PRACTICA 04

METAHEURÍSTICAS
SLIME MOULD ALGORITHM
MICROORGANISMOS

JOSE MANUEL OSUNA LUQUE 20224440-B

JOSUNALQ@CORREO.UGR.ES

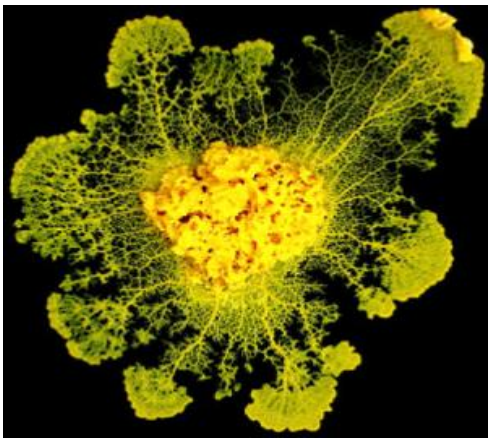
2019/2020

Índice

Introducción al Algoritmo

El problema elegido, ***Slime Mould Algorithm***, antes de explicar en qué consiste, lo mejor será definir en qué se basa. En la naturaleza, la presencia del limo, o moho de limo es considerado un organismo primitivo de la familia Fungi (clasificación usada para denotar a los organismos vivos eucariotas incluyendo levaduras, mohos y setas). El nombre científico de este moho de limo es ***Physarum Polycephalum***, el cual no es ni un animal, ni una planta, ni tan siquiera un hongo, es un organismo algo extraño y amorfo, compuesto de una única célula gigante, que, pese a no tener cerebro, es capaz de ***aprender de la experiencia***.

Un detalle muy interesante de este organismo, es que, además de ser capaz de aprender aun sin poseer cerebro, es capaz incluso de ***transmitir*** la experiencia obtenida a un molde de limo, o moho de limo compañero cuando se combinan.



En el momento en el que se produjo dicho descubrimiento, se comenzó a cuestionar que la vida inteligente pudo haber aparecido incluso antes que los propios cerebros, con esa habilidad tan personal de los humanos, el ***aprendizaje***. Cuando se demostró por primera vez en la historia que un organismo con una única célula sin cerebro o sistema nervioso era capaz de aprender, forzó una posible reconsideración sobre el origen del aprendizaje a raíz de la experiencia, el cual hubiera ocurrido mucho antes de la evolución de formas de vida con cerebro.

Una vez que se ha explicado su origen en la naturaleza, se puede apreciar una ligera similitud en algoritmos aplicados en el ámbito de la Inteligencia Artificial, siendo, ni más ni menos, que el Aprendizaje. Es común en el estudio de dicha rama de la informática la construcción de algoritmos que hagan uso de una variable (normalmente llamada eta, η), como se usa, por ejemplo, en el Algoritmo de *Gradiente Descendente Estocástico*.

Basar un Algoritmo en un **Suceso** Natural o Biológico no es algo extraño, de hecho, la mayoría de los Algoritmos Metaheurísticos están inspirados en *Manifestaciones del Mundo Real*, intentando buscar una mejor solución heurística para problemas de optimización. Esto lo consiguen simulando *Reglas del Mundo Físico*, o incluso *Fenómenos Biológicos*. Teniendo esto en cuenta, los Algoritmos Metaheurísticos podrían dividirse en dos categorías:

- Métodos Basados Comportamientos Colectivos (Swarm-Based Method) → Simula sucesos físicos aplicando Reglas Matemáticas (Enfriamiento Simulado o Búsqueda Tabú).

- Técnicas Evolutivas → Originado por el propio proceso evolutivo en la naturaleza. Métodos de Optimización Global cuya robustez y aplicación es bastante mejor (Algoritmos Genéticos).

Sin importar el tipo de Algoritmo, ambos tienen dos etapas en común: Exploración y Explotación.

