



TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE NUEVO LEÓN
División de Estudios Profesionales

Trabajo de Titulación
Titulación Opción I: Tesis

Proyecto: "Adaptación de algoritmo evolutivo para resolver problemas de la calidad del aire en zonas industriales del área metropolitana de Monterrey."

ALUMNO(S):

Mario González Rodríguez

No. CONTROL:

16480863

CARRERA:

Ingeniería en Sistemas Computacionales

ASESOR DE RESIDENCIA:

Dr. José Isidro Hernández Vega

REVISORES:

Guadalupe, N.L.

Junio, 2021



Aceptación de documento de Tesis

Cd. Guadalupe, Nuevo León, **9/Febrero/2021**

ING. MAGALY BENÍTEZ TAMEZ

JEFA DE DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN

PRESENTE:

La Comisión de Revisión de Tesis nos es grato comunicarle que, conforme a los lineamientos de los planes de estudio del 2010 del Tecnológico Nacional de México para la obtención del grado de Ingeniería en Sistemas Computacionales de este Instituto, y después de haber sometido a revisión académica el proyecto de Tesis titulado: **“Adaptación de algoritmo evolutivo para resolver problemas de la calidad del aire en zonas industriales del área metropolitana de monterrey”**, realizado por **Mario González Rodríguez**, Número de Control: **16480863**, dirigida por el **Dr. José Isidro Hernández Vega**, y habiendo realizado las correcciones que le fueron indicadas, acordamos **ACEPTAR** el documento final de proyecto de Tesis. Así mismo le solicitamos tenga a bien extender la documentación correspondiente para continuar el proceso de titulación integral por tesis del sustentante.

Sin otro particular, agradecemos la atención.

ATENTAMENTE

Excelencia en Educación Tecnológica
“CIENCIA Y TECNOLOGÍA AL SERVICIO DEL HOMBRE”

DIRECTOR DE TESIS

DR. JOSÉ ISIDRO HERNÁNDEZ VEGA

**DOCTORADO EN INGENIERÍA CON ORIENTACIÓN EN TECNOLOGÍAS DE LA INFORMACIÓN
CÉDULA: 12058578**

REVISOR

M. C. ELDA REYES VARELA
**MAESTRÍA EN CIENCIAS EN COMERCIALIZACIÓN
DE LA CIENCIA Y LA TECNOLOGÍA**
CÉDULA: 09093449

REVISOR

ING. LUIS ALEJANDRO REYNOSO GUAJARDO
INGENIERÍA EN SISTEMAS COMPUTACIONALES
CÉDULA: 5157710

c.c.p Departamento de Sistemas y Computación
c.c.p. Expediente
c.c.p. Interesados



Eloy Cavazos No. 2001 Col. Tolteca, C.P. 67170,
Guadalupe, Nuevo León Tel. (81) 8157 0500
www.tecnm.mx | nuevoleon.tecnm.mx



RESUMEN

En la actualidad la contaminación del aire es un problema que aqueja a la población en general debido a los efectos generados en la salud. Existen estudios que muestran un incremento en la tasa de mortalidad debido a problemas respiratorios generados por esta causa. Por lo cual es importante que el monitoreo de esta información esté al alcance de la población para prevenir problemas futuros.

Este proyecto abordó la problemática de monitoreo de contaminantes criterio mediante una red de sensores en una zona específica. Se implementó un algoritmo evolutivo diferencial para búsqueda óptima de concentración del contaminante Partículas Suspendidas de 10 micras (PM10), facilitando el análisis de los datos recabados en la detección de las concentraciones máximas y mínimas de los contaminantes en un periodo de tiempo analizado.

Mediante los datos históricos de los contaminantes recabados fue posible conocer el comportamiento de los contaminantes de estudio.

Dentro de las actividades realizadas para el proyecto se realizó la representación de la solución para el algoritmo evolutivo, se identificó la función de calidad del problema, se definió la población de soluciones, se establecieron mecanismo de selección y operadores de cruza para el algoritmo, se realizaron pruebas experimentales y se realizó un análisis de resultados con el algoritmo programado. Los resultados obtenidos con el algoritmo evolutivo permitieron dar estimaciones óptimas de los niveles de contaminantes estudiados bajo un esquema de predicción.

PALABRAS CLAVE: Algoritmos evolutivos, evolutivo diferencial, monitoreo contaminantes, partículas suspendidas de 10 micras(PM10).

ABSTRACT

Air pollution is a problem that currently affects the population due to the problems it generates to health. There are studies that show an increase in the mortality rate from respiratory problems generated by this cause. For this reason, monitoring air pollution is important to be available to the population to prevent future problems.

This project addresses the problem of monitoring criteria pollutants through a network of sensors in a specific area. A differential evolutionary algorithm was implemented to search for the optimal concentration of the pollutant Suspended particles of 10 microns (PM10), facilitating the analysis of the data collected in the detection of the maximum and minimum concentrations of pollutants in a period of time analyzed. Through the historical data of the pollutants collected it was possible to know the behavior of the pollutants studied.

Among the activities carried out for the project, the representation of the solution for the evolutionary algorithm was performed, the quality function of the problem was identified, the population of solutions was defined, selection mechanism and crossover operators were established for the algorithm, experimental tests were carried out and an analysis of the results was carried out with the programmed algorithm. The results obtained with the evolutionary algorithm allow to give optimal estimates of the pollutant levels studied under a prediction scheme.

KEY WORDS: Evolutionary algorithms, differential evolutionary, pollutant monitoring, 10 micron suspended particles (PM10).

DEDICATORIAS

Dedico esta tesis a mis padres y a mi hermano por haberme convertido en la persona que soy actualmente, sin su apoyo no habría podido conseguir los logros que he obtenido hasta ahora.

AGRADECIMIENTOS

Agradezco a todas las personas que han contribuido a quien soy ahora, a mis padres Andrés y Lucy y a mi hermano Andrés por el apoyo que me han dado y que siempre han estado ahí durante toda mi vida.

Al Doctor José Isidro Hernández Vega, asesor de este proyecto, le agradezco por todo el apoyo, enseñanzas y tiempo que me ha brindado hasta ahora durante la realización de este periodo de residencia profesional y realización de tesis.

A los profesores Elda Reyes Varela, Marta Alicia Casillas Careaga, Arturo Hinojosa Ramírez, Ma. de Lourdes Leyva Cerda, Juan Ángel Gutiérrez Delgado y al resto de mis instructores que dedicaron sus enseñanzas durante mis estudios y que por los cuales aprendí las bases de Ingeniería en Sistemas Computacionales.

A mis amigos y compañeros durante mis estudios en la carrera, por los cuales aprendí y me divertí tanto durante este trayecto.

A mis compañeros de residencia Jonathan y Jessica con quienes gracias a sus contribuciones y ayuda se logró concluir satisfactoriamente el proyecto.

Agradezco también al Instituto Tecnológico de Nuevo León y a su personal administrativo por las atenciones y buen trato que he recibido durante toda mi vida estudiantil, por ese lugar que durante 4 años llegue a considerar una segunda casa.

CONTENIDO

RESUMEN	i
ABSTRACT	ii
DEDICATORIAS	iii
AGRADECIMIENTOS	iv
CONTENIDO	v
ÍNDICE DE FIGURAS	viii
ÍNDICE DE TABLAS	x
CAPITULO I- INTRODUCCION	1
1.1 Motivación del problema	1
1.2 Planteamiento del problema a resolver.....	2
1.2.1 Descripción del problema.....	2
1.3 Antecedentes	3
1.3.1 Trabajos relacionados con el tema de investigación	3
1.3.2 Diferencia de los trabajos relacionados con respecto a la propuesta de la tesis	6
1.4 Hipótesis.....	7
1.5 Objetivos.....	7
1.6 Justificación del proyecto.....	8
1.7 Impacto o beneficio en la solución a un problema relacionado con el sector productivo o la generación de conocimiento científico o tecnológico.....	9
1.8 Lugares en donde se desarrolla el proyecto	9
1.9 Infraestructura.....	9
CAPITULO II- MARCO TEÓRICO	10
2.1 Principios de optimización	10
2.1.1 Técnicas de optimización	10
2.1.2 Programación Lineal	11
2.1.3 Programación no lineal.....	12

Componentes de un algoritmo bio-inspirado.....	12
2.2 Algoritmos evolutivos y algoritmos genéticos	13
2.2.1 Definiciones y características de algoritmos genéticos y evolutivos.....	13
2.3 Algoritmos Evolutivos.....	15
2.3.1 Introducción al tema e historia	16
2.3.2 Representación de la solución	17
2.3.3 Función de aptitud.....	17
2.3.4 Población de soluciones.....	17
2.3.5 Mecanismos de selección de padres	17
2.3.6 Cruza y mutación	17
2.3.7 Mecanismo de reemplazo	18
2.3.8 Inicialización y Terminación	18
2.4 Herramientas para programar algoritmos evolutivos	19
2.4.1 EA Visualizer	19
2.4.2 ESCaPaDE	20
2.4.3 GAGA.....	20
2.5 Algoritmo Evolutivo Diferencial (DE)	21
2.5.1 Visión General.....	21
2.5.2 Elementos y comportamiento	22
2.5.3 Ejemplo y funcionamiento	23
2.6 Índice de calidad de aire	26
CAPITULO III. METODOLOGIA DE SOLUCIÓN	30
3.1 Análisis del problema.....	30
3.2 Diseño de la solución.....	30
3.2.1 Representación de la solución.	30
3.2.2 Identificar función de calidad.	30
3.2.3 Población de soluciones.....	32
3.2.4 Mecanismo de selección de padres (selección).....	32

3.2.5 Operadores de variación (cruza y/o mutación).....	32
3.2.6 Mecanismo de reemplazo (selección de sobrevivientes).	33
CAPITULO IV- EXPERIMENTACION Y ANÁLISIS DE RESULTADO	36
4.1 -Experimento 1- “Knapsack Problem”.....	36
4.1.1 Análisis del problema	36
4.1.2 Diseño de la solución	37
4.1.3 Implementación	38
4.1.4 Pruebas del algoritmo en el primer experimento	45
4.1.5 Conclusiones del primer experimento	50
4.2 Experimento 2- Prediccion de niveles del contaminante PM10	50
4.2.1 Procedimiento y descripción de las actividades realizadas	50
4.2.2 Implementación	53
4.2.3 Pruebas.....	65
4.2.4 Conclusiones de los resultados obtenidos.	79
CAPITULO V- CONCLUSIONES Y TRABAJOS FUTUROS	80
5.1 Conclusiones de Proyecto.....	80
5.2 Trabajos futuros	82
FUENTES DE INFORMACIÓN	83
ANEXOS A: Código del programa	86
Clase Individuo.....	86
ANEXOS B: Código del experimento 1.....	87
Clase EvolucionDiferencial.....	87
ANEXOS C: Código del experimento 2.....	92
Clase Fitness	92
Clase PWDE:	93
Clase CalcularPromedioMovil	102
Clase Main	103

ÍNDICE DE FIGURAS

Figura 1. Representación de porcentaje de Optimización.	14
Figura 2. Traficación esquema Evolución Diferencial	22
Figura 3. Tabla de ejemplo de Evolución Diferencial.....	24
Figura 4. Elección de parámetros para el vector U.....	33
Figura 5. Diagrama general de funcionamiento del algoritmo evolutivo propuesto.....	35
Figura 6. Función objetivo del knapsack problem.....	37
Figura 7. Restricciones del problema.	37
Figura 8. Población inicial aleatoria	46
Figura 9. Resultados del experimento 1.	47
Figura 10. Resultados del Experimento 2.....	47
Figura 11. Resultados del Experimento 3.....	48
Figura 12. Frecuencia de Soluciones Optimas encontradas en el experimento 3.	48
Figura 13. Frecuencia de generación en que el óptimo fue encontrado.....	49
Figura 14. Estadísticos obtenidos del experimento 4.	49
Figura 15. Ficheros de texto con los datos de entrada del algoritmo.....	54
Figura 16. Comparación de los resultados del día 1 de enero, estimaciones de los niveles de PM10 dadas por el algoritmo y valores reales captados para cada intervalo de tiempo.	67
Figura 17. Comparación de los niveles del contaminante PM10 con los valores mínimo y máximo estimados.	68
Figura 18. Comparación de los resultados del día 2 de febrero, estimaciones de los niveles de PM10 dadas por el algoritmo y valores reales captados para cada intervalo de tiempo.	69
Figura 19. Comparación de los niveles del contaminante PM10 con los valores mínimo y máximo estimados.	70
Figura 20. Comparación de los resultados del día 27 de agosto, estimaciones de los niveles de PM10 dadas por el algoritmo y valores reales captados para cada intervalo de tiempo.	71

Figura 21. Comparación de los niveles del contaminante PM10 con los valores mínimo y máximo estimados.	72
Figura 22. Comparación de los resultados del día 28 de agosto, estimaciones de los niveles de PM10 dadas por el algoritmo y valores reales captados para cada intervalo de tiempo.	73
Figura 23. Comparación de los niveles del contaminante PM10 con los valores mínimo y máximo estimados.	74
Figura 24. Comparación de los resultados del día 27 de noviembre, estimaciones de los niveles de PM10 dadas por el algoritmo y valores reales captados para cada intervalo de tiempo.	75
Figura 25. Comparación de los niveles del contaminante PM10 con los valores mínimo y máximo estimados.	76
Figura 26. comparación de los resultados del día 3 de diciembre, estimaciones de los niveles de PM10 dadas por el algoritmo y valores reales captados para cada intervalo de tiempo.	77
Figura 27. Comparación de los niveles del contaminante PM10 con los valores mínimo y máximo estimados.	78

ÍNDICE DE TABLAS

Tabla 1. Normativas de contaminantes criterio.....	26
Tabla 2. Redondeo de cifras significativas para contaminantes criterio.	27
Tabla 3. Concentraciones base para contaminantes criterio.	27
Tabla 4. Niveles del Índice Aire y Salud.....	28
Tabla 5. Fórmulas de color para el índice aire y salud.	29
Tabla 6. Comparación de los resultados obtenidos del 1 de enero, niveles del contaminante PM10, la estimación dada por el algoritmo evolutivo y su margen de error absoluto.....	67
Tabla 7. Resultados 1 de enero utilizando un rango aproximado para los niveles del contaminante PM10 con un mínimo y máximo estimado.....	68
Tabla 8. Comparación de los resultados obtenidos del 2 de febrero, niveles del contaminante PM10, la estimación dada por el algoritmo evolutivo y su margen de error absoluto.....	69
Tabla 9. Resultados 2 de febrero utilizando un rango aproximado para los niveles del contaminante PM10 con un mínimo y máximo estimado.....	70
Tabla 10. Comparación de los resultados obtenidos del 27 de agosto, niveles del contaminante PM10, la estimación dada por el algoritmo evolutivo y su margen de error absoluto.....	71
Tabla 11. Resultados 27 de agosto utilizando un rango aproximado para los niveles del contaminante PM10 con un mínimo y máximo estimado.....	72
Tabla 12. Comparación de los resultados obtenidos del 28 de agosto, niveles del contaminante PM10, la estimación dada por el algoritmo evolutivo y su margen de error absoluto.....	73
Tabla 13. Resultados 28 de agosto utilizando un rango aproximado para los niveles del contaminante PM10 con un mínimo y máximo estimado.....	74
Tabla 14. Comparación de los resultados obtenidos del 27 de noviembre, niveles del contaminante PM10, la estimación dada por el algoritmo evolutivo y su margen de error absoluto.....	75
Tabla 15. Resultados 27 de noviembre utilizando un rango aproximado para los niveles del contaminante PM10 con un mínimo y máximo estimado.....	76
Tabla 16. Comparación de los resultados obtenidos del 3 de diciembre, niveles del contaminante PM10, la estimación dada por el algoritmo evolutivo y su margen de error absoluto.....	77

Tabla 17. Resultados 3 de diciembre utilizando un rango aproximado para los niveles del contaminante PM10 con un mínimo y máximo estimado..... 78

CAPITULO I- INTRODUCCION

1.1 Motivación del problema

Monterrey se encuentra ubicado al noreste de México, es la capital del estado de Nuevo León. Al año 2015 contaba con una población de 1,133,814 habitantes; sin embargo, la zona conurbada, integrada por la ciudad de Monterrey y otros 11 municipios de Nuevo León (Apodaca, Escobedo, Guadalupe, Monterrey, San Nicolás de los Garza, San Pedro Garza García, García, Santa Catarina, Santiago, Salinas Victoria, y Juárez) cuenta con aproximadamente 4 millones 700 mil habitantes (INEGI, 2015).

La ciudad es sede de importantes grupos industriales y financieros, por lo que en ella operan industrias de diversa índole. Está comunicada por autopistas con la frontera con los Estados Unidos (autopistas a Nuevo Laredo y a Reynosa), al Golfo de México (carretera Panamericana) y al resto del país (troncales Saltillo-Torreón-Mazatlán y

Matehuala-San Luis Potosí-Ciudad de México). Cuenta con servicio de carga por ferrocarril hacia la ciudad y puerto de Tampico y otras zonas del país.

El análisis de la información proveniente de la red de monitoreo del AMM indica que, históricamente, en esta zona metropolitana se rebasan las normas de calidad de aire establecidas por la Secretaria de Salud (SS) principalmente para las partículas aerodinámicas con diámetro menor a 10 micras (PM10), en menor medida para el ozono (O3) y para monóxido de carbono (CO). (INECC, 2010).

Una de las problemáticas derivadas del monitoreo de contaminantes es que no se identifican a tiempo las fuentes fijas que están sobrepasando los límites permitidos, además de predecir cuál es el comportamiento de los contaminantes en un periodo de tiempo. Actualmente las estaciones de monitoreo fijo presentan diversas limitantes que se describen en seguida.

La red de estaciones meteorológicas fijas, presenta límites de alcance de operación y monitoreo, pueden cubrir 2 km² de radio por 25 metros de alto. La limitante que más afecta la ejecución del monitoreo es la altura en su alcance. Por lo tanto, no se tiene un conocimiento real de la cantidad y tipo de contaminantes que se están emitiendo a la atmosfera, con el paso del tiempo se mezclan con

otros contaminantes para después caer a la tierra por algún otro medio natural (lluvia, vientos, etc.) y afectar la salud humana.

El alcance de estos contaminantes que emiten los parques industriales, se desconoce su afectación en viviendas que estén próximas a estas zonas industriales. Es importante conocer los niveles de contaminación a los que se ven afectadas las personas, su concentración en un periodo de tiempo ya que esto puede influir en cierta medida la vida de los habitantes de zonas cercanas.

Se abordó la problemática de monitoreo puntual por zonas específicas bajo un histórico de mediciones del contaminante criterio de partículas suspendidas, dándole una solución alterna de monitoreo mediante algoritmos evolutivos. Implementar y codificar un algoritmo evolutivo mediante la técnica diferencial facilitó el análisis del comportamiento de los contaminantes de estudio bajo datos históricos. Identificando emisiones mínimas y máximas, siguiendo la normativa NOM-172-SERMANAT-2019 vigente en México.

1.2 Planteamiento del problema a resolver

1.2.1 Descripción del problema

El problema se plantea de la siguiente manera: Dada una red de sensores que se encuentran monitoreando los contaminantes ambientales criterio en una zona del área metropolitana de Monterrey próxima a una zona industrial. Se cuenta con una base de datos histórica con la información captada por estos nodos, esta base de datos está ya en un formato digital para ser procesada para explotación de información para la toma de decisiones.

El gran volumen de datos no permite realizar un análisis rápido y flexible por algoritmos tradicionales de búsqueda y optimización. El conjunto de datos no es fácil de manipular, debido a su tamaño y complejidad.

En este problema de contaminación ambiental se buscó encontrar la mejor alternativa de entre un conjunto de posibilidades. El espacio de búsqueda (número de posibles soluciones) es demasiado grande, lo que puede ocasionar que el tiempo para encontrar la mejor solución sea muy elevado. El área de la investigación de operaciones, ofrece un conjunto de técnicas para resolver

problemas de optimización. Estas técnicas proveen resultados altamente competitivos cuando las características del problema son afines a la forma de trabajar de la técnica en cuestión. Cuando ellas no proveen resultados competitivos o su aplicación es difícil, se puede recurrir a soluciones heurísticas. Los algoritmos evolutivos son capaces de encontrar buenas soluciones en problemas de optimización con un costo computacional razonable. Para el problema es importante conocer cómo ha sido el comportamiento de los contaminantes criterio, para la tesis se tomó el de partículas suspendidas PM10 en determinada fecha, hora, cantidad de concentración mínima y máxima del conjunto de datos de la red de sensores.

A través de datos históricos de los contaminantes podemos conocer cuál ha sido el comportamiento mínimo y máximo de partículas suspendidas PM10.

Se puede presentar este análisis del comportamiento de los contaminantes mediante algoritmos evolutivos, analizando resultados y así tener una información significativa para la toma de decisiones, aplicando protocolos de contingencia o normas necesarias de acuerdo al caso.

1.3 Antecedentes

1.3.1 Trabajos relacionados con el tema de investigación

En el trabajo Aplicación de Técnica de Inteligencia Artificial a la Predicción de Contaminantes Atmosférico de M. G. Januch (2012) propone un modelo de predicción de las concentraciones de los contaminantes SO₂ y PM10 para cada caseta de monitorización de la ciudad de Salamanca, México. Los modelos propuestos utilizan información real de las concentraciones de los contaminantes y las variables meteorológicas obtenidas de la Red de Monitorización Atmosférica de Salamanca (REDMAS). Cada modelo propuesto utiliza técnicas de inteligencia artificial y reconocimiento de patrones, que permitan utilizar la información obtenida por la REDMAS y las variables meteorológicas para predecir las concentraciones de los contaminantes que se han reportado como preocupantes en la región.

Modelado de partículas PM10 Y PM2.5 mediante redes neuronales artificiales:

En su trabajo “Modelado de partículas PM10 y PM2.5 mediante redes neuronales artificiales sobre clima tropical de San Francisco de Campeche, A.A. Espinoza (2017) desarrolla una metodología computacional basada en redes neuronales con 3 capas. Esta arquitectura utiliza una base de datos tomando los siguientes campos: Días de la semana, tiempo del día, temperatura ambiente, presión atmosférica, velocidad del viento, dirección del viento, humedad relativa y radiación solar. La mejor red neuronal fue compuesta por 30 neuronas en una capa oculta usando el algoritmo de optimización de Levenberg-Marquardt logrando generar predicciones con un coeficiente de determinación de 93.01% para PM2.5 y 90.10% para PM10.

Evolución Diferencial para resolver problemas Multiobjetivo:

L.V Santana en su tesis “Un Algoritmo Basado en Evolución Diferencial para Resolver Problemas Multiobjetivo “(2004) plantea un método de Evolución diferencial para resolver problemas Multiobjetivo, en este tipo de problemas se busca optimizar dos o más funciones las cuales se interponen entre sí. El algoritmo propuesto por el autor se plantea de dos formas diferentes: El primero utiliza una “malla adaptativa” la cual tiene como fin almacenar las mejores soluciones posibles que son obtenidas durante la ejecución del algoritmo. En el segundo algoritmo se utiliza la dominancia.

Para el manejo de soluciones de problemas Multiobjetivo se utiliza el concepto “óptimo de Pareto” el cual el autor L.V. Santana define como “aquel vector de variables en el cual no se pueden mejorar las soluciones del problema en una función objetivo sin empeorar cualquiera de las demás”. Además del uso de la “dominancia de Pareto” esto es, el hecho de que una solución sea mejor a otra, para lograr esta condición la solución debe ser mejor en al menos un objetivo sin ser peor en el resto de los objetivos.

La investigación del autor L.V. Santana brinda definiciones claras relacionadas con la evolución diferencial y su aplicación, por lo que me ayudo a entender mejor el funcionamiento de esta técnica de la computación evolutiva.

Predicción de contaminantes en el aire mediante evolución diferencial con Random Forest

Predicción de contaminantes en el aire mediante evolución diferencial con método de bosque aleatorio por Rubal (2017). En este artículo de investigación se aborda el uso de la evolución diferencial combinada con el método de random forest para la predicción tomando los datos recabados de la ciudad de Delhi y Patna. Este último método es un algoritmo de clasificación que consta de muchos árboles de decisiones. Utiliza el ensacado y la aleatoriedad de características al construir cada árbol individual para intentar crear un bosque de árboles no correlacionado cuya predicción por parte del comité es más precisa que la de cualquier árbol individual.

El resultado de esta técnica híbrida propuesta por Rubal es comparada con las predicciones generadas por el Bayesian network and multi-label classifier y la técnica heterogénea de evolución diferencial, en los ámbitos de precisión, área bajo la curva, índice de acierto y correlación. Obteniendo en estas comparaciones mejores resultados para la predicción de las variables contaminantes que los otros métodos utilizados.

1.3.2 Diferencia de los trabajos relacionados con respecto a la propuesta de la tesis

Tras la búsqueda de investigaciones relacionadas con la propuesta, no se encontró alguno que fuera desarrollado mediante algoritmos evolutivos bajo la técnica diferencial haciendo un análisis con partículas PM10, el encontrado “Predicción de contaminantes en el aire mediante evolución diferencial de Rubal (2017) hace un análisis general de los niveles de contaminantes en el aire, además de abordar una técnica nueva combinando dos modelos de programación, como lo es “Random Forest” y “Evolución diferencial”.

