Instituto Politécnico Nacional

Escuela Superior de Cómputo

Trabajo Terminal
16-2-0009

Análisis comparativo del desempeño de Técnicas Evolutivas aplicadas a la predicción de distribución de robos

Trabajo para cumplir con la opción de titulación curricular en la carrera de:

"Ingeniero en Sistemas Computacionales"

Presenta:

Macías Duarte Carlos Antonio *2011630173*

Directores

M. en C. José David Ortega Pacheco
M. en C. Edgar Armando Catalán Salgado

Ciudad de México a 8 de junio de 2016

Instituto Politécnico Nacional
Escuela Superior de Cómputo
Subdirección Académica

No. de TT: 16-2-0009 Fecha de Presentación: 3 de Junio del 2016

Documento Técnico
Serie Amarilla

Análisis comparativo del desempeño de Técnicas Evolutivas aplicadas a la predicción de distribución de robos

Presenta:

Macías Duarte Carlos Antonio[1] *2011630173*

Directores

M. en C. José David Ortega Pacheco M. en C. Edgar Armando Catalán Salgado

Resumen

En este reporte se presenta la motivación, desarrollo y resultados finales del presente trabajo, donde se propone el uso de algoritmos de Cómputo Evolutivo en conjunto con la Prueba χ^2 de Pearson para la tarea de predicción de distribuciones de robo en una determinada zona basado en información previa de estos y donde se realizo también la comparación del desempeño de diversas técnicas evolutivas en esta tarea.

Palabras Clave: Cómputo Evolutivo, Prueba χ^2 de Pearsont Predicción de Probabilidades, Robos, Análisis comparativo

[1] camaciasd@gmail.com

Escuela Superior de Cómputo
Subdirección Académica
Departamento de Formación Integral E Institucional
Comisión Académica de Trabajo Terminal

México, Ciudad de México, a 7 de Junio del 2016

Dr. Flavio Arturo Sánchez Garfias
Presidente de la Comisión Académica de Trabajo Terminal
Presente

Por medio del presente, se informa que el alumno que integran el **TRABAJO TERMINAL**: 16-2-0009, titulado Änálisis comparativo del desempeño de Técnicas Evolutivas aplicadas a la predicción de distribución de robosc̈oncluyo satisfactoriamente su trabajo.

Los discos (DVDs) fueron revisados ampliamente por sus servidores y corregidos, cubriendo el alcance y el objetivo planteados en el protocolo original y de acuerdo a los requisitos establecidos por la Comisión que Usted preside.

Atentamente

M. en C. José David Ortega Pacheco

M. en C. Edgar Armando Catalán Salgado

Advertencia

"Este documento contiene información desarrollada por la Escuela Superior de Cómputo del Instituto Politécnico Nacional, a partir de datos y documentos con derecho de propiedad y por lo tanto, su uso quedará restringido a las aplicaciones que explícitamente se convengan."

La aplicación no convenida exime a la escuela su responsabilidad técnica y da lugar a las consecuencias legales que para tal efecto se determinen.

Información adicional sobre este reporte técnico podrá obtenerse en:

La Subdirección Académica de la Escuela Superior de Cómputo del Instituto Politécnico Nacional, situada en Av. Juan de Dios Bátiz s/n Teléfono: 57296000, extensión 52000.

Índice general

Índice de figuras

Índice de tablas

Índice de definiciones

Índice de algoritmos

Índice de código

Capítulo 1

Introducción

La tarea de predecir el resultado de fenómeno es algo común de encontrar en muchas áreas de utilidad para el ser humano, por ejemplo la predicción del clima a partir de factores ambientales, la predicción de enfermedades a partir de síntomas en un paciente o la predicción de movimientos en el mercado según datos financieros actuales son solo algunas de ellas. Existen diferentes herramientas computacionales como las Redes Neuronales Artificiales o las Maquinas de Soporte Vectorial que son utilizada para realizar la tarea de predicción.

Un área donde la predicción es muy útil es en la probabilidad de que ocurra un robo en alguna zona especifica. El saber esta información es de utilidad para planificar rutas de patrullaje, planes de seguridad, rutas de transporte de bienes, etc.

Actualmente no existen propuestas para hacer uso de técnicas de Cómputo Evolutivo para realizar esta tarea de predicción, en el presente trabajo se presenta una propuesta de uso de estas técnicas para dicha tarea en el caso de predicción de probabilidad de robos y se realizara un análisis comparativo entre el desempeño de estas

1.1. Problemática

Actualmente en México existen problemas en materia de la seguridad de la población, diferentes tipos de crímenes como robos, asesinatos, extorsiones entre otros ocurren casi diariamente en alguna parte del país y ponen en peligro la integridad de personas y negocios.

Uno de los crímenes más comunes es el robo en diferentes modalidades, el poder tener un idea de la probabilidad de que ocurra un robo en un determinado punto o zona podría ser de gran utilidad para mejorar la seguridad en dichos lugares y para que las personas que transitan o habitan en estas usen esta información para su seguridad propia.

1.2. Trabajo Previo

En esta sección se presentan una serie de trabajos que sirven como antecedentes para este divididos en 2 categorías:

Es la ESCOM-IPN no se ha hecho previamente trabajo sobre el uso de algoritmos evolutivos para la tarea de predicción, por lo que la Tabla 1.1 presente una lista de Trabajos Terminales donde se demuestra diferentes usos para algoritmos evolutivos.

Tabla 1.1: Trabajos Terminales

TT	Titulo	Alumnos	Directores
2006-0047	Videojuego de estratega con mundo persistente en linea e interfaz 3D, empleando cómputo evolutivo y algoritmos geneticos	David Carrillo Cisneros, Juan Manuel Luna Pérez, Eduardo José Maeda Cervera	Martha Rosa Cordero López, Marco Antonio Dorantes Gonzales
2014-A049	Sistema de defensa de objetivo móvil basado en Cómputo Evolutivo	Nieto Romero Xanat Alejandra	Mario Augusto Ramírez Morales, Jesús Alfredo Ramírez Nuño
2014-B094	Algoritmo Evolutivo para la optimización de portafolios de inversión	Alan Gustavo Plata Godínez, Héctor Rodríguez Gónzalez, Jorge Emyr Vázquez Juárez	Miriam Pescador Rojas, Mario Augusto Ramírez Morales
2014-A019	Evolución Diferencial y Algoritmo de Enjambre de Partículas. Para el cálculo de Testores Típicos.	Borja Cazales David, Díaz García Sergio Adonais	Olga Kolesnikova, Salvador Godoy Calderón
15-2-007	Selección de caractéristicas discriminantes en patrones médicos mediante Modelos Asociativos y Algoritmoe Genéticos	Macías Peñaloza Ana Rosa	Miriam Pescarode Rojas, Mario Aldape Pérez

La tabla 1.2 presenta trabajos de investigación donde se realizo la tarea de predicción en algún área, en el trabajo [1] se puede ver el uso de los Algoritmos Evolutivos para la predicción de distribuciones de robos para poder realizar simulación de movimientos de patrullas.

Tabla 1.2: Trabajos de Investigación sobre predicción

Titulo	Autores	Resumen
A bio-inspired crime simulation model[1]	Vasco Furtadoa, Adriano Meloa, André L.V. Coelhoa, Ronaldo Menezesb, Ricardo Perrone	Este trabajo presenta una simulación de crímenes basado en un sistema multiagente, para determinar las probabilidades de que se dé un robo en una punto de la zona donde se realiza la simulación se hizo uso de Algoritmos Genéticos con la Prueba χ^2 de Pearson como función de aptitud para la determinación de la probabilidad de que un robo se dé basado en información de robos anteriores.
Application of neural networks on multivariate time series modeling and prediction[2]	Min Han, Mingming Fan	En este trabajo se hace uso de redes neuronales artificiales y el principio de análisis de componentes para modelado y predicción de series de tiempo multivariable.
Nonlinear prediction of chaotic time series using support vector machines[3]	S. Mukherjee, E. Osuna, F. Girosi	En este trabajo se realiza un análisis de la variabilidad en el desempeño de los diferentes parámetros de una Maquina de Soporte Vectorial para el estudio de unas series de tiempo particulares.
Prediction in dynamic system - a divide and conquer approach[4]	G. Chakraborty, H. Watanabe, B. Chakraborty	Este trabajo propuso una manera general de atacar problemas de toma de decisiones en espacios complejos de información dinámica y gran número de variables. La propuesta se basa en el cambio de la información de un espacio continuo a uno discreto de la información para crear subgrupos de datos que después serán utilizados para entrenar redes neuronales para la toma de decisiones.
Application of dynamic liquid level prediction model based on improved SVR in sucker rod pump oil Wells[5]	Hou Yanbin, Gao Xianwen, Wang Mingshun, Li Xiangyu, Liu Yu, Wu Bing	Propone la combinación de Regresión de Soporte Vectorial hacienda uso de algoritmos genéticos en la optimización sus parámetros para la predicción del nivel de líquido en pozos de bombeo en campos petroleros.

1.3. Solución Propuesta

La solución propuesta a la problemática planteada en 1.1 es el realizar la selección algunas técnicas evolutivas diferentes al Algoritmo Genético utilizado en [1] y utilizarlas para realizar la tarea de predicción de distribuciones de robos y realizar un análisis comparativo de los resultados arrojados de la implementación de cada una.

1.4. Objetivos

El objetivo del presente trabajo es realizar un análisis comparativo de la efectividad de dos o más algoritmos evolutivos para la predicción de zonas de riesgo relacionadas con robos.

1.4.1. Objetivos Particulares

Los objetivos particulares del trabajo son:

- Recopilar un banco de datos de robos de una o varias zonas.
- Elegir los algoritmos evolutivos para el estudio comparativo.
- Implementación de los algoritmos evolutivos para problemas de minimización
- Implementar el modelo de predicción mediante una función de distribución.
- Realizar pruebas de los algoritmos evolutivos con la función de distribución en la tarea de predicción
- Reportar los resultados experimentales con la comparación de los algoritmos evolutivos.

1.5. Justificación

La predicción de la frecuencia en la que pueden darse robos en ciertos puntos de un área determinada es de gran utilidad para tareas de seguridad como por ejemplo programación de patrullajes e investigación de delitos, además de que es de utilidad para la comunidad de esta zona tener este conocimiento para que la gente tenga en cuenta esto durante sus rutinas diarias.

La diverisdad de formas en que se puede repartir la probabilidad de que ocurra un robo en diferentes puntos de una determinada zona crea un amplio espacio de búsqueda para determinar la o las mejores distribuciones en cuanto a la posibilidad de que se den estos, por lo que el uso de técnicas heurísticas para la búsqueda de soluciones es la herramienta adecuada para encontrar la mejor solución a este problema [6], pues realizan su trabajo sin necesidad de explorar todo el espacio de búsqueda lo que ahorra el uso de tiempo y recursos de cómputo y en su ejecución y aunque no garantizan el encontrar la mejor solución si no una de las más cercanas, en este caso no es de relevancia el encontrar la mejor distribución ya que lo que se busca es la predicción de un escenario que puede o no darse así con exactitud.

Los algoritmos evolutivos son técnicas de búsqueda heurística que han sido probados en diferentes tareas con un alto rango de eficiencia y con una implementación relativamente sencilla [7], por lo que el uso de estos para realizar la búsqueda de la mejor distribución de frecuencias de robos en una zona determinada es una decisión acertada.

Existen diferentes tipos de algoritmos evolutivos que trabajan de manera diferente, y no existe una forma de comparar su funcionamiento en general más que en la aplicación de estos a determinado problema, es por eso que el uso

de más de una técnica evolutiva para solucionar esta problemática es de gran utilidad para poder comparar el comportamiento y la eficiencia de estos para elegir el mejor para realizar esta tarea.

El presente trabajo ofrece información útil para la predicción de la distribución de robos en un futuro en una zona haciendo uso de algoritmos evolutivos para determinar esta.

Capítulo 2

Cómputo Evolutivo

El CE rápidamente se ha convertido en la mayor área de estudio para el Aprendizaje Máquina y los Sistemas de Optimización y recientemente ha entrado en el área de diseño de hardware[8].

Se puede definir el campo del CE como:

Definición 1 (*Cómputo Evolutivo*)

> *Disciplina del enfoque sub-simbólico o Bottom-Up de la IA, compuesta por un conjunto de técnicas heurísticas que imita la evolución y otros mecanismos observados en la naturaleza para la resolución de problemas intratables por otras técnicas[9].*

Defiriendo un poco con la definición 1, las técnicas del CE también pueden ser utilizadas para resolver problemas que son fácilmente tratables con otros tipos de técnicas, pero debido a que éstas ya resuelven dichos problemas en tiempo razonable, no vale la pena el aplicar CE a la solución de los mismos, es más, resulta en muchos casos contraproducente. De ahí puede observarse, que como todo, el CE tiene ventajas y desventajas dependiendo del problema al que se aplique.

Ahora se hablara de lo que es una heurística.

La palabra heurística viene del griego *euriskein*, y significa *encontrar*, esta palabra se volvió popular por el libro *How to solve it*[10] de George Pólya.

Se dice que una técnica es heurística si es capaz de encontrar en tiempo razonable y sin hacer uso de muchos recursos soluciones casi óptimas o muy cercanas a la óptima a un problema intratable[6].

Ésta es una de las características principales del CE, todas sus técnicas están pensadas para resolver problemas irresolubles y en poco tiempo, lo que las hace técnicas heurísticas.

Definición 2 (*Técnica Heurística*)

> *Es aquella que es capaz de resolver en un tiempo razonable y con pocos recursos computacionales un problema intratable en tiempo polinomial por otras técnicas, llegando a soluciones muy cercanas a las óptimas.*

Se puede ver como en las definiciones 1 y 2 que todas las técnicas del CE son heurísticas, pero no todas las técnicas heurísticas pertenecen al CE.

Otro punto importante que hay que aclarar, es el termino Metaheurística, el prefijo *meta* significa *más allá*, por lo que se puede entender que una Metaheurística es una técnica heurística que busca maneras más grandes de abarcar su cometido, aunque para fines prácticos, se toman como sinónimos estas dos palabras.

2.1. Historia del CE

La evolución fue vista como un método de aprendizaje primero por Cannon[11] en los años 30 y unos años después Turing[12] reconoció una relación obvia entre el aprendizaje y la evolución.

En la década de 1960 Campbell conjeturó que en todos los procesos que llevan a la expansión del conocimiento siempre hay una etapa que involucra un proceso ciego de variación y supervivencia selectiva.

Probablemente el primer intento serio de aplicar la teoría de la evolución para la solución de una problemática fue realizado por Box y sus colegas en 1957, ellos desarrollaron una técnica que denominaron operación evolutiva, la cual se aplicó a una planta de manufactura para manejarla, y que se implantó sobre la base de los votos de un comité de jefes técnicos. Bajo este esquema, la planta se veía como a una especie en evolución. La calidad del producto avanzaba a través de mutaciones aleatorias y la selección era determinada por el comité[13, 14].

La tabla 2.1 menciona los trabajos que dieron origen al área del CE y sus aportaciones al campo.

Tabla 2.1: Trabajos Pioneros del CE

Autor	Año	Resumen
George E. P. Box[15]	1957	Primer intento serio de aplicar la evolución en la resolución de un problema. llegó a establecer claramente la analogía entre estos cambios y las mutaciones que ocurren en la naturaleza, e hizo ver también que el proceso de ajuste de parámetros que efectuaba con técnicas estadísticas era similar al proceso de selección natural.
R. M. Friedberg [16]	1958	Primer intento de crear un programa que se mejorara a sí mismo.
Hans Joachim Bremermann [17, 18, 19, 20]	1958	Primero en ver la evolución como un proceso de optimización e introducir el concepto de aptitud. Define a un individuo como una cadena binaria. Primero en definir un método de reproducción, selección y mutación e indica la importancia de la última para evitar caer en óptimos locales. Fue de los primeros en usar el termino población e intuir la coevolución.
Nils Aall Barricelli [21, 22]	1954	Ofreció una de las primeras simulaciones que usaban principios evolutivos. El principal énfasis de su investigación consistía en determinar las condiciones que los genes deben cumplir para poder dar pie al desarrollo de formas de vida más avanzadas. Sus conclusiones fueron que los genes deben satisfacer lo siguiente: Una cierta capacidad para reproducirse, una cierta capacidad para cambiar a formas alternas (a través de mutaciones) y una necesidad de simbiosis (por ejemplo, a través de vida parásita) con otros genes u organismos.
J. Reed y R. Toombs [23]	1967	Concluyó que las cruza no parecía mejorar la velocidad de la adaptación selectiva, y la mutación era el principal operador para esto.
Lawrence J. Fogel [24]	1966	Introduce la primera técnica evolutiva que funciona más o menos, consistía en evolucionar autómatas de estado finito. Introdujo con ésto los conceptos de población y selección.
Peter Bienert, Ingo Rechenberg y Hans-Paul Schwefel	1965	Desarrollaron una técnica de optimización de funciones continuas, modificando un vector de números reales mediante operadores probabilísticos usando ciertos criterios para dirigir la búsqueda. Estableciendo la mutación como operador primario y en versiones recientes como operador secundario la cruza.

2.2. Bases del CE

Como se mencionó en la definición 1 el CE está compuesto por una gran variedad de técnicas heurísticas, algunas de las más importantes se puede ver en la figura 2.1.

Figura 2.1: Clasificación de los Algoritmos Evolutivos

Todos los algoritmos evolutivos tienen características similares, se basan en evolucionar una **población**(Def. 8) de **individuos**(Def. 3) mediante un proceso donde se aplican varias operadores genéticas (Def. 14). Los procesos dependen de la **aptitud**(Def. 10) que muestran en el **ambiente**(Def. 9) donde se desarrollan[25].

Antes de continuar, se tiene que aclarar la diferencia entre la evolución natural y la evolución artificial. Mientras que la primera no tiene una meta fija y por tanto está abierta a un proceso de adaptación ilimitado, la segunda sí tiene establecida desde un principio la búsqueda de una solución a un problema predefinido.

Es por esto que la evolución artificial jamás tendrá la misma capacidad en cuanto a diversidad y creatividad generada en la evolución natural, ya que por definición éstos deben atender la tarea de solucionar un problema específico[8].

En la naturaleza un individuo es un ser viviente, cuyo objetivo es sobrevivir y su existencia no tiene ningún propósito general. En el CE esto es diferente.

En la evolución artificial, un individuo tiene una razón de ser, ya que el objetivo de ésta es la resolución de un problema. Como se plantea, el objetivo de los individuos es resolver problemas, podemos por tanto decir que:

Definición 3 (*Individuo*)

> *Posible solución a un problema que se está tratando.*

Ahora bien, como se sabe, en la naturaleza todos los individuos poseen un código genético que los distingue unos de otros, se llama genoma o cromosoma en la naturaleza a la combinación de genes que definen a un individuo. El cromosoma de un individuo está formado por los genes que determinan sus características, igualmente en CE, el cromosoma es la representación de una posible solución, es decir, es un vector de genes.

Definición 4 (*Cromosoma*)

> *Representación de un* individuo *formada por un conjunto de* genes.

Ahora para definir el cromosoma de un individuo, se debe definir qué es lo que compone su cromosoma, las definiciones 5, 6 y 7 brindarán un panorama completo para comprender qué es un cromosoma.

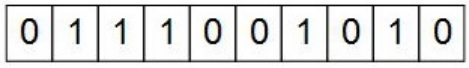

Figura 2.2: Ejemplo de Cromosoma con Genotipo Binario

Definición 5 (*Gen*)

Característica de un individuo, cuyo dominio esta definido por los alelos *del dominio de éste.*

Definición 6 (*Alelo*)

Es un valor posible que puede ser tomado por un gen, y está limitado por el dominio de valores de dicho gen y por el genotipo del cromosoma.

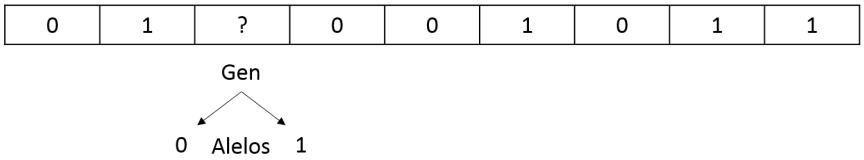

Figura 2.3: Alelos para un Gen de cromosoma Binario

Definición 7 (*Genotipo*)

Es la codificación utilizada para representar un cromosoma.

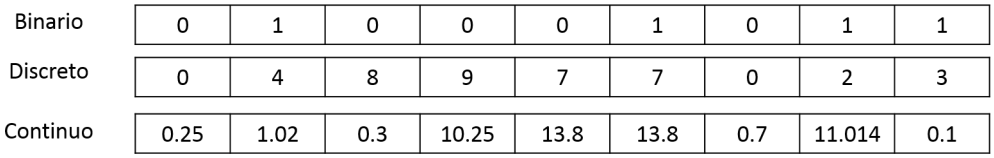

Figura 2.4: Ejemplos de Cromosomas con diferente Genotipo

Con las definiciones de Gen (Def. 5), Alelo (Def. 6) y Genotipo (Def. 7) queda completo el concepto de individuo, que en resumen es una posible solución a un problema, donde su representación es llamada cromosoma, está formada por genes y la forma en que está codificado se le llama genotipo.

Ahora, en la naturaleza no existe un único individuo solamente, existen varios individuos similares, que comparten características generales entre ellos pero a la vez son únicos entre si, a estos conjuntos de individuos se les llamara población.

Definición 8 (*Población*)

> *Conjunto de* individuos *que se desarrollan en el mismo* ambiente.

En esta definición de Población se encuentra también con el concepto de ambiente, en la naturaleza el ambiente donde se desarrolla una población se conforma de las características geográficas, climatologías y demográficas que lo rodean mientras que en el CE el ambiente se define como el problema que se va a atacar, con la analogía de que un individuo debe estar adaptado para sobrevivir en su ambiente, por lo que una solución bien adaptada debe ser capaz de resolver el problema lo mejor posible.

Definición 9 (*Ambiente*)

> *Problema que se intenta resolver.*

La eficiencia con la que un individuo se acerca a ser la mejor solución al problema en el que se desarrolla, se le llama aptitud del individuo, y es un indicador numérico que nos dice si un individuo es bueno o malo para resolver el problema que ataca. Este valor se obtiene con una Función de Aptitud, que es una operación que nos indica cuál es la aptitud de un individuo.

Definición 10 (*Aptitud*)

> *Valor numérico que indica qué tan apto es un* individuo *para ser una solución apropiada o no.*

Definición 11 (*Función de Aptitud*)

> *Aquella que determina la* aptitud *de un* individuo.

Algunas veces las funciones de aptitud trabajan en el dominio del genotipo del individuo, otras sobre su fenotipo.

La diferencia entre ambos es que el primero es una forma de representar las características del individuo, y la segunda es la característica en si.

Por ejemplo, en la naturaleza del código genético de un ser vivo esta formado de combinaciones de diferentes de azúcares para representar sus características, por ejemplo, una combinación de diferentes azúcares da a una persona la propiedad de tener ojos de un determinado color como el azul mientras que unas combinación diferente podría darle un color distinto como el verde, el genotipo sería las combinaciones de azúcares que guardan dichas características y el fenotipo es el color en sí mismo.

Las figuras 2.6 y 2.5 dan un ejemplo de ésto.

Definición 12 (*Fenotipo*)

Decodificación del cromosoma.

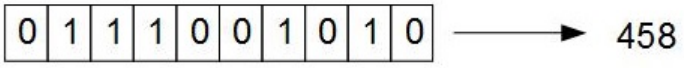

Figura 2.5: Decodificación de un cromosoma (Genotipo a Fenotipo)

Figura 2.6: Codificación de un cromosoma (Fenotipo a Genotipo)

Hasta este punto se ha especificado de manera detallada cómo se utiliza el comportamiento de los seres vivos para solucionar problemas, sólo falta lo relacionado con el proceso de evolución de los mismos.

La evolución de las especies implica un proceso que da como resultado individuos mejor adaptados a las características de su ambiente para tener mayor oportunidad de sobrevivir. Para esto los diferentes individuos de una misma población combinan sus características entre ellos, y el proceso de evolución determina cuáles son mejores para pasar a la nueva población generada. A cada población diferente se le llama generación.

Definición 13 (*Generación*)

Población *generada por la aplicación de* operadores genéticos *en una población previa que sustituyó a esta.*

Los operadores genéticos son los medios por los cuales se realiza esta recombinación de características, esto se da por la cruza de individuos, donde

se heredan características entre individuos de una generación para pasar a la siguiente, y la mutación, donde se heredan ciertas características con cambios aleatorios en ellas.

Definición 14 (*Operadores Genéticos*)

Operador que recibe los cromosomas *de un conjunto de* individuos *para generar nuevos.*

Definición 15 (*Cruza*)

Operador Genético *que genera un nuevo* individuo *a partir de la combinación de genes de dos o más.*

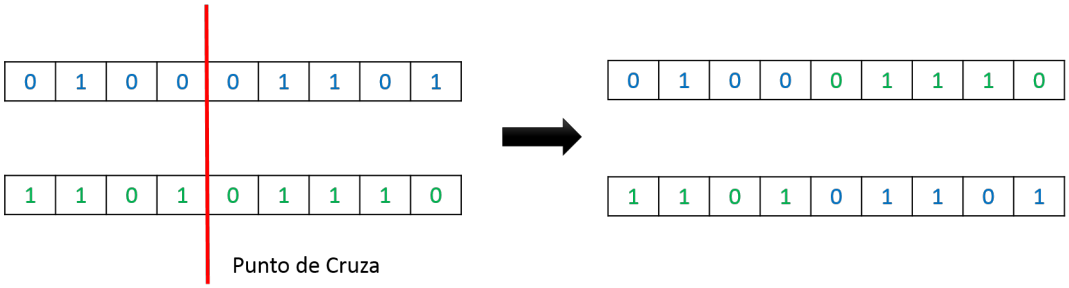

Figura 2.7: Ejemplo de Cruza

Definición 16 (*Mutación*)

Operador Genético *que genera un nuevo* individuo *a partir de cambios aleatorios y/o controlados en los* genes *del* cromosoma *de otro* individuo

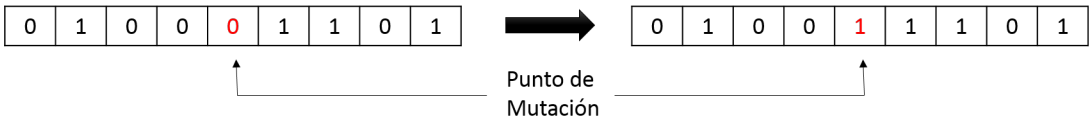

Figura 2.8: Ejemplo de Mutación

Ahora, no todos los nuevos individuos generados sobreviven, en la naturaleza los mejores adaptados son los que lo logran, aquí se habla una vez más de la aptitud(Def. 10) de los individuos, que se utiliza para seleccionar quiénes formarán parte de la nueva generación y quiénes no.

Definición 17 (*Selección*)

Proceso mediante el cual, un conjunto de individuos *de una* generación *son escogidos para aplicarles* operadores genéticos *y/o sean parte de la siguiente* generación

Con las definiciones Generación (Def. 13), Operadores Genéticos (Def. 14) y Selección (Def. 17) se completan los términos faltantes para definir el proceso de evolución.

Básicamente la evolución artificial consiste en crear una población inicial como primera generación de soluciones a un problema de manera aleatoria o controlada, y a partir de ésta crear nuevas generaciones aplicando los operadores genéticos sobre las mismas hasta llegar a una generación donde existan individuos cuya aptitud sea suficiente para que se les considere soluciones al problema planteado.

El algoritmo 1 muestra la estructura básica de todos los algoritmos evolutivos, si bien ésta varía de uno a otro y aplican diferentes formas los operadores genéticos en general la mayoría trabaja así.

Algoritmo 1: Estructura General de un Algoritmo Evolutivo

Entrada:
g: Número de Generaciones
a: Aptitud Objetivo
Salida:
s : Mejor Solución

```
1  p ← inicializaPoblacion() : Generar Población inicial;
2  c ← 1 : Inicializar Contador de Población;
3  t ← aleatorio(0, 1);
4  mientras (c ≤ g) ∨ (a × t ≤ Ma ≤ a) hacer
5      pP ← seleccionarPadres(p);
6      pT ← operacionesGeneticas(pP);
7      p ← seleccionarNuevaPoblación(p, pT);
8      Ma ← mejorAptitud(p);
9      c ← c + 1;
10 s ← mejorIndividuo(p);
11 devolver s;
```

2.3. Aplicaciones del CE

A continuación se mencionan las posibles aplicaciones del CE[26]:

Planeación Por ejemplo, en la tarea de diseño de rutas, las técnicas de CE son muy útiles para tratar problemas similares al clásico Problema del Agente Viajero, que consiste en la búsqueda de rutas más cortas o que conlleven un menor costo el recorrerse. Otro ejemplo para problemas de planeación sería la programación de actividades, el CE puede ser utilizado para la planificación de tareas en un periodo determinado de tiempo.

Diseño Para tareas de diseño, el CE es muy útil a la hora de diseñar sistemas eléctricos y digitales así como la optimización en el diseño de circuitos integrados. También es usado para el diseño de topologías de Redes Neuronales.

Simulación e Identificación En el trabajo de simulación, se ha hecho uso del CE para el modelado de sistemas y procesos observados en la naturaleza y también en la industria para la verificación de procesos, datos de experimentación y optimazación. Al contrario de la simulación, la identificación, el CE se ha utilizado para poder identificar el funcionamiento de algunos sistemas y fenómenos.

Control Se aprovecha el CE para el diseño de controladores para sistemas y también para la participación activa de estas técnicas en procesos de control. Ésto es principalmente en sistemas dinámicos ya que se ajustan a la naturaleza de los algoritmos evolutivos.

Clasificación La tarea de clasificación es una de las funciones más comunes en sistemas expertos, el CE es muy utilizado junto con el Reconocimiento de Patrones y las Redes Neuronales para dar solución a problemáticas de este tipo siempre trabajando en conjunto.

2.4. Optimización de Problemas Multiobjetivo

En las tareas diarias es común que se intenten optimizar diversos aspectos de un mismo problema a la vez, por lo que una solución debe satisfacer lo necesario para resolver cada uno de estos aspectos lo más posibles sin concentrarse solo en uno de estos, ya que esto podría causar ineficiencias en el resto de los aspectos del mismo y no seria una solución optima, este tipo problemas con conocidos como problemas multiobjetivo.

Para estos casos en 1881 Francis Ysidro Edgeworth dio una propuesta para la aplicación de óptimo global a estos problemas, en 1896 fue generalizada por

Vilfredo Pareto y se le denomino a este concepto como *Optimo de Pareto* y su definición puede observarse en 18.

Definición 18 (*Optimo de Pareto*)

De un conjunto de posibles solución existe una que es para optimizar n funciones simultáneamente, es igual o mejor en cada una de estas que cualquier otra solución y es estrictamente mejor al menos en una.

Matemáticamente se puede expresar como que una solución $y \in X$ para un conjunto de n funciones f_i tal que para el caso de minimización se puede observar en la ecuación 2.1 y en el caso de maximización en la ecuación 2.2.

$$\begin{array}{ll} f_i(y) \leq f_i(x) & \forall\, 1 \leq i \leq n \\ \wedge & \\ \exists\, 1 \leq j \leq n & \ni f_j(y) < f_j(x) \end{array} \tag{2.1}$$

$$\begin{array}{ll} f_i(y) \geq f_i(x) & \forall\, 1 \leq i \leq n \\ \wedge & \\ \exists\, 1 \leq j \leq n & \ni f_j(y) > f_j(x) \end{array} \tag{2.2}$$

En el contexto de optimización multiobjetivo se dice que una solución u domina a una solución v si cumple con alguna de las ecuaciones 2.1 o 2.2 dependiendo del caso y se denoto como se ve en 2.3.

$$u \preceq v \tag{2.3}$$

Entonces se dice que el o los óptimos de Pareto de una problema son todas las soluciones no dominadas por ninguna otra.

Capítulo 3

Prueba χ^2 de Pearson

La prueba χ^2 de Pearson fue investigada por primera vez en 1900 por Karl Pearson[27], esta prueba es utilizada para verificar la validez entre un hipótesis propuesta contra los resultados observados en un experimento o en algún fenómeno sucedido[28, 29, 30].

La prueba consiste en que se tienen un conjunto de k resultados posibles para un mismo experimento o fenómeno, se propone una hipótesis sobre la probabilidad de que cada uno de los posibles resultados suceda mediante un conjunto H de probabilidades hipotéticas p_i de ocurrencia para cada resultado i con $1 \leq i \leq k$.

De un conjunto de n ejecuciones del experimento o fenómeno donde se observa que cada resultado i ocurrió O_i veces ($\Sigma_{i=1}^{k} O_i = n$), a lo que se le llama **Ocurrencia observada de** i, se comparan estas contra el número de resultados esperados para cada resultado i dada su probabilidad p_1 en la hipótesis H que se denota como $E_i = n * p_i$ y se le conoce como **Ocurrencia esperada de** i, esta comparación se hace por medio de la ecuación 3.1.

$$\chi^2 = \Sigma_{i=1}^{k} \frac{(O_i - E_i)^2}{E_i} \tag{3.1}$$

El criterio de aceptación de la hipótesis H esta dado por la ecuación 3.2 donde t son los grados de libertad de la distribución χ^2 y es igual a $k - 1$.

$$\chi^2 < \chi_t^2 * t \tag{3.2}$$

El valor de χ^2_t se obtiene de la tabla de distribuciones de χ^2 (Ver tablas A.1 y A.2 en el Anexo A).

Pueden observarse las siguientes propiedades en la ecuación 3.1:

- Los valores de las ocurrencia observada y esperada son positivos, es decir $O_i > 0$ y $E_i > 0$.
- Por lo establecido en el punto anterior el valor de χ^2 también cumple con que $\chi^2 \geq 0$.
- La única manera en la que se cumpla que $\chi^2 = 0$ es que $O_i = E_i \; \forall i, 1 \leq i \leq k$.
- Del punto anterior, ya que si $\chi^2 = 0$ implica que la ocurrencia esperada sea igual a la observada en cada caso en la hipótesis H, y ya que al ser la misma es una hipótesis totalmente valida, entonces entre más cercano sea el valor de χ^2 a 0, más factible es la hipótesis propuesta por H.

Capítulo 4

Selección de Herramientas

En este capítulo se describirán las herramientas seleccionadas para el desarrollo de este trabajo.

4.1. Lenguaje de Programación

En esta selección se mencionan características y una breve descripción del lenguaje de programación seleccionado, así como de las herramientas de desarrollo para la implementación con dicho lenguaje.

La selección de uno o varios lenguajes de programación es una tarea muy importante antes de empezar el desarrollo de cualquier proyecto, se deben considerar los lenguajes de programación conocidos y desconocidos disponibles para trabajar.

Para esto se consideran las librerías que poseen los lenguajes, herramientas para trabajar con ellos, así como la experiencia trabajando con aquellos conocidos.

Para este trabajo, los lenguajes de programación que se consideraron fueron los siguientes:

- C
- C++
- Java

- Python
- Common LISP

De esta lista de lenguajes el seleccionado para el desarrollo de este trabajo fue Common LISP, a continuación se describen las principales características de este lenguaje.

4.1.1. Common LISP

LISP (del inglés *List Processing*), es un lenguaje de programación desarrollado en la década de los años 60 del siglo pasado en el MIT por John McCarthy[31], es actualmente el segundo lenguaje de programación mas antiguo en uso.

Desde sus inicios ha estado altamente relacionado con el desarrollo de proyectos sobre inteligencia artificial[32], esto se debe a las características del mismo lenguaje.

Existen varias implementaciones de LISP, esto debido a la tardanza para crear un estándar del lenguaje, sin embargo en 1984 aparece el dialecto Common LISP, cuyo estándar ANSI fue definido en 1994.

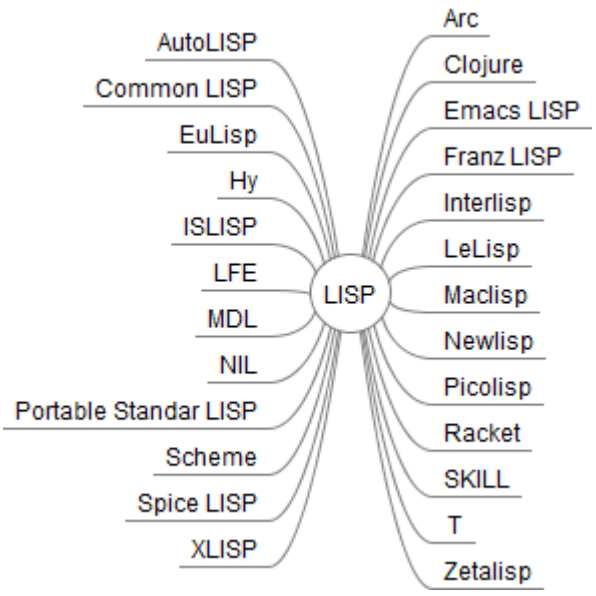

Figura 4.1: Dialectos de LISP

LISP es un lenguaje de programación multiparadigma y multiplataforma, LISP fue el primer lenguaje de programación homoicónico, por lo que Common LISP como todos los dialectos LISP utiliza expresiones-S tanto en código como en la estructura de los datos que maneja, por lo que el manejo del código desde el mismo programa es fácil.