+ Exploración: Búsqueda de un Espacio Solución tan aleatorio y global como sea posible.

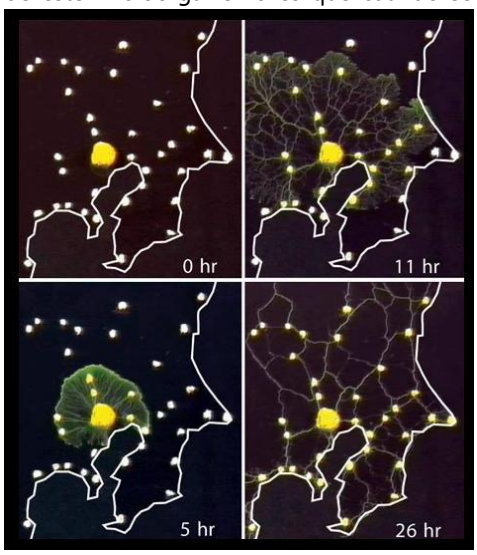
+ Explotación: Búsqueda de la solución más acertada y precisa, al tiempo que mientras la aleatoriedad de un espacio de soluciones disminuye, la precisión en la solución aumenta.

Adaptación del Algoritmo

Se ha explicado hasta ahora lo que sería el aliciente natural en el que basar el Algoritmo que se representará, pero, no se ha explicado cómo trabaja dicho microorganismo. Por tanto, se hará una breve presentación de ello. El Algoritmo dicho, principalmente simula el comportamiento y los cambios morfológicos del moho de limo conocido como Physarum Polycephalum.

El moho de limo (Physarum Polycephalum) tiene como etapa principal el ***Plasmodium*** (o Plasmodio), el resultado de la división no completa de una única célula. En esta etapa, el limo *busca comida*, y para ello, secreta unas sustancias conocidas como **enzimas** (un término que se ha visto en problemas de Inteligencia Artificial al usar *Agentes Reactivos*). Durante este proceso, la parte frontal del limo se extiende, llegando a forma una red interconectada que contiene el citoplasma. Debido a esta peculiar característica, es capaz de buscar en diversas fuentes de alimentación al mismo tiempo, en esa red interconectada.

Llegados a este punto, se puede asimilar, a que este proceso no es del todo distinto a tener un *grafo* con varios *nodos solución*, siendo estos la comida que busca. Un detalle interesante de este microorganismo es que cuando se aproxima a la comida (dígase de la forma "siente que está cerca la comida"), el *bio-oscilador* produce una oleada de propagaciones que hace incrementar el fluido citoplasmático a través de esa "venita" nerviosa, por así denominarla, que está más próxima de la comida. Si esta característica se traduce a un problema de búsqueda de una solución óptima, esa segregación más fuerte y rápida que hace el limo se puede contemplar como "peso" a la hora de hablar de una solución. A través de esta combinación con los pesos, se puede generar un feedback balanceado entre positivo y negativo (una vez se establezca un umbral) haciendo posible a este hallar una estabilidad suficiente para encontrar el ***camino más óptimo para dar una solución óptima*** (para el microorganismo, encontrar comida).



La calidad de las diversas fuentes de alimentación será diferente, por lo que el Limo tiene la capacidad de decantarse por aquel punto en el que sea *mayor la concentración* del alimento. Un algoritmo basado en esto usaría la idea para encontrar la solución más óptima, el mejor camino hacia la mejor solución. Sin embargo, no todo es tan sencillo, realizar esto implica dar un valor más elevado a la velocidad de propagación, que al mismo tiempo supone un riesgo en la búsqueda. El Limo debe tomar decisiones realmente rápidas, debiendo evitar además cualquier peligro que presente el ambiente en el que se desarrolla (¿óptimos locales?)