Otra diferencia encontrada con el trabajo Modelado de partículas PM10 Y PM2.5 mediante redes neuronales artificiales de A.A. Espinoza (2017) es el uso de redes neuronales como medio de predicción, en lugar del método de evolución diferencial aplicado durante la realización de esta tesis.

Finalmente, en el trabajo “Un Algoritmo Basado en Evolución Diferencial para Resolver Problemas Multiobjetivo” de L.V Santana (2004), se plantea una aplicación generalizada del método, centrándose en la optimización de soluciones a problemas Multiobjetivo, por el contrario del trabajo de esta tesis, la cual tiene su aplicación centrada en la predicción de niveles de contaminantes.

1.4 Hipótesis

La concentración mínima y máxima de un contaminante criterio puede ser obtenida de una base de datos histórica mediante un algoritmo evolutivo, evaluando un periodo de tiempo determinado, encontrando una solución buena en un tiempo de cómputo razonable.

1.5 Objetivos

1.5.1 General

Implementar un algoritmo evolutivo para la búsqueda óptima de concentración del contaminante criterio partículas suspendidas PM10.

1.5.2 Específicos

- Representar la solución mediante un algoritmo evolutivo.
- Identificar la función de calidad
- Implementar el algoritmo en una herramienta de programación
- Probar el algoritmo mediante experimentos.
- Evaluar sus resultados

1.6 Justificación del proyecto

Actualmente las opciones en cuanto al monitoreo de contaminantes en el Área Metropolitana de Monterrey están un tanto limitadas, disponen de poca información o presentan fallas en algunos sectores.

Se realizó la investigación correspondiente a los algoritmos evolutivos con el fin de poder diseñar un algoritmo que contribuya a la predicción del contaminante PM10, así como realizar un análisis de los datos históricos recabados mediante sensores. El uso de técnicas heurísticas como los algoritmos evolutivos permite realizar un análisis de un conjunto de datos en un tiempo computacional razonable y dar una aproximación a soluciones óptimas.

Se propone la implementación de la evolución diferencial para el análisis de los datos recabados mediante el monitoreo del contaminante PM10. En conjunto con una aplicación móvil o dashboard para la difusión de la información.

Esta investigación permite medir el desempeño de un algoritmo de evolución diferencial en el análisis de información recabada por los sensores (contaminantes criterio, temperatura, humedad, etc.).

Este proyecto soluciona la problemática del análisis de la información recolectada mediante diversos sensores para contaminantes criterio y otras variables como lo es la temperatura. Sin el debido análisis de los datos recabados no es posible mostrar los resultados.

El fin de este algoritmo será brindar los resultados obtenidos tras el procesamiento de la información a los habitantes del área.

1.7 Impacto o beneficio en la solución a un problema relacionado con el sector productivo o la generación de conocimiento científico o tecnológico

Este proyecto busca vincularse con la Secretaría de Desarrollo Sustentable del Estado de Nuevo León a través del Sistema Integral de Monitoreo Ambiental (SIMA) quien tienen la finalidad de contar con información continua y fidedigna de los niveles de contaminación ambiental en el Área Metropolitana de Monterrey.

Contribuye a la formación como Ingeniero en Sistemas Computacionales dentro del área de Inteligencia Artificial, ya que el desarrollo de este proyecto contribuyo enormemente al reforzamiento de los conocimientos aprendidos en materias como Programación Estructurada, Investigación de Operaciones, Administración de Bases de Datos e Ingeniería de Software, Inteligencia Artificial, Lenguajes y Autómatas.

1.8 Lugares en donde se desarrolla el proyecto

El proyecto tuvo lugar a desarrollarse en el Instituto Tecnológico de Nuevo León en conjunto con el Instituto Tecnológico de Saltillo. Además, se tomaron en cuenta los datos recolectados de la estación INECC “La Pastora” con dirección Calle: AV. ELOY CAVAZOS Y PABLO LIVAS (Interior Parque Zoológico No. Exterior: S/N) para el entrenamiento del algoritmo evolutivo.

1.9 Infraestructura

Para la realización de este proyecto de investigación se contó con acceso a la biblioteca digital del Instituto Tecnológico de Nuevo León (<https://itnl.bibliotecasdigitales.com/?category=3>) para consulta de información sobre el tema. Bases de Datos para consulta de artículos de investigación del Consorcio Nacional de Recursos de Información Científica y Tecnológica, CONRICYT. (<https://www.conricyt.mx/>). Además de la página de comunidad científica de ResearchGate (<https://www.researchgate.net>) para consultar artículos relacionados fuera del territorio nacional. Debido a la pandemia por COVID-19 no pudo hacerse uso de los laboratorios de Centro de computo, por lo que su realización de manera virtual.

CAPITULO II- MARCO TEÓRICO

2.1 Principios de optimización

El autor Brook Taylor(1685-1731) nos dice lo siguiente acerca de la optimización: “el propósito de la optimización es encontrar o identificar la mejor solución posible entre todas las soluciones potenciales, para un problema dado, en términos de uno o varios criterios de efectividad y desempeño”.

2.1.1 Técnicas de optimización

Técnicas clásicas de optimización:

De acuerdo al tipo de problema estos pueden clasificarse de la siguiente manera dependiendo del grado de incertidumbre:

Modelos Deterministas: en este todos los datos son conocidos.

Modelos Probabilísticos: los datos son considerados inciertos, son dados por distribución de probabilidad.

Modelos Híbridos: combinación de los modelos anteriores.

Los problemas se pueden clasificar en modelos de optimización cuando el objetivo sea maximizar o minimizar una cantidad, estando sujeta a limitaciones que restringen las decisiones y modelos de predicción.

Estos modelos se clasifican en:

Secuenciación: Como decidir el orden a procesar en un tiempo mínimo.

Localización: Asignar recursos limitados de forma eficiente o problemas de transporte a diversos destinos.

Rutas: Encontrar la ruta óptima del origen al destino.

Búsqueda de información: Búsqueda de información para tomar una decisión.

Reemplazamiento: Trata de predecir el tiempo de reemplazo por deterioro, vejez, trata de decidir el tiempo adecuado para el reemplazo de equipo.

Inventario: determinar la cantidad ideal de productos que deben tenerse disponibles

Colas o líneas de espera: Estudia el número de clientes que solicita un servicio en función del tiempo de llegada y cuanto espera para ser atendido.

Teoría de juegos: Cuando dos o más sujetos compiten por un recurso.

2.1.2 Programación Lineal

Formulación matemática:

En la programación lineal los problemas se basan en la optimización, ya sea minimización o maximización de una función lineal conocida como función objetivo, una serie de restricciones lineales. Estos problemas se pueden representar como:

$$\max z = cx \quad (1)$$

$$\text{s.a } Ax = b \quad (2)$$

$$x \geq 0$$

donde cx es la función objetivo a maximizar (o minimizar), $x \in \mathbb{R}^{+n}$ representa el vector de variables a determinar, $c \in \mathbb{R}^n$ es el vector de costos asociado a las variables, $A \in \mathcal{M}_{m \times n}$ es la matriz de coeficientes y $b \in \mathbb{R}^{+m}$ el vector de términos independientes (o rhs) relativos a las restricciones.

Método Simplex

Es un algoritmo de programación lineal el cual consiste en un proceso algebraico que en cada iteración calcula una solución factible. Cuenta con un criterio de parada que revisa que tan óptima es la solución.

Este método se basa en 3 propiedades de las soluciones factibles punto-extremo

Si hay exactamente una única solución óptima, debe ser una solución factible de punto-extremo, si hay dos o más debe ser de punto extremo adyacente.

Hay un número finito de soluciones

Si una solución factible es mejor que todas las adyacentes entonces es la solución óptima.

2.1.3 Programación no lineal

Esta técnica surge a partir de las restricciones de la programación lineal las limitaciones de la hipótesis de linealidad y la dificultad de definir una única función objetivo.

Formulación matemática:

$$\max f(x) \quad (3)$$

$$\text{s.a } g_i(x) \leq b_i, \forall i = 1, 2, \dots, m \quad (4)$$

$$x \geq 0$$

donde $f(x)$ y $g_i(x)$ son funciones dadas de n variables de decisión

Tipos de problemas de Programación no lineal:

- Optimización no restringida: es un problema sin restricciones ($\max f(x)$).
- Optimización restringida linealmente: se da si todas las funciones de restricciones son lineales pero la función objetivo no es lineal.
- Programación cuadrática: problema restringido linealmente con función objetivo cuadrática
- Programación convexa: abarca una amplia clase de problemas, entre los cuales, como casos especiales, se puede mencionar todos los tipos anteriores cuando $f(x)$ es una función cóncava que debe maximizarse.

Componentes de un algoritmo bio-inspirado

Los algoritmos bio-inspirados se basan en emplear analogías de sistemas naturales o sociales para la resolución de problemas. Estos simulan el comportamiento de dichos sistemas para el diseño de métodos heurísticos no deterministas los cuales son aplicables en áreas como la búsqueda y aprendizaje.

Características de un algoritmo bio-inspirado:

- No determinísticos
- Adaptables
- Pueden ser multiagentes

2.2 Algoritmos evolutivos y algoritmos genéticos

Computación evolutiva:

La computación evolutiva se define como un área de las ciencias de la computación que engloba las técnicas que simulan la selección natural, inspirados en la teoría de evolución de Charles Darwin, según esta los organismos que mejor se adaptan a un ambiente tienen una mayor probabilidad de supervivencia. Esta rama de la computación es un enfoque alternativo mediante el cual se pueden resolver problemas complejos de búsqueda y aprendizaje a través de modelos computacionales de procesos evolutivos.

2.2.1 Definiciones y características de algoritmos genéticos y evolutivos.

Algoritmos genéticos:

Según el autor D. Goldberg en su libro “Genetic algorithms in search, optimization, and machine learning” (1953) define un algoritmo genético como un algoritmo de búsqueda basado en las mecánicas de la selección natural y la genética, combinando la supervivencia del más apto junto con estructuras de datos con un estructurado pero aleatorio intercambio de información. De esta manera se logra formar un algoritmo de búsqueda con algo de instinto humano en la búsqueda de cada generación, creando un nuevo conjunto de criaturas artificiales usando bits y piezas del más óptimo, colocando una nueva parte ocasionalmente para tener una buena medida. Mientras es aleatorizado, los algoritmos genéticos no son una simple ejecución aleatoria, ellos toman ventaja de la información histórica para especular en nuevos puntos de búsqueda con un rendimiento esperado.

Los algoritmos genéticos fueron desarrollados por John Holland, sus colegas y estudiantes en la universidad de Michigan, teniendo 2 objetivos principales:

- 1) La abstracción y rigurosa explicación de adaptar procesos de sistemas naturales.
- 2) Para diseñar sistemas artificiales de software capaces de retener los mecanismos importantes de sistemas naturales.

¿En qué se diferencia un algoritmo genético de algún método tradicional?

- Trabajan con un código del conjunto de parámetros, no los parámetros en sí.
- Buscan en puntos de población, no en puntos unitarios.
- Usan una función objetivo, no derivadas o algún conocimiento auxiliar.
- Usan reglas de transición probabilística no reglas deterministas.

Funcionamiento de un algoritmo genético simple:

La mecánica de un algoritmo genético es bastante simple, consistiendo únicamente en copiar Strings e ir intercambiando parcialmente Strings. La simplicidad y el poder de este efecto son el principal atractivo de estos algoritmos.

Un algoritmo genético simple que da buenos resultados en problemas prácticos está compuesto por dos operadores:

- Reproducción: Es el proceso en el que el String es copiado dependiendo de la función objetivo f (función óptima), este operador es versión artificial de la selección natural. La forma más sencilla de implementarlo es haciendo una ruleta de rueda, donde cada espacio correspondiente a un String tiene distinto tamaño dependiendo de su valor de optimización como se muestra en la imagen.

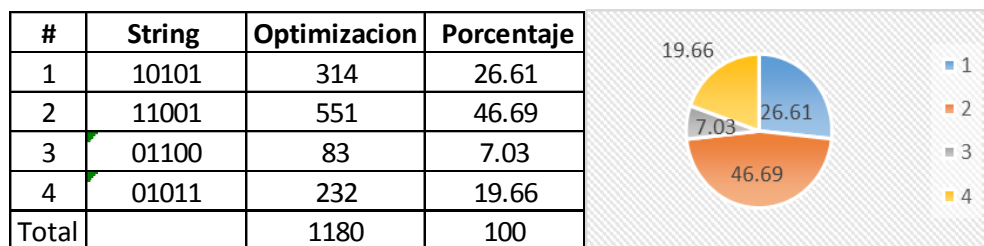


Figura 1. Representación de porcentaje de Optimización.

De esta manera, cada que se requiere una nueva generación, simplemente se gira la ruleta y esta produce el nuevo descendiente, teniendo los Strings más óptimos una mayor probabilidad de reproducción.

- Mezcla y mutación: Después de la reproducción, este proceso es una mezcla que procede en dos pasos:

- Los nuevos descendientes son apareados aleatoriamente.
- Cada par de Strings se somete a una mezcla de la siguiente manera:
 - En una posición k a lo largo del String es seleccionada uniformemente aleatoria entre 1 y el tamaño del String menos 1 $[1, L-1]$ dos nuevos Strings son creados intercambiando todos los caracteres entre las posiciones $k+1$ y L inclusive. Por ejemplo, con $k=4$:
 $A = 1101|01 \rightarrow 110110$
 - La mezcla resultante produce un nuevo String que es el primo ('), esos Strings son parte de la nueva generación.
 $A' = 110110$

2.3 Algoritmos Evolutivos

Este tipo de algoritmos es normalmente utilizado en la búsqueda de soluciones no deterministas y en la optimización. Estos algoritmos se basan en la teoría de evolución, siguiendo leyes como la selección, cruza y mutación.

Este tipo de algoritmos se basan en la generación de poblaciones de solución las cuales serán sometidas a un proceso de supervivencia con base a la función óptima. Por cada generación que avanza genera mejores soluciones al problema.

Estructura básica de un algoritmo evolutivo:

```

Begin
Inicializar al azar población con soluciones candidatas
Evaluar cada solución
Repetir hasta (condición finalización == verdadero)
  Do
    1. Seleccionar padres.
    2. Recombinar pares de padres.
    3. Mutar los hijos resultantes.
    4. Evaluar los nuevos individuos.
    5. Seleccionar individuos para la generación próxima.
  End
End

```

2.3.1 Introducción al tema e historia

Las bases de lo que se conoce actualmente como algoritmos evolutivos se remontan a la década de los sesenta y los setenta, entre estas décadas fueron propuestos 3 enfoques de la computación evolutiva: Programación evolutiva, estrategias evolutivas y algoritmos genéticos. Cada uno de estos se diferencian del otro en aspectos como los operadores o la forma de representar soluciones.

S.Wright (1932). “The roles of mutation, inbreeding, crossbreeding and selection in evolution” y W. Cannon (1932) “The wisdom of the body”. fueron los primeros en tomar la evolución natural como un proceso de aprendizaje, en dicho proceso la información correspondiente a la genética de las especies cambia por prueba y error. Wright también introduce el concepto de “aptitud” como una forma de representar los valores de aptitud para cada genotipo. Además, concluye el hecho de que la selección y mutación deben encontrarse en un balance para lograr el éxito en los procesos evolutivos.

Entre los sesenta y los setenta cuando las computadoras ya eran más accesibles se realizaron avances relevantes en el ámbito de la programación evolutiva. J. Crosby en su libro “Computers in the study of evolution (1967)” identifico tres esquemas:

- Métodos basados en el estudio de formulaciones matemáticas que emergen en el análisis determinista de la dinámica poblacional.
- Enfoques que simulan la población, pero no representan explícitamente una población.
- Esquemas que simulan la población, pero mantienen una representación explícita de la población.

En el año 1957 G. Box propuso la técnica “Operación evolutiva” o EVOP, utilizada en la optimización de procesos en industria química, contaba con la implementación de una solución padre mediante la cual se generaban más soluciones (los hijos) a partir de la modificación de parámetros del padre. Este puede considerarse como uno de los primeros algoritmos evolutivos.

2.3.2 Representación de la solución

Este tipo de algoritmos representa la población como un conjunto de posibles soluciones y estas se representan de distinta manera dependiendo de la representación utilizada:

- Cadenas binarias. → Algoritmos genéticos.
- Vector de números reales. → Estrategias Evolutivas.
- Autómatas Finitos. → Programación evolutiva.
- Árboles LISP. → Programación Genética

2.3.3 Función de aptitud

También conocida como función de evaluación o Fitness, es aquella función que representa los requisitos a los cuales deberán de adaptarse las posibles soluciones. Utilizando esta función se mide el desempeño de las soluciones.

2.3.4 Población de soluciones

Es la estructura que contiene las representaciones de las posibles soluciones generalmente es de un tamaño fijo. Los operadores de selección normalmente consideran a toda la población, esto es, las chances reproductivas son relativas a la población actual.

2.3.5 Mecanismos de selección de padres

Es la probabilidad de que una posible solución genere descendencia. Esta depende del fitness de cada individuo. Por lo regular la selección suele ser probabilística teniendo que los individuos con mejor fitness son más propensos a ser padres, sin embargo, esto no es garantizado, aun el peor padre puede llegar a tener descendencia.

2.3.6 Cruza y mutación

Es el proceso mediante el cual se generan nuevas soluciones, estos operadores dependen del tipo de representación.

Cruza:

Es el proceso que combina la información de dos de los padres, en este proceso la mayoría de los hijos son peores o en el mejor de los casos iguales a los padres.

Aunque algunos podrían llegar a ser mejores. Este principio ha sido utilizado desde el principio de la civilización (agricultura/ganadería) para mejorar las cosechas y el ganado.

Mutación:

Este proceso generalmente actúa en un genotipo y devuelve otro, la aleatoriedad es esencial y la distingue de otros operadores. La mutación debe ser ciega si se espera tener una probabilidad mayor a cero de llegar al óptimo global. Es el único operador utilizado en la Programación Evolutiva.

2.3.7 Mecanismo de reemplazo

En la mayoría de algoritmos evolutivos se utiliza una población de tamaño fijo, por lo que es necesario una forma de pasar de padres + hijos a nuevos padres. Esto es normalmente determinístico.

Normalmente se lleva de 3 formas:

Basado en fitness: Se evalúan los padres y los hijos seleccionando solamente los mejores.

Basado en edad: Se crea la misma cantidad de hijos que padres y se eliminan los padres.

Combinación de ambos, conocida como elitismo.

2.3.8 Inicialización y Terminación

Generalmente la inicialización de la población es al azar, buscando una buena mezcla y provisión de valores.

La terminación se basa en si se ha cumplido alguna de las condiciones:

- Se alcanzó el fitness esperado
- Se alcanzó el número máximo de generaciones de la función
- Se alcanzó un mínimo de diversidad
- No se mejoró el fitness en las últimas generaciones.

2.4 Herramientas para programar algoritmos evolutivos

2.4.1 EA Visualizer

Es un programa escrito en java cuyo propósito general es ser una herramienta de visualización general para algoritmos evolutivos (EA). El código de este programa ha sido escrito de manera que puede ignorar la sobrecarga al transferir los datos de su EA a una visualización del mismo, actualizarlo en cada generación e incluso dejar espacio para la interacción con el EA.

Esta herramienta cuenta con una estructura general para el EA, dividiendo el proceso evolutivo en pequeños pasos recopilados. Lo único necesario para codificar un nuevo EA es codificar código Java que funcione con algunas restricciones que provienen del EAVisualizer en la entrada y salida de los operadores del algoritmo evolutivo. Si es necesario, puede, por ejemplo, especificar qué operadores están permitidos con qué genotipos. Las combinaciones y, por lo tanto, la creación real de EA se realizan cuando el Visualizador de EA se está ejecutando y luego puede comenzar la evolución.

De manera similar, es posible definir vistas en un EA. Para ser más precisos, una vista recibe del EAVisualizer toda la información necesaria de la generación anterior y luego le pide a la vista que se dibuje. El sistema ya realiza el diseño y la ubicación de la vista y todo lo que necesita hacer es escribir código Java para lo que sea que desee ver, ya sea una descripción general de su función de aptitud o información estadística específica sobre su operador de recombinación.

2.4.2 ESCaPaDE

Evolution Strategies capable of adaptive evolution o Estrategias de evolución capaces de evolución adaptativa es un software que proporciona un entorno sofisticado para Estrategias evolutivas. ESCAPADE se basa en KORR, la implementación de Schwefel de un (μ, λ) - estrategia evolutiva. El sistema proporciona un conjunto elaborado de herramientas de seguimiento para recopilar datos de una ejecución de optimización de KORR (es una simulación de evolución final).

Los módulos principales son:

- Configuración de parámetros.
- Control de tiempo de ejecución.
- KORR.
- Monitores de datos genéricos.
- Datos personalizados.
- Monitores y soporte de monitoreo.

Durante una ejecución de optimización, los módulos de supervisión son invocados por el algoritmo principal (KORR o alguna otra implementación de ES o GA) para realizar el registro de cantidades internas. El sistema no está equipado con ningún tipo de interfaz gráfica. Todos los parámetros para la simulación se pasan como opciones de línea de comando. En la salida, cada monitor de datos escribe sus datos en archivos de registro separados.

2.4.3 GAGA

Genetic Algorithms for General Application o Algoritmos genéticos para aplicaciones generales. Fue programado por Hillary Adams de la universidad de York en Pascal. El programa fue posteriormente modificado por Ian Poole. y traducido al lenguaje C por Jon Crowcroft en University College London.

Es un algoritmo genético de tareas independientes El usuario debe proporcionar la función de destino a optimizar (minimizado / maximizado) y algunos parámetros técnicos de GA, y espere la salida. Está adecuado para la minimización de muchas funciones de costes "difíciles".

2.5 Algoritmo Evolutivo Diferencial (DE)

Este método fue propuesto por R. Storn y K. Price en un artículo para Journal of Global optimization (1997). En dicho artículo el método es descrito como “Una estrategia básica que emplea la diferencia de dos vectores de parámetros seleccionados como una fuente de vectores aleatorias para un tercer vector parámetro”.

2.5.1 Visión General

Método de evolución diferencial

Es un método de búsqueda directa en paralelo que utiliza NP vectores de parámetros

$$X_{i,G} = 0,1,2, \dots NP - 1 \quad (5)$$

Donde:

- G es cada generación.
- NP es el número de Vectores parámetro, este número no cambia durante el proceso de minimización.

Se toma a NP como la población, la población inicial será elegida aleatoriamente si nada es conocido acerca del sistema.

Como regla debemos asumir una probabilidad de distribución uniforme para cada decisión aleatoria.

La idea crucial de un DE es un esquema para generar vectores de parámetros de prueba. Este los genera agregando un vector de diferencia entre dos poblaciones miembros a un tercer miembro. Si este vector resultante arroja un valor de función objetivo menor que un miembro de la población predeterminado, el recién generado reemplazara al vector con el que es comparado en la siguiente generación. Además, el mejor vector de parámetros se evalúa para cada generación G para poder realizar un seguimiento del proceso de minimización.

2.5.2 Elementos y comportamiento

Se han probado algunas variantes para DE, las 2 que mencionan R. Storn y K. Price en su artículo son las siguientes:

Esquema DE1

La primera variante DE trabaja de la siguiente manera:

Para cada vector $X_{i,G} = 0,1,2, \dots, NP - 1$, un vector de prueba V es generado de acuerdo con:

$$\underline{v} = \underline{x}_{r_1,G} + F \cdot (\underline{x}_{r_2,G} - \underline{x}_{r_3,G}), \quad (6)$$

con $r_1, r_2, r_3 \in [0, NP-1]$ y son mutuamente diferentes, $F > 0$.

Donde:

V es el vector de prueba

$X_{r_1,G}$ Es un vector parámetro elegido aleatoriamente.

r_1, r_2, r_3 son elegidos aleatoriamente en el intervalo $[0, NP-1]$ y son diferentes al índice que se está ejecutando i .

F es un real y constante factor que controla la amplificación de la variación diferencial.

F esta entre 0 y 2 y normalmente se toman valores entre 0.9 y 1.2

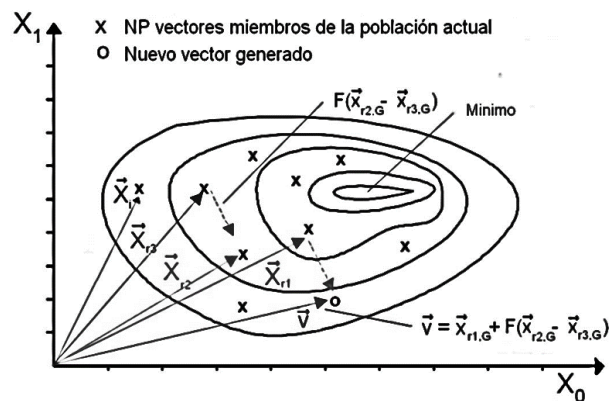


Figura 2. Traficación esquema Evolución Diferencial

Muestra de una función objetivo utilizando la generación mediante DE1

$$u_j = \begin{cases} v_j, & \text{para } j = \langle n \rangle_D, \langle n + 1 \rangle_D, \dots, \langle n + L - 1 \rangle_D; \\ (x_{i,G})_j, & \text{de otra forma.} \end{cases}$$

donde los paréntesis $\langle \rangle_D$ denotan la función módulo con módulo D.

