



IE3041 - DISEÑO E INNOVACIÓN DE INGENIERÍA

Mini Proyecto Inicial: Swarm Robotics y Swarm Intelligence

Eduardo Santizo Olivet(16089)
Asesor: Dr. Luis Alberto Rivera
Sección: 10
26 de febrero de 2020

Índice

| | |
|---|-----------|
| Marco Teórico | 3 |
| Particle Swarm Optimization (PSO) | 3 |
| Orígenes e Historia | 3 |
| Funcionamiento | 4 |
| Mejoras Posteriores | 5 |
| Otros Algoritmos de Swarm Intelligence | 6 |
| Genetic Algorithm (GA) | 6 |
| Glowworm Swarm Optimization (GSO) | 7 |
| Pruebas y Resultados | 9 |
| Resultados: Simulación de PSO | 9 |
| Pruebas: Variación de Parámetros κ , ϕ_1 y ϕ_2 | 12 |
| Bibliografía | 15 |

Marco Teórico

Particle Swarm Optimization (PSO)

Orígenes e Historia

El algoritmo de Particle Swarm Optimization (PSO) consiste de un método de optimización estocástica¹, basado en la emulación de los comportamientos de animales que se movilizan en conjunto. Los inicios del algoritmo se remontan a 1987, cuando Reynolds [5] desarrolla un método para animar una parvada de «boids», objetos similares a aves cuyo comportamiento individual se ve regido por 3 simples reglas: Cada «boid» solo es capaz de adquirir información de sus vecinos próximos, estos evitarán colisiones con otros «boids» y cada uno adquirirá la velocidad promedio de sus vecinos sin alejarse de la parvada.

Se dice que Kennedy y Eberhart [2] tomaron el algoritmo de Reynolds y comenzaron a experimentar con el mismo. Luego de múltiples pruebas y observaciones, estos se percataron que el algoritmo no solo simulaba un comportamiento natural, sino que presentaba las cualidades de un método de optimización. En base a esto, modificaron las reglas del algoritmo original (Por ejemplo, suprimiendo la regla de la proximidad hacia otras entidades) y propusieron dos variaciones el algoritmo de PSO: La implementación GBEST y LBEST.

La implementación GBEST asume la existencia de un ente global que es capaz de monitorear, recordar y transmitir la posición y velocidad de todas las partículas a todas las demás. La implementación LBEST se asemeja mucho más al enfoque local tomado por Reynolds, donde cada entidad solo es capaz de conocer información procedente de sus vecinos próximos.

Cabe mencionar que se le denominó «Particle Swarm Optimization» en lugar de «Particle Flock Optimization» ya que dentro de las simulaciones realizadas por Kennedy y Eberhart, se pudo observar que el comportamiento conjunto de las partículas se asemejaba más al movimiento de un enjambre que al de una parvada o banco [3].

¹Estocástico: Cuyo funcionamiento depende de factores tanto predecibles dado el estado previo del sistema, así como en factores aleatorios

Funcionamiento

El algoritmo propone la creación de un conjunto de « m » partículas, cada una con un vector de posición y velocidad correspondiente. Estas partículas se desplazan sobre la superficie de una función objetivo cuyos parámetros (Variables independientes) son las « n » coordenadas de cada partícula. A dicha función objetivo se le denomina «función de costo» y al escalar que genera como resultado se le denomina «costo».

$$\vec{x}_i = [x_{i1} \ x_{i2} \ x_{i3} \ \dots \ x_{in}]$$

$$Costo = f(\vec{x}) = f(x_{i1}, x_{i2}, \dots, x_{in})$$

El objetivo de las partículas es encontrar un conjunto de coordenadas que generen el valor de costo más pequeño posible dentro de una región dada. Para esto, las partículas se ubican en posiciones iniciales aleatorias y se permite que las mismas «exploren» de manera iterativa hasta encontrar el mínimo global de la región.

Para conseguir esto, en cada iteración cada partícula calcula el valor del costo correspondiente a su posición actual y compara con el costo obtenido en la posición previa. Si el costo actual es inferior, se dice que la partícula ha encontrado un nuevo «personal best» ($\vec{p}(t)$), por lo que se almacena en memoria tanto el valor de costo como la posición que lo ha generado.

$$\vec{p}_{pos}(t) = [x_{min1} \ x_{min2} \ x_{min3} \ \dots \ x_{min^n n^n}]$$

$$p_{cost}(t) = f(\vec{p}_{pos}(t))$$

Este proceso se repite para cada partícula en el enjambre, por lo que al finalizar cada iteración del algoritmo se contará con « m » estimaciones de $\vec{p}(t)$. El valor mínimo de todas estas estimaciones se le conoce como el «mínimo global» de dicha iteración. Si este mínimo es inferior al de la iteración previa, se dice que se ha encontrado un nuevo «global best» ($\vec{g}(t)$). Nuevamente, se almacena tanto el valor del costo, como la posición que ha generado dicho valor.

