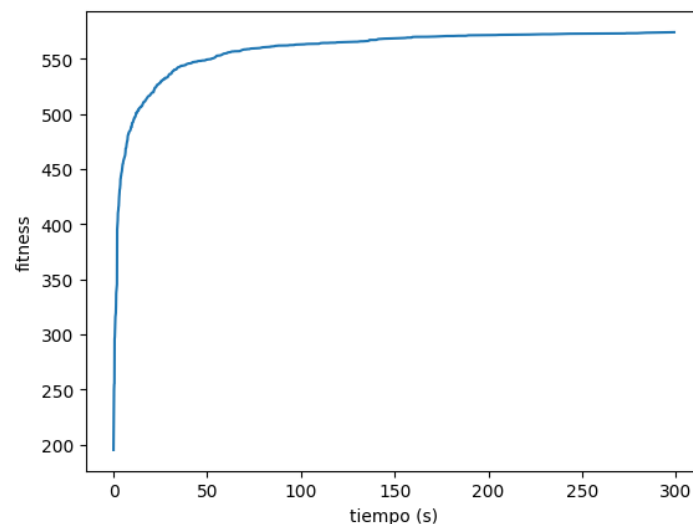
N=1000



Con $N=1000$ tarda más de 10 minutos en converger, pero las soluciones empiezan a tener algunos genes que claramente pueden mejorarse, debido a que incluyen pokemons con bajas estadísticas que se pueden reemplazar por uno mejor

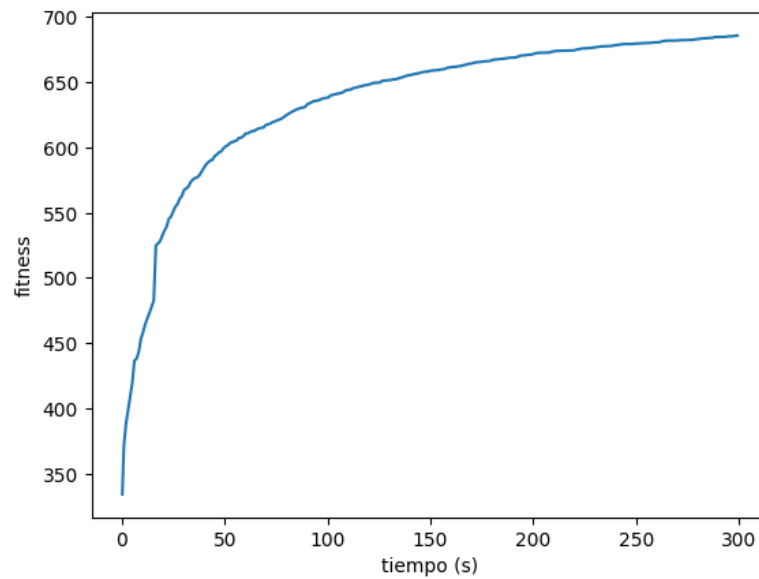
4.2.- Precalculo de los puntuación individual

En este experimento, se va a optimizar calculando previamente la puntuación ofensiva y defensiva para cada tipo. El cálculo dura alrededor de **70 segundos** de todos puntajes individuales para **D=1024, N=100 y tasa de mutación 1/95**



Se tenía previsto que mejorará sustancialmente, porque ya no tiene que consultar en la base datos externa y hacer todas para el cálculo de los puntajes para cada tipo. Aunque se puede apreciar una ligera mejora, no compensa el tiempo que tarda

en precalcular. No obstante, estos datos podrían ser utilizados en un experimento futuro. Con **N=1000** sucede lo mismo que en el experimento anterior.