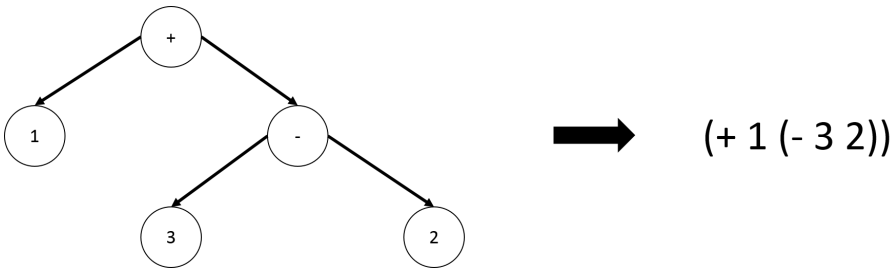

Figura 4.2: Estructura de una Expresión-S

A las expresiones-S también se les llaman Notación Prefija de Cambridge, la figura 4.2 muestra la estructura de una expresión-S es como un árbol equivalente a una lista. Siempre el nodo donde nace una nueva rama es un operador, y sus hojas pueden ser operadores para otras sub-ramas o valores para el operador.

Las listas se forman de la siguiente manera: como primer elemento el nodo que da origen a la rama, y sus hojas los demás elementos leyendo el árbol de izquierda a derecha, cuando uno de esos nodos en una sub-rama, representa una lista anidada, se resuelven desde las listas más anidadas hacia afuera. En la figura 4.3 se puede ver un ejemplo de esto.

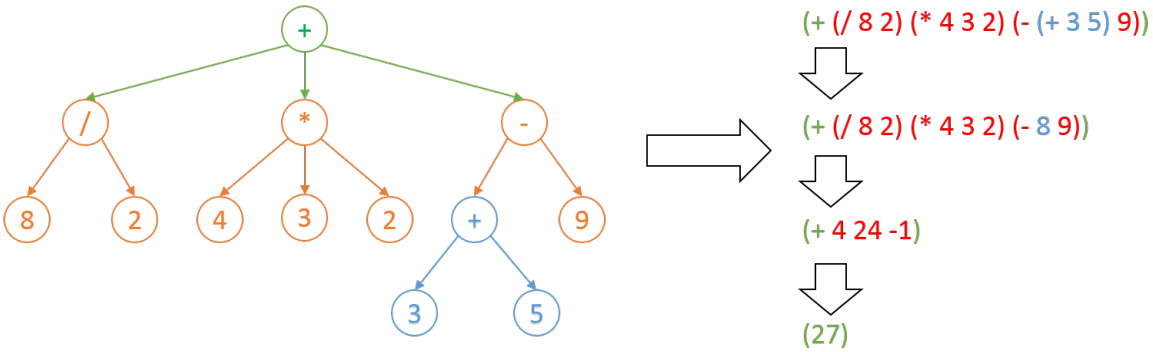

Figura 4.3: Forma en la que se resuelve una Espresión-S

Estas estructuras como se dijo antes pueden ser tanto para escribir el código como para almacenar la información que procesan los programas, razón principal

para utilizar Common LISP en el desarrollo de este trabajo.

Recordando la sección 2.2 sobre algoritmos evolutivos, la representación de una posible solución (un cromosoma) es un arreglo de posibles valores que den solución al problema en cuestión, y a su vez se manejan conjuntos de soluciones posibles para el proceso de evolución artificial.

Common LISP maneja una gran cantidad de tipos de datos, y el manejo del tamaño de memoria para cada uno es variable, por lo que el limite de valores soportados por una variable es dinámico, lo que lo hace ideal para tareas donde se requieren variables de tamaños muy grandes o con alta precisión, como en este caso.

Por estas razones es que se eligió el lenguaje Common LISP para la parte del desarrollo de las técnicas evolutivas que se seleccionaron para el desarrollo de este trabajo.

4.1.1.1. SBCL

El *Steel Bank Common Lisp* o SBCL es un compilador e intérprete para el lenguaje Common LISP en su versión ANSI, es un software libre y de código abierto que tiene implementaciones en una gran cantidad de plataformas[33].

Este compilador fue elegido para el desarrollo del trabajo, ya que cumple con el estándar ANSI de Common LISP y es uno de los compiladores más utilizados por la comunidad de desarrolladores de LISP.

Este software es multiplataforma lo que permite al código de escrito en LISP se ejecutado sin dificultad en diferentes sistemas operativos.

4.2. Otras Herramientas

En esta sección se describirán otras herramientas utilizadas para el desarrollo de este trabajo.

4.2.1. gnuplot

En cualquier trabajo de investigación, es necesario llevar un control de la información resultante de los experimentos, los valores de las variables utilizados, los resultados obtenidos, estadisticas de todo el desarrollo de la innvestigación.

El manejo de gráficas es una útil herramienta para interpretar información como la que se requiere en este trabajo, es de suma importancia elegir una herramienta de software adecuada, razón por la cual se utilizará gnuplot.

gnuplot es un programa de generación de gráficos desde linea de comando desarrollado para diferentes plataformas, el código fuente tiene un licencia de distribución libre.

Fue desarrollado por científicos y estudiantes para la visualización de funciones matemáticas e información interactiva[34].

Las características de gnuplot cubren las necesidades requeridas para controlar la visualización de información utilizada.

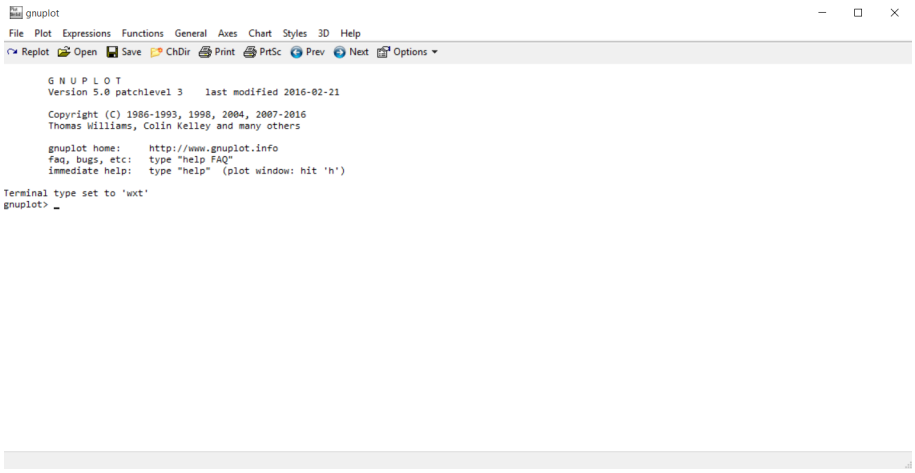

Figura 4.4: gnuplot

4.2.2. Sublime Text

Es un editor de texto y editor de código fuente escrito en C++, inicio como una extensión de Vim, puedes descargarse de manera gratuita pero no es software gratuito, aunque la versión de evaluación no cuenta con caducidad y es plenamente funcional[35].

Este editor contiene varias características por las que fue elegido para el desarrollo de este trabajo. En primera tiene una gran cantidad de lenguajes que soporta (Entre ellos LISP) lo que facilita la creación, revisión y reestructuración de código.

Cuenta con características que permiten tener varios archivos abiertos y organizados de tal manera que sea fácil la consulta de estos.

También cuenta con resaltado de palabras reservadas de lenguaje, atajos personalizados en el teclado y otras herramientas de ayuda al programador.

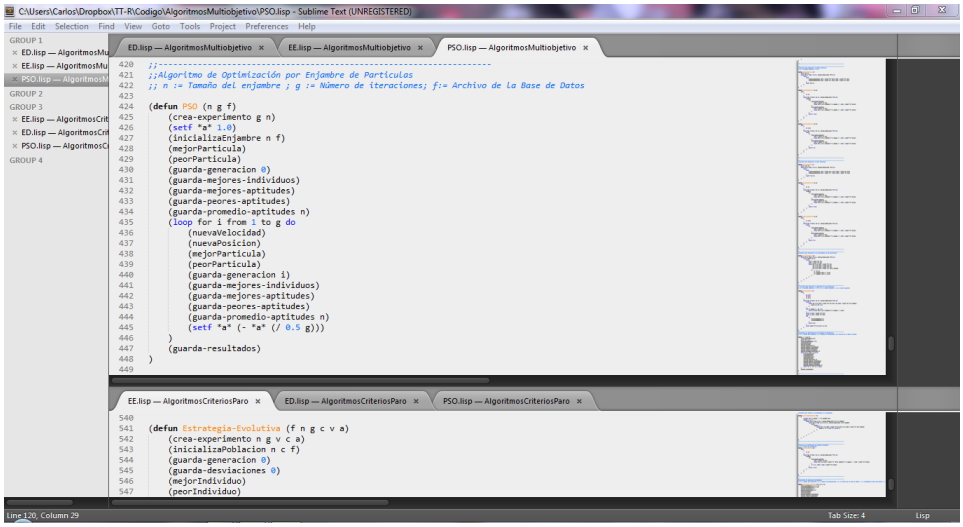

Figura 4.5: Editor Sublime Text

Capítulo 5

Selección de Algoritmos Evolutivos

Existen diferentes algoritmos evolutivos, cada uno con sus propias características, para el presente trabajo se seleccionaron los 3 siguientes:

- Evolución Diferencial (**DE**)
- Optimización por Enjambre de Partículas (**OEP**)
- Estrategias Evolutivas(**EE**)

El funcionamiento de cada uno de estos algoritmos sera explicado a continuación.

5.1. Evolución Diferencial (ED)

La *Evolución Diferencial* (ED) es un algoritmo evolutivo que usa como operador primario la mutación y que utiliza tres padres para generar un nuevo individuo[36], este algoritmo fue diseñado para trabajar con conjuntos de espacios de búsqueda continuos por lo que se ajusta a la problemática presente.

Fue propuesto por Rainer Storn y Kenneth Price[37] en sus intentos por resolver el polinomio de Chebychev, en donde concibieron la idea de utilizar la diferencia entre los individuos de la población para perturbar a los mismos.

La propuesta es que cada nuevo individuo en el cambio entre una generación y otra sera generado por tres padre ρ_a, ρ_b y ρ_c elegidos aleatoriamente diferentes entre si.

Para esto el ED establece que primero debe aplicarse el operador de mutación, esto se consigue perturbando a uno de los padres (ρ_a) con la diferencia entre los otros dos (ρ_b y ρ_c) como se ve en la ecuación 5.1.

$$\tau_a = \rho_a + F(\rho_b - \rho_c) \tag{5.1}$$

En la ecuación 5.1 el parámetro F es para dar un control sobre la proporción de la mutación generalmente tomado en un intervalo $[0, 2]$.

Después se continua con la aplicación del operador de cruza entre el individuo formado por la mutación (τ_a) y su padre principal, esto haciendo uso de un parámetro C para controlar la tasa de recombinación como muestra la ecuación 5.2 en cada i.

$$\delta_{a,i} = \begin{cases} \tau_{a,i} & si \quad rand(0,1) < C \\ \rho_{a,i} & si \quad rand(0,1) \geq C \end{cases} \tag{5.2}$$

Finalmente el proceso de selección para saber si el nuevo individuo δ_a pasa a la siguiente generación o el anterior ρ_a se conserva, esto es mediante la comparación de la aptitud de ambos como muestra la ecuación 5.3.

$$\rho_a = \begin{cases} \delta_a & si \quad aptitud(\delta_a) > aptitud(\rho_a) \\ \rho_a & si \quad aptitud(\delta_a) \leq aptitud(\rho_a) \end{cases} \tag{5.3}$$

El proceso continua a lo largo de las generaciones hasta llegar a la última, el algoritmo 2 describe el funcionamiento completo de la ED.

Como se menciono en la sección 2.2 y se puede ver en el algoritmo 1, otra variable de entrada y criterio posible de paro es una aptitud esperada y un rango de aceptación para esta, eso puede incluirse modificando la condición de paro del ciclo de evolución de la ED.

Existen varias estrategias utilizadas para la implementación de la ED, la anterior es conocida como *ED/aleatorio/1/bin*, otras propuestas por Price y Storn son:

- *ED/mejor/1/exp*

Algoritmo 2: Evolución Diferencial (ED)

Entrada:
n: Tamaño de población
g: Número de generaciones
Salida:
q : Mejor individuo de la ultima generación

```
1  p ← generarPoblacion(n) : Genera la población inicial;
2  i ← 1 : Contador de generaciones;
3  q ← mejorIndividuo(p) : Mejor individuo de la primera generación;
4  mientras i ≤ t hacer
5  |  f ← elegirPadres(p) : Formar grupos de padre;
6  |  t ← mutacion(f) : Aplicación de la mutación;
7  |  r ← cruza(t, p) : Aplicación de la cruz;
8  |  p ← seleccion(r, p) : Selección para la nueva generación;
9  |  q ← mejorIndividuo(p) : Elegir al mejor individuo de la generación;
10 |  i ← i + 1;
11 devolver q;
```

- *ED/aleatorio/1/exp*
- *ED/aleatorio-al-mejor/1/exp*
- *ED/mejor/2/exp*
- *ED/aleatorio/2/exp*
- *ED/mejor/1/bin*
- *ED/aleatorio-al-mejor/1/bin*
- *ED/mejor/2/bin*
- *ED/aleatorio/2/bin*

La forma *ED/a/b/c* puede interpretarse de la siguiente manera:

ED : Se refiere a la Evolución Diferencial

a : Es el individuo que sera perturbado, el padre principal, puede ser el mejor de la generación actual o elegido al azar.

b : Es el número de vectores que serán perturbados, por cada uno deben elegirse otros dos para perturbarlos.

c : Es el tipo de cruza, existen dos: exponencial y binomial:

Exponencial(exp) : Cuando se cumple la segunda condición de la ecuación 5.2, a partir de esa posición todas los gentes posteriores permanecen sin cambios

Binomial (bin) : La cruza se realiza en todas las posiciones del individuo como se planteo en la ecuación 5.2.

El algoritmo ED ha sido utilizado para trabajos de toma de decisiones,diseño en ingeniería, diagnostico de fallos en sistemas industriales, representación mínima de fusión de sensores, determinación de epicentros de sismos, etc.[38, 39, 40]

5.2. Optimización por Enjambre de Particulas (OEP)

La observación de ciertos comportamientos en la naturaleza de grupos de seres vivos como los siguientes:

- Grupos de hormigas que encuentran caminos más cortos entre sus nidos y sus sus fuentes de alimento.
- Formaciones complejas durante el vuelo de parvadas de aves.
- Construcción de montículos de termitas de grandes dimensiones con control de temperatura.
- Comunicación entre abejas para la localización de alimento.

Todos estos comportamientos nos demuestran que las especies pueden transmitir información entre sus miembros para lograr tareas complejas.

Inspirado en este comportamiento se creo el algoritmo de *Optimización por Enjambre de Partículas* (OEP), que crea un enjambre compuesto por varios individuos que van mejorando en conjunto como grupo[6, 41].

Cada partícula del enjambre tiene una posición y velocidad aleatoria originalmente, que van cambiando con cara iteración lo que va permitiendo que las partículas "vuelen"por el espacio de búsqueda guidas por su experiencia propia y por la colectiva.

Esto hace que las principales características del OEP son la evaluación, comparación y la imitación.

Cada partícula al igual que los seres vivos en la naturaleza tiene la capacidad de evaluar para poder mejorar su aprendizaje de alguna actividad.

También es capaz de poder compararse con las demás partículas para poder medir su propio rendimiento.

Y finalmente, la imitación, es una características que no todos los seres vivos tienen, basado en las anteriores una partícula puede cambiar su comportamiento basado en lo observado en partículas con mejores resultados comparadas con ella.

La figura 5.1 ilustra el movimiento de una partícula i.

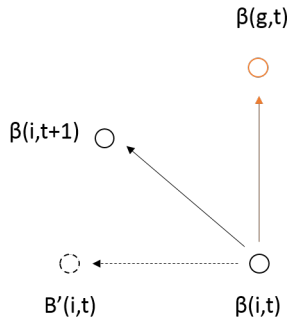

Figura 5.1: Movimiento de una Particula i

La partícula $\beta_{i,t}$ en el tiempo t es afectada tanto por su mejor posición que ha tenido ($\beta'_{i,t}$) hasta el mismo tiempo y la partícula $\beta_{g,t}$ que es la mejor partícula hasta el momento para conseguir su nueva posición en el tiempo $t+1$.

La velocidad $V_{i,t}$ de cada partícula se actualiza mediante la ecuación 5.4:

$$V_{i,t+1} = \alpha V_{i,t} + rand(0,\varphi_1)[\beta'_{i,t} - \beta_{i,t}] + rand(0,\varphi_2)[\beta_{g,t} - \beta_{i,t}] \tag{5.4}$$

La primera parte de la ecuación (Ecuación 5.5) representa el factor de auto mejoramiento basado en la comparación de la partícula consigo misma en su propia experiencia, mientras que la segunda parte (Ecuación 5.6) representa la influencia social del mejor miembro del enjambre.

$$rand(0,\varphi_1)[\beta'_{i,t} - \beta_{i,t}] \tag{5.5}$$

$$rand(0, \varphi_2)[\beta_{g,t} - \beta_{i,t}] \tag{5.6}$$

Se han propuesto versiones del algoritmos basadas solamente de una de estas dos características, en este trabajo se usara la versión que incluye ambas, el mejoramiento basado en experiencia particular y la de grupo.

Los otros parámetros α y $rand(0, \varphi)$ son un parámetro de escala y una un valor aleatoria de una distribución uniforme en el intervalo $[0, \varphi]$ respectivamente, estos determinan de manera aleatoria el porcentaje de la influencia de la velocidad de la partícula y de la influencia de la experiencia propia y colectiva.

Finalmente la nueva posición de cada partícula esta dada por la ecuación 5.7.

$$\beta_{i,t+1} = \beta_{i,t} + V_{i,t+1} \tag{5.7}$$

El algoritmo 3 describe el funcionamiento del OEP:

Algoritmo 3: Optimización por Enjambre de Partículas (OEP)

Entrada:
n: Tamaño del enjambre
t: Número de iteraciones
Salida:
g : Partícula con la mejor posición

```
p ← inicializaEnjambre(n) : Inicialización del enjambre;
i ← 1 : Contador de iteraciones;
g ← mejorParticula(p) : Mejor partícula;
mientras i ≤ t hacer
    nuevaVelocidad(p, g) : Actualiza la velocidad de las partículas;
    nuevaPosicion(p) : Calcular nuevas posiciones;
    g ← mejorParticula(p) : Obtener la Mejor partícula;
    i ← i + 1;
devolver g
```

El algoritmo de OEP ha sido usado para atacar problemas como la obtención de redes bayesianas para el diagnóstico de la hipertensión arterial, estimación de parámetros de regresión no lineal, planificación de zonas electorales, etc.[42, 43, 44].

5.3. Estrategias Evolutivas (EE)

Las Estrategias Evolutivas (EE) fueron propuesta en la década de los 60 del siglo pasado por Bienert, Rechenberg y Schwefel en Alemania[6]. Este algoritmo propone el uso de la mutación como principal operador para la generación de nuevos individuos que satisfagan el problema que se intenta resolver.

Las estrategias evolutivas plantean que de un grupo de μ padres se generen a través de mutación λ hijos (en la literatura se propone que $\lambda = 7\mu$).

Existen dos versiones de este algoritmo, el $(\mu + \lambda) - EE$ y el $(\mu, \lambda) - EE$. El primero establece que se generen los λ hijos de los μ padres y de la suma de ambos grupos se elijen los μ mejores para formar la nueva generación, mientras que la segunda sigue un mismo procedimiento, pero la diferencia radica en que los individuos que pasan a formar parte de la nueva generación solo se eligen del conjunto de los hijos, ningún padre pasa a ser parte de esta.

La forma en que se da la mutación en este algoritmo es la siguiente:

1. Se selecciona un padre x
2. A cada elemento x_i del padre se le suma un elemento δ_i que esta dado por una distribución normal $N(0, \sigma_i^2)$

El proceso de mutación y evolución en este algoritmo queda ilustrado en las imágenes 5.2 y 5.3 respectivamente.

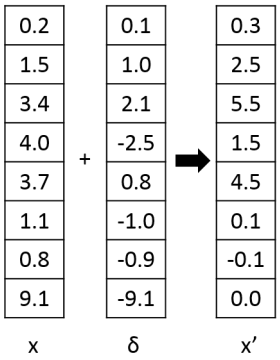

Figura 5.2: Mutación de un padre x para generar un hijo x'

Se puede agregar un factor de auto-adaptación permitiendo que las desviaciones estándar σ_i cambien con el paso de las iteraciones del algoritmo, para esto hay dos formas: la adaptación determinista y la aleatoria.

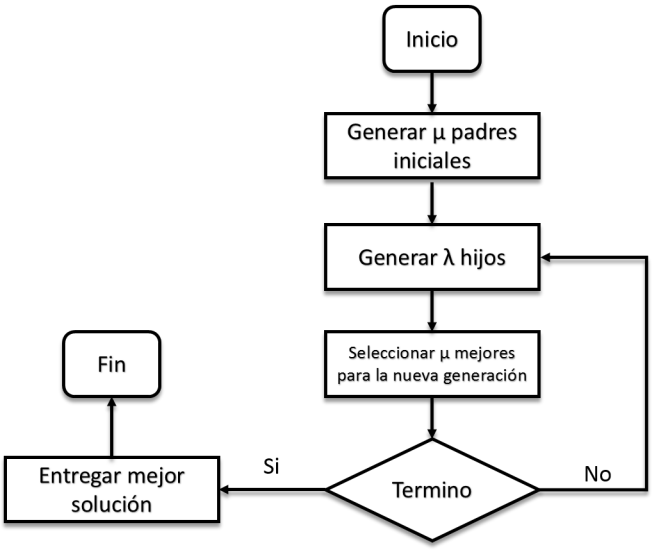

Figura 5.3: Flujo de una Estrategia Evolutiva

La primera se lleva acabo mediante una técnica heurística propuesta por Rechenberg y mejorada despues por Greenwood conocida como *La regla de éxito* 1/5[45].

La regla ya mejorada por Greenwood establece lo que se indica en la ecuación 5.8.

$$\sigma = \begin{cases} \text{mín}(\frac{\sigma}{c}, D) & \text{si} \quad p_s > \frac{1}{5} \\ c\sigma & \text{si} \quad \frac{1}{20} \leq p_s \leq \frac{1}{5} \\ \text{mín}(2\sigma, D) & \text{si} \quad p_s < \frac{1}{20} \\ \sigma & \text{si} \quad p_s = \frac{1}{5} \end{cases} \tag{5.8}$$

Las variables que controlan el comportamiento de la ecuación 5.8 tienen los siguiente valores:

p_s : Es el porcentaje de mutaciones cuya aptitud supera a sus padres en alrededor de $10n$ experimentos, donde n son las dimensiones del espacio de búsqueda.

c : Una constante con valor 0,85

D : Es el diámetro del espacio de búsqueda, este valor más la condición de $\frac{1}{20}$ fueron las modificaciones introducidas por Greenwood a la regla establecida por Rechenberg.

La regla original solo no contempla el factor D y la verificación de p_s con respecto al valor de $\frac{1}{20}$.

La segunda manera de actualizar las desviaciones estándar de los individuos de la EE es haciendo a estas parte de su representación como se muestra en la figura 5.4.

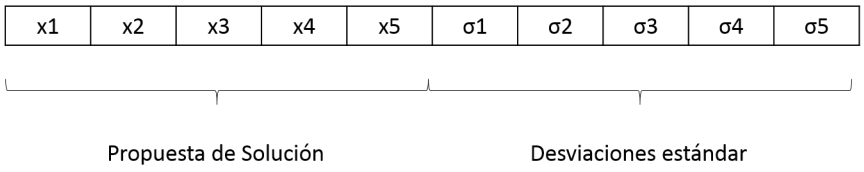

Figura 5.4: Códificación de un individuo en una EE para actualizar aleatoriamente las desviaciones

Para esto los valores de σ_i' para el hijo generado por el padre x se hace de acuerdo a la siguiente regla:

$$\sigma_i' = \sigma_i \exp(t\delta) \tag{5.9}$$

δ es un número aleatorio de una distribución normal $N(0,1)$ y t es una constante dada que es proporcional a $\frac{1}{\sqrt{n}}$ donde n es la cantidad de dimensiones del espacio de búsqueda. Los valores para las variables del hijo generado utilizan la nueva desviación estándar para ser generados como se menciono anteriormente.

El algoritmo 4 nos muestra el funcionamiento de las EE para cualquiera de

sus versiones.

Algoritmo 4: Estrategias Evolutivas (EE)

Entrada:
n: Tamaño de población
t: Número de iteraciones
Salida:
g : Mejor individuo

```
1 u ← generarPoblacion(n) : Genera la población inicial;
2 i ← 1 : Contador de generaciones;
3 g ← mejorIndividuo(u) : Mejor individuo generado;
4 mientras i ≤ t hacer
5     h ← mutarPadres(u) : Mutación de padres para generar hijos;
6     u ← seleccionarIndividuos(u, h) : Seleccionar los individuos para la
        nueva generación;
7     g ← mejorIndividuo(u);
8     i ← i + 1;
9 devolver g
```

La función de selección de individuos para la nueva generación varia según sea la versión del algoritmo, la mostrada en el algoritmo 4 tiene como parámetros tanto a padres como a hijos, pero dependiendo el caso puede ser solo los hijos los que necesite para esta.

Algunas versiones mas de EE incluyen el operador de cruza como un operador secundario después de la mutación, donde se aplica alguna técnica de cruza ya establecida anteriormente en la literatura o proponiendo una nueva. En este trabajo no se utilizara la cruza para este algoritmo.

Este algoritmo ha sido utilizado para resolver problemas como la forma de cuerpos que son sujetos a viento, entrenamiento de redes neuronales, etc.[46]

Capítulo 6

Diseño y Desarrollo de Prototipos

El modelo que se utilizara para la experimentación de este trabajo sera mediante el desarrollo de prototipos.

Cada prototipo de algoritmo recibirá la información de cuantas veces ocurrió uno de los resultados de algún evento en un cierto tiempo y los parámetros del mismo y como resultado arrojara la mejor solución encontrada por este.

Durante el proceso el algoritmo recopilara información como las poblaciones generadas, un histórico de las mejores soluciones encontradas y datos extras dependiendo de cada algoritmo. Esto queda ilustrado en la figura 6.1 que muestra un esquema general del funcionamiento de los prototipos.

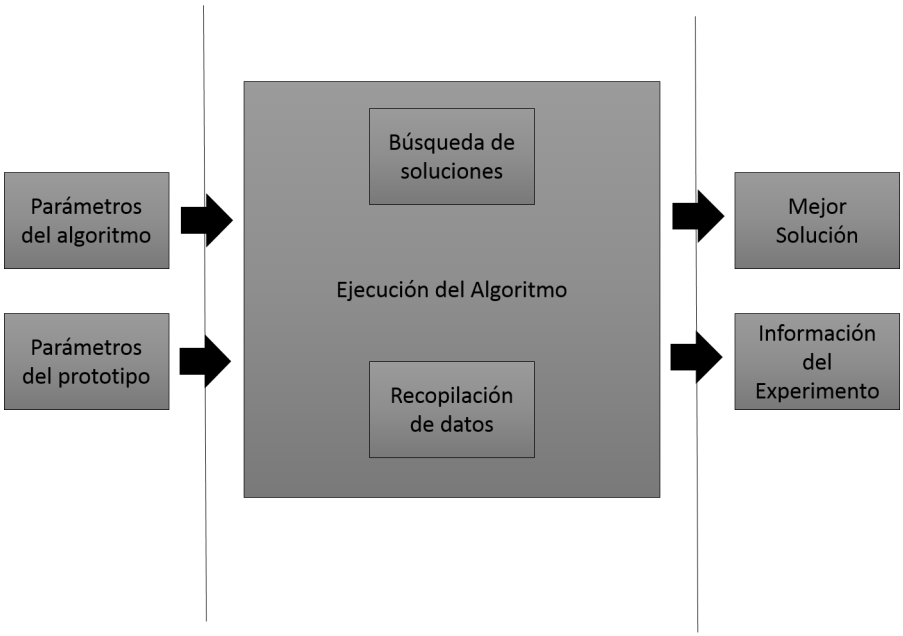

Figura 6.1: Esquema General de los Prototipos

A continuación se describe en que consiste cada uno de los prototipos que se desarrollaran:

Prototipo 1 Se programaran los algoritmos para resolver funciones de dos variables, todas serán funciones de minimización, el objetivo de estos prototipos es el asegurar que se entiende el funcionamiento de los algoritmos y cada una de sus etapas.

Prototipo 2 Se realizaran ajustes a los prototipos 1 de cada algoritmo para volverlos multivariables y que hagan uso de la Prueba χ^2 de Pearson como función objetivo.

Prototipo 3 Se ajustaran los anteriores prototipos de cada algoritmo para que sean ahora algoritmos multiobjetivo que resuelvan más de una función de optimización a la vez. En este caso diferentes Pruebas χ^2 de Pearson generadas por diferentes distribuciones generadas de manera aleatoria.

Prototipo 4 Se implementara un criterio de paro correspondiente a un valor de que corresponda a un porcentaje de aceptación que se busca para las diferentes Pruebas χ^2 de Pearson que se intentan optimizar simultáneamente.

Con estos cuatro prototipos se realizaran cinco experimentos, los primeros cuatro con el prototipo correspondiente y el quinto con el prototipo cuatro, las diferencias entre el experimento cuatro y cinco radica en que el primero utilizara como información de entrada distribuciones generadas por el código D.3 y el segundo con datos reales de frecuencias de robos. En este capitulo se describe la ejecución y resultados de los primeros cuatro experimentos, en el capitulo 7 se encuentran los datos de ejecución y resultados del experimento con datos reales.

El código de ejecución de los experimentos se puede consultar en D.6 y para su ejecución en linea de comando se hace como se observa en 6.1, esto es valido tanto para plataformas Windows como Linux.

Código 6.1: Ejecución de Experimentos

```
1         sbcl --script archivo.lisp
```

En cada ejecución de unos de los algoritmos en un experimento la información generada en estos sera guardada en una estructura de directorios distinta dependiendo de cada algoritmo y la versión de este que se utilice.

Esta estructura de directorios para cada uno de los algoritmos se puede ver en las figuras 6.2, 6.3 y 6.4.

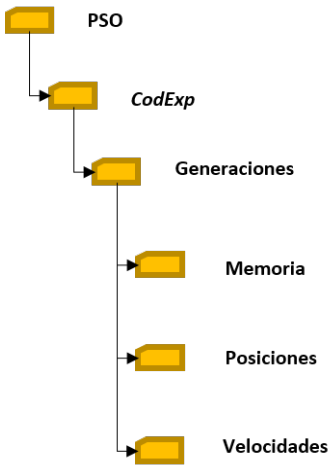

Figura 6.2: Estructura de Directorios de un experimento con el Algoritmo OEP

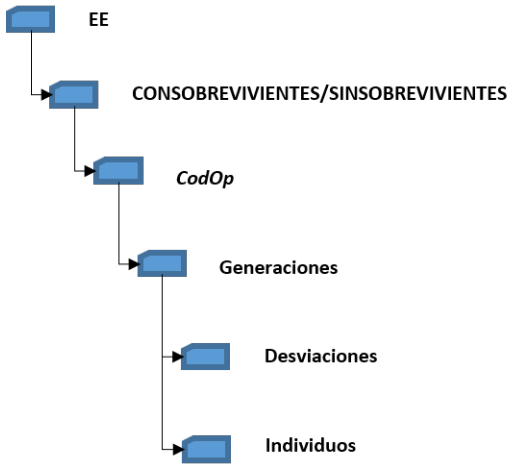

Figura 6.3: Estructura de Directorios de un experimento con el Algoritmo EE

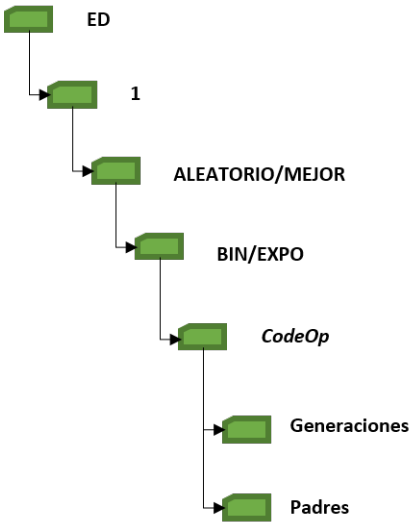

Figura 6.4: Estructura de Directorios de un experimento con el Algoritmo ED

Las carpetas que muestran una estructura de nombre estilo *a/b* tomaran como nombre una de las opciones separadas por / dependiendo de la versión del algoritmo que se ejecuta.

La carpeta *CodExp* tiene como nombre un número generado en cada experimento por el reloj de la computadora donde se ejecuta.

También cada experimento genera un archivo de texto plano con la información del experimento en esta carpeta, un ejemplo de como esta estructurado este archivo se pude ver en la figura 6.5, ademas se generaran archivos que guarden la mejor solución y el historial de mejores soluciones, también en formato de texto plano.

Figura 6.5: Ejemplo de Archivo de datos del experimento

Las subcarpetas de ese directorio tienen los históricos por generaciones con la información que genera cada algoritmo, guardando cada generación en un archivo de texto plano que tiene la información de los individuos que las componen.

6.1. Prototipo 1: Minimización de Funciones

El objetivo del primer prototipo es hacer uso de los algoritmos seleccionados para la minimización de funciones de pruebas, esto para poder comprender y ver el funcionamiento de cada uno de estos algoritmos para poder realizar más adelante ajustes a estos.

La funciones seleccionadas son:

- Función de Langermann
- Función de Griewangk
- Función de Schwefel
- Función de Rosenbrock
- Función de Shubert

Todas estas funciones son multivariable, aunque para este experimento se utilizaran sus versiones para dos variables para poder observar mejor el funcionamiento de los algoritmos, en el Apéndice B se describen a detalles las características de estas funciones y el código de implementación en lenguaje Common Lisp de estas es D.1.

El código de los primeros prototipos de los algoritmos (D.6,D.10,D.14),reciben ademas de los parámetros propios del algoritmo también la función objetivo a optimizar y los limites del espacio de búsqueda.

6.1.1. Experimentación Algoritmo OEP

Para el algoritmo OEP se utilizaron poblaciones de 10, 100 y 1000 partículas, también se utilizaron 100, 500 y 1000 ciclos de búsqueda con cada una de estas poblaciones.

Los resultados de estas ejecuciones se encuentran en la tabla 6.1

Tabla 6.1: Mejores aptitudes reportadas por el prototipo 1 del algoritmos OEP

Función		Langermann	Griewangk	Schwefel	Rosenbrock	Shubert
Población	Ciclos					
10	100	-4.082695	0.007396	0.0	6.474534e-4	-186.71216
	500	-4.127576	0.027125	0.0	5.201528e-9	-186.62779
	1000	-4.155809	0.0	-6.103515e-5	3.398113e-9	-186.62779
100	100	-4.155809	0.0	0.0	1.776356e-10	-186.73094
	500	-4.155809	0.607395	0.0	0.0	-186.73096
	1000	-4.155809	0.0	-6.103515e-5	0.0	-186.73096
1000	100	-4.155809	0.0	-6.103515e-5	0.0	-186.73096
	500	-4.155809	0.0	-6.103515e-5	0.0	-186.73096
	1000	-4.155809	0.0	-6.103515e-5	0.0	-186.73096

En alguno casos el algoritmo OEP llego a soluciones muy cercanas a las optimas y en otros si llego a este, lo que demuestra que la efectividad de este algoritmo para atacar problemas de minimización como este caso el de predicción usando la Prueba χ^2 de Pearson.

En general el algoritmo OEP mostró buenos resultados en los diferentes espacios de búsqueda que plantean cada una de las funciones de minimizan resueltas por el.

6.1.2. Experimentación Algoritmo ED

Durante la ejecución del algoritmo ED se realizaron ejecuciones con 4 de sus versiones con poblaciones de 10, 100 y 1000 individuos y generaciones de 100, 500 y mil iteraciones de igual manera que con el algoritmo OEP.

Los mejores resultados de estas ejecuciones se muestran en las tablas 6.2, 6.3, 6.4 y 6.5.