Una cierta secuencia de elementos del vector en \vec{u} son elementos idénticos a \vec{v} , los demás elementos de \vec{u} adquieren el valor original de $\vec{x}_{i,G}$.

Esquema ED2

El esquema ED2 trabaja de igual forma que el ED1 pero genera el vector \vec{v} de acuerdo a:

$$\vec{v} = \vec{x}_{i,G} + \lambda \cdot (\vec{x}_{mejor,G} - \vec{x}_{i,G}) + F \cdot (\vec{x}_{r2,G} - \vec{x}_{r3,G}),$$

introduciendo una variable de control adicional λ . La idea es incorporar al esquema al mejor vector de la población $\vec{x}_{mejor,G}$. Esta característica puede ser muy útil para las funciones objetivo no-críticas. La figura 4.3 ilustra el proceso de generación del vector. La construcción de \vec{u} se hace usando \vec{v} y $\vec{x}_{i,G}$, aunque el proceso de decisión es idéntico al ED1.

2.5.3 Ejemplo y funcionamiento

La profesora E. Mesa en su video “Evolución Diferencial” nos brinda un ejemplo que ayuda a entender mejor el funcionamiento del algoritmo.

Dada una población inicial aleatoria uniformemente distribuida entre los límites de las variables, se tiene lo siguiente:

Matriz generada:

	[,1]	[,2]
[1,]	22.379214	66.577334
[2,]	27.578162	54.791840
[3,]	90.812179	59.702216
[4,]	89.993639	5.776780
[5,]	69.007305	35.449146
[6,]	15.057464	47.572414
[7,]	66.020246	2.327892
[8,]	2.521445	4.795300
[9,]	11.663414	93.655827
[10,]	13.656551	78.229402

Figura 3. Tabla de ejemplo de Evolución Diferencial

1-Por cada individuo NP elegir 3 individuos aleatorios diferentes a sí mismo para el individuo 1 son R1=5, R2=2, R3=9

2- Calcular V(Mutación), Usamos el factor F (en 1.2).

$$V=(R1) + F(R2-R3)$$

$$V= (69.01,35.45) + F ((27.58,54.79) - (11.66,93.66))$$

$$V= (69.01,35.45) + (1.2)(15.72,-38.87)$$

$$V= (69.01,35.45) + (18.86,-46.64)$$

$$V= (87.87,-11.19)$$

3- Escoger el nuevo individuo U entre V y x1 con un cruzamiento de 0.4

$$u_{ji,G+1} = \begin{cases} v_{ji,G+1} & \text{if } (randb(j) \leq CR) \text{ or } j = rnbr(i) \\ x_{ji,G} & \text{if } (randb(j) > CR) \text{ and } j \neq rnbr(i) \end{cases}, \\ j = 1, 2, \dots, D.$$

Se generan un numero aleatorio uniformemente distribuido, si este es menor al factor de cruzamiento (CR) se toma el valor del vector V generado, si fuese mayor se toma el valor del vector X1.

j es un aleatorio de dimensiones, este asegura que por lo menos 1 de los valores será tomado del vector V

Teniendo que:

X1=(22.38,66.58)

V= (87.87,-11.19)

Random 1= 0.45 y Random 2= 0.8

Aleatorio de dimensiones = 2

Para el primer aleatorio, 0.45 es mayor que 0.4 por lo que se toma el valor de X1 (22.38), para el segundo valor j=2 entonces aquí se tomara el valor de V (-11.19)

Resultando en U1= (22.38,-11.19)

Ya que los limites son rebasados por -11.19, se sustituye por un aleatorio uniforme que este dentro de los limites aceptados (39.87)

Por lo que U1= (22.38,39.87)

Este proceso se repetirá con x2, x3...xn.

2.6 Índice de calidad de aire

El índice de calidad del aire es una Norma Mexicana la cual se rige en todo el territorio Nacional, esta norma tiene como objetivo establecer los lineamientos para la obtención y comunicación del Índice de Calidad del Aire y Riesgos a la Salud.

Esta norma mide los niveles de los contaminantes criterio, los cuales son:

- Ozono (O₃).
- Monóxido de carbono (CO).
- Dióxido de azufre (SO₂).
- Dióxido de nitrógeno (NO₂).
- Las partículas suspendidas iguales o menores a 10 micrómetros (PM₁₀).
- Las partículas suspendidas iguales o menores a 2.5 micrómetros (PM_{2.5}).

Forma de medición de los contaminantes del aire:

Tabla 1. Normativas de contaminantes criterio

Contaminante	Método de medición y procedimientos de calibración NOM
ozono (O ₃)	NOM-036-SEMARNAT-1993
dióxido de nitrógeno (NO ₂)	NOM-037-SEMARNAT-1993
dióxido de azufre (SO ₂)	NOM-038-SEMARNAT-1993
monóxido de carbono (CO)	NOM-034-SEMARNAT-1993

Unidades de medida y Cifras Significativas

Tabla 2. Redondeo de cifras significativas para contaminantes criterio.

Contaminante	Unidad de medida	Cifras decimales significativas
PM ₁₀	µg/m ³	0
PM _{2.5}	µg/m ³	0
ozono (O ₃)	ppm	3
dióxido de nitrógeno (NO ₂)	ppm	3
dióxido de azufre (SO ₂)	ppm	3
monóxido de carbono (CO)	ppm	2

Concentraciones base para el cálculo del INDICE AIRE y SALUD

Tabla 3. Concentraciones base para contaminantes criterio.

Contaminante	Concentración base
PM ₁₀	Concentración promedio móvil ponderado de 12 horas*
PM _{2.5}	
ozono (O ₃)	Concentración promedio móvil de 8 horas
monóxido de carbono (CO)	
dióxido de nitrógeno (NO ₂)	Concentración promedio horaria
ozono (O ₃)	
dióxido de azufre (SO ₂)	Concentración promedio móvil de 24 horas (como aproximación al promedio de 24 horas)

Calculo de Concentración promedio Móvil ponderado de 12 horas:

Formula:

$$\bar{C} = \frac{\sum_{i=1}^N C_i W^{i-1}}{\sum_{i=1}^N W^{i-1}}$$

Donde:

$$W = \begin{cases} w & \text{si } w > 0.5 \\ 0.5 & \text{si } w \leq 0.5 \end{cases} \text{ y } w = 1 - \frac{C_{max} - C_{min}}{C_{max}}$$

$$\bar{C} = \frac{\sum_{i=1}^{12} (C_i W^{i-1})}{\sum_{i=1}^{12} (W^{i-1})}$$

\bar{C} = Concentración promedio móvil ponderada.

$N = 12$

Σ = Sumatoria de datos.

C_i = Concentración promedio horaria de la hora i .

i = hora consecutiva de medición (la hora más reciente de medición es la hora 1 y la primera hora de medición en el conjunto de datos considerados en el cálculo sería la hora 12).

W = Factor de ponderación.

w = Valor del peso.

C_{max} = Concentración promedio horaria máxima en el periodo de 12 horas.

C_{min} = Concentración promedio horaria mínima en el periodo de 12 horas.

Nota:

Para aplicar esta metodología de cálculo es necesario que se dé cumplimiento a las siguientes dos condiciones:

- a) Contar con datos para al menos dos de las tres horas más recientes de medición. Si esta condición no se cumple no se debe efectuar el cálculo del subíndice correspondiente para esa hora.

El valor de i (hora consecutiva de medición) debe mantenerse aún en situaciones en las que haya horas en las que no se cuente con concentraciones medidas.

Índices AIRE y SALUD para cada nivel de los contaminantes criterio.

Tabla 4. Niveles del Índice Aire y Salud.

Índice AIRE y SALUD	Nivel de riesgo asociado	Intervalo de PM ₁₀ promedio móvil ponderado de 12 horas* (µg/m³)	Intervalo de PM _{2.5} promedio móvil ponderado de 12 horas* (µg/m³)	Intervalo de O ₃ promedio de una hora (ppm)	Intervalo de O ₃ promedio móvil de ocho horas (ppm)	Intervalo de NO ₂ promedio de una hora (ppm)	Intervalo de SO ₂ promedio móvil de 24 horas (ppm)	Intervalo de CO promedio móvil de ocho horas (ppm)
Buena	Bajo	≤ 50	≤ 25	≤ 0.051	≤ 0.051	≤ 0.107	≤ 0.008	≤ 8.75
Aceptable	Moderado	>50 y ≤ 75	>25 y ≤ 45	>0.051 y ≤ 0.095	>0.051 y ≤ 0.070	>0.107 y ≤ 0.210	>0.008 y ≤ 0.110	>8.75 y ≤ 11.00
Mala	Alto	>75 y ≤155	>45 y ≤79	>0.095 y ≤0.135	>0.070 y ≤0.092	>0.210 y ≤0.230	>0.110 y ≤0.165	>11.00 y ≤13.30
Muy Mala	Muy Alto	>155 y ≤235	>79 y ≤147	>0.135 y ≤0.175	>0.092 y ≤0.114	>0.230 y ≤0.250	>0.165 y ≤0.220	>13.30 y ≤15.50
Extremadamente Mala	Extremadamente Alto	> 236	> 147	> 0.175	> 0.114	> 0.250	> 0.220	> 15.50

Fórmulas de Color

Tabla 5. Fórmulas de color para el índice aire y salud.

Color	R	G	B	C	M	Y	K
Verde	0	228	0	40	0	100	0
Amarillo	255	255	0	0	0	100	0
Naranja	255	126	0	0	51	100	0
Rojo	255	0	0	0	100	100	0
Morado	143	63	151	51	89	0	0

CAPITULO III. METODOLOGIA DE SOLUCIÓN

Para el desarrollo del proyecto se siguió un modelo basado en el desarrollo de algoritmo evolutivos y sus componentes: representación de la solución, función de calidad, población de soluciones, mecanismo de selección, operadores de variación, mecanismos de reemplazo, El algoritmo evolutivo diferencial está basado en la propuesta de R. Storn, K. Price en su reporte técnico “Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces” (1997).

3.1 Análisis del problema

Abordando la problemática establecida en el punto “descripción del problema”, se plantea la siguiente situación: Dado un conjunto de sensores meteorológicos y de contaminantes criterio, se plantea la elaboración de un algoritmo capaz de dar estimaciones a futuro en un intervalo de tiempo determinado.

3.2 Diseño de la solución

3.2.1 Representación de la solución.

En el método de Evolución Diferencial se maneja una población como un conjunto de vectores, donde cada vector es un “individuo” y cada individuo es una posible solución al problema. Por esto la solución se representa mediante un arreglo o ArrayList de objetos de la clase Individuo, esta clase contiene en si un arreglo correspondiente a los parámetros que definen los valores del individuo en sus distintas características, para el problema de la mochila, estos representaban si se llevaba o no cierto objeto, en el caso de la predicción de los contaminantes PM10 es el nivel del contaminante para dicho individuo.

3.2.2 Identificar función de calidad.

En los algoritmos evolutivos, la función de calidad es el medio sobre el cual se puede evaluar el desempeño de las posibles soluciones al problema, buscando normalmente minimizar o maximizar dicha función.

En el caso del problema de los contaminantes, la función de calidad está dada por la comparación entre los valores del vector Individuo y valores reales registrados

por la estación la pastora, los cuales se explicarán en el punto “mecanismo de selección de padres”. Este proceso de evaluación busca minimizar el error absoluto.

Comparación de las estimaciones y valores reales obtenidos:

Para la evaluación del desempeño del algoritmo, se optó por calcular el error absoluto obtenido, teniendo el valor real como la medición obtenida para cierta hora en específico, y el valor aproximado como la estimación de salida que da el algoritmo. La fórmula a utilizar es la siguiente:

$$e = \frac{|V_{Real} - V_{Aprox}|}{V_{Real}} \times 100\% \quad (7)$$

Donde:

E: es el error absoluto

Vreal: Es el valor registrado de la medición del contaminante PM10 en cierta hora específica.

Vaprox: Es el valor de salida del algoritmo evolutivo en cierta hora específica.

El valor absoluto de la resta entre el valor real y el valor aproximado será dividido entre el valor real y multiplicado por 100%, esto para obtener el porcentaje de error absoluto. Esta fórmula es de utilidad para evaluar la aproximación de los resultados del algoritmo con los valores reales que se esperarían obtener como salida.

El error absoluto mostrado previamente fue utilizado en 2 procesos del algoritmo, para la medición del desempeño de todos los individuos, en el cual el Valor real era el promedio de las 30 mediciones seleccionadas, y en la evaluación del desempeño del algoritmo, en este último siendo el valor real los datos capturados para una hora específica de manera oficial.

3.2.3 Población de soluciones.

Una población de soluciones es la estructura que contiene las representaciones de las posibles soluciones generalmente es de un tamaño fijo. Para la implementación del algoritmo se tomará una población de 30 individuos.

3.2.4 Mecanismo de selección de padres (selección).

Obtención de los datos:

El algoritmo tomara los datos recabados por la estación la pastora correspondiente al intervalo del 01/01/2020 al 30/07/2020 como referencia para la generación de predicciones. Dando un total de 4250 muestras del contaminante PM10. Cada una corresponde a la medición de una hora específica del día en el intervalo mencionado. Teniendo aproximadamente 177 muestras para cada hora de las 0:00-23:00.

Selección de los datos a utilizar:

Se seleccionarán al azar siguiendo una distribución uniforme 60 de las 177 muestras mencionadas previamente, correspondientes a los niveles del contaminante de partículas suspendidas PM10, las muestras seleccionadas fueron divididas en 30 que serán la población inicial y 30 que serán utilizadas para evaluar a las posibles soluciones.

Este proceso de selección se aplicará para las 24 horas, por lo que en total serán seleccionadas 1440 muestras, 720 usado como población inicial y 720 como evaluación de posibles soluciones.

3.2.5 Operadores de variación (cruza y/o mutación).

Siguiendo lo establecido con el método de evolución diferencial, el método de variación se basa en la generación de un vector de prueba V

Para cada vector $X_{i,G} = 0,1,2, \dots NP - 1$, un vector de prueba V es generado de acuerdo con:

$$\underline{v} = \underline{x}_{r_1,G} + F \cdot (\underline{x}_{r_2,G} - \underline{x}_{r_3,G}),$$

con $r_1, r_2, r_3 \in [0, NP-1]$ y son mutuamente diferentes, $F > 0$.

Donde:

- V es el vector de prueba
- $X_{r_1,G}$ Es un vector parámetro elegido aleatoriamente.
- R_1, r_2, r_3 son elegidos aleatoriamente en el intervalo $[0, NP-1]$ y son diferentes al índice que se está ejecutando i .
- F es un real y constante factor que controla la amplificación de la variación diferencial.

Se generará un nuevo vector U el cual es la cruce entre el vector XI y el vector V generado previamente

$$u_{ji,G+1} = \begin{cases} v_{ji,G+1} & \text{if } (randb(j) \leq CR) \text{ or } j = rnbr(i) \\ x_{ji,G} & \text{if } (randb(j) > CR) \text{ and } j \neq rnbr(i) \end{cases},$$

$$j = 1, 2, \dots, D.$$

Figura 4. Elección de parámetros para el vector U

Se generan un numero aleatorio uniformemente distribuido, si este es menor al factor de cruzamiento (CR) se toma el valor del vector V generado, si fuese mayor se toma el valor del vector $X1$.

j es un aleatorio de dimensiones, este asegura que por lo menos 1 de los valores será tomado del vector V . Estos vectores U serán considerados para el mecanismo de reemplazo como una posible NuevaPoblacion.

3.2.6 Mecanismo de reemplazo (selección de sobrevivientes).

Para la selección de los sobrevivientes, se eligió hacer uso de un mecanismo similar a la selección de los más aptos, haciendo ciertas variaciones explicadas a continuación:

Los individuos de la población son sometidos a un ordenamiento dependiendo de sus resultados en base a la función de calidad explicada anteriormente, se ordenarán los 30 individuos correspondientes a la población junto con los 30

individuos producidos por los operadores de variación, siendo este conjunto de 60 individuos ordenados del mejor al peor.

Se ha definido una ecuación basada en el cálculo de la suma de los números de 1 a n, teniendo probabilidad diferente para cada individuo dependiendo de su posición en el ordenamiento de los individuos, retirando la sumatoria correspondiente a los individuos que ya han sido seleccionados.

La probabilidad de que un individuo sea seleccionado está dada por:

$$P(A) = \frac{(n - p)}{\frac{(n)(n + 1)}{2} - \sum_{i=0}^x (n - p_i)}$$

Donde:

n es el número total de individuos.

p es la posición en la que se encuentra en la tabla de individuos (desde 0 a n-1).

x es la cantidad de individuos que ya han sido seleccionados antes que él.

La sumatoria representa n menos la posición del individuo seleccionado en la iteración i.

p_i es la posición del individuo seleccionado en la iteración i.

Ejemplo con 30 Individuos ordenados de la posición 0 a 29:

Individuo	(n-p)
0	30
1	29
2	28
3	27
27	3
28	2
29	1

Probabilidad del primer individuo de ser elegido en la primera iteración:

$$P(a) = \frac{30}{\frac{(30)(31)}{2} - \sum_{i=0}^1 (30 - p_i)}$$

Donde la sumatoria representa los números ya seleccionados previamente, al no haber sido elegido ninguno esta será 0.

Por lo que la probabilidad está dada por $30/465$.

Para la segunda iteración la probabilidad de que el primer individuo sea elegido dado que no fue seleccionado en las iteraciones anteriores, suponiendo que el número elegido en la primera iteración fuese el 2, la sumatoria sería $0+28$ por lo que la probabilidad es de $30 / (465-28) = 30/437$.

Teniendo 7 iteraciones, suponiendo que los números elegidos fuesen 2,3,4,15,20,21.

La sumatoria sería $0+28+27+26+15+10+9 = 115$.

Por lo que la probabilidad sería $30 / (465-115) = 30/350$.

3.3 DIAGRAMA GENERAL DEL ALGORITMO EVOLUTIVO PROPUESTO

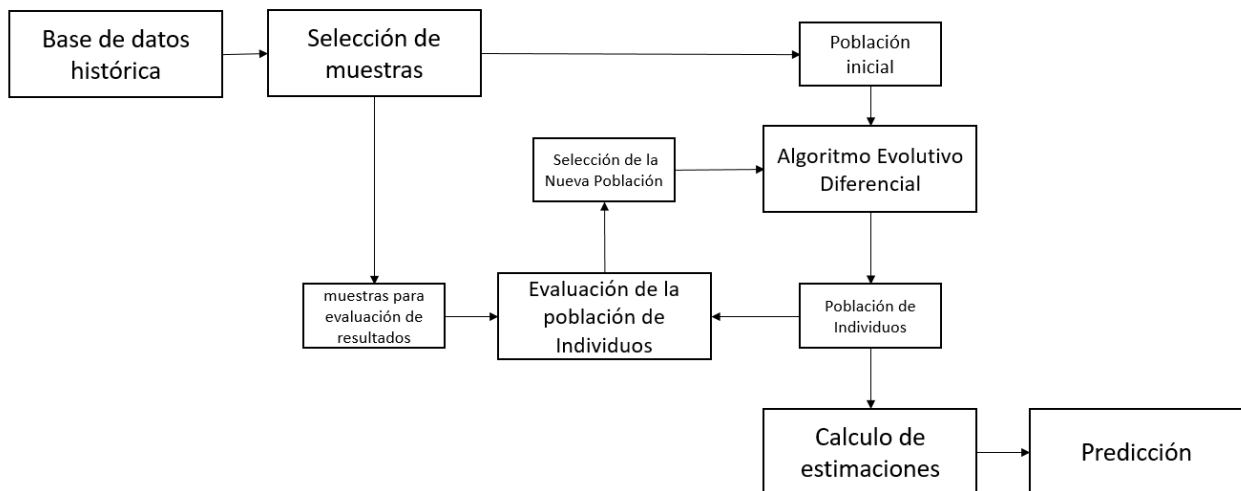


Figura 5. Diagrama general de funcionamiento del algoritmo evolutivo propuesto.

CAPITULO IV- EXPERIMENTACION Y ANÁLISIS DE RESULTADO

Con el fin de demostrar el correcto funcionamiento del algoritmo evolutivo implementado, se planteó realizar un experimento previo al correspondiente a la predicción de los niveles del contaminante PM10. Este experimento planteado fue la resolución del “Knapsack Problem” o “Problema de la mochila”.

4.1 -Experimento 1- “Knapsack Problem”.

4.1.1 Análisis del problema

El problema de la mochila o 0-1 Knapsack problem (KP) es dado un conjunto de n objetos y una mochila con:

p_j = Beneficio del objeto j

w_j = peso del objeto j

C = capacidad de la mochila

Seleccione un subconjunto de objetos sujeto a lo siguiente

Maximizar z

$$z = \sum_{j=1}^n p_j x_j$$

Sujeto a

$$\sum_{j=1}^n w_j x_j \leq c,$$

$$x_j = 0 \text{ o } 1 \quad j \in N = \{1, \dots, n\},$$

$$x_j = \begin{cases} 1 & \text{Si el objeto fue seleccionado} \\ 0 & \text{de cualquier otra forma} \end{cases}$$

KP es uno de los más importantes y uno de los más estudiados intensivamente en los problemas de programación discreta. La razón de este interés se debe básicamente a tres factores:

- Puede ser visto como un problema simple de programación lineal.
- Aparece como un subproblema en problemas más complejos.

c) Puede representar un gran número de soluciones prácticas.

4.1.2 Diseño de la solución

Para la implementación del problema de la mochila (el cual es parte experimental para comprobar el correcto funcionamiento del algoritmo e implementarlo próximamente en la predicción de contaminantes) se decidió por utilizar un algoritmo de evolución diferencial basado en la propuesta de R. Storn y K. Price.

Representación de la solución

En el método de evolución diferencial se maneja una población de n individuos, cada individuo es un vector de parámetros compuesto por números reales. Para el caso experimental del problema de la mochila se optó por aplicar un redondeo de los valores mayores a 0.5 a 1 y los valores menores a 0.5 a 0. Esto con el objetivo de cumplir con los requisitos del problema de la mochila.

Identificar la función de calidad

Para el problema de la mochila o knapsack problem 0-1 la función de calidad está dada por maximizar Z la cual se muestra en la figura 6. Esta es la sumatoria desde $j=1$ hasta n de $p_j x_j$, donde p_j es el beneficio de llevar el objeto j y x_j es un valor binario que representa si se lleva el objeto (1) o no (0).

$$z = \sum_{j=1}^n p_j x_j$$

Figura 6. Función objetivo del knapsack problem.

Restricciones del problema

La función objetivo está sujeta a la siguiente restricción: La sumatoria desde $j=1$ hasta n de $w_j x_j$ debe ser menor o igual a la capacidad de la mochila. Donde w_j es el peso de cada objeto.

$$\sum_{j=1}^n w_j x_j \leq c,$$

Figura 7. Restricciones del problema.

Población de soluciones

Una población de soluciones es la estructura que contiene las representaciones de las posibles soluciones generalmente es de un tamaño fijo. Para la implementación del algoritmo se tomará una población de 30 individuos.

Mecanismo de selección de padres (selección)

El método de evolución diferencial define que la población inicial será elegida aleatoriamente si nada es conocido acerca del sistema. Si por el contrario se tienen datos previos, la población deberá inicializarse con estos valores. Para la experimentación con el problema de la mochila se inicializará de manera aleatoria la población inicial.

Para las siguientes n generaciones se decidió reemplazar a la población de los padres con sus respectivos hijos.

Mecanismo de reemplazo (supervivientes)

Una vez generados todas las soluciones U explicadas en el punto “Operadores de variación”, estas deberán sustituir a sus padres y serán la NuevaPoblacion.