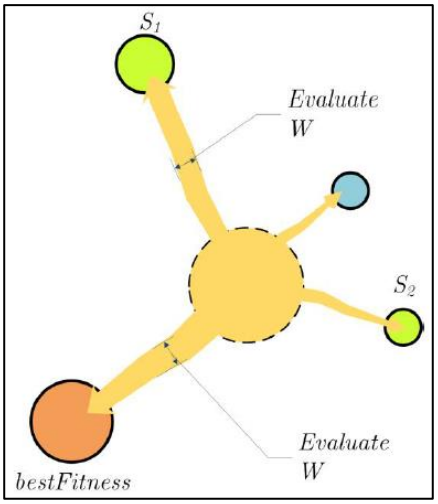
Esto se puede asimilar un poco a los *Algoritmos Genéticos*, que trabajan con poblaciones, compuestas por cromosomas. El Torneo Binario hacía una elección elitista para ver qué Cromosoma seguía siendo un posible candidato para cruzarse con otro Cromosoma. Es decir, "desechar" a un padre suponía no seguir buscando por esa solución. Muy similar a lo que ocurre con el Limo, el cual debe decidir cuándo deja de explorar un área. Si ocurre el caso de que no posee suficiente información para moverse a otra área de búsqueda, la mejor forma para realizar dicha "migración" es estimar de forma heurística la regla a seguir. Por ello, si el Limo encuentra un área en la que la calidad es buena, es difícil que este se aleje o separa de ella (*Limitación de la Región*). Si encuentra una fuente mejor, se divide y se dirige hacia ella. Esta cualidad del Limo le permite explotar así ambas fuentes de alimentación.

Implementación del Algoritmo

El Limo se acerca a la comida mediante el olor. A un nivel matemático se intentará imitar su comportamiento, la forma de expandirse del Limo. Para conseguir algo, hay que apoyarse en unos elementos globales previamente, y trabajar sobre ellos.

- Un Elemento alpha, cuyo valor oscila entre [-a, a] aleatoriamente.
    + Este valor ***a*** se obtiene a partir de una fórmula → (arctg(- (t/max_iters)+1)
- Un Elemento beta, cuyo valor oscila entre [-1, 1], con una tendencia a 0 con la ejecución del algoritmo.
- Un Elemento t que servirá como contador de las iteraciones llevadas a cabo.
- Un Elemento expansion (vector) que representa la localización del Limo.
- Un Elemento listaS (vector) que guarda la Función Objetivo de X en cada iteración.
- Un Elemento SmellIndex que guarda el orden de las Funciones Objetivo de mejor a peor.
- Un Elemento bestF que representa el mejor valor Fitness (Función Objetivo) hasta el momento.
- Un Elemento XA (vector) y XB (vector) representarían dos selecciones individuales aleatorias.
- Un Elemento W (vector) que representa el peso del Moho de Limo.
- Un Elemento bFi que representa el valor Fitness más óptimo obtenido en la iteración actual.
- Un Elemento wFi que representa el peor valor Fitness obtenido en la iteración actual.
- Un Elemento rValue que simplemente es un valor aleatorio entre 0 y 1, [0, 1]

Lo que se pretende simular es una expansión del Limo a través de una superficie en la que buscará comida, que se traducirá en una buena función Objetivo. Por esto mismo, cuando la concentración de comida sea elevada (cuando la función objetivo sea muy buena, tenga un feedback positivo, y siendo en este caso, cuando su valor sea menor, ya que se pretende minimizar), biológicamente hablando, el ancho de la vena del Limo expandiéndose será mayor (siendo aquí donde se usa esa variable Weight), en caso contrario, este valor será menor, y por tanto incitará a explorar otras regiones, abandonado esta.