Para la actualización de la posición, las partículas suman a su posición actual una nueva velocidad calculada utilizando su velocidad actual ($\vec{V}(t)$), las posiciones asociadas al personal y global best ($\vec{p}_{pos}(t)$, $\vec{g}_{pos}(t)$), y dos escalares aleatorios uniformemente distribuidos (R_1 , R_2). Debido a que el

«personal best» proviene de la memoria individual de cada partícula sobre su mejor posición hasta el momento, a la sección de la ecuación de la velocidad que utiliza $\vec{p}_{pos}(t)$ se le denomina el «componente cognitivo». Por otro lado, debido a que el «global best» proviene de la memoria colectiva sobre la mejor posición alcanzada hasta el momento, a la sección de la ecuación de la velocidad que utiliza $\vec{g}_{pos}(t)$ se le denomina «componente social».

$$\begin{aligned}\vec{V}(t+1) &= \vec{V}(t) \\ &\quad + C_1(\vec{p}_{pos}(t) - \vec{x}(t)) && \text{Componente Cognitivo} \\ &\quad + C_2(\vec{g}_{pos}(t) - \vec{x}(t)) && \text{Componente Social}\end{aligned}$$

$$\vec{X}(t+1) = \vec{X}(t) + \vec{V}(t+1)$$

Cabe mencionar que la implementación detallada previamente consiste de la implementación GBEST. La implementación LBEST no se describe en mayor detalle por los autores debido a que es muy similar a la anterior. La única diferencia con respecto al GBEST, es que en esta no existe un único global best. Debido a que cada partícula es capaz de comunicarse con sus vecinos próximos, pueden llegar a existir múltiples versiones del global best.

Mejoras Posteriores

Debido a que la primera implementación del PSO consistía de la modificación de un algoritmo pre-existente basado en evidencia puramente experimental, este carecía de fundamentos teóricos que justificaran su funcionamiento y fallas. Con el objetivo de solucionar esto, Clerc y Kennedy [1] se dieron a la tarea de tomar un enfoque más riguroso para el análisis del algoritmo.

A partir de su análisis se llegó a determinar que la razón por la que el algoritmo presentaba un comportamiento oscilatorio o divergente en ciertas situaciones era porque no existía control sobre la influencia que tenía cada uno de los componentes de $\vec{V}(t+1)$ sobre el valor final de este vector. Tomando en cuenta esto, se diseñaron múltiples métodos para restringir y asegurar la convergencia del algoritmo. Uno de los métodos más comúnmente implementados consiste de una modificación a la regla de actualización de la velocidad conocida como «modelo tipo 1»:

$$\begin{aligned}
 \vec{V}(t+1) = & \omega \vec{V}(t) && \text{Término Inercial} \\
 & + R_1 C_1 (\vec{p}_{pos}(t) - \vec{x}(t)) && \text{Componente Cognitivo} \\
 & + R_2 C_2 (\vec{g}_{pos}(t) - \vec{x}(t)) && \text{Componente Social}
 \end{aligned}$$

En este, las nuevas variables de restricción agregadas están dadas por las siguientes expresiones:

$$\begin{aligned}
 \omega &= \chi & \chi &= \frac{2\kappa}{|2 - \phi - \sqrt{\phi^2 - 4\phi}|} \\
 C_1 &= \chi \phi_1 & \phi &= \phi_1 + \phi_2 \\
 C_2 &= \chi \phi_2
 \end{aligned}$$

Como se puede observar, bajo estas modificaciones, la velocidad es ahora dependiente de tres variables nuevas: ϕ_1 , ϕ_2 y κ . Clerc y Kennedy sugieren que $\phi_1 = \phi_2 = 2.05$ y $\kappa = 1$, aunque como regla general, para asegurar la convergencia del algoritmo se debe cumplir con que $\kappa > (1 + \phi - 2\sqrt{\phi})|C_2|$. Puede que otras combinaciones generen convergencia, pero esto se torna en un resultado altamente dependiente de la disposición aleatoria inicial de las partículas.