4.1.3 Implementación

Representación de la solución

En este algoritmo cada individuo es una posible solución al problema. El código correspondiente a la clase Individuo que se muestra a continuación, la cual está compuesta por un arreglo de tipo double el cual corresponde a cada uno de los parámetros necesarios de la población para el problema.

```
public class Individuo implements Comparable<Individuo> {
    Double[]Vectores;
    public Individuo(int CantVariables)
    {Vectores = new Double[CantVariables];}
    public void setVectPos(int pos, double val)
    {Vectores[pos]=val;}
    public double getVectPos(int pos)
    {return Vectores[pos];}
}
```

Inicio del algoritmo

El algoritmo inicia su ejecución con la asignación de variable rng, llamada a los métodos crearLimitesPares, generarPoblacion, inicializando la variable que lleva el control de las generaciones en 1 y creando el individuo best, el cual será mostrado al terminar la ejecución del programa.

```
public static void main(String[] args) {
    rng = new Random();
    crearLimitesPares();
    generarPoblacion();
    GEN_ACT = 1;
    best = new Individuo(CANT_VAR);
```

A partir de ahí se inicia un ciclo dowhile que se ejecutara mientras la variable GEN_ACT sea menor que el numero de iteraciones indicado. Dentro de este ciclo se realizan los operadores de cruza, mutacion y selección.

```
nuevaPoblacion = new ArrayList<>();
    //iteracion
    for (int i = 0; i < TAM_POBLACION; i++) {
        //Este metodo nos trae 3 numeros aleatorios excluyendo i
        int[] rand = Elegir3Aleat(i);
        //Llamamos ahora a calcularV este recibe los 4
individuos,
        //individuo i, r1, r2 y r3
        Individuo V;
        V = CalcularV(Poblacion.get(i), Poblacion.get(rand[0]),
Poblacion.get(rand[1]),
        Poblacion.get(rand[2]));
```

Población de soluciones

El programa genera una población de 30 individuos de los cuales la primera generación se inicializa con valores aleatorios entre los límites inferior y superior de cada parámetro (para el problema de la mochila son 0 y 1 respectivamente). como se muestra a continuación.

```
private static void crearLimitesPares() {
    limites = new double[CANT_VAR][2];
    //El primero es el limite inferior y el segundo el limite
superior
    for (int i = 0; i < CANT_VAR; i++) {
        limites[i][0] = 0;
        limites[i][1] = 1;
    }
}
private static void generarPoblacion() {
```

```

Poblacion = new ArrayList<>();
for (int i = 0; i < TAM_POBLACION; i++) {
    //Se crea un nuevo individuo
    Individuo ind = new Individuo(CANT_VAR);
    //se generan los valores aleatorios del individuo
    for (int j = 0; j < CANT_VAR; j++) {
        double val = rng.nextDouble() * limites[j][1] +
limites[j][0];
        ind.setVectPos(j, val);
    }
    Poblacion.add(ind);
}
}

```

El método generarPoblacion inicializa el ArrayList correspondiente a la población de soluciones que se maneja durante el proceso del algoritmo evolutivo. Este método requiere de las variables TAM_POBLACION que representa el tamaño de la población de soluciones y de CANT_VAR que representa el número de parámetros que contiene cada vector (en este caso el objeto Individuo) solución. La inicialización de las variables TAM_POBLACION Y CANT_VAR.

```

// Tamaño de la Poblacion Se recomiendan mas de 30
// Numero de iteraciones a realizar
// Cantidad de variables de cada individuo
static int NUM_ITER = 50, TAM_POBLACION = 30, GEN_ACT = 0;
//Lista de los individuos que habra
static ArrayList<Individuo> Poblacion, nuevaPoblacion;

```

Operadores de variación (cruza y mutación)

El método de Evolucion diferencial requiere de valores aleatorios uniformemente distribuidos para lo cual importamos la clase Random de Java.

```

//Factor de mutacion y cruzamiento (generalmente es un numero bajo)
static double F = 1.2, Cr = 0.4;
static Random rng;

```

Para crear el vector V se requiere tomar 3 números aleatorios distintos de xi que estén dentro del tamaño de la población, para realizar esto se diseñó un método Elegir3Aleat.

```

private static int[] Elegir3Aleat(int per) {
    int[] aleat = new int[3];
    boolean Fin = false;
    do {
        for (int i = 0; i < 3; i++) {
            aleat[i] = rng.nextInt(TAM_POBLACION);
            if (aleat[i] == per) {
                i--;
            }
        }
    }
}

```

```

        if (aleat[0] != aleat[1] && aleat[0] != aleat[2] && aleat[1] != aleat[2]) {
            Fin = true;
        }
    } while (Fin != true);
    return aleat;
}

```

El método mostrado devuelve un arreglo de tipo entero de tamaño 3, este arreglo representa los 3 números aleatorios, estos números aleatorios deben cumplir con 2 condiciones:

- 1) Todos los números deben ser distintos de xi(en el método se representa como la variable per).
- 2) Los 3 números deben ser diferentes entre sí.

Siguiendo estas restricciones se decidió crear un ciclo do while que se repitiera mientras los numerosa generados por la clase Random no cumplieran con dichas reglas. Una vez estas se cumplen se retorna el arreglo aleat.

Una vez se ha obtenido el arreglo aleat se llama al método CalcularV el cual siguiendo la metodología del algoritmo de evolución diferencial aplica las operaciones correspondientes a la formula.

```

//Llamamos ahora a calcularV este recibe los 4 individuos,

        //individuo i, r1, r2 y r3
        Individuo V;
        V = CalcularV(Poblacion.get(i), Poblacion.get(rand[0]),
Poblacion.get(rand[1]),
                Poblacion.get(rand[2]));
//Recibe 4 "Individuos" El primero es el de la iteracion, los otros 3 son
los elegidos aleatoriamente
    private static Individuo CalcularV(Individuo x, Individuo i,
Individuo j, Individuo k) {
        Individuo v = new Individuo(CANT_VAR);
        for (int y = 0; y < CANT_VAR; y++) {
            //Aplicacion de la formula V = R1 + F(R2-R3)
            double Valor = i.getVectPos(y) + (F * (j.getVectPos(y) -
k.getVectPos(y)));
            //Se agrega el resultado al individuo V
            v.setVectPos(y, Valor);
        }
        //Retornamos el Individuo v
        return v;
    }
}

```

Mecanismo de reemplazo (supervivientes)

Una vez se ha generado el Individuo V se procede con el mecanismo de reemplazo. Para el algoritmo de evolución diferencial esto se hace mediante el Individuo U, el cual es una mezcla entre el Individuo xi y V. Para esto llamamos al método ElegirU el cual recibe los Individuos previamente mencionados.

```
//Se genera el individuo U, quien se agregara al vector de la nueva poblacion
```

```
Individuo U;  
U = ElegirU(Poblacion.get(i), V);
```

El método ElegirU aplica una mezcla entre los 2 Individuos siguiendo los siguientes criterios:

- 1) Uno de los parámetros (j) elegido de manera aleatoria en el rango {0-TAM_POBLACION} deberá ser tomado del Individuo V.
- 2) El criterio para elegir el resto de los parámetros es mediante un numero aleatorio uniformemente distribuido (r), bajo la condición de que si este es mayor al factor de cruzamiento (Cr) se tomara del individuo xi, de no cumplir la condición se toma del Individuo V.
- 3) Si se ha elegido el parámetro del Individuo V, debe verificarse que el valor correspondiente a dicho parámetro este dentro de los limites inferior y superior de dicho parámetro. Si no llega a cumplir esta condición, deberá reemplazarse por un valor aleatorio entre los rangos aceptados.

```
private static Individuo ElegirU(Individuo xi, Individuo v) {  
    Individuo u = new Individuo(CANT_VAR);  
    int j = rng.nextInt(CANT_VAR);  
    for (int i = 0; i < CANT_VAR; i++) {  
        double r = rng.nextDouble();  
        double elegido;  
        //Verificamos que r sea mayor al Cruzamiento y que i sea  
diferente de j  
        //Recordemos que j es el valor que si o si se tomara del  
Vector v  
        if (r > Cr && i != j) {  
            elegido = xi.getVectPos(i);  
        } else {  
            //Verificamos si el numero no excede los limites  
establecidos  
            elegido = v.getVectPos(i);  
        }  
    }  
}
```

```

        if (elegido < limites[i][0] || elegido > limites[i][1]) {
            //Si excede los limites se cambia por un valor
aleatorio
            //entre los limites aceptados
            elegido = rng.nextDouble() * limites[j][1] +
limites[j][0];
        }
    }
    u.setVectPos(i, elegido);
}
return u;
}

```

Una vez realizado el procedimiento con todas las soluciones de la población se procede a evaluar cada una de las soluciones con respecto a la función objetivo (función Fitness), para fines demostrativos se muestra el mejor de cada generación y el mejor obtenido durante la ejecución del algoritmo de todas las iteraciones.

Para esto se tienen 3 variables de tipo entero FitGen que representa el valor Z alcanzado con base a la función objetivo, gCAcum que representa la capacidad de c alcanzada por dicho Individuo y gMF que representa la posición en el ArrayList NuevaPoblacion del Individuo con mayor Fitness de la generación. En la sentencia If se verifica que el Fitness de un Individuo sea mayor al mayor registrado previamente y que esta solución cumpla con la condición de no exceder el peso limite aceptado.

En caso de cumplir con ambos requisitos se igualarán los valores a las variables correspondientes.

```

//Se guarda el mejor de cada generacion
static int FitGen,gMF, gCAcum;

FitGen=0;
gCAcum=0;
gMF=0;
for (int i = 0; i < TAM_POBLACION; i++) {
    int[] Fitness = EvaluarFitness(nuevaPoblacion.get(i));
    if(Fitness[0]>FitGen && Fitness[1]<=c)
    {
        FitGen=Fitness[0];
        gCAcum=Fitness[1];
        gMF=i;
    }
}

```


El método EvaluarFitness tiene como objetivo calcular Z dado los valores del Individuo X, este al experimentarse en un problema de tipo binario (0-1) y manejando el sistema como de tipo double, aplica un redondeo a los valores mayores a 0,5 convirtiéndolos en 1 y los valores menores en 0.

En caso de que se convierta en 1 se sumara el valor del Objeto i a la variable fitness (representa Z), y el peso a la variable acum (representa c). Al finalizar se asigna los valores de fitness y acum al arreglo val para poder retornarlo.

```
public static int[] EvaluarFitness(Individuo x) {
    int[] val = new int[2];
    int fitness = 0;
    int acum = 0;
    //Aqui se evalua la funcion fitness para cada individuo
    for (int i = 0; i < CANT_VAR; i++) {
        double d = x.getVectPos(i);
        if (d > 0.5) {
            fitness += pj[i];
            acum += wj[i];
        }
    }
    val[0]=fitness;
    val[1]=acum;
    return val;
}
```

De la misma forma se evalúa cada uno de los Individuos con el mejor registrado, en caso de ser una mejor solución se le asignan los valores de este a MayorFit, cAcum, Pmf y gen, que representan el mayor Z obtenido de todas las soluciones, la capacidad de dicha solución, la posición de dicha solución en el ArrayList Población y la generación en la que se produjo dicha solución.

```
if(Fitness[0]>MayorFit && Fitness[1]<=c)
    {
        best=nuevaPoblacion.get(i);
        MayorFit=Fitness[0];
        cAcum=Fitness[1];
        pMF=i;
        gen=GEN_ACT;
    }
}
MostrarMejor(nuevaPoblacion.get(gMF),FitGen,gCACum,
gMF,GEN_ACT);
Poblacion = nuevaPoblacion;
GEN_ACT++;
```

Para fines demostrativos se imprime el mejor de cada generación al finalizar el ciclo. Por último, se incrementa en uno el contador GEN_ACT y se iguala el ArrayList Población a NuevaPoblacion.

Tras finalizar las iteraciones se muestra la mejor solución generada durante el proceso con base a la función objetivo (Fitness).

```
MostrarMejor (best, MayorFit, cAcum, pMF, gen) ;
```

4.1.4 Pruebas del algoritmo en el primer experimento

Con el fin de probar la efectividad del algoritmo en la búsqueda de la mejor solución se propusieron los siguientes ejemplos:

Ejemplo con 7 parámetros

$p_j = \{70, 20, 39, 37, 7, 5, 10\};$

$w_j = \{31, 10, 20, 19, 4, 3, 6\};$

$c = 50$

Valor Óptimo Esperado: 107

Cadena binaria de la solución esperada: 1 1 0 0 0 1 1

Ejemplo con 8 parámetros

$p_j = \{15, 100, 90, 60, 40, 15, 10, 1\};$

$w_j = \{2, 20, 20, 30, 40, 30, 60, 10\};$

$c = 102$

Valor Óptimo Esperado: 280

Cadena binaria de la solución esperada: 1 1 1 1 0 1 0 0

Ejemplo con 10 parámetros

$p_j = \{10, 20, 30, 35, 30, 28, 32, 45, 50, 25\};$

$w_j = \{3, 2, 7, 4, 1, 6, 4, 8, 8, 5\};$

$c = 35$

Valor Óptimo Esperado: 247

Cadena binaria de la solución esperada: 1 1 0 1 1 0 1 1 1 1

Ejemplo con 16 parámetros

$p_j = \{350, 400, 450, 20, 70, 8, 5, 5, 360, 398, 445, 23, 74, 8, 7, 6\};$

$w_j = \{25, 35, 45, 5, 25, 3, 2, 2, 25, 35, 45, 5, 25, 3, 2, 2\};$

$c = 208$

Valor Óptimo Esperado: 2130

Cadena binaria de la solución esperada: 0 1 1 1 0 1 1 0 1 1 1 1 1 0 1 1 1

Análisis de resultados del primer experimento

Analizando el correcto funcionamiento del algoritmo, comenzamos por revisar los Individuos de la población inicial, podemos apreciar que estos se generan de manera aleatoria correctamente.

Individuo 0 0 1 0 1 1 1 0	Individuo 11 1 1 0 0 1 0 0	Individuo 21 1 1 0 1 1 0 0
Individuo 1 0 0 1 1 0 0 0	Individuo 12 0 0 0 0 1 0 0	Individuo 22 0 1 1 0 0 0 1
Individuo 2 1 0 0 1 1 1 1	Individuo 13 1 1 1 1 1 1 0	Individuo 23 1 1 1 1 0 1 0
Individuo 3 0 1 1 0 0 0 1	Individuo 14 1 1 0 0 1 1 0	Individuo 24 1 0 0 0 1 1 1
Individuo 4 0 1 0 1 1 1 1	Individuo 15 0 1 0 0 0 1 0	Individuo 25 0 0 0 1 0 0 1
Individuo 5 1 0 1 1 1 0 1	Individuo 16 0 0 1 0 0 1 1	Individuo 26 1 0 1 0 0 0 0
Individuo 6 1 1 1 1 0 1 0	Individuo 17 1 1 1 1 1 0 1	Individuo 27 0 0 1 0 1 1 1
Individuo 7 1 1 1 0 0 0 1	Individuo 18 1 0 1 1 0 0 1	Individuo 28 1 0 0 1 0 0 1
Individuo 8 1 0 0 1 1 1 0	Individuo 19 0 1 0 0 0 0 1	Individuo 29 0 1 0 0 1 0 0
Individuo 9 1 1 0 0 1 0 0	Individuo 20 0 0 1 0 0 0 1	
Individuo 10 1 1 0 0 1 1 1		

Figura 8. Población inicial aleatoria

En la figura 9 se muestran los resultados obtenidos al ejecutar el programa para el ejemplo de 7 parámetros, como se puede apreciar, en este ejemplo la respuesta se encontró la mayoría de las veces en las primeras 5 iteraciones. Esto se debe a la cantidad de combinaciones posibles que se tienen con 7 parámetros (2^7) que serían 128 posibles, teniendo una población de 30 Individuos nos daría una probabilidad de 0.2343 de que la respuesta se encuentre desde un inicio al tener la primera población generada de manera aleatoria.

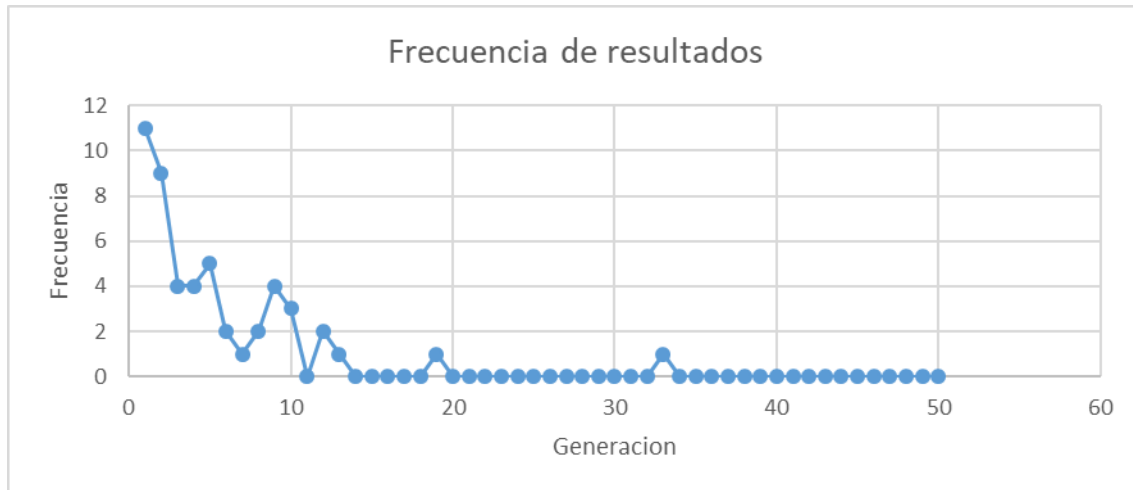


Figura 9. Resultados del experimento 1.

Para el segundo experimento, tomando en cuenta 8 parámetros puede apreciarse que se ha disminuido la cantidad de veces que encuentra la respuesta en las primeras 10 iteraciones, aunque estos siguen siendo mayoría. Para este caso la probabilidad de encontrar la respuesta era de 0.1171 en la primera iteración.

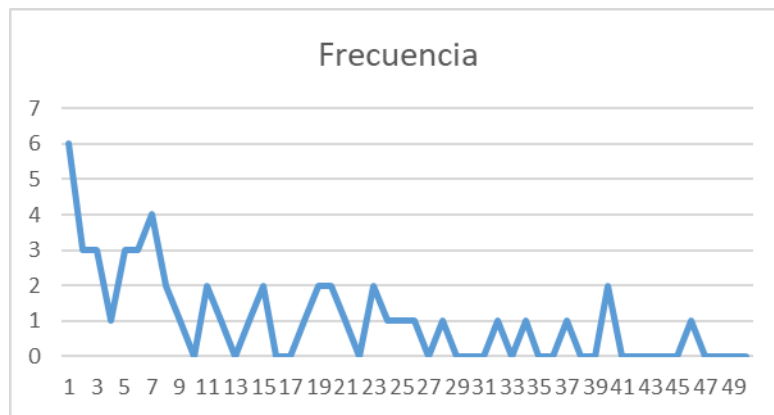


Figura 10. Resultados del Experimento 2.

Al realizar el tercer experimento podemos apreciar que ya es bastante improbable encontrar la solución esperada en la primera generación (0.029) por lo que ahora puede apreciarse correctamente la capacidad del algoritmo para encontrar óptimos tras el transcurso de las iteraciones.

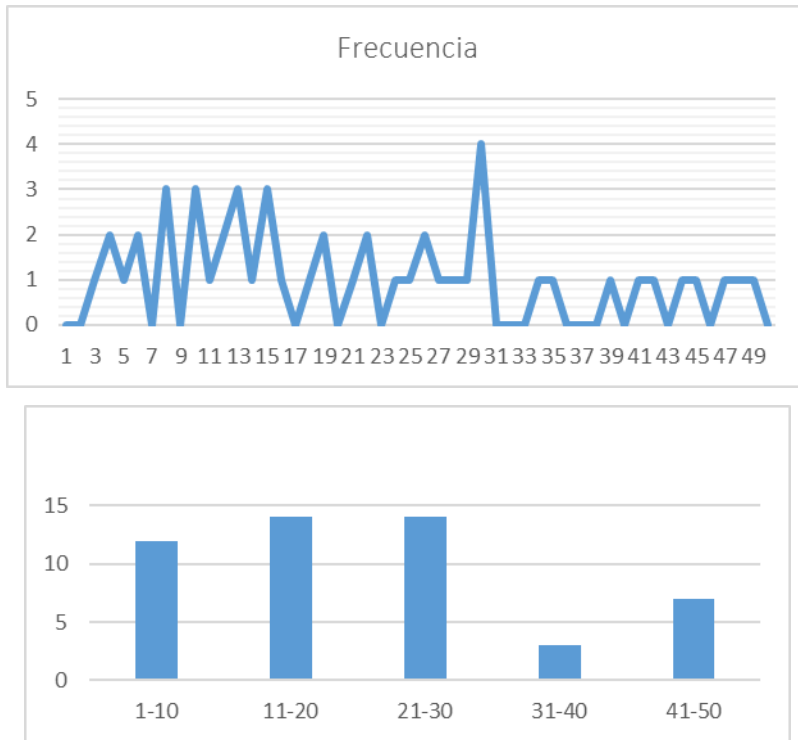


Figura 11. Resultados del Experimento 3.

Otro aspecto a considerar en el experimento 3 es el hecho de que conforme se incrementó el número de parámetros, la probabilidad de encontrar la solución óptima se redujo del 100% en los casos anteriores a un 76%. Tomando en cuenta que las técnicas meta heurísticas tienen como objetivo acercarnos a una respuesta cercana a la óptima en un tiempo computacional razonable. La media obtenida de las 50 ejecuciones fue de 245.58.



Figura 12. Frecuencia de Soluciones Optimas encontradas en el experimento 3.

En el cuarto experimento, el cual involucraba 16 parámetros, cuyas posibles combinaciones eran 65,536. Revisando la estadística de en qué generación se encontraba la mejor solución en cada una de las 50 ejecuciones del programa realizadas se muestra lo siguiente.

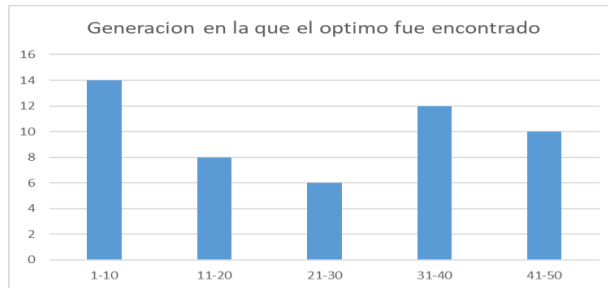


Figura 13. Frecuencia de generación en que el óptimo fue encontrado.

Como se puede apreciar en la figura 12, el óptimo ya no es encontrado con facilidad en las primeras iteraciones, esto porque la probabilidad de encontrarlo aleatoriamente es demasiado baja (0.0004), aquí es donde se muestra el verdadero potencial del algoritmo, ya que es bastante complicado llegar a la solución óptima, el método de evolución diferencial nos brinda una solución cercana a la esperada como se muestra en la figura 13, en este caso la solución óptima era 2130. Se observa una mayor variedad de posibles Z encontradas entre las mejores soluciones.

Optimo Obtenid	Frecuencia
2093	0.02
2097	0.02
2102	0.02
2105	0.02
2106	0.02
2107	0.1
2108	0.08
2109	0.02
2111	0.02
2113	0.04
2114	0.04
2115	0.04
2116	0.08
2117	0.04
2120	0.08
2121	0.06
2122	0.08
2123	0.08
2124	0.04
2125	0.04
2129	0.02
2130	0.04

Figura 14. Estadísticos obtenidos del experimento 4.

La media obtenida con estos datos fue de 2115.64, la cual es bastante cercana al óptimo esperado del problema que es 2130. Teniendo un error de 0.1436.

4.1.5 Conclusiones del primer experimento

Ante los datos obtenidos durante la realización del algoritmo evolutivo se puede concluir que:

- El programa funciona correctamente tanto en el caso que se utilice una cantidad pequeña de parámetros como si se usara una cantidad considerable.
- Si bien en la experimentación donde se manejaban 16 parámetros se llegó solamente en 2/50 ocasiones a la respuesta óptima, el resto de las soluciones estaba bastante cerca del óptimo esperado.
- Se espera que al adaptar este algoritmo a la predicción de los contaminantes en el INDICE AIRE y SALUD se obtengan resultados satisfactorios.
- En cuanto a lo que se puede corregir del algoritmo, se estima que los resultados podrían mejorar si se aplicara como criterio de supervivencia el mantener las mejores soluciones encontradas entre los padres y los hijos. Esto podrá comprobarse o descartarse durante la realización del algoritmo de predicción.

4.2 Experimento 2- Predicción de niveles del contaminante PM10

4.2.1 Procedimiento y descripción de las actividades realizadas

De acuerdo con el planteamiento de actividades de la residencia profesional y tesis, las actividades a realizar se describen a continuación:

Elaborar la representación de la solución por algoritmos evolutivos.

Una vez se cuente con la investigación previa, se optó por diseñar un algoritmo de evolución diferencial como solución, por lo que se investigó más a fondo esta técnica y elaboro una representación acorde a la problemática para su posterior implementación.

Diseño e implementación del algoritmo evolutivo

Contando con una idea clara de los algoritmos de evolución diferencial, comenzó a diseñarse la implementación del algoritmo en java Netbeans. Contemplando el diseño del algoritmo por completo, desde su fase de generación de la población, mecanismos de cruce y mutación hasta la selección de los sobrevivientes.

Prueba y correcciones del algoritmo evolutivo

Para la etapa de pruebas fue necesario contar con datos previos recabados, por lo que se optó por tomar parte de la base de datos que provee el Sistema Nacional de Información de la Calidad del Aire, SINAICA en su estación “La Pastora”, de aquí se recabaron los datos correspondientes del día 1 de enero del 2020 al 30 de julio de 2020. Con estos datos fue posible realizar evaluaciones y análisis de los datos con el fin de ajustar el algoritmo para un mejor desempeño.