Para representar una actualización del Limo, de su localización, hablando de un nivel matemático, se usarán la siguiente fórmula:

$$\overrightarrow{X^*} = \begin{cases} rand \cdot (UB - LB) + LB, rand < z \\ \overrightarrow{X_b(t)} + \overrightarrow{vb} \cdot \left(W \cdot \overrightarrow{X_A(t)} - \overrightarrow{X_B(t)}\right), r < p \\ \overrightarrow{vc} \cdot \overrightarrow{X(t)}, r \geq p \end{cases}$$

Empezando por el valor **rand**, este, al igual que r, simplemente es un valor aleatorio en un rango entre 0 y 1. El valor **z** será más un parámetro ajustando a razón de los experimentos y las pruebas realizadas. **UB** y **LB** establecen los límites de búsqueda.

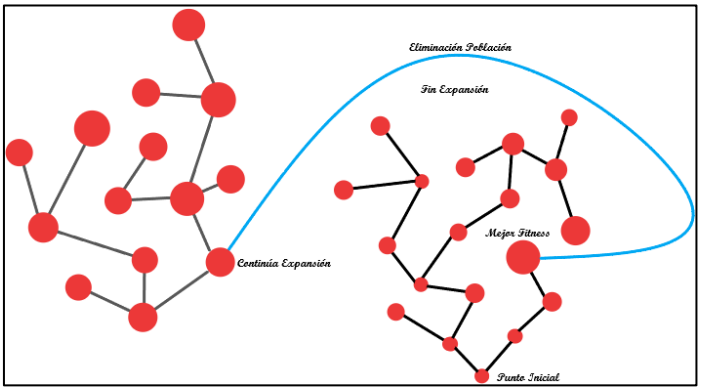
El Limo se expande por una superficie, y esto hay que representarlo de alguna forma en la creación del algoritmo, a nivel matemático. Los elementos antes definidos, como son **alpha** y **beta** se usarán, jugando al peso **W** (todos vectores), para simular la oscilación que el Limo produce al moverse en busca de comida (nuestra, ya mencionada Función Objetivo a minimizar). W será la frecuencia de oscilación del Limo al acercarse a una concentración de comida, permitiendo que pueda acercarse a una mejor función objetivo mucho más rápido. Alpha oscila entre un valor [-a, a] que con el paso del tiempo irá aproximándose a cero, al igual que beta, salvo que su rango es [-1, 1]. La sinergia de alpha y beta denotaran el comportamiento selectivo del moho de limo. Incluso habiendo encontrado una buena función objetivo, se expandirá por otros caminos para hallar, o comprobar que no hay otra mejor. Con esta forma de actuar, se evita el óptimo local.

Diseño Propio basado en el Algoritmo

Por motivos basados en no encontrar una forma de adaptar el algoritmo a las prácticas vistas, y, en especial, al problema elegido del Clustering, haciendo uso de la forma de trabajo del Moho de Limo (así como de su composición biológica y su morfología), se ha pensado en una posible variante de este Algoritmo. El Moho de Limo, en la naturaleza, como se ha mencionado, se expande sobre una superficie alentado por la búsqueda de una concentración de comida. Su desarrollo se hace subdividiéndose en pequeñas protuberancias, o venas, dispersas sobre un camino. Sobre esta idea se piensa rápidamente grafos, o en Búsqueda por Trayectorias, en entornos.

La expansión que el Limo hace sobre la superficie, y la que se contempla en el algoritmo, se ha interpretado como una Población de soluciones, con la salvedad de que la población estará vacía, empezando con una solución inicial en ella (lo que sería un Cromosoma). Esta población irá creciendo usando operadores, como, por ejemplo, la generación de vecinos (es una primera consideración), hasta un máximo establecido (por ejemplo 100 elementos). Cada nuevo vecino, o cada nuevo "cromosoma", solución, se considerará una porción del Limo y pertenecerán a la población, es decir, a la expansión del mismo al crecer por la supuesta superficie.