Tabla 6.2: Mejores aptitudes reportadas por el prototipo 1 del algoritmos ED/-Mejor/1/bin

Población	Ciclos	Langermann	Griewangk	Schwefel	Rosenbrock	Shubert
10	100	-3.998876	0.047921	16.59899	0.302507	-162.17336
	500	-4.154344	0.093089	0.145763	0.013253	-186.71802
	1000	-4.0992	0.035952	0.013854	00001584	-186.6585
100	100	-4.150286	0.0358666	4.2724e-4	0.001369	-186.6418
	500	-4.127202	0.001634	0.001312	0.004186	-186.73091
	1000	-4.083874	0.007424	1.2207e-4	7.2157e-4	-186.72984
1000	100	-4.15578	0.007554	2.4414e-4	0.006865	-186.73071
	500	-4.155806	0.007396	0.0	1.8244e-4	-186.7309
	1000	-4.127576	4.7683e-7	0.0	5.4768e-5	-186.73096

Tabla 6.3: Mejores aptitudes reportadas por el prototipo 1 del algoritmos ED/-Mejor/1/exp

Población	Ciclos	Langermann	Griewangk	Schwefel	Rosenbrock	Shubert
10	100	-3.701435	0.148501	241.39081	0.174817	-186.13664
	500	-4.0641	0.7411251	140.8377	0.60092	-121.30166
	1000	-3.757869	0.29949	0.6045	0.003904	-185.52936
100	100	-3.677508	0.15238	8.60614	8.25011e-5	-186.7238
	500	-4.12409	0.41308	1.180908	0.018034	-186.72566
	1000	-4.15474	0.52545	0.09082	0.001123	-183.0843
1000	100	-4.153871	0.174416	0.011352	0.010054	-186.72475
	500	-4.127267	0.022445	2.4414e-4	2.2259e-4	-186.72621
	1000	-4.154936	0.005405	0.009177	7.3173e-4	-186.73087

Tabla 6.4: Mejores aptitudes reportadas por el prototipo 1 del algoritmos ED/Aleatorio/1/bin

Población	Ciclos	Langermann	Griewangk	Schwefel	Rosenbrock	Shubert
10	100	-2.882578	0.23653	151.0739	0.3426	-117.92509
	500	-4.07757	0.040937	0.039062	0.009309	-186.23605
	1000	-4.07757	0.002242	0.001708	0.011051	-186.72868
100	100	-3.775902	0.224225	15.133179	0.014704	-184.1084
	500	-4.106888	0.007875	0.006347	6.9891e-4	-186.73024
	1000	-4.153344	0.001305	0.0	0.001034	-186.73068
1000	100	-4.052852	0.013827	0.179443	3.3993e-4	-186.25227
	500	-4.154812	0.001969	3.6621e-4	4.44785e-7	-186.7274
	1000	-4.155603	8.2433e-5	0.0	2.8378e-5	-186.7309

Tabla 6.5: Mejores aptitudes reportadas por el prototipo 1 del algoritmos ED/Aleatorio/1/exp

Población	Ciclos	Langermann	Griewangk	Schwefel	Rosenbrock	Shubert
10	100	-3.129757	0.036198	5.613647	0.095104	-119.84841
	500	-3.759628	0.005614	1.212829	1.8573e-4	-170.50189
	1000	-4.100582	0.012513	0.002075	0.003623	-164.62337
100	100	-4.134682	0.20344	21.750854	0.003651	-182.06752
	500	-4.139694	0.020344	0.008056	3.7244e-4	-186.71046
	1000	-4.14459	0.008545	0.001464	2.2771e-4	-186.71848
1000	100	-4.131365	0.071914	0.1145	3.8536e-4	-186.70668
	500	-4.154862	8.6003e-4	6.1035e-5	1.7610e-6	-186.73077
	1000	-4.155398	0.002222	4.8828e-4	9.7197e-6	-186.73076

En general este algoritmo demostró una dar buenos resultados en sus diferentes configuraciones en cada versión, exceptuando en casi todos los casos en la configuración de la población y número de iteraciones más pequeñas menos en la función de Rosenbrock.

Las versiones del algoritmo donde se realizan las operaciones genéticas sobre el mejor individuo parecen en los casos de poblaciones e iteraciones más pequeñas llevar una cierta ventaja, y se pude observar como en estos también se consiguió en algunas ocasiones el optimo global en las funciones de Schwefel y Shubert.

6.1.3. Experimentación Algoritmo EE

En el caso de el algoritmo EE, ya que se considera ademas del tamaño de población y el número de ciclos de ejecución el número de hijos que cada individuo generara se probaron solo 3 poblaciones pequeñas, de 1, 5 y 10 individuos en ejecuciones de 100 y 500 generaciones y en cada uno generación de solo un hijo por individuo ó 5, salvo el caso de la versión sin sobrevivientes ya que solo se considera la generación de 5 hijos por individuo, pues la generación de un solo individuo sin presencia de los mejores de una generación pasada se acerca mucho a una búsqueda a ciegas debido a que el único operador genético en este algoritmo es la mutación.

En la tabla 6.6 se ven los mejores resultados de las ejecuciones.

Tabla 6.6: Mejores aptitudes reportadas por el prototipo 1 del algoritmos EE

Versión			$(\mu + \lambda) - EE$				
Función			Langermann	Griewangk	Schwefel	Rosenbrock	Shubert
Población	Hijos	Ciclos					
1	1	100	-0.736647	25.992905	1082.3987	0.185853	-79.05589
		500	-0.135165	42.530693	1220.366879	112.322044	-6.459881
	5	100	-0.171775	66.092514	719.8543	0.956434	-9.431548
		500	-2.165105	128.84393	797.810154	0.483802	-51.352293
5	1	100	-0.324833	33.635611	870.533492	0.179141	-28.804182
		500	-2.19334	22.957792	415.223550	0.139539	-12.111357
	5	100	-0.620001	60.697817	452.275566	7.863608	-19.67609
		500	-1.461239	62.763935	617.947663	0.067364	-26.394348
10	1	100	-1.729666	16.341271	517.365	0.077729	-20.92452
		500	-3.39392	35.294662	314.6313	6.01602	-76.9097
	5	100	-1.53032	1.950451	307.03613	0.018625	-152.5566
		500	-1.75354	23.001473	144.285882	0.009858	-42.230619
Versión			$(\mu, \lambda) - EE$				
Función			Langermann	Griewangk	Schwefel	Rosenbrock	Shubert
Población	Hijos	Ciclos					
1	5	100	-0.892456	43.679856	855.598498	461.76053	7.824939
		500	-2.177188	12.772299	484.589138	461.760548	7.824939
5	5	100	-2.193325	9.559026	785.12319	57.424404	7.824939
		500	-0.400637	2.560799	527.58383	461.76053	7.824939
10	5	100	0.027008	2.626076	321.275976	461.76053	7.824939
		500	0.124370	11.128436	546.851556	461.76053	7.824939

Como se observa en la tabla, en general este algoritmo no dio buenos resultados, salvo en la función de Rosenbrock donde hubo el mejor acercamiento al optimo global.

A pesar de esto se puede observar en varios de los casos como el aumento en el número de iteraciones y de hijos generados en varios casos mejoraron los resultados, por lo que se debe considerar poblaciones más grandes y mayor número de ciclos en futuras ejecuciones de este algoritmo.

Este algoritmo mostró que puede realizar búsquedas que van mejorando con las iteraciones, sin embargo se necesitan probar configuraciones con poblaciones mayores y más iteraciones para mejores resultados.

6.1.4. Conclusiones del experimento 1

Los algoritmos OEP y ED demostraron buenos resultados en sus ejecuciones, el primero un poco mejores que el segundo, por lo que se debe considerar en futuros experimentos cambiar los parámetros de este, y para el algoritmo EE, a pesar de haber demostrado resultados muy malos, se puede observar que estos pueden mejorar al darle parámetros más grandes de entrada, por lo que se considera esto en los próximos experimentos esperando mejorar su desempeño.

6.2. Prototipo 2: Minimización de Prueba χ^2 de Pearson

El objetivo de este experimento es verificar la implementación de los algoritmos evolutivos para la resolución de la Prueba χ^2 de Pearson.

En el capitulo 3 se menciono que esta función tiene dos parámetros, el primero es una distribución de n fenómenos observados con i diferentes resultados, y el otro es un hipótesis sobre la distribución de las probabilidades de que ocurra cada uno de los diferentes resultados. La implementación de esta función en lenguaje Common LISP se puede ver en D.2.

Antes de la ejecución de este experimento, se generaron 9 grupos de distribuciones (Ver Apéndice C) con los códigos D.3 y D.4, con diferentes tamaños y posibles resultados, estas están almacenadas en archivos en formato de texto plano, la forma en que están guardados es en la primera linea separados por un espacio el número de resultados posibles y después la cantidad de incidencias totales de estos, y después los valores de incidencia de cada resultado separados por un salto de linea, esto se puede ver en la figura 6.6.

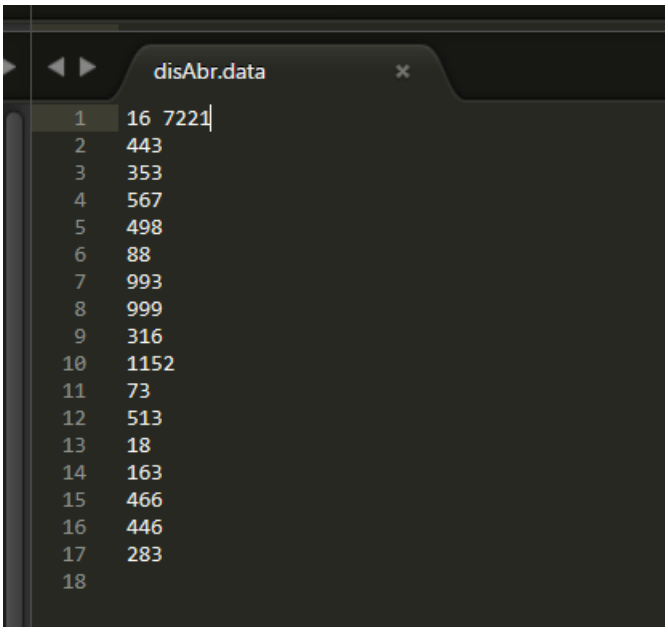

Figura 6.6: Ejemplo de Archivo de Distribución

En este experimento se utilizaran 3 casos que están definidos en la tabla 6.7.

Tabla 6.7: Casos para el experimento 2

Caso	1	2	3
Grupo	1	4	7
Distribución	1	1	1

En las figuras 6.7, 6.8 y 6.9 se observan los óptimos globales de los casos 1, 2 y 3 respectivamente.

0.15	0.13	0.38	0.17	0.17

Figura 6.7: Optimo Global del Caso 1

0.035	0.135	0.122	0.047	0.174	0.127	0.078	0.048	0.124	0.11

Figura 6.8: Optimo Global del Caso 2

0.0407	0.0178	0.0533	0.0434	0.0176
0.0651	0.0556	0.0295	0.0153	0.0411
0.0107	0.0068	0.0359	0.0242	0.009
0.0615	0.0557	0.0616	0.0609	0.0591
0.0203	0.0337	0.0614	0.06	0.0598

Figura 6.9: Optimo Global del Caso 3

Para revisar la probabilidad de que tan acertadas son las hipótesis que cada algoritmo lanzara como resultado se hace uso de la ecuación 3.2 que se despeja de para quedar como se muestra en la función 6.1.

$$\frac{\chi^2}{t} < \chi_t^2 \tag{6.1}$$

Como se menciono en 3 el valor de t es igual al número de posibles resultados de un fenómeno menos uno y el valor de χ_t^2 se obtiene de las tablas A.1 y A.2.

Con esto se medirán que tan buenos son los resultados arrojados por los algoritmos.

6.2.1. Experimentación Algoritmo OEP

El algoritmo OEP presento los resultados mostrados en la tabla 6.8 durante su ejecución.

Tabla 6.8: Mejores aptitudes reportadas por el prototipo 2 del algoritmos OEP

Población	Generaciones	Caso 1	Caso 2	Caso 3
10	100	0.12624	103.733315	3405.5188
	500	1.861e-8	118.33224	3125.745
	1000	6.0632e-14	13.592612	1533.1652
50	100	1.6616e-7	48.68301	2340.22
	500	6.0632e-14	13.847027	1440.6982
	1000	6.0632e-14	8.071136	1427.5852
100	100	1.9308e-8	32.387844	2740.193
	500	6.0632e-14	0.0637	1443.7672
	1000	6.0632e-14	2.8628e-4	962.92834

Tabla 6.9: Mejores iteraciones del prototipo 2 del algoritmo OEP

Casos	1	2	3
CodExp	3668982391	3668983067	3668983157
Población	10	100	100
Ciclos	1000	1000	1000
Mejor resultado	6.0632e-14	2.8628e-4	962.92834

En las 9 diferentes configuraciones para optimizar la Prueba χ^2 de Pearson presentaron diferentes resultados, entre más cercano sea el resultado a 0.0, más aceptable es la hipótesis propuesta por el algoritmo.

Las gráficas 6.10, 6.11 y 6.12 muestran la evolución de la mejor aptitud del enjambre en cada uno de los casos.

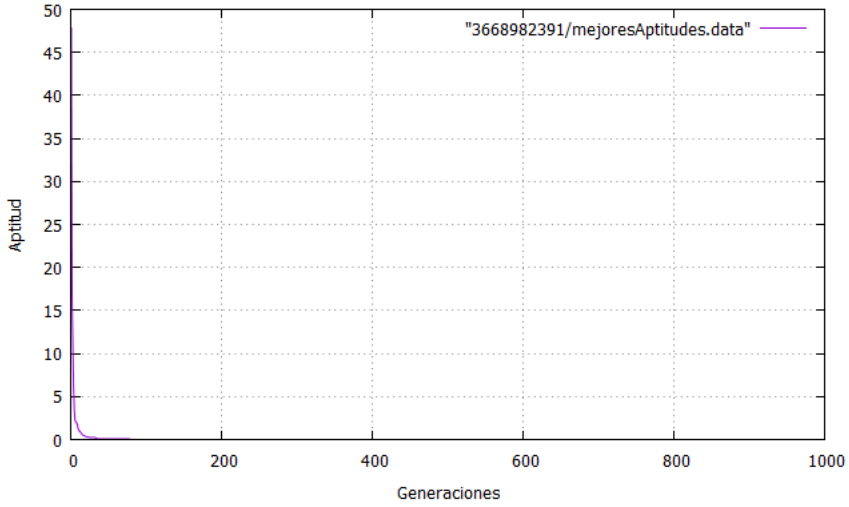

Figura 6.10: Evolución de la mejor aptitud del Caso 1 con el prototipo 2 del algoritmos OEP

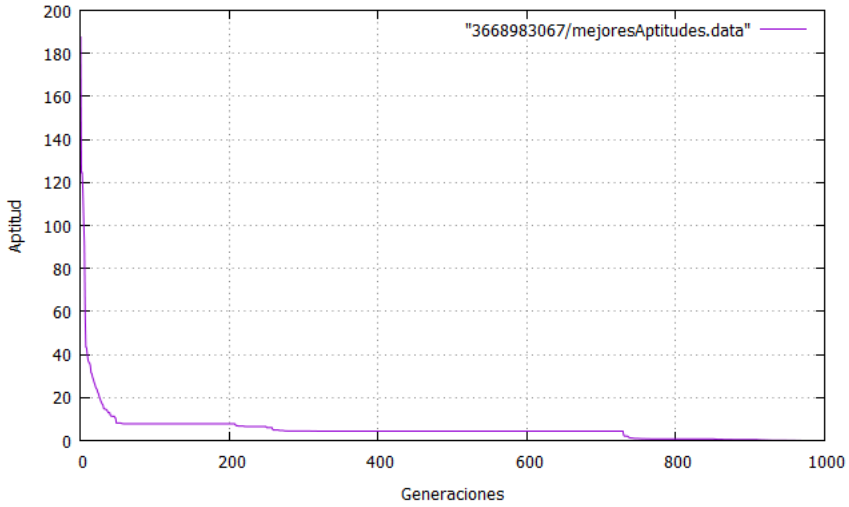

Figura 6.11: Evolución de la mejor aptitud del Caso 2 con el prototipo 2 del algoritmos OEP

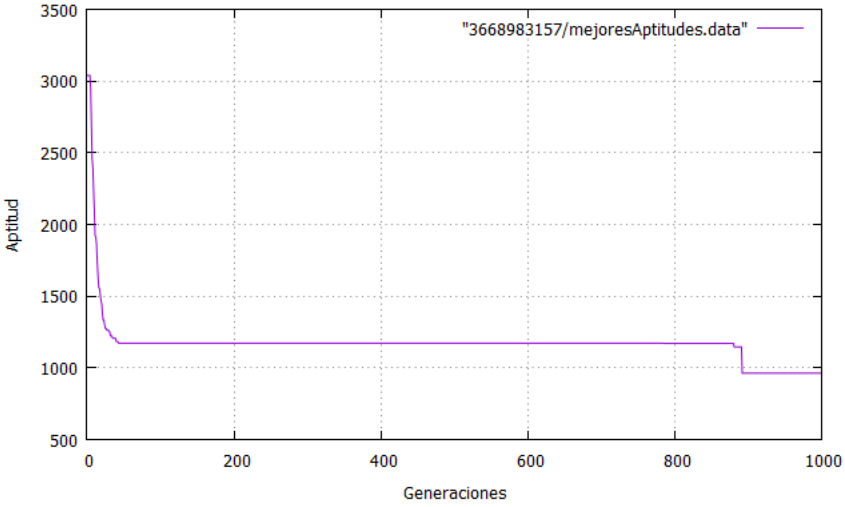

Figura 6.12: Evolución de la mejor aptitud del Caso 3 con el prototipo 2 del algoritmos OEP

Al comparar entre los resultados obtenidos el más cercano al optimo y el más lejano a este entre las soluciones conseguidas por el algoritmo OEP haciendo uso de la función 6.1 y el valor de χ_t^2 de las tablas en el Apéndice A se obtiene el rango de la probabilidad de que tan acertadas son las soluciones proporcionadas por el algoritmo, estas se pueden consultar en la tabla 6.10.

El caso 1 corresponde a la distribución de 5 resultados posibles, como se observa en la tabla la solución más cercana al optimo global dio una certeza del 99.9 % obtenida en varias configuraciones del algoritmo, mientras que la más lejana dio un resultado de 99.5 %, esta se obtuvo en la configuración del algoritmo con una población de 10 partículas en 100 iteraciones, además en la gráfica 6.10 se observa que la convergencia al optimo global se realizo de manera rápida.

En el caso de los 10 resultados posibles la mejor solución encontrada dio también una certeza del 99.9 %, sin embargo esta solución solo fue alcanzada por la configuración con 1000 partículas en mil generaciones, que en contraste con la mejor solución encontrada en el caso uno con una población 100 veces menor en su primera coincidencia con este valor de certeza, indica una gran diferencia en los parámetros necesarios para la búsqueda de la mejor solución entre distribuciones con números de posibles resultados diferentes. Esto queda más claro al ver que de las soluciones encontradas para este caso la más lejana al optimo obtuvo solo una certeza del 12.5 %.

En el último caso las configuraciones del algoritmo solo demostraron alcanzar una certeza de la solución del 1.0 %, lo que indica que se necesitan realizar grandes cambios en los parámetros de este algoritmo para grupos de posibles resultados grandes.

Tabla 6.10: Rangos de probabilidades conseguidos con el prototipo 2 del algoritmo OEP

Casos	1	2	3
Resultado más Cercano	99.9 %	99.9 %	1.0 %
Resultado menos Cercano	99.5 %	12.5 %	0.0 %

6.2.2. Experimentación Algoritmo ED

Los resultados de las pruebas con las 4 versiones del algoritmo ED se encuentran en las tablas 6.11, 6.12, 6.13 y 6.14 se puede observar igual que en el experimento 1, las versiones con el operador binario de cruza dieron mejores resultados que el operador exponencial.

Tabla 6.11: Mejores aptitudes reportadas por el prototipo 2 del algoritmos ED/-Mejor/1/bin

Población	Ciclos	Caso 1	Caso 2	Caso 3
10	100	0.37239	51.1856	3440.6248
	500	2.8997e-4	6.265629	2103.278
	1000	4.6507e-4	0.372331	1063.4875
50	100	0.00455	1.281826	1549.4636
	500	1.5817e-4	0.062236	479.92526
	1000	4.718e-5	0.101887	470.43286
100	100	0.003778	0.941699	1117.4059
	500	9.1068e-6	0.042878	602.8561
	1000	1.3477e-6	0.004182	205.6526

Tabla 6.12: Mejores aptitudes reportadas por el prototipo 2 del algoritmos ED/-Mejor/1/exp

Población	Ciclos	Caso 1	Caso 2	Caso 3
10	100	0.738399	335.59412	3546.211
	500	2.5500133	117.304596	4475.4316
	1000	3.740443	213.88144	8368.84
50	100	6.947045	22.496933	2852.8418
	500	3.044786	108.47366	4615.2417
	1000	0.056134	58.77605	2953.9233
100	100	0.461887	149.82414	2943.6738
	500	0.574576	82.44696	1688.0438
	1000	0.2530356	40.661488	2897.629

Tabla 6.13: Mejores aptitudes reportadas por el prototipo 2 del algoritmos ED/Aleatorio/1/bin

Población	Ciclos	Caso 1	Caso 2	Caso 3
10	100	0.328756	94.20679	7740.537
	500	5.125031e-5	2.366275	3534.3108
	1000	4.70059e-7	0.021181	1683.1412
50	100	0.2936241	43.81505	4341.3647
	500	3.9482e-5	0.3502225	2761.6998
	1000	7.5535e-9	0.005363	1398.5867
100	100	0.188669	20.72014	3040.5217
	500	3.027e-5	0.66922	2464.4692
	1000	2.5948e-8	0.009641	1422.8671

Tabla 6.14: Mejores aptitudes reportadas por el prototipo 2 del algoritmos ED/Aleatorio/1/exp

Población	Ciclos	Caso 1	Caso 2	Caso 3
10	100	2.84725	299.92618	5296.722
	500	4.186247	197.52345	6695.4824
	1000	2.710254	220.95445	6392.8496
50	100	0.40225	121.07112	5287.3925
	500	1.494259	39.50338	5507.946
	1000	0.27307	90.15931	4108.427
100	100	2.168944	103.39682	3503.2876
	500	1.189528	83.45371	4451.8833
	1000	0.1581136	115.9488	4554.3496

Tabla 6.15: Mejores iteraciones del prototipo 2 del algoritmo ED

Casos	1	2	3
Versión	*ED/Aleatorio/1/bin*	*ED/Mejor/1/bin*	*ED/Mejor/1/bin*
CodExp	3669050529	3669037614	3669037645
Población	50	100	100
Ciclos	1000	1000	1000
Mejor resultado	7.5535e-9	0.004182	205.6526

La diferencia entre las versiones de Mejor individuo e individuo Aleatorio mostraron menos diferencias que entre las de los operadores, aunque las versiones con Mejor individuo para ser perturbado mostró mejores resultados.

En el caso 1 la mejor solución se encontró en la versión *ED/Aleatorio/1/bin* en la configuración de población de tamaño 50 y 500 generaciones, en la versión *ED/Mejor/1/bin* con la configuración de población de 100 individuos en 1000 generaciones se encontró la mejor solución a los casos 2 y 3.

La evolución de la mejor aptitud en cada uno de los casos se ve en las gráficas 6.13, 6.14 y 6.15.

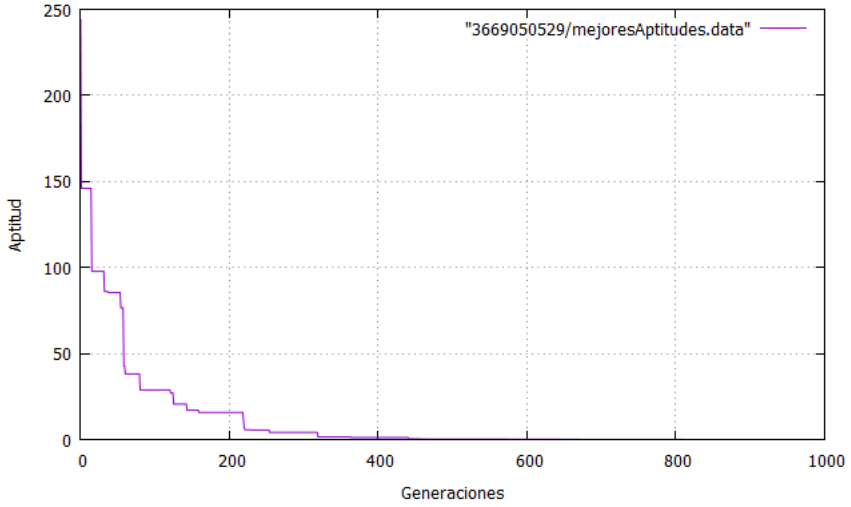

Figura 6.13: Evolución de la mejor aptitud del Caso 1 con el prototipo 2 del algoritmos ED

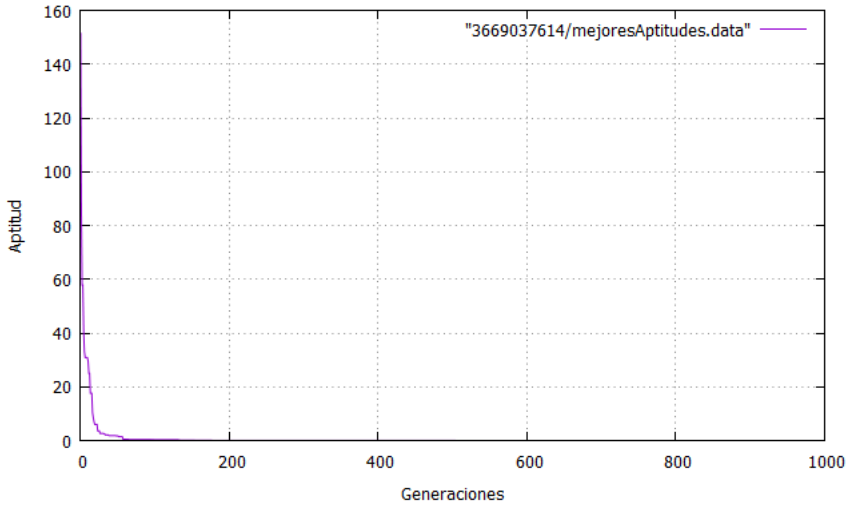

Figura 6.14: Evolución de la mejor aptitud del Caso 2 con el prototipo 2 del algoritmos ED

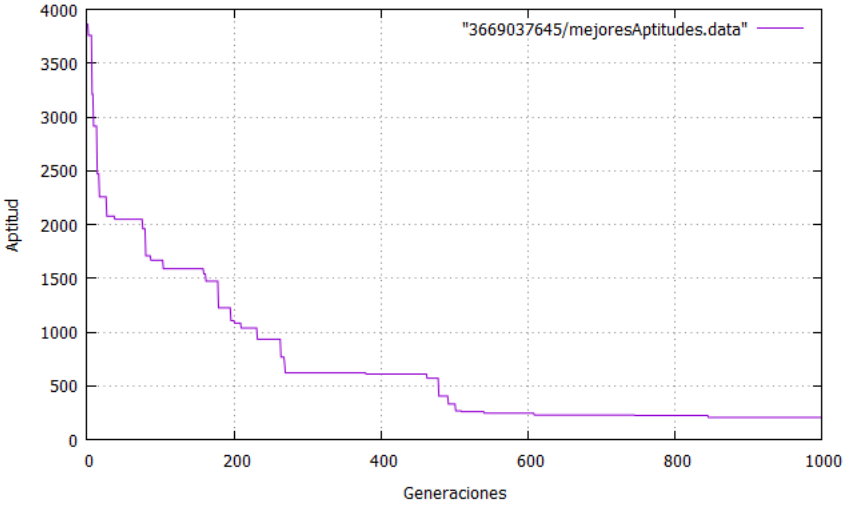

Figura 6.15: Evolución de la mejor aptitud del Caso 3 con el prototipo 2 del algoritmos ED

En la tabla 6.16 se observa el rango de probabilidad de certeza que se obtuvo con este algoritmo, muestra los diferentes rangos obtenidos en cada una de las versiones del algoritmo. En esta se puede apreciar más claramente que la versión *ED/Mejor/1/bin* dio mejores resultados que las otras y dejo claro que el operador de cruza binario es el que ha dado mejores resultados a este algoritmo hasta el momento.

Tabla 6.16: Rangos de probabilidades conseguidos con el prototipo 2 del algoritmo ED

Versión	ED/Mejor/1/bin		
Casos	1	2	3
Resultado más Cercano	99.9 %	99.9 %	99.5 %
Resultado menos Cercano	99.5 %	75.0 %	0.0 %
Versión	ED/Mejor/1/exp		
Casos	1	2	3
Resultado más Cercano	99.9 %	97.5 %	0.0 %
Resultado menos Cercano	75.0 %	0.0 %	0.0 %
Versión	ED/Aleatorio/1/bin		
Casos	1	2	3
Resultado más Cercano	99.9 %	99.9 %	0.0 %
Resultado menos Cercano	95.5 %	25.0 %	0.0 %
Versión	ED/Aleatorio/1/exp		
Casos	1	2	3
Resultado más Cercano	99.9 %	87.5 %	0.0 %
Resultado menos Cercano	80.0 %	0.0 %	0.0 %

A pesar de los buenos resultados para el caso 1 en las diferentes versiones del algoritmo en el rango de probabilidad de aceptación en los casos 2 y 3 este rango es muy muy amplio en el primero y el segundo las mejores resultados dieron un muy mal porcentaje de aceptación. Por lo que se deben utilizar parámetros más grandes para probar encontrar mejores soluciones a estos.

6.2.3. Experimentación Algoritmo EE

Para esta ejecución del algoritmo EE, se tomaron en cuenta los resultados obtenidos en el experimento uno, por lo que se probaron grupos de poblaciones más grandes y generaciones también, ademas de que en todas las ejecuciones se utilizo la generación de 7 hijos por individuo como se sugiere en la literatura como se menciona en 5.3.

La tabla 6.17 muestra los resultados de esta ejecución con cada una de las configuraciones del algoritmo y la tabla 6.18 muestra las mejores soluciones encontradas.

Tabla 6.17: Mejores aptitudes reportadas por el prototipo 2 del algoritmos EE

Versión		$(\mu+\lambda)-EE$		
Población	Generaciones	Caso 1	Caso 2	Caso 3
10	500	7.0805585	188.329935	2408.012031
	1000	5.821375	119.179194	2404.830766
50	500	0.795793	144.413023	2183.51345
	1000	1.172876	134.597787	2376.55318
100	500	3.857796	130.236608	2387.145575
	1000	2.215798	102.050634	2406.015503
Versión		$(\mu,\lambda)-EE$		
Población	Generaciones	Caso 1	Caso 2	Caso 3
10	500	20.3784494	188.084282	2404.16569
	1000	20.798842	188.006275	2048.211721
50	500	20.791776	187.908928	2398.102766
	1000	20.80307	188.115074	2404.465737
100	500	20.786137	187.959814	2398.057588
	1000	20.794241	187.975126	2405.598889

Tabla 6.18: Mejores iteraciones del prototipo 2 del algoritmo EE

Casos	1	2	3
Versión	$(\mu+\lambda)-EE$	$(\mu+\lambda)-EE$	$(\mu,\lambda)-EE$
CodExp	3669246116	3669264953	3669246011
Población	50	100	10
Ciclos	500	1000	1000
Mejor resultado	0.795793	102.050634	2048.211721

Las gráficas 6.16, 6.17 y 6.18 muestran la evolución de la búsqueda de mejor

aptitud de los 3 casos. En la gráfica 6.18 muestra en un inicio un comportamiento caótico debido a la no supervivencia de individuos en la configuración $(\mu, \lambda) - EE$ del algoritmo, pero después de unas generaciones presenta una evolución uniforme.

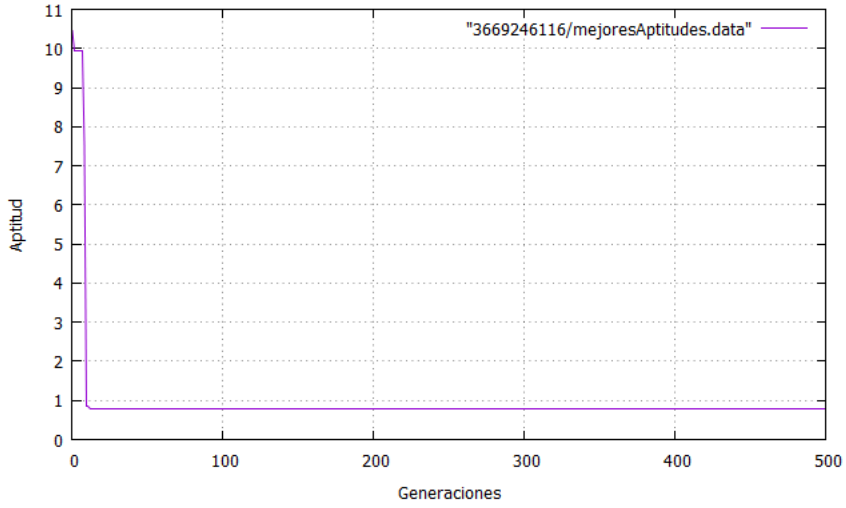

Figura 6.16: Evolución de la mejor aptitud del Caso 1 con el prototipo 2 del algoritmos EE

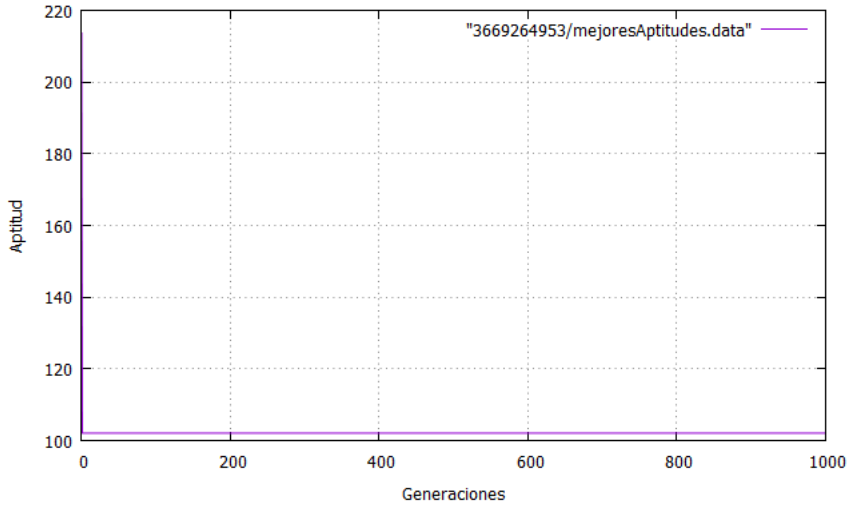

Figura 6.17: Evolución de la mejor aptitud del Caso 2 con el prototipo 2 del algoritmos EE

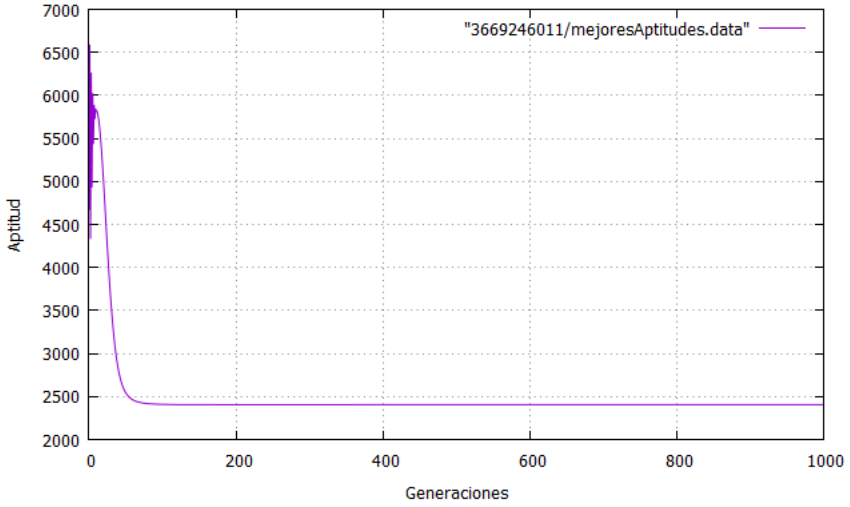

Figura 6.18: Evolución de la mejor aptitud del Caso 3 con el prototipo 2 del algoritmos EE

Los cambios en los parámetros mejoraron el rendimiento en el algoritmo al ver como en el caso 1 donde la función de optimización tiene 5 variables en la versión con sobrevivientes alcanzo una aptitud muy cercana al optimo global, sin embargo para los demás casos el desempeño del algoritmo no dio buenos resultados como se ve en la tabla 6.19, por lo que se deben de utilizar parámetros más grandes para los casos con un número más grande de variables.