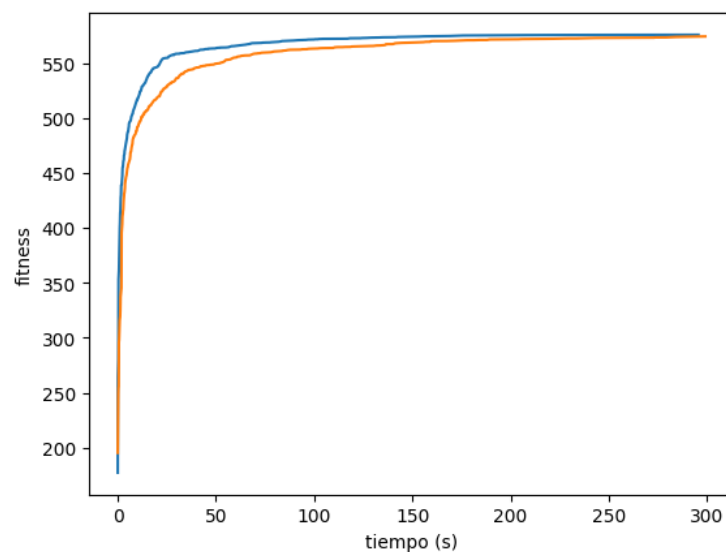


4.3.- Pareto

N = 100

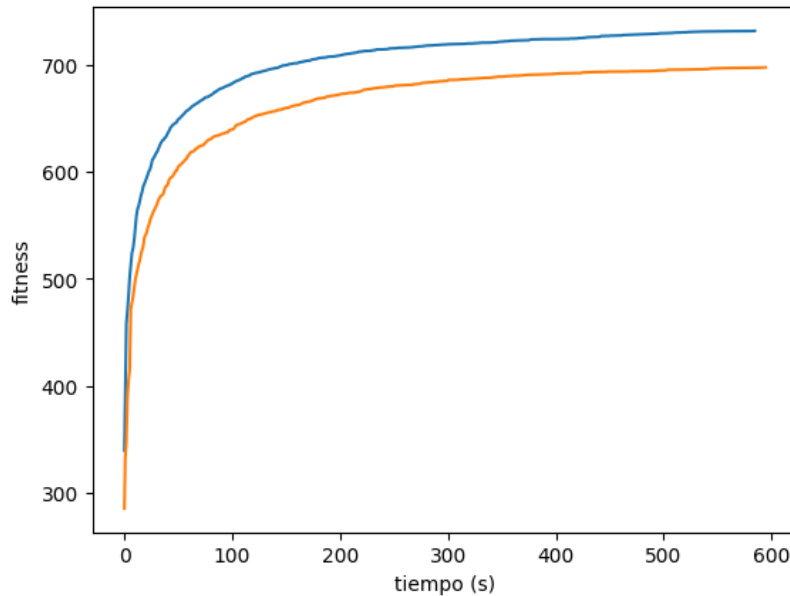
Aprovechando los datos anteriormente calculados, podríamos excluir aquellos genes que aportan menos que otros en todas sus estadísticas individuales. Para ello se va a utilizar la frontera de Pareto. El número de dimensiones para cada punto va a ser $4 \cdot 18 + 1 = 73$, ya que hay 4 puntajes dependientes del tipo, 18 tipos y la velocidad que es independiente del tipo.

Al aplicar Pareto, el número de genes posibles pasa de ser **D=1024** especies a **390**. El tiempo que se tarda en realizar este cálculo es depreciable, ya que no llega a las decimas de segundos.



Como podemos ver tarda menos en converger y obtiene resultados finales un poco mejores y a aumentado muy poco el fitness al converger. No obstante, si tenemos en cuenta el tiempo que ha tardado en precalcular, no queda que sea más rápido.

N = 1000



Al hacer el experimento con **N=1000** sorprende como alcanza **un fitness 731**, superando al anterior por 32 puntos, que llegaba a 697.

5.- Experimentos con enfriamiento simulado

5.1.- Descenso exponencial de la temperatura sin Pareto

Igual que en el primer experimento genético, se opta por hacer una solución sencilla con el objetivo de ir perfeccionándola en un futuro, en caso de que sea necesario.

En este apartado para compararlo con el genético se va a partir de un individuo formado únicamente por el mismo pokemon con muy malas estadísticas. Igual que el anterior.

En el primer algoritmo, el vecino obtenido que se obtiene es un **clon mutado**, igual que en el [apartado 4.1](#). La temperatura desciende exponencialmente, en cada iteración se multiplica por un **factor de enfriado**.