Cuando llega al máximo de su expansión (bien por temas de memoria o para conseguir una rápida convergencia) establecido, se produce una selección totalmente elitista, traducido en un cambio generacional completo. Se mantiene solo la porción del Limo (el cromosoma) de mejor función objetivo (como se ha mencionado en repetidas veces, el objetivo del problema es minimizar este valor, por lo que la porción elegida es la que tenga un Fitness más reducido). El resto de la población queda eliminada. ¿Por qué hacer esto? El Limo, su funcionamiento era que si encontraba una fuente de alimentación con un gran valor (sea un valor calórico, por ejemplo, por poner una situación) desviaba su atención a una búsqueda regional sobre dicha zona, quitando importancia a las demás. Es la característica principal del Limo, la expansión sobre una superficie dejándose llevar por el "olor" más fuerte, por la mayor concentración de comida. Al mantenerse siempre por una región y abandonar el resto (pero sin dejar de expandirse) es lo que consigue preservar la obtención de una solución óptima, evitando así el conformarse con el óptimo local.



Elegir siempre la mejor solución para expandirse puede ser un error, porque siempre se mantiene en la misma línea, y puede llegar un momento en el que se haya llegado a un óptimo local. Por ello, la expansión del Limo es aleatoria, y solo se elige el mejor nodo cuando se ha terminado de expandir hasta un máximo de porciones. Aunque para apremiar la rápida convergencia, cada vez que se produce una generación de vecinos, aquel que se guardará en la población será el mejor posible (se crean los vecinos pertinentes y se les aplica un Torneo Binario).

Se asegurará en todo momento que cuando se generen hijos, no se guarden en la población un hijo que ya exista en ella (no debe haber inconsistencia).

Implementación del Diseño basado en Physarum Polycephalum

Durante la construcción de este, se han obviado algunos elementos que no han hecho falta para simular la expansión del Limo y su funcionamiento. Algunas características han tenido que adaptarse al Clustering, ya que la idea no bastaba para hacerlo realidad.

El Algoritmo se apoya en varias funciones esenciales, pero antes de pasar a ello, se indicará la representación del problema:

El algoritmo trabaja con un objeto que guarda la composición del Moho de Limo, uno llamado **slime**. El DataFrame slime guarda una población que va muriendo cada n número de elementos en ella (prefijado en 50). Esto se ha creado así para no tener problemas de memoria, y tampoco dar opción a tener un abanico de posibilidades demasiado amplio que perjudique la búsqueda de la solución óptima, hallar su camino.

Al hacer esa purga en la población, el objeto slime se queda con un elemento (el mejor) cuando llega a 50 individuos en la población (un valor que puede variar modificando la variable global TAMSLIME). Para esto hace uso de una función llamada **ReestructurarSlime()** cuyo funcionamiento es sencillo: ordena el slime a razón de la función objetivo, recoge el mejor individuo (el primero ahora), y elimina el resto. Tras esto, actualiza el tamaño del slime a 1. Para esto hace uso de función llamada **ActualizarRangoSlime(True)** (true porque, efectivamente, ha sufrido un sesgo, en caso contrario solo aumenta en 1 el tamaño que tuviera). Para ordenarlo, se llamada a una función nombrada **SmellIndex()**. En cada iteración se comprueba que el tamaño del Slime no supere al máximo prefijado, TAMSLIME.

Para que el Slime crezca, se hace uso de una función denominada **Expandir()**, la cual es compuesta, ya que esta, en su interior, se apoya en otra denominada **ElegirCandidato()**.

Comenzando por la de elegir un elemento como elemento para expandirse el Limo, la función *ElegirCandidato*(), está basada en la ecuación usada para actualizar las posiciones del Limo en el algoritmo SMA. Se ha intentado construir una versión lo más fiel posible, aunque con ciertos cambios. La ecuación es la siguiente:

$$\overrightarrow{X^*} = \begin{cases} rand \cdot (UB - LB) + LB, rand < z \\ \overrightarrow{X_b(t)} + \overrightarrow{vb} \cdot \left(W \cdot \overrightarrow{X_A(t)} - \overrightarrow{X_B(t)}\right), r < p \\ \overrightarrow{vc} \cdot \overrightarrow{X(t)}, r \geq p \end{cases}$$