Tabla 6.19: Rangos de probabilidades conseguidos con el prototipo 2 del algoritmo EE

Versión	$(\mu + \lambda) - EE$		
Casos	1	2	3
Resultado más Cercano	99.5 %	25.0 %	0.0 %
Resultado menos Cercano	75.0 %	1.0 %	0.0 %
Versión	$(\mu, \lambda) - EE$		
Casos	1	2	3
Resultado más Cercano	25.0 %	1.0 %	0.0 %
Resultado menos Cercano	25.0 %	1.0 %	0.0 %

6.2.4. Conclusiones del experimento 2

Se obtuvieron buenos resultados para la implementación de los algoritmos para la Prueba χ^2 de Pearson en los casos más sencillos, sin embargo para casos más complejos se deberá considerar un aumento considerable en los valores de

los parámetros de los algoritmos para futuras ejecuciones de los algoritmos.

Se puede concluir que los algoritmos fueron bien adaptados para funcionar con la prueba de Pearson como función objetivo pero se debe de ser más cuidadoso con los parámetros de los algoritmos dependiendo de la cantidad de posibles resultados de cada caso sobre el que se ejecuta.

6.3. Prototipo 3: Minimización Multiobjetivo

El objetivo de este experimento es probar el modelo de optimización multiobjetivo.

Como se menciono en 2.4, la optimización multiobjetivo se trata de la optimización de múltiples funciones al mismo tiempo, para este caso, se intentan optimizar varias Pruebas χ^2 de Pearson simultáneamente, en total son 9 grupos diferentes de distribuciones para esto, estas pueden consultarse en el Apéndice C.

Cada algoritmo busca el optimizar lo que serian varias observaciones de un mismo fenómeno al mismo tiempo con el objetivo de encontrar un hipótesis más general de la distribución de probabilidades de este fenómeno.

Para esta ejecución basándose en los resultados del experimento anterior se consideraron parámetros mayores para todos los algoritmos y la aptitud registrada en los experimentos es el promedio del resultado de las pruebas en cada distribución de cada grupo.

6.3.1. Experimentación Algoritmo OEP

Para el algoritmo OEP se consideraron valores de tamaño de población de 100 y 250 partículas en 1000, 2000 y 3000 generaciones, los resultados obtenidos están el la tabla 6.20 y la tabla 6.21 muestra los mejores resultados para cada grupo.

Tabla 6.20: Mejores aptitudes reportadas por el prototipo 3 del algoritmos OEP

Población	100			250		
Ciclos	1000	2000	3000	1000	2000	3000
Grupo 1	2.627294	2.603314	2.61651	2.60955	2.571719	2.570555
Grupo 2	2.282344	1.95674	1.946399	2.173505	2.172385	2.118113
Grupo 3	9.832045	8.528906	8.689489	11.536684	9.082963	9.560207
Grupo 4	4.243468	3.098725	2.596288	2.3787732	3.092928	3.41195
Grupo 5	5.81207	4.278241	5.131664	3.858564	3.860808	4.494706
Grupo 6	11.526077	13.652125	14.849123	10.071304	9.784254	13.126961
Grupo 7	186.53278	101.286446	4.252332	118.83316	67.820595	89.40673
Grupo 8	288.24762	5.575316	5.575651	252.76125	126.49319	148.32693
Grupo 9	590.6189	546.4438	470.83954	335.47086	11.573435	11.573425

Tabla 6.21: Mejores iteraciones del prototipo 3 del algoritmo OEP

Grupo	CodExp	Mejor Resultado	Población	Ciclos
1	3669690335	2.5705555	250	3000
2	3669431195	1.946399	100	3000
3	3669407706	8.528906	100	2000
4	3669599101	2.3787732	250	1000
5	3669599278	3.858564	250	1000
6	3669612851	9.784254	250	2000
7	3669432698	4.252332	100	3000
8	3669421757	5.575316	100	2000
9	3669658673/3669844810	11.573425	250	2000/3000

En la tabla 6.22 se reportan los rangos de aceptación promedios encontrados en esta ejecución. Puede observarse que en general el rendimiento de este algoritmo fue bastante bueno, a pesar de que el rango de probabilidades en los casos con mayor número de distribuciones que fueron los grupos 3, 6 y 9 aumento considerablemente y debe tenerse en cuenta en futuras ejecuciones del algoritmo para dar valor a los parámetros de este.

Tabla 6.22: Rangos de probabilidades conseguidos con el prototipo 3 del algoritmo OEP

Grupo	1	2	3	4	5	6	7	8	9
Más cercano	95.0 %	95.0 %	66.7 %	99.9 %	99.9 %	99.9 %	99.9 %	99.9 %	99.9 %
Menos cercano	95.0 %	95.0 %	50.0 %	99.9 %	99.9 %	99.5 %	99.9 %	97.5 %	40.0 %

Las gráficas 6.19, 6.20, 6.21, 6.22, 6.23, 6.24, 6.25, 6.26 y 6.27 se muestra la evolución de la aptitud de las mejores soluciones para cada grupo.

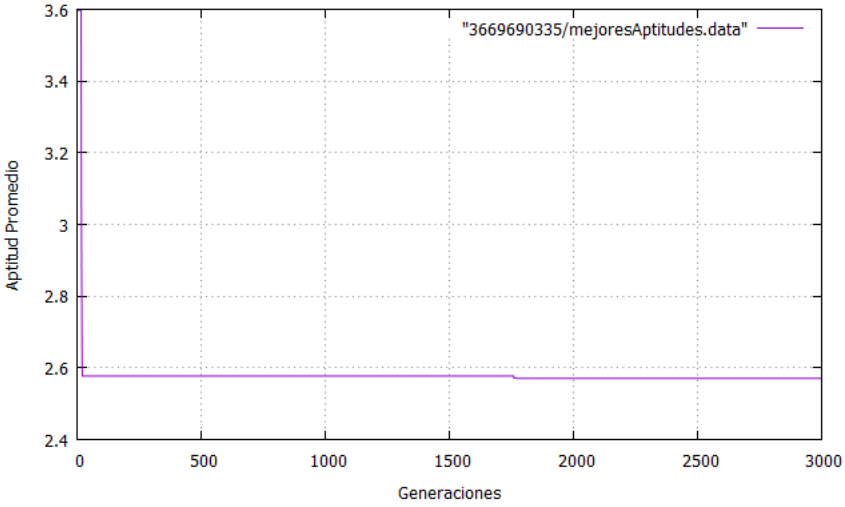

Figura 6.19: Evolución de la mejor aptitud del Grupo 1 con el prototipo 3 del algoritmos OEP

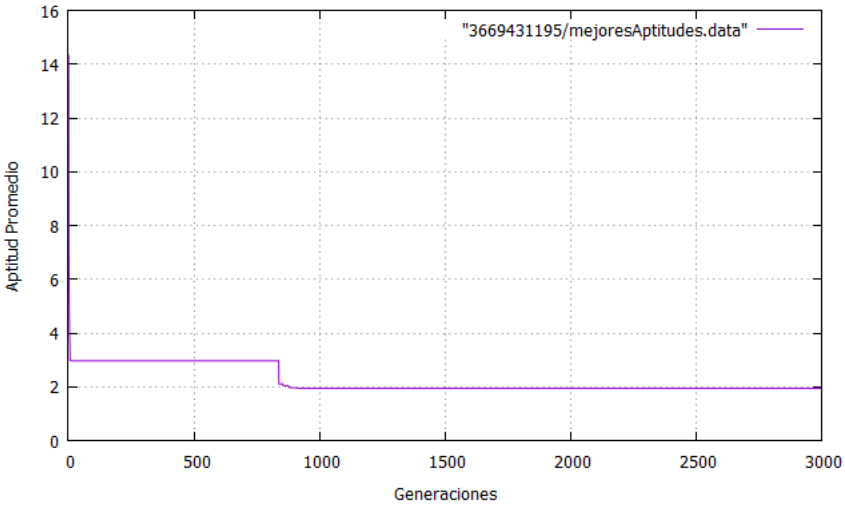

Figura 6.20: Evolución de la mejor aptitud del Grupo 2 con el prototipo 3 del algoritmos OEP

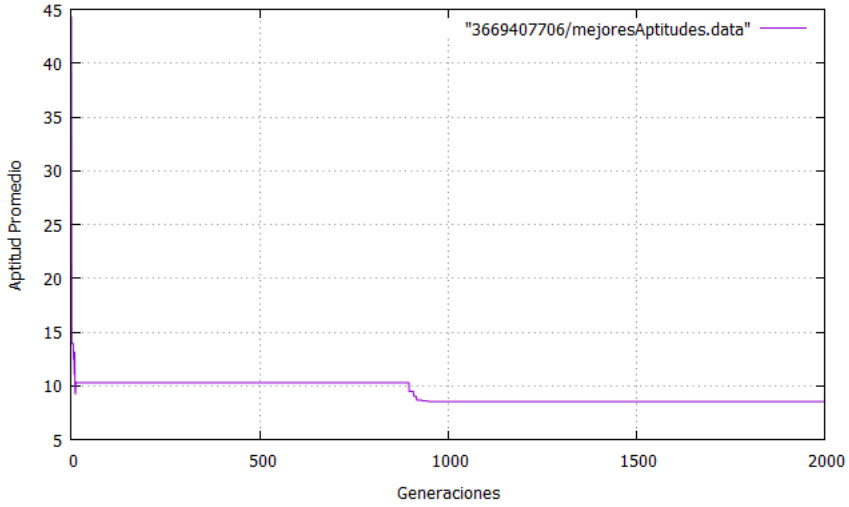

Figura 6.21: Evolución de la mejor aptitud del Grupo 3 con el prototipo 3 del algoritmos OEP

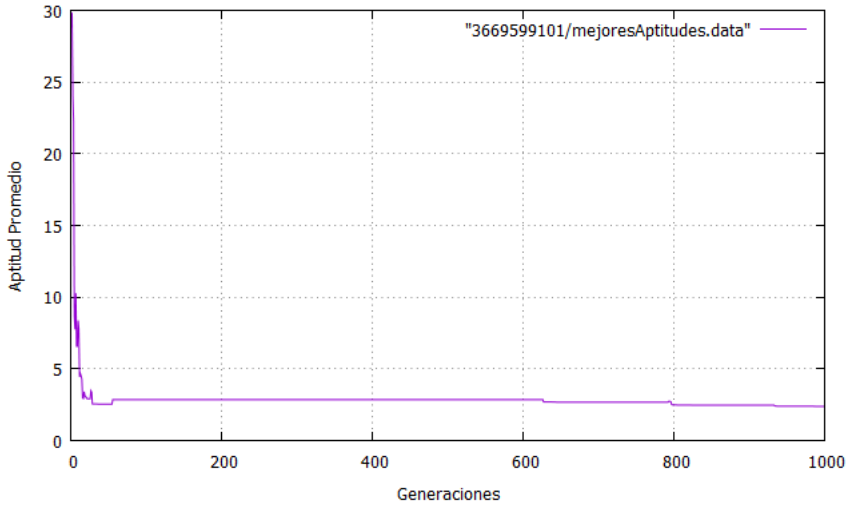

Figura 6.22: Evolución de la mejor aptitud del Grupo 4 con el prototipo 3 del algoritmos OEP

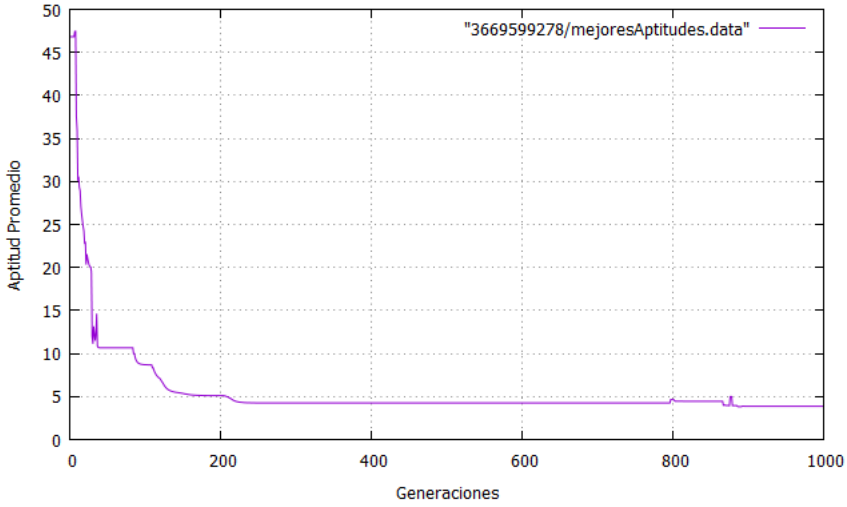

Figura 6.23: Evolución de la mejor aptitud del Grupo 5 con el prototipo 3 del algoritmos OEP

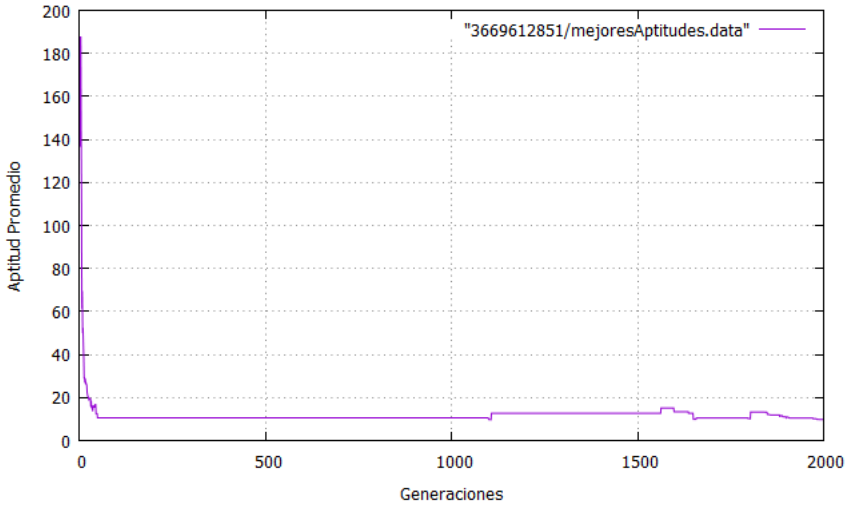

Figura 6.24: Evolución de la mejor aptitud del Grupo 6 con el prototipo 3 del algoritmos OEP

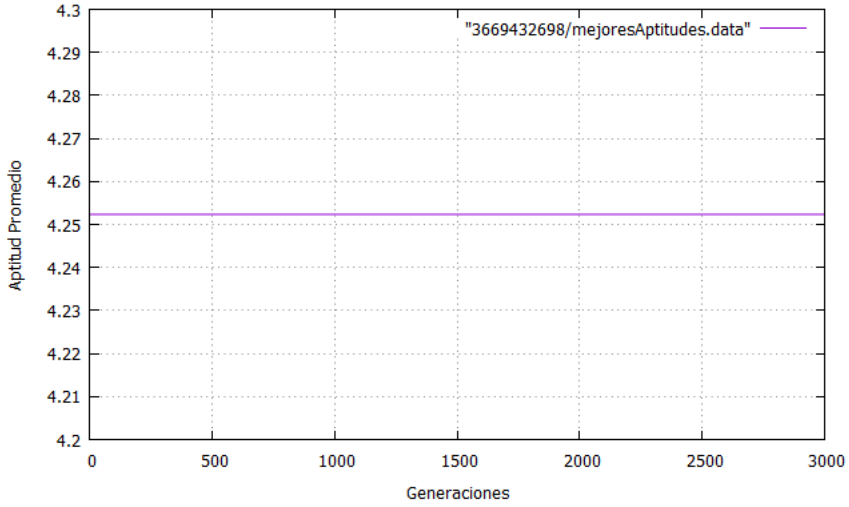

Figura 6.25: Evolución de la mejor aptitud del Grupo 7 con el prototipo 3 del algoritmos OEP

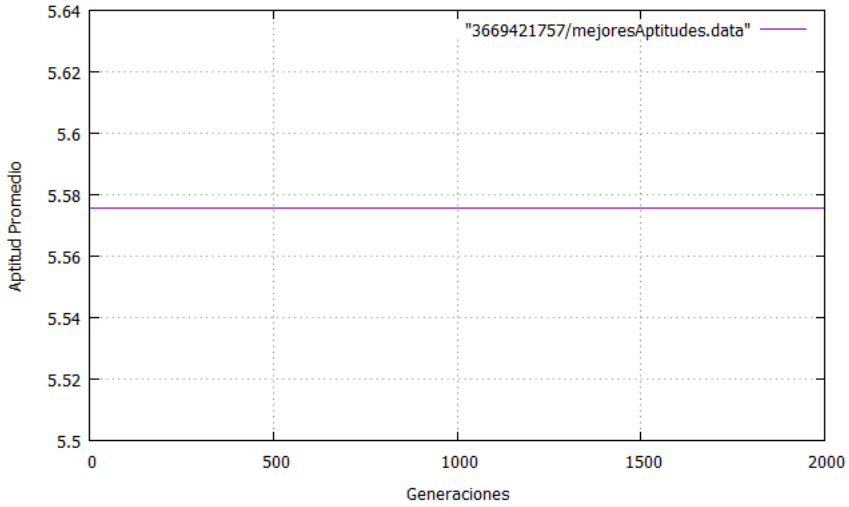

Figura 6.26: Evolución de la mejor aptitud del Grupo 8 con el prototipo 3 del algoritmos OEP

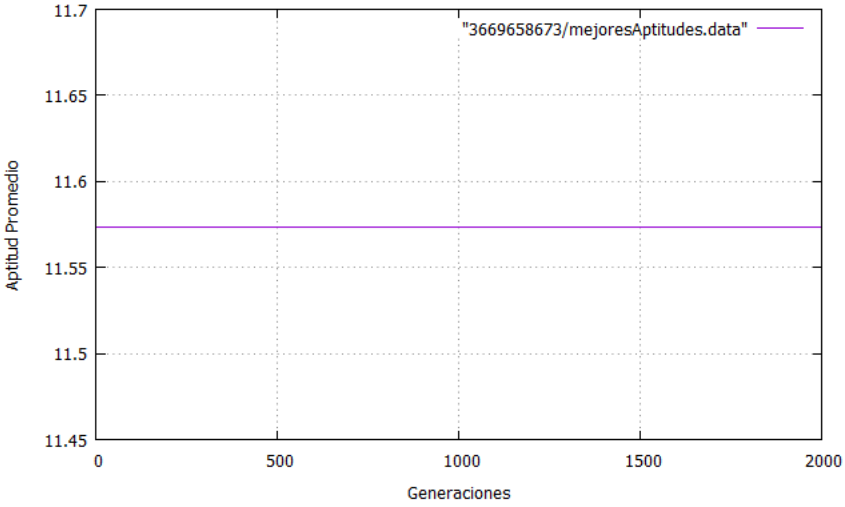

Figura 6.27: Evolución de la mejor aptitud del Grupo 9 con el prototipo 3 del algoritmos OEP

Puede apreciarse en algunas gráficas como por ejemplo en la grafica 6.23 que en la evolución de la aptitud hay algunos picos de subida y bajada en sus valores, esto es porque la elección de la mejor solución se hace de acuerdo al concepto de Optimo de Pareto (Def. 18), por lo que la mejor solución es la que mejor se ajusta para optimizar todas las pruebas de Pearson de cada grupo simultáneamente, por lo que el tener un mejor aptitud promedio no es un indicativo en este caso de que una solución sea mejor que otra.

De igual manera en las gráficas de los grupos 7 al 9 se aprecia como puede darse el caso de que una misma solución se mantenga mucho tiempo, es estos casos desde un inicio, y no exista mejoría durante el proceso.

6.3.2. Experimentación Algoritmo ED

Para la experimentación con el algoritmo ED se utilizaron los mismos parámetros en cuanto a tamaño de población y generaciones que con el algoritmo OEP.

En las tablas 6.23, 6.24, 6.25 y 6.26 se presentan los resultados de la aplicación del algoritmos en las diferentes configuraciones en sus cuatro versiones para los 9 grupos de distribuciones generados previamente.

Tabla 6.23: Mejores aptitudes reportadas por el prototipo 3 del algoritmos ED/-Mejor/1/bin

Población		100			250	
Ciclos	1000	2000	3000	1000	2000	3000
Grupo 1	2.564341	2.564329	2.564328	2.564327	2.564328	2.564327
Grupo 2	1.934205	1.934191	1.934191	1.934198	1.93419	1.93419
Grupo 3	8.196705	8.196642	8.196641	8.196642	8.196641	8.19664
Grupo 4	2.363969	2.356812	2.356422	2.3564105	2.355888	2.355748
Grupo 5	3.713551	3.708863	3.708318	3.709905	3.7083526	3.708252
Grupo 6	8.360806	8.35988	8.358495	8.35641	8.358387	8.357952
Grupo 7	45.775536	28.81774	17.979816	12.947618	3.506722	10.41239
Grupo 8	44.886593	38.422737	40.77715	40.429176	15.593128	11.321014
Grupo 9	161.7227	160.75174	75.33247	73.942505	38.7387	30.372185

Tabla 6.24: Mejores aptitudes reportadas por el prototipo 3 del algoritmos ED/-Mejor/1/exp

Población		100			250	
Ciclos	1000	2000	3000	1000	2000	3000
Grupo 1	3.295475	2.58637	3.06712	2.908667	2.754113	2.836349
Grupo 2	1.984237	1.981992	2.984336	2.15252	1.938354	2.021918
Grupo 3	12.229928	8.6662	9.209817	9.288602	8.221861	8.554353
Grupo 4	5.050975	35.98939	23.190533	32.655838	7.804738	18.554493
Grupo 5	71.4403	67.2619	36.50528	38.117643	19.032692	24.538149
Grupo 6	137.37283	92.87549	70.85508	113.30078	126.697	133.6469
Grupo 7	248.0934	479.45108	377.36578	287.47876	403.50308	220.87083
Grupo 8	708.84045	825.7925	803.3752	809.3801	5.504159	488.24683
Grupo 9	1652.2052	1524.3063	1390.2661	1220.1824	1350.5858	1095.6754

Tabla 6.25: Mejores aptitudes reportadas por el prototipo 3 del algoritmos ED/Aleatorio/1/bin

Población		100			250	
Ciclos	1000	2000	3000	1000	2000	3000
Grupo 1	2.564327	2.564327	2.564327	2.564327	2.564327	2.564327
Grupo 2	1.934194	1.93419	1.93419	1.934205	1.93419	1.93419
Grupo 3	8.19664	8.196639	8.196639	8.196642	8.193419	8.196638
Grupo 4	2.361154	2.355668	2.355669	2.378176	2.35567	2.355669
Grupo 5	3.715761	3.708212	3.708206	3.788642	3.708207	3.7082067
Grupo 6	8.369716	8.357856	8.357852	8.462266	8.357857	8.357853
Grupo 7	190.17647	64.85536	31.30705	154.00633	4.2523327	29.37281
Grupo 8	344.375	102.1742	53.861088	290.8767	114.40238	61.02335
Grupo 9	857.4102	264.3187	133.53918	585.95667	302.04303	107.47452

Tabla 6.26: Mejores aptitudes reportadas por el prototipo 3 del algoritmos ED/Aleatorio/1/exp

Población		100			250	
Ciclos	1000	2000	3000	1000	2000	3000
Grupo 1	3.2963314	3.739041	2.57277	2.904775	4.486316	2.752133
Grupo 2	2.025086	2.688282	2.850057	2.500149	2.245151	2.382659
Grupo 3	10.248367	8.585597	16.385347	9.769546	14.566411	10.353872
Grupo 4	40.475456	28.150867	29.812372	27.49585	11.236071	12.766271
Grupo 5	67.44884	59.846325	52.432014	50.5088	39.4511	28.613598
Grupo 6	37.385531	65.30812	71.72191	59.177223	34.507072	78.82737
Grupo 7	305.80368	388.07288	432.98978	390.0711	4.2523327	258.25916
Grupo 8	760.705	950.9168	763.307	846.891	475.03906	555.8849
Grupo 9	1849.3561	1443.255	2037.0934	1271.4847	1253.0757	11.573425

La tabla 6.27 muestra los mejores resultados obtenidos por el algoritmo ED en cada uno de los grupos y las gráficas 6.28, 6.29, 6.30, 6.31, 6.32, 6.33, 6.34, 6.35 y 6.36 muestran la evolución de la aptitud de estos.

Tabla 6.27: Mejores iteraciones del prototipo 3 del algoritmo ED

Grupo	CodExp	Versión	Aptitud	Población	Generaciones
1	Varios	Ambas con cruza binaria	2.564327	250, 100/250	1000/3000, 1000/2000/3000
2	Varios	Ambas con cruza binaria	1.93419	250, 100/250	2000/3000, 2000/3000
3	3669620816	ED/Aleatorio/1/bin	8.193419	250	2000
4	3669516151	ED/Aleatorio/1/bin	2.355668	100	2000
5	3669535057	ED/Aleatorio/1/bin	3.708206	100	3000
6	3669600656	ED/Mejor/1/bin	8.35641	250	1000
7	3669659071	ED/Mejor/1/bin	3.506722	250	2000
8	3669659260	ED/Mejor/1/exp	5.504159	250	2000
9	3669844915	ED/Aleatorio/1/exp	11.573425	250	3000

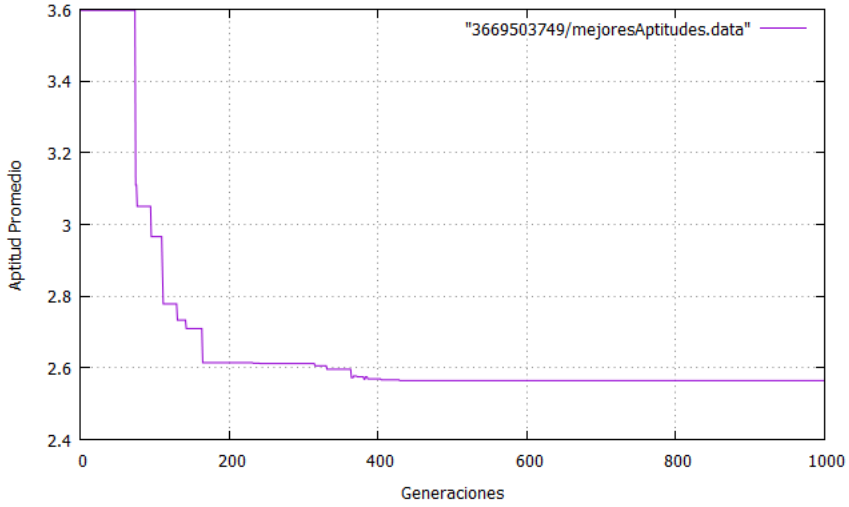

Figura 6.28: Evolución de la mejor aptitud del Grupo 1 con el prototipo 3 del algoritmos ED

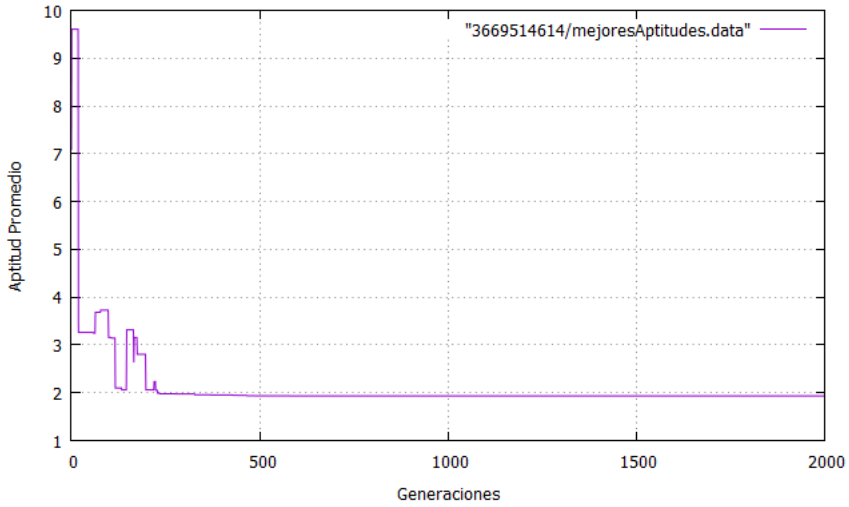

Figura 6.29: Evolución de la mejor aptitud del Grupo 2 con el prototipo 3 del algoritmos ED

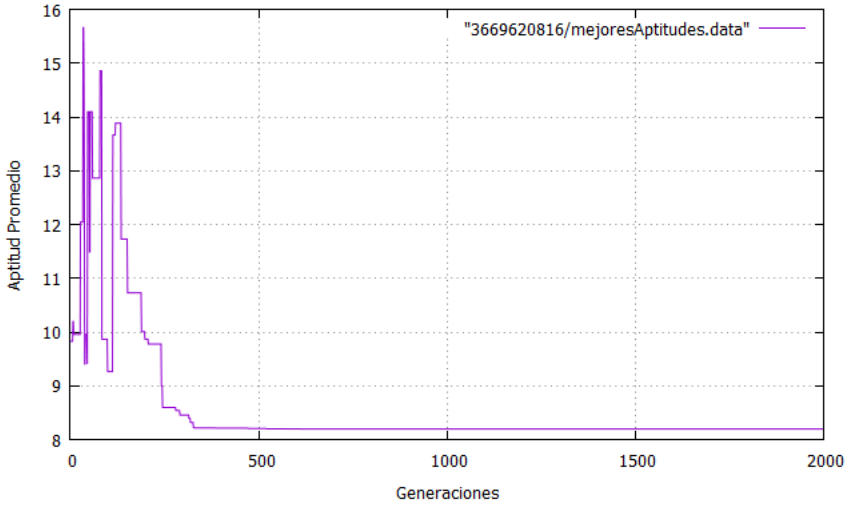

Figura 6.30: Evolución de la mejor aptitud del Grupo 3 con el prototipo 3 del algoritmos ED

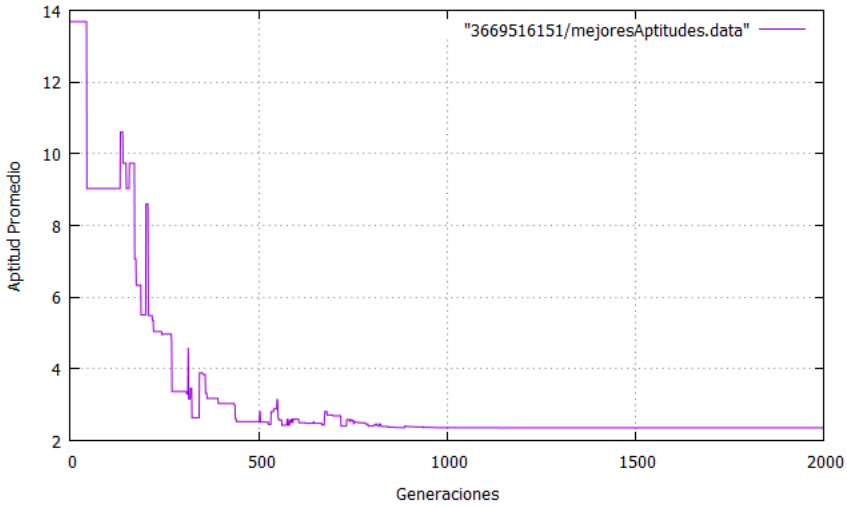

Figura 6.31: Evolución de la mejor aptitud del Grupo 4 con el prototipo 3 del algoritmos ED

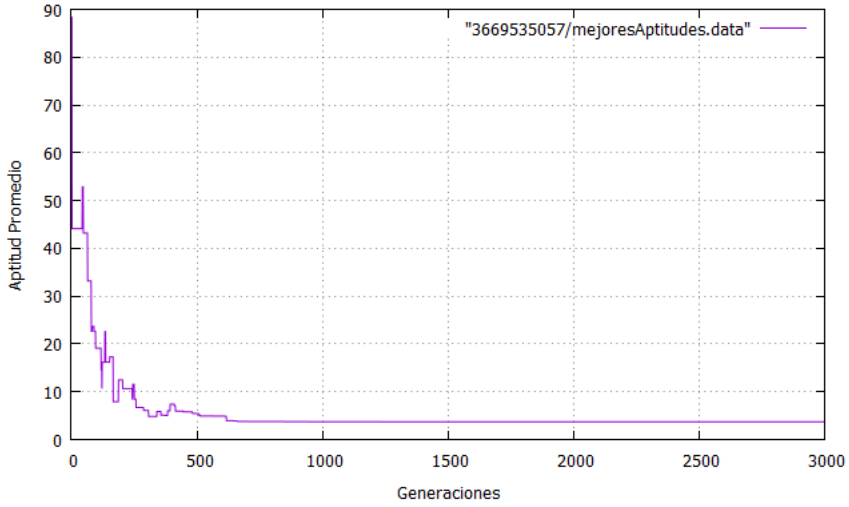

Figura 6.32: Evolución de la mejor aptitud del Grupo 5 con el prototipo 3 del algoritmos ED

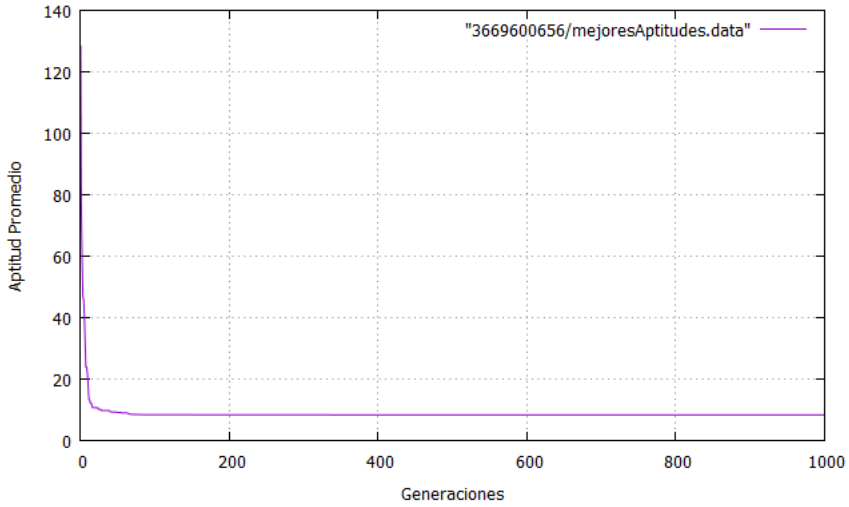

Figura 6.33: Evolución de la mejor aptitud del Grupo 6 con el prototipo 3 del algoritmos ED

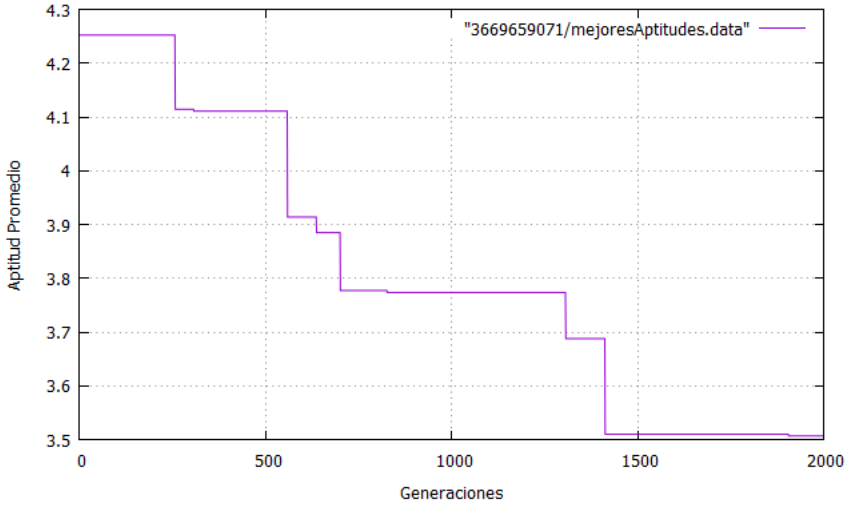

Figura 6.34: Evolución de la mejor aptitud del Grupo 7 con el prototipo 3 del algoritmos ED

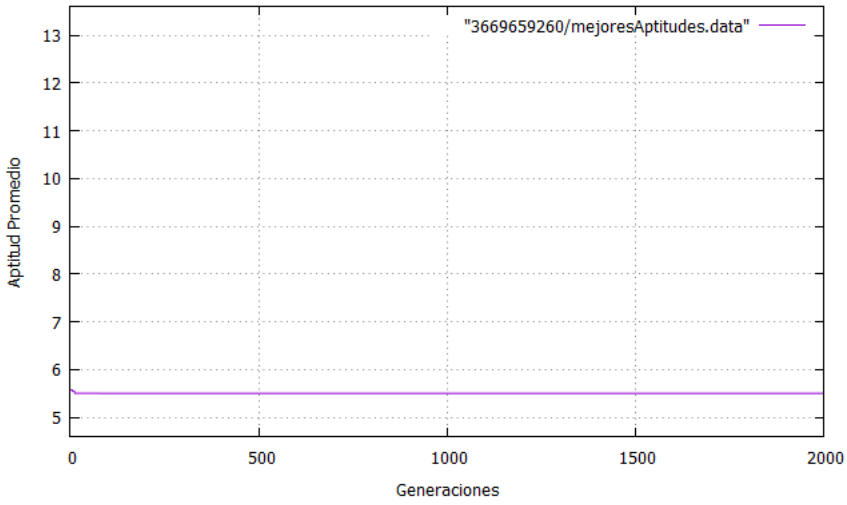

Figura 6.35: Evolución de la mejor aptitud del Grupo 8 con el prototipo 3 del algoritmos ED

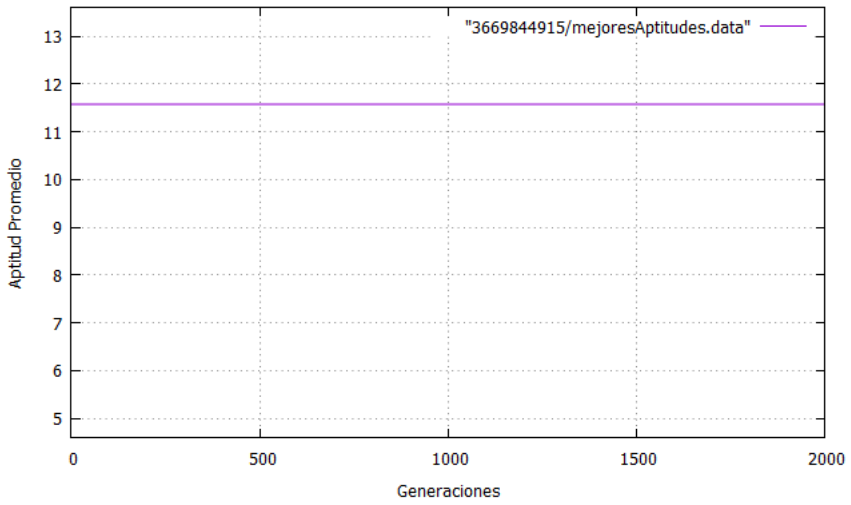

Figura 6.36: Evolución de la mejor aptitud del Grupo 9 con el prototipo 3 del algoritmos ED

En la tabla 6.28 se puede apreciar que las versiones del algoritmo con operador binario de cruza mostraron mejores resultados en todos los grupos y rangos más pequeños en la diferencia entre los mejores resultados obtenidos, la versión ED/Mejor/1/bin una ventaja en los grupos más grandes sobre el rango de las probabilidades que abarcan sus resultados.