Otros Algoritmos de Swarm Intelligence

Genetic Algorithm (GA)

Creado por John Holland en 1975, el algoritmo genético consiste de un algoritmo evolutivo altamente dependiente de procesos aleatorios. Debido a que este utiliza una simplificación del proceso de selección natural para optimizar una «fitness function», este se ve sujeto a mecanismos inherentemente aleatorios como lo son la recombinación genética o las mutaciones.

En este algoritmo, cada agente se denomina cromosoma. Un cromosoma consiste de un vector de valores binarios, donde cada valor binario se considera un «gen». Debido a que cada gen únicamente puede representar dos estados, cualquier otro tipo de información de mayor complejidad que desee optimizarse debe codificarse utilizando código binario. Esto implica que los cromosomas del algoritmo, al igual que en una cadena genética real, cuentan con una representación «genotípica» (La cadena binaria como tal) y una representación «fenotípica» (Lo que representa la cadena binaria) [6].

| | Genotipo | Fenotipo |
|-------------|----------|----------|
| Cromosoma 1 | [0 1 1] | = 3 |
| Cromosoma 2 | [1 1 0] | = 6 |
| Cromosoma 3 | [1 1 1] | = 7 |
| Cromosoma 4 | [0 1 1] | = 3 |
| Cromosoma 5 | [0 0 0] | = 0 |

En términos de su implementación, el algoritmo inicia con una inicialización aleatoria de los valores de todos los cromosomas. Luego se toma la representación fenotípica de cada cromosoma y se evalúa en la «fitness function». Aquellos cromosomas que retornen un valor por encima de un determinado threshold, se consideran aptos para reproducirse, por lo que se incluyen en una «mating pool». Dentro de este subconjunto, se agrupan todos los cromosomas en parejas, ya sea de forma secuencial (1-2, 3-4, etc.) o de forma aleatoria y se someten a recombinación.

En este proceso, los cromosomas padres se subdividen en 2 secciones de largo aleatorio y se crean dos cromosomas hijos que consisten de la mezcla de esas dos secciones. Los dos cromosomas resultantes se agregan a la población y se someten a un proceso aleatorio de mutación de baja probabilidad (De esta manera se asegura el mantenimiento de la diversidad de la población, además de prevenir la convergencia prematura del algoritmo). El algoritmo llega a su fin cuando las nuevas generaciones no son lo suficientemente diferentes de sus padres, lo que indica convergencia [4].

Al igual que en el caso del PSO, este algoritmo cuenta con múltiples variaciones debido a su antigüedad. Una de las más simples, consiste en codificar los datos a optimizar utilizando datos no binarios. En algunas implementaciones, cada gen consiste de una variable independiente a optimizar, por lo que la representación genotípica y fenotípica van a ser exactamente iguales.

Glowworm Swarm Optimization (GSO)

Propuesto por Kishnanad y Ghose en 2005, este algoritmo consiste de una de las implementaciones de Swarm Intelligence más recientes. En este, los agentes dispuestos a optimizar la función objetivo (O «fitness function») se denominan luciérnagas. Cada luciérnaga cuenta con tres parámetros

característicos: Una posición en el espacio ($x_m(t)$), un nivel de «luciferina» ($l_m(t)$) y un rango de vecindad ($r_m(t)$).

Durante la inicialización, las luciérnagas son distribuidas de manera aleatoria en una región específica de la función específica y sus parámetros adquieren valores dictados por constantes pre-definidas. Luego, se inicia el proceso de búsqueda. Como primer paso se actualiza el nivel de luciferina de cada luciérnaga. Para esto se utiliza la siguiente ecuación:

$$l_m(t) = (1 - p) \cdot l_m(t - 1) + \gamma J(x_m(t))$$

donde la constante p consiste del factor de evaporación de luciferina, γ consiste de la constante de luciferina y J consiste de la función objetivo. Una vez actualizado el nivel de luciferina de las « m » luciérnagas presentes en la población, estas proceden a actualizar su posición utilizando la siguiente ecuación:

$$x_m(t) = x_m(t - 1) + s \left(\frac{x_n(t - 1) - x_m(t - 1)}{\|x_n(t - 1) - x_m(t - 1)\|} \right)$$

