Vol.14 Núm.2

ISSN 2007-5448

RECIBE

Revista electrónica

DE COMPUTACIÓN, INFORMÁTICA, BIOMÉDICA Y ELECTRÓNICA

Índice

Computación e Informática

Electrónica

Búsqueda de patrones basada en trayectorias extraídas de la respuesta de sistemas de segundo orden.

Pattern search based on trajectories extracted from second-order system responses.

Jesus Edgar Elizondo Nuñez[1]
jesus.elizondo5693@alumnos.udg.mx

Carlos Guzman1
carlos.guzman8813@alumnos.udg.mx

Elivier Reyes1
elivier.reyes8810@alumnos.udg.mx

Hector Escobar1
hector.11294@gmail.com

Alberto Luque1
alberto.lchang@academicos.udg.mx

[1] Universidad de Guadalajara

Resumen: Recientemente, en la literatura se han introducido varios esquemas metaheurísticos nuevos. Aunque todos estos enfoques consideran fenómenos muy diferentes como metáforas, los patrones de búsqueda utilizados para explorar el espacio de búsqueda son muy similares. Por otro lado, los sistemas de segundo orden son modelos que presentan diferentes comportamientos temporales según el valor de sus parámetros. Tales comportamientos temporales pueden concebirse como patrones de búsqueda con múltiples comportamientos y configuraciones simples. En este artículo, se presentan un conjunto de nuevos patrones de búsqueda para explorar eficientemente el espacio de búsqueda. Estos emulan la respuesta de un sistema de segundo orden. El conjunto propuesto de patrones de búsqueda se ha integrado como una estrategia completa de búsqueda, llamada Algoritmo de Segundo Orden (SOA), para obtener la solución global de problemas de optimización complejos. Para analizar el rendimiento del esquema propuesto, se ha comparado en un conjunto de problemas representativos de optimización, que incluyen formulaciones de referencia multimodales, unimodales e híbridas. Los resultados numéricos demuestran que el método SOA propuesto exhibe un rendimiento notable en términos de precisión y altas tasas de convergencia.

Palabras clave: métodos metaheurísticos; patrones de búsqueda; sistemas de segundo orden; métodos evolutivos.

Summary:

Recently, several new metaheuristic frameworks have been introduced in the literature. Although these approaches rely on very different metaphors, the search patterns used to explore the search space are quite similar. On the other hand, second-order systems are models that exhibit various temporal behaviors depending on the values of their parameters. These temporal behaviors can be interpreted as search patterns with multiple behaviors and simple configurations. This paper presents a set of new search patterns designed to efficiently explore the search space, which emulate the response of a second-order system. The proposed set of search patterns is integrated into a complete search strategy called the **Second-Order Algorithm (SOA)** to find the global solution to complex optimization problems. To evaluate the performance of the proposed scheme, it has been compared across a set of representative optimization problems, including multimodal, unimodal, and hybrid benchmark formulations. Numerical results demonstrate that the proposed SOA method exhibits remarkable performance in terms of accuracy and high convergence rates.

Keywords: metaheuristic methods; search patterns; second-order systems; evolutionary methods.

1. Introducción

Los algoritmos metaheurísticos se refieren a esquemas de optimización genéricos que emulan el funcionamiento de diferentes procesos naturales o sociales. En los enfoques metaheurísticos, la estrategia de optimización se lleva a cabo mediante un conjunto de agentes de búsqueda. Cada agente mantiene una posible solución al problema de optimización y se genera inicialmente considerando una solución factible aleatoria. Una función objetivo determina la calidad de la solución de cada agente. Utilizando los valores de la función objetivo, en cada iteración, se modifica la posición de los agentes de búsqueda, empleando un conjunto de patrones de búsqueda que regulan sus movimientos dentro del espacio de búsqueda. Estos patrones de búsqueda son modelos abstractos inspirados en procesos naturales o sociales (Cuevas et al., 2020). Estos pasos se repiten hasta que se alcanza un criterio de parada. Los esquemas metaheurísticos han confirmado su supremacía en diversas aplicaciones del mundo real en circunstancias donde los métodos clásicos no pueden ser adoptados.

En esencia, no existe una clasificación clara de los métodos metaheurísticos. A pesar de esto, se han propuesto varias categorías que consideran diferentes criterios, como la fuente de inspiración, el tipo de operadores o la cooperación entre los agentes. En relación con la inspiración, los algoritmos metaheurísticos inspirados en la naturaleza se clasifican en tres categorías: basados en la evolución, basados en enjambres y basados en la física. Los enfoques basados en la evolución corresponden a las estrategias de búsqueda más consolidadas que utilizan elementos de evolución como operadores para producir patrones de búsqueda. En consecuencia, operaciones como la reproducción, la mutación, la recombinación y la selección se utilizan para generar patrones de búsqueda durante sus operaciones. Los ejemplos más representativos de técnicas basadas en la evolución incluyen las Estrategias Evolutivas (EE) (Beyer & Schwefel, 2002; Hansen, 2023; T, 1991), Algoritmos Genéticos (AG) (Tang et al., 1996), Differential Evolution (DE) (Storn & Price, 1997) y Self-Adaptative Differential Evolution (JADE) (Zhang & Sanderson, 2007). Las técnicas inspiradas en enjambres utilizan esquemas de comportamiento extraídos de la interacción colaborativa de diferentes animales o especies de insectos para producir una estrategia de búsqueda. Recientemente en la literatura se ha publicado un gran número de enfoques basados en enjambres. Entre los enfoques inspirados en enjambres más populares se encuentra el Algoritmo de Búsqueda de Cuervos (CSA) (Askarzadeh, 2016), Colonia Artificial de Abejas (ABC) (Karaboga et al., 2014), Algoritmo de Optimización por Enjambre de Partículas (PSO) (Kennedy & Eberhart, 1995; Marini & Walczak, 2015; Poli et al., 2007), Algoritmo de Luciérnagas (FA) (Yang, 2009), Cuckoo Search (CS) (Yang & Deb, 2009), Bat Algorithm (BA) (Yang, 2010), Gray Wolf Optimizer (GWO) (Mirjalili et al., 2014), Moth-flame optimization algorithm (MFO) (Mirjalili, 2015) por mencionar algunos. Los algoritmos metaheurísticos que consideran un esquema basado en la física utilizan modelos físicos simplificados para producir patrones de búsqueda para sus agentes. Algunos ejemplos de las técnicas basadas en la física más representativas incluyen el States of Matter Search (SMS) (Cuevas et al., 2014; Valdivia-Gonzalez et al., 2017), Simulated Annealing (SA) algorithm (Kirkpatrick et al., 1983; Rutenbar, 1989; Siddique & Adeli, 2016), Gravitational Search Algorithm (GSA) (Rashedi et al., 2009), Water Cycle Algorithm (WCA) (Eskandar et al., 2012), Big Bang-Big Crunch (BB-BC) (Erol & Eksin, 2006) y Electromagnetism-like Mechanism (EM) (Birbil & Fang, 2003). La Figura 1 muestra visualmente la taxonomía de la clasificación metaheurística. Aunque todos estos enfoques consideran fenómenos muy diferentes como metáforas, los patrones de búsqueda utilizados para explorar el espacio de búsqueda se basan exclusivamente en elementos espirales o modelos de

atracción (Kennedy & Eberhart, 1995; Marini & Walczak, 2015; Mirjalili et al., 2014; Poli et al., 2007; Yang, 2009, 2010). Bajo tales condiciones, el diseño de muchos métodos metaheurísticos se refiere a la configuración de un patrón de búsqueda reciclado que ha demostrado ser exitoso en enfoques anteriores para generar nuevos esquemas de optimización mediante una modificación marginal.

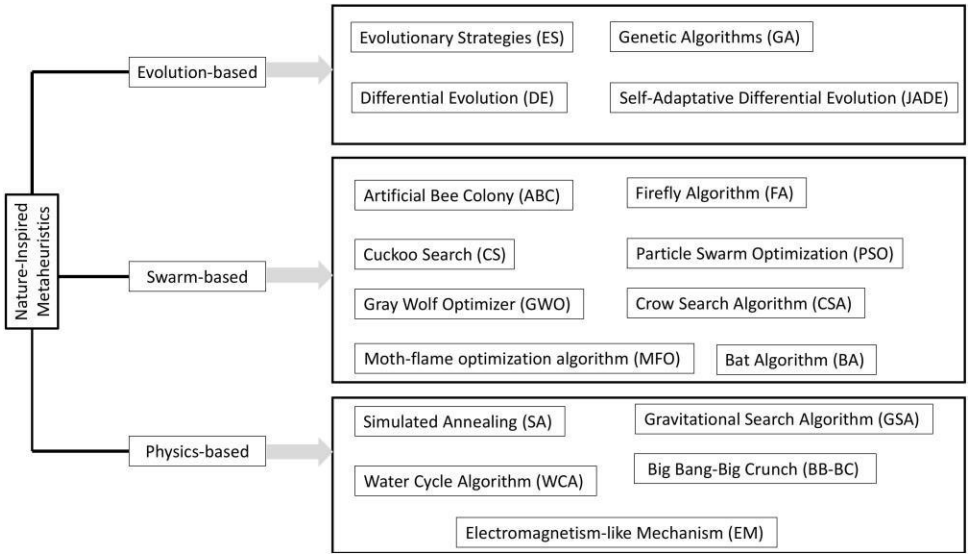


Figura 1. Taxonomía visual de los esquemas metaheurísticos inspirados en la naturaleza.

Por otro lado, el orden de una ecuación diferencial se refiere al grado más alto de derivada considerado en el modelo. Por lo tanto, un modelo cuya formulación de entrada-salida es una ecuación diferencial de segundo orden se conoce como un sistema de segundo orden (Zill, 2012). Uno de los elementos principales que hace que un modelo de segundo orden sea importante es su capacidad para presentar comportamientos muy diferentes, dependiendo de la configuración de sus parámetros. A través de sus distintos comportamientos, como oscilatorio, sub amortiguado o sobre amortiguado, un sistema de segundo orden puede mostrar respuestas temporales distintas (Haidekker, 2020). Estos comportamientos pueden observarse como trayectorias de búsqueda desde la perspectiva de los esquemas metaheurísticos. Por lo tanto, con sistemas de segundo orden, es posible generar movimientos oscilatorios dentro de una región determinada o construir patrones de búsqueda complejos alrededor de diferentes puntos o secciones del espacio de búsqueda.

En este artículo, se introduce un conjunto de nuevos patrones de búsqueda para explorar eficientemente el espacio de búsqueda. Estos patrones emulan la respuesta de un sistema de segundo orden. El conjunto propuesto de patrones de búsqueda se ha integrado como una estrategia completa de búsqueda, llamada Algoritmo de Segundo Orden (SOA), para obtener la solución global de problemas de optimización complejos. Para analizar el rendimiento del esquema propuesto, se ha comparado en un conjunto representativo de problemas de optimización, que incluyen formulaciones multimodales, unimodales e híbridas. Los resultados competitivos demuestran los resultados prometedores de los patrones de búsqueda propuestos.

Las principales contribuciones de esta investigación se pueden expresar de la siguiente manera:

1. Se introduce un nuevo algoritmo de optimización basado en la física, denominado

SOA. Utiliza patrones de búsqueda obtenidos a partir de la respuesta de sistemas de segundo orden.

2. Se proponen nuevos patrones de búsqueda como alternativa a los conocidos en la literatura.

3. Se evalúa la significancia estadística, la velocidad de convergencia y la proporción de explotación-exploración de SOA en comparación con otros algoritmos metaheurísticos populares.

4. SOA supera a otros algoritmos competidores en dos conjuntos de problemas de optimización.

El resto de este documento está estructurado de la siguiente manera: Se presenta una breve introducción de los sistemas de segundo orden en la sección 2; en la sección 3, se discuten los patrones de búsqueda más importantes en los métodos metaheurísticos; en la sección 4, se definen los patrones de búsqueda propuestos; en la sección 5, se describe la medición de la exploración-explotación; en la sección 6, se introduce el esquema propuesto; en la sección 7 se presentan los resultados numéricos; en la sección 8, se discuten las principales características del enfoque propuesto; en la sección 9, finalmente, se extraen las conclusiones.

2. Sistemas de Segundo Orden

Un modelo cuya formulación de entrada R(s) - salida C(s) es una función de transferencia de lazo cerrado de segundo orden se conoce como un sistema de segundo orden. Uno de los elementos principales que hace que un modelo de segundo orden sea importante es su capacidad para presentar comportamientos muy diferentes según la configuración de sus parámetros. Un modelo de segundo orden genérico se puede formular bajo la siguiente expresión (Zill, 2012):

$$\frac{C(s)}{R(s)} = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \quad (1)$$

Donde ζ y ω_n representan la razón de amortiguamiento y ω_n la frecuencia natural, respectivamente, mientras que s simboliza el dominio de Laplace.

El comportamiento dinámico de un sistema se evalúa en términos de la respuesta temporal obtenida mediante una señal de paso unitario como entrada R(s). El comportamiento dinámico se define como la forma en que el sistema reacciona, tratando de alcanzar el valor de uno a medida que evoluciona el tiempo. El comportamiento dinámico del sistema de segundo orden se describe en términos de ζ y ωn (Haidekker, 2020). Suponiendo tales parámetros, el sistema de segundo orden presenta tres comportamientos diferentes: Subamortiguado ($0 < \zeta < 1$), críticamente amortiguado ($\zeta = 1$), y sobre amortiguado ($\zeta > 1$).

2.1. Comportamiento Subamortiguado ($0 < \zeta < 1$)

En este comportamiento, los polos (raíces del denominador) de la ecuación (1) son complejos

conjugados y se encuentran en la mitad izquierda del plano s. Bajo tales condiciones, la respuesta subamortiguada C_u(s) el sistema en el dominio de Laplace se puede caracterizar de la siguiente manera:

$$C_U(s) = \frac{\omega_n^2}{s(s+\zeta\omega_n)^2 + \omega_n^2(1-\zeta^2)} \quad (2)$$

Aplicando operaciones de fracciones parciales y la transformada inversa de Laplace, se obtiene la respuesta temporal que describe el comportamiento subamortiguado C_u(t) como se indica en la ecuación (3):

$$c_U(t) = 1 - \frac{e^{-\zeta\omega_n t}}{\sqrt{1-\zeta^2}} \sin\left(\omega_n\sqrt{1-\zeta^2} + \tan^{-1}\left(\frac{\sqrt{1-\zeta^2}}{\zeta}\right)\right) \quad (3)$$

Si ζ = 0, se presenta un caso especial en el cual la respuesta temporal del sistema es oscilatoria. La salida de estos comportamientos se visualiza en la Figura 2 para los casos de ζ = 0, ζ = 0.2, ζ = 0.5 y ζ = 0.707. En el comportamiento subamortiguado, la respuesta del sistema comienza con una alta aceleración. Por lo tanto, la respuesta produce un sobrepaso que supera el valor de uno. El tamaño del sobrepaso depende inversamente del valor de ζ.

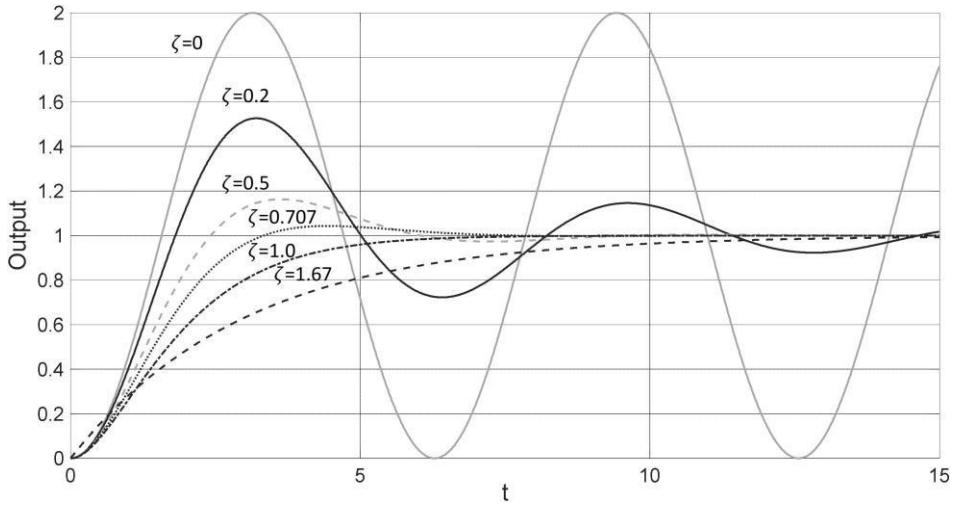


Figura 2. Respuestas temporales del sistema de segundo orden considerando sus diferentes comportamientos: Sub amortiguado (0 < ζ < 1), críticamente amortiguado (ζ = 1), y sobre amortiguado (ζ > 1).

2.2. Comportamiento críticamente amortiguado (ζ = 1)

En el comportamiento críticamente amortiguado, los dos polos de la función de transferencia de la ecuación (1) presentan números reales con el mismo valor. Por lo tanto, la respuesta del sistema críticamente amortiguado C_c(s) en el dominio de Laplace puede ser descrita como:

$$C_C(s) = \frac{\omega_n^2}{s(s+\omega_n)^2} \quad (4)$$

Aplicando la transformada inversa de Laplace a (4), la respuesta temporal al comportamiento criticamente amortiguado C_t(s) es representado por el modelo:

$$c_C(t) = 1 - e^{-\omega n t}(1 + \omega_n t) \tag{5}$$

En este tipo de comportamiento, la respuesta del sistema presenta un patron similar a lo sistemas de primer orden. El cual, alcanza el valor de uno sin experimentar ningún excedente. La salida de este sistema esta representado visualmente en la Figura 2.

2.3. Comportamiento Sobreamortiguado (ζ > 1)

En este caso, los polos de la función de transferencia son numeros reales con con diferente valor. La respusta de $C_o(s)$ en el dominio de Laplace es modelada como se indica en (6)

$$C_O(s) = \frac{\omega_n^2}{s(s + \zeta\omega_n + \omega_n\sqrt{\zeta^2 - 1})(s + \zeta\omega_n - \omega_n\sqrt{\zeta^2 - 1})} \tag{6}$$

Por su parte, la ecuación (7) representa la respuesta temporal del sistema sobreamortiguado tras aplicar la transformada inversa de Laplace a (6)

$$c_O(t) = 1 - e^{-(\zeta - \sqrt{\zeta^2 - 1})\omega_n t} \tag{7}$$

En un comportamiento sobreamortiguado, el sistema reacciona lentamente hasta alcanzar el valor de uno. La desaceleración de la respuesta depende de ζ, donde un mayor valor representa una repuesta más lenta. La respuesta de este comportamiento e representada en la figura 2 con un valor de ζ=1.67.

3. Patrones de búsqueda en Metaheurísticos

Un patrón de búsqueda es un conjunto de movimientos producidos por una regla o modelo con el fin de examinar soluciones prometedoras en el espacio de búsqueda. La generación de patrones de búsqueda eficientes para la correcta exploración del valor óptimo dentro del espacio de búsqueda puede ser complicada, especialmente cuando existen múltiples óptimos locales. Recientemente, varios esquemas metaheurísticos han sido introducidos en la literatura. Aunque todos estos enfoques consideran fenómenos muy diferentes como metáforas, los patrones de búsqueda utilizados para explorar el espacio de búsqueda son muy similares.