Tabla 6.28: Rangos de probabilidades conseguidos con el prototipo 3 del algoritmo ED

Versión	ED/Mejor/1/bin								
Grupo	1	2	3	4	5	6	7	8	9
Más Cercano	95.0 %	97.5 %	66.7 %	99.9 %	99.9 %	99.9 %	99.9 %	99.9 %	99.9 %
Menos Cercano	95.0 %	97.5 %	66.7 %	99.9 %	99.9 %	99.9 %	99.9 %	99.9 %	99.9 %
Versión	ED/Mejor/1/exp								
Grupo	1	2	3	4	5	6	7	8	9
Más cercano	95.0 %	95.0 %	66.7 %	99.9 %	97.5 %	50.0 %	99.5 %	99.9 %	0.1 %
Menos cercano	90.0 %	95.0 %	50.0 %	90.0 %	50.0 %	5.0 %	66.7 %	5.0 %	0.1 %
Versión	ED/Aleatorio/1/bin								
Grupo	1	2	3	4	5	6	7	8	9
Más cercano	95.0 %	97.5 %	66.7 %	99.9 %	99.9 %	99.9 %	99.9 %	99.9 %	99.9 %
Menos cercano	95.0 %	97.5 %	66.7 %	99.9 %	99.9 %	99.9 %	99.9 %	90.0 %	5.0 %
Versión	ED/Aleatorio/1/exp								
Grupo	1	2	3	4	5	6	7	8	9
Más cercano	95.0 %	95.0 %	66.7 %	99.5 %	95.0 %	90.0 %	99.9 %	66.7 %	99.9 %
Menos cercano	87.5 %	90.0 %	33.3 %	87.5 %	50.0 %	40.0 %	80.0 %	1.0 %	0.1 %

6.3.3. Experimentación Algoritmo EE

Para este experimento se mantuvo la generación de 7 hijos por individuo, se programo que los algoritmos utilizaran configuraciones de 20 y 50 indivi6duos de tamaño de población en 1000, 2000 y 3000 generaciones. En la tabla 6.29 se muestran los resultados de ejecución de este algoritmo.

Tabla 6.29: Mejores aptitudes reportadas por el prototipo 3 del algoritmos EE

Versión	$(\mu + \lambda) - EE$					
Población	20			50		
Ciclos	1000	2000	3000	1000	2000	3000
Grupo 1	3.596335	3.596335	5.515029	3.596335	3.596335	3.596335
Grupo 2	10.561561	10.561561	11.372815	5.175407	5.175407	5.175407
Grupo 3	20.273205	20.273205	20.601502	31.577662	31.577662	31.577662
Grupo 4	44.178521	44.178521	36.136444	10.757705	27.345397	3.659672
Grupo 5	82.334551	82.334551	78.145123	42.199831	27.345297	82.334551
Grupo 6	180.759554	180.759554	118.50987	193.034089	62.157133	118.509887
Grupo 7	4.252332	263.070756	4.252332	249.382719	205.766243	253.519715
Grupo 8	513.395827	598.395567	513.395827	522.372924	514.4157	519.165144
Grupo 9	1077.023087	1041.430977	11.573425	1041.430977	1053.497680	1055.548627
Versión	$(\mu, \lambda) - EE$					
Población	20			50		
Ciclos	1000	2000	3000	1000	2000	3000
Grupo 1	11.086605	11.082893	11.077723	11.074525	11.077587	11.078493
Grupo 2	20.999978	20.999985	21.002386	21.008705	21.003824	21.002386
Grupo 3	49.172014	49.165294	49.163635	49.231054	49.160936	49.179274
Grupo 4	48.873671	48.881767	48.896519	48.890467	48.876336	48.879346
Grupo 5	100.017832	99.976082	99.926364	99.957021	99.895739	99.926364
Grupo 6	211.665649	211.735784	211.776402	211.544034	211.77541	211.765575
Grupo 7	287.940438	288.139646	288.067193	287.7701	287.770105	288.070113
Grupo 8	598.172992	598.194637	598.259145	598.386148	598.205785	598.3218
Grupo 9	1181.187788	1181.031336	1181.060762	1181.045457	1181.060762	1181.1554

Los mejores resultados obtenidos por este algoritmo pueden apreciarse en la tabla 6.30 y en la tabla 6.31 se puede apreciar los rangos del porcentaje de aceptación obtenidos por el algoritmo EE para este caso y las gráficas **??** a la **??** muestran la evolución de la mejor aptitud promedio para cada grupo en el caso de la mejor solución obtenida.

Tabla 6.30: Mejores iteraciones del prototipo 3 del algoritmo EE

Grupo	CodExp	Versión	Aptitud	Población	Generaciones
1	Varios	$(\mu + \lambda) - EE$	3.596335	20 y 50	1000/2000 y 1000/2000/3000
2	Varios	$(\mu + \lambda) - EE$	5.175407	50	1000, 2000 y 3000
3	Varias	$(\mu + \lambda) - EE$	20.273205	20	1000 y 2000
4	3670290541	$(\mu + \lambda) - EE$	3.659672	50	3000
5	3670325612	$(\mu + \lambda) - EE$	27.345297	50	2000
6	3670341421	$(\mu + \lambda) - EE$	62.157133	50	2000
7	3669856088	$(\mu + \lambda) - EE$	4.252332	20	1000
8	Varios	$(\mu + \lambda) - EE$	513.395827	20	1000/3000
9	3670205096	$(\mu + \lambda) - EE$	11.573425	20	3000

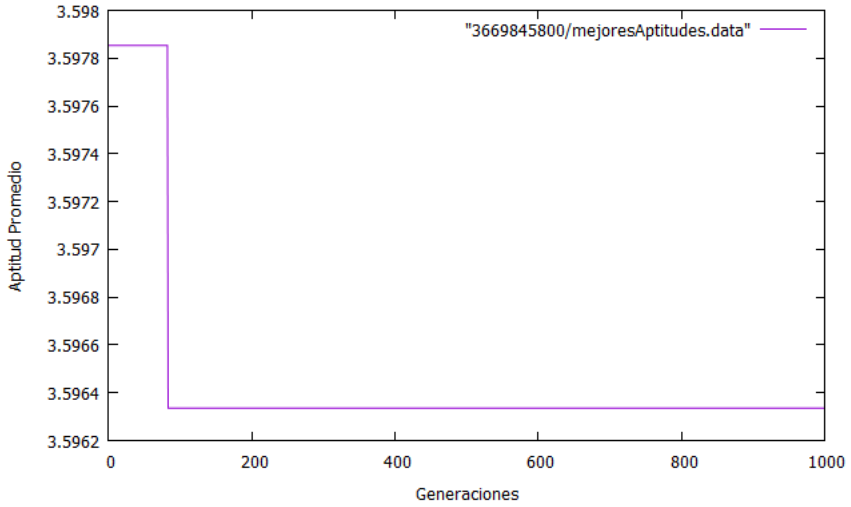

Figura 6.37: Evolución de la mejor aptitud del Grupo 1 con el prototipo 3 del algoritmos EE

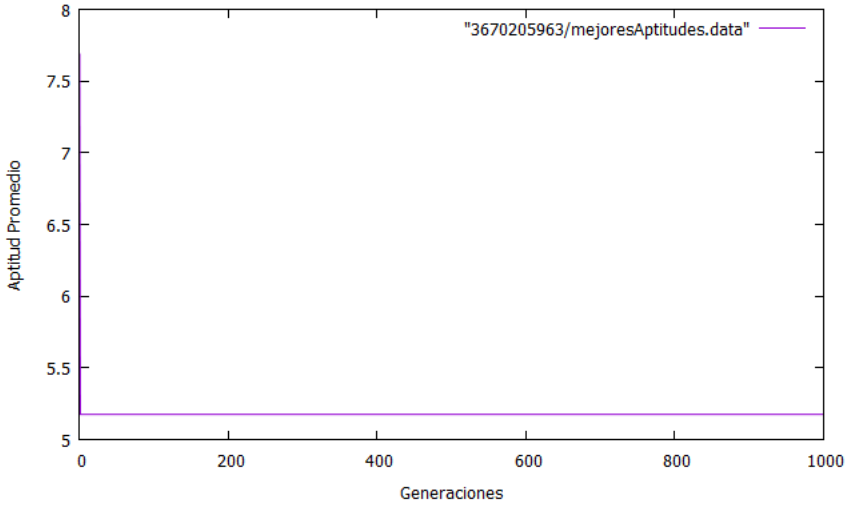

Figura 6.38: Evolución de la mejor aptitud del Grupo 2 con el prototipo 3 del algoritmos EE

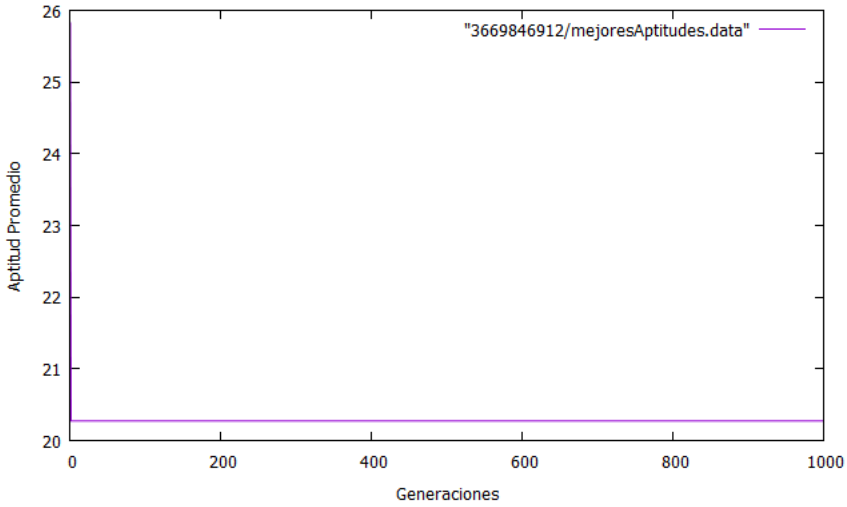

Figura 6.39: Evolución de la mejor aptitud del Grupo 3 con el prototipo 3 del algoritmos EE

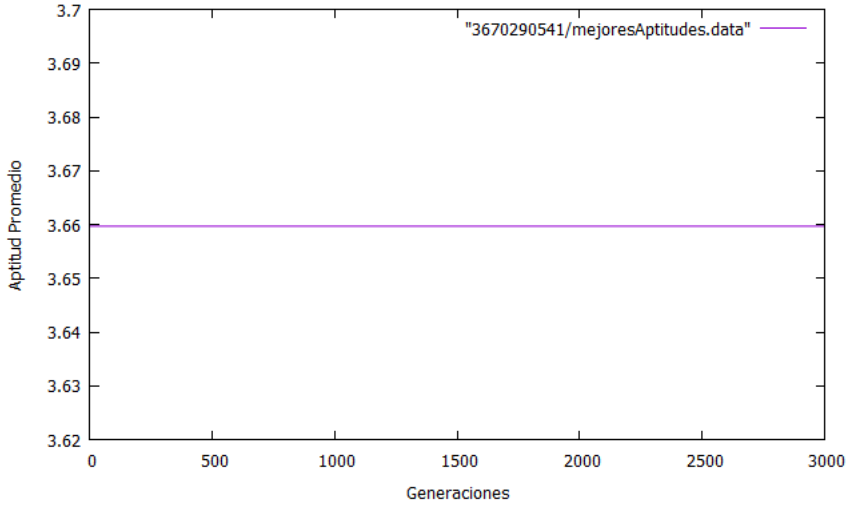

Figura 6.40: Evolución de la mejor aptitud del Grupo 4 con el prototipo 3 del algoritmos EE

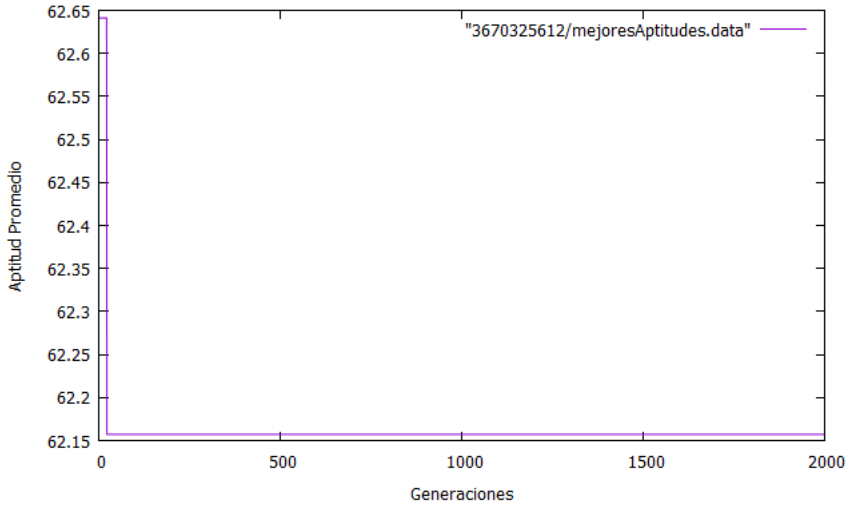

Figura 6.41: Evolución de la mejor aptitud del Grupo 5 con el prototipo 3 del algoritmos EE

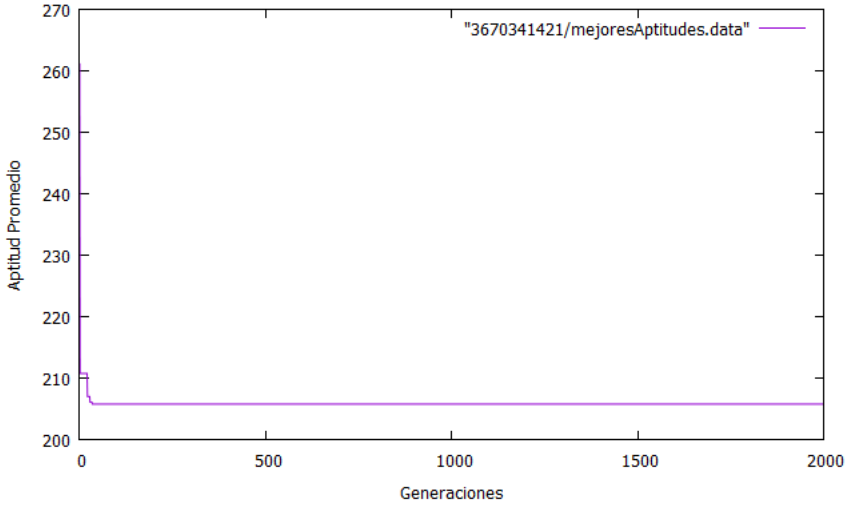

Figura 6.42: Evolución de la mejor aptitud del Grupo 6 con el prototipo 3 del algoritmos EE

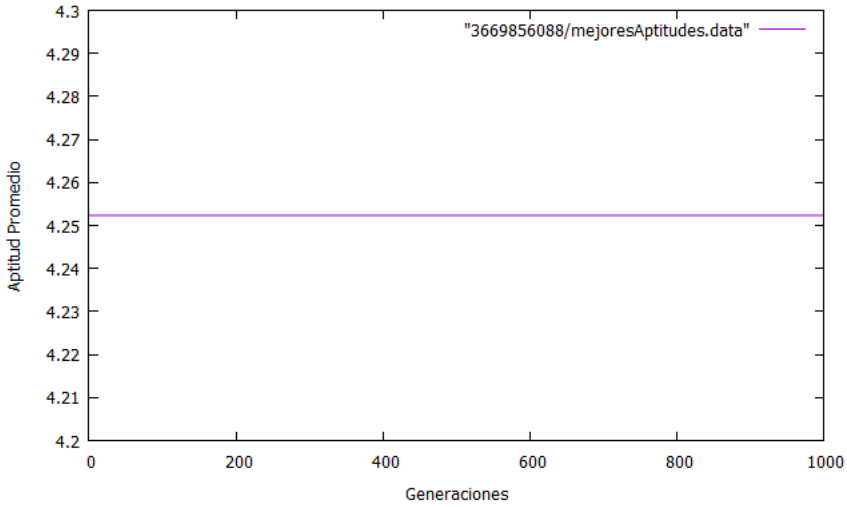

Figura 6.43: Evolución de la mejor aptitud del Grupo 7 con el prototipo 3 del algoritmos EE

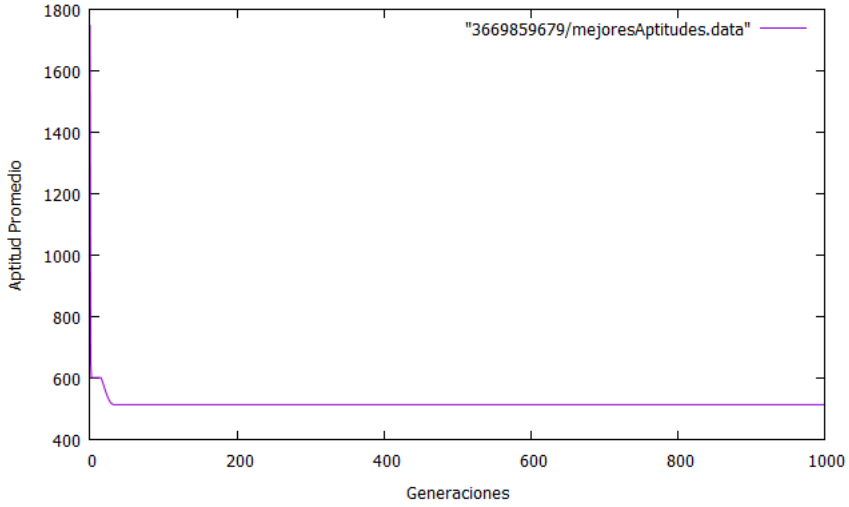

Figura 6.44: Evolución de la mejor aptitud del Grupo 8 con el prototipo 3 del algoritmos EE

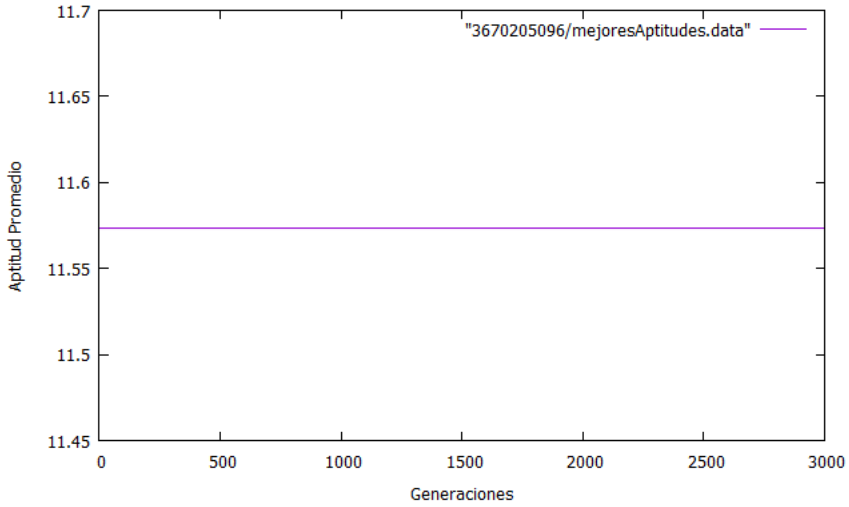

Figura 6.45: Evolución de la mejor aptitud del Grupo 9 con el prototipo 3 del algoritmos EE

Tabla 6.31: Rangos de probabilidades conseguidos con el prototipo 3 del algoritmo EE

Versión	$(\mu+\lambda)-EE$								
Grupo	1	2	3	4	5	6	7	8	9
Más Cercano	90.0 %	80.0 %	25.0 %	99.9 %	95.0 %	66.7 %	99.9 %	50.0 %	99.9 %
Menos Cercano	80.0 %	50.0 %	5.0 %	80.0 %	40.0 %	1.0 %	97.5 %	20.0 %	0.5 %
Versión	$(\mu,\lambda)-EE$								
Grupo	1	2	3	4	5	6	7	8	9
Más cercano	50.0 %	25.0 %	1.0 %	75.0 %	25.0 %	0.5 %	97.5 %	40.0 %	0.1 %
Menos cercano	50.0 %	25.0 %	1.0 %	75.0 %	25.0 %	0.5 %	97.5 %	40.0 %	0.1 %

Este algoritmo en su versión con sobrevivientes consiguió buenos resultados en los casos con menos distribuciones, aunque presentan un rango de probabilidades medianamente grande, que es importante considerar cuando se hace uso de este algoritmo.

6.3.4. Conclusiones del experimento 3

En general los tres algoritmos tuvieron un desempeño bueno, siendo el algoritmo EE el que dio el menor rendimiento, aunque no del todo malo, por lo que se debe considerar cambios leves a los parámetros utilizados en esta iteración para lograr mejorar ese rendimiento en el algoritmo.

6.4. Prototipo 4: Criterios de Paro

Como se observo en gráficas de la evolución de aptitudes en los experimentos anteriores, es común que la mejor solución sea obtenida antes de la ultima generación del algoritmo, por lo que es en muchos casos importante pasar definir condiciones de paro para el algoritmo que el total de las generaciones que se le programo para su ejecución.

Para este experimento se le pasara a los algoritmos un parámetro extra, este es un número que corresponde a al valor en la tabla de la distribución χ^2 que corresponda al los grados de libertad y el porcentaje de aceptación que se espera encontrar y este utilizara la función 6.1 para confirmar esto.

Los valores que serán pasados a los algoritmos dependen de cada grupo, estos se muestran en la tabla 6.32 y son para verificar una certeza del 90.0 % de las respuestas obtenidas.

Tabla 6.32: Valores del parámetro de paro para los algoritmos en el experimento 4

Grupos	1,2 y 3	4, 5 y 6	7, 8 y 9
Valor	1.064	4.168	4.168
Grados de libertad	4	9	24

Cada algoritmo se ejecutara con una misma configuración 5 veces sobre cada grupo para verificar el funcionamiento de este criterio de paro para este problema.

6.4.1. Experimentación Algoritmo OEP

Para el algoritmo OEP se utilizo una configuración de una población de 250 partículas en 1500 generaciones, en la tabla 6.33 se reportan los resultados de la aplicación del algoritmo 5 veces con la misma configuración en cada grupo de pruebas, puede verse que en la mayoría de los casos el algoritmo hizo uso de todas las generaciones para la búsqueda y en otros hizo uso de pocas generaciones para esto, lo que deja ver que la forma en que se generan aleatoriamente las poblaciones de las partículas influye en el la forma y velocidad en que estas recorren el espacio de búsqueda.

Tabla 6.33: Mejores aptitudes reportadas por el prototipo 4 del algoritmos OEP

Aptitud Promedio					
Ejecución	1	2	3	4	5
Grupo 1	2.604136	2.5657802	2.591746	2.571466	2.646968
Grupo 2	2.532383	2.6759	2.198274	8.525844	8.61149
Grupo 3	9.70403	8.803237	8.655015	6.280862	7.724787
Grupo 4	8.721284	7.84239	7.174182	14.845964	14.762204
Grupo 5	16.755777	18.144108	12.655541	26.77409	30.533518
Grupo 6	27.033224	29.53132	22.283041	97.243866	79.95087
Grupo 7	106.44413	76.99966	98.147255	224.27441	141.98338
Grupo 8	132.72305	207.75421	223.5186	296.56546	450.3948
Grupo 9	394.34586	359.16376	435.52148	11.573425	322.13086
Número de Generaciones					
Ejecución	1	2	3	4	5
Grupo 1	1500	1500	1500	1500	1500
Grupo 2	7	9	1	1500	1500
Grupo 3	1500	1500	1500	13	15
Grupo 4	7	6	10	12	9
Grupo 5	15	11	7	20	15
Grupo 6	27	9	49	1500	1500
Grupo 7	1500	1500	1500	1500	1500
Grupo 8	1500	1500	1500	1500	1500
Grupo 9	1500	1500	1500	1	1500

La tabla 6.34 muestra las mejores soluciones de este algoritmo y la tabla 6.35 el rango de probabilidades obtenidos en cada caso.

Tabla 6.34: Mejores iteraciones del prototipo 4 del algoritmo OEP

Grupo	CodExp	Aptitud	Generaciones
1	3669868374	2.5657802	1500
2	3669868797	2.198274	1
3	3669869691	6.280862	13
4	3669869701	7.174182	10
5	3669869714	12.655541	7
6	3669869740	22.283041	49
7	3669872742	76.99966	1500
8	3669878139	132.72305	1500
9	3669919026	11.573425	1

La evolución de la mejor aptitud en cada grupo a través de las generaciones para cada grupo de pruebas se muestra en las gráficas 6.46, 6.47, 6.48, 6.49, 6.50, 6.51, 6.52, 6.53 y 6.54 y la tabla 6.35 muestra los rangos de porcentajes obtenidos por todas las iteraciones del algoritmo en cada grupo.

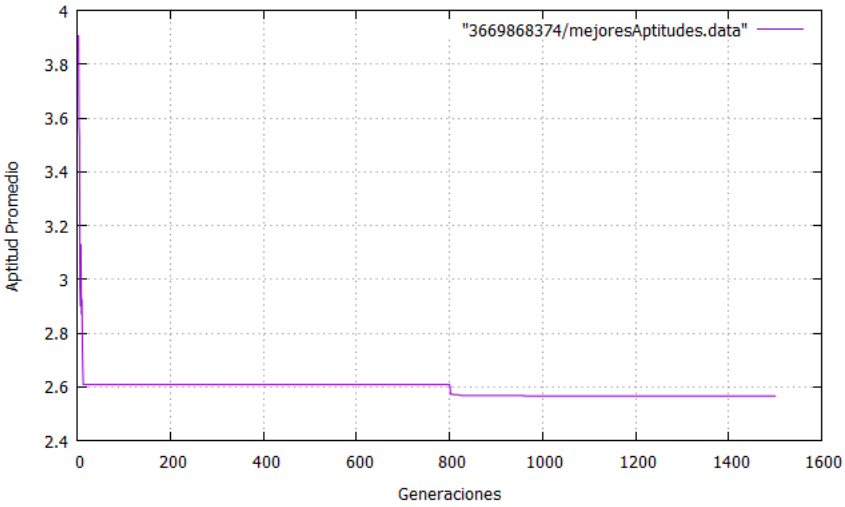

Figura 6.46: Evolución de la mejor aptitud del Grupo 1 con el prototipo 4 del algoritmos OEP

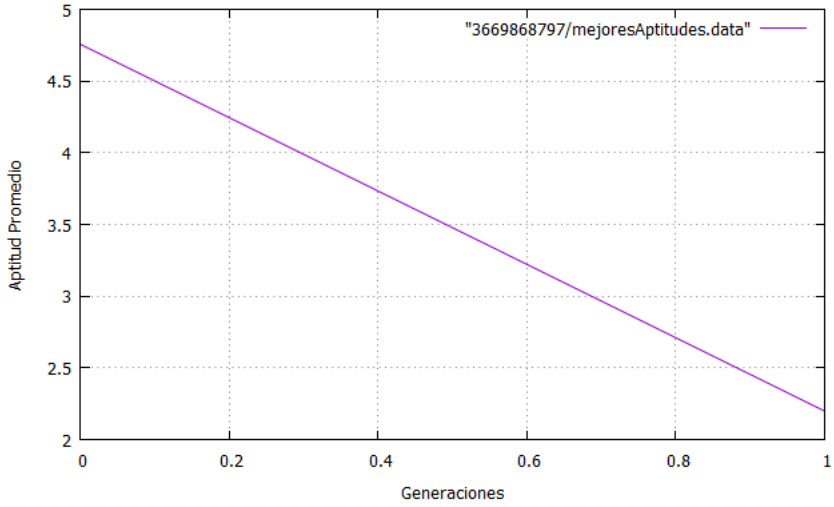

Figura 6.47: Evolución de la mejor aptitud del Grupo 2 con el prototipo 4 del algoritmos OEP

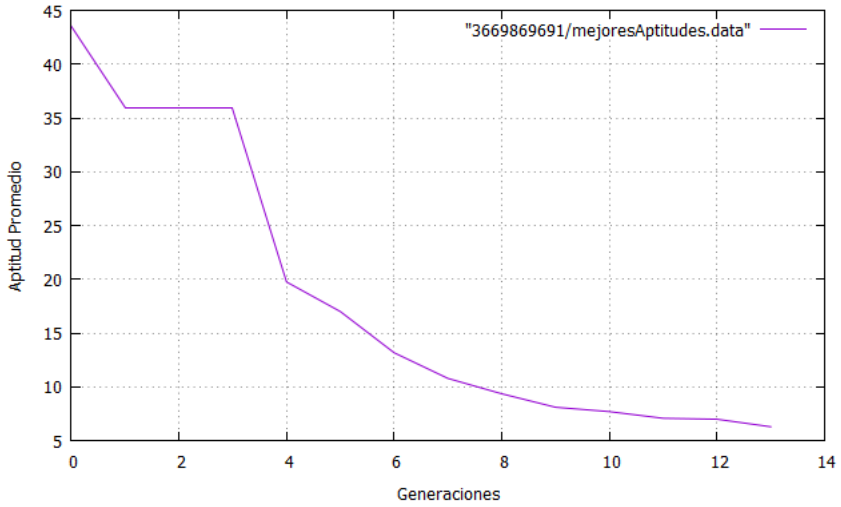

Figura 6.48: Evolución de la mejor aptitud del Grupo 3 con el prototipo 4 del algoritmos OEP

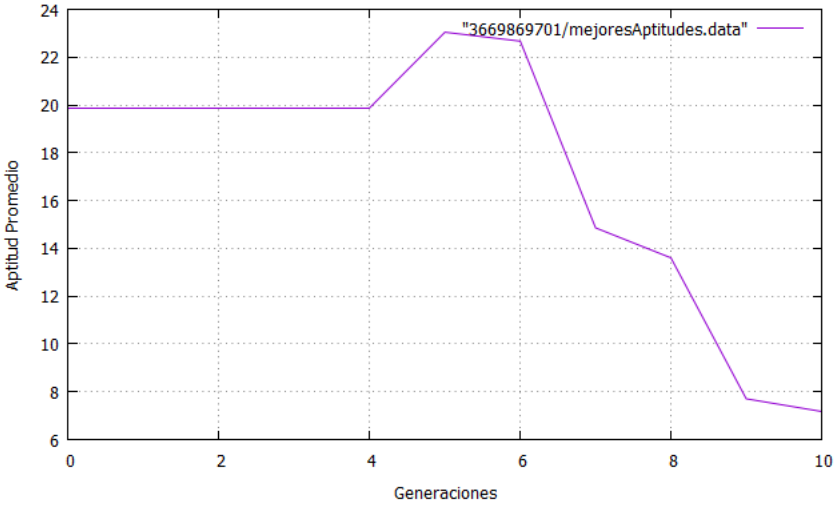

Figura 6.49: Evolución de la mejor aptitud del Grupo 4 con el prototipo 4 del algoritmos OEP

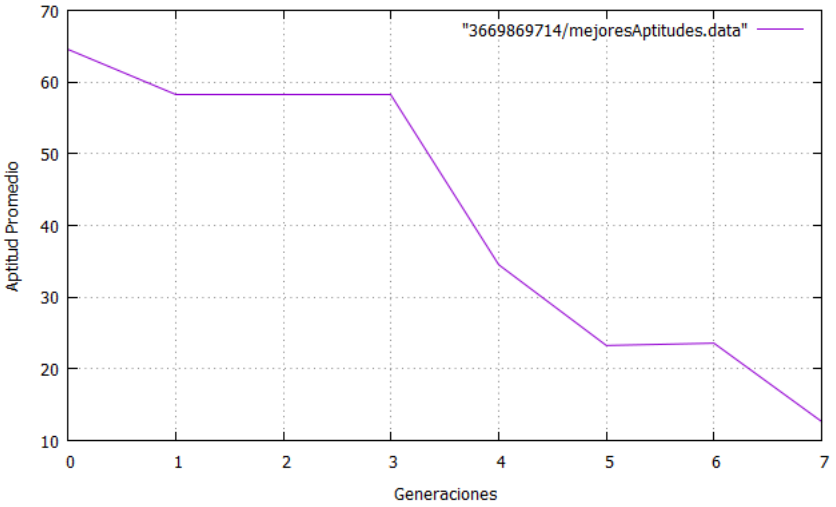

Figura 6.50: Evolución de la mejor aptitud del Grupo 5 con el prototipo 4 del algoritmos OEP

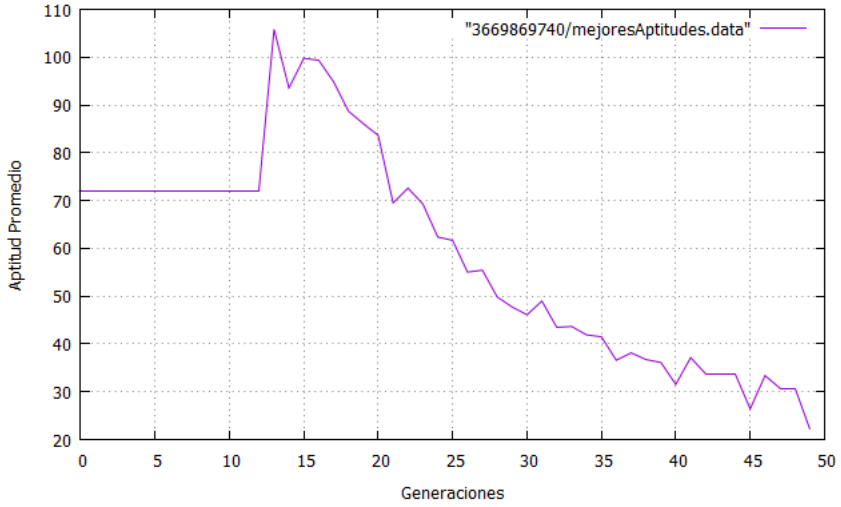

Figura 6.51: Evolución de la mejor aptitud del Grupo 6 con el prototipo 4 del algoritmos OEP

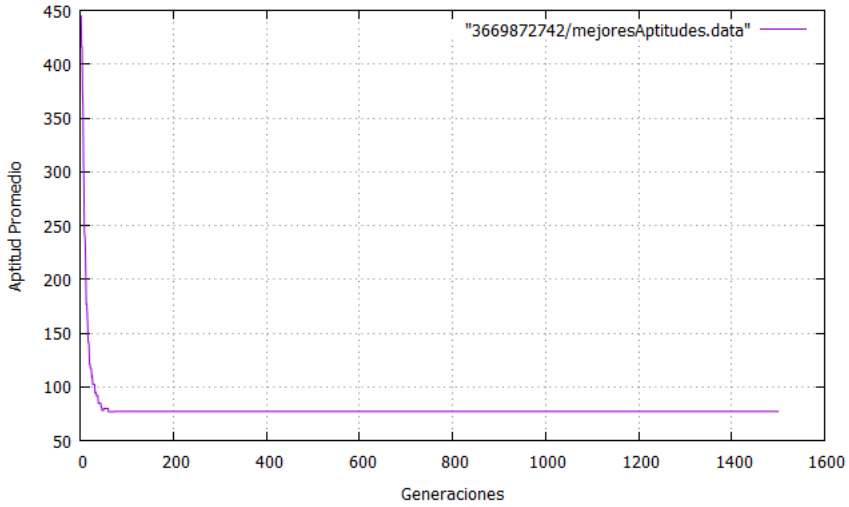

Figura 6.52: Evolución de la mejor aptitud del Grupo 7 con el prototipo 4 del algoritmos OEP

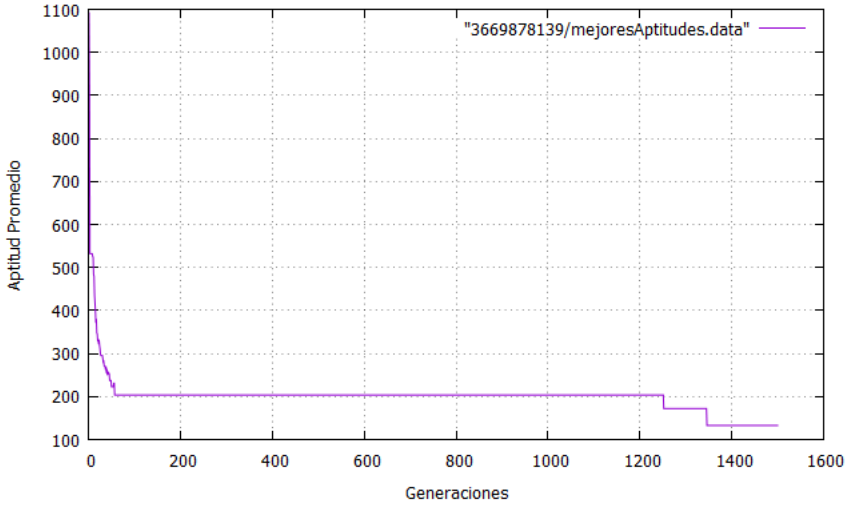

Figura 6.53: Evolución de la mejor aptitud del Grupo 8 con el prototipo 4 del algoritmos OEP

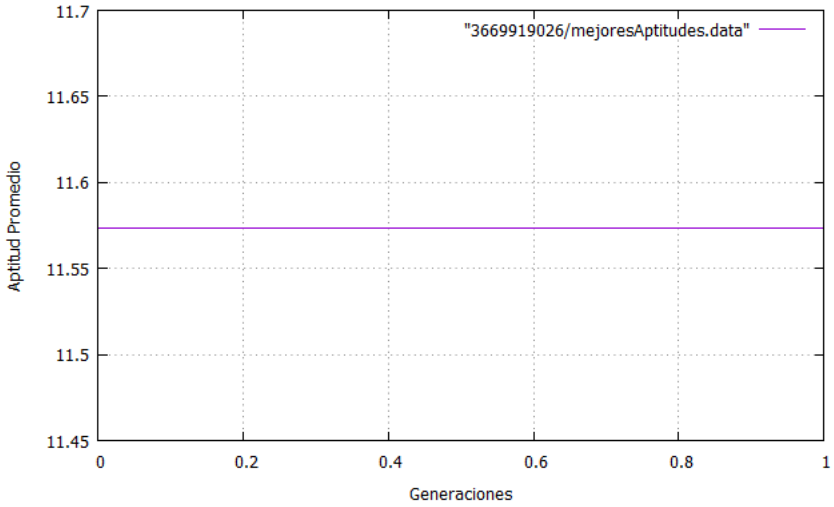

Figura 6.54: Evolución de la mejor aptitud del Grupo 9 con el prototipo 4 del algoritmos OEP

Tabla 6.35: Rangos de probabilidades conseguidos con el prototipo 4 del algoritmo OEP

Grupo	1	2	3	4	5	6	7	8	9
Más cercano	95.0 %	95.0 %	90.0 %	99.9 %	99.5 %	97.5 %	99.9 %	99.9 %	99.9 %
Más Lejano	95.0 %	66.7 %	50.0 %	99.5 %	90.0 %	25.0 %	99.5 %	75.0 %	75.0 %

Se puede ver que en general el algoritmo puede llegar a una buena solución a este problema pero en la el rango de probabilidades que obtuvo en los casos con los grupos más grandes aumenta demasiado, y aunque en su mejor solución consiguió resultados de aceptación arriba del 90.0 % no se puede asegurar que esto se logre con esta configuración del algoritmo siempre.