donde s es la magnitud del paso que cada luciérnaga avanzará durante cada iteración, $x_n(t - 1)$ la posición previa de la luciérnaga vecina más cercana y $x_m(t - 1)$ la posición previa de la propia luciérnaga. Ahora bien, una luciérnaga solo es considerada una vecina si la distancia entre ellas es menor al denominado rango de vecindad ($r_m(t)$) y la luminosidad de la luciérnaga n es mayor al de la luciérnaga m . Si existen múltiples candidatos para vecino, las luciérnagas eligen una única vecina utilizando la siguiente ecuación de probabilidad:

$$p_m(t) = \frac{l_m(t) - l_n(t)}{\sum_{k \in W(i')} l_k(t) - l_n(t)}$$

donde la probabilidad de que la luciérnaga m se mueva hacia la n es igual a la diferencia en sus niveles de luciferina, dividido la suma de todas las diferencias de luciferina existentes entre la luciérnaga m y sus vecinas. La luciérnaga elige a la vecina con la probabilidad más alta y se mueve una distancia s en su dirección. Finalmente, se actualiza el rango de vecindad utilizando la siguiente ecuación:

$$r_m(t + 1) = \min \{r_s, \max [0, r_m(t) + \beta (n_d - |n_m(t)|)]\}$$

donde r_s consiste del rango de sensado (Constante que limita el tamaño de r_s), n_d consiste del número deseado de vecinos por luciérnaga, $|n_m(t)|$ consiste del número de vecinos de la luciérnaga m durante la iteración t y β consiste de una constante de modelado. Al igual que en el PSO, el algoritmo se detiene cuando la diferencia entre la posición previa y la posición actual de cada luciérnaga es inferior a cierto «threshold».

Una de las características más importantes del GSO, es que este método es capaz de optimizar funciones «multimodales» o con múltiples mínimos locales. Debido a esto, esta es apta para ser utilizada en procesos de optimización numérica, aunque a cambio, esta posee una precisión relativamente baja y una lenta tasa de convergencia. Algunas modificaciones han buscado mejorar esto expandiendo el rango de vecindad de las luciérnagas para que abarque toda la región de búsqueda y reduciendo al mínimo el número de posibles vecinos de cada luciérnaga. Esto ha traído mejoras la precisión y rapidez de convergencia del método respectivamente [4].

Pruebas y Resultados

Resultados: Simulación de PSO

A continuación se presentan la visualización gráfica de la simulación del algoritmo de PSO programada para cada una de las funciones de costo. Para algunas de las visualizaciones en 3D más difíciles de entender, se proporciona la visualización 2D. Se utilizaron los valores recomendados de κ , ϕ_1 y ϕ_2 .

Pruebas: Variación de Parámetros κ , ϕ_1 y ϕ_2

Tal y como se mencionó en la sección «Mejoras Posteriores», una de las mejoras más significativas del algoritmo de PSO, consistió en la introducción de una condición que restringe los diferentes componentes que actualizan la velocidad para así asegurar la convergencia del método. La justificación de porque estas restricciones aseguran la convergencia matemática, es sumamente compleja y requeriría de mucho tiempo para poder llegar a comprender todas las piezas que componen la explicación.

Debido a esto, se decidió hacer un conjunto de pruebas donde se varían los valores de los parámetros κ , ϕ_1 y ϕ_2 (Variables independientes de la función de restricción χ), para así observar de manera empírica cuales son los efectos de variar estos sobre la convergencia del algoritmo. Se produjeron 15 variaciones de cada parámetro entre 0.1 y su valor recomendado ($\kappa = 1$, $\phi_1 = 2.05$ y $\phi_2 = 2.05$) [1]. Esto implicaba 3375 combinaciones distintas.

Una corrida de prueba entonces se consideró como la ejecución del algoritmo de PSO utilizando cada una de las 3375 variaciones de parámetros. El resultado de cada corrida era un vector columna de 3375 filas, donde en cada casilla se almacenaba el número de iteraciones que le tomaba al algoritmo converger. Además, debido al carácter estocástico del método, los resultados no siempre eran iguales, por lo que se optó por realizar un total de 20 corridas y obtener la mediana de los resultados al finalizar.

Considerando que se disponía de 5 funciones de costo distintas (Paraboloide, Ackley, Rastrigin, Levy y Dropwave), todo el procedimiento anterior se repitió un total de 5 veces. Los resultados fueron graficados en un plot tipo «scatter» donde el eje X, Y y Z consistían de ϕ_1 , ϕ_2 y κ , respectivamente y la intensidad de color en una región indicaba la convergencia del método en menos iteraciones². A continuación se presentan los resultados obtenidos:

²La intensidad del «alpha» de cada punto en el «scatter plot» es inversamente proporcional al número de iteraciones que le tomó converger al algoritmo de PSO. En otras palabras, mientras más oscuro, menos iteraciones.