En la ecuación, aunque no queda muy claro en el paper, da a comprender que, en el primer rango, la elección de la expansión es aleatoria, un elemento cualquiera de la población, ya que UB y LB denotan, respectivamente, el límite de la población.

xB lo representa como un vector, que guarda la mejor posición hasta el momento en la ejecución del algoritmo, vB es un elemento (llamado beta en la presentación) que sirve como oscilador, al igual que vC para el camino que tomaría el Limo. W son unos pesos con los que se deja llevar. Tanto xA como xB son dos elementos de la población aleatorios.

El elemento X es la población como tal.

La adaptación realizada es la siguiente: Se respetan los rangos de elección, por lo que previamente se tienen calculados los valores z, y p, dónde z es un valor preajustado antes del inicio del algoritmo por el programador y p es calculado mediante una ecuación (y se va actualizando durante la ejecución del Algoritmo). Dividiendo las opciones de expansión en tres (3), en la primera condición se ha mantenido igual, un valor aleatorio a razón del tamaño de la población en ese momento. En la segunda opción, solo ha usado el valor de los pesos, multiplicándolos con la función objetivo, y

eligiendo el mejor valor. Esto, dicho así, no tendría sentido, ya que sería lo mismo que elegir la mejor función objetivo, sin embargo, se comprenderá mejor cuando se explique como se calculan los pesos, W. En la tercera opción, se ha optado por elegir el mejor elemento encontrado hasta ahora, denominado xBest.

```
Def ElegirCandidato():
        # son elementos distintos
        rand = random(0,1)
        r = random(0,1)

        IF rand < z THEN
                ab = abs(rand * (upperB – lowerB) + lowerB)
                pos = parteEntera(ab)
                RETURN slime[ab].solucion
        ELIF r < p THEN
                RETURN AproximacionComida()
        ELSE
                RETURN xBest.Solucion # porque xBest guarda la solución, desviación, infact, y Fitness
```

Para calcular los pesos, W, se hace uso de la función que ofrecen, la cual es también una aplicación basada en límites.

$$\overrightarrow{W(SmellIndex(i))} = \begin{cases} 1 + r \cdot log\left(\frac{bF - S(i)}{bF - wF} + 1\right), condition \\ 1 - r \cdot log\left(\frac{bF - S(i)}{bF - wF} + 1\right), \quad others \end{cases}$$

La condición que menciona es solo que el valor i esté por debajo de la mitad inferior del tamaño de la población. Aquí está el factor que carga de sentido el usar los pesos en la función **AproximacionComida()** indicada en el pseudocódigo. Aplica un valor random a la operación, además de otras cuentas, por lo que no solo depende de la función objetivo. Haciendo varias pruebas, se ha podido apreciar que una solución con mejor fitness tenga un peor valor al multiplicarlo con su peso, que otra que tenga mejor fitness. Favorece tanto la explotación como la exploración. La única modificación es que, al estar ordenados (es lo que indica la función SmellIndex), como se busca minimizar la función, los mejores están los primeros, y tiene un valor más pequeño, por lo que en la primera mitad se resta, y en la segunda se suma (da mejores resultados).

En **AproximacionComida()**, se recorre toda la población, y se multiplica cada peso w con su solución. Se devuelve el elemento que mejor valor al multiplicar la función objetivo con su peso resulte.

Por último, *Expandir*() solo llama a *ElegirCandidato*() y con el elemento que recibe, genera un nuevo vecino con la función GeneracionVecinos(sol), teniendo en cuenta que ningún cluster se quede vacío, y que el nuevo cluster elegido no sea el mismo, ya que lo que hace la función es cambiar de cluster a un elemento elegido al azar.