En cuanto a las correcciones que recibió el algoritmo, se optó por modificar el método de selección de sobrevivientes para mejorar las posibles soluciones del algoritmo. Esto debido a que en las pruebas iniciales del algoritmo el método de selección de la población para las siguientes generaciones era tomar a los descendientes de la población actual. Para la etapa final del algoritmo, el método de selección de la población es mediante un proceso de elección por sorteo, en el cual se ordena la población actual y los descendientes en base a su desempeño con respecto a la función objetivo, las posibilidades de ser elegido varían dependiendo de su posición en dicho ordenamiento. Teniendo mayores probabilidades los individuos posicionados en las primeras posiciones.

Obtención de los datos:

El algoritmo toma los datos recabados por la estación la pastora correspondiente al intervalo del 01/01/2020 al 30/07/2020 como referencia para la generación de predicciones. Dando un total de 4250 muestras del contaminante PM10. Cada una corresponde a la medición de una hora específica del día en el intervalo mencionado. Teniendo aproximadamente 177 muestras para cada hora de las 0:00-23:00.

Selección de los datos a utilizar:

Se seleccionan al azar siguiendo una distribución uniforme 60 de las 177 muestras mencionadas previamente, correspondientes a los niveles del contaminante de partículas suspendidas PM10, las muestras seleccionadas fueron divididas en 30

que serán la población inicial y 30 que serán utilizadas para evaluar a las posibles soluciones.

Este proceso de selección se aplicará para las 24 horas, por lo que en total serán seleccionadas 1440 muestras, 720 usado como población inicial y 720 como evaluación de posibles soluciones.

Recopilación de los resultados del algoritmo evolutivo:

El resultado de la ejecución del algoritmo son 30 posibles soluciones por cada hora. Con el fin de reducir la variación entre resultados, se ejecutó 10 veces el algoritmo y se calculó un promedio entre todas las posibles soluciones, dicho promedio representa un valor estimado para cierta hora en niveles de pm10.

El promedio correspondiente a las 24 horas se utiliza para la predicción de los cambios con respecto a los niveles de pm10 a lo largo del día. Este método se basa en la idea de que existe una relación entre los niveles de contaminantes y las horas del día y es una propuesta para el desarrollo del algoritmo evolutivo diferencial.

La forma de uso de los promedios es la siguiente:

- Se calcula la diferencia entre cierta hora específica y la hora anterior para cada una de las 24 horas.
- Se calcula el promedio móvil a 12 horas para conocer los niveles de pm10 actuales.
- En base a los niveles de pm10 actuales, se aplica un incremento o decremento de estos niveles siguiendo las diferencias obtenidas por los promedios.

4.2.2 Implementación

En los algoritmos evolutivos, se maneja una población, en la cual cada individuo es una posible solución al problema, en este programa los individuos se representan mediante la clase Individuo, la cual implementa la clase comparable, esta clase se compone por un arreglo llamado vectores, el cual representa los valores de cada atributo de los individuos, y la variable val, en este caso representa el valor de los niveles de contaminantes en PM10.

```
/* @author Mario Gonzalez*/
public class Individuo implements Comparable<Individuo> {
    Double[]Vectores;
    int val;
    public Individuo(int CantVariables)
    {
        Vectores = new Double[CantVariables];
    }
    public void setVectPos(int pos, double val)
    {
        Vectores[pos]=val;
    }
    public double getVectPos(int pos)
    {
        return Vectores[pos];
    }
    public double getVal() {
        return getVectPos(0);
    }

    public void setVal(int val) {
        this.val = val;
    }
    @Override
    public int compareTo(Individuo i) {
        if(getVal()<i.getVal()){return -1;}
        if(getVal()>i.getVal()){ return 1;}
        return 0;
    }
}
```

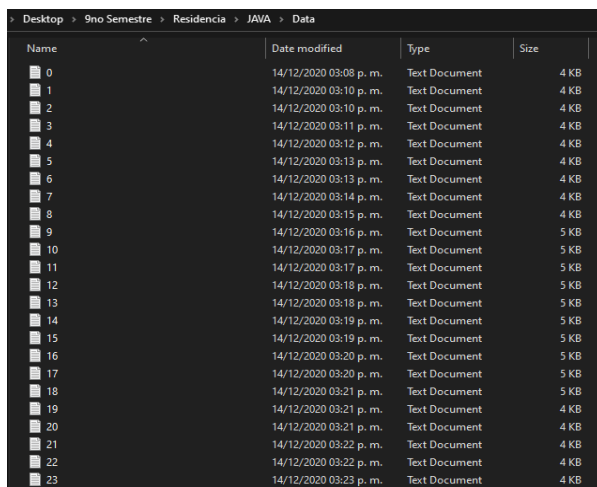
El constructor de la clase Individuo tiene como utilidad indicar el tamaño del arreglo, además de los métodos get y set para asignar y obtener los valores de un parámetro en específico.

Inicio del algoritmo

El algoritmo lee los ficheros de texto correspondientes a la hora específica de la cual se realizará el proceso de estimación, esto se realizará para las 24 horas del día. El código mediante el cual realiza la lectura es el siguiente

```
FileReader file = new FileReader("../Data/" + hora + ".txt");  
sc = new Scanner(file);
```

Aquí se enlaza el objeto de tipo Scanner "sc" con el fichero de lectura en la ruta establecida para su posterior lectura. Cada uno de estos ficheros contiene los datos correspondientes a las lecturas captadas por la estación la pastora del SINAICA, como se mencionó previamente.



Name	Date modified	Type	Size
0	14/12/2020 03:08 p. m.	Text Document	4 KB
1	14/12/2020 03:10 p. m.	Text Document	4 KB
2	14/12/2020 03:10 p. m.	Text Document	4 KB
3	14/12/2020 03:11 p. m.	Text Document	4 KB
4	14/12/2020 03:12 p. m.	Text Document	4 KB
5	14/12/2020 03:13 p. m.	Text Document	4 KB
6	14/12/2020 03:13 p. m.	Text Document	4 KB
7	14/12/2020 03:14 p. m.	Text Document	4 KB
8	14/12/2020 03:15 p. m.	Text Document	4 KB
9	14/12/2020 03:16 p. m.	Text Document	5 KB
10	14/12/2020 03:17 p. m.	Text Document	5 KB
11	14/12/2020 03:17 p. m.	Text Document	5 KB
12	14/12/2020 03:18 p. m.	Text Document	5 KB
13	14/12/2020 03:18 p. m.	Text Document	5 KB
14	14/12/2020 03:19 p. m.	Text Document	5 KB
15	14/12/2020 03:19 p. m.	Text Document	5 KB
16	14/12/2020 03:20 p. m.	Text Document	5 KB
17	14/12/2020 03:20 p. m.	Text Document	5 KB
18	14/12/2020 03:21 p. m.	Text Document	5 KB
19	14/12/2020 03:21 p. m.	Text Document	4 KB
20	14/12/2020 03:21 p. m.	Text Document	4 KB
21	14/12/2020 03:22 p. m.	Text Document	4 KB
22	14/12/2020 03:22 p. m.	Text Document	4 KB
23	14/12/2020 03:23 p. m.	Text Document	4 KB

Figura 15. Ficheros de texto con los datos de entrada del algoritmo.

El algoritmo de evolución diferencial trabaja con límites inferior y superior por lo que estos se inicializan en la siguiente parte del código:

```
switch (hora) {
    case 0:
        //Límites PM10
        limites[0][0] = 3;
        limites[0][1] = 130;

        break;
    case 1:
        //Límites PM10
        limites[0][0] = 2;
        limites[0][1] = 126;
        break;
    case 2:
        //Límites PM10
        limites[0][0] = 3;
        limites[0][1] = 130;
        break;
    ...
    case 23:
        //Límites PM10
        limites[0][0] = 2;
        limites[0][1] = 123;
        break;
}
```

Estos límites son generados basándose en los niveles mínimos y máximos alcanzados durante el periodo de recolección de los datos, estos varían con cada una de las horas, por lo que dependiendo de la hora deberá elegirse entre distintos valores para estos límites.

```
ArrayList<Individuo> DATOS = new ArrayList<>();
for (int i = 0; i < cant; i++) {
    Individuo ind = new Individuo(CANT_VAR);
    for (int j = 0; j < CANT_VAR; j++) {
        double val = sc.nextDouble();
        ind.setVectPos(j, val);
    }
    DATOS.add(ind);
}
```

Se crea un ArrayList de Individuos el cual almacenara los datos que recibe de los ficheros de texto mencionados previamente, estos datos se guardan en objetos de la clase individuo.

```
int[] elegidos = new int[cant];
int[] pobs = new int[TAM_POBLACION];
int[] test = new int[TAM_POBLACION];
```

Generación de la población inicial

El algoritmo crea 3 arreglos de tipo entero, los cuales serán utilizados para la selección de las 60 muestras explicadas en el punto “Selección de datos a utilizar”.

```
Poblacion = new ArrayList<>();
RealEval = new ArrayList<>();
for (int i = 0; i < TAM_POBLACION; i++) {
    Poblacion.add(DATOS.get(pobs[i]));
    RealEval.add(DATOS.get(test[i]));
}
```

En esta parte del código, se agregan al arraylist población los elegidos para ser la población inicial y, al arraylist RealEval los individuos elegidos para ser utilizados en la evaluación de las poblaciones.

Mezcla

Una vez generada la población inicial, y la población utilizada para la evaluación, el algoritmo procede a iniciar las etapas del algoritmo de evolución diferencial.

```
//Este metodo nos trae 3 numeros aleatorios excluyendo i
int[] rand = Elegir3Aleat(i);
//Llamamos ahora a calcularV este recibe los 4
individuos,
//individuo i, r1, r2 y r3
Individuo V;
V = CalcularV(Poblacion.get(i), Poblacion.get(rand[0]),
Poblacion.get(rand[1]),Poblacion.get(rand[2]));
```

Para el proceso de mezcla del algoritmo el cual se realizará con todos los individuos, el algoritmo elige 3 individuos al azar excluyendo al individuo a mezclar, a partir de los 4 individuos (i,r1,r2,r3) se procede a calcular el nuevo individuo V, esto mediante el método CalcularV

```
//Recibe 4 "Individuos" El primero es el de la iteracion, los otros 3
son los elegidos aleatoriamente
private Individuo CalcularV(Individuo x, Individuo i, Individuo j,
Individuo k) {
    Individuo v = new Individuo(CANT_VAR);
    for (int y = 0; y < 1; y++) {
        //Aplicacion de la formula  $V = R1 + F(R2-R3)$ 
        double Valor = i.getVectPos(y) + (F * (j.getVectPos(y) -
k.getVectPos(y)));
        //Se agrega el resultado al individuo V
        v.setVectPos(y, Valor);
    }
    //Retornamos el Individuo v
    return v;
}
```

Aplicando la formula $V = R1 + F(R2-R3)$ se realiza la mezcla de los individuos $r1, r2$ y $r3$, el individuo resultante V será utilizado para la mezcla entre el individuo V y el individuo i .

Una vez se ha generado el Individuo V se procede con el mecanismo de mezcla. Para el algoritmo de evolución diferencial esto se hace mediante el Individuo U , el cual es una mezcla entre el Individuo x_i y V . Para esto llamamos al método `ElegirU` el cual recibe los Individuos previamente mencionados, el resultado de U se agrega al arraylist "Nueva Poblacion".

Individuo U ;

```
U = ElegirU(Poblacion.get(i), V);  
nuevaPoblacion.add(U);
```

El método `ElegirU` aplica una mezcla entre los 2 Individuos siguiendo los siguientes criterios:

- 1) Uno de los parámetros (j) elegido de manera aleatoria en el rango $\{0-TAM_POBLACION\}$ deberá ser tomado del Individuo V .
- 2) El criterio para elegir el resto de los parámetros es mediante un numero aleatorio uniformemente distribuido (r), bajo la condición de que si este es mayor al factor de cruzamiento (Cr) se tomara del individuo x_i , de no cumplir la condición se toma del Individuo V .
- 3) Si se ha elegido el parámetro del Individuo V , debe verificarse que el valor correspondiente a dicho parámetro este dentro de los limites inferior y superior de dicho parámetro. Si no llega a cumplir esta condición, deberá reemplazarse por un valor aleatorio entre los rangos aceptados.

```

private Individuo ElegirU(Individuo xi, Individuo v) {
    Individuo u = new Individuo(CANT_VAR);
    int j = rng.nextInt(CANT_VAR);
    for (int i = 0; i < 1; i++) {
        double r = rng.nextDouble();
        double elegido;
        //Verificamos que r sea mayor al Cruzamiento y que i sea
diferente de j
        //Recordemos que j es el valor que si o si se tomara del
Vector v
        if (r > Cr && i != j) {
            elegido = xi.getVectPos(i);
        } else {
            //Verificamos si el numero no excede los limites
establecidos
            elegido = v.getVectPos(i);
            if (elegido < limites[i][0] || elegido > limites[i][1]) {
                //Si excede los limites se cambia por un valor
aleatorio
                //entre los limites aceptados
                elegido = rng.nextDouble() * (limites[i][1] -
limites[i][0]) + limites[i][0];
            }
        }
        u.setVectPos(i, elegido);
    }
    return u;
}

```

Una vez realizado el procedimiento con todas las soluciones de la población se procede a evaluar cada una de las soluciones con respecto a la función objetivo (función Fitness), El valor Fitness se basa el error absoluto el cual se mostró en la ecuación 7, siendo el valor real “r” el promedio de las muestras seleccionadas para la evaluación. Al final del método se retorna el valor obtenido del error absoluto.

```

public double[] EvaluarFitness(Individuo x) {
    double[] sum = new double[6];
    double[] val = new double[6];
    double r = prom;
    double d = x.getVectPos(0);
    double marg = r-d;
    marg = (marg / r) * 100;
    val[0] = Math.abs(marg);
    return val;
}

```

Para la evaluación del Fitness, utilizamos la clase con el mismo nombre, la cual implementa los métodos de la clase comparable, esto con el fin de poder ordenar el arreglo de dicha clase con el método Arrays.sort.

Selección de los sobrevivientes

```
public class Fitness implements Comparable<Fitness> {
    double val;
    int pos, gen;
    ...
    @Override
    public int compareTo(Fitness t) {
        if(getVal()<t.getVal())
        {
            return -1;
        }
        if(getVal()>t.getVal())
        {
            return 1;
        }
        return 0;
    }
}
```

Esta clase, almacena 3 valores. Val representa la medición del contaminante en las partículas superiores a 10 micras PM10, pos almacena la posición del individuo en el ArrayList Población o NuevaPoblacion, esto para saber en dónde está el individuo que fue seleccionado. Por último, gen almacena un número que indica si el individuo pertenece al ArrayList Población o pertenece a NuevaPoblacion. La probabilidad de que un individuo sea seleccionado está dada por:

$$P(A) = \frac{(n-p)}{\frac{(n)(n+1)}{2} - \sum_{i=0}^x (n-p_i)} \quad (8)$$

Donde:

n es el número total de individuos.

p es la posición en la que se encuentra en la tabla de individuos (desde 0 a n-1).

x es la cantidad de individuos que ya han sido seleccionados antes que él.

La sumatoria representa n menos la posición del individuo seleccionado en la iteración i.

pi es la posición del individuo seleccionado en la iteración i.


```

Collections.sort(POB);
ArrayList<Individuo> Seleccionados = new ArrayList();
//Creamos arreglo de probabilidad variable que se utilizara
int prob = TAM_POBLACION * 2;
int x = prob * (prob + 1) / 2;
int[] nums = new int[60];
for(int i=0;i<prob;i++)
{
    nums[i]=prob-i;
}

```

Un nuevo ArrayList “Seleccionados” se crea para almacenar en dicho arreglo los individuos seleccionados para ser la población de la siguiente generación. El arreglo nums corresponde a la probabilidad que tiene cada uno de los individuos con respecto al fitness de los individuos. El algoritmo elegirá a los individuos que serán la población para la siguiente generación.

Resultados del algoritmo

El proceso se repetirá dependiendo del número de iteraciones que se indiquen en el algoritmo, en este caso es de 15.

```

while (GEN_ACT < NUM_ITER);
return Poblacion;

```

El resultado de la ejecución del algoritmo es la población resultante de la última iteración.

```

PWDE Evol = new PWDE();
int[] prom = new int[24];
for (int i = 0; i < 24; i++) {
    Double sum = 0.0;
    for (int w = 0; w < 10; w++) {
        ArrayList<Individuo> X = Evol.Proceso(i);

        for (int j = 0; j < 30; j++) {
            double d = X.get(j).getVectPos(0);
            sum += d;
        }
    }
    sum /= 300;
    prom[i] = sum.intValue();
}

```

La clase main crea el objeto Evol de la clase PWDE, en la cual se ejecuta el proceso del algoritmo evolutivo. Se crea un arreglo prom, de tamaño 24, uno para cada una de las horas del día. El ArrayList X se iguala a la población resultante de la ejecución del algoritmo, para posteriormente en un ciclo tomar los valores de salida de todos los niveles del contaminante PM10. Al final se calcula un promedio

de los resultados obtenidos y este se asigna a la posición del arreglo prom correspondiente a la hora.

Los valores almacenados en el arreglo prom serán utilizados posteriormente para realizar las estimaciones. Se toma como punto de inicio para las estimaciones el promedio móvil ponderado a 12 horas de las partículas superiores a 10 micras PM10. El proceso de cálculo se realiza en base a la norma mencionada en el apartado “Índice Aire y Salud” del marco teórico. El código es el siguiente.

```
public class CalculadorPromedioMov {
    Double CProm;//Concentracion Promedio
    Double FactPond;//Factor de Ponderacion;
    Double w; //Valor del peso
    int Cmax = 0, Cmin = 0; //Concentracion promedio minima y maxima
    public int CalcularPromedio(int[] C) {
        //Buscar el mayor y el menor
        Cmin = C[0];
        for (int i = 0; i < 12; i++) {
            if (C[i] > Cmax) {
                Cmax = C[i];
            }
            if (C[i] < Cmin && C[i] != 0) {
                Cmin = C[i];
            }
        }
        System.out.println("Cmin: " + Cmin + " Cmax: " + Cmax);
        //Calculo del Factor de Ponderacion
        double x = Cmax - Cmin;
        double x2 = x / Cmax;
        w = (double) 1 - x2;
        System.out.println("Factor de ponderacion: " + w);
        //Falta ajustar a 2 decimales.
        //Revisar si w es mayor a 0.5, en caso contrario cambiar el valor
        por 0.5 para el factor de ponderacion
        FactPond = (w > 0.5) ? w : 0.5;
        double numerador = 0, denominador = 0;
        //Realizar Sumatoria
        for (int i = 0; i < 12; i++) {
            //En caso de que no se cuente con el promedio de una hora se
            omite en el calculo.
            if (C[i] != 0) {
                double pot = Math.pow(FactPond, i);
                numerador += C[i] * pot;
                denominador += pot;
            }
        }
        //Se obtiene el promedio (debera redondearse)
        CProm = numerador / denominador;
        System.out.println("Resultado sin redondeo; " + CProm);
        return CProm.intValue();
    }
}
public CalculadorPromedioMov() {
}
```

El algoritmo recibirá 24 mediciones introducidas por el usuario, las 12 primeras indicarán el promedio móvil ponderado utilizando el código mencionado previamente y las siguientes 12 horas serán utilizadas para comparar las estimaciones dadas por el algoritmo con los valores reales. Para los primeros 12 valores serán almacenados en el arreglo HorasPrevias de forma inversa, esto es necesario para el cálculo del promedio móvil. También será necesario conocer a partir de qué hora se realizarán las estimaciones para poder ajustar los valores a utilizar.

Los valores reales que serán comparados con las estimaciones del algoritmo se almacenan en el arreglo esperados.

```
System.out.println("¿Que hora es? "
    + "\nEJ: 1:00 = 1, 2:40 = 2");
    int hora = sc.nextInt();
    System.out.println("Introduzca los valores correspondientes a las
12 horas previas");
    int[] HorasPrevias = new int[12];
    //Tomamos las 12 horas previas para calcular el promedio movil
actual
    for (int i = 11; i >= 0; i--) {
        HorasPrevias[i] = sc.nextInt();
    }
System.out.println("Introduce los resultados esperados");
    int[] esperados = new int[12];
    for (int i = 0; i < 12; i++) {
        esperados[i] = sc.nextInt();
    }
```

Se calculan las diferencias entre los promedios de cada una de las horas, esto restando el valor de cierta hora con el valor de la hora anterior siguiendo el siguiente código:

```
int[] dif = new int[24];
dif[0] = (prom[0] - prom[23]);
for (int i = 1; i < 24; i++) {
    dif[i] = (prom[i] - prom[i - 1]);
}
```

```

for (int s = 0; s < 12; s++) {
    Prom += dif[(s + hora) % 24];
    int rango = (s+1)*2;
    int min = Prom-rango;
    int max = Prom+rango;
    System.out.println("Estimacion :" + Prom + " Esperado " +
esperados[s]);
    System.out.println("Aproximado: Min:" + min + " Max:" + max);
    double error = 0;
    if (esperados[s] >= min && esperados[s] <= max) {
        System.out.println("Dentro del rango esperado");
    } else {
        double marg = esperados[s] - Prom;
        marg = (marg / esperados[s]) * 100;
        error = Math.abs(marg);
        aciertoProm += error;
        System.out.println("Margen de error " + error);
    }
}
}

```

En un ciclo for de 0 a 12, se realizan las estimaciones sumando las diferencias de las horas al promedio móvil calculado previamente. El rango será explicado en el apartado “Pruebas aplicadas al algoritmo desarrollado para conocer su efectividad”. Si el resultado esperado está dentro del rango mínimo y máximo, no se calcula el margen de error, por el contrario, si este está fuera de dicho rango, se calculará el margen de error absoluto, este es el mismo utilizado previamente en la ejecución del algoritmo evolutivo.

Por último, se calcula el promedio del margen de error entre las 12 predicciones dadas.

```

aciertoProm /= 12;
System.out.println("-----");
System.out.println("Promedio de error" + aciertoProm);

```

Pruebas aplicadas al algoritmo desarrollado para conocer su efectividad.

Para la etapa de pruebas, se evaluará el desempeño del algoritmo mediante el cálculo del margen de error absoluta mostrada en el punto “Comparación de las estimaciones y valores reales obtenidos” comparando los valores de salida del algoritmo con los valores reales capturados por la estación de SINAICA la pastora.

Con el fin de comprobar la efectividad del algoritmo, se realizaron pruebas en varios días elegidos teniendo como única condición contar con las mediciones de los niveles del contaminante Partículas superiores a 10 micras o PM10. correspondientes a todas las horas de dicho día.

Los días elegidos fueron los siguientes:

- 1 de enero del 2020.
- 2 de febrero del 2020.
- 27 y 28 de agosto del 2020*.
- 27 de noviembre del 2020.
- 3 de diciembre del 2020.

* **Nota:** el 27 y 28 de agosto fueron elegidos para probar el algoritmo en distintas horas del día, para esto se necesitaban 48 horas consecutivas por lo que se tomaron ambos días.

También se trabajará con un rango estimado, este rango de un máximo y un mínimo se calcula a partir de la salida del algoritmo, agregando un +-2 por cada hora que ha pasado desde que se inició la predicción. Esta operación busca abarcar los cambios drásticos que ocurren en los niveles de contaminantes a ciertas horas.

4.2.3 Pruebas

Cada una de las pruebas correspondientes a los días mencionados contiene 2 tablas y 2 gráficos los cuales indican la siguiente información:

Tabla 1:

La tabla 1 contiene 4 columnas, la primera columna “Hora”, es un intervalo de tiempo al cual corresponden los datos capturados en las columnas “estimaciones” y “Resultado esperado”. Dicho intervalo de tiempo corresponde al día mencionado en el título de la tabla.

Para la segunda columna, “Estimaciones” corresponde a la predicción dada por el algoritmo de los niveles del contaminante PM10 en $\mu\text{g}/\text{m}^3$.

Los valores correspondientes a la tercera columna “Resultado esperado” indican los niveles del contaminante PM10 en $\mu\text{g}/\text{m}^3$ capturados por la estación La Pastora de SINAICA.

La cuarta columna “% de error” indica el error absoluto bajo la fórmula mostrada en el apartado “Comparación de las estimaciones y valores reales obtenidos”.

Por último, el apartado al final de la tabla “Promedio de error” es el cálculo del promedio obtenido de los 12 datos en el % de error.

Gráfico 1:

El gráfico 1 es una ayuda visual de los resultados mostrados por la tabla 1, el eje Y representa los niveles del contaminante PM10, mientras que el eje X indica el intervalo de tiempo en el que se presentó dichos niveles del contaminante mencionado. El valor en azul “Estimaciones”, al igual que en la tabla uno corresponde a la predicción dada por el algoritmo de los niveles del contaminante PM10 en $\mu\text{g}/\text{m}^3$. Y el valor en naranja indica los niveles del contaminante PM10 en $\mu\text{g}/\text{m}^3$ capturados por la estación La Pastora de SINAICA.