6.4.2. Experimentación Algoritmo ED

Para el algoritmo ED se cambiaron los parámetros y se le do una población de 300 individuos en 2000 generaciones, las tablas 6.36, 6.37, 6.38 y 6.39 reportan las soluciones en cada versión del algoritmo para este caso.

Tabla 6.36: Mejores aptitudes reportadas por el prototipo 4 del algoritmos ED/-Mejor/1/bin

Aptitud Promedio					
Ejecución	1	2	3	4	5
Grupo 1	2.564327	2.564328	2.564327	2.564327	2.564327
Grupo 2	3.847288	3.00539	2.548878	2.692511	2.670648
Grupo 3	8.196642	8.19664	8.196642	8.196642	8.196641
Grupo 4	6.437071	8.959951	8.659617	6.928549	5.63061
Grupo 5	13.935492	13.466413	13.926859	15.073215	10.075403
Grupo 6	18.030033	29.365885	21.93201	22.030375	17.306337
Grupo 7	36.923782	40.147358	38.383682	41.464767	36.64431
Grupo 8	78.0725	75.32356	70.37365	74.700676	79.04875
Grupo 9	137.94522	158.23834	123.453964	161.22145	129.81653
Número de Generaciones					
Ejecución	1	2	3	4	5
Grupo 1	2000	2000	2000	2000	2000
Grupo 2	1	2	1	1	1
Grupo 3	2000	2000	2000	2000	2000
Grupo 4	5	7	6	7	5
Grupo 5	5	5	4	5	4
Grupo 6	7	4	8	4	5
Grupo 7	195	154	284	286	195
Grupo 8	268	392	741	473	170
Grupo 9	367	335	1109	205	322

Tabla 6.37: Mejores aptitudes reportadas por el prototipo 4 del algoritmos ED/-Mejor/1/exp

Aptitud Promedio					
Ejecución	1	2	3	4	5
Grupo 1	2.724525	2.955209	3.083088	2.578694	2.590907
Grupo 2	3.337917	2.354886	3.015639	2.978731	2.901563
Grupo 3	9.597012	8.389196	8.934113	8.352327	8.314767
Grupo 4	18.639664	37.066956	19.824085	9.620779	18.231604
Grupo 5	20.04482	55.100677	13.815654	20.757318	20.283566
Grupo 6	65.851875	65.545395	69.503914	63.59269	97.9334
Grupo 7	320.24582	283.67297	205.49408	374.37518	285.99725
Grupo 8	723.2518	693.1894	725.78235	745.84735	674.22546
Grupo 9	11.573435	10004.1697	976.67523	979.005	1396.2097
Número de Generaciones					
Ejecución	1	2	3	4	5
Grupo 1	2000	2000	2000	2000	2000
Grupo 2	8	36	564	608	1256
Grupo 3	2000	2000	2000	2000	2000
Grupo 4	2000	2000	2000	2000	2000
Grupo 5	2000	2000	1369	2000	2000
Grupo 6	2000	2000	2000	2000	2000
Grupo 7	2000	2000	2000	2000	2000
Grupo 8	2000	2000	2000	2000	2000
Grupo 9	1	2000	2000	2000	2000

Tabla 6.38: Mejores aptitudes reportadas por el prototipo 4 del algoritmos ED/Aleatorio/1/bin

Aptitud Promedio					
Ejecución	1	2	3	4	5
Grupo 1	2.564327	2.564327	2.564327	2.564327	2.564327
Grupo 2	2.056782	2.456411	1.995896	2.110947	2.427086
Grupo 3	8.196641	8.196638	8.196638	8.196638	8.196638
Grupo 4	5.084077	6.467075	6.168217	4.556554	5.332811
Grupo 5	7.757051	14.768695	12.44209	9.015654	11.922293
Grupo 6	24.223495	21.722681	14.203817	20.537312	22.034555
Grupo 7	64.23611	59.183502	80.007996	84.81769	50.890926
Grupo 8	92.56669	136.65222	133.96556	5.575651	125.00226
Grupo 9	227.88847	264.0195	11.573425	300.54745	227.88847
Número de Generaciones					
Ejecución	1	2	3	4	5
Grupo 1	2000	2000	2000	2000	2000
Grupo 2	60	218	184	155	85
Grupo 3	2000	2000	2000	2000	2000
Grupo 4	168	149	104	101	184
Grupo 5	237	102	133	132	172
Grupo 6	163	169	218	109	185
Grupo 7	2000	2000	2000	2000	2000
Grupo 8	2000	2000	2000	1	2000
Grupo 9	2000	2000	1	2000	2000

Tabla 6.39: Mejores aptitudes reportadas por el prototipo 4 del algoritmos ED/-Mejor/1/exp

Aptitud Promedio					
Ejecución	1	2	3	4	5
Grupo 1	3.2954	2.892445	2.738899	2.71836	3.209321
Grupo 2	2.197542	2.43821	2.198354	2.214487	2.464094
Grupo 3	12.461293	10.946877	12.249832	14.331052	9.019796
Grupo 4	17.460669	20.588358	11.778821	15.904266	15.909055
Grupo 5	28.84492	35.79315	37.11096	50.738483	40.274387
Grupo 6	79.10051	87.07296	58.28166	40.934246	65.47824
Grupo 7	367.93845	511.87918	268.87918	287.281	271.6722
Grupo 8	675.0369	5.575651	768.1459	660.15436	634.5908
Grupo 9	965.6432	1201.2499	1374.4576	1450.9242	1473.3595
Número de Generaciones					
Ejecución	1	2	3	4	5
Grupo 1	2000	2000	2000	2000	2000
Grupo 2	203	2000	380	652	682
Grupo 3	2000	2000	2000	2000	2000
Grupo 4	2000	2000	2000	2000	2000
Grupo 5	2000	2000	2000	2000	2000
Grupo 6	2000	2000	2000	2000	2000
Grupo 7	2000	2000	2000	2000	2000
Grupo 8	2000	1	2000	2000	2000
Grupo 9	2000	2000	2000	2000	2000

En las gráficas 6.55 a 6.63 se observa la evolución de la aptitud de los grupos de prueba con el algoritmo ED en las mejores soluciones encontradas

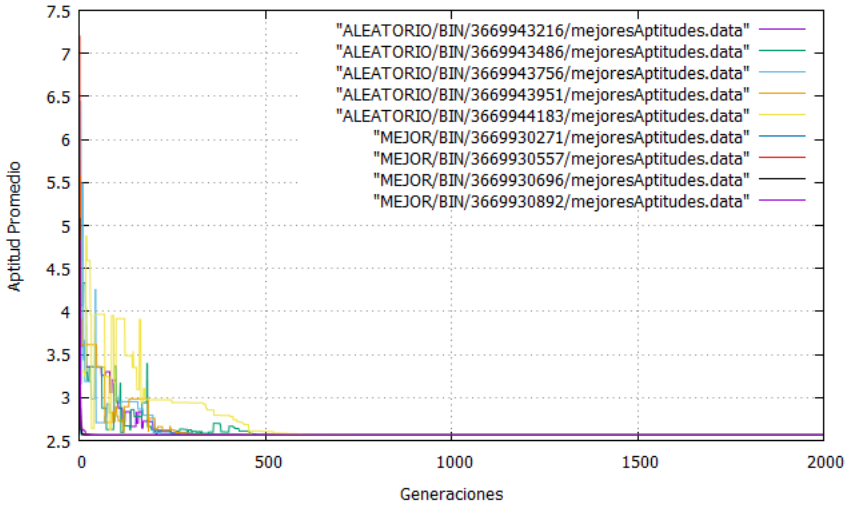

Figura 6.55: Evolución de la mejor aptitud del Grupo 1 con el prototipo 4 del algoritmos ED

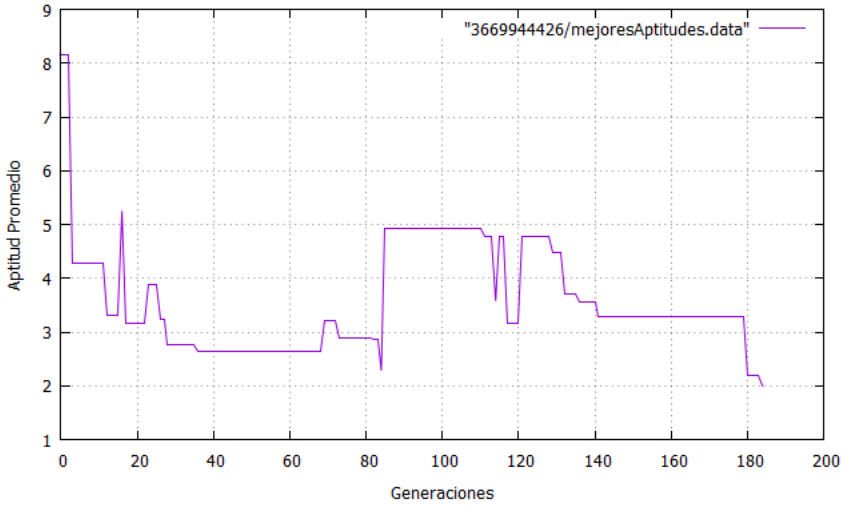

Figura 6.56: Evolución de la mejor aptitud del Grupo 2 con el prototipo 4 del algoritmos ED

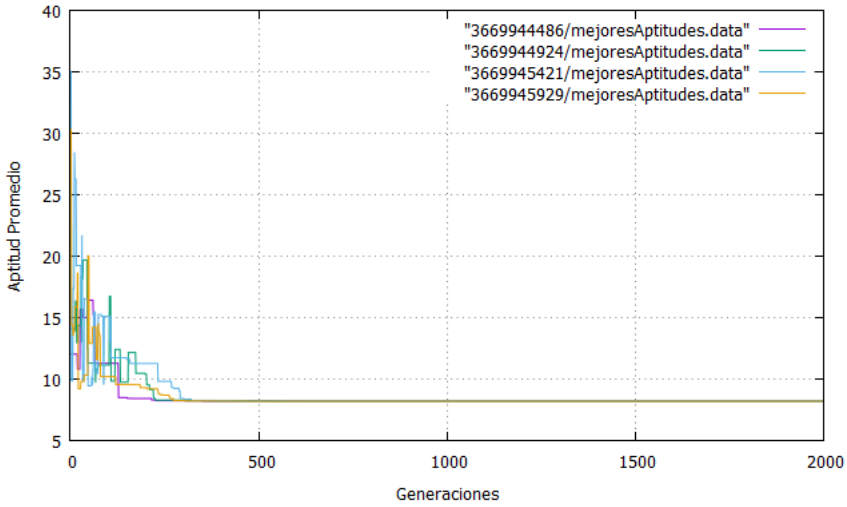

Figura 6.57: Evolución de la mejor aptitud del Grupo 3 con el prototipo 4 del algoritmos ED

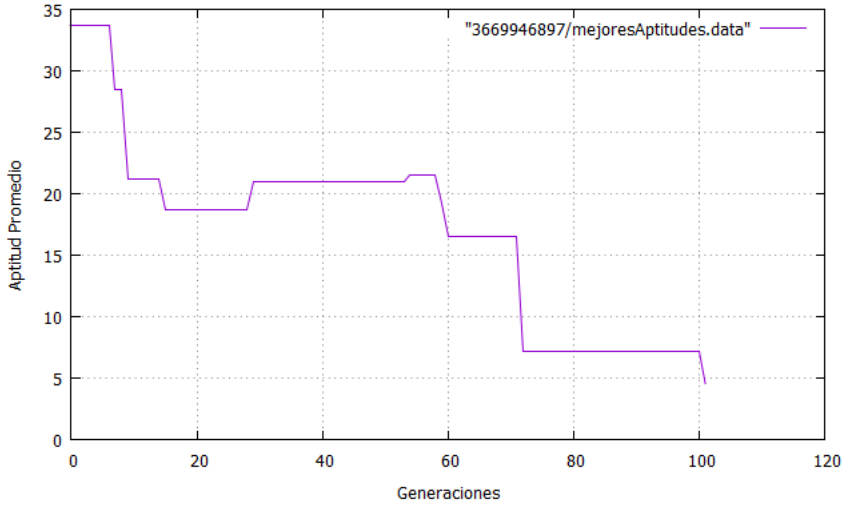

Figura 6.58: Evolución de la mejor aptitud del Grupo 4 con el prototipo 4 del algoritmos ED

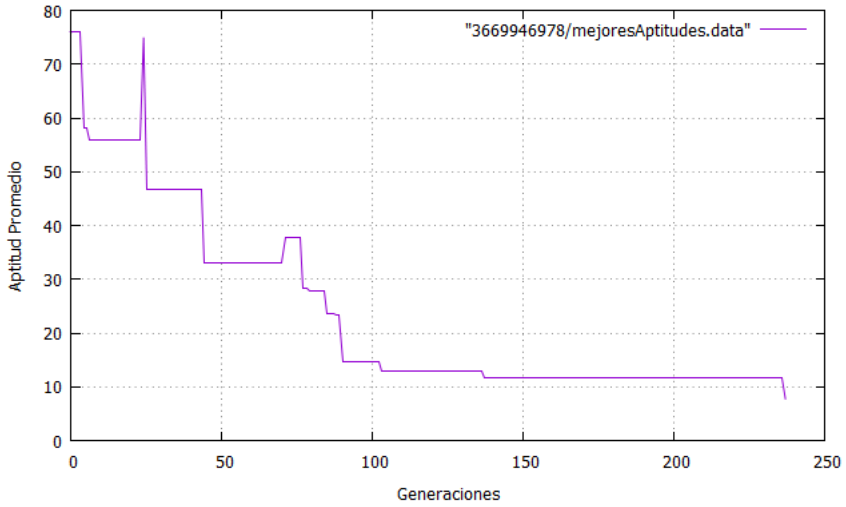

Figura 6.59: Evolución de la mejor aptitud del Grupo 5 con el prototipo 4 del algoritmos ED

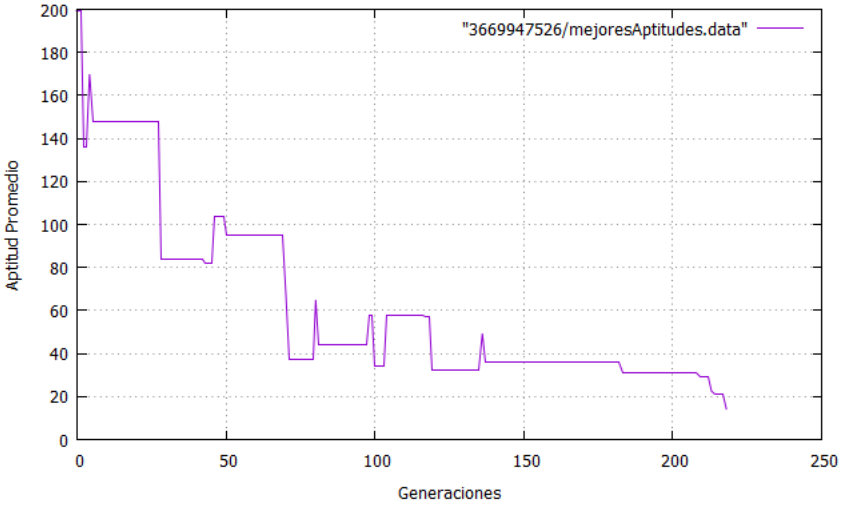

Figura 6.60: Evolución de la mejor aptitud del Grupo 6 con el prototipo 4 del algoritmos ED

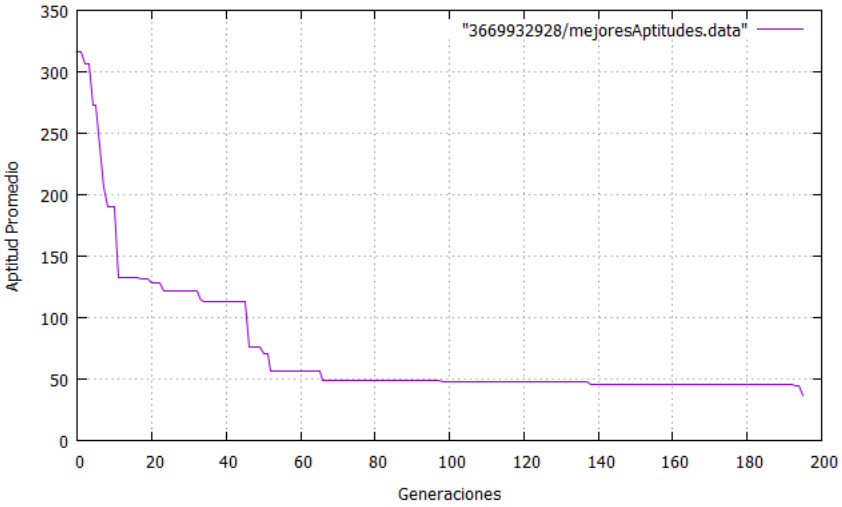

Figura 6.61: Evolución de la mejor aptitud del Grupo 7 con el prototipo 4 del algoritmos ED

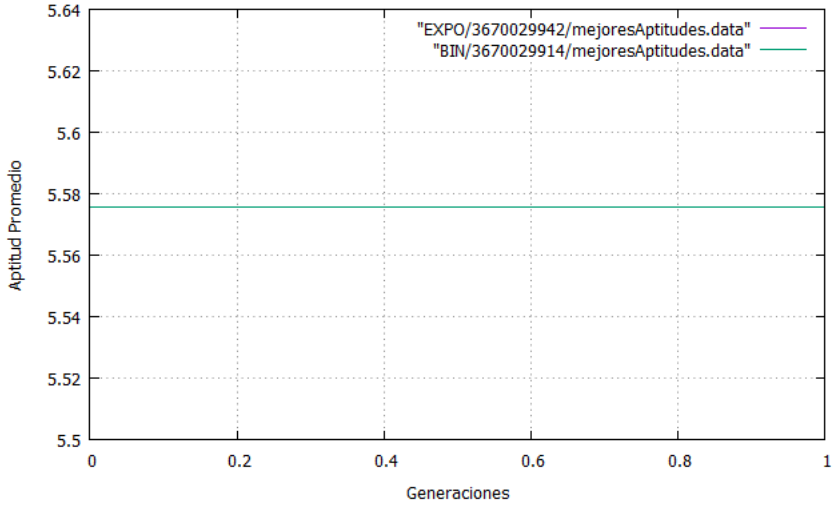

Figura 6.62: Evolución de la mejor aptitud del Grupo 8 con el prototipo 4 del algoritmos ED

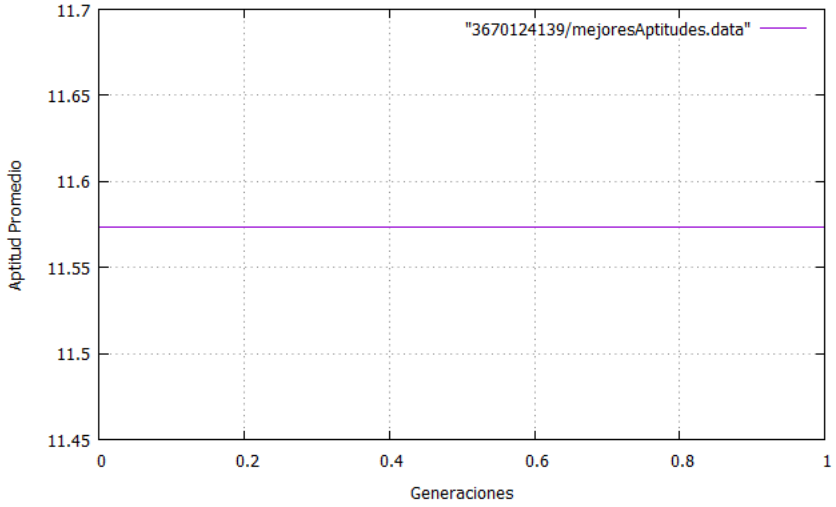

Figura 6.63: Evolución de la mejor aptitud del Grupo 9 con el prototipo 4 del algoritmos ED

Se puede observar como el la versión del algoritmo con el mejor individuo para generar el nuevo y con el operador de cruza binario obtuvo los mejores resultados, ya que estos al haber terminado en casi todos sus casos sin utilizar

las 3000 generaciones encontraron soluciones que satisfacen el criterio de paro de un mínimo de aceptación del 90.0 %. En la tabla 6.40 se observa las mejores soluciones encontradas y en la tabla 6.41 los rangos del porcentaje de aceptación encontrados en cada versión.

Tabla 6.40: Mejores iteraciones del prototipo 4 del algoritmo ED

Grupo	Versión	CodExp	Aptitud Promedio	Generaciones
1	ED/*/1/bin	Varios	2.564327	2000
2	ED/Aleatorio/1/bin	3669944426	1.995896	184
3	ED/Aleatorio/1/bin	Varios	8.196638	2000
4	ED/Aleatorio/1/bin	3669946897	4.556554	101
5	ED/Aleatorio/1/bin	3669946978	7.757051	237
6	ED/Aleatorio/1/bin	3669947526	14.203817	218
7	ED/Mejor/1/bin	3669932928	36.64431	195
8	ED/Aleatorio/1/*	Varios	5.575651	1
9	ED/Aleatorio/1/bin	3670124139	11.573425	1

Tabla 6.41: Rangos de probabilidades conseguidos con el prototipo 4 del algoritmo ED

Versión	ED/Mejor/1/bin								
Grupo	1	2	3	4	5	6	7	8	9
Limite Superior	95.0 %	90.0 %	66.7 %	99.9 %	99.9 %	99.0 %	99.9 %	99.9 %	99.9 %
Limite Inferior	95.0 %	90.0 %	66.7 %	99.9 %	99.5 %	95.0 %	99.9 %	99.9 %	99.9 %
Versión	ED/Mejor/1/exp								
Grupo	1	2	3	4	5	6	7	8	9
Limite Superior	95.0 %	95.0 %	66.7 %	99.9 %	99.5 %	50.0 %	99.5 %	25.0 %	99.9 %
Limite Inferior	90.0 %	90.0 %	50.0 %	90.0 %	66.7 %	25.0 %	90.0 %	12.5 %	0.0 %
Versión	ED/Aleatorio/1/bin								
Grupo	1	2	3	4	5	6	7	8	9
Limite Superior	95.0 %	95.0 %	66.7 %	99.9 %	99.9 %	99.5 %	99.9 %	99.9 %	99.9 %
Limite Inferior	95.0 %	95.0 %	66.7 %	99.9 %	99.5 %	97.5 %	99.9 %	99.9 %	95.0 %
Versión	ED/Aleatorio/1/exp								
Grupo	1	2	3	4	5	6	7	8	9
Limite Superior	95.0 %	95.0 %	66.7 %	99.5 %	95.0 %	80.0 %	97.5 %	99.9 %	1.0 %
Limite Inferior	90.0 %	95.0 %	40.0 %	97.5 %	75.0 %	33.3 %	50.0 %	12.5 %	0.0 %

Las versiones del algoritmo ED con cruce binario demostraron ser las más eficientes, encontrando las mejores soluciones en cada caso, manteniendo rangos de probabilidades pequeños y en su mayoría logrando converger en pocas generaciones. Lo que deja claro el buen rendimiento de este algoritmo para la tarea de optimización multiobjetivo para multiples pruebas χ^2 de PEarson.

6.4.3. Experimentación Algoritmo EE

Para el algoritmo EE se realizaron 5 iteraciones para cada grupo de distribuciones, se configuro las dos versiones del algoritmo con una tamaño de población de 30 en 2000 generaciones. Los resultados obtenidos se reportan en las tablas 6.42 y 6.43.

Tabla 6.42: Mejores aptitudes reportadas por el prototipo 4 del algoritmos ($\mu + \lambda$)-EE

Aptitud Promedio					
Ejecución	1	2	3	4	5
Grupo 1	3.596219	6.679757	4.534359	4.162400	2.888600
Grupo 2	10.702016	7.594477	6.027207	7.119918	9.445312
Grupo 3	11.340267	11.888689	20.358592	20.402092	20.426840
Grupo 4	48.903716	46.016003	48.931377	43.546118	41.862866
Grupo 5	88.459130	83.315271	94.797060	67.840968	74.940373
Grupo 6	130.548972	10.801105	149.284785	147.1846	144.164455
Grupo 7	248.427577	273.546547	251.105491	273.1849	288.242946
Grupo 8	598.387021	539.1561	550.116147	563.154107	538.083429
Grupo 9	1107.5184	1157.554300	1099.998866	1148.186115	1115.161571
Número de Generaciones					
Ejecución	1	2	3	4	5
Grupo 1	2000	2000	2000	2000	2000
Grupo 2	2000	2000	2000	2000	2000
Grupo 3	2000	2000	2000	2000	2000
Grupo 4	2000	2000	2000	2000	2000
Grupo 5	2000	2000	2000	2000	2000
Grupo 6	2000	1	2000	2000	2000
Grupo 7	2000	2000	2000	2000	2000
Grupo 8	2000	2000	2000	2000	2000
Grupo 9	2000	2000	2000	2000	2000

Tabla 6.43: Mejores aptitudes reportadas por el prototipo 4 del algoritmos (μ, λ)-EE

Aptitud Promedio					
Ejecución	1	2	3	4	5
Grupo 1	11.076476	11.077123	11.077110	11.076229	11.075980
Grupo 2	20.995291	20.992753	20.992146	20.992147	20.997626
Grupo 3	49.143836	49.137825	49.1408863	49.134201	49.182480
Grupo 4	48.862843	48.880165	48.865541	48.871346	48.887447
Grupo 5	99.923458	99.899535	99.900621	99.898479	99.8752327
Grupo 6	211.678491	211.708495	211.6971	211.718912	211.672447
Grupo 7	287.984246	287.9843	288.083473	287.9843	288.084528
Grupo 8	597.950101	597.940906	598.387002	598.251847	597.9918
Grupo 9	1180.935795	1180.548617	1180.483157	1180.706409	1180.365705
Número de Generaciones					
Ejecución	1	2	3	4	5
Grupo 1	2000	2000	2000	2000	2000
Grupo 2	2000	2000	2000	2000	2000
Grupo 3	2000	2000	2000	2000	2000
Grupo 4	2000	2000	2000	2000	2000
Grupo 5	2000	2000	2000	2000	2000
Grupo 6	2000	2000	2000	2000	2000
Grupo 7	2000	2000	2000	2000	2000
Grupo 8	2000	2000	2000	2000	2000
Grupo 9	2000	2000	2000	2000	2000

En la tabla 6.44 muestra los mejores resultados obtenidos por este algoritmo, puede apreciarse que la versión del algoritmo con sobrevivientes entre generaciones es la que mejores resultados a dado hasta el momento.

Tabla 6.44: Mejores iteraciones del prototipo 4 del algoritmo EE

Grupo	Versión	CodExp	Aptitud Promedio	Generaciones
1	$(\mu + \lambda) - EE$	3670385705	2.888600	2000
2	$(\mu + \lambda) - EE$	3670391486	6.027207	2000
3	$(\mu + \lambda) - EE$	3670398355	11.340267	2000
4	$(\mu + \lambda) - EE$	3670434702	41.862866	2000
5	$(\mu + \lambda) - EE$	3670463908	67.840968	2000
6	$(\mu + \lambda) - EE$	3673467135	10.801105	1
7	$(\mu + \lambda) - EE$	3673471553	248.427577	2000
8	$(\mu + \lambda) - EE$	3673488920	538.083429	2000
9	$(\mu + \lambda) - EE$	3673560083	1099.998866	2000

Las mejore soluciones se consiguieron con la versión con sobrevivientes del algoritmo, las gráficas a continuación muestran la evolución de la mejor aptitud de las mejores soluciones encontradas.