Con estas funciones explicadas, falta el cuerpo del Algoritmo como tal. Este ha quedado bastante simplificado ya que todo el esfuerzo de código está implementado en funciones aparte. Está compuesto de un bucle cuya condición de parada se puede cumplir de dos formas: Una (1) es llegando al máximo de iteraciones, MAX_ITERACIONES, prefijado por el programador, la segunda opción es que el algoritmo haya conseguido en su mejor solución una infactibilidad de 0 (solución óptima). Como Alpha y Beta ya no se usaban, se pensó en una opción para implementar dichos valores y crear una especie de *Mecanismo de Enfriamiento* (como en el algoritmo SA), porque al jugar con la aleatoriedad esa infactibilidad de 0 puede no llegar nunca, o los datos pueden ser inmensos.

```
Def Diseño_SMA():

        t = 0 # Contempla las iteraciones llevadas a cabo
        parada = False
        finAlgoritmo = False

        WHILE not finAlgoritmo THEN

                IF upperB == TAMSLIME THEN
                        ReestructurarSlime()

                Expandir()
                ActualizarLimo()

                CalcularW()
                ActualizarP()
                t++

                finAlgoritmo = t >= MAX_ITERACIONES OR parada
```

Aparece la variable parada, ¿qué es? Bien, ese elemento comienza en False, y controla si la mejor solución encontrada hasta el momento ha llegado a ser óptima. Este valor se actualiza en la función ActualizarLimo() que lo único que hace es actualizar xBest, la variable que es usada para controlar la mejor solución hasta el momento en toda la ejecución, pero además dos variables más, xBestIter y xWorstIter. Estos dos últimas son variables guardan el mejor y el peor elemento de la población actual (se recuerda que la población puede sufrir un sesgo, y quedar con un único elemento. Si al actualizar el Limo, la variable concreta xBest resulta que su infactibilidad es 0, parada se torna True, y finaliza el algoritmo.

$$p = \tanh|S(i) - DF|$$

La fórmula anterior es la que indicaba el paper, y la usada para calcular el valor de p. No quedaba claro qué era S(i), solo que S era la población. En el pseudocódigo que dan se entiende que por cada elemento de la población se calcula un valor p, sin embargo, eso no tenía sentido con las fórmulas antes mencionadas, ya que trataban p como un único elemento. La decisión tomada para ese valor S(i) es elegir el último elemento añadido a la población, el valor de su función objetivo. El valor DF indica el mejor Fitness obtenido hasta el momento en toda la ejecución del objetivo.

Experimentos

Se ha llevado a cabo la ejecución del algoritmo con los cuatro modelos de datos usados en las prácticas anteriores, IRIS, ECOLI, RAND Y NEWTHYROID, junto a sus conjuntos de restricciones del 10% y 20%.

Ejecución 10% - Problema PAR

	IRIS				ECOLI			
	Tasa_C	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T
SMA	0.11502172742837923	0,00	0.11502172742837923	43.0864953994751	1.159180211484343	461,00	13.52760123017594	1113.0631465911865

	RAND				NEWTHYROID			
	Tasa_C	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T
	0.11826035491701258	0,00	0.11826035491701258	46.456844329833984	0.6563525952722555	141,00	5.812901043586395	299.61873292922974

Ejecución 20% - Problema PAR

	IRIS				ECOLI			
	Tasa_C	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T
SMA	0.11502172742837923	0,00	0.11502172742837923	66.4359610080719				

	RAND				NEWTHYROID			
	Tasa_C	*Tasa_inf*	*Agr.*	T	*Tasa_C*	*Tasa_inf*	*Agr.*	T
	0.11826035491701258	0,00	0.11826035491701258	87.06644344329834	0.631158514822412	378,00	7.541625436081147	453.777548789978

Tras realizar los Experimentos se puede apreciar que el Algoritmo base creado trabaja bastante bien cuando el problema tiene una cantidad considerable, y el número de las restricciones no es tan elevado. A medida que aumenta el valor de estos, el tiempo de ejecución, por tanto, también aumenta (era algo con lo que se contaba), pero la Función Objetivo no alcanza el valor de solución óptima global., como ocurre en Ecoli o Newthyroid. No son valores muy elevados, pero sin duda no es lo que se quiere.