Tabla 2:

La tabla 2 representa los datos indicados en la tabla 1, cambiando las estimaciones del algoritmo por un rango entre un mínimo y máximo, este rango se calcula a partir de la salida del algoritmo, agregando un ± 2 por cada hora que ha pasado desde que se inició la predicción. El margen de error se calcula si el “resultado esperado” solo si el valor de este campo es menor al campo “min” o mayor al campo “max”, lo que significa que esta fuera del rango que fue establecido. Por el contrario, si está dentro de dicho rango, el margen de error será 0. El cálculo de dicho margen es el utilizado para obtener el error absoluto.

Gráfico 2:

El gráfico 2, al igual que el grafico 1 es una ayuda visual en este caso para la tabla 2, el eje Y representa los niveles del contaminante PM10 mientras que el eje X indica las horas transcurridas en la realización de las predicciones de los niveles del contaminante PM10. Este grafico muestra las series de datos correspondientes al mínimo, máximo y valor real esperado mostrados en la tabla 2.

Esta gráfica es de utilidad para ver el comportamiento del incremento y decremento de los niveles del contaminante PM10, así como también permite ver si el rango propuesto permite abarcar los niveles reales del contaminante.

Resultados 1 de enero

Tabla 1

Tabla 6. Comparación de los resultados obtenidos del 1 de enero, niveles del contaminante PM10, la estimación dada por el algoritmo evolutivo y su margen de error absoluto.

Comparación datos 1 de enero 11:00 a 2 de enero 0:00			
Hora	Estimaciones	Resultado esperado	% de error
11:00 - 12:00	30	28	7.14
12:00 - 13:00	25	26	3.85
13:00 - 14:00	22	21	4.76
14:00 - 15:00	20	15	33.33
15:00 - 16:00	14	10	40.00
16:00 - 17:00	12	9	33.33
17:00 - 18:00	11	14	21.43
18:00 - 19:00	11	9	22.22
19:00 - 20:00	11	10	10.00
20:00 - 21:00	8	10	20.00
21:00 - 22:00	13	9	44.44
22:00 - 23:00	13	10	30.00
	Promedio de error		22.54

Gráfico 1

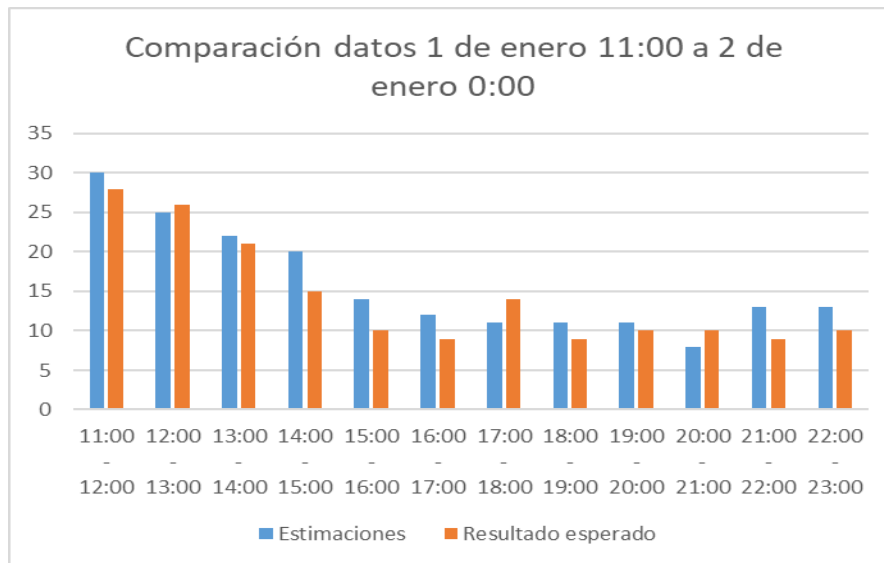


Figura 16. Comparación de los resultados del día 1 de enero, estimaciones de los niveles de PM10 dadas por el algoritmo y valores reales captados para cada intervalo de tiempo.

Como puede apreciarse en las figuras previas, los cambios en los niveles del contaminante PM10 actúan de forma inconstante, la propuesta de estimación del

algoritmo son niveles cercanos a la media, por lo que dicha estimación es difícil acierte completamente, sin embargo, estará cerca de la solución real.

Tabla 2

Tabla 7. Resultados 1 de enero utilizando un rango aproximado para los niveles del contaminante PM10 con un mínimo y máximo estimado.

Hora	Min	Max	Resultado esperado	Margen
1	28	32.00	28	0
2	21	29.00	26	0
3	16	28.00	21	0
4	12	28.00	15	0
5	4	24.00	10	0
6	0	24.00	9	0
7	0	25.00	14	0
8	0	27.00	9	0
9	0	29.00	10	0
10	0	28.00	10	0
11	0	35.00	9	0
12	0	37.00	10	0
			Promedio de error	0.00

Gráfico 2.

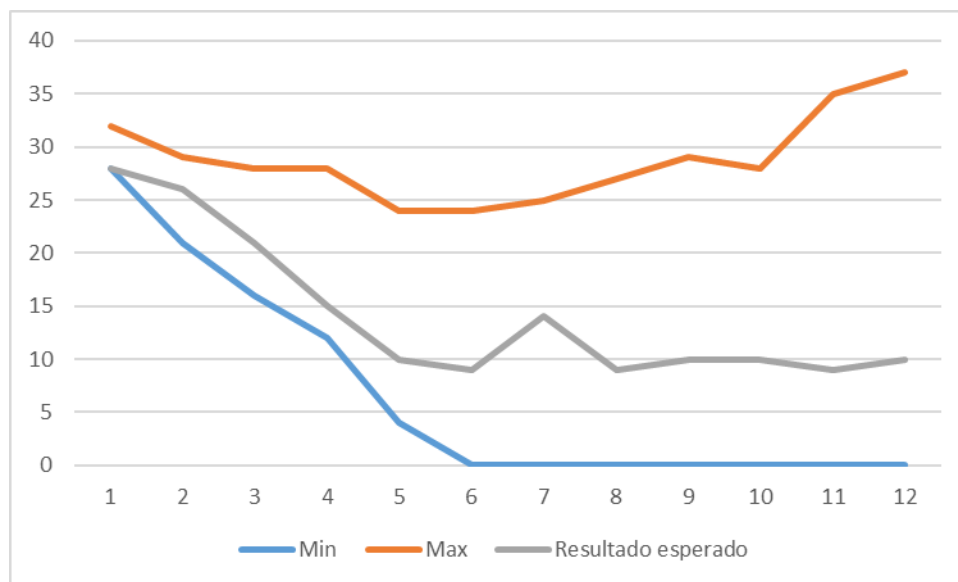


Figura 17. Comparación de los niveles del contaminante PM10 con los valores mínimo y máximo estimados.

Utilizando el margen propuesto, los valores reales para este caso, se encuentran dentro del rango dado.

Resultados 2 de febrero

Tabla 8. Comparación de los resultados obtenidos del 2 de febrero, niveles del contaminante PM10, la estimación dada por el algoritmo evolutivo y su margen de error absoluto.

Comparación datos 2 de febrero 11:00 a 3 de febrero 0:00			
Hora	Estimaciones	Resultado esperado	% de error
11:00 - 12:00	75	79	5.06
12:00 - 13:00	70	86	18.60
13:00 - 14:00	63	67	5.97
14:00 - 15:00	57	59	3.39
15:00 - 16:00	60	61	1.64
16:00 - 17:00	58	42	38.10
17:00 - 18:00	53	36	47.22
18:00 - 19:00	54	59	8.47
19:00 - 20:00	56	68	17.65
20:00 - 21:00	52	77	32.47
21:00 - 22:00	52	82	36.59
22:00 - 23:00	54	72	25.00
	Promedio de error		20.01

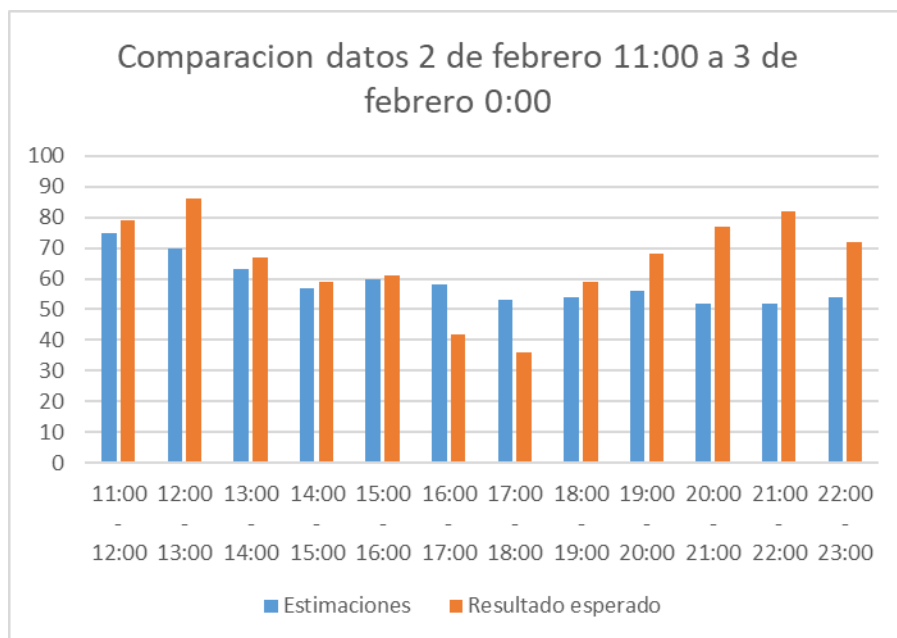


Figura 18. Comparación de los resultados del día 2 de febrero, estimaciones de los niveles de PM10 dadas por el algoritmo y valores reales captados para cada intervalo de tiempo.

Como se ha mencionado en los puntos anteriores, la estimación de un cambio más constante del algoritmo provoca que algunas veces se aleje de los valores reales, al presentar cambios más pequeños como se aprecia entre las 13:00 y las 15:00 horas, el algoritmo se acerca más a la solución correcta. Por el contrario, entre las 19:00 y 22:00 se presentan cambios más bruscos en los niveles de PM10, estos cambios normalmente no ocurrirían en dichas horas.

Tabla 9. Resultados 2 de febrero utilizando un rango aproximado para los niveles del contaminante PM10 con un mínimo y máximo estimado.

Hora	Min	Max	Resultado esperado	Margen
1	74.00	77.00	79	2.53164557
2	68.00	74.00	86	13.95348837
3	60.00	69.00	67	0
4	53.00	65.00	59	0
5	55.00	70.00	61	0
6	52.00	70.00	42	23.80952381
7	46.00	67.00	36	27.77777778
8	46.00	70.00	59	0
9	47.00	74.00	68	0
10	42.00	72.00	77	6.493506494
11	41.00	74.00	82	9.756097561
12	42.00	78.00	72	0
Promedio de error			7.03	

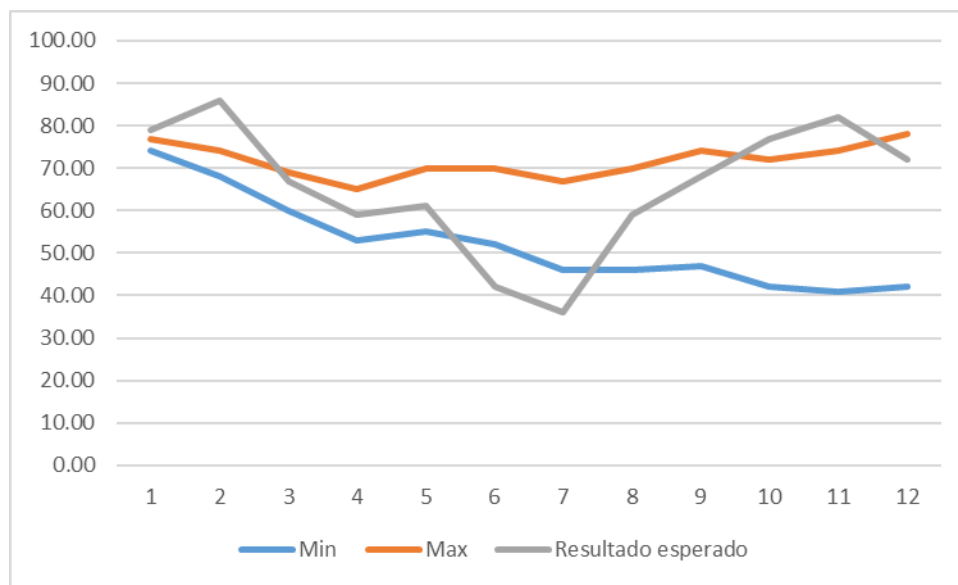


Figura 19. Comparación de los niveles del contaminante PM10 con los valores mínimo y máximo estimados.

Para este día, el cual presentó muchos cambios en los niveles del contaminante PM10, es difícil para el algoritmo abarcar estos posibles cambios en sus estimaciones por lo que aun con el margen de mínimo y máximo, algunos valores están fuera de lo que se esperaría entre los ascensos y descensos.

Primera prueba 27-agosto

Tabla 10. Comparación de los resultados obtenidos del 27 de agosto, niveles del contaminante PM10, la estimación dada por el algoritmo evolutivo y su margen de error absoluto.

Comparación datos 27 de agosto 23:00 a 28 de agosto 11:00			
Hora	Estimaciones	Resultado esperado	% de error
23:00 - 0:00	37	35	5.71
0:00 - 1:00	35	25	40.00
1:00 - 2:00	31	40	22.50
2:00 - 3:00	31	46	32.61
3:00 - 4:00	29	39	25.64
4:00 - 5:00	29	47	38.30
5:00 - 6:00	30	51	41.18
6:00 - 7:00	35	54	35.19
7:00 - 8:00	41	57	28.07
8:00 - 9:00	45	58	22.41
9:00 - 10:00	51	65	21.54
10:00 - 11:00	59	62	4.84
	Promedio de error		26.50

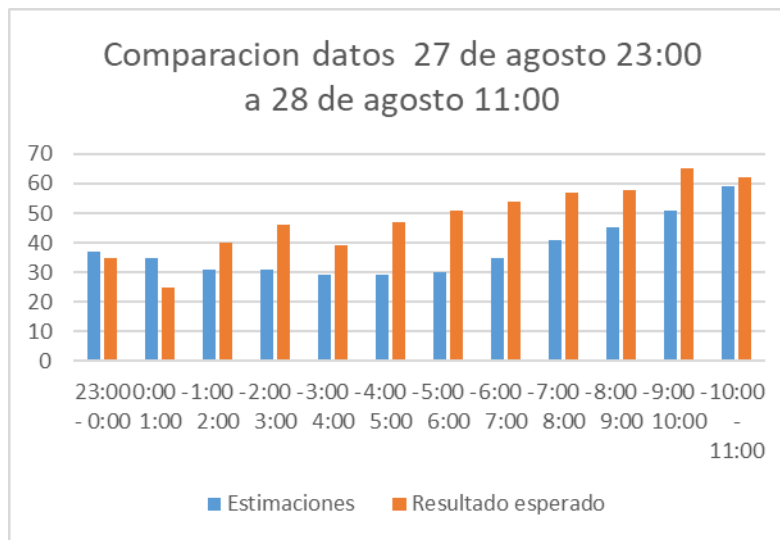


Figura 20. Comparación de los resultados del día 27 de agosto, estimaciones de los niveles de PM10 dadas por el algoritmo y valores reales captados para cada intervalo de tiempo.

Para este día, se presentó un alto promedio de error, esto debido al incremento que se dio entre la 1:00 y las 3:00, el algoritmo esperaba un incremento sin embargo este no pudo alcanzar al incremento que se dio en los valores reales.

Tabla 11. Resultados 27 de agosto utilizando un rango aproximado para los niveles del contaminante PM10 con un mínimo y máximo estimado.

Valores con margen				
Hora	Min	Max	Resultado esperado	Margen
1	35.00	39.00	35	0
2	31.00	39.00	25	24
3	25.00	37.00	40	7.5
4	23.00	39.00	46	15.2173913
5	19.00	39.00	39	0
6	17.00	41.00	47	12.76595745
7	16.00	44.00	51	13.7254902
8	19.00	51.00	54	5.555555556
9	23.00	59.00	57	3.50877193
10	25.00	65.00	58	0
11	29.00	73.00	65	0
12	35.00	83.00	62	0
			Promedio de error	6.86

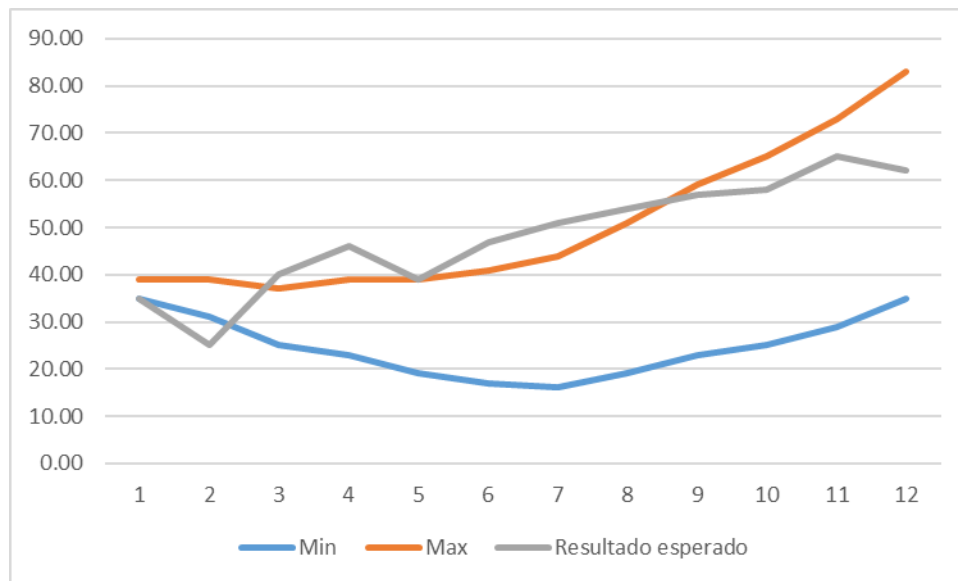


Figura 21. Comparación de los niveles del contaminante PM10 con los valores mínimo y máximo estimados.

Resultados 28 agosto

Tabla 12. Comparación de los resultados obtenidos del 28 de agosto, niveles del contaminante PM10, la estimación dada por el algoritmo evolutivo y su margen de error absoluto.

Comparación datos 28 de agosto 11:00 a 29 de agosto 0:00			
Hora	Estimaciones	Resultado esperado	% de error
11:00 - 12:00	63	62	1.61
12:00 - 13:00	55	68	19.12
13:00 - 14:00	52	66	21.21
14:00 - 15:00	45	56	19.64
15:00 - 16:00	46	45	2.22
16:00 - 17:00	42	41	2.44
17:00 - 18:00	42	32	31.25
18:00 - 19:00	42	33	27.27
19:00 - 20:00	43	34	26.47
20:00 - 21:00	42	24	75.00
21:00 - 22:00	39	53	26.42
22:00 - 23:00	45	33	36.36
	Promedio de error		24.08

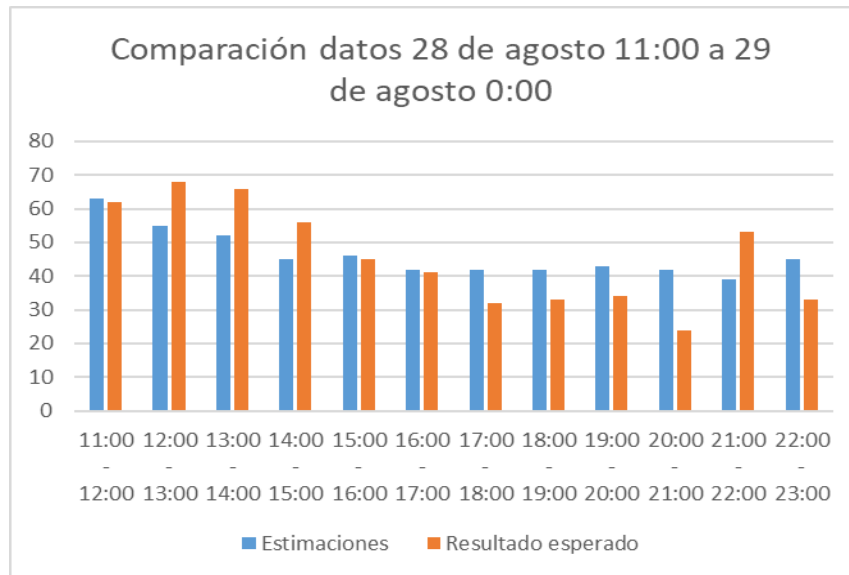


Figura 22. Comparación de los resultados del día 28 de agosto, estimaciones de los niveles de PM10 dadas por el algoritmo y valores reales captados para cada intervalo de tiempo.

Como se ha mencionado previamente, el algoritmo busca mantener sus estimaciones cercanas a la media esperada, por lo que los valores que indica son cercanos, pero en casos con incrementos o descensos altos en los niveles de contaminantes es donde presenta más fallos.

Tabla 13. Resultados 28 de agosto utilizando un rango aproximado para los niveles del contaminante PM10 con un mínimo y máximo estimado.

Hora	Min	Max	Resultado esperado	Margen
1	61.00	65.00	62	0
2	51.00	59.00	68	13.23529412
3	46.00	58.00	66	12.12121212
4	37.00	53.00	56	5.357142857
5	36.00	56.00	45	20
6	30.00	54.00	41	0
7	28.00	56.00	32	0
8	26.00	58.00	33	0
9	25.00	61.00	34	0
10	22.00	62.00	24	0
11	17.00	61.00	53	0
12	21.00	69.00	33	0
Promedio de error			4.23	

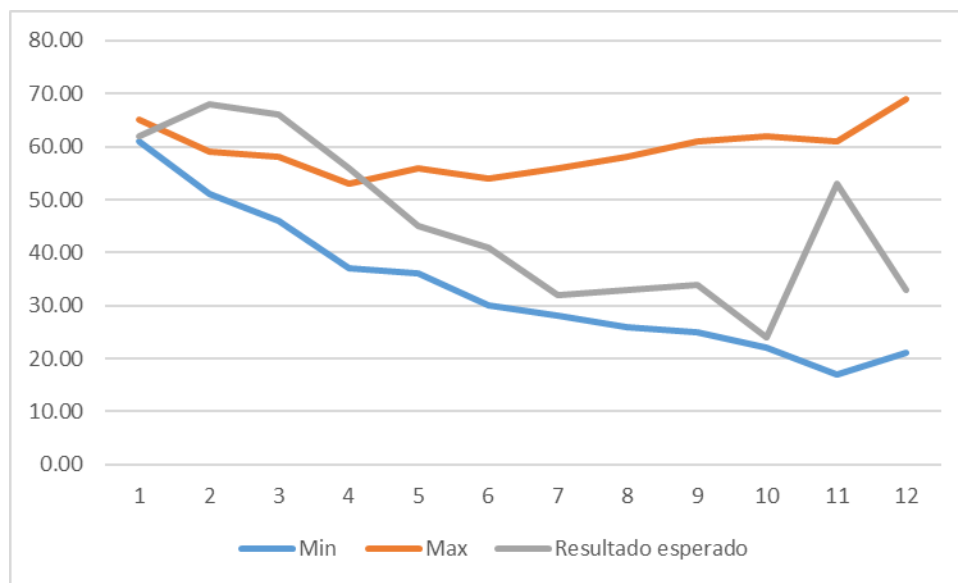


Figura 23. Comparación de los niveles del contaminante PM10 con los valores mínimo y máximo estimados.

En este caso, con el margen propuesto es posible abarcar los incrementos dados en 8 de las 12 horas estimadas, por lo que el resultado es satisfactorio. Las necesidades de agrandar el margen conforme pasan las horas ayuda a los cambios drásticos en los niveles de PM10 a los cuales, el algoritmo no es capaz de abarcar por sí solo.

Resultados 27 de noviembre

Tabla 14 Comparación de los resultados obtenidos del 27 de noviembre, niveles del contaminante PM10, la estimación dada por el algoritmo evolutivo y su margen de error absoluto.