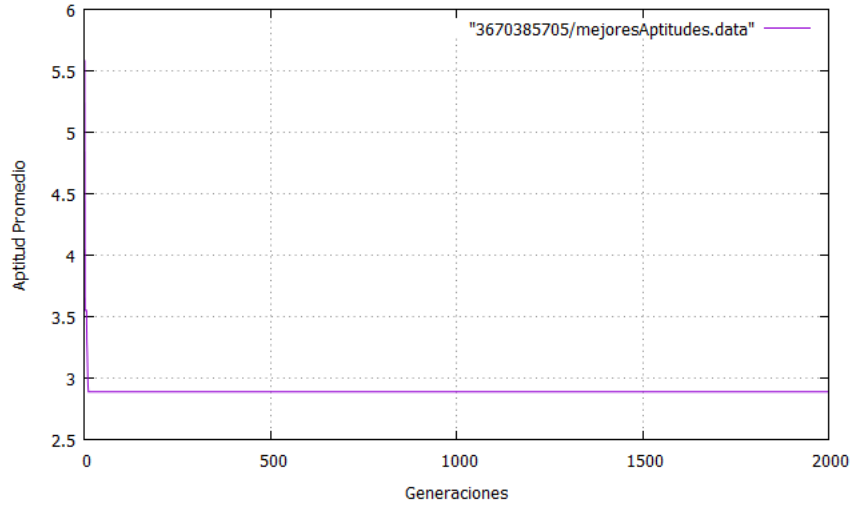

Figura 6.64: Evolución de la mejor aptitud del Grupo 1 con el prototipo 4 del algoritmos EE

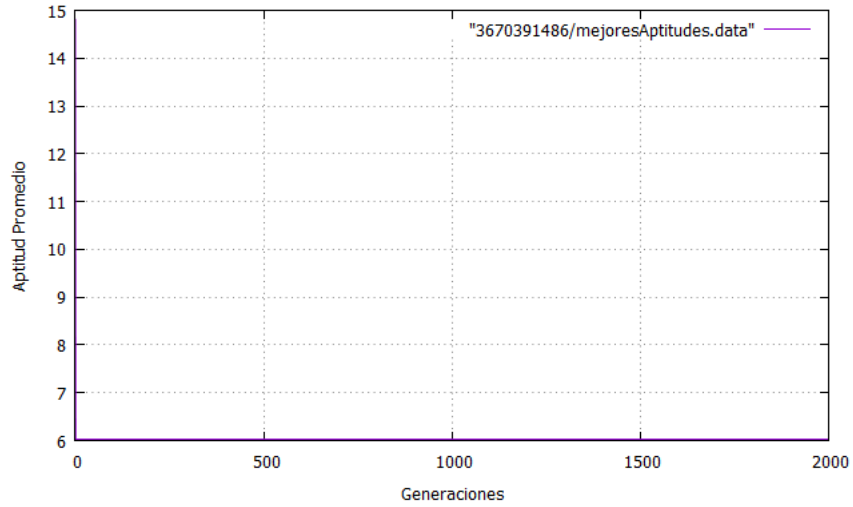

Figura 6.65: Evolución de la mejor aptitud del Grupo 2 con el prototipo 4 del algoritmos EE

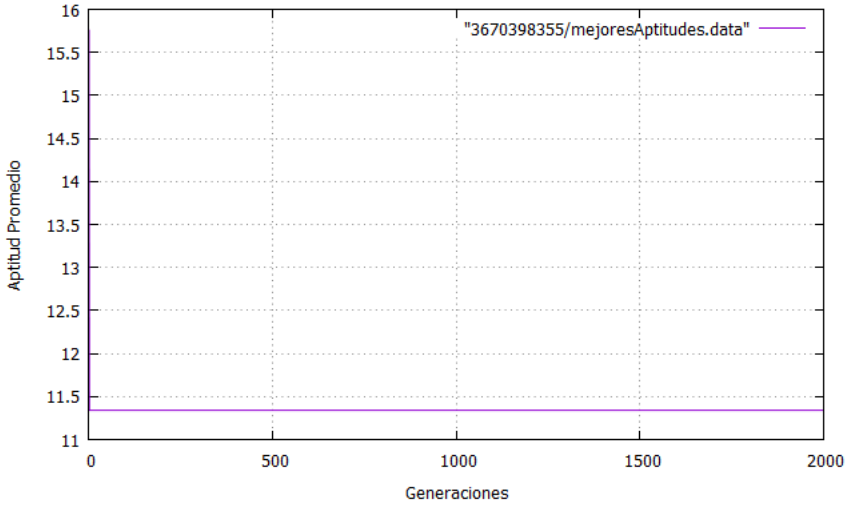

Figura 6.66: Evolución de la mejor aptitud del Grupo 3 con el prototipo 4 del algoritmos EE

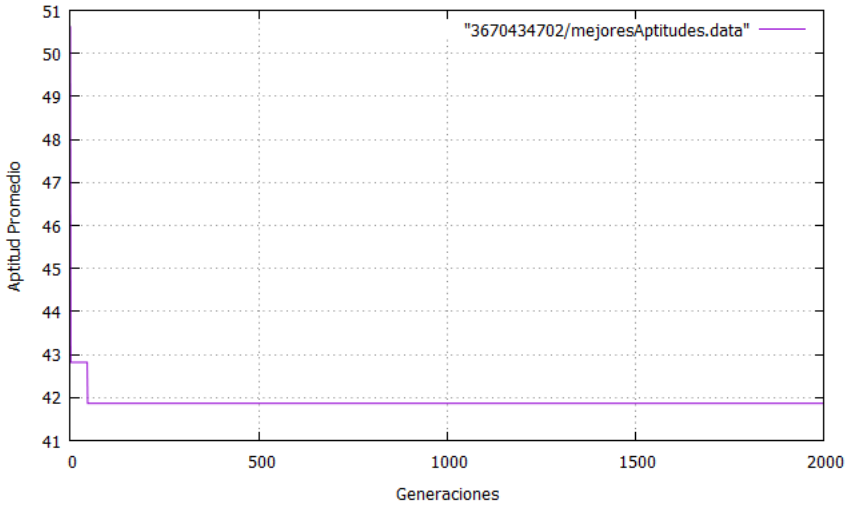

Figura 6.67: Evolución de la mejor aptitud del Grupo 4 con el prototipo 4 del algoritmos EE

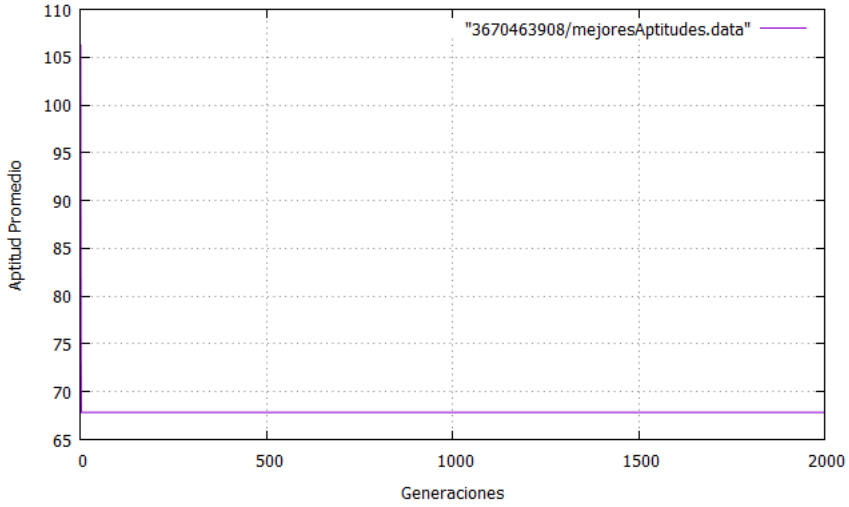

Figura 6.68: Evolución de la mejor aptitud del Grupo 5 con el prototipo 4 del algoritmos EE

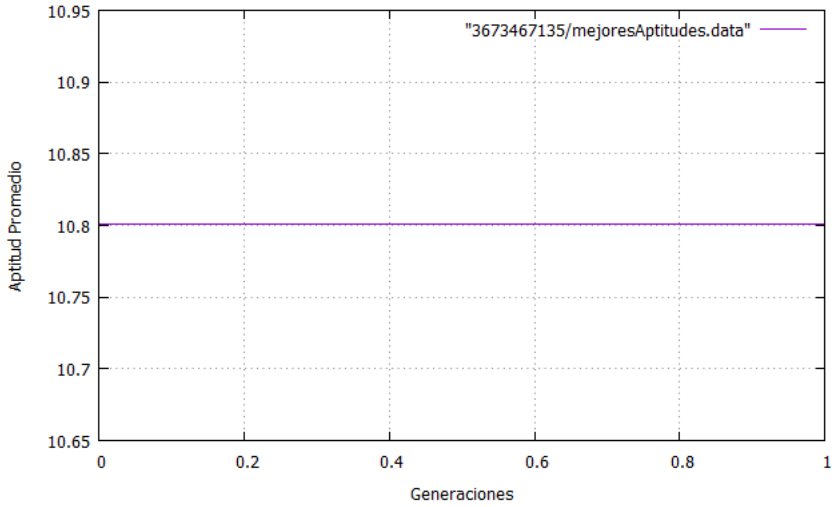

Figura 6.69: Evolución de la mejor aptitud del Grupo 6 con el prototipo 4 del algoritmos EE

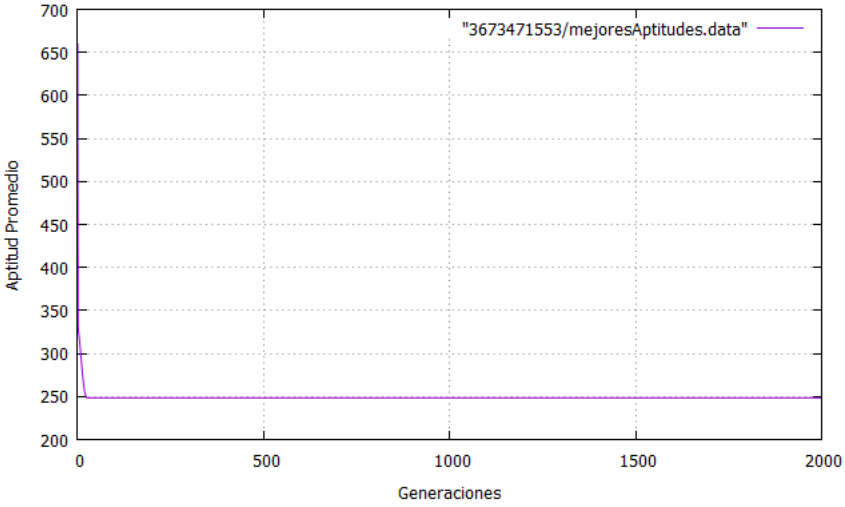

Figura 6.70: Evolución de la mejor aptitud del Grupo 7 con el prototipo 4 del algoritmos EE

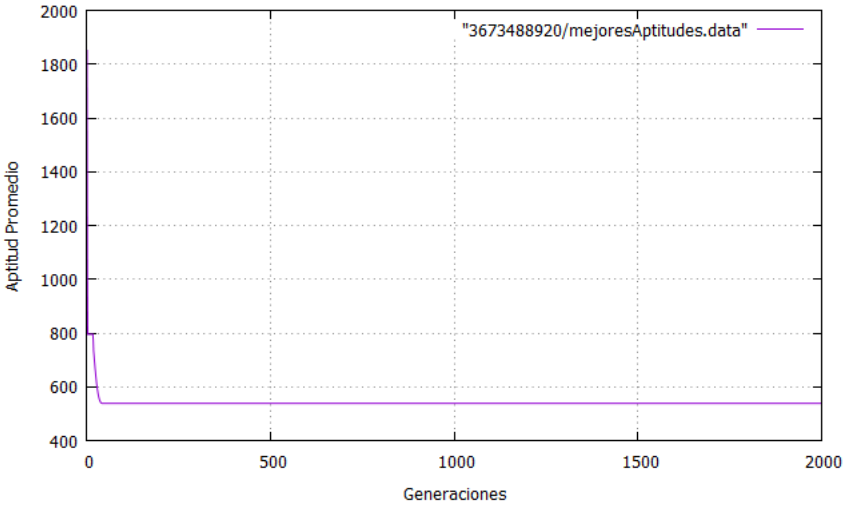

Figura 6.71: Evolución de la mejor aptitud del Grupo 8 con el prototipo 4 del algoritmos EE

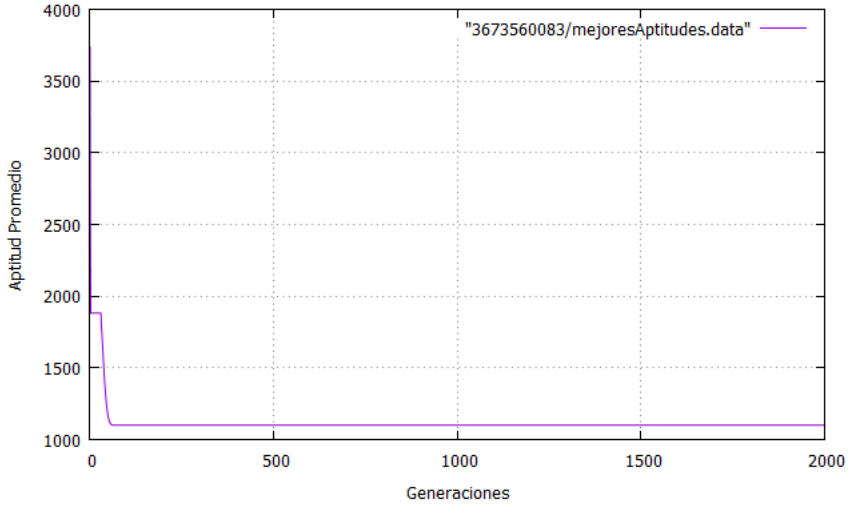

Figura 6.72: Evolución de la mejor aptitud del Grupo 9 con el prototipo 4 del algoritmos EE

Los rangos de probabilidades reportados en la tabla 6.45 muestran que la versión sin sobrevivientes fue la que se mantuvo con el rango menor de variación, pero dio resultados malos, mientras que los rangos de la versión con sobrevivientes fue un poco grande, pero en general los resultados obtenidos no fueron satisfactorios.

Tabla 6.45: Rangos de probabilidades conseguidos con el prototipo 4 del algoritmo EE

Versión	$(\mu + \lambda) - EE$								
Grupo	1	2	3	4	5	6	7	8	9
Más Cercano	90.0 %	80.0 %	50.0 %	80.0 %	50.0 %	99.9 %	99.0 %	50.0 %	0.1 %
Menos Cercano	75.0 %	50.0 %	25.0 %	75.0 %	25.0 %	80.0 %	97.5 %	40.0 %	0.1 %
Versión	$(\mu, \lambda) - EE$								
Grupo	1	2	3	4	5	6	7	8	9
Más cercano	50.0 %	25.0 %	1.0 %	75.0 %	25.0 %	40.0 %	97.5 %	40.0 %	0.1 %
Menos cercano	50.0 %	25.0 %	1.0 %	75.0 %	25.0 %	40.0 %	97.5 %	40.0 %	0.1 %

En este experimento el algoritmo EE no dio resultados muy buenos más que para los casos de los grupos con menos distribuciones. Por lo que se deben hacer consideraciones muy serias para ajustar los parámetros de este algoritmo.

6.4.4. Conclusiones del experimento 4

Los algoritmos OEP y ED mostraron un buen rendimiento y lograron en la mayoría de los casos encontrar soluciones que satisficieron el criterio de paro de la probabilidad esperada, evitando así el uso de todas las generaciones de búsqueda, pero el algoritmo EE no mostró un buen desempeño, si bien en algunos casos logro conseguir respuestas aceptables, aun le falta mejorar mucho el rendimiento a este algoritmo por lo que se tienen que hacer cambios a sus parámetros.

Capítulo 7

Experimentación con datos reales

En este capitulo se describirán la ejecución de los algoritmos evolutivos sobre los datos reales de robos seleccionados, el reporte de su desempeño y las conclusiones de la comparación del desempeño de estos.

7.1. Datos de Robos

Aquí se reportan los datos de robos utilizados para la tarea de predicción, los resultados de ejecución de los diferentes algoritmos sobre estos y la comparación entre estos.

Para las pruebas para este experimento se tomaron los datos de 3 estados de la república sobre robos comunes, los estado son de Baja California, Colima y la Ciudad de México, esta información es el total de robos comunes ocurridos en cada uno de sus municipios por mes durante el 2014.

Las tablas 7.1, 7.2 y 7.3 muestran la información de la cantidad de robos ocurridos en cada uno de estos estados por municipio en el año 2014 obtenidos de la página del Secretariado Ejecutivo[47].

Tabla 7.1: Número de robos en el estado de Baja California en el 2014

Municipio	Ene.	Feb.	Mar.	Abr.	May.	Jun.	Jul.	Ago.	Sep.	Oct.	Nov.	Dic.
Ensenada	590	541	607	599	593	544	556	590	529	560	507	522
Mexicali	1810	1695	1868	1833	1764	1659	1881	1835	1849	1887	1658	1533
P. de Rosarito	143	138	150	165	154	157	131	136	120	156	136	116
Tecate	193	142	214	154	155	166	171	168	174	169	176	159
Tijuana	2027	1816	1970	1919	1852	1807	2011	1867	1950	1934	1616	1624

Tabla 7.2: Número de robos en el estado de Colima en el 2014

Municipio	Ene.	Feb.	Mar.	Abr.	May.	Jun.	Jul.	Ago.	Sep.	Oct.	Nov.	Dic.
Armería	4	7	5	2	5	4	4	2	6	4	4	11
Colima	127	127	110	83	73	99	123	122	90	97	105	104
Comala	4	4	8	4	3	12	1	4	4	2	2	3
Coquimatlán	4	3	1	0	1	1	1	1	2	0	4	0
Cuauhtémoc	4	8	6	2	2	2	2	2	6	1	3	4
Ixtlahuacán	0	1	0	0	0	1	0	0	2	0	2	0
Manzanillo	94	91	87	93	81	78	77	85	48	95	55	82
Minatitlán	0	0	1	0	0	0	0	0	0	1	0	0
Tecomán	56	53	47	47	48	40	43	51	49	42	64	45
V. de Álvarez	60	50	53	59	37	63	70	63	53	58	33	43

Tabla 7.3: Número de robos en la Ciudad de México en el 2014

Municipio	Ene.	Feb.	Mar.	Abr.	May.	Jun.	Jul.	Ago.	Sep.	Oct.	Nov.	Dic.
Álvaro O.	466	433	427	443	455	460	450	434	367	443	464	398
Azcapotzalco	368	383	398	353	381	356	315	362	326	351	372	317
Benito J.	521	519	530	567	645	550	565	547	544	556	558	574
Coyoacán	516	468	488	498	523	464	475	550	503	559	519	507
Cuajimalpa	84	66	100	88	91	83	86	84	86	81	93	84
Cuauthémoc	857	860	971	993	1013	1025	1042	1045	1008	1137	962	1094
Gustavo A. M.	879	879	1009	999	1018	941	1023	884	879	953	780	760
Iztacalco	295	296	325	316	296	291	310	275	286	302	300	273
Iztapalapa	1150	1043	1109	1152	1266	1164	1221	1213	1212	1277	1162	1078
Magdalena C.	78	78	89	73	69	75	61	67	83	66	60	64
Miguel H.	432	446	528	513	557	521	565	539	442	468	418	419
Milpa Alta	22	24	28	18	30	20	28	30	20	27	27	31
Tláhuac	171	144	129	163	151	126	161	106	135	119	131	100
Tlalpan	413	402	440	466	468	438	485	416	470	444	457	388
V. Carranza	391	382	375	446	429	392	378	381	358	367	349	383
Xochimilco	260	209	276	283	280	273	291	259	263	238	228	218

Para cada experimento se utilizaron la información de los meses de Enero a Abril para la predicción y después se probaron los resultados obtenidos con los datos de los meses de Mayo a Diciembre para comprobar si mantuvieron la misma certeza esperada. Esta información es proporcionada a los algoritmos de la misma manera que se hizo para los experimentos con los experimentos 2, 3 y 4 en un formato de texto plano como se mostró en la figura 6.6.

La tabla 7.4 muestra los valores del parámetro de criterio de paro para una

certeza del 90.0 %.

Los resultados obtenidos con los datos de estos meses después serán probados sobre los datos de los meses de Mayo a Diciembre para ver si cumplen con el mismo porcentaje de certeza.

Se ejecutaran 10 veces cada algoritmo en el caso de OEP y ED y 5 en el algoritmo EE para verificar el funcionamiento del algoritmo con cada grupo de datos de robo.

Tabla 7.4: Valores del parámetro de paro para los algoritmos con datos de robo

Estado	Baja California	Colima	Ciudad de México
Valor	1.064	4.168	8.547
Grados de Libertad	4	9	15

7.2. Resultados del Algoritmo OEP con datos reales de robos

El algoritmo OEP se aplico 10 veces sobre cada conjunto, los parámetros utilizados fueron una población de 300 partículas en 3000 generaciones. En la tabla 7.5 se muestran los resultados de estas ejecuciones y las gráficas 7.1, 7.2 y 7.3 muestran la evolución de la aptitud de las mejores soluciones.

Tabla 7.5: Mejores aptitudes reportadas con datos de robos del algoritmos OEP

Baja California		Colima		Ciudad de México	
Aptitud	Generaciones	Aptitud	Generaciones	Aptitud	Generaciones
4.59352	3000	9.935835	199	34.846226	3000
6.069123	3000	10.496959	51	56.638737	3000
4.931637	3000	8.339227	1009	49.84957	3000
5.59982	3000	9.431943	2218	64.365814	3000
4.042833	3000	13.538473	144	40.588337	3000
5.206283	3000	10.981133	2647	150.08952	3000
3.631779	3000	11.59186	55	115.28459	3000
7.0159	3000	13.139587	196	56.618874	3000
4.394632	3000	13.191641	146	48.042015	3000
6.32919	3000	10.141668	38	87.37459	3000

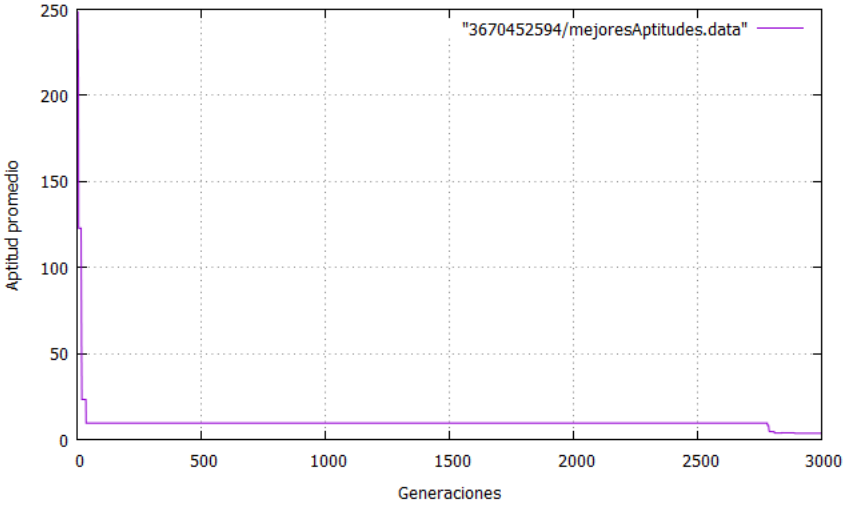

Figura 7.1: Evolución de la aptitud para el caso del estado de Baja California con el algoritmo OEP

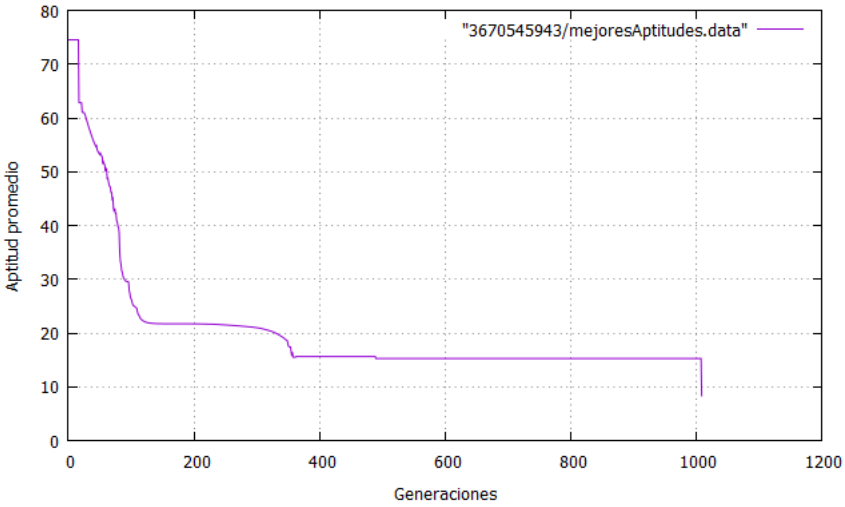

Figura 7.2: Evolución de la aptitud para el caso del estado de Colima con el algoritmo OEP

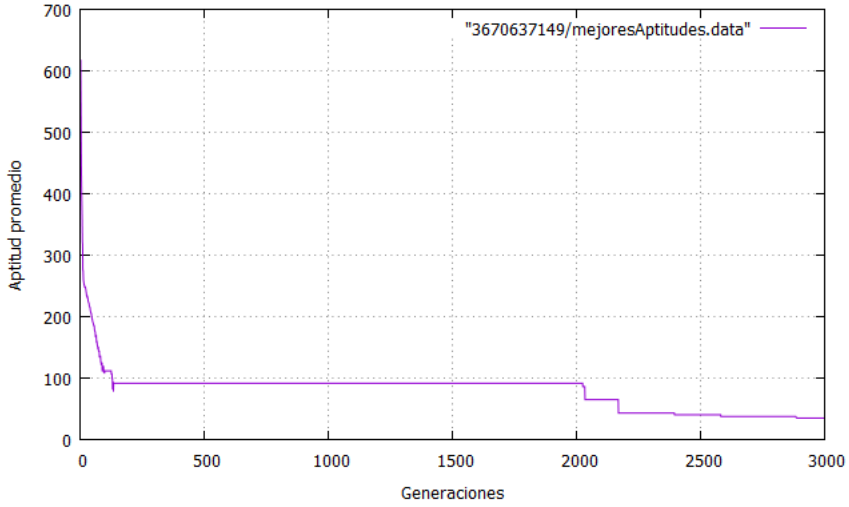

Figura 7.3: Evolución de la aptitud para el caso de la Ciudad de México con el algoritmo OEP

Se puede puede observar que para los casos con menos y más posibles resultados del lugar donde puede ocurrir un robo el algoritmo hizo uso de todas las generaciones para el proceso de búsqueda, mientras que en el intermedio que corresponde al estado de Colima no fue necesario realizar todas las iteraciones del algoritmo para encontrar una buena solución, aunque el rango entre las generaciones utilizadas en este caso es muy grande, siendo el menor el uso de 38 generaciones el mayor el de 2218 generaciones.

La tabla 7.6 muestra el rango de probabilidades que se consiguió con las soluciones encontradas en las 10 iteraciones del algoritmo.

Tabla 7.6: Rangos de probabilidades conseguidos con el algoritmo OEP

Estado	Baja California		Colima		Ciudad de México	
	Porcentaje	Experimento	Porcentaje	Experimento	Porcentaje	Experimento
Más Cercano	90.0 %	3670452594	99.9 %	3670545943	99.9 %	3670637149
Menos Cercano	75.0 %	3670455328	99.5 %	3670548464	80.0 %	3670666927

La distribución de probabilidades de robos de las mejores soluciones encontradas por cada algoritmo se muestran en las figuras 7.4, 7.5 y 7.6, estas se aplicaran sobre los datos de robos de los meses de Mayo a Diciembre con la Prueba χ^2 de Pearson para confirmar si conservan el porcentaje de aceptación de la hipótesis con esta información.

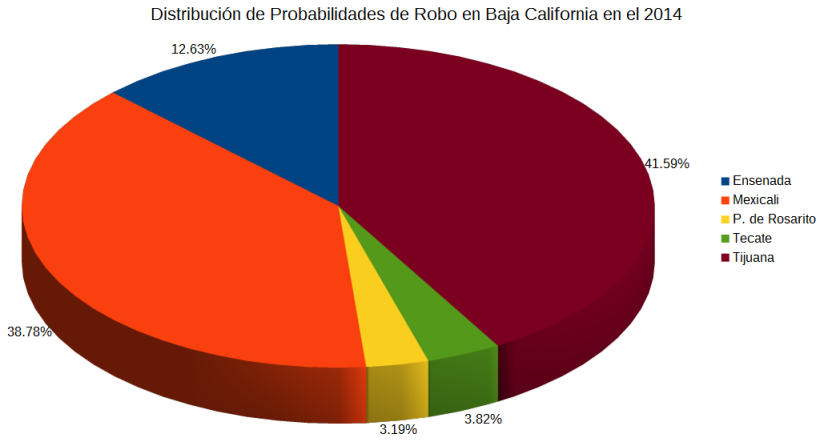

Figura 7.4: Mejor solución para los robos en el estado de Baja California por el algoritmo OEP

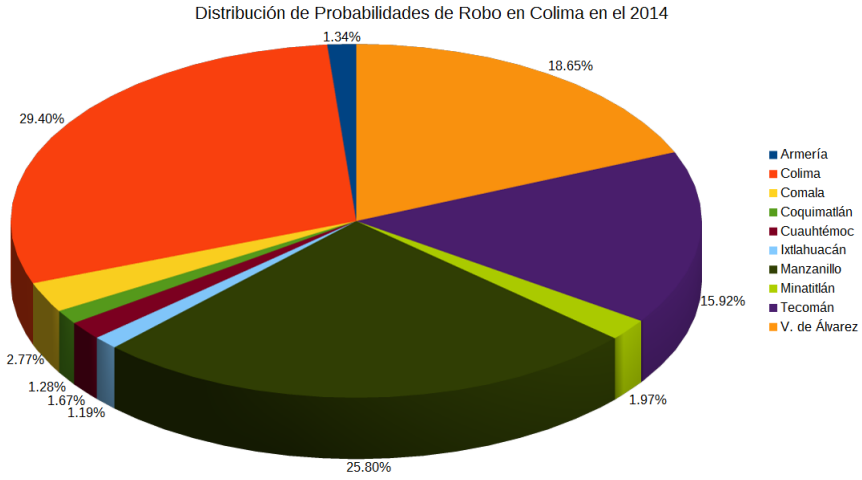

Figura 7.5: Mejor solución para los robos en el estado de Colima por el algoritmo OEP

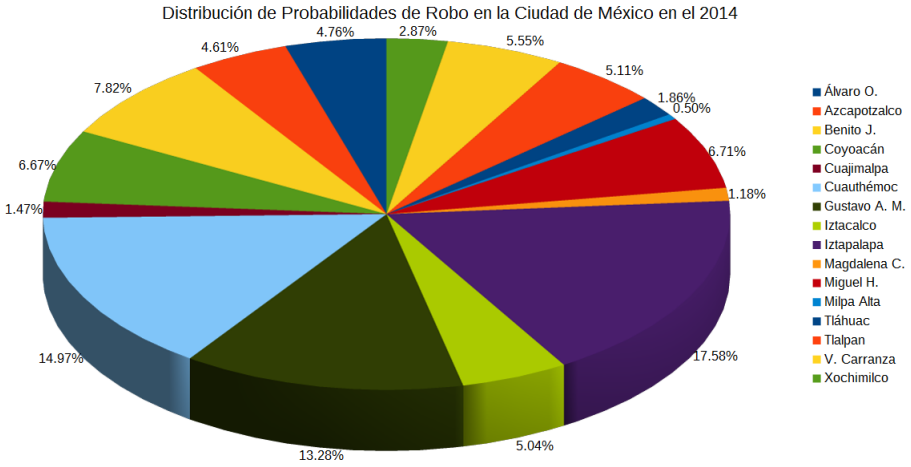

Figura 7.6: Mejor solución para los robos en Ciudad de México por el algoritmo OEP

En la tabla 7.7 se reportan los porcentajes de aceptación de los resultados del algoritmo para los estados de Baja California, Colima y la Ciudad de México para los meses de Mayo a Diciembre del 2014.

Puede apreciarse que para el caso de Colima la solución encontrada fue bastante buena al cumplir una aceptación del 90.0 %, a excepción del mes de noviembre, para el caso de la Ciudad de México paso algo similar en dos meses (Agosto y Noviembre en este caso) y para el estado de Baja California la mejor solución solo consiguió una certeza mayor al 90.0 % en la mitad de los casos.

Tabla 7.7: Porcentajes de predicción obtenidos por el algoritmo OEP

Estado	Baja California							
Mes	Mayo	Junio	Julio	Agosto	Septiembre	Octubre	Noviembre	Diciembre
Porcentaje	90.0 %	95.0 %	66.7 %	90.0 %	50.0 %	80.0 %	50.0 %	95.0 %
Estado	Colima							
Mes	Mayo	Junio	Julio	Agosto	Septiembre	Octubre	Noviembre	Diciembre
Porcentaje	97.5 %	99.0 %	90.0 %	95.0 %	97.5 %	95.0 %	87.5 %	90.0 %
Estado	Ciudad de México							
Mes	Mayo	Junio	Julio	Agosto	Septiembre	Octubre	Noviembre	Diciembre
Porcentaje	90.0 %	90.0 %	80.0 %	90.0 %	97.5 %	97.5 %	75.0 %	95.0 %

7.3. Resultados del Algoritmo ED con datos reales de robos

Para cada una de las versiones del algoritmo ED que se trabajaron se utilizo la misma configuración que para el algoritmos OEP, una población de 300 individuos en 3000 generaciones y se realizaron 10 iteraciones de cada una de estas. Las tablas 7.8, 7.9, 7.10 y 7.11 contienen los resultados de las iteraciones del algoritmo en sus 4 versiones.

Tabla 7.8: Mejores aptitudes reportadas con datos de robos del algoritmos ED/-Mejor/1/bin

Versión		ED/Mejor/1/bin			
Baja California		Colima		Ciudad de México	
Aptitud	Generaciones	Aptitud	Generaciones	Aptitud	Generaciones
3.617725	3000	12.751558	12	25.285467	37
3.617732	3000	8.771133	4	24.826199	58
3.617725	3000	7.065938	6	25.616709	34
3.617730	3000	11.027037	5	18.094816	56
3.617736	3000	11.57715	4	20.778286	95
3.617723	3000	10.304935	5	22.314327	79
3.617730	3000	11.684412	5	21.523218	40
3.617729	3000	9.494832	4	23.34248	29
3.617734	3000	12.078531	3	20.484644	112
3.617725	3000	8.676829	3	21.326916	90

Tabla 7.9: Mejores aptitudes reportadas con datos de robos del algoritmos ED/-Mejor/1/exp

Versión		ED/Mejor/1/exp			
Baja California		Colima		Ciudad de México	
Aptitud	Generaciones	Aptitud	Generaciones	Aptitud	Generaciones
3.693352	3000	26.42338	3000	306.4713	3000
16.836393	3000	31.379444	3000	503.6977	3000
13.243774	3000	25.497614	3000	259.80194	3000
19.880154	3000	34.23687	3000	622.5872	3000
6.552989	3000	41.16091	3000	463.60895	3000
7.236886	3000	33.684868	3000	552.8866	3000
22.83366	3000	16.72671	3000	471.30533	3000
9.3164	3000	46.1437	3000	233.467448	3000
4.703574	3000	41.404377	3000	319.7758	3000
3.893437	3000	19.498535	3000	519.7007	3000

Tabla 7.10: Mejores aptitudes reportadas con datos de robos del algoritmos ED/Aleatorio/1/bin

Versión		ED/Aleatorio/1/bin			
Baja California		Colima		Ciudad de México	
Aptitud	Generaciones	Aptitud	Generaciones	Aptitud	Generaciones
3.617720	3000	8.501623	88	24.895264	690
3.617720	3000	7.487829	128	26.795942	732
3.617720	3000	10.402041	83	27.214062	626
3.617726	3000	9.872558	88	26.652952	702
3.617719	3000	11.275526	112	27.106625	665
7.867635	3000	10.840998	129	22.884188	735
3.617720	3000	8.401878	96	23.858187	680
3.617720	3000	6.280022	81	25.312464	521
3.617720	3000	10.252933	16	16.565928	756
3.617720	3000	9.96657	93	23.560726	567

Tabla 7.11: Mejores aptitudes reportadas con datos de robos del algoritmos ED/Aleatorio/1/exp

Versión		ED/Aleatorio/1/exp			
Baja California		Colima		Ciudad de México	
Aptitud	Generaciones	Aptitud	Generaciones	Aptitud	Generaciones
3.693584	3000	37.247185	3000	522.437	3000
8.721437	3000	31.098179	3000	495.77838	3000
8.829178	3000	30.699982	3000	463.40106	3000
3.648800	3000	24.34643	3000	422.70898	3000
20.9059	3000	38.98315	3000	605.5958	3000
11.237223	3000	13.234553	3000	526.7756	3000
15.987518	3000	31.048697	3000	502.53906	3000
6.990506	3000	41.329327	3000	379.8599	3000
23.160458	3000	29.122046	3000	410.42517	3000
6.195704	3000	27.564108	3000	328.7921	3000

Las gráficas 7.7, 7.8 y 7.9 muestran la evolución de las aptitudes de las mejores soluciones obtenidos por las 4 versiones del algoritmo, puede observarse que la convergencia a una solución optima se alcanzo en pocas generaciones para los casos de Colima y la Ciudad de México (81 generaciones y 756 respectivamente), pero el problema de los pocos resultados persiste en los casos con pocos posibles resultados como en el de Baja California haciendo uso de todas las generaciones programadas.

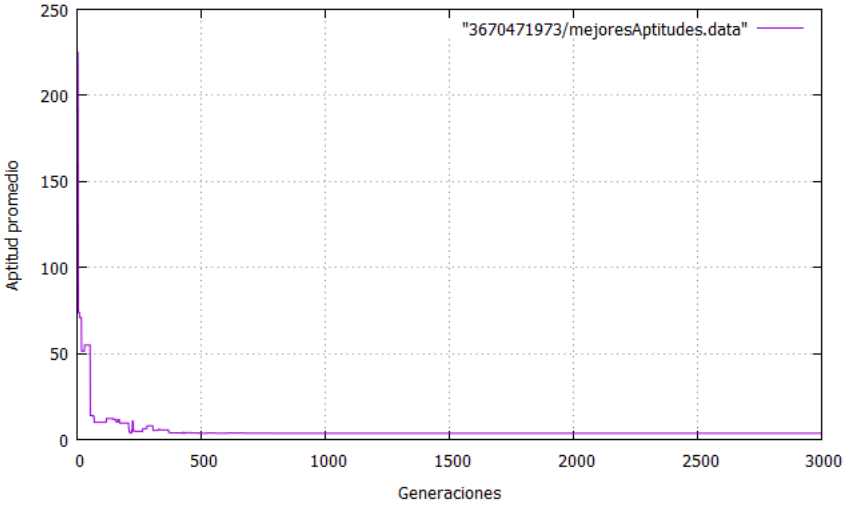

Figura 7.7: Evolución de la aptitud para el caso del estado de Baja California con el algoritmo ED

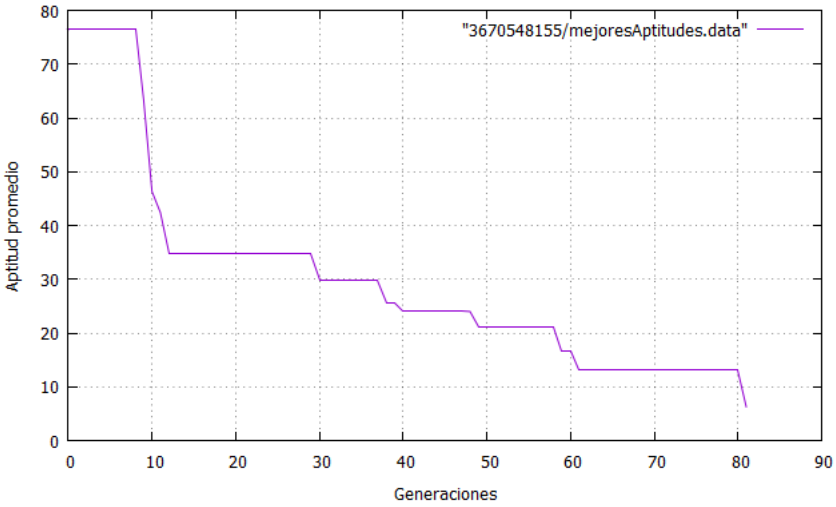

Figura 7.8: Evolución de la aptitud para el caso del estado de Colima con el algoritmo ED

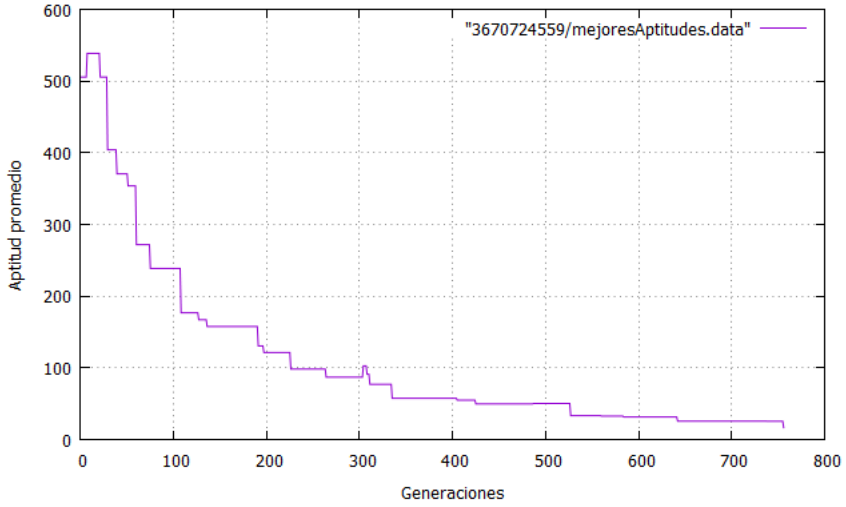

Figura 7.9: Evolución de la aptitud para el caso de la Ciudad de México con el algoritmo ED

En los resultados obtenidos en los experimentos con este algoritmo se puede apreciar que las versiones con operador binario tuvieron un mejor efecto en la búsqueda de soluciones, ya que en estos en los casos de los estados de Colima y la Ciudad de México no hizo falta el hacer uso de las 3000 generaciones para encontrar una respuesta que cumpliera con el 90.0 % de aceptación para los datos de robos de los primeros 4 meses del 2014. En la tabla 7.12 se muestran los rangos de probabilidades obtenidos con el algoritmo.