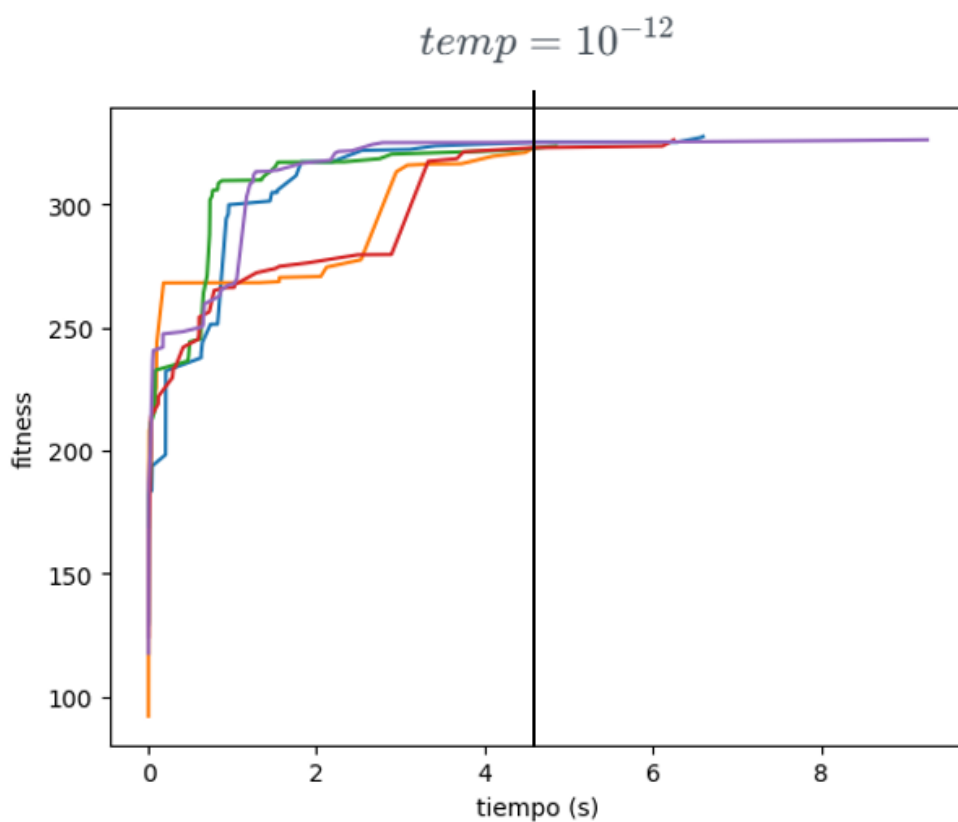
$$t(i) = t(0) * enfriamiento^i$$

$$t(i + 1) = t(1) * enfriamiento$$

El ratio de mutación de cada gen va a ser el mismo que en los experimentos anteriores dependiendo de la talla de N.