La exploración y la explotación corresponden a las características más importantes de un patrón de búsqueda. La exploración se refiere a la capacidad de un patrón de búsqueda para examinar un conjunto de soluciones distribuidas en áreas distintas del espacio de búsqueda. Por otro lado, la explotación representa la capacidad del patrón de búsqueda para mejorar la precisión y calidad de las soluciones existentes evaluando la localidad de dichas soluciones. La combinación de ambos mecanismos en un patrón de búsqueda es crucial para lograr el éxito al resolver un problema de optimización particular.

Los problemas de optimización, desde la perspectiva Metaheurística, inicializan una población $P^k(\{x_1^k, \ldots, x_N^k\})$ de N soluciones candidatas (individuos) que evolucionan tras cada generación, desde un punto inicial $k = 1$, hasta un número determinado $k = Maxgen$. Cada individuo

$x_i^k (i \in [1, \dots, N])$ perteneciente a la población, corresponde a un elemento dimensional-d $\{x_{i,1}^k, \dots, x_{i,d}^k\}$, el cual simboliza las variables de decisión involucradas en el problema de optimización.

En cada generación, el patrón de búsqueda actúa sobre cada individuo de la población P^k, para producir una nueva población P^{k+1}. La calidad de cada individuo x_i^k se evalúa en con respecto a la función objetivo $J(x_1^k)$, cuyo valor representa que tan apta es la solución x_i^k. Conforme evoluciona el proceso metaheurístico, se conservan los mejores individuos $\boldsymbol{b}$ $\{b_1, \dots, b_d\}$, dado que estos representan las mejores soluciones encontradas hasta el momento.

En general, los patrones de búsqueda actúan sobre cada individuo x_i^k, usando como referencia los mejores elementos almacenados en $\boldsymbol{b}$**.** Posteriormente, y, dependiendo del modelo metaheurístico empleado, un conjunto de reglas de movimiento modifica la posición de x_i^k hasta que la posición $\boldsymbol{b}$ ha sido alcanzada. La idea detrás de este mecanismo es examinar las soluciones que se encuentran en la trayectoria de x_i^k a $\boldsymbol{b}$, con el objetivo de encontrar mejores soluciones que las pertenecientes a al actual valor de $\boldsymbol{b}$. Los patrones de búsqueda para generar las trayectorias de $x^k{}_i$ a $\boldsymbol{b}$ difieren del modelo empleado.

Dos de los modelos de búsqueda más populares son las trayectorias de “atracción” y “espiral”. Los modelos de atracción generan movimientos de atracción x_i^k a $\boldsymbol{b}$. Los modelos de atracción son empleados ampliamente por una extensa cantidad de métodos metaheurísticos tales como PSO (Kennedy & Eberhart, 1995; Marini & Walczak, 2015; Poli et al., 2007), FA (Yang, 2009), CS (Yang & Deb, 2009), BA (Yang, 2010), GSA (Rashedi et al., 2009), EM (Birbil & Fang, 2003), y DE (Storn & Price, 1997). Por su parte, los modelos de espiral producen una trayectoria en espiral alrededor de las mejores soluciones pertenecientes a $\boldsymbol{b}$. Algunos de los modelos que emplean una trayectoria en espiral son WOA y GWO. La Figura 3, muestra las trayectorias que se generan con los modelos de atracción y de espiral.

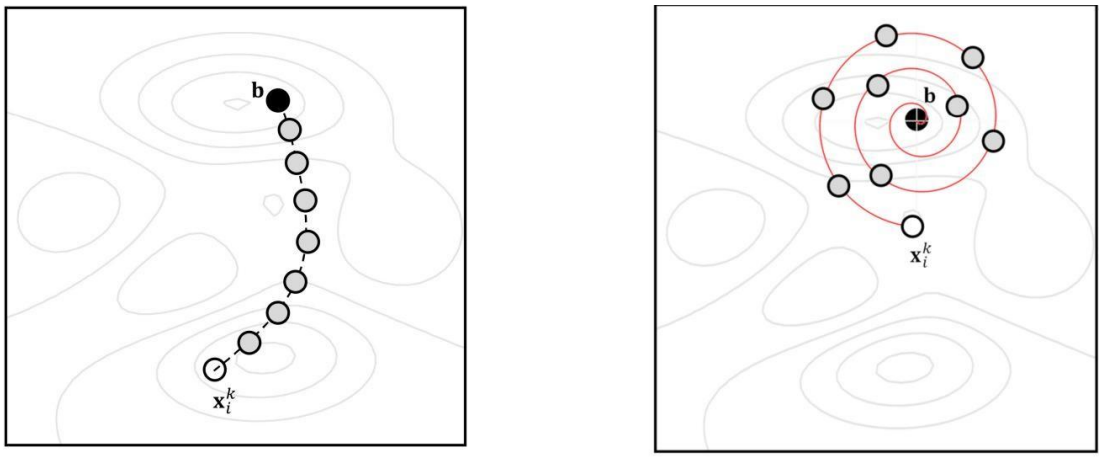


Figura 3. Trayectorias producidas por a) atracción, y b) espiral.

4. Patrón de búsqueda propuesto

En este artículo, se presentan un conjunto de nuevos patrones de búsqueda para explorar el espacio de búsqueda de manera eficiente. Estos patrones emulan la respuesta de un sistema de segundo

orden. El conjunto de patrones de búsqueda propuesto se emplea como una estrategia de búsqueda completa con la finalidad de obtener la solución global de problemas de optimización complejos. Dado que el esquema propuesto se basa en la respuesta de sistemas de segundo orden, puede considerarse como un algoritmo basado en la física. En este enfoque, la respuesta temporal del sistema de segundo orden se utiliza para generar la trayectoria desde la posición de $x^k_i = \{x^k_{i,1}, \dots, x^k_{i,d}\}$ hasta $\boldsymbol{b}\{b_1, \dots, b_d\}$. Con el uso de estos modelos, es posible generar trayectorias más complejas que permiten una mejor evaluación del espacio de búsqueda. Ante tales condiciones, se consideran las tres respuestas de un sistema de segundo orden para producir patrones de búsqueda distintos. Estos son el subamortiguado, críticamente amortiguado y sobreamortiguado, modelados por las expresiones:

$$x^k_{i,j} = \left(1 - \frac{e^{-\zeta\omega_n k}}{\sqrt{1-\zeta^2}} \sin\left(\omega_n\sqrt{1-\zeta^2} + \tan^{-1}\left(\frac{\sqrt{1-\zeta^2}}{\zeta}\right)\right)\right)(b_j - x^k_{i,j}) \tag{8}$$

$$x^k_{i,j} = (1 - e^{-\omega n t}(1 + \omega_n t))(b_j - x^k_{i,j}) \tag{9}$$

$$x^k_{i,j} = (1 - e^{-(\zeta - \sqrt{\zeta^2 - 1})\omega_n k})\,(b_j - x^k_{i,j}) \tag{10}$$

Donde $i(\in [1, N])$ corresponden a los agentes de búsqueda, mientras que $j(\in [1, N])$ simboliza las variables de decisión o dimensión del problema. Dado que el comportamiento de cada patrón de búsqueda depende del valor de ζ, es fácil combinar elementos para producir trayectorias interesantes. En la Figura 4 se muestran algunos ejemplos de trayectorias producidas por diferentes valores de ζ.

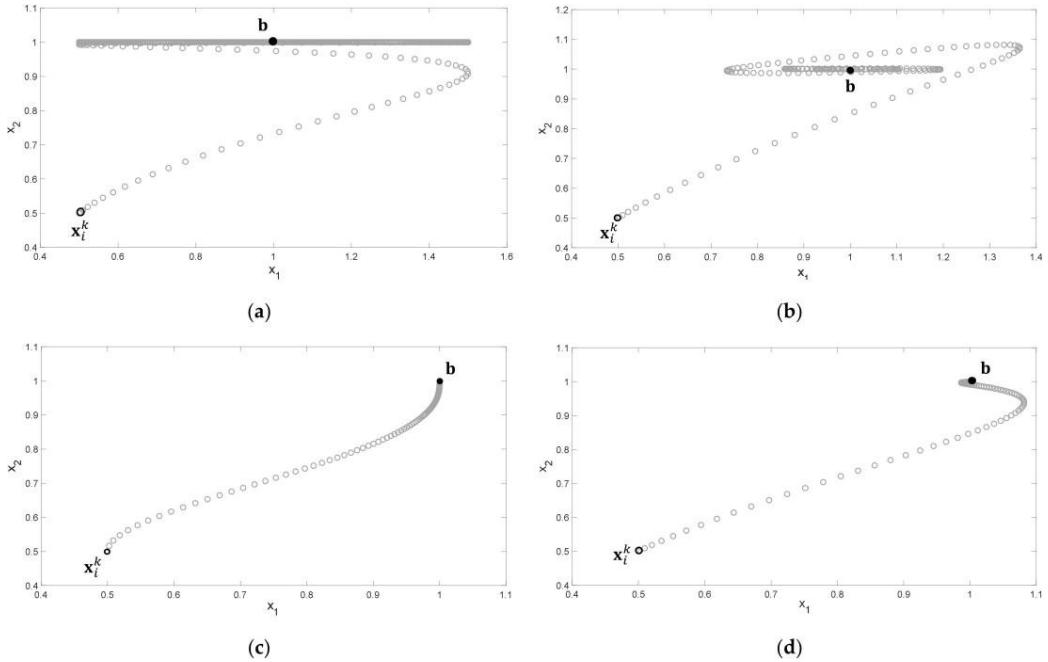


Figura 4. Algunos ejemplos de las trayectorias producidas usando diferentes valores de ζ, a) $x^k_{i,1} \leftarrow \zeta = 0$, y $x^k_{i,2} \leftarrow \zeta = 1$, b) $x^k_{i,1} \leftarrow \zeta = 0.1$, y $x^k_{i,2} \leftarrow \zeta = 0.5$, c) $x^k_{i,1} \leftarrow \zeta = 1$, y $x^k_{i,2} \leftarrow \zeta = 1.67$, d) $x^k_{i,1} \leftarrow \zeta = 0.5$, y $x^k_{i,2} \leftarrow \zeta = 1$

En dicha Figura, se asume un caso de dos dimensiones (d=2), donde la posición inicial de x_i^k es (0.5, 0.5), y la posición final o mejor posición es (1,1). La Figura 4a, muestra el escenario de $x_{i,1}^k \leftarrow \zeta = 0$ y $x_{i,2}^k \leftarrow \zeta = 1$. Por su parte, la Figura 4b muestra el caso de $x_{i,1}^k \leftarrow \zeta = 0.1$, y $x_{i,2}^k \leftarrow \zeta = 0.5$. Mientras que la Figura 4c presenta el caso $x_{i,1}^k \leftarrow \zeta = 1$, y $x_{i,2}^k \leftarrow \zeta = 1.67$. Finalmente, la Figura 4d muestra el escenario $x_{i,1}^k \leftarrow \zeta = 0.5$, y $x_{i,2}^k \leftarrow \zeta = 1$. De estas figuras, se puede apreciar que las respuestas de segundo orden pueden producir una gran cantidad de trayectorias complejas, que incluye a gran parte de otros patrones de búsqueda encontrados en la literatura. En todos estos casos (a)-(d), el valor asignad a ω_n ha sido 1.

5. Equilibrio entre exploración y explotación

Los métodos metaheurísticos emplean un conjunto de agentes de búsqueda para examinar el espacio de búsqueda con el objetivo de identificar una solución satisfactoria para una formulación de optimización. En los esquemas metaheurísticos, los agentes de búsqueda que presentan los mejores valores de fitness tienden a regular el proceso de búsqueda, produciendo una atracción hacia ellos. En estas condiciones, a medida que evoluciona el proceso de optimización, la distancia entre los individuos disminuye, al tiempo que se acentúa el efecto de explotación. Por otro lado, cuando la distancia entre individuos aumenta, las características del proceso de exploración son más evidentes.

Para calcular la distancia relativa entre los individuos (aumento y disminución), se utiliza un indicador de diversidad conocido como índice de diversidad dimensional (Morales-Castañeda et al., 2020). Según este enfoque, la diversidad se formula de la siguiente manera:

$$Div_j = \frac{1}{N}\sum_{i=1}^{N}|median(x^j) - x_{i,j}| Div = \frac{1}{d}\sum_{j=1}^{d} Div_j \tag{11}$$

donde $median(x^j)$ simboliza la mediana de la dimensión j de todos los agentes de búsqueda. x_i, j representa la decisión variable j del individuo i. N es el número de individuos en la población P^k mientras que d corresponde al número de dimensiones de la formulación de optimización. La diversidad Div_j (de la j-ésima dimensión) evalúa la distancia relativa entre la variable j de cada individuo y su valor mediano. La diversidad completa Div (de toda la población) corresponde a la diversidad promedio en cada dimensión. Ambos elementos Div_j y Div se calculan en cada iteración.

Habiendo evaluado los valores de diversidad, el nivel de exploración y explotación se puede calcular como el porcentaje del tiempo que una estrategia de búsqueda invierte explorando o explotando en términos de sus valores de diversidad. Estos porcentajes se calculan en cada iteración mediante los siguientes modelos,

$$XLP\% = (\frac{Div}{Div_{max}}) \times 100 XPT\% = (\frac{|Div - Div_{max}|}{Div_{max}}) \times 100 \tag{12}$$

Donde Div_{max} simboliza el valor máximo de diversidad obtenido durante el proceso de optimización. El porcentaje de exploración $XPL\%$ corresponde al tamaño de la exploración como la tasa entre Div y Div_{max}. Por otro lado, el porcentaje de explotación $XPT\%$ simboliza el nivel de explotación. $XPT\%$

se calcula como el porcentaje complementario a $XPL\%$ ya que la diferencia entre Div_{max} y Div se genera debido a la concentración de individuos.

6. Algoritmo metaheurístico propuesto

El conjunto de patrones de búsqueda basados en los sistemas de segundo orden se ha integrado como una estrategia de búsqueda completa para obtener la solución global de problemas complejos de optimización. En esta sección se describe el método metaheurístico completo, llamado Algoritmo de Segundo Orden (SOA).

El esquema considera cuatro etapas diferentes: (A) Inicialización, (B) generación de trayectoria, (C) reinicio de elementos defectuosos y (D) evitar mecanismo de convergencia prematura. Los pasos (B)–(D) se ejecutan secuencialmente hasta que se alcanza un criterio de parada. La Figura 5 muestra el diagrama de flujo del método metaheurístico completo.

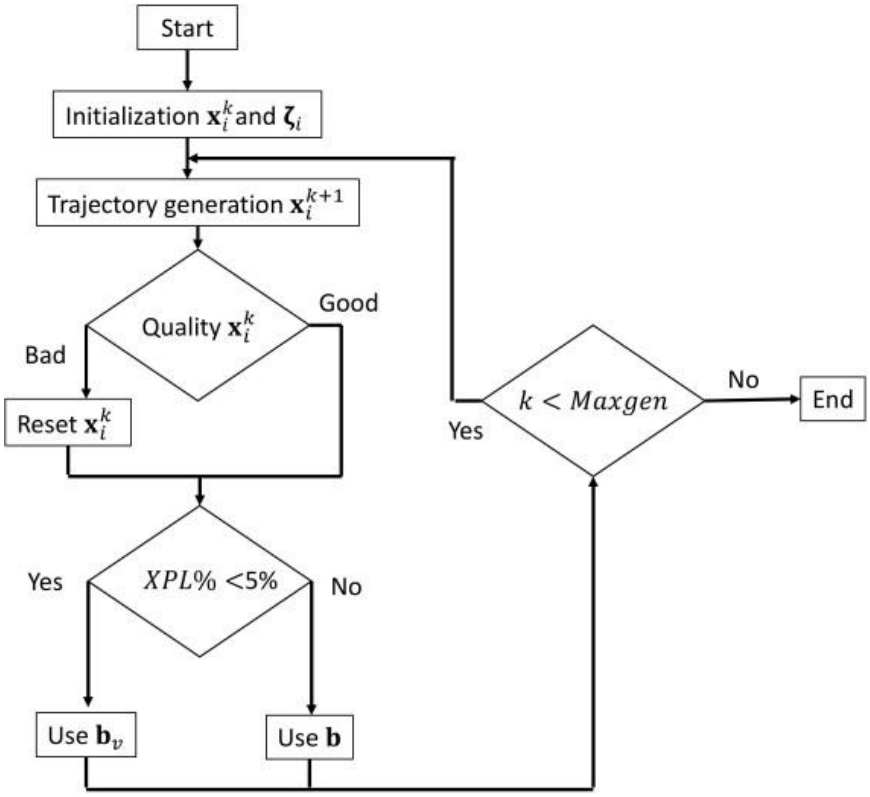


Figura 5. Diagrama de flujo del método metaheurístico propuesto basado en la respuesta de sistemas de segundo orden.

6.1. Inicialización

En la primera iteración $k = 0$, se produce aleatoriamente una población P^0 de N agentes $\{x_1^0, \cdots, x_N^0\}$ considerando la siguiente ecuación:

$$x_{i,j}^0 = rand \cdot (b_j^{high} - b_j^{low}) + b_j^{low} = 1,2, \cdots, N; j = 1, \cdots, d \quad (13)$$

Donde b_j^{high} y b_j^{low} son los límites de la variable de decisión j y $rand$ es un número aleatorio distribuido uniformemente entre [0,1].

A cada individuo x_i de la población se le asigna un vector $\zeta_i = \{\zeta_{i,1}, \cdots, \zeta_{i,d}\}$ cuyos elementos $\zeta_{i,j}$ determinan el comportamiento de la trayectoria de cada j-ésima dimensión. Inicialmente, cada elemento $\zeta_{i,j}$ se establece en un valor aleatorio entre [0,2]. Bajo este intervalo, todos los comportamientos de segundo orden son posibles: subamortiguado $(0 < \zeta < 1)$, críticamente amortiguado $(\zeta = 1)$ y sobreamortiguado $(\zeta > 1)$.

6.2. Generación de Trayectoria

Una vez inicializada la población, se obtiene el mejor elemento de la población $\boldsymbol{b}$. Luego, la nueva posición x_i^{k+1} de cada agente x_i^k se calcula como una trayectoria generada por un sistema de segundo orden. Una vez determinadas todas las nuevas posiciones en la población P^k, también se define el mejor elemento $\boldsymbol{b}$.

6.3. Restablecimiento de Elementos Defectuosos

A cada agente x_i^k se le permite moverse en su propia trayectoria durante diez iteraciones. Después de diez iteraciones, si el agente de búsqueda x_i^k mantiene el peor desempeño en términos de la función de aptitud, se reinicializa tanto en la posición como en su vector ζ_i. En tales condiciones, el agente de búsqueda estará en otra posición y con la capacidad de realizar otro tipo de comportamiento de trayectoria.