Comparación datos 27 de noviembre 11:00 a 28 de noviembre 0:00			
Hora	Estimaciones	Resultado esperado	% de error
11:00 - 12:00	99	108	8.33
12:00 - 13:00	95	113	15.93
13:00 - 14:00	89	128	30.47
14:00 - 15:00	89	83	7.23
15:00 - 16:00	89	90	1.11
16:00 - 17:00	84	80	5.00
17:00 - 18:00	80	80	0.00
18:00 - 19:00	82	82	0.00
19:00 - 20:00	81	64	26.56
20:00 - 21:00	76	57	33.33
21:00 - 22:00	80	62	29.03
22:00 - 23:00	80	90	11.11
Promedio de error			14.01

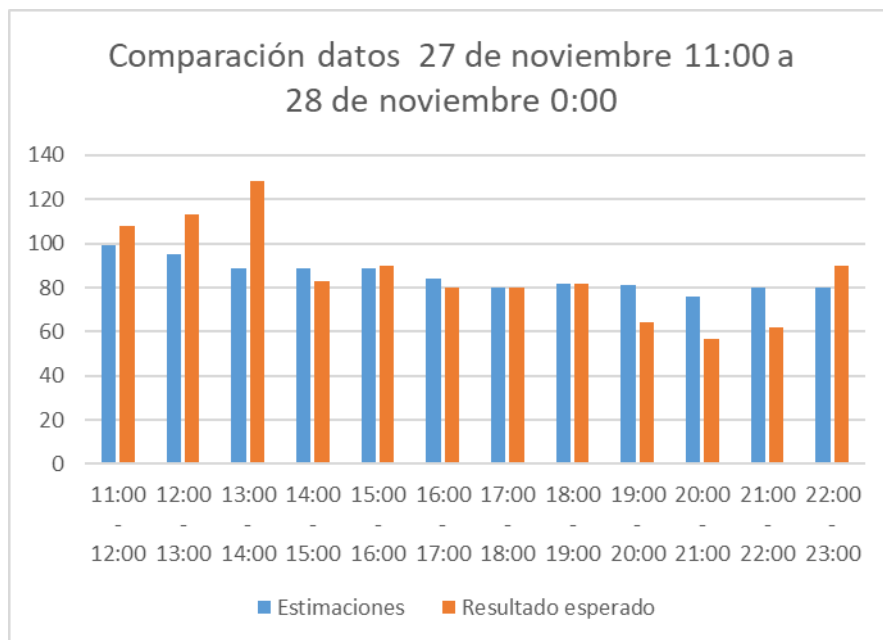


Figura 24. Comparación de los resultados del día 27 de noviembre, estimaciones de los niveles de PM10 dadas por el algoritmo y valores reales captados para cada intervalo de tiempo.

Como puede apreciarse en la figura, al mantenerse estables los niveles del contaminante PM10, el algoritmo acierta en 2 ocasiones y se mantiene cercano al resultado esperado en otras, los mayores márgenes de error se presentan al momento de haber el mayor incremento a las 13:00 horas.

Tabla 15. Resultados 27 de noviembre utilizando un rango aproximado para los niveles del contaminante PM10 con un mínimo y máximo estimado.

Hora	Min	Max	Resultado esperado	Margen
1	97.00	101.00	108	12.38938053
2	91.00	99.00	113	25.78125
3	83.00	95.00	128	25.78125
4	81.00	97.00	83	2.409638554
5	79.00	99.00	90	0
6	72.00	96.00	80	0
7	66.00	94.00	80	0
8	66.00	98.00	82	0
9	63.00	99.00	64	1.5625
10	56.00	96.00	57	1.754385965
11	58.00	102.00	62	6.451612903
12	56.00	104.00	90	0
			Promedio de error	6.34

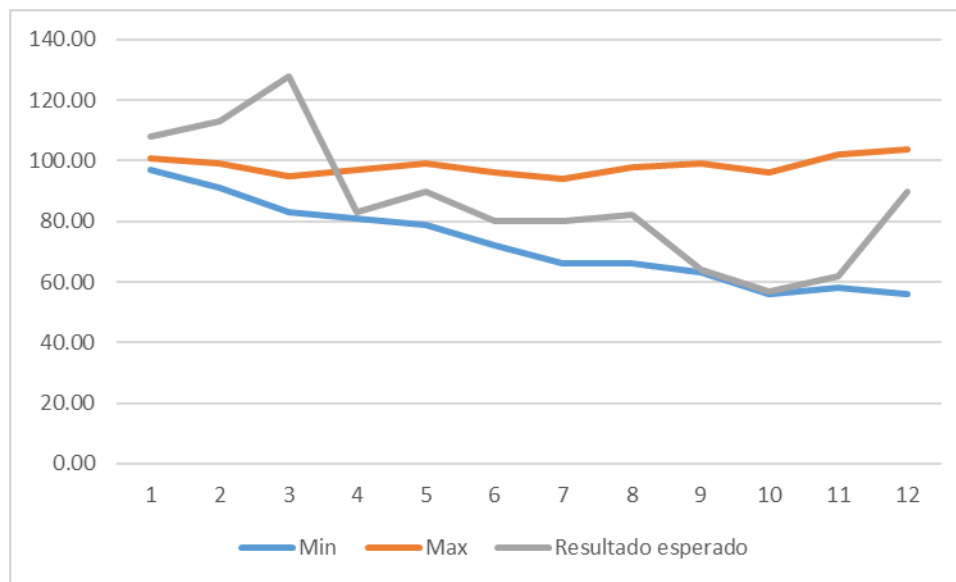


Figura 25. Comparación de los niveles del contaminante PM10 con los valores mínimo y máximo estimados.

Al igual que en pruebas anteriores, parte de los resultados esperados se encuentran dentro del margen establecido.

Resultados 3 de diciembre

Tabla 16. Comparación de los resultados obtenidos del 3 de diciembre, niveles del contaminante PM10, la estimación dada por el algoritmo evolutivo y su margen de error absoluto.

Comparación datos 3 de diciembre 11:00 a 4 de diciembre 0:00			
Hora	Estimaciones	Resultado esperado	% de error
11:00 - 12:00	65	64	1.56
12:00 - 13:00	55	50	10.00
13:00 - 14:00	55	39	41.03
14:00 - 15:00	52	35	48.57
15:00 - 16:00	53	39	35.90
16:00 - 17:00	46	37	24.32
17:00 - 18:00	49	41	19.51
18:00 - 19:00	47	65	27.69
19:00 - 20:00	47	40	17.50
20:00 - 21:00	45	26	73.08
21:00 - 22:00	46	38	21.05
22:00 - 23:00	43	56	23.21
Promedio de error			28.62

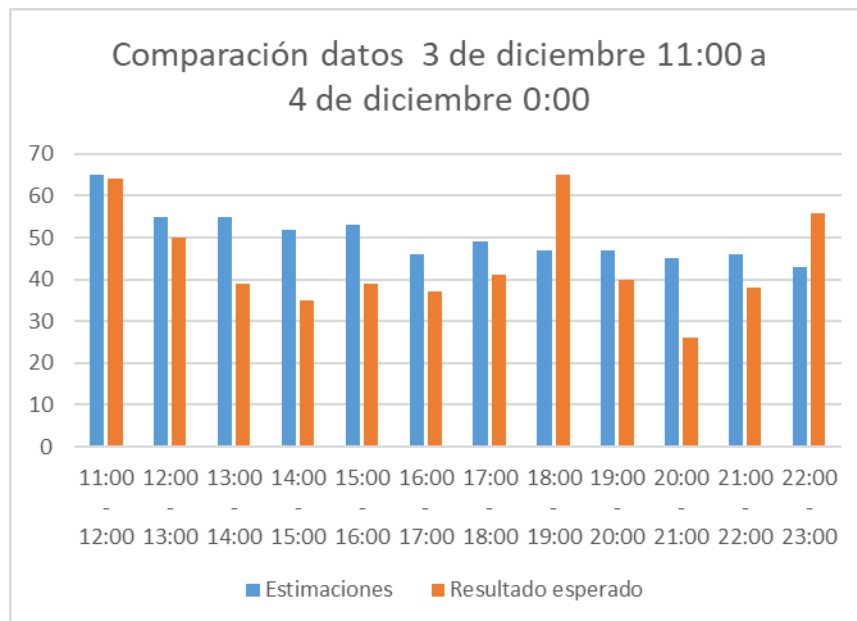


Figura 26. comparación de los resultados del día 3 de diciembre, estimaciones de los niveles de PM10 dadas por el algoritmo y valores reales captados para cada intervalo de tiempo.

Tabla 17. Resultados 3 de diciembre utilizando un rango aproximado para los niveles del contaminante PM10 con un mínimo y máximo estimado.

Hora	Min	Max	Resultado esperado	Margen
1	63.00	67.00	64	0
2	51.00	59.00	50	2
3	49.00	61.00	39	25.64102564
4	44.00	60.00	35	25.71428571
5	43.00	63.00	39	10.25641026
6	34.00	58.00	37	8.108108108
7	35.00	63.00	41	0
8	31.00	63.00	65	3.076923077
9	29.00	65.00	40	0
10	25.00	65.00	26	3.846153846
11	24.00	68.00	38	0
12	19.00	67.00	56	0
			Promedio de error	6.55

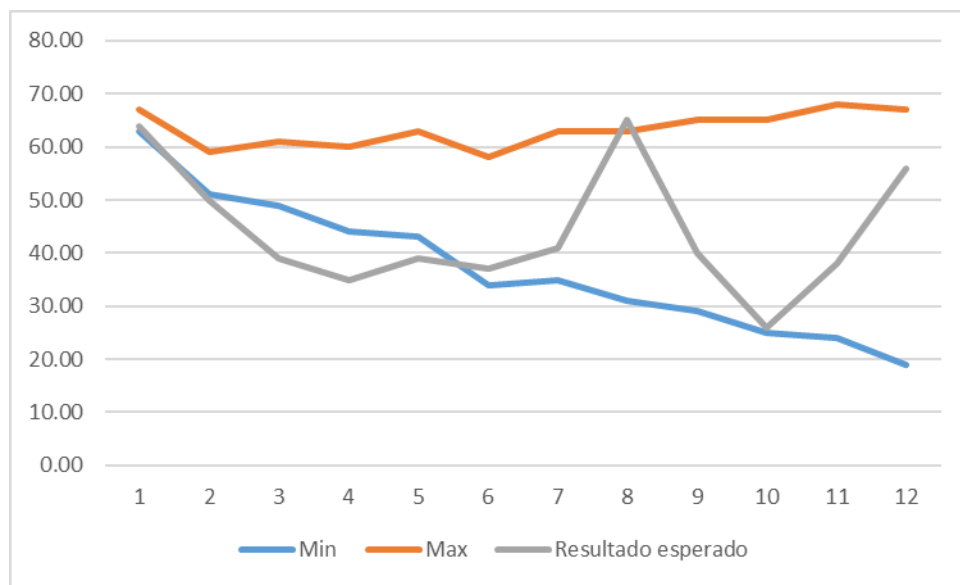


Figura 27. Comparación de los niveles del contaminante PM10 con los valores mínimo y máximo estimados.

4.2.4 Conclusiones de los resultados obtenidos.

Analizando los resultados obtenidos se puede apreciar un porcentaje de error absoluto entre 20% y 30% al compararse con el valor exacto de salida con respecto a los niveles del contaminante PM10. Esto ocurre debido a que la predicción de salida del algoritmo es un valor medio entre los valores estimados para cierta hora específica.

Al buscar un punto medio en donde podrían estar los niveles de los contaminantes, es necesario un segundo sistema el cual sea capaz de acercar este valor lo más posible al valor real estimado.

Aplicando el margen de ± 2 por cada hora transcurrida es posible abarcar los cambios drásticos en los niveles de contaminantes que ocurren durante el día, sin embargo, el margen es cada vez más grande y por lo tanto menos preciso. Por lo que sería más óptimo aplicar algún otro algoritmo extra para dicha función el cual pudiese predecir cuándo ocurrirán estos cambios drásticos, sin embargo, para eso podría ser necesario un modelo de predicción meteorológico el cual este enfocado en el movimiento y dispersión de contaminantes en el aire.

CAPITULO V- CONCLUSIONES Y TRABAJOS FUTUROS

5.1 Conclusiones de Proyecto.

Se abordó la problemática de monitoreo puntual por zonas específicas bajo un histórico de mediciones del contaminante criterio de partículas suspendidas PM10, dándole una solución alterna de monitoreo mediante algoritmos evolutivos. Se implementó y codificó un algoritmo evolutivo mediante la técnica diferencial, el cual facilitó el análisis del comportamiento de los contaminantes de estudio bajo datos históricos. Identificando emisiones mínimas y máximas, siguiendo la normativa NOM-172-SERMANAT-2019 vigente en México.

El desarrollo del algoritmo evolutivo diferencial está basado en la propuesta de R. Storn, K. Price en su reporte técnico “Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces“(1997). Para la implementación de este algoritmo, se llevó a cabo un análisis del comportamiento de los niveles del contaminante PM10 en un intervalo de tiempo determinado, estos datos fueron recabados por la estación “La Pastora” del SINAICA en el municipio de Guadalupe, Nuevo León México, en un periodo de tiempo correspondiente del 1 de enero del 2020 al 31 de julio del 2020. Los resultados obtenidos fueron utilizados para determinar el funcionamiento implementado en el algoritmo para la realización de las estimaciones finales.

En los procesos de experimentación del algoritmo, al ser codificado desde cero, se optó por probarlo en su primera fase con un ejercicio de optimización: el problema de la mochila, esto con el fin de comprobar que funcionase correctamente en sus procesos de cruce, mutación y selección de la población. Durante la segunda fase de experimentación se realizaron pruebas de predicción de niveles del contaminante PM10, siendo evaluado mediante el porcentaje de error absoluto comparando los valores reales capturados contra las estimaciones dadas por el algoritmo evolutivo. La selección de los días que fueron tomados para pruebas estaban sujetos a contar con las mediciones correspondientes a las 24 horas de dicho día, esto para hacer las estimaciones y evaluaciones correctamente. Estos

días fueron recabados por la estación “La Pastora” del SINAICA entre el 1 de enero de 2020 y el 10 de diciembre del 2020.

En cuanto al objetivo de la tesis “Implementar un algoritmo evolutivo para la búsqueda óptima de concentración del contaminante criterio partículas suspendidas PM10”. Se cumple la implementación del algoritmo evolutivo realizando estas predicciones de la concentración de los niveles del contaminante PM10, sin embargo, las estimaciones del algoritmo presentan aún un error absoluto promedio de entre 20% y 30%, esto se reduce significativamente implementando la propuesta dada para el segundo experimento, llegando a promediar entre un 5% y 15% de error. Debido a la incertidumbre que existe entre los posibles cambios en los niveles del contaminante PM10, resultó complicado reducir este promedio de error absoluto.

Tomando la hipótesis planteada desde los inicios de este trabajo de tesis: “La concentración mínima y máxima de un contaminante criterio puede ser obtenida de una base de datos histórica mediante un algoritmo evolutivo, evaluando un periodo de tiempo determinado, encontrando una solución buena en un tiempo de computo razonable”, se cumple en su totalidad, ya que para la ejecución del algoritmo evolutivo, encontrar los mínimos y máximos de un conjunto de datos seleccionados contribuye a las funciones utilizadas en el algoritmo de predicción. Estos mínimos y máximos son utilizados para encontrar una media razonable, la cual será utilizada para dar las estimaciones en los niveles del contaminante PM10. El sistema evalúa en intervalos de tiempo determinados, ya que se basa en la idea de que los niveles del contaminante PM10 descienden o ascienden de manera similar con respecto a las horas del día. En cuanto al tiempo de ejecución, al presentar una complejidad computacional de $O(n^2)$ representa bajos costos en tiempo y realización de procesos, siendo posible su ejecución en equipos de computo básicos y en un tiempo de ejecución razonable.

En escenarios donde se presentan menores incrementos y decrementos de los niveles del contaminante PM10 el algoritmo produce mejores resultados, esto debido a que este busca un punto medio entre los mínimos y máximos posibles,

aun sin esto el algoritmo es capaz de dar aproximaciones cercanas a los niveles reales del contaminante PM10, dado los experimentos realizados y presentados en los resultados de este reporte de tesis.

5.2 Trabajos futuros

En cuanto a la problemática planteada para esta tesis, un área de oportunidad podría ser la implementación de este algoritmo en un medio al alcance de la población, para este proyecto de experimentación hizo falta su implementación en este aspecto. Con respecto a investigaciones futuras, el análisis y predicción de los niveles de contaminantes es muy amplio, es posible enfocar el análisis en distintos contaminantes criterio que no fueron tomados en cuenta para este algoritmo, como PM2.5, considerando el análisis de los niveles del contaminante en ciertas horas específicas, este fenómeno también puede ser investigado para encontrar los motivos que provocan dicho cambio.

Otra área de oportunidad puede ser la implementación de un sistema de monitoreo utilizando otros algoritmos evolutivos para comprobar su efectividad en estos casos, como las ideas proporcionadas por los autores mencionados en el punto de “antecedentes”, en los cuales utilizaban técnicas como Random Forest, redes neuronales o lógica difusa. Los tipos de algoritmos evolutivos son bastante amplios, por lo que probar distintas técnicas para el mismo problema de monitoreo puede dar resultados interesantes.

FUENTES DE INFORMACIÓN

Cortina Januchs, M. G. (2012). Aplicación de técnicas de inteligencia artificial a la predicción de contaminantes atmosféricos (Doctoral dissertation, Telecomunicacion).

INEGI. (2015). Panorama sociodemográfico de Nuevo León. Obtenida el 15 de julio de 2020, de http://internet.contenidos.inegi.org.mx/contenidos/Productos/prod_serv/contenidos/espanol/bvinegi/productos/nueva_estruc/inter_censal/panorama/702825082291.pdf

(INECC). (2010) ESTUDIO DE EMISIONES Y ACTIVIDAD VEHICULAR EN EL ÁREA METROPOLITANA DE MONTERREY, N.L. Obtenido el 10 de agosto de 2020, de: https://www.gob.mx/cms/uploads/attachment/file/112367/2010_CGCSA_RSD_Monterrey.pdf

Espinosa Guzmán, A.A. May Tzuc, O. Pantí Balam, I. Reyes Trujeque, J. Pérez Quintana, I. V. & Bassam, A. (2017). MODELADO DE PARTÍCULAS PM10 Y PM2.5 MEDIANTE REDES NEURONALES ARTIFICIALES SOBRE CLIMA TROPICAL DE SAN FRANCISCO DE CAMPECHE, MÉXICO. *Química Nova*, 40(9), 1025-1034.

Recuperado el 10 de agosto de 2020, de <https://doi.org/10.21577/0100-4042.20170115>

Santana, L. V. (2004, noviembre). *Un Algoritmo Basado en Evolución Diferencial para Resolver Problemas Multiobjetivo*. Recuperado el 10 de agosto de 2020, de: <https://www.cs.cinvestav.mx/TesisGraduados/2004/tesisLuisVSantana.pdf>.

Rubal. (2017, 10 October). *Air pollution prediction via Differential evolution strategies with random forest method*. IRJET. Recuperado el 8 de agosto de 2020, de: <https://www.irjet.net/archives/V4/i10/IRJET-V4I10198.pdf>

F. Hillier and G. Lieberman, *Introducción a la investigación de operaciones*. Editorial McGraw-Hill, 2001

Merino M. (s.f.) *Técnicas Clásicas de Optimización* Recuperado el 10 de agosto del 2020, de: http://www.ehu.eus/mae/html/prof/Maria_archivos/plnlapuntes.pdf

ACADEMIA MEXICANA DE COMPUTACIÓN, A, C. (2017). *La computación en México por especialidades académicas (Primera Edición)*. Obtenido el 28 de julio de 2020, recuperado de

<https://www.gelbukh.com/CV/Publications/2017/La%20Computacion%20en%20Mexico%20por%20especialidades%20academicas.pdf>

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning*.

R. Storn, K. Price (1997) *Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces*. Technical Report

NORMA Oficial Mexicana NOM-172-SEMARNAT-2019, *Lineamientos para la obtención y comunicación del Índice de Calidad del Aire y Riesgos a la Salud*.

Bosman, P. *EA Visualizer* Obtenido el 20 de agosto de 2020, Recuperado de : <http://www.cems.uwe.ac.uk/~apipe/Int%20and%20Adapt%20Sys/Revision%20material%20CD%20image/evonet.dcs.napier.ac.uk/demo70.html>

Riquelme Medina, I. (2014). Revisión de los Algoritmos Bio-inspirados. Obtenido el 10 de agosto de 2020, Recuperado de: https://www.researchgate.net/publication/303803004_Revision_de_los_Algoritmos_Bioinspirados

Forbes N. (2004) IMITATION OF LIFE. How Biology Is Inspiring Computing.

Ribeiro Filho, J., Alippi, C., & Treleaven, P. (s. f.). *Genetic Algorithm Programming Environments*. delta. Obtenido el 15 de Agosto de 2020, Recuperado de <http://delta.cs.cinvestav.mx/~ccoello/compevol/gaprogramming.pdf>

Introducción a Algoritmos evolutivos. (s. f.). UC3M. Obtenido el 17 de agosto de 2020, Recuperado de: <http://www.exa.unicen.edu.ar/escuelapav/cursos/bio/l2.pdf>

E. Mesa (Hablemos de Optimización). (2018, diciembre 11). Evolución diferencial [Archivo de video] Recuperado de: <https://www.youtube.com/watch?v=FOGGoKytVvc&t=942s>

ANEXOS A: Código del programa

Clase Individuo

```
/**
 *
 * @author Mario Gonzalez
 */
public class Individuo implements Comparable<Individuo> {
    Double[]Vectores;

    public Individuo(int CantVariables)
    {
        Vectores = new Double[CantVariables];
    }
    public void setVectPos(int pos, double val)
    {
        Vectores[pos]=val;
    }
    public double getVectPos(int pos)
    {
        return Vectores[pos];
    }
    public double getVal() {
        return getVectPos(0);
    }
    public void setVal(int val) {
        this.val = val;
    }
    int val;
    @Override
    public int compareTo(Individuo i) {
        if(getVal()<i.getVal())
        {
            return -1;
        }
        if(getVal()>i.getVal())
        {
            return 1;
        }
        return 0;
    }

    @Override
    public String toString(){
        String text="";
        for(int i=0;i<Vectores.length;i++)
        {
            text+="["+ i + "] " + Vectores[i]+" ";
        }
        return text;
    }
}
```

ANEXOS B: Código del experimento 1

Clase EvolucionDiferencial

```
import java.util.ArrayList;
import java.util.Random;

/**
 *
 * @author Mario González Rodríguez
 */
public class EvolucionDiferencial {

    // Tamaño de la Poblacion Se recomiendan mas de 30
    // Numero de iteraciones a realizar
    // Cantidad de variables de cada individuo
    static int NUM_ITER = 50, TAM_POBLACION = 30, GEN_ACT = 0;
    //Lista de los individuos que habra
    static ArrayList<Individuo> Poblacion, nuevaPoblacion;
    //Arreglo para determinar los limites superior e inferior de los
valores
    static double[][] limites;
    //Factor de mutacion y cruzamiento (generalmente es un numero bajo)
    static double F = 1.2, Cr = 0.4;
    static Random rng;
    //Resultado que se busca obtener de la funcion fitness

    //Valores del problema de ejemplo, pj corresponde a los valores de
cada objeto
    //wj corresponde a los pesos correspondientes a cada objeto
    //c es la capacidad de la mochila

    //COMB 65,536
    //16 Objetos OPTM 2130-207 0 1 1 1 0 1 1 0 1 1 1 1 0 1 1 1
    static int[] pj = {350, 400, 450, 20, 70, 8, 5, 5,360, 398, 445, 23, 74,
8, 7, 6};
    static int[] wj = {25, 35, 45, 5, 25, 3, 2, 2,25, 35, 45, 5, 25, 3, 2,
2};
    static int c = 208, CANT_VAR=16;
    //Almacenan el mejor de todas las generaciones
    static Individuo best;
    static int MayorFit,pMF,cAcum,gen;
    //Se guarda el mejor de cada generacion
    static int FitGen,gMF, gCAcum;

    public static void main(String[] args) {
        rng = new Random();
        crearLimitesPares();
        generarPoblacion();
        GEN_ACT = 1;
        best = new Individuo(CANT_VAR);
        //Mostramos la poblacion inicial que se ha generado
        MostrarPoblacion();
        //Realiacion del ciclo
        do {
```

```

nuevaPoblacion = new ArrayList<>();
//iteracion
for (int i = 0; i < TAM_POBLACION; i++) {
    //Este metodo nos trae 3 numeros aleatorios excluyendo i
    int[] rand = Elegir3Aleat(i);
    //Llamamos ahora a calcularV este recibe los 4
individuos,
    //individuo i, r1, r2 y r3
    Individuo V;
    V = CalcularV(Poblacion.get(i), Poblacion.get(rand[0]),
Poblacion.get(rand[1]),
    Poblacion.get(rand[2]));
    //Se genera el individuo U, quien se agregara al vector
de la nueva poblacion
    Individuo U;
    U = ElegirU(Poblacion.get(i), V);
    nuevaPoblacion.add(U);
}
FitGen=0;
gCAcum=0;
gMF=0;
for (int i = 0; i < TAM_POBLACION; i++) {
    int[] Fitness = EvaluarFitness(nuevaPoblacion.get(i));
    if(Fitness[0]>FitGen && Fitness[1]<=c)
    {
        FitGen=Fitness[0];
        gCAcum=Fitness[1];
        gMF=i;
    }
    if(Fitness[0]>MayorFit && Fitness[1]<=c)
    {
        best=nuevaPoblacion.get(i);
        MayorFit=Fitness[0];
        cAcum=Fitness[1];
        pMF=i;
        gen=GEN_ACT;
    }
}
MostrarMejor(nuevaPoblacion.get(gMF),FitGen,gCAcum,
gMF,GEN_ACT);
Poblacion = nuevaPoblacion;
GEN_ACT++;
} //Evaluacion con respecto a si se encontro la solucion o se
realizaron las iteraciones
while (GEN_ACT < NUM_ITER);

MostrarMejor(best,MayorFit,cAcum,pMF,gen);
}

private static void crearLimitesPares() {
    limites = new double[CANT_VAR][2];
    //El primero es el limite inferior y el segundo el limite
superior
    for (int i = 0; i < CANT_VAR; i++) {
        limites[i][0] = 0;
        limites[i][1] = 1;
    }
}