N=6

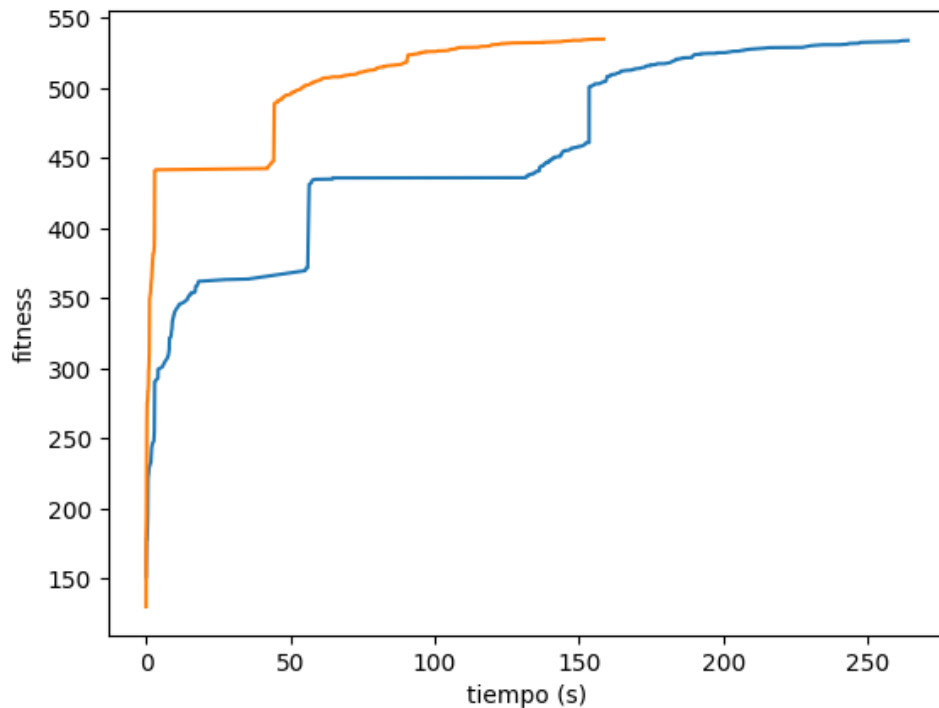
Con N=6 podemos ver que alcanza un fitness similar al anterior, pero converge más rápido, incluso en los peores casos. Para este experimento se ha usado una **temperatura inicial de 10.000** y una **tasa de enfriamiento de 0.98**. El programa se ejecuta hasta llegar a una temperatura mínima.



Con **temperatura de 10^{12}** converge. El tiempo que tarda en alcanzar punto de convergencia es **de un máximo de 4 segundos** superando por 6 segundos al genético.

N=100

Al aumentar la talla, también debemos de ajustar la temperatura para que haga más iteraciones y le dé tiempo a mejorar. Para ello se ha ajustado el **ratio de enfriamiento a 0.995**, la **temperatura inicial a 10^7** y la **temperatura mínima de 10^{-14}** .

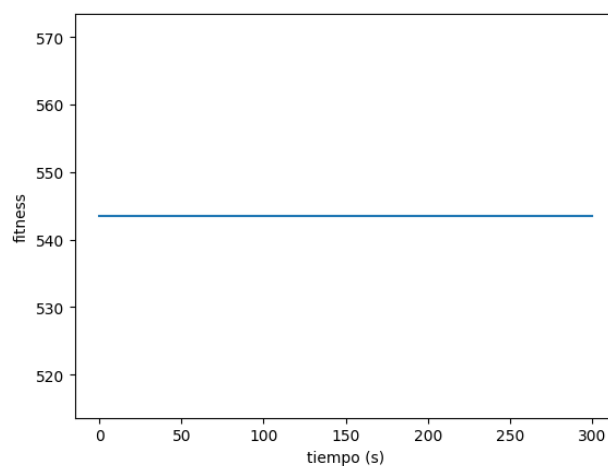


Aunque tarda menos en converger, los resultados son peores, alcanzando solo 534 en repetidos experimentos, mientras que el genético podía alcanzar resultados que podían superar los 550.

También se puede ver mejor como el algoritmo si encuentra un equipo bueno antes, en las iteraciones posteriores descartará más nuevos candidatos cosa que hace que su ejecución sea más rápida.

5.2.- Usando como punto de partida el resultado del algoritmo genético

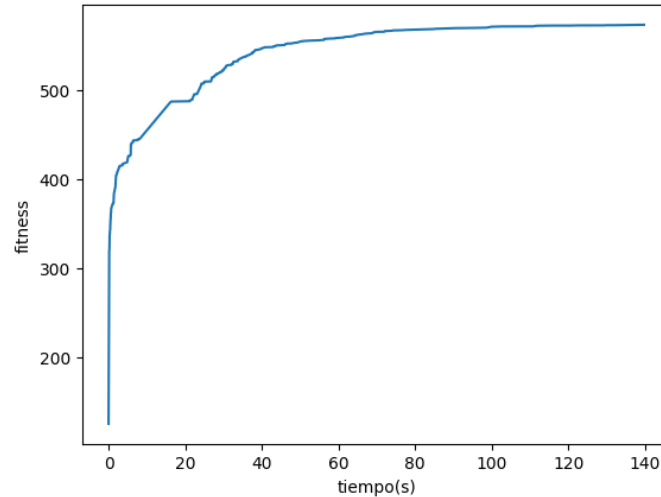
Hemos visto que para **N=100** el algoritmo genético da mejores resultados que el de enfriamiento simulado. No obstante, quizás al partir de un buen equipo el de enfriamiento simulado da mejores resultados.