6.4. Evadir el Mecanismo de Convergencia Prematura

Si el porcentaje de exploración $XPL\%$ es inferior al 5%, el mejor valor $\boldsymbol{b}$ se reemplaza por el mejor valor virtual $\boldsymbol{b}_v$. El elemento $\boldsymbol{b}_v$ se calcula como el valor promedio de los cinco mejores individuos de la población. La idea detrás de este mecanismo es identificar una nueva posición para generar diferentes trayectorias que eviten que el proceso de búsqueda quede atrapado en un óptimo local.

7. Resultados experimentales

Para evaluar los resultados del algoritmo SOA propuesto, se ha llevado a cabo un conjunto de experimentos. Dichos resultados se han comparado con los obtenidos por la Colonia Artificial de Abejas (ABC) (Karaboga et al., 2014), la Estrategia de Evolución de Adaptación de la Matriz de Covarianza (CMAES) (Hansen, 2023), el Algoritmo de Búsqueda de Cuervos (CSA) (Askarzadeh, 2016), la Evolución Diferencial (DE) (Storn & Price, 1997), el Algoritmo de Optimización de Polilla (MFO) (Mirjalili, 2015) y la Optimización de Enjambre de Partículas (PSO) (Kennedy & Eberhart, 1995), que se consideran los esquemas metaheurísticos más populares en muchos estudios de optimización (Boussaïd et al., 2013). Para la comparación, todos los métodos se han configurado de acuerdo con sus parámetros publicados.

En nuestro análisis, el tamaño de la población N se ha fijado en 50 agentes de búsqueda. El número máximo de iteraciones (Maxgen) para todas las funciones se ha fijado en 1000. Este criterio de parada se ha decidido para mantener la compatibilidad con trabajos similares publicados en la literatura (Han et al., 2014; Meng & Pan, 2016). Para evaluar los resultados, se consideran tres

indicadores diferentes: La media de las mejores soluciones hasta el momento (AB), la mediana de las mejores soluciones hasta el momento (MB) y la desviación estándar (SD) de las mejores soluciones hasta el momento. En el análisis, cada problema de optimización se resuelve 30 veces con cada algoritmo, por lo tanto se obtienen 30 resultados. A partir de todos estos valores, el valor medio de todas las mejores soluciones encontradas representa la solución media Best-so-far (AB).

Asimismo, se calcula la mediana de los 30 resultados para generar MB y se estima la desviación estándar de los 30 datos para obtener SD de las mejores soluciones hasta el momento. Los indicadores AB y MB corresponden a la precisión de las soluciones, mientras que SD a su dispersión y, por tanto, a la robustez del algoritmo.

7.1 Funciones multimodales

En esta subsección, el enfoque SOA se evalúa considerando 12 funciones unimodales (f1(x)-f12(x)) detalladas en la Tabla 1 del Apéndice A. Este tipo de funciones presentan superficies de optimización que implican múltiples óptimos locales. Por esta razón, estas funciones presentan más complicaciones en su solución. En este análisis, se examina el rendimiento del método SOA en comparación con ABC, CMAES, CSA, DE, MFO y PSO en términos de las funciones multimodales. Los experimentos se llevan acabo con funciones objetivo que operan en 30 dimensiones (d = 30).

		ABC	DE	CMAES	CSA	PSO	MFO	SOA
$f_1(x)$	AB	8.9132622	0.7932535	2.8976×10^{-19}	55.918504	0.2012803	27.983271	0.1119774
	MD	8.4392750	0.7993166	2.4779×10^{-19}	57.038263	4.1459×10^{-23}	25.365199	1.0714×10^{-10}
	SD	2.6748059	0.1378538	1.5343×10^{-19}	6.2032578	1.1022968	12.283254	0.2272439
$f_2(x)$	AB	2	2	2	1,897,783.3	27.4	2	2
	MD	2	2	2	35,691.155	2	2	2
	SD	9.9512×10^{-12}	0	0	9,636,632.1	113.43495	0	0
$f_3(x)$	AB	2	2	2	3,620,834.4	34.723128	2	2
	MD	2	2	2	676,981.46	9	2	2
	SD	2.3308×10^{-11}	0	0	7,265,352.7	113.36672	0	0
$f_4(x)$	AB	0.1371551	0.002	1.7942×10^{-6}	0.0862919	2.2285×10^{-8}	5.5194×10^{-10}	1.164×10^{-11}
	MD	0.1349076	0.01	0	0.0892256	0	0	7.7118×10^{-12}
	SD	0.0399861	0.123	5.4833×10^{-6}	0.0213332	1.2206×10^{-7}	3.0231×10^{-9}	1.0995×10^{-11}
$f_5(x)$	AB	13,781,291	1,331,987.7	22,307.195	44,274,761	82.539625	85.756149	71.964984
	MD	13,876,263	1,365,502.1	72.377516	46,153,728	81.698488	85.665615	72.362277
	SD	3,237,147.7	306,385.34	50,923.637	10,118,180	7.1916726	3.2857088	0.9929591
$f_6(x)$	AB	1.152×10^{85}	5.850×10^{81}	1.812×10^{83}	6.429×10^{83}	1.397×10^{81}	3.0883×10^{81}	1.0051×10^{81}
	MD	4.622×10^{84}	3.405×10^{81}	7.928×10^{82}	2.939×10^{83}	5.977×10^{80}	7.9607×10^{80}	4.901×10^{80}
	SD	1.685×10^{85}	7.33×10^{81}	3.047×10^{83}	7.551×10^{83}	2.197×10^{81}	5.6651×10^{81}	1.9457×10^{81}
$f_7(x)$	AB	30.033333	30	30	58.766666	30	33.633333	30
	MD	30	30	30	59	30	30	30
	SD	0.18257419	0	0	1.77498583	0	5.4550409	0
$f_8(x)$	AB	9.2	8.0666666	1.0666666	19,543.266	0.0333333	2000.0333	0
	MD	9	8	0	19,797	0	0	0
	SD	2.5784250	1.9464084	3.1941037	2077.7459	0.1825741	4842.3277	0
$f_9(x)$	AB	−745.05202	−1125.4815	−1127.8626	−725.09353	−1071.7869	−1031.2617	−1146.3478
	MD	−743.4462	−1174.9722	−1132.5748	−719.10777	−1068.9596	−1033.6178	−1145.2467
	SD	25.137593	78.706	25.809999	25.815652	34.055847	34.244188	10.928362
$f_{10}(x)$	AB	110,282.54	665,278.86	−4930	1,170,939.0	45,556.260	222,833.73	−501.79356
	MD	96,461.061	673,449.49	−4930	1,126,234.2	5076.8152	71,582.051	−332.82466
	SD	44,933.417	129,147.27	3.7318×10^{-9}	159,175.64	75,990.880	305,159.37	663.92006
$f_{11}(x)$	AB	−18.26109	−26.056561	−29.6576	−16.504756	−28.367666	−28.863589	−30
	MD	−18.131984	−26.092349	−29.9286	−16.183447	−28.14029	−29.070145	−30
	SD	1.6366873	0.5428906	0.466	1.16917142	1.5408536	1.2837415	0
$f_{12}(x)$	AB	1502.3129	369.60375	786.36819	519.17242	196.95838	261.52332	11.905761
	MD	1457.6865	368.82916	778.72369	465.93238	213.00730	252.76747	0.3841574
	SD	420.70611	35.389136	215.93893	228.69860	86.542862	106.52353	29.544101

Tabla 1. Resultados de minimización de funciones de referencia multimodales.

En la Tabla 1 se muestran los mejores resultados promediados (AB) considerando 30 experimentos independientes. También se presentan los valores medios (DM) y las desviaciones estándar (DE).

En la Tabla 1 se observa que el esquema SOA propuesto obtiene un mejor rendimiento que los algoritmos ABC, CMAES, CSA, DE, MFO y PSO en las funciones f1(x), f4(x), f5(x), f6(x), f8(x), f9(x), f10(x), f11(x) y f12(x). Sin embargo, los resultados de SOA son similares a los obtenidos por DE, CMAES y MFO en las funciones f2(x), f3(x) y f7(x).

8. Conclusiones

Un patrón de búsqueda es un conjunto de movimientos producidos por una regla o modelo, con el fin de examinar soluciones prometedoras del espacio de búsqueda. En este artículo, se introduce un conjunto de nuevos patrones de búsqueda para explorar el espacio de búsqueda de forma eficiente. Emulan la respuesta de un sistema de segundo orden. En estas condiciones, se consideran tres respuestas diferentes de un sistema de segundo orden para producir tres patrones de búsqueda distintos, como subamortiguado, críticamente amortiguado y sobreamortiguado. Estos conjuntos de patrones de búsqueda propuestos se han integrado como una estrategia de búsqueda completa, denominada Algoritmo de Segundo Orden (SOA), para obtener la solución global de problemas de optimización complejos.

La forma de los patrones de búsqueda permite equilibrar las capacidades de exploración y explotación atravesando eficientemente el espacio de búsqueda y evitando las regiones subóptimas. La eficacia de la SOA propuesta se ha evaluado mediante 20 funciones de referencia estándar. Los resultados demuestran la eficacia de los patrones de búsqueda generados en las problemáticas más complejas.

Para comparar el rendimiento del esquema SOA, también se han probado en el mismo entorno experimental muchas otras técnicas de optimización populares, como la colonia de abejas artificial (ABC), la estrategia de evolución de adaptación de la matriz de covarianza (CMAES), el algoritmo de búsqueda de cuervos (CSA), la evolución diferencial (DE), el algoritmo de optimización de llama de polilla (MFO) y la optimización de enjambre de partículas (PSO). Las futuras líneas de investigación incluyen temas como las capacidades multiobjetivo, la incorporación de mapas caóticos y la inclusión de procesos de aceleración para resolver otros problemas de optimización a escala real.

10. Referencias

Askarzadeh, A. (2016). A novel metaheuristic method for solving constrained engineering optimization problems: Crow search algorithm. Computers & Structures, 169, 1–12. https://doi.org/10.1016/j.compstruc.2016.03.001

Beyer, H.-G., & Schwefel, H.-P. (2002). Evolution strategies – A comprehensive introduction. Natural Computing, 1(1), 3–52. https://doi.org/10.1023/A:1015059928466

Birbil, Ş. İ., & Fang, S.-C. (2003). An Electromagnetism-like Mechanism for Global Optimization. Journal of Global Optimization, 25(3), 263–282. https://doi.org/10.1023/A:1022452626305

Boussaïd, I., Lepagnot, J., & Siarry, P. (2013). A survey on optimization metaheuristics. Information Sciences, 237, 82–117. https://doi.org/10.1016/j.ins.2013.02.041

Cuevas, E., Echavarría, A., & Ramírez-Ortegón, M. A. (2014). An optimization algorithm inspired by the States of Matter that improves the balance between exploration and exploitation. Applied Intelligence, 40(2), 256–272. https://doi.org/10.1007/s10489-013-0458-0

Cuevas, E., Gálvez, J., Avila, K., Toski, M., & Rafe, V. (2020). A new metaheuristic approach based on agent systems principles. Journal of Computational Science, 47, 101244. https://doi.org/10.1016/j.jocs.2020.101244

Erol, O. K., & Eksin, I. (2006). A new optimization method: Big Bang–Big Crunch. Advances in Engineering Software, 37(2), 106–111. https://doi.org/10.1016/j.advengsoft.2005.04.005

Eskandar, H., Sadollah, A., Bahreininejad, A., & Hamdi, M. (2012). Water cycle algorithm – A novel metaheuristic optimization method for solving constrained engineering optimization problems. Computers & Structures, 110–111, 151–166. https://doi.org/10.1016/j.compstruc.2012.07.010

Haidekker, M. A. (2020). Linear Feedback Controls: The Essentials. Elsevier.

Han, M., Liu, C., & Xing, J. (2014). An evolutionary membrane algorithm for global numerical optimization problems. Information Sciences, 276, 219–241. https://doi.org/10.1016/j.ins.2014.02.057

Hansen, N. (2023). The CMA Evolution Strategy: A Tutorial (arXiv:1604.00772). arXiv. https://doi.org/10.48550/arXiv.1604.00772

Karaboga, D., Gorkemli, B., Ozturk, C., & Karaboga, N. (2014). A comprehensive survey: Artificial bee colony (ABC) algorithm and applications. Artificial Intelligence Review, 42(1), 21–57. https://doi.org/10.1007/s10462-012-9328-0

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. Proceedings of ICNN'95 - International Conference on Neural Networks, 4, 1942–1948 vol.4. https://doi.org/10.1109/ICNN.1995.488968

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by Simulated Annealing. Science, 220(4598), 671–680. https://doi.org/10.1126/science.220.4598.671

Marini, F., & Walczak, B. (2015). Particle swarm optimization (PSO). A tutorial. Chemometrics and Intelligent Laboratory Systems, 149, 153–165. https://doi.org/10.1016/j.chemolab.2015.08.020

Meng, Z., & Pan, J.-S. (2016). Monkey King Evolution: A new memetic evolutionary algorithm and its application in vehicle fuel consumption optimization. Knowledge-Based Systems, 97, 144–157. https://doi.org/10.1016/j.knosys.2016.01.009

Mirjalili, S. (2015). Moth-flame optimization algorithm: A novel nature-inspired heuristic paradigm.

Knowledge-Based Systems, 89, 228–249. https://doi.org/10.1016/j.knosys.2015.07.006

Mirjalili, S., Mirjalili, S. M., & Lewis, A. (2014). Grey Wolf Optimizer. Advances in Engineering Software, 69, 46–61. https://doi.org/10.1016/j.advengsoft.2013.12.007

Morales-Castañeda, B., Zaldívar, D., Cuevas, E., Fausto, F., & Rodríguez, A. (2020). A better balance in metaheuristic algorithms: Does it exist? Swarm and Evolutionary Computation, 54, 100671. https://doi.org/10.1016/j.swevo.2020.100671

Poli, R., Kennedy, J., & Blackwell, T. (2007). Particle swarm optimization. Swarm Intelligence, 1(1), 33–57. https://doi.org/10.1007/s11721-007-0002-0

Rashedi, E., Nezamabadi-pour, H., & Saryazdi, S. (2009). GSA: A Gravitational Search Algorithm. Information Sciences, 179(13), 2232–2248. https://doi.org/10.1016/j.ins.2009.03.004

Rutenbar, R. A. (1989). Simulated annealing algorithms: An overview. IEEE Circuits and Devices Magazine, 5(1), 19–26. https://doi.org/10.1109/101.17235

Siddique, N., & Adeli, H. (2016). Simulated Annealing, Its Variants and Engineering Applications. International Journal on Artificial Intelligence Tools, 25(06), 1630001. https://doi.org/10.1142/S0218213016300015

Storn, R., & Price, K. (1997). Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces. Journal of Global Optimization, 11(4), 341–359. https://doi.org/10.1023/A:1008202821328

T, B. (1991). A Survey of Evolution Strategies. Proc. of Fourth Internal. Conf. on Genetic Algorithms. https://cir.nii.ac.jp/crid/1573950398979648512

Tang, K. S., Man, K. F., Kwong, S., & He, Q. (1996). Genetic algorithms and their applications. IEEE Signal Processing Magazine, 13(6), 22–37. https://doi.org/10.1109/79.543973

Valdivia-Gonzalez, A., Zaldívar, D., Fausto, F., Camarena, O., Cuevas, E., & Perez-Cisneros, M. (2017). A States of Matter Search-Based Approach for Solving the Problem of Intelligent Power Allocation in Plug-in Hybrid Electric Vehicles. Energies, 10(1), Article 1. https://doi.org/10.3390/en10010092

Yang, X.-S. (2009). Firefly Algorithms for Multimodal Optimization. In O. Watanabe & T. Zeugmann (Eds.), Stochastic Algorithms: Foundations and Applications (pp. 169–178). Springer. https://doi.org/10.1007/978-3-642-04944-6_14

Yang, X.-S. (2010). A New Metaheuristic Bat-Inspired Algorithm. In J. R. González, D. A. Pelta, C. Cruz, G. Terrazas, & N. Krasnogor (Eds.), Nature Inspired Cooperative Strategies for Optimization (NICSO 2010) (pp. 65–74). Springer. https://doi.org/10.1007/978-3-642-12538-6_6

Yang, X.-S., & Deb, S. (2009). Cuckoo Search via Lévy flights. 2009 World Congress on Nature &

Biologically Inspired Computing (NaBIC), 210–214. https://doi.org/10.1109/NABIC.2009.5393690

Zhang, J., & Sanderson, A. C. (2007). JADE: Self-adaptive differential evolution with fast and reliable convergence performance. 2007 IEEE Congress on Evolutionary Computation, 2251–2258. https://doi.org/10.1109/CEC.2007.4424751

Zill, D. G. (2012). A First Course in Differential Equations with Modeling Applications. Cengage Learning.

Funciones, un algoritmo, BigQuery y la ausencia de frameworks

Functions, an algorithm, BigQuery, and the absence of frameworks

Pedro Cano[1]
pedroc777@gmail.com

Resumen: En este artículo vamos explicar cómo solucionar el siguiente problema: realizar una inserción de datos en una tabla de BigQuery usando el lenguaje de programación Python, cuando no podemos usar frameworks, chatGPT, GitHub Copilot, etc., para obtener una solución.

La solución se centra en desarrollar una serie de funciones que nos permitan validar los tipos de datos de los campos de la tabla involucrados en la inserción, ordenar los datos a insertar y darles la estructura requerida según los tipos de datos presentes. Para esto, también mostramos el desarrollo de un algoritmo centrado en realizar dichos procesos.

Palabras clave: datos, campos, funciones, algoritmo, Python, BigQuery, SQL.

Abstract: In this article, we are going to explain how to solve the following problem: perform a data insertion in a BigQuery table using the Python programming language, when we cannot use frameworks, chatGPT, GitHub Copilot, etc., to obtain a solution.

The solution focuses on developing a series of functions that allow us to validate the data types of the table fields involved in the insertion, order the data to be inserted and give them the required structure according to the data types present. For this, we also show the development of an algorithm focused on performing such processes.

Keywords: data, fields, functions, algorithm, Python, BigQuery, SQL.

[1] Universidad Abierta y a Distancia de México

1. Introducción

En este escrito pretendemos mostrar cómo se hace la inserción de datos en una tabla de BigQuery usando el lenguaje de programación Python. Esto es algo se ha realizado cientos de veces (y se seguirá haciendo) en la industria, pero la peculiaridad en este caso es que la explicación mostrada toma como base una hipotética aplicación en la que no pueden usarse frameworks, recursos como chatGPT o GitHub Copilot, bibliotecas que nos permitan realizar esta acción y recepción de datos de manera convencional (la razón es que en este contexto, dichas herramientas no tienen la solución para nuestro problema, no están integradas como tal o no se cuenta con los recursos para acceder a ellas). Lo único que recibe esta aplicación es una cadena de texto que nosotros debemos manipular para lograr nuestro objetivo.

Ahora bien, para poder desarrollar este escrito, primero explicamos a detalle el problema, es decir, damos un contexto del problema planteado arriba. Posteriormente mencionamos algunas cuestiones relacionadas con la función que realiza la inserción en BigQuery y pasamos a dar cuenta de la solución planteada, la cual se compone de una serie de funciones programadas en Python que permiten obtener las estructuras deseadas por BigQuery/SQL.