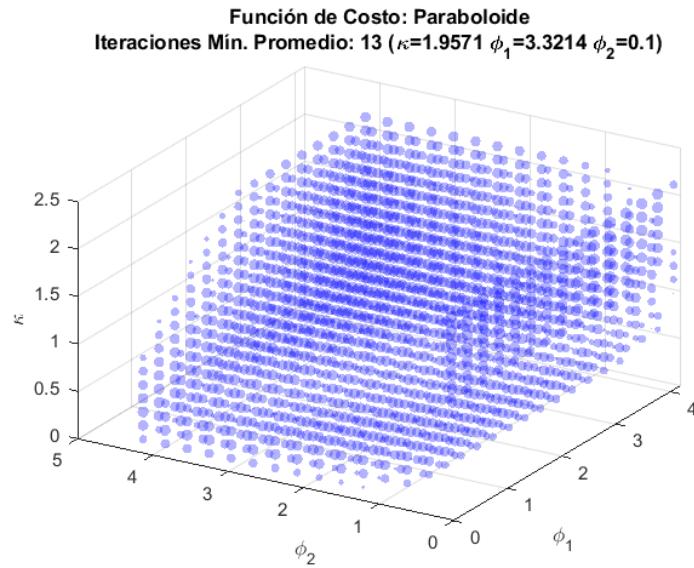


Figura 1: Nube de puntos obtenida para un paraboloid

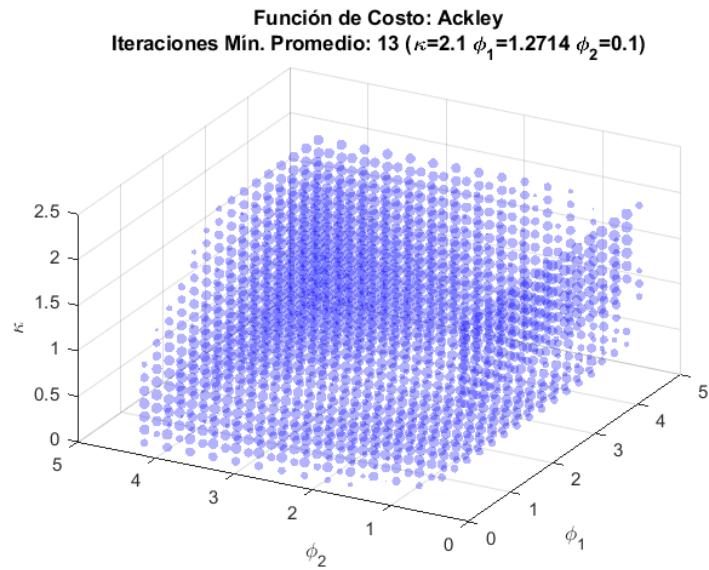


Figura 2: Nube de puntos obtenida para una función «Ackley»

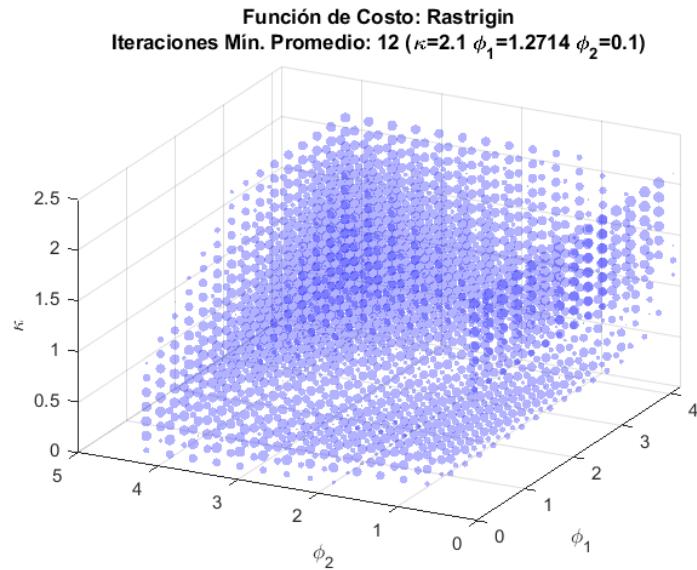


Figura 3: Nube de puntos obtenida para una función «Rastrigin»