```

```

    }

    private static void generarPoblacion() {
        Poblacion = new ArrayList<>();
        for (int i = 0; i < TAM_POBLACION; i++) {
            //Se crea un nuevo individuo
            Individuo ind = new Individuo(CANT_VAR);
            //se generan los valores aleatorios del individuo
            for (int j = 0; j < CANT_VAR; j++) {
                double val = rng.nextDouble() * limites[j][1] +
limites[j][0];
                ind.setVectPos(j, val);
            }
            Poblacion.add(ind);
        }
    }

    private static int[] Elegir3Aleat(int per) {
        int[] aleat = new int[3];
        boolean Fin = false;
        do {
            for (int i = 0; i < 3; i++) {
                aleat[i] = rng.nextInt(TAM_POBLACION);
                if (aleat[i] == per) {
                    i--;
                }
            }
            if (aleat[0] != aleat[1] && aleat[0] != aleat[2] && aleat[1]
!= aleat[2]) {
                Fin = true;
            }
        } while (Fin != true);
        return aleat;
    }

    //Recibe 4 "Individuos" El primero es el de la iteracion, los otros 3
son los elegidos aleatoriamente
    private static Individuo CalcularV(Individuo x, Individuo i,
Individuo j, Individuo k) {
        Individuo v = new Individuo(CANT_VAR);
        for (int y = 0; y < CANT_VAR; y++) {
            //Aplicacion de la formula  $V = R1 + F(R2-R3)$ 
            double Valor = i.getVectPos(y) + (F * (j.getVectPos(y) -
k.getVectPos(y)));
            //Se agrega el resultado al individuo V
            v.setVectPos(y, Valor);
        }
        //Retornamos el Individuo v
        return v;
    }

    private static Individuo ElegirU(Individuo xi, Individuo v) {
        Individuo u = new Individuo(CANT_VAR);
        int j = rng.nextInt(CANT_VAR);
        for (int i = 0; i < CANT_VAR; i++) {
            double r = rng.nextDouble();
            double elegido;

```

```

        //Verificamos que r sea mayor al Cruzamiento y que i sea
diferente de j
        //Recordemos que j es el valor que si o si se tomara del
Vector v
        if (r > Cr && i != j) {
            elegido = xi.getVectPos(i);
        } else {
            //Verificamos si el numero no excede los limites
establecidos
            elegido = v.getVectPos(i);
            if (elegido < limites[i][0] || elegido > limites[i][1]) {
                //Si excede los limites se cambia por un valor
aleatorio
                //entre los limites aceptados
                elegido = rng.nextDouble() * limites[j][1] +
limites[j][0];
            }
        }
        u.setVectPos(i, elegido);
    }
    return u;
}

public static int[] EvaluarFitness(Individuo x) {
    int[] val = new int[2];
    int fitness = 0;
    int acum = 0;
    //Aqui se evalua la funcion fitness para cada individuo
    for (int i = 0; i < CANT_VAR; i++) {
        double d = x.getVectPos(i);
        if (d > 0.5) {
            fitness += pj[i];
            acum += wj[i];
        }
    }
    val[0]=fitness;
    val[1]=acum;
    return val;
}

public static void MostrarPoblacion() {
    System.out.println("-----");
    System.out.println("Generacion " + GEN_ACT);
    for (int i = 0; i < TAM_POBLACION; i++) {
        System.out.println("Individuo " + i);
        String pob = "";
        for (int j = 0; j < CANT_VAR; j++) {
            double d = Poblacion.get(i).getVectPos(j);
            if (d > 0.5) {
                pob += "1 ";
            } else {
                pob += "0 ";
            }
        }
        System.out.println(pob);
    }
}
}

```

```

    public static void MostrarMejor(Individuo x, int Fitness, int c, int
p,int gen){
        System.out.println("-----");
        System.out.println("MEJOR DE LA GENERACION " + gen);
        String pob="";
        for (int j = 0; j < CANT_VAR; j++) {
            double d = x.getVectPos(j);
            if (d > 0.5) {
                pob += "1 ";
            } else {
                pob += "0 ";
            }
        }
        System.out.println("Ind " + p + " CON " + pob + " con " +
Fitness + " puntaje de fitness y peso " + c);
        System.out.println("-----");
    }
}

```


ANEXOS C: Código del experimento 2

Clase Fitness

```
/**
 *
 * @author Mario Gonzalez
 */
public class Fitness implements Comparable<Fitness> {
    double val;
    int pos, gen;

    public Fitness(double val, int pos, int gen) {
        this.val = val;
        this.pos = pos;
        this.gen = gen;
    }

    public double getVal() {
        return val;
    }

    public void setVal(double val) {
        this.val = val;
    }

    public int getPos() {
        return pos;
    }

    public void setPos(int pos) {
        this.pos = pos;
    }

    public int getGen() {
        return gen;
    }

    public void setGen(int gen) {
        this.gen = gen;
    }

    @Override
    public int compareTo(Fitness t) {
        if(getVal()<t.getVal())
        {
            return -1;
        }
        if(getVal()>t.getVal())
        {
            return 1;
        }
        return 0;
    }
}
```

Clase PWDE:

Esta clase es la utilizada para el proceso de evolución diferencial en el experimento 2 en lugar de utilizar la mostrada en el experimento 1.

```
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.text.DecimalFormat;
import java.text.NumberFormat;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Random;
import java.util.Scanner;

/**
 *
 * @author Mario González Rodríguez
 */
public class PWDE {

    // Tamaño de la Poblacion Se recomiendan mas de 30
    // Numero de iteraciones a realizar
    // Cantidad de variables de cada individuo
    static int NUM_ITER = 15, TAM_POBLACION = 30, GEN_ACT = 0, CANT_VAR =
6;
    //Lista de los individuos que habra
    public ArrayList<Individuo> Poblacion, nuevaPoblacion, RealEval;
    //Arreglo para determinar los limites superior e inferior de los
valores
    static double[][] limites;
    //Factor de mutacion y cruzamiento (generalmente es un numero bajo)
    static double F = 0.9, Cr = 0.4;
    static Random rng;
    double prom;
    static int MayorFit, pMF, cAcum, gen;
    //Se guarda el mejor de cada generacion
    static int FitGen, gMF, gCAcum;
    static Scanner sc;
    static final String[] vars = {"PM10", "Temp", "Velocidad V",
"Humedad", "Radiacion S.", "Prec Pluv"};

    public PWDE()
    {

    }

    public ArrayList Proceso(int hora) throws FileNotFoundException,
IOException{
        rng = new Random();
        FileReader file = new FileReader("../Data/" + hora + ".txt");
        sc = new Scanner(file);
        //System.out.println("Introduce la hora");
        crearLimitesPares(sc.nextInt());
    }
}
```

```

        //System.out.println("Introduce el numero de datos a
introducir");
        generarPoblacion(sc.nextInt());
        sc.close();
        prom=0;
        for(int i=0;i<30;i++)
        {
            prom+=RealEval.get(i).getVal();
        }
        prom/=30;
        GEN_ACT = 1;
        //Realiacion del ciclo
        do {
            nuevaPoblacion = new ArrayList<>();
            //iteracion
            for (int i = 0; i < TAM_POBLACION; i++) {
                //Este metodo nos trae 3 numeros aleatorios excluyendo i
                int[] rand = Elegir3Aleat(i);
                //Llamamos ahora a calcularV este recibe los 4
individuos,
                //individuo i, r1, r2 y r3
                Individuo V;
                V = CalcularV(Poblacion.get(i), Poblacion.get(rand[0]),
Poblacion.get(rand[1]),
                Poblacion.get(rand[2]));
                //Se genera el individuo U, quien se agregara al vector
de la nueva poblacion
                Individuo U;
                U = ElegirU(Poblacion.get(i), V);
                nuevaPoblacion.add(U);
            }
            double ft = 100;
            Individuo bst = new Individuo(CANT_VAR);
            double sumfits = 0;
            double sumpm = 0;
            ArrayList<Fitness> POB = new ArrayList();
            for (int xx = 0; xx < TAM_POBLACION; xx++) {
                double[] Fit = EvaluarFitness(Poblacion.get(xx));
                Fitness f = new Fitness(Fit[0], xx, 0);
                POB.add(f);
            }
            for (int s = 0; s < TAM_POBLACION; s++) {
                double[] Fit = EvaluarFitness(nuevaPoblacion.get(s));
                if (Fit[0] < ft) {
                    ft = Fit[0];
                    bst = nuevaPoblacion.get(s);
                }
                Fitness f = new Fitness(Fit[0], s, 1);
                POB.add(f);
                sumfits += Fit[0];
                sumpm += nuevaPoblacion.get(s).getVectPos(0);
            }
            sumfits /= TAM_POBLACION;
            sumpm /= TAM_POBLACION;
            FitGen = 100;
            gCAcum = 0;
            gMF = 0;

```

```

        Poblacion = SeleccionarPob(POB);
        GEN_ACT++;
    } //Evaluacion con respecto a si se encontro la solucion o se
realizaron las iteraciones
    while (GEN_ACT < NUM_ITER);
    //Mostrar(Poblacion);
    //MostrarMejor(best, MayorFit, cAcum, pMF, gen);
    return Poblacion;
}

public ArrayList SeleccionarPob(ArrayList<Fitness> POB) {

    Collections.sort(POB);
    ArrayList<Individuo> Seleccionados = new ArrayList();
    //Creamos arreglo de probabilidad variable que se utilizara
    int prob = TAM_POBLACION * 2;
    int x = prob * (prob + 1) / 2;
    int[] nums = new int[60];
    for(int i=0; i<prob; i++)
    {
        nums[i]=prob-i;
    }
    int[] elegidos = new int[TAM_POBLACION * 2];
    for (int i = 0; i < TAM_POBLACION; i++) {
        //Numero elegido
        int y = rng.nextInt(x);
        int sum = 0, cont = 0;
        int iter=0;
        prob = TAM_POBLACION * 2;
        while (true) {
            sum += nums[iter];
            if (y < sum) {
                if (elegidos[cont] == 0) {
                    if (POB.get(cont).getGen() == 0) {
                        elegidos[cont] = 1;
                        x-=nums[iter];
                        nums[iter]=0;

                    Seleccionados.add(Poblacion.get(POB.get(cont).getPos()));
                } else {
                    elegidos[cont] = 1;
                    x-=nums[iter];
                    nums[iter]=0;

                    Seleccionados.add(nuevaPoblacion.get(POB.get(cont).getPos()));
                }
                break;
            } else {
                i--;
                break;
            }
        }

        iter++;
        prob--;
        cont++;
    }
}

```

```

    }
    return Seleccionados;
}

private void crearLimitesPares(int hora) {
    limites = new double[CANT_VAR][2];
    //Temperatura
    limites[1][0] = 3;
    limites[1][1] = 30;
    //Velocidad del viento
    limites[2][0] = 0.801;
    limites[2][1] = 41.9;
    //Humedad Relativa
    limites[3][0] = 3;
    limites[3][1] = 96;
    //Radiacion Solar
    limites[4][0] = 7;
    limites[4][1] = 11;
    //Precipitacion pluvial
    limites[5][0] = 0;
    limites[5][1] = 18;
    switch (hora) {
        case 0:
            //Limites PM10
            limites[0][0] = 3;
            limites[0][1] = 130;

            break;
        case 1:
            //Limites PM10
            limites[0][0] = 2;
            limites[0][1] = 126;
            break;
        case 2:
            //Limites PM10
            limites[0][0] = 3;
            limites[0][1] = 130;
            break;
        case 3:
            //Limites PM10
            limites[0][0] = 2;
            limites[0][1] = 123;
            break;
        case 4:
            //Limites PM10
            limites[0][0] = 3;
            limites[0][1] = 130;
            break;
        case 5:
            //Limites PM10
            limites[0][0] = 4;
            limites[0][1] = 130;
            break;
        case 6:
            //Limites PM10
            limites[0][0] = 7;
            limites[0][1] = 140;
    }
}

```

```

        break;
case 7:
    //Limites PM10
    limites[0][0] = 6;
    limites[0][1] = 140;
    break;
case 8:
    //Limites PM10
    limites[0][0] = 3;
    limites[0][1] = 140;
    break;
case 9:
    //Limites PM10
    limites[0][0] = 3;
    limites[0][1] = 160;
    break;
case 10:
    //Limites PM10
    limites[0][0] = 4;
    limites[0][1] = 210;
    break;
case 11:
    //Limites PM10
    limites[0][0] = 2;
    limites[0][1] = 210;
    break;
case 12:
    //Limites PM10
    limites[0][0] = 3;
    limites[0][1] = 204;
    break;
case 13:
    //Limites PM10
    limites[0][0] = 3;
    limites[0][1] = 210;
    break;
case 14:
    //Limites PM10
    limites[0][0] = 3;
    limites[0][1] = 210;
    break;
case 15:
    //Limites PM10
    limites[0][0] = 2;
    limites[0][1] = 170;
    break;
case 16:
    //Limites PM10
    limites[0][0] = 5;
    limites[0][1] = 180;
    break;
case 17:
    //Limites PM10
    limites[0][0] = 3;
    limites[0][1] = 140;
    break;
case 18:

```

```

        //Limites PM10
        limites[0][0] = 2;
        limites[0][1] = 130;
        break;
    case 19:
        //Limites PM10
        limites[0][0] = 5;
        limites[0][1] = 150;
        break;
    case 20:
        //Limites PM10
        limites[0][0] = 5;
        limites[0][1] = 160;
        break;
    case 21:
        //Limites PM10
        limites[0][0] = 2;
        limites[0][1] = 140;
        break;
    case 22:
        //Limites PM10
        limites[0][0] = 3;
        limites[0][1] = 140;
        break;
    case 23:
        //Limites PM10
        limites[0][0] = 5;
        limites[0][1] = 148;
        break;
    }
}

//Este metodo inicializa la poblacion con variables aleatorias en los
limites.
private void generarPoblacion() {
    Poblacion = new ArrayList<>();
    for (int i = 0; i < TAM_POBLACION; i++) {
        //Se crea un nuevo individuo
        Individuo ind = new Individuo(CANT_VAR);
        //se generan los valores aleatorios del individuo
        for (int j = 0; j < CANT_VAR; j++) {
            double val = rng.nextDouble() * limites[j][1] +
limites[j][0];
            ind.setVectPos(j, val);
        }
        Poblacion.add(ind);
    }
}

//Este metodo recibe una poblacion de inicio.
public void generarPoblacion(int cant) {
    ArrayList<Individuo> DATOS = new ArrayList<>();
    for (int i = 0; i < cant; i++) {
        Individuo ind = new Individuo(CANT_VAR);
        for (int j = 0; j < CANT_VAR; j++) {
            double val = sc.nextDouble();
            ind.setVectPos(j, val);
        }
    }
}

```

```

        }
        DATOS.add(ind);
    }
    int[] elegidos = new int[cant];
    int[] pobs = new int[TAM_POBLACION];
    int[] test = new int[TAM_POBLACION];
    int x = 0, y = 0;
    for (int i = 0; i < TAM_POBLACION * 2; i++) {
        int eleg = rng.nextInt(cant);
        if (elegidos[eleg] == 0) {
            if (i % 2 == 0) {
                pobs[x] = eleg;
                x++;
            } else {
                test[y] = eleg;
                y++;
            }
            elegidos[eleg] = 1;
        } else {
            i--;
        }
    }
    Poblacion = new ArrayList<>();
    RealEval = new ArrayList<>();
    for (int i = 0; i < TAM_POBLACION; i++) {
        Poblacion.add(DATOS.get(pobs[i]));
        RealEval.add(DATOS.get(test[i]));
    }
    // System.out.println("Poblacion elegida");
    // for (int i = 0; i < TAM_POBLACION; i++) {
    //     System.out.println(Poblacion.get(i).toString());
    // }
    // System.out.println("Poblacion para test");
    // for (int i = 0; i < TAM_POBLACION; i++) {
    //     System.out.println(RealEval.get(i).toString());
    // }
    DATOS.clear();
}

private int[] Elegir3Aleat(int per) {
    int[] aleat = new int[3];
    boolean Fin = false;
    do {
        for (int i = 0; i < 3; i++) {
            aleat[i] = rng.nextInt(TAM_POBLACION);
            if (aleat[i] == per) {
                i--;
            }
        }
        if (aleat[0] != aleat[1] && aleat[0] != aleat[2] && aleat[1]
!= aleat[2]) {
            Fin = true;
        }
    } while (Fin != true);
    return aleat;
}

```



```

//Recibe 4 "Individuos" El primero es el de la iteracion, los otros 3
son los elegidos aleatoriamente
private Individuo CalcularV(Individuo x, Individuo i, Individuo j,
Individuo k) {
    Individuo v = new Individuo(CANT_VAR);
    for (int y = 0; y < 1; y++) {
        //Aplicacion de la formula  $V = R1 + F(R2-R3)$ 
        double Valor = i.getVectPos(y) + (F * (j.getVectPos(y) -
k.getVectPos(y)));
        //Se agrega el resultado al individuo V
        v.setVectPos(y, Valor);
    }
    //Retornamos el Individuo v
    return v;
}

private Individuo ElegirU(Individuo xi, Individuo v) {
    Individuo u = new Individuo(CANT_VAR);
    int j = rng.nextInt(CANT_VAR);
    for (int i = 0; i < 1; i++) {
        double r = rng.nextDouble();
        double elegido;
        //Verificamos que r sea mayor al Cruzamiento y que i sea
diferente de j
        //Recordemos que j es el valor que si o si se tomara del
Vector v
        if (r > Cr && i != j) {
            elegido = xi.getVectPos(i);
        } else {
            //Verificamos si el numero no excede los limites
establecidos
            elegido = v.getVectPos(i);
            if (elegido < limites[i][0] || elegido > limites[i][1]) {
                //Si excede los limites se cambia por un valor
aleatorio
                //entre los limites aceptados
                elegido = rng.nextDouble() * (limites[i][1] -
limites[i][0]) + limites[i][0];
            }
        }
        u.setVectPos(i, elegido);
    }
    return u;
}

public double[] EvaluarFitness(Individuo x) {
    double[] val = new double[6];
    double r = prom;
    double d = x.getVectPos(0);
    double marg = r-d;
    marg = (marg / r) * 100;
    val[0] = Math.abs(marg);
    return val;
}

public void MostrarPoblacion() {
    System.out.println("-----");
}

```

```

        System.out.println("Generacion " + GEN_ACT);
        for (int i = 0; i < TAM_POBLACION; i++) {
            System.out.println("Individuo " + i);
            String pob = "";
            for (int j = 0; j < CANT_VAR; j++) {
                pob += Poblacion.get(i).getVectPos(j) + " ";
            }
            System.out.println(pob);
        }
    }

    public void MostrarMejor(Individuo x, int Fitness, int c, int p, int
gen) {
        System.out.println("-----");
        System.out.println("MEJOR DE LA GENERACION " + gen);
        String pob = "";
        for (int j = 0; j < CANT_VAR; j++) {
            double d = x.getVectPos(j);
            if (d > 0.5) {
                pob += "1 ";
            } else {
                pob += "0 ";
            }
        }
        System.out.println("Ind " + p + " CON " + pob + " con " + Fitness
+ " puntaje de fitness y peso " + c);
        System.out.println("-----");
    }

    public void Mostrar(Individuo x) {
        NumberFormat formatter = new DecimalFormat("#0.00");
        String sout = "";
        for (int j = 0; j < CANT_VAR; j++) {
            sout += formatter.format(x.getVectPos(j)) + " ";
        }
        System.out.println(sout);
    }

    public void Mostrar(ArrayList<Individuo> x) {
        NumberFormat formatter = new DecimalFormat("#0.00");
        for (int i = 0; i < x.size(); i++) {
            String sout = "";
            for (int j = 0; j < CANT_VAR; j++) {
                sout += vars[j] + " " + " " +
formatter.format(x.get(i).getVectPos(j)) + " ";
            }
            System.out.println(sout);
        }
    }
}
}

```

Clase CalcularPromedioMovil

```
/**
 *
 * @author Usuario
 */
public class CalculadorPromedioMov {

    Double CProm;//Concentracion Promedio
    Double FactPond;//Factor de Ponderacion;
    Double w; //Valor del peso
    int Cmax = 0, Cmin = 0; //Concentracion promedio minima y maxima

    public int CalcularPromedio(int[] C) {
        //Buscar el mayor y el menor
        Cmin = C[0];
        for (int i = 0; i < 12; i++) {
            if (C[i] > Cmax) {
                Cmax = C[i];
            }
            if (C[i] < Cmin && C[i] != 0) {
                Cmin = C[i];
            }
        }
        System.out.println("Cmin: " + Cmin + " Cmax: " + Cmax);
        //Calculo del Factor de Ponderacion
        double x = Cmax - Cmin;
        double x2 = x / Cmax;
        w = (double) 1 - x2;
        System.out.println("Factor de ponderacion: " + w);
        //Falta ajustar a 2 decimales.
        //Revisar si w es mayor a 0.5, en caso contrario cambiar el valor
por 0.5 para el factor de ponderacion
        FactPond = (w > 0.5) ? w : 0.5;
        double numerador = 0, denominador = 0;
        //Realizar Sumatoria
        for (int i = 0; i < 12; i++) {
            //En caso de que no se cuente con el promedio de una hora se
omite en el calculo.
            if (C[i] != 0) {
                double pot = Math.pow(FactPond, i);
                numerador += C[i] * pot;
                denominador += pot;
            }
        }
        //Se obtiene el promedio (debera redondearse)
        CProm = numerador / denominador;
        System.out.println("Resultado sin redondeo; " + CProm);
        return CProm.intValue();
    }

    public CalculadorPromedioMov() {
    }

}
```

Clase Main

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.Scanner;

/**
 *
 * @author Mario Gonzalez
 */
public class Main {

    Double CProm;//Concentracion Promedio

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) throws IOException {
        Scanner sc = new Scanner(System.in);
        System.out.println("¿Que hora es? "
            + "\nEJ: 1:00 = 1, 2:40 = 2");
        int hora = sc.nextInt();
        System.out.println("Introduzca los valores correspondientes a las
12 horas previas");
        int[] HorasPrevias = new int[12];
        //Tomamos las 12 horas previas para calcular el promedio movil
actual
        for (int i = 11; i >= 0; i--) {
            HorasPrevias[i] = sc.nextInt();
        }
        System.out.println("Calculando...");
        PWDE Evol = new PWDE();
        double[][] limites = new double[24][2];
        int[] prom = new int[24];
        for (int i = 0; i < 24; i++) {
            Double sum = 0.0;
            for (int w = 0; w < 10; w++) {
                ArrayList<Individuo> X = Evol.Proceso(i);
                limites[i][0]=X.get(0).getVectPos(0);
                for (int j = 0; j < 30; j++) {
                    double d = X.get(j).getVectPos(0);
                    sum += d;
                    if(d>limites[i][1])
                    {
                        limites[i][1]=d;
                    }
                    if(d<limites[i][0])
                    {
                        limites[i][0]=d;
                    }
                }
            }
        }
    }
}
```

```

    }
    sum /= 300;

    prom[i] = sum.intValue();
}
CalculadorPromedioMov CPM = new CalculadorPromedioMov();
int Prom = CPM.CalcularPromedio(HorasPrevias);
System.out.println("Concentracion promedio movil a 12 horas : " +
Prom);
int[] dif = new int[24];
dif[0] = (prom[0] - prom[23]);
//System.out.println("Hora 0 " + "Promedio: " + prom[0] + "
diferencia " + dif[0]);
for (int i = 1; i < 24; i++) {
    dif[i] = (prom[i] - prom[i - 1]);
    //System.out.println("Hora " + i + " Promedio: " + prom[i] + "
diferencia " + dif[i]);
}
int difHora = 0;
for (int s = 0; s < 12; s++) {
    double d = prom[(s + hora) % 24];
    difHora += (int) Math.abs(HorasPrevias[11 - s] - d);
}
System.out.println(difHora);
//Obtenemos el promedio de diferencias
difHora /= 12;
//Dividimos entre 4 para reducir el rango de estimacion
System.out.println("Introduce los resultados esperados");
int[] esperados = new int[12];
for (int i = 0; i < 12; i++) {
    esperados[i] = sc.nextInt();
}
double aciertoProm = 0;
for (int s = 0; s < 12; s++) {
    Prom += dif[(s + hora) % 24];
    int rango = (s+1)*2;
    int min = Prom-rango;
    int max = Prom+rango;
    System.out.println("Estimacion : " + Prom + " Esperado " +
esperados[s]);
    System.out.println("Aproximado: Min:" + min + " Max:" + max);
    double error = 0;
    if (esperados[s] >= min && esperados[s] <= max) {
        System.out.println("Dentro del rango esperado");
    } else {
        double marg = esperados[s] - Prom;
        marg = (marg / esperados[s]) * 100;
        error = Math.abs(marg);
        aciertoProm += error;
        System.out.println("Margen de error " + error);
    }
}
aciertoProm /= 12;
System.out.println("-----");
System.out.println("Promedio de error" + aciertoProm);
}
}

```