La descripción de esta solución se compone de varias partes. La primera de ellas es una función cuya misión es validar los tipos de datos de los campos involucrados en la inserción mencionada (para saber si son STRING, INTEGER, BOOLEAN, etc.). Posteriormente, se explica cómo esta función se usa para ejecutar cualquiera de las siguientes dos opciones: 1) la función que ordena los valores a insertar cuando todos los campos de la inserción son de tipo STRING; 2) la función que ordena los valores cuando uno o varios de estos no son de tipo STRING.

Esta última función es un tanto más extensa, pues conlleva el desarrollo de un algoritmo que llamamos *búsqueda por saltos*, el cual, con la ayuda de listas, contadores y ciclos for, nos ayuda a ordenar y dar formato a los datos involucrados en cada inserción. Por este motivo, la explicación de esta función implica también la descripción del algoritmo en abstracto, el algoritmo en el código y un ejemplo de cómo se ejecuta cada iteración del mismo. Una vez que se terminan estas explicaciones, también se termina el escrito, pues después de estos ordenamientos sólo resta ejecutar la inserción que nos permite realizar la inserción.

La última cuestión que debemos aclarar antes de empezar a desarrollar nuestros objetivos, es BigQuery.

Esta (Google, 2023) es una herramienta desarrollada por Google basada en una arquitectura sin servidor que permite almacenar grandes cantidades de información en la nube y que usa SQL para realizar varias de las operaciones pertenecientes a una base de datos tradicional, como consultas, inserciones, actualizaciones, procedimientos almacenados y demás. En este aspecto, junto con el desarrollo del algoritmo mencionado y en el hecho de no poder usar herramientas que faciliten la solución mencionada, recae la importancia del escrito, pues por un lado usamos una herramienta que se encuentra en auge (por el incremento del cómputo en la nube), y por otro nos ponemos en un escenario en el que, por ciertos motivos, nos vemos imposibilitados para usar las herramientas mencionadas.

2. Explicación del problema

Supongamos un escenario: necesitamos insertar información en un proyecto de BigQuery (Jenn, 2022[i]) (de ahora en adelante abreviado como BQ) y necesitamos que dicha información se obtenga a partir de una interfaz gráfica de usuario. Dicho de otro modo: necesitamos que una tabla de BQ se alimente de la información que un usuario le proporciona a través de una aplicación móvil o de escritorio mediante la escritura de los datos. Suena de lo más común, pero la particularidad de este escenario es que dicha aplicación sólo recibe datos mediante cadenas de texto, esto es, únicamente recibe expresiones del tipo "$dato_1, dato_2, dato_3, \dots, dato_n$" ¿Por qué?

Porque en este caso atípico no podemos usar frameworks u otras herramientas (como chatGPT o GitHub Copilot) que nos faciliten esta misión, también debemos suponer el frontend que manejamos en este escenario no nos permite recibir datos de tipo INTEGER, STRING, BOOLEAN o cualquier otro tipo de dato aceptado por BQ, porque no cuenta con las herramientas de desarrollo para ello. La razón para esto puede deberse a que nuestra aplicación se desarrolla en un entorno muy nuevo que no cuenta con integraciones de las herramientas mencionadas, falta presupuesto para pagar por alguna de ellas y demás; lo importante es que sólo puede recibir cadenas de texto y con ellas debemos realizar las inserciones.

Ahora bien, esto es un problema porque las sentencias de inserción en SQL (que es el lenguaje de programación usado por BQ) tienen dos requerimientos que entran en conflicto con el hecho de que nuestra aplicación hipotética sólo trabaje con cadenas de texto. El primero de ellos es la estructura o el orden que deben tener los datos para ser insertados:

```
INSERT INTO producto (productoNombre, productoDesc, productoPrecio, tPid, tiendaId) values
("iPhone XL","iPhone de última generación",20000.00, 2, 1), ("MOTO G","Almacenamiento: 16 Gb, SO:
Android",10000.00, 1, 2),
("RM","Teléfono pequeño y resitente",20000.00, 4, 3),
("ZTE","Almacenamiento: 4GB, SO: Android",6000.00, 3, 4),
("Nokia 5120","Teléfono austero",500.00, 2, 5),
("GALAXY Z Omega","Almacenamiento: 64 Gb, SO: Android",50000.00, 5, 5),
("iPhone 5","Almacenamiento: 60 gb, SO: iOS",6000.00, 1, 4),
("BlackBerry 7230","Almacenamiento: 4 Gb, SO: RM",1000.00, 2, 3),
("Xiaomi Redmi Note 8","Almacenamiento: 64 Gb, SO: Android 11 MIUI 12.5",30000.00, 5, 3),
("Huawei Nova 9","Almacenamiento: 64 Gb, SO: EMUI 12 ",12999.00, 3, 2),
("BlackBerry Z10","Almacenamiento: 32 Gb, SO: BlackBerry 10",15000.00, 4, 1);
```

Figura 1. Sentencia de inserción de SQL.

Como podemos observar, los datos están dispuestos en forma de filas y cada fila tiene un número de elementos que coincide con el número de campos en los que se quiere hacer la inserción. Si la aplicación recibe una cadena de texto en una única fila que no tiene saltos de línea y que tampoco cuenta con una especie de delimitador que indique dónde termina una fila de datos y empieza la siguiente (que es el caso en nuestro escenario), entonces vamos a recibir un error cuando intentemos hacer la inserción.

Por otra parte, en este ejemplo podemos observar que algunos datos que no son de tipo STRING, tienen que ser escritos en la sentencia sin los caracteres “” o „‟. En una cadena de texto del tipo “$dato_1, dato_2, dato_3, \dots, dato_n$”, al ser de tipo STRING, no es posible determinar si dato $dato_2$ es de tipo INTEGER o si $dato_3$ es BOOLEAN. Para BQ todo esto va a llegar como STRING, y cuando intente introducirlo en un campo que no es de ese tipo, se va a generar un error.

Por estos motivos, en los siguientes apartados explicaremos cómo resolver estos problemas mediante funciones desarrolladas en Python[ii]. Sin embargo, antes de explicar la solución debemos aclarar que existen dos casos que se relacionan con los requerimientos mencionados anteriormente: 1) la cadena de texto contiene datos que se van a insertar en campos cuyo tipo de dato es únicamente STRING; 2) la cadena de texto contiene datos que se van a insertar en campos en los que el tipo de dato de alguno de ellos es diferente de STRING[iii]. En el caso 1) únicamente se debe generar la estructura requerida por la inserción explicada arriba, es decir, se deben escribir los paréntesis, comas y “” o „‟ que delimitan a los datos de tipo STRING, así como sus respectivos saltos de línea. En el caso 2), también debe realizarse dicho procedimiento de escritura, pero deben omitirse “” o „‟ donde el tipo de dato no sea STRING. Dicho esto, es momento de explicar la solución.

3. **Consideraciones previas para la solución: la función que inserta los datos.**

El primer paso lógico para hacer una inserción es definir una función que contenga una sentencia del tipo `"""INSERT INTO `"""+dataSet+""".`"""+tabla+"""` ("""+campos+""") VALUES"""+valores+"""""""` donde *dataSet* es el conjunto de datos donde se encuentra la tabla en la que se van a escribir los datos, *tabla* es el nombre de ésta, *campos* se refiere a los campos y *valores* es el conjunto de valores que se va a insertar en la tabla. Sin embargo, en este escenario existen más pasos, pues debemos tener en cuenta las validaciones a realizar que se derivan de los dos casos explicados al final del apartado anterior, es decir, antes de realizar la inserción debemos validar si los campos involucrados en esta inserción son todos de tipo STRING, o si alguno de ellos no lo es. Después de esta validación, la función debe determinar si los datos se ordenan de una manera u otra, pues ambos casos requieren de procesos diferentes para su ordenamiento. Y, finalmente, debe construirse la sentencia SQL y ejecutarse la inserción. Teniendo todo esto en cuenta, la función de inserción tiene la siguiente estructura (Lakshmanan, 2021)[iv]:

```
def insercion (cliente,projectId:str,dataSet:str,tabla:str,campos:str,valores:str): listaCampos =
    campos.split(",")
    validacion = validarDatos(cliente,projectId,dataSet,tabla,campos)

    if(type(validacion).__name___== 'str'):
        valores = generarValoresString(valores,len(listaCampos)) else:
    valores = generarValoresNoString(validacion,valores)


    query_string = """INSERT INTO `"""+dataSet+""".`"""+tabla+"""` ("""+campos+""")
    VALUES"""+valores+"""""""
    query_job = cliente.query(query_string)
```

Figura 2. Función de inserción en Python para BQ.

La descripción de los parámetros de esta función es la siguiente:

1. *cliente*: es el objeto Client usado para ejecutar las sentencias SQL en BQ.
2. *projectId*: es el id del proyecto de BQ que se usa para extraer los metadatos en la función *validarDatos*().
3. *dataSet*: es el nombre del dataset del proyecto de BQ que se usa para extraer los metadatos en la función *validarDatos*().
4. *tabla*: es el nombre de la tabla de BQ de la que se van a extraer los metadatos en la función *validarDatos*().
5. *campos*: son los campos cuyos metadatos se van a extraer en la función *validarDatos*() y en los que se va a insertar información.
6. *valores*: los valores que se van a insertar en la tabla.

Por otro lado, *listaCampos* almacena la lista obtenida de dividir el parámetro *campos* usando una coma como parámetro. *validacion* almacena el resultado del método *validarDatos*() (este método se explica en el siguiente apartado). La estructura if-else nos ayuda a decidir si los campos a insertar son únicamente de tipo STRING o de otro tipo (siguiendo lo mencionado acerca de los casos en cuestión). En cualquiera de los dos casos, los métodos mencionados ordenan los datos de modo que se obtenga la estructura requerida por SQL (estos métodos se explicarán más adelante). Por último, se construye la sentencia SQL y se ejecuta la inserción.

Como podemos observar, realizar la inserción en este escenario requiere de varios subprocesos que cuentan con cierta complejidad y que serán explicados a continuación.

4. La función que valida los datos.

La función que valida los datos tiene el objetivo de terminar si en el conjunto de campos involucrados en la inserción existe alguno cuyo tipo de dato es diferente de STRING o si la totalidad de ellos son de este tipo. Una vez que se determina esto, la función devuelve una lista con los metadatos (Holowczak, 2022)[v] o propiedades de los campos (o más propiamente de la tabla) en los que se insertará la información. Necesitamos esto para saber qué función de ordenamiento usaremos después. La función tiene la siguiente estructura:

```
def validarDatos (cliente, projectId: str,dataSet: str,tabla: str,campos: str)->str: full_table_path =

    projectId  + "."+ dataSet + "." + tabla
    listaCampos = campos.split(",")
    metaDatosTabla = cliente.get_table(full_table_path) found = []

    for j in range(len(listaCampos)):
for k in range(len(metaDatosTabla.schema)): if(listaCampos[j] == metaDatosTabla.schema[k].name):
    found.append(metaDatosTabla.schema[k])



    for x in found:
if(x.field_type != "STRING"): retorno = found
        break else:
    retorno = campos



    return retorno
```

Figura 3. Función de validación de datos.

La descripción de los parámetros es la siguiente:

1. *cliente*: es el objeto Client usado para ejecutar las sentencias SQL de BQ.
2. *projectId*: es el id del proyecto de BQ donde se encuentra la tabla de la que vamos a extraer metadatos.
3. *dataSet*: es el nombre del dataset del proyecto de BQ en el que se encuentra la tabla de la que vamos a extraer metadatos.
4. *tabla*: es el nombre de la tabla de BQ de la que se van a extraer los metadatos.
5. *campos*: son los campos cuyos metadatos se van a extraer.

Por otro lado, las variables tienen las siguientes funciones: *full_table_path*: sirve para construir la ruta completa de

la tabla cuyos metadatos se van a extraer. *listaCampos*: almacena la lista obtenida al dividir la cadena recibida en *campos*, usando la coma como parámetro. *metaDatosTabla* almacena los metadatos de la tabla a la que se quiere insertar información, esto se hace en forma de objeto (de la clase Schema). Esto se logra usando la API de BQ/GCP y *projectId*, *dataSet* y *tabla* como parámetros. *found* es una lista en la que se van a guardar los campos cuyos tipos de dato sea diferente de STRING.

Ahora bien, los ciclos for anidados que se encuentran entre la línea 32 y la 35 tienen la siguiente lógica: el primero recorre, uno a uno, todos los elementos de *listaCampos*, mientras el segundo hace lo mismo, pero con la lista *metaDatos.schema*. Luego se encuentra un if en el que la condición es: si alguno de los elementos de *listaCampos* es igual a alguno de los nombres contenidos en *metaDatos.schema*, entonces guarda ese elemento en la lista *found*.

Una vez que se han terminado los recorridos mencionados, se recorre *found* con la intención de lograr el objetivo de *validarDatos*: la estructura for-in recorre elemento a elemento a *found* y dentro tiene otra estructura if- else, cuya condición es que si alguno de los tipos de datos de los campos guardados en la lista en cuestión es diferente de STRING, entonces *retorno* (la variable a retornar) almacena todo en la lista *found* y se detiene el ciclo. En otro caso, se termina de recorrer la lista y si todos los campos son de tipo STRING, se devuelve *retorno* pero almacenando la cadena de texto que contiene los campos involucrados.

Cuando ya se ha realizado este proceso, en la función *insertar*() se encuentra una variable que guarda los resultados de la función recién descrita. Luego, en la estructura if-else mencionada (Figura 2., líneas 133- 136) se determina si hay que usar el método que ordena los valores para campos cuyos tipos de datos son todos STRING o no. En el siguiente apartado describiremos *generarValoresString*(), que es el método que se ejecuta cuando *validarDatos*() devuelve una cadena de texto.

5. La función que ordena los valores cuando todos los campos son de tipo STRING.

generarValoresString() es una función que nos ayuda a obtener la estructura de datos requerida por BQ, a saber:

$$\begin{matrix}(d_1, & d_2, & d_3, & d_4), \\ (d_5, & d_6, & d_7 & d_8), \\ \vdots & \vdots & \vdots & \vdots \\ (d_{m-3}, & d_{m-2}, & d_{m-1} & d_m);\end{matrix} \qquad \textbf{(1)}$$

Donde $d_1, d_2, d_3, \ldots d_n$ representa el conjunto de datos dispuestos en renglones. En cada renglón, se encuentran paréntesis que encierran un determinado número de datos separados por una coma. El número de datos que se encuentra entre paréntesis depende del número de campos que se van a insertar. Por ejemplo, si
con una inserción vamos a agregar valores a 4 campos, entonces entre cada uno de los paréntesis habrá 4 datos. El número total de renglones va a depender de la cardinalidad del conjunto de datos a insertar[vi].

En este sentido, *generarValoresString*() nos ayuda a obtener una estructura como la mencionada cuando los campos de la cadena de texto recibida son todos de tipo STRING. La estructura de la función es la siguiente:

```
def generarValoresString(valores: str, numeroCampos: int)->str: listaCadena = valores.split(",")
    contador = 0 w = ""
    z = ""
    y = ""

while(contador < len(listaCadena)): for i in range(numeroCampos):
    y = listaCadena[contador]
    z += ("'" + y + "',")
    contador += 1

    w += "(" + z[:-1] + "),\n" z = ""

    return w[:-2] + ";"
```

Figura 4. Función que genera la estructura requerida por la inserción de SQL.

La descripción de los parámetros es la siguiente:

1. *valores*: se trata es una cadena de texto que contiene los valores a insertar en una tabla de BQ.
2. *numeroCampos*: es un número entero que nos indica el número de campos involucrados en la inserción de la tabla de BQ (cuando se llama la función, lo que se recibe aquí es el tamaño de la lista obtenida de dividir la cadena *campos,* lo cual se hace en la función *insercion*()).

Las variables declaradas entre las líneas 2 y 4 tienen las siguientes funciones. *listaCadena* almacena una lista que resulta de dividir la cadena *valores* usando una coma como parámetro. *contador* es una variable que nos sirve para extraer los elementos de *listaCadena* y para realizar el conteo de las iteraciones del ciclo while. w, z, y son variables que nos ayudan a realizar las concatenaciones mostradas arriba.

En cuanto al ciclo while, este nos ayuda a recorrer *listaCadena*. El número de iteraciones del ciclo for depende del número de campos a insertar, de modo que las concatenaciones mostradas ahí se realizan el mismo número de veces que *numeroCampos*. Así, por ejemplo, si *numeroCampos* = 4, entonces dichas concatenaciones se van a realizar 4 veces. Al mismo tiempo, esto también implica que se van a concatenar 4 elementos, es decir, el número de elementos de *listaCadena* que se van a concatenar es el mismo que el de *numeroCampos*. Dicho sea de paso: *numeroCampos* va a determinar el número de renglones de la estructura en cuestión.

Por otro lado, las concatenaciones se realizan de la siguiente manera:

1. El elemento extraído de *listaCadena* con *contador* se guarda en y.
2. En z se guarda y concatenado con la coma y las comillas que requiere SQL por tratarse de un dato de tipo STRING.
3. A *contador* se le suma 1 para pasar al siguiente elemento.
4. Cuando se acaban las iteraciones del ciclo for, la cadena concatenada en z se concatena con los paréntesis que delimitan a los datos del renglón y se guardan en w. En este punto también se verifica el valor de *contador* para ver si se sigue cumpliendo la condición del while. Si sí, se repite el proceso. En caso contrario se termina.
5. z se iguala con “” (en cierto sentido se vacía) para repetir el proceso y formar otro renglón. Esto se repite hasta que se acaben los elementos de *listaCadena* y se deje de cumplir la condición mencionada. En este caso, w es el valor que se regresa con ciertas modificaciones que eliminan caracteres que no se necesitan y agrega un ; para indicar que ahí se termina esa parte de la sentencia.

Una vez que se ejecuta esta función, la cadena que obtenemos es la siguiente (que tiene la estructura requerida por BQ y se obtiene a partir de la cadena de texto “Avenida 1,Python1,Avenida2,Python2”):

```
('Avenida 1','Python1'),
('Avenida2','Python2');
```

Figura 5. Resultado de ejecutar *generarValoresString*().

Como mencionamos, esta aplicación nos sirve para cuando los campos con los que estamos trabajando son todos de tipo STRING (la muestra de ello es la Figura 5). Sin embargo, cuando los datos son de tipo INTEGER, BOOLEAN, FLOAT, etc., o cuando en este conjunto hay campos de tipo STRING y otros, debemos realizar un procedimiento parecido, pero con ciertas modificaciones que aumentan un tanto la complejidad de lo ya explicado. Esto se trata en el siguiente apartado.

6. La función que ordena los valores cuando no todos los campos son de tipo STRING.

Como bien dijimos, el hecho de que en una inserción de SQL haya campos cuyo tipo de datos sea STRING, junto con campos cuyo tipo de dato es de cualquier otro, aumenta la complejidad de lo explicado en el apartado anterior, pues ahora los renglones de datos no tendrán esta estructura: $("d_1","d_2","d_3",\dots,"d_n")$, sino que tendrán otras como $(d_1,"d_2","d_3",\dots,d_n)$, $("d_1",d_2,d_3,\dots,d_n)$, etc., porque cualquiera de estos puede ser un entero, booleano, flotante, etc.