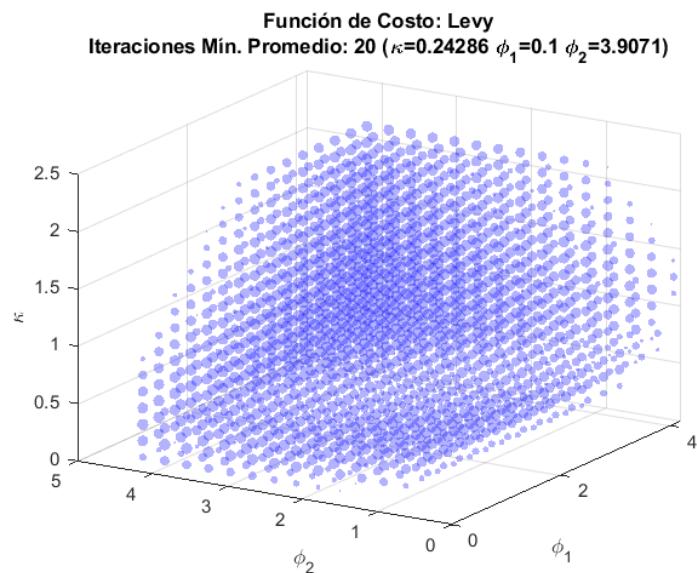


Figura 4: Nube de puntos obtenida para una función «Levy Lvl. 4»

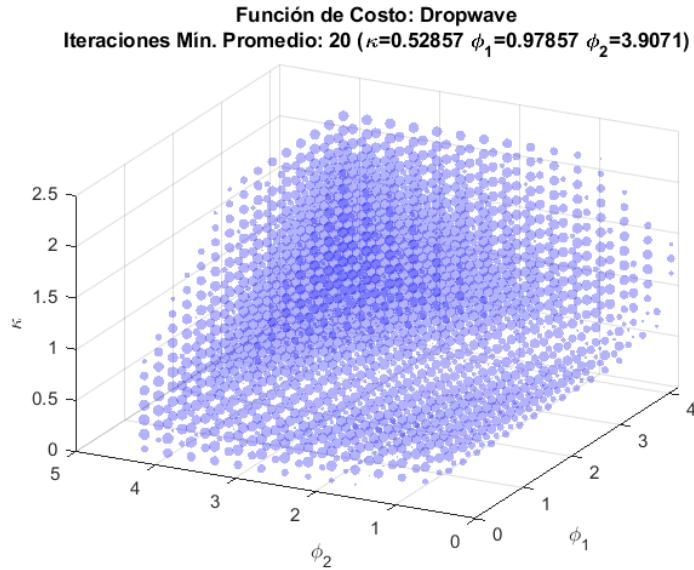


Figura 5: Nube de puntos obtenida para una función «Dropwave»

Cabe mencionar que se evitó iniciar el barrido de parámetros en 0 ya que, bajo estas condiciones, el algoritmo parecía converger en su segunda iteración. Esto se debía a que los parámetros, con estos valores, hacían que $C_1 = C_2 = \omega = 0$. Debido a que estas constantes multiplican a su vez a cada uno de los componentes de $V(t+1)$, la velocidad de la partícula se vuelve nula y la posición permanece estática. Dado que uno de los criterios de convergencia programados consistía en presentar un cambio en la posición inferior a 0.01 unidades, el algoritmo se detenía en su segunda iteración (Ya que la primera consistía en la inicialización).

Bibliografía

- [1] Maurice Clerc and James Kennedy. The particle swarm - explosion, stability and convergence in a multidimensional complex space. *Transactions on Evolutionary Computation*, 6(1):58–73, 2002.
- [2] Russell Eberhart and James Kennedy. A new optimizer using particle swarm theory. *Sixth International Symposium on Micro Machine and Human Science*, pages 39–43, 1995.

- [3] E. García-Gonzalo and J.L. Fernández-Martínez. A brief historical review of particle swarm optimization (pso). *Journal of Bioinformatics and Intelligent Control*, 1:3–16, 2012.
- [4] S. Nefti-Meziani M. Wahab and A. Atyabi. A comprehensive review of swarm optimization algorithms. *PLOS One*, 2015.
- [5] C. W. Reynolds. Flocks, herds and schools: A distributed behavioral model. *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, pages 25–34, 1987.
- [6] Towards Data Science. Introduction to optimization with genetic algorithms, 2018.