Como se puede observar en la gráfica, **no hay ninguna mejora**.

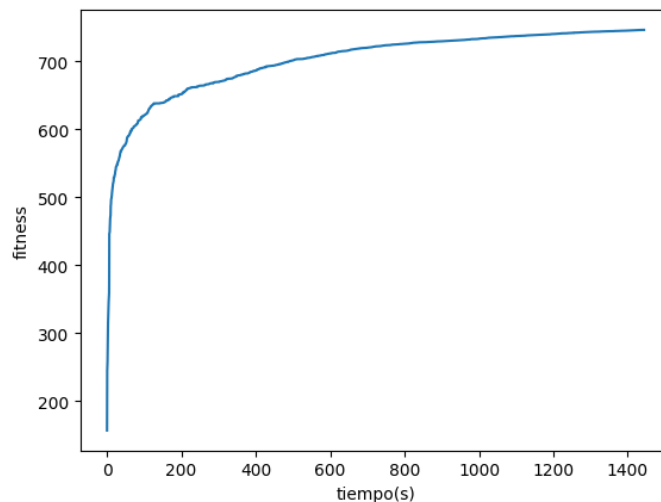
5.3.- Pareto

N=100



Al usar el filtrado de Pareto con el enfriamiento simulado mejora los resultados respecto a usar todos los genes. Aunque sigue tardando casi lo mismo en converger, alcanza un **fitness 573, respecto a 555** que alcanzaba el genético con este filtrado. Además, tarda casi lo mismo que el genético en alcanzar los mismos resultados.

N=1000



No obstante, con la talla **N=1000**, aunque mejoran los resultados respecto a no usar Pareto significativamente, no consigue dar resultados tan buenos como el genético, dando un fitness de **730 frente a 770**. Además, tarda más en converger, por lo tanto no sería recomendable usar este algoritmo frente al genético para este tamaño de tallas.

6.- Conclusiones

En este trabajo, se ha demostrado que en primer lugar hay que probar un algoritmo simple, antes que otros más complejos, ya que puede dar sorprendentemente buenos resultados y siempre se puede ir añadiendo complejidad u optimizando en caso de que sea necesario, para problemas más difíciles de solventar. Esto a quedado demostrado para este caso de uso, ya que al intentar optimizar calculando con anterioridad las estadísticas individuales de cada pokemon, se ha obtenido un resultado mucho más lento, por el tiempo del precálculo de todos los genes de alrededor de un minuto.

Pero también se ha visto, como toda esa información y trabajo de adicional, puede ser utilizado en otros experimentos, en este caso para filtrar aquellos genes que no tienen ninguna ventaja frente a otros usando Pareto. Esto ayudó a reducir la talla **D** significativamente, pasando de 1024 a 390, al tener un número más limitado de candidatos a escoger, el fitness aumente en general y lo haga ligeramente más rápido.

Se demostró como la talla **N** del problema afecta mucho no solo al tiempo en que tarda en converger, si no a la calidad de los resultados, tanto para el algoritmo genético y para enfriamiento simulado. También, dependiendo de la talla del problema se deben de ir cambiando ciertos parámetros y un algoritmo puede ser mejor que otro. **En este proyecto, hemos visto que la implementación del genético escala mejor, obteniendo mejores fitness respecto al enfriamiento simulado en tallas de N a partir de 100 sin usar Pareto y 1000 usando Pareto.** Aunque el enfriamiento simulado es más rápido convergiendo conforme más pequeña es la talla. Además, se vio que para cada algoritmo era necesario cambiar la tasa de mutación y la tasa de enfriamiento (en el caso del genético) para obtener buenos resultados, dependiendo del valor **N**.