La complejidad en este contexto radica en saber qué elementos van a llevar comillas y cuáles no. Se ha desarrollado una solución para esto, pero antes debemos explicar ciertos aspectos necesarios para llegar a ella. Por lo pronto, empezaremos por decir que en lugar de mostrar la totalidad de la estructura de la función *generarValoresNoString*(), mostraremos secciones de la misma, pues cada sección tiene una misión específica e implica descripciones más puntuales que las de las funciones anteriores.

La primera sección es la de los parámetros y las variables iniciales:

```
def generarValoresNoString(validacion:list, valores: str)->str:

listaCadena = valores.split(",") listaValidacion = [] listaValidacion2 = []
w = ""
z = ""
diferencia = len(validacion)
```

Figura 6. Sección de parámetros y variables iniciales de la función *generarValoresNoString*().

Los parámetros *validación* y *valores*, y las variables *listaCadena*, w e y tienen exactamente las mismas funciones que en la función *generarValoresString*(), por ello no las explicaremos aquí. Por otro lado, *listaValidacion* y *listaValidacion2* son listas que nos permiten guardar por separado los resultados obtenidos en *validación* (esto lo explicaremos más adelante). *diferencia* es una variable que nos permite almacenar el tamaño de la lista *validacion*. Este valor será importante más adelante porque nos permitirá realizar procedimientos relevantes en esta función.

La siguiente sección es la de validación. Redundante, sí, pero aún con la validación ya obtenida en *validacion* necesitamos hacer otra por el siguiente motivo: la lista en cuestión contiene instancias de la clase *Schema* que indican el nombre y el tipo de dato de los campos involucrados en una inserción. Entre estos campos puede haber algunos que sean de STRING y otros de tipo diferente. En este caso, es necesario separar el conjunto de los que no son STRING de los que sí, para saber cuáles llevan comillas y cuáles no. Por ello, la sección de validación se estructura de la siguiente manera:

```
    for x in validacion:
if(x.field_type != 'STRING'): listaValidacion.append(validacion.index(x))

    for y in validacion:
if(y.field_type == 'STRING'): listaValidacion2.append(validacion.index(y))
```

Figura 7. Sección de parámetros y variables iniciales de la función *generarValoresNoString*().

Aquí se encuentran dos ciclos for-in. El primero recorre la totalidad de *validacion*. Recordemos que dicha lista contiene objetos de la clase *Schema*, razón por la cual en cada iteración se pregunta si la propiedad *field_type* de cada uno de estos objetos es diferente de STRING. Si esto es verdadero, entonces los índices en los que se encuentran tales objetos se almacenan en la lista *listaValidacion* con el método *append*(). En resumen: *listaValidacion* contiene los índices de *validacion* en los que se encuentran valores cuyo tipo de dato es diferente de STRING. La misma lógica se sigue en el segundo for-in, con la diferencia de que en *listaValidacion2* se almacenan los índices de los campos cuyo tipo de dato son STRING. Hacemos esto porque el saber en qué índices se encuentran estos campos nos ayudará más adelante (en conjunto con *diferencia*) a determinar qué datos deben ir entre comillas y cuáles no, pero aún faltan dos pasos (secciones) para llegar a ese punto.

La siguiente sección es la de igualación de tamaño entre listas. Esto más que se ser una necesidad de la función en cuestión, es un requerimiento de Python: si intentamos ejecutar esta función sin esta sección, entonces obtenemos el error "IndexError: list index out of range" debido a la diferencia de tamaños que existe a veces entre *listaValidacion* y *listaValidacion2*. Estas listas deben ser del mismo tamaño para que, cuando sean recorridas, no obtengamos el error mencionado[vii].

La estructura de la sección es la siguiente:

```
if(len(listaValidacion)>len(listaValidacion2)):
diff = len(listaValidacion) - len(listaValidacion2) for i in range(0,diff):
            listaValidacion2.append("-") elif(len(listaValidacion)<len(listaValidacion2)):
diff = len(listaValidacion2) - len(listaValidacion) for i in range(0,diff):
listaValidacion.append("-")
```

Figura 8. Sección de igualación entre listas de la función *generarValoresNoString*().

Entre las líneas 70 y 77 se encuentra una estructura if-elif. La parte del if verifica si el tamaño de *listaValidacion* es mayor que el tamaño de *listaValidacion2*. En caso afirmativo, el tamaño de *listaValidacion2* se resta al de *listaValidacion* y se usa *diff* (la diferencia) en un ciclo for in para llenar con guiones los índices que le faltan a *listaValidacion2* para tener el mismo tamaño de la otra lista[viii].

En la siguiente sección, y con el mismo objetivo de no obtener el error "IndexError: list index out of range", debemos igualar el tamaño de ambas listas con el tamaño de la lista *validación*.

La estructura de esta sección es la siguiente:

```
for i in range(0,diferencia-len(listaValidacion)): listaValidacion.append("-")
listaValidacion2.append("-")
```

Figura 9. Sección de igualación de listas con el tamaño de *validación* de la función *generarValoresNoString*().

Como se puede observar, se trata únicamente de un ciclo for que va de 0 a el número resultante de restar el tamaño de las listas obtenido en la sección anterior y el tamaño de la lista *validacion*[ix].

Ahora bien, todo lo que hemos venido explicando coincide en este punto, pues varios de estos aspectos se van a utilizar para explicar algo que llamamos algoritmo de *búsqueda por saltos*. Lo hemos llamado así a falta de un mejor término y porque requiere de ciertos "saltos" entre elementos (determinados por la variable *diferencia*) que nos ayudan a identificar qué elementos deben tener tal o cual formato. La explicación de este algoritmo se va a dividir en tres partes: 1) el algoritmo explicado en abstracto; 2) el algoritmo en el código (de la siguiente sección de la función); y 3) el algoritmo con un ejemplo.

7. Algoritmo de *búsqueda por saltos* (en abstracción)

Coloquialmente entendemos que un algoritmo es un conjunto finito de pasos claros y distintos que están destinados a resolver un problema[x]. El problema que tenemos aquí es: necesitamos generar con Python una estructura textual (aceptada por SQL) que nos permita insertar un conjunto de datos en una tabla de BQ, cuando los campos involucrados en la inserción tienen tipos de dato diferentes de STRING (todo esto con el trasfondo de nuestra aplicación peculiar mencionada al principio; los pasos a seguir se muestran más adelante).

Ahora bien, lo primero que debemos observar para resolver dicho problema es recordar parte de la estructura de la *Figura 1*, la cual antes de la palabra reservada *values* tiene los nombres de los campos en los que se van a insertar datos, y después (dispuestos en renglones y columnas) tiene a los datos que se van a insertar. Considerado esto, tenemos una figura como la siguiente:

```
    (productoNombre, productoDesc, productoPrecio, tPid, tiendaId) values
("iPhone XL","iPhone de última generación",20000.00, 2, 1), ("MOTO G","Almacenamiento: 16 Gb, SO:
Android",10000.00, 1, 2),
("RM","Teléfono pequeño y resitente",20000.00, 4, 3),
("ZTE","Almacenamiento: 4GB, SO: Android",6000.00, 3, 4),
("Nokia 5120","Teléfono austero",500.00, 2, 5),
("GALAXY Z Omega","Almacenamiento: 64 Gb, SO: Android",50000.00, 5, 5),
("iPhone 5","Almacenamiento: 60 gb, SO: iOS",6000.00, 1, 4),
("BlackBerry 7230","Almacenamiento: 4 Gb, SO: RM",1000.00, 2, 3),
("Xiaomi Redmi Note 8","Almacenamiento: 64 Gb, SO: Android 11 MIUI 12.5",30000.00, 5, 3),
("Huawei Nova 9","Almacenamiento: 64 Gb, SO: EMUI 12 ",12999.00, 3, 2),
("BlackBerry Z10","Almacenamiento: 32 Gb, SO: BlackBerry 10",15000.00, 4, 1);
```

Figura 10. Estructura de campos y datos de una sentencia de inserción.

Como podemos observar, en la parte posterior de *values* están los campos de la sentencia, los cuales determinan, por decirlo de algún modo, las columnas en las que se disponen los datos, pues debajo del nombre de cada campo, se encuentran datos que corresponden a sus respectivos tipos de dato: *productoNombre* es un campo de tipo STRING y los elementos encontrados abajo son de tipo STRING, por eso se escriben entre comillas; *productoPrecio* es de tipo FLOAT, y los datos encontrados abajo son de tipo FLOAT, lo que implica que no lleven comillas.
Podemos abstraer lo mostrado en la Figura 10 para obtener la siguiente estructura:

$$(c_0, \quad c_1, \quad c_2, \quad c_3) \quad \textbf{(2)}$$
$$(d_0, \quad d_1, \quad d_2, \quad d_3),$$
$$(d_4, \quad d_5, \quad d_6, \quad d_7), (d_8, \ d_9, \ d_{10}, \ d_{11}), \ (d_{12}, \ d_{13}, \ d_{14}, \ d_{15}), (d_{16}, \ d_{17}, \ d_{18}, \ d_{19}), \ (d_{20}, \ d_{21}, \ d_{22}, \ d_{23}), \ (d_{24}, \ d_{25}, \ d_{26}, \ d_{27}), \ (d_{28}, \ d_{29}, \ d_{30}, \ d_{31}), \ (d_{32}, \ d_{33}, \ d_{34}, \ d_{35}), \ (d_{36}, \quad d_{37}, \ d_{38}, \quad d_{39}),$$
$$(d_{40}, \quad d_{41}, \quad d_{42}, \quad d_{43});$$

Donde c_0 a c_3 simbolizan los campos involucrados en la inserción y d_0 a d_{43} representan los datos ubicados debajo que se van a insertar a la tabla. Podemos hacer una abstracción aún mayor y obtener una estructura como la siguiente:

$$\begin{matrix} (c_0, & c_1, & \cdots & c_n) \\ (d_0, & d_1, & d_2, & d_3), \\ (d_4, & d_5, & d_6, & d_7), \\ \vdots & \vdots & \vdots & \vdots \\ (d_{m+1}, & d_{m+2}, & \cdots & d_{m+n}); \end{matrix} \quad \textbf{(3)}$$

En esta expresión c y d tienen exactamente la misma función que en la anterior. n representa el número de campos, m representa la posición de cada dato en cada renglón y $m+n$ representa la última posición en un renglón de acuerdo al número total de campos. Antes de seguir avanzando debemos hacer algunas aclaraciones:

1. Es evidente que la estructura mostrada arriba se asemeja a una matriz, de hecho si nos centramos únicamente en la parte de los datos podríamos representarla como una. Sin embargo, dado que no estamos tratando únicamente con números, sino con diferentes tipos de datos, la consideraremos más como un arreglo.
2. Debemos notar (porque esto será importante en unos momentos) que n es la misma cantidad que *diferencia*, pues ambas nos indican el tamaño del conjunto de campos involucrados en nuestra inserción.
3. Las posiciones de los datos del primer renglón siempre coinciden con las posiciones de los campos. Es decir, d_0 siempre coincide con c_0, d_1 con c_1 y así sucesivamente[xi].

Una vez realizadas estas explicaciones, ya podemos empezar a explicar el algoritmo propiamente. Lo primero que debemos notar es que gracias a los arreglos descritos, podemos ver cuáles son los datos que le corresponden a cada campo. Por ejemplo, si tenemos un arreglo de 4 campos y 12 datos como el siguiente:

$$\begin{matrix} (c_0, & c_1, & c_2 & c_3) \\ (d_0, & d_1, & d_2, & d_3), \\ (d_4, & d_5, & d_6, & d_7), \\ (d_8, & d_9, & d_{10}, & d_{11}); \end{matrix} \quad \textbf{(4)}$$

En el que podemos observar que a c_0 le corresponden d_0, d_4,y d_8. Algo parecido pasa con el resto de campos. Para nosotros es evidente porque lo tenemos ordenado visualmente y podemos incluso señalarlo, pero para la computadora no. La máquina necesita que de algún modo, ya sea en Python, JavaScript o Java, le indiquemos qué elementos le corresponden a cada campo (pues esto también le ayudará a saber cuáles llevan comillas y cuáles no) y para que sepa esto, debemos indicárselo con n o *diferencia* y teniendo en cuenta la aclaración número tres. Por ejemplo, si queremos indicarle a la computadora qué datos le corresponden a c_1, primero debemos tener en cuenta que el primer dato que le corresponde es d_1. Luego, para saber cuáles son los datos siguientes, suponemos que $m = 1$ (por estar d_1 en la posición 1) y le sumamos $n = 4$. De este modo
$m + n = 1 + 4 = 5$, por lo tanto, el siguiente dato es d_5 por estar en la posición 5. Siguiendo esta misma lógica, el siguiente dato será d_9[xii].
Este dato (d_9) también será el último, pues como mencionamos hace algunas páginas, la división del número de datos entre el número de campos determina el número de renglones. De modo que si $|\{c_0, c_1, c_2, c_3\}| = 4$ y $|\{d_0, d_1, d_2, d_3, d_4, d_5, d_6, d_7, d_8, d_9, d_{10}, d_{11}\}| = 12$, entonces $12/4 = 3$, lo que nos indica que sólo habrá tres renglones en este arreglo y que a cada campo sólo le corresponden tres datos porque sólo existen tres renglones.

Ahora bien, supongamos que c_0 es de tipo BOOLEAN, c_1 es de tipo INTEGER, y c_2 y c_3 son de tipo STRING. Con lo ya explicado, sabemos qué elementos le corresponden a c_0 y c_1, los cuales no deben llevar comillas, mientras que los de c_2 y c_3 sí las llevan, por lo que la cadena que debemos pasar a la sentencia de inserción debe verse de la siguiente manera:

$$\begin{array}{llll} (c_0, & c_1, & c_2 & c_3) \\ (d_0, & d_1, & "d_2", & "d_3"), \\ (d_4, & d_5, & "d_6", & "d_7"), \\ (d_8, & d_9, & "d_{10}", & "d_{11}"); \end{array} \qquad \textbf{(5)}$$

Como podemos observar, d_0, d_4, d_8 y d_1, d_5, d_9 no llevan comillas, pues los tipos de dato de sus campos correspondientes no lo exigen. Los datos restantes sí las necesitan porque sus respectivos campos son de tipo STRING.

Antes de pasar a la siguiente sección, haremos un pequeño resumen de los pasos del algoritmo en cuestión:

1. Determinar el número de renglones del arreglo mediante la división del número de datos entre el número de campos.
2. Determinar el número de campos para saber qué tantos elementos se deben saltar en cada iteración (y qué tantos deben escribirse en cada renglón).
3. Determinar, a partir de las posiciones de los campos, qué datos serán los primeros involucrados al empezar a realizar las sumas (o saltos).
4. Realizar las sumas que nos permiten saber qué elementos le corresponden a cada campo.
5. Agregar o quitar las comillas a cada dato, según el tipo de dato del campo que le corresponde.
6. Detenerse hasta que se haya alcanzado el número de renglones y la totalidad de datos.

En la siguiente sección, explicaremos cómo todo esto se representa en el código. Será un tanto más complicado, pues dado que la máquina carece del recurso visual del arreglo, todo el proceso se realizará a través de ciclos for, listas y contadores.

8. Algoritmo de *búsqueda por saltos* (en código)

En esta sección mostraremos cómo se codifica el algoritmo de *búsqueda por saltos*. Por tanto, lo primero que haremos será mostrar una parte de la estructura que tiene el código en Python:

```
for i in range(0,len(listaCadena)): for j in range(0,diferencia):
    if(listaValidacion[j] == i):
    y = listaCadena[i]
    z += (""+y+",")
        listaValidacion[j] += diferencia elif(listaValidacion2[j] == i):
    y = listaCadena[i]
    z += ("'"+y+"',")
    listaValidacion2[j] += diferencia

    w += "" + z[:-1] + "," z = ""
```

Figura 11. Primera parte del código del algoritmo de *búsqueda por saltos*.

Como podemos observar, este código se compone de dos ciclos for, uno anidado dentro del otro. El primero de ellos recorre todos los elementos de *listaCadena* (que contiene los valores a insertar en la tabla de BQ), y el segundo sólo se repite un número de veces igual al número de campos que se van a usar en la inserción. Si bien en este punto no se forman propiamente los renglones, sí se toma en cuenta *diferencia* para tener un orden en la asignación de las comillas y llevar el conteo de los renglones para detenerse cuando sea necesario (primer paso del algoritmo). Esto también implica que por cada elemento recorrido de *listaCadena*, se van a ejecutar n (*diferencia*) iteraciones del segundo for (segundo paso del algoritmo).

Por otra parte, tenemos una estructura if-else. La lógica de esta estructura es la siguiente: si un elemento de *listaValidacion* (la lista que contiene los índices de los campos cuyo tipo de dato no es STRING) es igual al índice que lleva el conteo de *listaCadena* (i), entonces el elemento de *listaCadena* se guarda en la variable y; y se concatena con un carácter vacío y una coma para acumularse en z (sin comillas porque en este apartado se encuentran los datos que no son STRING); y por último, al elemento de *listaValidacion* se le suma diferencia.

Antes proseguir debemos aclarar ciertas cuestiones. La primera es que en las primeras iteraciones de este ciclo for (las primeras n iteraciones que coinciden con el número de campos involucrados en la inserción), el programa está determinando cuáles son los primeros datos que coinciden con los campos. Es decir, *listaValidacion* contiene los índices de los campos, pero también son los índices de los datos iniciales con los que se harán los saltos mencionados (lo cual es el tercer paso de nuestro algoritmo).

La segunda aclaración es que los saltos en esta parte del código se realizan con $listaValidacion[i] += diferencia$ (aquí se ejecuta el paso número cuatro del algoritmo[xiii]): una vez que se encuentra una coincidencia entre el elemento de esta lista y el índice del dato, al elemento de la lista se le suma *diferencia* para realizar el salto y obtener el siguiente índice, de modo que en la siguiente iteración, si vuelve a haber una coincidencia (entre el nuevo valor de *listaValidacion* y el índice del dato), se repetirá el proceso y se obtendrá el valor del siguiente dato a insertar.

La tercera aclaración es que el procedimiento del elif es exactamente igual al que acabamos de describir, con la diferencia de que en esta condición es donde se asignan las comillas a los datos indicados porque *listaValidacion2* contiene los índices de los campos que son de tipo STRING.

Una vez que se termina el proceso de validación y se alcanza el límite de iteraciones para el segundo for, se vuelve a hacer una concatenación (una por cada iteración del primer for) en la que a z se le quita el último elemento, se concatena con un carácter vacío y una coma, y se acumula en la variable w. Luego z se "vacía" y se repite el proceso hasta alcanzar el límite de *listaCadena*. Si bien en esta parte del código no se forman los renglones, sí se hace el conteo indicado para perfilar la construcción de los mismos.