Comentarios

La realización de esta práctica comenzó siendo un intento de adaptación del algoritmo basado en el microorganismo ***Physarum Polycephalum***, o ***Moho de Limo***, cuyo comportamiento se basa en la búsqueda de comida guiado por el olor de la concentración de esta, siendo capaz de tomar las decisiones pertinentes para localizar y llegar al mayor foco de alimento. Se tomó como referencia el estudio realizado sobre este, y el desarrollo del algoritmo denominado ***Slime Mould Algorithm***. Sin embargo, cuando se intentó construir adaptándolo al Clustering, el problema de prácticas realizado en la asignatura, surgieron varias complicaciones. Tampoco se llegó a comprender o llegó a quedar claro la estructura y representación usada en el paper, más allá de que el Slime, el Limo, era considerado como una población de soluciones.

Ante la adversidad, se intentó adaptar lo que se pudo, y reconstruir el resto. El resultado es un algoritmo que, en una primera ejecución a IRIS y con el 10% de restricciones, en un tiempo que rondaba los 20 segundos encontró una solución (tras 2240 iteraciones). Se hicieron varias comprobaciones de prueba y la media de tiempo rondaba entre los 25 y 60 segundos, un tiempo que parece bastante razonable.

Ante esta situación, se llegó a la conclusión de que quizás adaptar el algoritmo resultante haciendo un híbrido con búsqueda local para rellenar esa población con ella podría hacer al algoritmo mucho más lento. En la práctica anterior, búsqueda local por sí sola ya tardaba un tiempo mayor que el tiempo que tarda este algoritmo que intenta similar el comportamiento del Moho de Limo.

Son resultados bastante llamativos, pero quizás no es la construcción inicial que se esperaba del Algoritmo, de hecho, quizás no se pueda considerar una implementación del Algoritmo ***Slime Mould Algorithm***, si no una variable, que por algún motivo da resultados óptimos, quizás por una demasiado mala implementación. La aleatoriedad implementada, así como el uso de una población, hace que no sea un **Greedy** como tal, porque no en todas las iteraciones toma la mejor solución para trabajar sobre ella. Este es el principal motivo por el que evita esos óptimos locales, y es capaz de alcanzar la solución óptima.

No obstante, como puede apreciarse en los experimentos, a medida que crece el problema, las soluciones comienzan a no ser las deseadas. Es aquí cuando entra en juego la idea de implementar una hibridación con Búsqueda Local, por ejemplo. Quizás su implementación hace que para IRIS que tarda una media de 40 segundos, aumente su tiempo de cómputo, sin embargo, de cara a pensar que los problemas venideros sean de un tamaño considerable, quizás merece la pena aumentar el tiempo de ejecución para problemas pequeños si con ello se consigue una implementación capaz de afrontar problemas de cualquier tamaño.

Con el tiempo disponible, es lo máximo que ha sido posible implementar, aunque bajo un punto de vista subjetivo, se considera que sí que puede ser una buena aproximación al algoritmo que se empezó a implementar, aunque quizás la falta nociones y conocimiento por parte del propio estudiante también hayan sido el principal problema para no adaptar de forma perfecta dicha teoría.

Referencias Bibliográficas

https://docs.scipy.org/doc/

https://docs.python.org/

https://www.researchgate.net/

http://www.alimirjalili.com/

Shimin Li, Huiling Chen, Mingjing Wang, Ali Asghar Heidari, Seyedali Mirjalili, Slime mould algorithm: A new method for stochastic optimization, Future Generation Computer Systems, 2020. https://doi.org/10.1016/j.future.2020.03.055