Para la generación de los renglones se codificaron otras cuantas líneas que se explican a continuación:

```
    w = w[:-1]

    listStr = w.split(",") listStr2 = []
    contador = 0
    rango = len(listStr)/diferencia

for k in range(0, int(rango)): listStr2.append("")

    for i in range(0,int(rango)):
    for j in range(0, diferencia):
    listStr2[i] += (listStr[contador]+",") contador += 1

for i in range(0,len(listStr2)): listStr2[i] = "("+listStr2[i][:-1]+")"

    w = ""
for i in range(0,len(listStr2)): w +=  listStr2[i] + ",\n"

    w = w[:-2] + ";"
```

Figura 12. Segunda parte del código del algoritmo de *búsqueda por saltos* (generación de renglones).

En la línea 104 se retoma la variable *w* para quitarle el último carácter, pues este impide obtener la estructura deseada. Luego en *listStr* se almacena una lista que se obtiene de dividir la cadena obtenida en la línea recién descrita (se guardan los datos a insertar, pero esta vez con el formato que le corresponde a cada uno, con o sin comillas). *listStr2* nos permite inicializar una lista que usaremos más adelante. *contador* es un contador que nos permite realizar el conteo en loops posteriores. *rango* es la variable que almacena el número de renglones, el cual se obtiene, como dijimos, de dividir el tamaño del conjunto de datos entre el número de campos.

El loop de las líneas 111 y 112 tiene el objetivo de llenar *listStr2* con espacios vacíos para que tenga el mismo tamaño de *rango*. Hacemos esto para evitar el error "IndexError: list index out of range". Si no lo hacemos, como la lista no tiene elementos ni índices, cualquier operación que se realice sobre ella se considerará como fuera de rango.

El ciclo de las líneas 115 a 118 tiene el objetivo de agrupar los datos en renglones. El primer loop itera de 0 al número de renglones obtenido. El segundo va de 0 a diferencia, esto con el objetivo de agrupar el número debido de datos en cada renglón. Esto se hace obteniendo cada elemento de *listStr2* (con ayuda de *contador*[xiv]) para concatenarlo con una coma y acumular esta cadena en cada uno de los elementos de *listStr2*. Dicho de otra manera, el primer loop itera una vez por cada renglón, y en cada uno de estos renglones se contatena/acumula el número de datos que le corresponden.

En el loop de las líneas 120 y 121 únicamente se agregan los paréntesis que delimitan a cada renglón y se elimina el último carácter, que es una coma que está de más. En la línea 123 se "vacía" la variable *w* para poder concatenar en ella cada elemento de *listStr2*, junto con una coma y su respectivo salto de línea. Por último, a *w* se le quitan los últimos dos caracteres (un salto de línea y una coma que sobran) y se le agrega un ; para poder retornarla.

Aquí es donde se termina la explicación del algoritmo de *búsqueda por saltos*, junto con algunos recursos que se tuvieron que agregar para completarlo. También se finaliza la explicación de la función *generarValoresNoString*(). A continuación se muestra un ejemplo de cómo funciona este código.

9. Algoritmo de *búsqueda por saltos* (en acción)

Para este ejemplo supongamos que tenemos una tabla llamada *Proveedores*, a la que queremos insertar dos datos ("1,proveedor1,2,proveedor2") en los campos *proveedorId* y *proveedorNom*. *proveedorId* es de tipo INTEGER y *proveedorNom* es de tipo STRING.

Ahora bien (aquí vamos a omitir algunos pasos para hacer más dinámica esta explicación), lo primero que debemos considerar es que lista validación va a tener los siguientes elementos:

1. SchemaField('proveedorId', 'INTEGER', 'NULLABLE', None, (), None).
2. SchemaField('proveedorNom', 'STRING', 'NULLABLE', None, (), None).

Los cuales tienen información acerca de los campos mencionados. La propiedad que importa es la llamada *field_type,* pues es la que contiene el tipo de dato de cada campo. Los ciclos for que trabajan con las listas de validación son los que se encargan de usar esta propiedad para almacenar los índices de dichos campos. En este ejemplo, *listaValidacion* tiene el número 0 como elemento y *listaValidacion2* tiene el número 1.

Luego de hacer la igualación de tamaños, es momento de pasar a los ciclos for que realizan los saltos.

1. Primera iteración del for posterior $i = 0$:
   a. Primera iteración del for interior:
      i. $j = 0$.
      ii. $listaValidacion[j] = 0$ (es igual a i).
      iii. $listaCadena[0] = 1$.
      iv. 1 se concatena en z sin comillas y con una coma.
      v. $listaValidacion[j] += diferencia = 0 + 2$, por tanto $listaValidacion[0] = 2$.
   b. Segunda iteración del for interior:
      i. $j = 1$.
      ii. $listaValidacion[j] = '-'$, pasamos a elif.
      iii. $listaValidacion2[j] = '-'$.
2. Segunda iteración del for posterior $i = 1$:
   a. Primera iteración del for interior:
      i. $j = 0$.
      ii. $listaValidacion[j] = 2$, pasamos a elif.
      iii. $listaValidacion2[j] = 1$ (es igual a i).
      iv. $listaCadena[1] = proveedor1$.
      v. $proveedor1$ se concatena en z con comillas y con una coma.
      vi. $listaValidacion2[j] += diferencia = 1 + 2$, por tanto $listaValidacion2[0] = 3$.
   b. Segunda iteración del for interior:
      i. $j = 1$.
      ii. $listaValidacion[j] = '-'$, pasamos a elif.
      iii. $listaValidacion2[j] = '-'$.
3. Tercera iteración del for posterior $i = 2$:
      i. $j = 0$.
      ii. $listaValidacion[j] = 2$ (es igual a i).
      iii. $listaCadena[2] = 2$.
      iv. 2 se concatena en z sin comillas y con una coma.
      v. $listaValidacion[j] += diferencia = 2 + 2$, por tanto $listaValidacion[0] = 4$.
   b. Segunda iteración del for interior:
      i. $j = 1$.
      ii. $listaValidacion[j] = '-'$, pasamos a elif.
      iii. $listaValidacion2[j] = '-'$.
4. Cuarta iteración del for posterior $i = 3$:
   a. Primera iteración del for interior:
      i. $j = 0$.
      ii. $listaValidacion[j] = 4$, pasamos a elif.
      iii. $listaValidacion2[j] = 3$ (es igual a i).
      iv. $listaCadena[3] = proveedor2$.
      v. $proveedor2$ se concatena en z con comillas y con una coma.
      vi. $listaValidacion2[j] += diferencia = 3 + 2$, por tanto $listaValidacion2[0] = 5$.
   b. Segunda iteración del for interior:
      i. $j = 1$.
      ii. $listaValidacion[j] = '-'$, pasamos a elif.
      iii. $listaValidacion2[j] = '-'$
      iv. En este punto se terminan las iteraciones.

Cuando se terminan las operaciones, obtenemos una cadena de texto como la siguiente: "1,'proveedor1',2,'proveedor2',", la cual, después de ser tratada por la segunda sección del código del algoritmo, permite obtener la siguiente estructura:

```
(1,'proveedor1'),
(2,'proveedor2');
```

Figura 13. Resultado del algoritmo y la función *generarValoresNoString*().

Como podemos observar, los elementos que son de tipo STRING llevan sus respectivas „", y los que son de tipo diferente no la llevan. Además, los paréntesis, comas, saltos de línea y ; se encuentran en los lugares indicados. Una vez que hemos terminado esta explicación, debemos regresar a la función *inserción*().

10. De regreso a la función *inserción*()

Retomando la *Figura 2*, podemos ver que ya hemos explicado los elementos necesarios para que se realice la inserción: explicamos las funciones *validarDatos*(), *generarValoresString*() y *generarValoresNoString*(), las cuales nos ayudan a tener la estructura necesitada por BQ para que se puedan introducir los datos. Una vez que se ha realizado la validación de las líneas 133-136, todos los datos generados con estas funciones y los recibidos como parámetros, se concatenan en la variable *query_string* y luego esta se pasa como parámetro para realizar la inserción. Si no hay errores, se realizará este proceso.

11. Conclusiones

Primero debemos señalar lo obvio: la complejidad computacional de las funciones es $O(n^2)$, debido a los numerosos for anidados que encontramos a lo largo de ellas. Sabemos que esto puede afectar el desempeño de las funciones cuando nos encontremos en escenarios con grandes cantidades de información, por lo cual se debe buscar una alternativa con una complejidad menos agresiva. Suponemos, pues, que se pueden explorar otras opciones que usen estructuras de datos (como Great Learning, 2022, que usa una estructura de datos para revertir una lista ligada) con un enfoque iterativo, o incluso buscar alguna opción recursiva que nos permita resolver este problema sin necesidad de caer en el peor de los casos. Sin embargo, en este espacio no es posible realizar esta tarea porque nos encontramos con una primera aproximación a la resolución del problema planteado y al algoritmo de *búsqueda por saltos* (el cual también se encuentra ligado a $O(n^2)$). Además esto se saldría de los objetivos del escrito.

En este mismo sentido, también queremos resaltar la importancia del algoritmo recién mencionado. Si bien la función principal de éste es generar la estructura requerida por BQ, tenemos la suposición de que puede tener otras aplicaciones. Es decir, pensamos que puede ser usado para realizar alguna especie de búsqueda en estructuras parecidas al arreglo mostrado en páginas anteriores o podría, por ejemplo, usarse cuando se desee realizar una búsqueda en arreglos de dos dimensiones con técnicas parecidas a la *búsqueda por filas* o *búsqueda por columnas* (algoritmos que tienen estructuras similares al de *búsqueda por saltos*). Así también, las demostraciones por inducción matemática y demás cuestiones relacionadas con análisis y diseño de algoritmos quedan pendientes, pero no olvidadas, para momentos posteriores.

Por otra parte también estamos conscientes de una de las objeciones más grandes que pueden hacerse a este escrito: el uso de algún framework o de la API de Google Cloud para Python, e incluso en este momento el de una "inteligencia artificial", como chatGPTno GitHub Copilot, podrían haber hecho la inserción más sencilla, incluso hubiera hecho más sencillo el desarrollo de toda la aplicación. Sí, es verdad. Sin embargo, el escenario en el que nos pusimos (el de la aplicación que únicamente trabaja con cadenas de texto) también nos exigió pensar en cómo sería trabajar sin estas erramientas. No estamos en contra de su uso, incluso pensamos que estos recursos nos ayudan a ser más eficientes y nos dan el tiempo para pensar en cosas de mayor complejidad, pero también quisimos pensar en cómo sería estar en un escenario en el que hubiera que codificar desde 0 y donde no hubiera este tipo de herramientas para ayudarnos.

En relación con lo anterior, lo que aquí desarrollamos puede ser integrado a algún framework o alguna biblioteca cuya aplicación se centre en BQ o incluso en SQL y demás. Esto es, se puede usar lo aquí descrito para mejorar procesos (si es posible y a reserva de la complejidad mencionada) trabajados en Python o en cualquier otro lenguaje. Aquí usamos dicho lenguaje de programación debido a su popularidad en el rubro de los datos, pero si hay un lenguaje en el que se puedan declarar funciones y arreglos, lo mostrado aquí es perfectamente replicable (muestra de ello son las pruebas realizadas en JavaScript para combatir ciertos errores).

Esta replicabilidad se debe a las estructuras matemáticas y computacionales mostradas arriba, las cuales, a pesar de las diferencias sintácticas y lingüísticas a las que estamos acostumbrados, son las mismas y funcionan de la misma manera, al menos en los contextos relacionados con el cómputo. Con esto también queremos recalcar que uno de los retos para este trabajo fue el poner en ejercicio la lógica más que la codificación, pues aquí lo que implicó más tiempo y esfuerzo fue encontrar el algoritmo, el cual no depende del lenguaje de programación, sino, como acabamos de decir, de las estructuras matemáticas y computacionales que se nos pusieron en frente.

Esto último sirve como punto de partida (y como punto final para este escrito) para trabajar, en espacios posteriores, las cuestiones que se han dejado abiertas, pues éstas también van más a allá de codificar con x o y lenguaje de programación y se encuentran más orientadas a las estructuras de los datos, los algoritmos y las matemáticas.

12. Fuentes

- Algoritmo de búsqueda. (30 de noviembre de 2022). En *Wikipedia*. https://es.wikipedia.org/wiki/Algoritmo_de_b%C3%BAsqueda
- Google. (2023) *¿Qué es BigQuery?*. Recuperado de: https://cloud.google.com/bigquery/docs/introduction?hl=es-419.
- Google. (2023) *Crear cuentas de servicio*. Recuperado de: https://support.google.com/a/answer/7378726?hl=es
- Holowczak. (2022, 19 de enero). *Python Programming with Google BigQuery*. Recuperado de: https://holowczak.com/python-programming-with-google-bigquery/7/
- Jenn, Jie. (2022, 13 de julio). *Getting Started With Google BigQuery API In Python* [video]. YouTube: https://www.youtube.com/watch?v=lLPdRRy7dfE&t=3s
- Lakshmanan, Lak. (2021, 10 de febrero). *How to trigger Cloud Run actions on BigQuery events*. Recuperado de: https://cloud.google.com/blog/topics/developers-practitioners/how-trigger-cloud-run- actions-bigquery-events
- Ponce, Jahaziel. (2021, 21 de febrero). Algoritmos de búsqueda. Recuperado de: https://jahazielponce.com/algoritmos-de-busqueda/

13. Notas

[i] Para realizar las inserciones en BQ usamos la librería de la API de ésta, siguiendo los pasos descritos por Jenn, 2022. De este recurso tomamos la manera de importar los módulos correspondientes y configurar el ambiente. Esto no se ve en el código citado en el artículo, pero es importante mencionarlo por el hecho de que toda esta solución tiene el objetivo de insertar información de BQ. De igual manera, esto nos ayudó, junto con Google, 2023, a configurar todo lo relacionado con la cuenta de servicio que da acceso a este recurso. Esto tampoco se encuentra en el artículo o en el código por cuestiones de privacidad.

[ii] En los siguientes apartados vamos citar código, pero para tener una referencia más general del mismo también se proporciona el siguiente repositorio de GitHub: link de github.

[iii] Cuando hablamos de tipos de datos diferentes de STRING, nos referimos principalmente a INTEGER, BOOLEAN, FLOAT y similares. Casos más específicos como el de DATE (que necesita formatos diferentes a los tipos de datos mencionados) no se tratan aquí, pero con algunas modificaciones al código presentado pueden trabajarse.

[iv] El artículo de Lakshmanan nos sirvió, principalmente, para saber cómo codificar las sentencias SQL dentro del código de Python: fue una bases para saber cómo usar las comillas (“”) dentro del flujo del código. Por otra parte y en menor medida, sirvió para confirmar que algo de este tipo puede hacer en BQ

[v] La idea de cómo obtener los metadatos viene de Holowczak, 2022. De aquí únicamente retomamos la sentencia que contiene la variable *full_table_path*, pues esta es la que nos ayuda a extraer la información requerida. La dejamos con este nombre como referencia y, como se puede observar, el tratamiento del objeto obtenido es diferente al encontrado en la fuente, pues la naturaleza de este artículo es un tanto diferente. Por otro lado, la idea de trabajar con los metadatos surgió de la necesidad de trabajar con varias tablas, es decir, el hecho de pensar en que esta solución se aplicara a cualquier tabla y no sólo a una con una estructura determinada, nos llevó a concluir que es mejor trabajar con los metadatos, pues con ellos podemos obtener el número de columnas y el tipo de cada una, lo cual es algo que se puede obtener de cualquier tabla y que puede funcionar con la solución. La ambición de esta especie de automatización viene del hecho de que, en la práctica, las tablas que encontramos tanto en las bases de datos convencionales como en BQ siempre son variables en cuanto a su estructura.

[vi] Aquí es conveniente señalar que, en caso de requerirse, para determinar el número de renglones se deben realizar los siguientes pasos:

1. Determinar el número de campos: $c_1, c_2, c_3, \dots, c_m$.
2. Determinar el número de datos: $d_1, d_2, d_3, \dots, d_n$.
3. Dividir el número de datos entre el número de campos: $n/m = r$, donde r es el número de renglones.

Por ejemplo, si tenemos 4 campos y 12 datos, entonces tendremos un total de 3 renglones. Además, cabe aclarar que, dentro y fuera de este contexto, para realizar una inserción en SQL es importante que el número de datos sea un múltiplo del número de campos. De esta manera el gestor de bases de datos (o en este caso BQ) no notificará sobre un error de exceso o falta de datos.

[vii] Si intentamos codificar esta misma función en JavaScript (el otro lenguaje en el que se hicieron pruebas para entender el porqué de esta situación) podremos notar que no se obtiene el mismo error, por lo que no tienen que codificarse esta sección y la siguiente. Por ello, esto es más un requerimiento de Python que una necesidad de lo aquí descrito.

[viii] Llenamos con guiones las listas con el único fin de igualar los tamaños entre las mismas y porque los guiones no se parecen a ninguna de las cadenas de texto que se usarán más adelante. Así no tenderemos problemas con los procedimientos mostrados posteriormente.

[ix] Esto se podría haber hecho todo en una única sección realizando la igualación de cada lista directamente con diferencia, pero lo hicimos así porque parece más explícito para entender la naturaleza de este problema.

[x] Para realizar este algoritmo nos basamos más en un análisis del problema al que nos enfrentamos. Este se encuentra descrito en los arreglos de datos y código descritos a lo largo del artículo. Sin embargo, el llamarle *búsqueda por saltos* viene del hecho en que observamos varios algoritmos de búsqueda para entender si lo que hicimos entraba dentro de esta categoría. En general, nos basamos en Ponce, 2021, Wikipedia, 2022, clases universitarias de algoritmos y estructuras de datos, y análisis y diseño de algoritmos..

[xi] Esto posiblemente sea objeto de una demostración por inducción matemática, pero como eso sobrepasa los objetivos de este escrito, la dejaremos para otra ocasión.

[xii] Este procedimiento es el que otorga al algoritmo el nombre de *búsqueda por saltos*, pues tener en cuenta el número de campos involucrados en la inserción, nos indica el número de elementos que se tienen que "saltar" en cada caso para encontrar qué elementos le corresponden a los campos. Este descubrimiento se hizo después de escribir todas estas estructuras en papel y observarlas.

[xiii] Los pasos 5 y 6 son más evidentes en el código, por eso no los vamos a señalar.

[xiv] *contador* se usa para retomar el conteo en el número en que quedó cada vez que se pasa al segundo loop. La necesidad de este contador también recae en el hecho de que $j = 0$ cada vez que se entra al segundo loop.

Various Applications of IoT-based Weather Monitoring Systems in the Agricultural Sector for Bangladeshi Farmers.

Varias aplicaciones de los sistemas de monitoreo meteorológico basados en IoT en el sector agrícola para los agricultores de Bangladesh.

Md Sofiqul Islam
mdsofiqulislamsumon66@gmail.com
Md.Saiful Islam
222198040@student.presidency.edu.bd
Jahangir Hossain Rabbi
222198040@student.presidency.edu.bd
A S M Binjer Anayet Biddut
222198040@student.presidency.edu.bd
Md.Salman Hossen
222198040@student.presidency.edu.bd

Department of Electrical and Electronics Engineering, Presidency University, Dhaka-Bangladesh

Abstract : Climate change is playing a significant role in agriculture. As an agriculturally efficient country like Bangladesh, smart farming is needed to effectively help farmers adapt to this climate change. The main objective of this IOT based project is to increase efficiency, especially in mushroom farming and shrimp fishing. In both cases, smart farming requires real-time weather information from farmers. This system enables farmers to monitor temperature, humidity, and water levels at any moment from anywhere in the world using their smartphones. This system focuses on local farmers in different districts of Bangladesh, where they are able to monitor the weather for their decision-making and optimize the concept of real-time scheduling for specific food crops. Local rice farmers who are using real-time weather monitoring systems can know the exact moment of their rains, which helps them determine the time to harvest their mature rice. This approach will open up a new avenue for our local farmers through which they can increase their capacity through mushroom cultivation and shrimp farming more than before, which will help boost the economy of a country like Bangladesh a lot.

Keyword : IoT,Esp-8266,DHT-11 Sensor, Farming-Bangladesh

1.Introduction

Most of the people of Bangladesh are very involved in agriculture, so they need to know the weather conditions very well. Most of the farmers in Bangladesh are local farmers, and if the crops they produce can be produced smartly, then there is a very good potential in the economic sector of Bangladesh. Good crop production has become a big problem in Bangladesh now. Rapid urbanization and population growth have led to climate change in Bangladesh, Holovatyy, A., 2021 [1]. As a result, the weather in Bangladesh is now very unpredictable. In such a situation, if you want to cultivate mushrooms and shrimp, whether the temperature and water level are constantly changing or not and whether there is a possibility of rainfall, farmers need to know real-time information about all these things, Annika, V. O. (2023). [2]. Every year in Bangladesh, it is seen that many farmers suffer extensive damage to their winter vegetables due to unexpected rains during the winter. At that time, farmers claimed that they could not be on alert in advance because they did not know whether the rain would come or not, which resulted in maximum damage to their crops.

Shrimp farming and mushrooms are among the agricultural products exported by Bangladesh. Mushroom and shrimp farmers in Bangladesh have not yet been able to significantly increase the effectiveness of their farming because they have not yet implemented smart cultivation, meaning they have not yet implemented the proper use of IoT-based weather monitoring systems, .Islam M.S.,(2024) [3]. Due to excessive water height and low water pressure, shrimp farmers in Chittagong often face the danger of many shrimp fish dying because they are not yet familiar with the use of IoT-based technology, which has resulted in their productivity not increasing much. Also, many farmers in Bangladesh cultivate salt, and measuring water levels is very important for this salt, and in this case, an IoT-based weather monitoring system is an acceptable system through which farmers can always be informed about the water level.

To address the climate change problem in Bangladesh, Bangladeshi farmers need to adapt very quickly to the best weather monitoring system built on our own technology, through which they will be able to collect real-time monitoring information and make decisions very easily. Our system is primarily designed for the farmers of our country so that they can get daily forecasts and make informed decisions very quickly to save their resources from being wasted[4].Our proposed system uses a variety of sensors that are integrated and perfectly installed to measure pressure, temperature, water level, humidity in a specific area and collect real-time data. Due to climate change, farmers in our country are worried about the weather in the coming months. If they have a summary of the real-time data from the previous year, they can easily use that summary to make a forecast for the next month. Since our system stores real-time data in the cloud system, farmers can easily review the data from the previous year or month and prepare a forecast for the next month. We have tried to make this proposed system as simple as possible so that all farmers at the village level can use it with very little difficulty and get the proper benefits and they can be financially successful.

The proposed system can update the farmer with accurate weather forecasts twice a day to help farmers make informed decisions on which activities to carry out without wasting resources[5]. A similar application was developed for greenhouse gases, where an IoT weather monitoring system was created to provide real-time data on greenhouses, updating weather conditions every moment[6]. This system stored real-time data, such as humidity, light, and air pressure, in IBM Store cloud storage. The advanced IoT weather monitoring system reviewed weather updates every moment and automatically notified the greenhouse manager via email if it saw any major changes in temperature, which is an excellent system that can quickly detect damage and balance the greenhouse temperature. As technology continues to advance, modern agricultural methods like greenhouse farming are becoming increasingly popular. These innovative practices enable effective monitoring and control of greenhouse environments, offering valuable insights for crop management. This can be achieved through cost-effective and low-power consumption systems, utilizing technologies such as Arduino. These systems can connect to WiFi networks and operate on the global internet system, facilitating the optimization of climatic conditions based on crop data[7].

2. Literature review

A literature review will focus on IoT-based weather monitoring systems using Esp-8266 and various sensors, including calibration techniques, data processing algorithms, and practical applications. It will describe various challenges that were faced and accurate information of weather conditions that sensors are provided. Along with power delivery, & consumption, its future fields will also be highlighted. Currently there are many ongoing studies on IoT based monitoring system and this paper will discuss its future. This paper will also study the basics of IoT based monitoring systems and will focus on its impact on various fields like agriculture, Industry and the future potentiality in numerous sectors[8].

Current existing methods for weather monitoring mainly use analog instruments such as thermometers, barometers, wind vanes, rain gauges to measure weather and climate changes. Most of the instruments use very simple analog technology and instead physically record the changes and store them in a database. This information is then transmitted to news stations and radio stations where weather reports are provided[9].In addition, existing weather monitoring systems use heavy equipment that is difficult to maintain and needs to be replaced frequently. Moreover, they have to be operated manually which is time

consuming. Apart from this, these machines consume a lot of power and also face a lot of problems in maintaining the accuracy of the data and if the temperature changes, it has to be checked manually. The data that is obtained manually is sent to the computer through the logger. Moreover, heavy winds make it very difficult to move the equipment from one location to another and many spaces are needed. The biggest problem to face is that it delays warning messages in case of sudden weather changes[10].

On the other hand, by this technique farmers don't need much equipment to measure the weather parameters. By using this technique they will aware the current situation in their farms or fields. If any bad condition occurred they will have enough time to solve the issues . Specially the shrimp farmers will get the temperature information in every moment which is necessary for the shrimp farming. Moreover it does not need a huge place to place it in the farms. So it is basically more convenient for the Bangladeshi farmers to operate and set up. So this IoT based weather monitoring system will have a huge impact on the agricultural sector in Bangladesh.

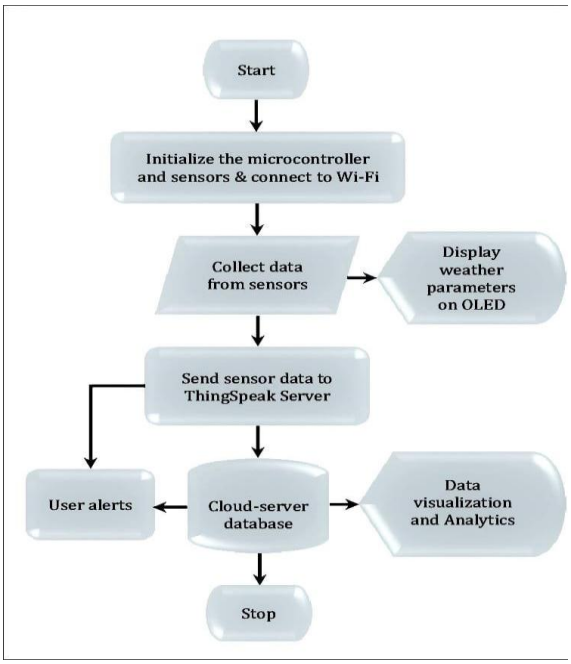
After the spread of the Internet, IOT has become a symbol of trust in various sectors, especially in the data sector, agricultural sector as well as business organizations. IoT basically means a system through which data can be exchanged and communicated continuously without any connection of wire. One of its purposes is to determine, observe, locate, and manage any object according to its predefined goal. And this idea has not only increased the capabilities of the Internet several times, but has also connected people to any object as well as among themselves. The IoT framework allows us to connect the various objects around us in a variety of ways.

The concept of IOT will stand as an emerging communication medium in the world with various everyday objects that will ensure that the objects are connected to a microcontroller or powerful communication device. This will ensure seamless communication between the objects themselves and with the users that will enable Internet use. IoT aims for the Ubiquitous introduction and expansion of the internet. Through it, we can connect electronic objects in our homes, surveillance cameras, cars, and important devices, which increases our own security several times. Individuals, government institutions, and business organizations analyze the data provided by IoT. Can be identified and used for various purposes. Contemporary technological advancements dictate the management and activities of various objects[11].

Daily weather monitoring affects our daily lives. These results significantly affect our various sectors, including agriculture, industry, and construction. But the primary impact can be felt by agriculture and industry. The weather monitoring system continuously displays the overall condition of a specific place. Basically, based on the data provided by the sensors, it predicts the weather of a specific region through some mathematical models. Daily weather monitoring affects our daily lives. These results significantly affect our various sectors including agriculture, industry, construction. But the primary impact can be felt by agriculture and industry[12]. Basically, based on the data provided by the sensors, it predicts the weather of a particular region through some mathematical models. At present, IoT is widely used by various reputed companies in their various canters such as the one based in Barcelona called open dot. This weather monitoring solution uses IoT technology to collect data on various topics including temperature, air pressure, humidity, gas, water level. Basically the sensors provide the data to the user in their mobile application and the user can access it in real time. So If our government want to plan how to increase our agricultural productivity by 2030 they will definitely need a modern solution to solve farmers limitations. If our government subsides heavily on the weather forecasting method like this one then our farmers can harvest much more crop than we have now. Because farmers don't get information in real quick to solve the weather issues in their farms. Government should spend around 100 millions to the real time weather forecasting like IoT based weather forecasting system.

Sensors are systematically installed at different locations and they collect the weather data of that location and display an accurate result. The main objective of this paper is to develop an effective weather monitoring system through which accurate information of important weather parameters of any place can be obtained and stored in the cloud. Usually sensors are connected to the environment so that they can present real data of different parameters. And of course, as the weather changes, the parameters and data

will continue to update and show the data for that particular period[13]. By this, our farmers will have a clue what to do next for their crops safety. For example, If this system give a data about heavy humidity then our rice or cabbage farmers will aware about mold or bacteria spread in the field. According to the information they can spry pesticides to the crops.



3.Methodology

IoT-based Weather Monitoring system is a system where using some sensors that collect to device, process and transfer the real time weather data. These sensors generally using for measuring temperature, humidity ,air pressure and environmental factors that are connected into internet either microcontroller or cloud based-platform[14]. The collected data is transmitted wirelessly, stored in a database, and used for analysis. Using IoT technology, this system ensures continuous monitoring, remote accessibility, and real-time updates, making it an effective and common solution for weather forecasting, agricultural planning, and disaster management [15].

3.1Working Principle

Here the first step is the process begins initializing to the system. Then initializing the microcontroller and sensors(like temperature, humidity, pressure)And the system established a connection with Wi-Fi for data transmission. Collect data from the sensors that are environmental parameters like temperature, humidity and pressure are gathered[16]. Started the parallel process that displays weather parameters on OLED and send sensor data to thingspeak server. The collected data shown on display then send into thingspeak server. Then sent data starting processing and stored the data further analyzing[17]. The stored data is used for visualization, trend analysis, and predictive weather modeling. Another step that if certain something went wrong (like over heat or humidity) alerts are sent to users.At the last when need to stop then stop the program[18]

Fig-1

3.2All Components

IoT based weather monitoring system, Some important sensors used in this project are 5V Buzzer, DTH11, Ultrasonic sensor, PIR sensor, LCD & 12C, MQ2 sensor, Nodemcu Board, Relay Module, 5V source etc. Below is a detailed idea about all these components.

Components Name :

1.Nodemcu (ESP-8266)	2.DHT-11 Sensor	3.Ultrasonic Sensor
4.Realy Module	5.LDR sensor	6. 5V Buzzer
7.Pair sensor	8.MQ-2 sensor	9.LCD & 12C Module
10.5V DC source		

5V Buzzer

The famous 5V passive buzzer is used in endless projects with all different kinds of microcontrollers including the Raspberry Pi and Arduino. It is great to add audio alert to electronic design. The buzzer will push straight into a breadboard for prototyping and can also be soldered to a standard 1.6mm PCB.

5V Buzzer(fig-2)

DTH11 Sensor

In this project, DTH11 is used to measure temperature and humidity. Basically, this sensor is used to detect the existence of temperature and humidity. It is a low-cost digital sensor for checking temperature and humidity. It can be easily connected to Arduino and take temperature and humidity readings**.**

DTH11 Sensor(fig-3)

Ultrasonic Sensor

An ultrasonic sensor is a device that measures the distance to an object using ultrasonic sound waves. It is a device that uses a transducer to send and receive ultrasonic pulses that relay back information about an object's proximity. High frequency sound waves reflect across boundaries to produce distinct echo patterns.

Ultrasonic Sensor (fig-4)

PIR Sensor

Passive infrared sensor(PIR) is basically a motion detector sensor that detects whether any object is moving. It is a sensor used in motion detectors such as automatically triggered light devices and security systems that measure infrared light emitting devices in their field of view.

PIR Sensor(fig-5)

LCD & 12C Module

LCD & 12C refers to Liquid crystal display integrated with 12c communication interface that easily connect to the internet or any kind of pcb. Its basically made for Audino based system for allowing any kind of information get to by user.

LCD & 12C Module(fig-6)

MQ-2 Sensor

MQ-2 is a versatile gas sensor. Its capable to detect gas, alcohol, carbon monoxide, liquefied petroleum gas, propane and smoke. Such this detect the multiple gases but it has lack of the difference between them.

MQ-2 Sensor(fig-7)

ESP- 8266 Board

Nodemcu-8266 board design as a small, affordable development board that allows to connect with the internet so easily via Wi-Fi with utilizing esp8266 cheap. It basically designs for Audino IDE project and python-based project. It powerful board for this type of project.

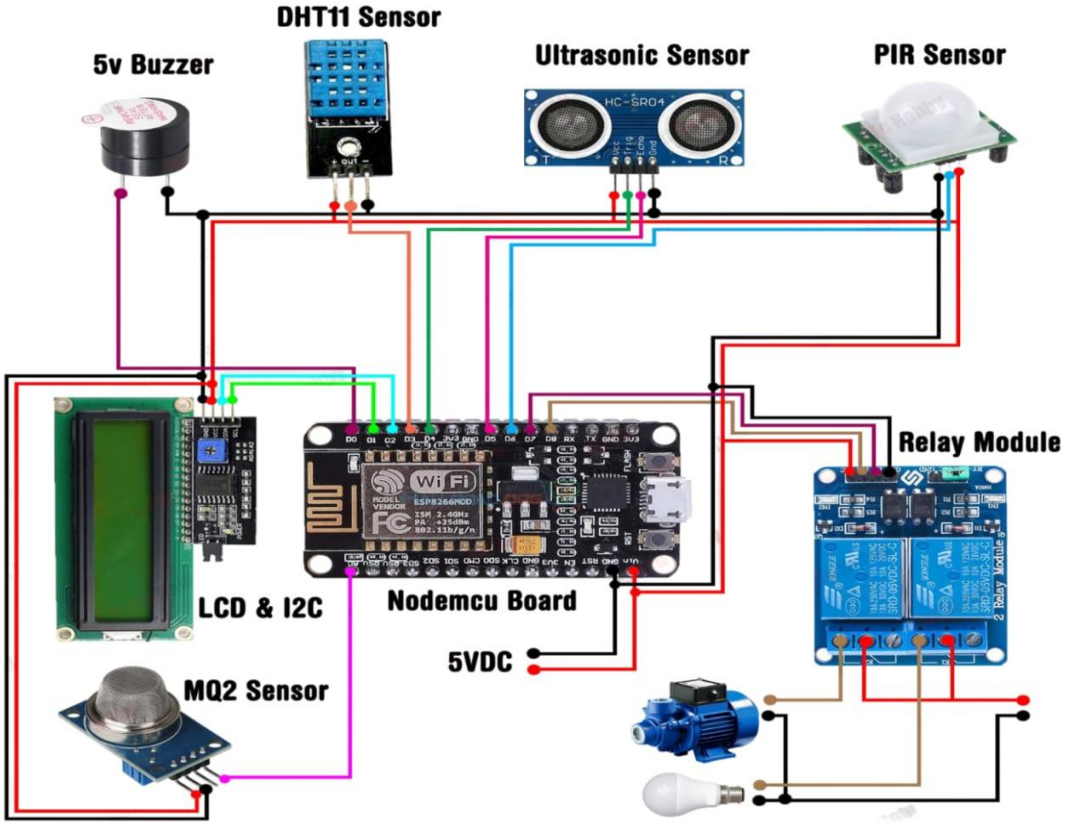
ESP- 8266 Board(fig-8)

Relay Module

A relay module's primary benefit is its ability to control high-power circuits using a low-power signal, effectively isolating the control circuit from the load circuit.

Relay (Fig-9)

3.3Circuit Diagram & System Working Process:



Circuit Diagram (Fig-10)

The circuit diagram shown in the figure above shows how all the sensors are connected to the ESP8266. This section will describe how the connections are made. First, we will connect the ground and power connections of all these sensors (including the relay module, MQ2 sensor, LCD module, DHT-11 sensor, 5V buzzer, ultrasonic sensor, PIR sensor) to the ground and power supply of the Esp-8266. Next, we will connect the Pair sensor, DHT-11 sensor, MQ-2 sensor, and 5V buzzer sensor to ESP8266 pin number D6, pin number D3, pin number A0, and pin number D0, respectively. The remaining two connection pins of the LCD module, the SDA pin and the SCL pin, will be connected to the pins numbered D2 and D1 of the ESP-8266, respectively. We will connect pin IN1 and pin IN2 of the relay module to pin numbers d7 and d8 of the ESP8266 respectively. Finally, if we connect an external source of five volts to this circuit, our entire system will receive power and run.

3.4Costs in System Design:

Table number-1 below shows the cost of building this system, including the cost of each sensor. Every effort was made to bill the system as low as possible.

Table-1

Serial Number	Sensors or parts name	Cost
1.	DHT-11	85 taka
2.	LDR Sensor	70 taka

3.	MQ-2 Sensor	110 taka
4.	ESP-8266	250 taka
5.	Relay Module	120 taka
6.	LCD Display with Module	250 taka
7.	PIR Sensor	80 taka
8.	5V Buzzer	10 taka
9.	Ultrasonic Sensor	70 taka
10.	5V DC source (Battery)	250 taka
11.	Connecting Wire	90 taka
12.	Stand	100 taka
	Total Cost :	**1,485 Taka(BDT)**

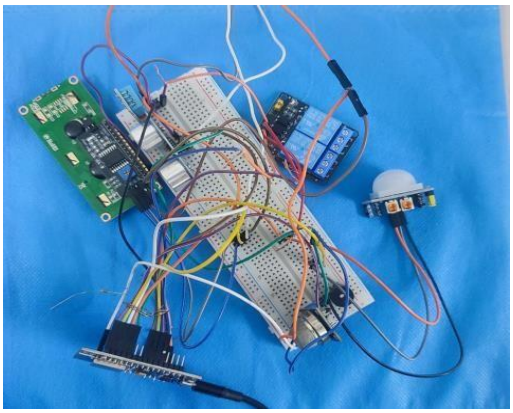
From table number-1 above we can see that to create this system it will take 1485 taka in Bangladeshi taka. We can easily say that this is a very low cost system. Any rural farmer in Bangladesh can buy this system and use it for his cultivation. Considering that the minimum income of any individual or farmer in Bangladesh is 10,000 taka as per the government data of 2024, we can say that farmers can easily buy this system and if the government provides this system to them for free, they will benefit even more. Since the cost of this system is very low, the government can easily provide it to the farmers for free.

3.5System Ability in Different Weather Conditions :

This system is capable of performing in all weather conditions from high temperatures to low temperatures and high water levels to low water levels, and it also provides an effective data in almost all cases. We have used this system in kutubdia, a southern region of Bangladesh, and found that it is capable of providing accurate data in high wind conditions. Not only that, this system uses a strong stand, through which it will be able to remain stable and provide data even during storms or floods. The percentage of effective value of this system or how it combines data is presented in the table in the results section [19].

3.6 Working progress

After all connection setup into the board then seems like to



. **Fig-11**

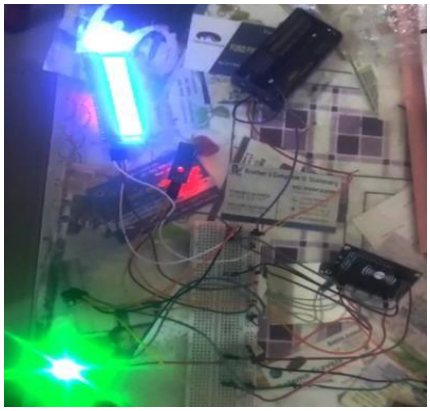


Fig-12

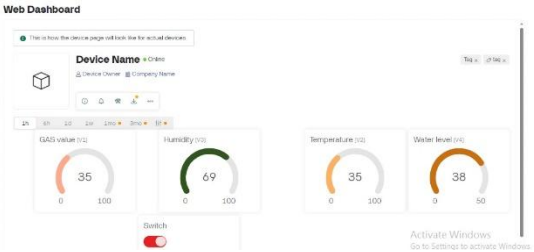
Here we can see that when all the connections are made, the power supply needs to be turned on to see if all the connections are successful. After turning on the power supply, we can see in the second picture that all the connections are fine. Also the all sensors were responding very well .

3.7Notification and Dashboard

After the successful implementation of the project, readings were taken in various ways and it responded very well. When the first measurement was taken, the temperature was 28.4 C, the humidity was 61%,pressure 300 and the water level was showing 100 because we had given more water at that time. Again, when I went to another area, some readings were also being given there, and we captured them. There, the temperature was showing 35 C, the humidity was 69, and the water level was 38. It was getting real-time measurements from the Blink ID server.

3.8 Process for Create Dashboard

We have used Blynk as a software in our system through which thirty thousand messages can be exchanged for free and through this Blynk we have created a dashboard. To create this dashboard, we will register in the blynk software and from there click on New Project. We will select the name of our project. In this case, we have selected the name of our project IoT based weather monitoring system. Then we will select our ESP-8266 board option and open a new control panel for each of our sensors where we will select our desired values, for example we will create one for humidity, one for gas sensor, one for water level and one for switching. Once our dashboard is created on the bylnk website, we will later create another control panel on our phone in the same way with the ID number of our dashboard and connect the phone's control panel to the dashboard we created via hotspot and upload the connected WiFi password and name to the programming, we will be able to control our system via the phone.



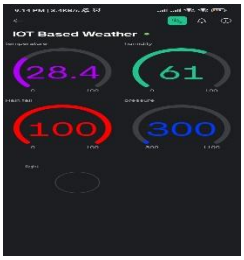


Fig-13(Dashboard) **Fig-14(Phone Control Panel)**

4.Result

This IoT based weather monitoring system developed with our own technology has been able to get the requested results after testing in several locations within Bangladesh, including salt farming located in Kutubdia, Cox's Bazar. Mushroom farming located in Rangpur and shrimp farming located in Pekua are

cultivated by farmers in this region. Our farmers can benefit from using this low-cost technology, which is presented through various graphs and tables in our results section.

4.1Salt Farming in Kutubdia, Cox's Bazar

Most of the people of Kutubdia Upazila, located in the southern part of Bangladesh, are involved in salt mining, through which they are achieving economic prosperity and contributing greatly to the economy of Bangladesh. But it is often seen that in Kutubdia Upazila and some other Upazilas of Cox's Bazar district, farmers do not get any immediate information about the water level due to not cultivating salt in a smart manner, resulting in wasting a lot of their money on salt[20]. In that case, if they are more efficient, then intelligent salt farming can make a huge contribution to our country's economy. Table 1 presents a sample of data storage using our best IoT weather monitoring system from a salt farm in Kutubdia.

Table 2: IoT based weather station in kutubdia salt farming filed (27 Jan 2025)

Time	Temperature	Humidity	Water level
12:00PM	22°C	50%	40
12:30PM	23°C	52%	50
1:00PM	24°C	54%	45

4.2Weather Forecast in Different location for Bangladeshi Farmers

By comparing different temperatures at different times in different regions of Bangladesh, this system will help farmers in those regions easily understand when they can economically benefit from growing which crops. Below are temperature charts for the months of May and June for three regions of Bangladesh, Lakshmipur, Netrakona, and Rangpur, as shown in Figure 1 and Figure 2.

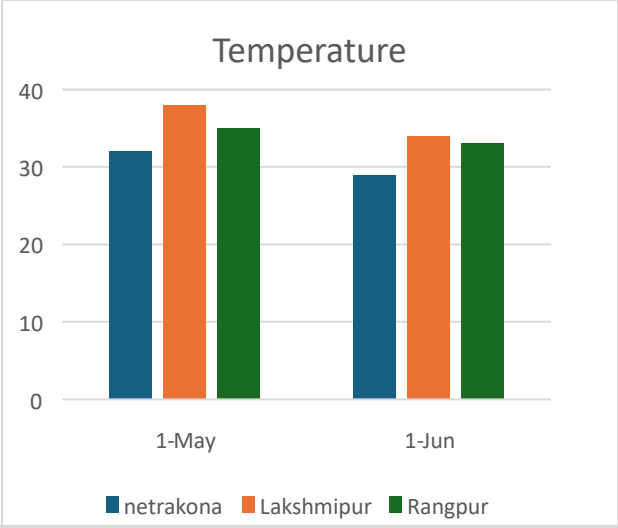


Fig -15

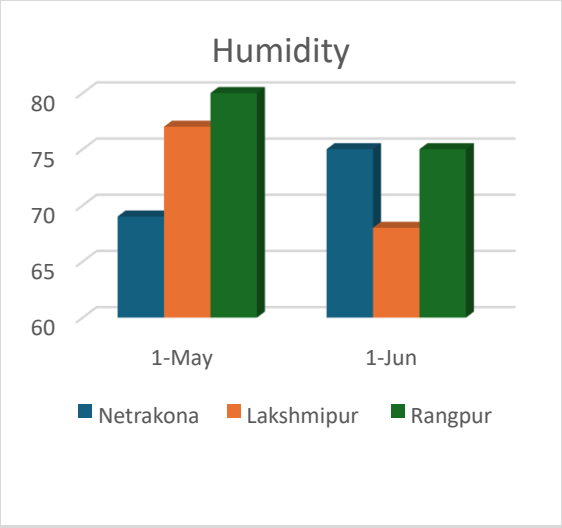


Fig -16

4.3System Performance Compare

Our system uses the maximum number of sensors to get accurate information. In market existing system we did not see the use of any ultrasonic sensors, but our system has ultrasonic sensors, due to which our system is able to easily measure the height of the water, which makes our system superior to other systems. Table number three below shows the overall performance of each sensor in our system.

Table-3(Performance):

Sensor Name	No.Test	No Response	Accurate value	Efficiency
PIR Sensor	20	20	19	95%
Ultrasonic Sensor	20	20	18	90%
MQ-2 Sensor	20	20	19	95%
DTH11 Sensor	20	20	19	95%

By reviewing the performance of the table-3, we can say that our IoT based weather monitoring system works effectively compared to other existing systems in the market which will play a very important role for the agriculture sector**.**

5.Discussion

IoT Best Weather Monitoring System is used in agriculture, smart city, marine navigation, disaster management, environmental and many more fields[21]. The importance of using IoT based weather monitoring system is immense.

5.1Application Of IoT Based Weather Monitoring System

The following figure -3 shows where IoT based weather monitoring systems are used and a brief description of them.

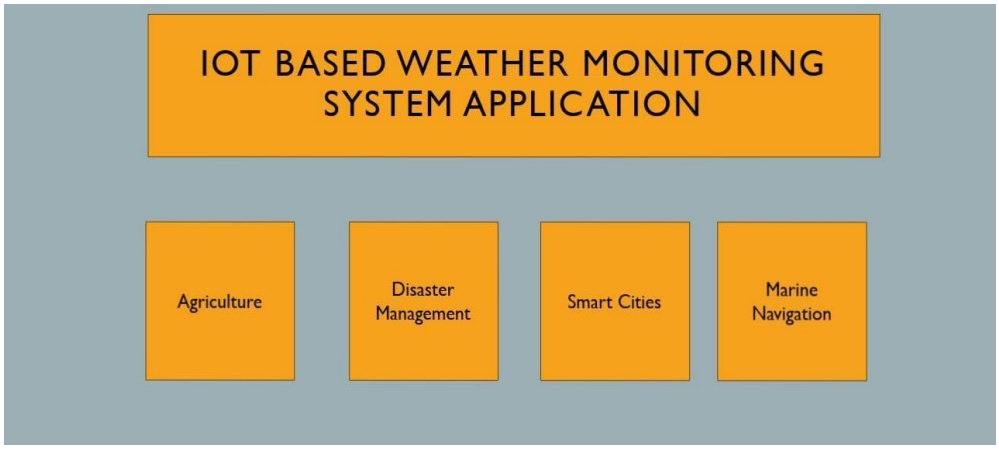


Fig-17:(Application)

Agriculture : The scope of use of weather monitoring systems in the agricultural sector is wide, from starting shrimp farming to mushroom farming, salt farming and which crops will grow well in the temperatures of that region, automatic notification via email when greenhouse gas temperatures change, and much more.

Disaster Management : IoT-based weather monitoring systems can also be used in disaster management, such as providing real-time information on flash floods to data centers, so that advance management can be taken to avoid floods.

Smart Cities : By using monitoring systems in smart cities, people in particular areas of the city will be able to know about the situation in real time and they will be able to manage all this accordingly.

Marine Navigation : Weather monitoring systems for maritime navigation can be called life-saving systems because through this system, weather conditions are known instantly and accordingly, sailors can also predict them and determine their destination based on the weather.

The Internet of Things (IoT) represents a powerful engine driving the data and communication generation industry, with its influence spreading worldwide. We have reviewed numerous technical and non-technical works to assess the current state of scholarly discourse, employing analysis sequence procedure models to prioritize IoT research areas[22]. In today's era, many prominent industries and companies are striving to advance this technology. It's crucial to enhance agricultural productivity, boost crop yields, and reduce human labor through the implementation of IoT techniques. IoT technology has the potential to significantly improve the efficiency of our agricultural systems. Consequently, research on IoT in agriculture is of paramount importance[23].

5.2Cost Benefit analysis :

From Table No. 1 in the Methods section, we can see that it cost us only 1485 Bangladeshi taka to build this system, which is much cheaper than all the existing IoT based weather monitoring systems in Bangladesh. Due to its low price, this system can be easily purchased by rural farmers in Bangladesh, such as Kutubdia, Lakhimpur , Rangpur, Kutubdia and other regions of Bangladesh. In this case, there will be no need for them and those who do business with this weather monitoring system to collect data from local areas. Due to its low cost, if its use starts in one rural area of Bangladesh, then its use will gradually spread to all rural areas of Bangladesh. Through this, mushroom cultivation and shrimp farming will be inspired in local areas of Bangladesh and many new mushroom cultivation and shrimp farming farms will be created, which will greatly reduce the unemployment rate in local areas.

5.3Future Implementation :

The Bangladesh government can financially support rural farmers to engage in this IoT-based weather monitoring system. In that case, the Bangladesh government can work with NGOs or create a system through which village families can purchase this weather monitoring system through low-interest loans. For example, the Bangladesh government can establish a large weather observation system in Kutubdia and connect many weather observation systems through a cloud-based system that can always store real-time information of meteorologists, which will later be useful for high-quality research work. Use machine learning to predict future weather patterns more accurately and provide advanced insights into crop management. Link the system with other agricultural platforms like market prices or pest control solutions. Partner with local agricultural departments to improve the system’s outreach and impact.

6.Conclusion :

In this research, the development and testing of the IoT-based weather monitoring system has been carried out, through which we can see that it is carrying very important information for our agricultural products and agricultural farms. By delivering its data in real time, we can see that a farm can easily and immediately receive accurate data about its products, which can prevent product waste and increase efficiency, while also increasing crop production. Through the effectiveness of this technology will carry an important aspect for Bangladeshi farmers.

Through the widespread use of such systems in the future, we can bring unprecedented changes and development to agricultural work, and it will also play a good role in raising awareness of climate change. Although this system is very important and will benefit the farmers of our country, the system can be

improved in some other areas. The collaboration of the sensors can be done more quickly so that we can get the data more accurately. In addition, the weather forecast period can be extended further so that our farmers can know well in advance what the next weather will be like and can take action accordingly. And we can look into how this system can be used on a larger scale. Our government also can do experimental use of this system in wide variety of agriculture fields like cow farming. As in Bangladesh the cow farming is growing day by day our farmers need to the accurate temperature in summer and winter duration. Also this system can be further improved to the bad weather condition.

7.Reference :

1.Holovatyy, A., 2021. Development of IOT weather monitoring system based on Arduino and ESP8266 Wi-Fi

2.Annika, V. O. (2023). Climate change and food security in sub-Saharan Africa: evolving African-based adaptability strategies. Journal of African Studies and Sustainable Development.

3.Islam M.S., Sunny M.S., Rabbi J.H., Pritom N.U., Shaowkat M.W. (2024) Arduino Based Sun-Light Detection, International Journal of Engineering and Advanced Technology Studies 12 (2), 30-42.

4.Bedair, H., Alghariani, M.S., Omar, E., Anibaba, Q.A., Remon, M.,Bornman, C., Kiboi, S.K., Rady, H.A., Salifu, A.M.A., Ghosh, S. and Guuroh, R.T., 2023. Global warming status in the African continent: sources, challenges, policies, and future direction. International Journal of Environmental Research, 17(3), p.45.

5.Kokulan, V., Akinremi, O.O. and Moulin, A.P., 2022. The seasonality of nitrate and phosphorus leaching from manure and chemical fertilizer added to a chernozemic soil in Canada (Vol. 51, No. 6, pp. 1259-1269).

6.Kodali, R. K., Rajanarayanan, S. C., & Boppana, L. (2019, December). IoT-based weather monitoring and notification system for greenhouses. In 2019 11th International Conference on Advanced Computing (ICoAC) (pp. 342-345). IEEE.

7.Joseph, F. J. J. (2019). IoT-based weather monitoring system for effective analytics. International Journal of Engineering and Advanced Technology, 8(4), 311-315.

8.Islam M.S., Sunny M.S., Rabbi J.R., Pritom N.U., Shaowkat M.W. (2024) Future Assessment of Low Cost EV Automobile Market in Bangladesh, International Journal of Engineering and Advanced Technology Studies, 12 (2), 12-29.

9.An IOT Based Weather Monitoring System1Dhannjay Verma,Ishan Choudhury,3Manish Singh, 4Abhijeet Shukla, 5Dharendra Kumar [12345]B.Tech[12345] Electrical and Electronics, [12345] Galgotia's College of Engineering and Technology, Greater Noida, India

10.Internet of Things (IOT) based Weather Monitoring System artment of Electronics and Communication, NIEIT, Mysuru Andreanna Grace Shires Department of Electronics and Communication, NIEIT,

11.A REVIEW PAPER ON ONLINE WEATHER MONITORING SYSTEM USING INTERNET OF THINGS Muskan Choudhary*1, Prof. Shivendu Dubey*2*1Department of CSE, GGITS, Jabalpur, India.*2Guide, Department of CSE, GGITS, Jabalpur, India.

12.Real Time Weather Monitoring System using IoT M. Sreerama Murthy1, R. P. Ram Kumar1, Billa Saikiran2*, Islavath Nagaraj2, Tejesh Annavarapu21Department of AIMLE, GRIET, Hyderabad, Telangana, India2UG Student, Department of AIMLE, GRIET, Hyderabad, Telangana, India

13.Internet of Things (IOT) based Weather Monitoring System Girija C Department of Electronics and Communication, NIEIT, Mysuru Andreanna Grace Shires Department of Electronics and Communication, NIEIT, Mysuru.

14.F. Meneghello, M. Calore, D. Zucchetto, M. Polese and A. Zanella, "IoT: Internet of Threats? A Survey of Practical Security Vulnerabilities in Real IoT Devices," in IEEE Internet of Things Journal, vol. 6, no. 5, pp. 8182-8201, Oct. 2019. doi: 10.1109/JIOT.2019.2935189

15.P. Fremantle and P. Scott, “A survey of secure middleware for the Internet of Things,” PeerJ Computer Science, vol. 3, p. e114, May 2017.

16.Y. Liu, Y. Kuang, Y. Xiao and G. Xu, "SDN-Based Data Transfer Security for Internet of Things," in IEEE Internet of Things Journal, vol. 5, no. 1, pp. 257-268, Feb. 2018 doi:10.1109/JIOT.2017.2779180

17.International Journal of Advanced Research in Computer and Commu-nication Engineering ISO3297:2007 Certified Vol. 5, Issue 9, September 2016.

18.Jitendra Singh , Rehan Mohammed , Mradul Kankaria , Roshan Panchal, Sachin Singh , Rahul Sharma, “Arduino Based Weather Monitoring System”, International Journal of Advanced in Management, Technology and Engineering Sciences 3, vol. 8, 2018.

19.Shubham R. Valentia, Vaibhav R. Wankhade, Pranjali G. Wangekar,Nikhil S. Mundane. “IoT Based Weather Monitoring System using Rasp-berry Pi.” International Research Journal of Engineering and Technology (IRJET) 1, vol. 06, 2019.

20. Jiang H, Shu H (2019) Optical remote-sensing data based research on detecting soil salinity at diferent depth in an arid-area oasis, Xinjiang, China. Earth Sci Informatics 12(1):43–56.

21.Li, Y., Ding, Y., Li, D., & Miao, Z. (2018). Automatic carbon dioxide enrichment strategies in the greenhouse: A review. Biosystems Engineering, 171, 101–119.

22.K. Mekki, E. Bajic, F. Chaxel, and F. Meyer, “A comparative study of LPWAN technologies for large-scale IoT deployment”, ICT Express, vol.5, no. 1, pp. 1–7, 2019.

23.Islam M.S., Sunny M.S., Rabbi J.R., Pritom N.U., Shaowkat M.W. (2024) Future Assessment of Low Cost EV Automobile Market in Bangladesh, International Journal of Engineering and Advanced Technology Studies, 12 (2), 12-29.