Tabla 7.12: Rangos de probabilidades conseguidos con el algoritmo ED

Versión	ED/Mejor/1/bin					
Estado	Baja California		Colima		Ciudad de México	
	Porcentaje	Experimento	Porcentaje	Experimento	Porcentaje	Experimento
Más Cercano	90.0 %	3670467589	99.9 %	3670547876	99.9 %	3670721580
Menos Cercano	90.0 %	3670466620	99.5 %	3670547872	99.9 %	3670721568
Versión	ED/Mejor/1/exp					
Estado	Baja California		Colima		Ciudad de México	
	Porcentaje	Experimento	Porcentaje	Experimento	Porcentaje	Experimento
Más Cercano	90.0 %	3670463850	99.0 %	3670556059	40.0 %	3670742693
Menos Cercano	20.0 %	3670468038	80.0 %	3670557046	0.0 %	3670730390
Versión	ED/Aleatorio/1/bin					
Estado	Baja California		Colima		Ciudad de México	
	Porcentaje	Experimento	Porcentaje	Experimento	Porcentaje	Experimento
Más Cercano	90.0 %	3670471973	99.9 %	3670548155	99.9 %	3670724559
Menos Cercano	66.7 %	3670472750	99.5 %	3670547998	99.9 %	3670722608
Versión	ED/Aleatorio/1/exp					
Estado	Baja California		Colima		Ciudad de México	
	Porcentaje	Experimento	Porcentaje	Experimento	Porcentaje	Experimento
Más Cercano	90.0 %	3670471598	99.5 %	3670553791	10.0 %	3670747787
Menos Cercano	20.0 %	3670475892	80.0 %	3670555236	0.0 %	3670733606

Puede observarse que la efectividad del algoritmo en las versiones con operador de cruza exponencial no dio muy buenos resultados para el caso de predicción para la Ciudad de México que es el que tiene el mayor número de posibles puntos de robo, y para el caso de Baja California el rango de probabilidades que se obtiene es demasiado grande, pero para el caso de las versiones con operador binario se mantienen resultados buenos con rangos de probabilidades cercanos a 0 en casi todos los casos excepto la configuración del padre aleatorio para el primer caso correspondiente al estado de Baja California.

Las mejores soluciones encontradas por este algoritmos se muestran en las gráficas 7.10, 7.11 y 7.12 para los casos de de los tres estados, estas se aplicaron a los meses de Mayo a Diciembre y se obtuvieron los porcentajes de aceptación que se registran en la tabla 7.13.

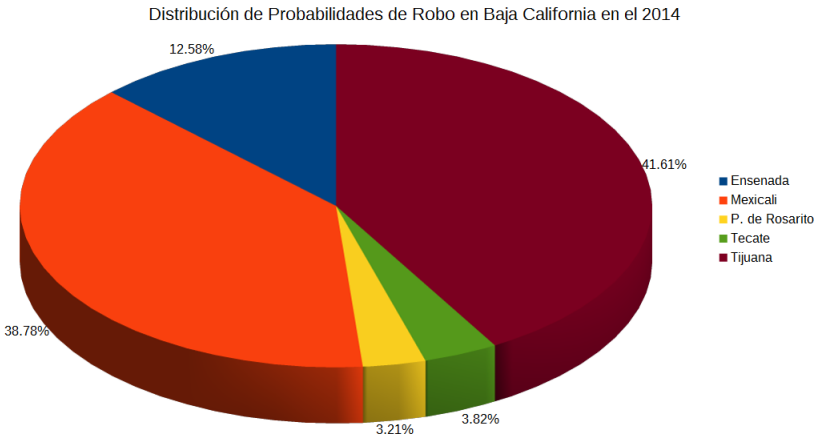

Figura 7.10: Mejor solución para los robos en el estado de Baja California por el algoritmo ED

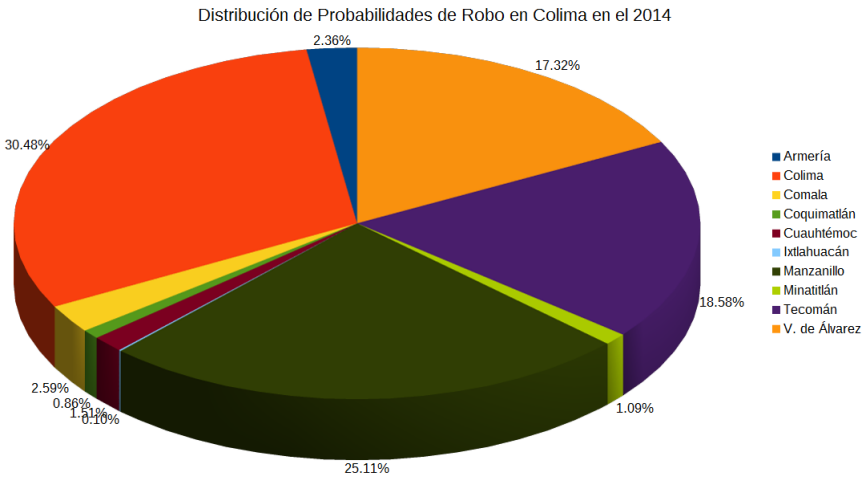

Figura 7.11: Mejor solución para los robos en el estado de Colima por el algoritmo ED

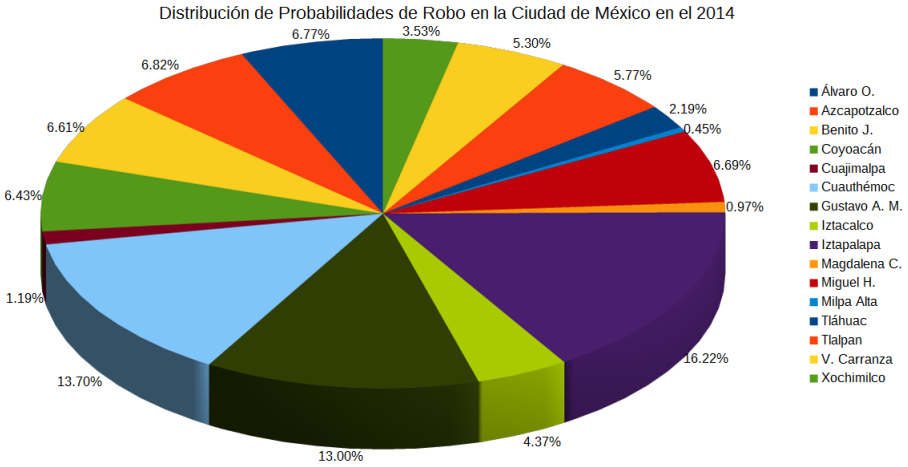

Figura 7.12: Mejor solución para los robos en Ciudad de México por el algoritmo ED

Para el caso de Baja California resulto algo similar que con el algoritmo OEP, solo 4 de los 8 meses consiguieron una aceptación mayor al 90.0 %, en el caso de Colima se obtuvieron en todas aceptaciones superiores al 95.0 % en la mayoría y finalmente en el caso de la Ciudad de México se alcanzo un buen resultado excepto para el caso de robos del mes de Diciembre donde solo se alcanzo una aceptación del 66.7 %.

Tabla 7.13: Porcentajes de predicción obtenidos por el algoritmo ED

Estado	Baja California							
Mes	Mayo	Junio	Julio	Agosto	Septiembre	Octubre	Noviembre	Diciembre
Porcentaje	90.0 %	95.0 %	80.0 %	90.0 %	50.0 %	80.0 %	50.0 %	90.0 %
Estado	Colima							
Mes	Mayo	Junio	Julio	Agosto	Septiembre	Octubre	Noviembre	Diciembre
Porcentaje	99.5 %	99.0 %	95.0 %	97.5 %	97.5 %	97.5 %	90.0 %	99.0 %
Estado	Ciudad de México							
Mes	Mayo	Junio	Julio	Agosto	Septiembre	Octubre	Noviembre	Diciembre
Porcentaje	97.5 %	99.0 %	90.0 %	95.0 %	90.0 %	90.0 %	95.0 %	66.7 %

Puede concluirse que el algoritmo ED muestra resultados muy buenos en para la tarea de predicción de robos, sobre todo las versiones de cruza binaria, sin mostrar mucha diferencia entre las versiones con el mejor individuo o un individuo aleatorio para ser perturbado en el proceso de cruza.

7.4. Resultados del Algoritmo EE con datos reales de robos

Para el la experimentación con el algoritmo EE se utilizo una población de 50 individuos, con generación de 7 hijos por individuo en 3000 generaciones, para este algoritmo se hicieron 5 iteraciones para cada caso debido a que este algoritmo difiere mucho en tiempo con respecto a los otros dos algoritmos aumentando considerablemente. La tabla 7.14 muestra los resultados de estas iteraciones con el algoritmos EE y las gráficas 7.13, 7.14 y 7.15 muestran la evolución de la aptitud de las mejores soluciones encontradas en cada versión.

Tabla 7.14: Mejores aptitudes reportadas con datos de robos del algoritmos EE

Versión		$(\mu + \lambda) - EE$			
Baja California		Colima		Ciudad de México	
Aptitud	Generaciones	Aptitud	Generaciones	Aptitud	Generaciones
261.6983	3000	73.788280	3000	426.184425	3000
921.959	3000	105.155014	3000	278.954245	3000
302.1289	3000	107.92633	3000	616.859086	3000
480.28262	3000	82.655968	3000	463.029196	3000
586.15753	3000	73.788280	3000	792.855113	3000
Versión		$(\mu, \lambda) - EE$			
Baja California		Colima		Ciudad de México	
Aptitud	Generaciones	Aptitud	Generaciones	Aptitud	Generaciones
2640.464499	3000	197.569599	3000	867.945282	3000
3640.8430174	3000	197.570908	3000	867.793472	3000
2641.454826	3000	197.580875	3000	867.99354291	3000
2641.115259	3000	197.559446	3000	867.845697	3000
2640.747139	3000	197.569599	3000	868.115987	3000

En la tabla 7.14 puede verse que el algoritmo en ningún momento logro encontrar una solución que cumpliera con el criterio de paro de una aceptación mínima del 90.0 % para todas las distribuciones, incluso puede apreciarse al comprarse con los resultados de los algoritmos anteriores que su desempeño fue bastante bajo comparado con ellos.

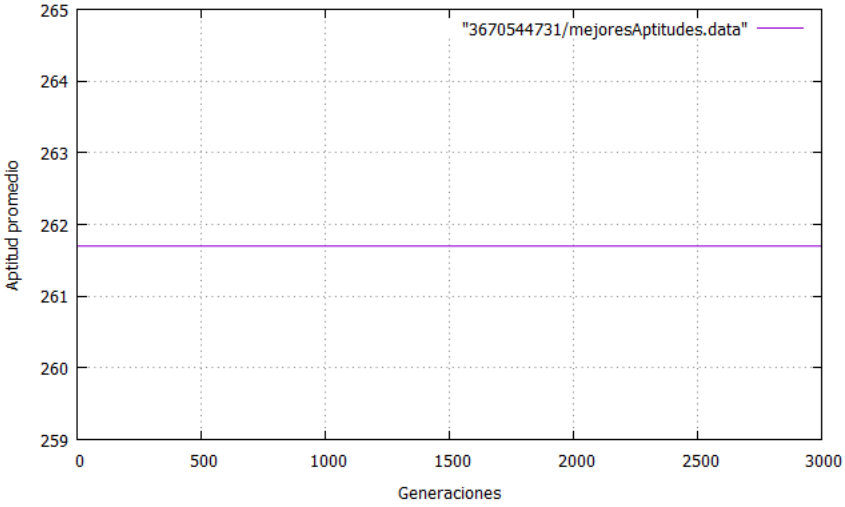

Figura 7.13: Evolución de la aptitud para el caso del estado de Baja California con el algoritmo EE

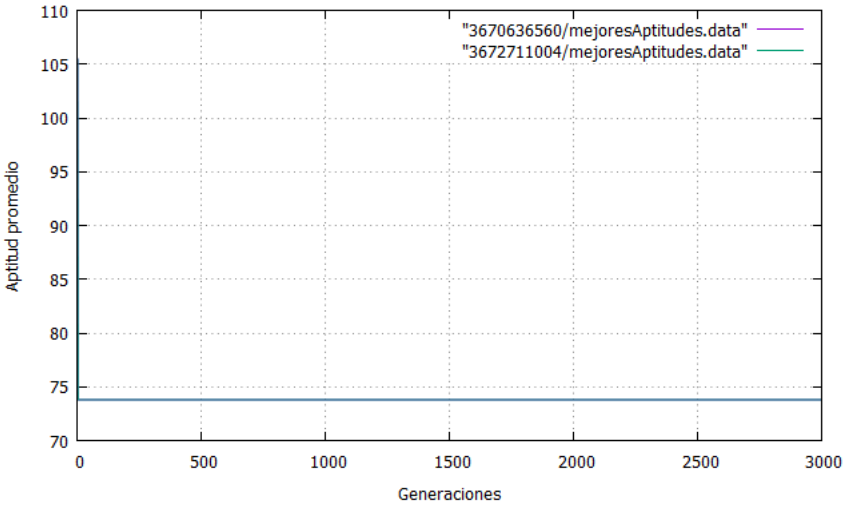

Figura 7.14: Evolución de la aptitud para el caso del estado de Colima con el algoritmo EE

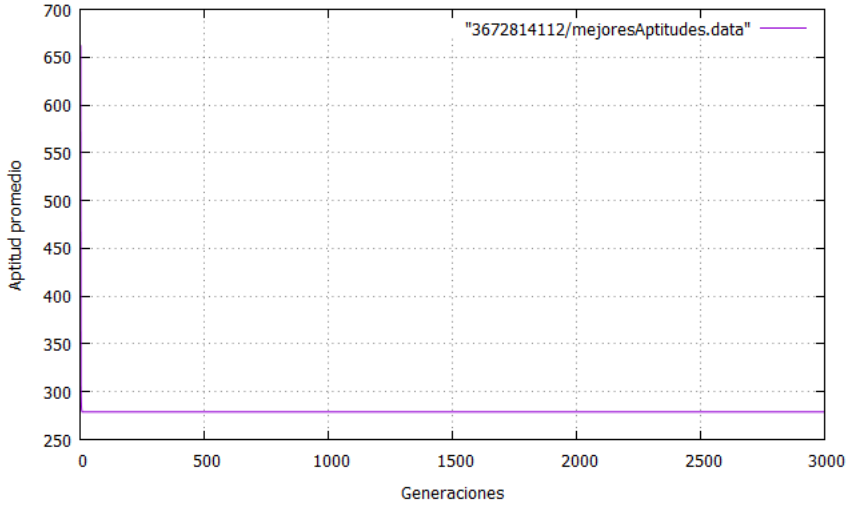

Figura 7.15: Evolución de la aptitud para el caso de la Ciudad de México con el algoritmo EE

Los rangos de probabilidades que se obtuvieron con este algoritmo se encuentran en la tabla 7.15, puede observarse que no se consiguieron buenos resultados con este algoritmo, las mejores soluciones no alcanzaron porcentajes altos en la mayoría no superan el 0.0 %, mientras que en los casos donde se consiguió un porcentaje alejado de este el rango de probabilidades quedo muy grande.

Tabla 7.15: Rangos de probabilidades conseguidos con el algoritmo EE

Versión	$(\mu + \lambda)$ - EE					
Estado	Baja California		Colima		Ciudad de México	
	Porcentaje	Experimento	Porcentaje	Experimento	Porcentaje	Experimento
Más Cercano	0.0 %	3670544731	40.0 %	Varios	20.0 %	3672814112
Menos Cercano	0.0 %	3670549911	12.5 %	3670747203	0.0 %	3672864089
Versión	(μ, λ) - EE					
Estado	Baja California		Colima		Ciudad de México	
	Porcentaje	Experimento	Porcentaje	Experimento	Porcentaje	Experimento
Más Cercano	0.0 %	3670541620	0.5 %	3670751896	0.0 %	3672812193
Menos Cercano	0.0 %	3670551574	0.5 %	3670745804	0.0 %	3672856094

Las figuras 7.16, 7.17 y 7.18 muestran las mejores distribuciones encontradas por el algoritmo EE para cada caso.

La Tabla 7.16 muestra los porcentajes de aceptación de estos resultados con los datos de robos de los meses de Mayo a Diciembre del 2014 que fueron encontrados con el algoritmo EE.

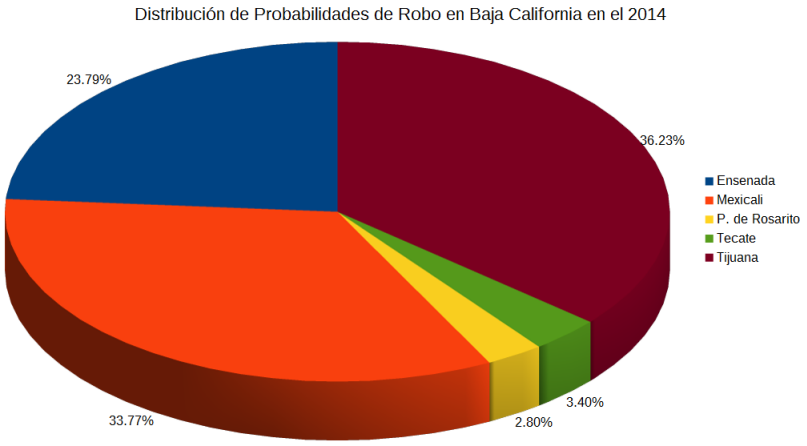

Figura 7.16: Mejor solución para los robos en el estado de Baja California por el algoritmo EE

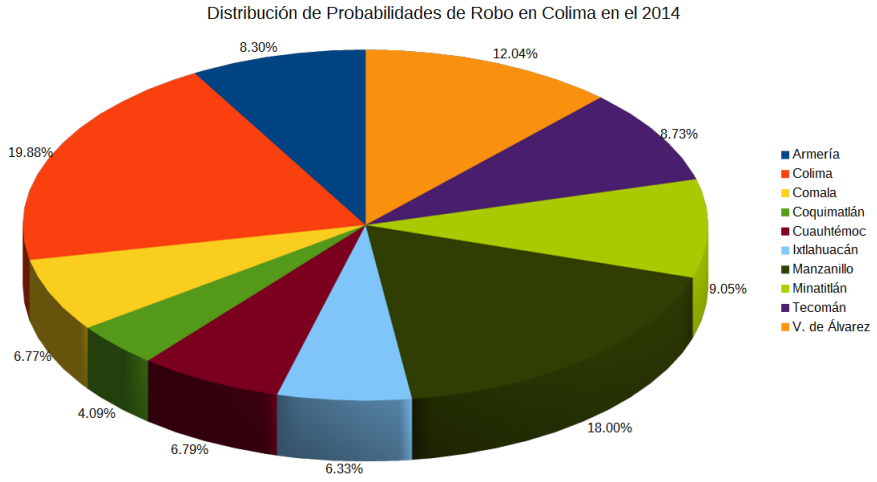

Figura 7.17: Mejor solución para los robos en el estado de Colima por el algoritmo EE

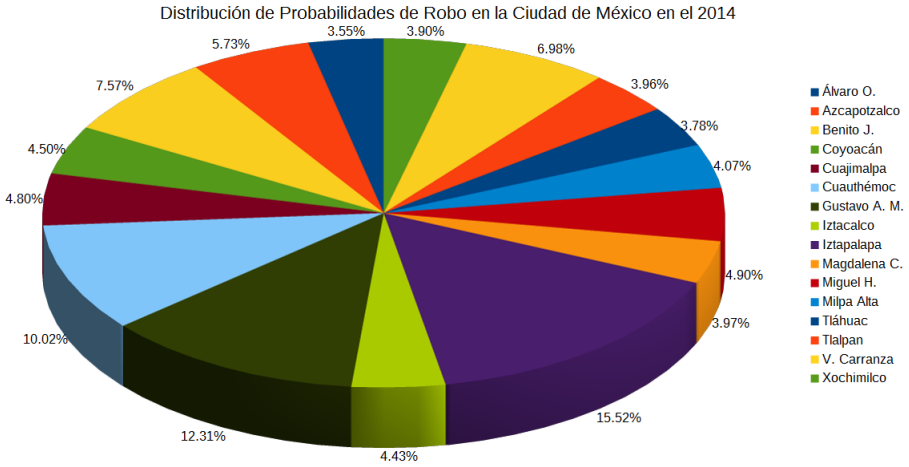

Figura 7.18: Mejor solución para los robos en Ciudad de México por el algoritmo EE

Tabla 7.16: Porcentajes de predicción obtenidos por el algoritmo EE

Estado	Baja California							
Mes	Mayo	Junio	Julio	Agosto	Septiembre	Octubre	Noviembre	Diciembre
Porcentaje	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %
Estado	Colima							
Mes	Mayo	Junio	Julio	Agosto	Septiembre	Octubre	Noviembre	Diciembre
Porcentaje	2.5 %	2.5 %	0.5 %	0.5 %	5.0 %	1.0 %	0.5 %	2.5 %
Estado	Ciudad de México							
Mes	Mayo	Junio	Julio	Agosto	Septiembre	Octubre	Noviembre	Diciembre
Porcentaje	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %	0.0 %

Los resultados del porcentaje de aceptación en los datos de los robos de los meses de Mayo a Diciembre muestran que el desempeño del algoritmo EE fue bastante malo, si bien para el estado de Colima no quedo en 0.0 %, se nota que el algoritmo necesita ajustes a sus parámetros como una población más grande o más ciclos de búsqueda. pero como se menciono en el inicio de esta sección, este algoritmo demostró el necesitar un tiempo mucho mayor en comparativa con los otros, lo que representa una gran desventaja considerando las soluciones encontrados por estos antes en menos tiempos son mucho mejores.

7.5. Comparación de resultados

En las gráficas 7.19. 7.20 y 7.21 se observa la comparativa entre las mejores soluciones encontradas por cada algoritmo para las distribuciones de robos en los 3 estados.

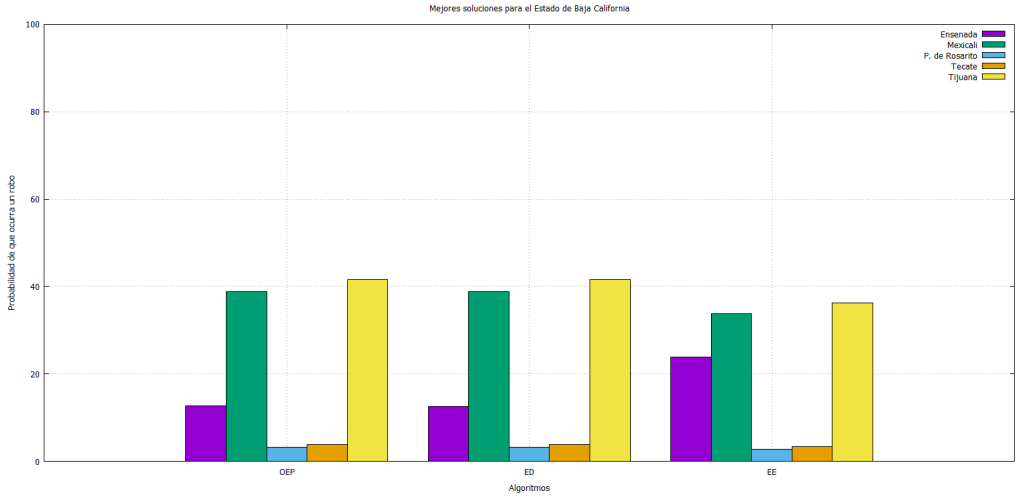

Figura 7.19: Comparación de las soluciones para la distribución de robos en el estado de Baja California

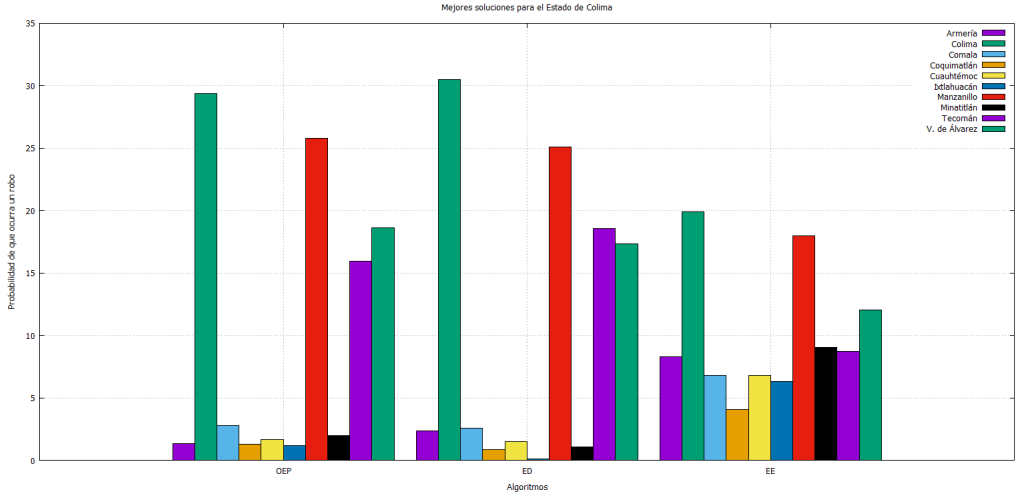

Figura 7.20: Comparación de las soluciones para la distribución de robos en el estado de Colima

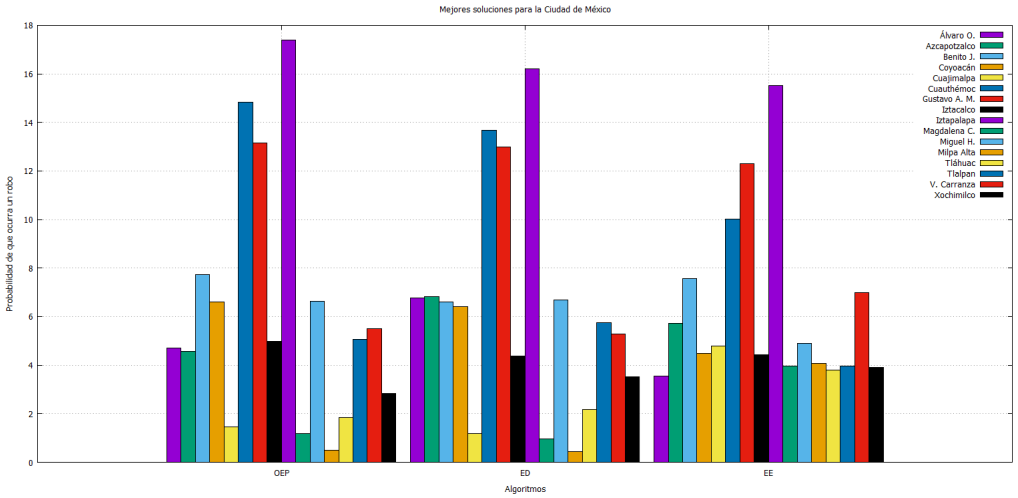

Figura 7.21: Comparación de las soluciones para la distribución de robos en la Ciudad de México

Puede verse que para el estado de Baja California las distribuciones de probabilidades obtenidas por los algoritmos OEP y ED son muy similares, algo similar se observa en el caso del estado de Colima y para la Ciudad de México se observan mayores diferencias pero se conservan casi las mismas tendencias en las probabilidades de de ocurrencias de robos, y en los tres casos el algoritmos EE dio una soluciones bastante distintas.

Al comparar las tablas de los resultados de los tres algoritmos se puede observar que los algoritmos OEP y ED fueron los que mejores resultados tuvieron, el algoritmo EE no obtuvo resultados buenos en ninguno de los tres casos de predicción de robos, un aumento en los parámetros del tamaño de población podría mejorar el rendimiento de este algoritmo, sin embargo, el tiempo de ejecución de este aumenta mucho con respecto de los demás algoritmos, por lo que el aumentar este parámetro aumentaría considerablemente el tiempo invertido por este algoritmo par conseguir soluciones cercanas a las obtenidas por los otros dos. También puede observarse que para el estado de Baja California al parecer hubo problemas para que los algoritmos convergieran a una buena solución, esto sucedió también en los experimentos 2 al 4 del capitulo anterior para casos con 5 posibles resultados.

En las tablas 7.6 y 7.12 se puede observar que las mejores soluciones encontradas en los algoritmos OEP y ED fueron bastante buenas, en el caso del algoritmo ED son las versiones de operación binaria para el cruce resultaron mejores, con resultados buenos para los tres estados y rangos de probabilidades pequeños en comparación con los del algoritmo OEP, mostrando que el algo-

ritmo ED tuvo mejores resultados en cuanto a poder converger a una mejor solución.

El algoritmo ED en sus versiones con cruce binario para los estados de Colima y la Ciudad de México lograron conseguir converger sin hacer uso de todas las generaciones, en el algoritmos OEP solo lo hizo con el estado de Colima, por lo que en este ámbito el algoritmo ED supero al algoritmo OEP para casos con una mayor cantidad de posibles soluciones para un evento.

Finalmente para el porcentaje de aceptación de los datos de robos de meses posteriores a los utilizados por los algoritmos para el proceso de búsqueda de la mejor distribución de robos, para el caso de Baja California las mejores soluciones de los algoritmos OEP y ED fueron similares, pero para los otros dos estados la mejor solución dada por el algoritmo ED demostró cumplir mejor con el porcentaje de aceptación que se esperaba en general para los datos de los robos.

En conclusión se puede decir que el algoritmo ED en sus versiones de cruza binaria resulto ser el mejor de los tres para realizar la tarea de predicción de robos haciendo uso de la Prueba χ^2 de Pearson, siendo mejor en diferentes aspectos que se tomaron en cuenta como el tiempo de ejecución, forma de recorrer el espacio de búsqueda para no usar todas las generaciones y la efectividad de la solución encontrada con los datos de robos futuros, el algoritmo OEP obtuvo buenos resultados cercanos a los del algoritmo ED por lo que demostro ser también una buena opción para la tarea de predicción, finalmente para el algoritmo EE se especula que puede mejorar su rendimiento en cuando a convergencia modificando sus parámetros, pero esto implicaría una gran perdida en cuanto a tiempo necesario para esto.

Capítulo 8

Conclusiones y Trabajo a Futuro

En el presente capitulo se reportan las conclusiones finales así como el posible trabajo a futuro que puede abordarse basado en la experimentación previa y sus resultados reportados en capítulos previos. Las conclusiones finales se tomaron en cuenta basados en los resultados obtenidos por los algoritmos, sus configuraciones y tiempo de ejecución de estos, y el trabajo final sugerido esta enfocado en aumentar el área de investigación de este mismo proyecto y aplicaciones distintas para el conocimiento adquirido en el mismo.

8.1. Conclusiones

La propuesta de investigación de este trabajo es probar el uso de técnicas evolutivas en conjunto con la Prueba χ^2 de Pearson para realizar la tarea de distribución de probabilidades de robos. Los tres algoritmos seleccionados para este trabajo fueron: Optimización por Enjambre de Partículas, Evolución Diferencial (4 versiones de este) y Estrategias Evolutivas (2 versiones de este).

En los resultados de los experimentos 2 al 5 se pudo apreciar que para casos con pocos posibles resultados para la Prueba χ^2 de Pearson los algoritmos tienen problemas para converger, lo que indica que deben considerarse bien el número de los puntos de robo que se análisis para el trabajo de predicción, ya que un número pequeño de posibles puntos para que ocurra un robo no implica que los algoritmos convergían rápidamente a una solución optima.

Analizando los resultados obtenidos por los algoritmos en el la experimentación con datos de robos, las versiones del algoritmo de Evolución Diferencial con cruza binaria consiguieron resultados buenos para los casos de Colima y Ciudad de México logrando converger a soluciones que cumplieron con el porcentaje de aceptación que se busco, y consiguieron porcentajes de aceptación buenos con los datos de los robos posteriores en el mismo año en general, por otro lado el algoritmo de Optimización por Enjambre de Partículas consiguió resultados similares para el estado de Colima pero para el caso de la Ciudad de México no fue así, a pesar de eso sus resultados no fueron malos, por lo que probablemente se deban hacer ajustes a sus parámetros para conseguir esto, esto puede ser en el tamaño de población aumentándola para ampliar el espació de búsqueda. En cuanto al tiempo de ejecución de los algoritmos, la Evolución Diferencial y la Optimización por Enjambre de Partículas tuvieron tiempos casi iguales de ejecución en sus iteraciones, mientras que las Estrategias Evolutivas aumentaron de manera drástica los tiempos de ejecución, siendo estos a veces hasta 5 veces más largos que los de los otros dos algoritmos para realizar la misma tarea.

Después de revisar los resultados finales de los algoritmos y el comparar sus tiempos de ejecución y rangos de probabilidades obtenidos se puede concluir que el algoritmo de Evolución Diferencial en sus versiones de cruza binaria demostró ser la mejor opción de las 3 técnicas estudiadas en este trabajo, seguido por el algoritmo de Optimización por Enjambre de Partículas con resultados muy cercanos y que posiblemente sean casi iguales con leves cambios en sus parámetros, y también se demostró que para este problema en particular el algoritmo de Estrategias Evolutivas no es una buena opción como posible solución.

8.2. Trabajo a futuro

Después de demostrar que se puede convertir el problema de predicción en un problema de optimización de funciones haciendo uso de la Prueba χ^2 de Pearson y este puede ser resuelto por técnicas evolutivas, se puede aplicar lo realizado en otras tareas de predicción de probabilidades basadas en datos estadísticos.

Algunas tareas de predicción que se pueden abordar son:

- Otros tipos de crímenes como secuestros y asesinatos.
- Clima.
- Ventas de bienes y/o servicios.

De igual manera, al ver los resultados de la comparación de la técnicas evolutivas utilizadas en este trabajo y observar que no todas pueden abordar el

problema de predicción de la manera en que se planteo adecuadamente, otro trabajo futuro que puede abordarse es la investigación con otras técnicas evolutivas o también otras heurísticas para atacar este mismo problema, por ejemplo otras técnicas que pueden ser investigadas son:

- Colonia de Hormigas.
- Sistema Inmune Artificial.
- Búsqueda Armónica Global.
- Algoritmos Meméticos
- Algoritmos Evolutivos Distribuidos

Estas no son las únicas, existen más y otras variantes de los mismos algoritmos utilizados en el presente trabajo que pueden ser investigadas también.

Apéndice A

Distribución χ^2

La gráfica A.1 muestra la forma de las distribuciones tipo χ^2, como se puede observar estas no son simétricas.

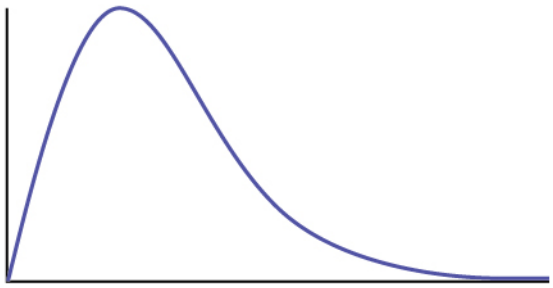

Figura A.1: Distribución χ^2

Las tablas A.1 y A.2 muestran los valores de χ^2 para grados de libertad del 1 al 60, para calcular esta para grados mayores a 60 se utiliza la ecuación A.1.

$$\chi^2_{t,\alpha} \approx 0{,}5(z_\alpha + \sqrt{2t-1}) \tag{A.1}$$

Donde z_α se obtiene de una distribución normal acumulativa.

Tabla A.1: Distribución χ^2 (Parte 1)

t	99.9 %	99.5 %	99.0 %	97.5 %	95.0 %	90.0 %	87.5 %	80.0 %	75.0 %	66.7 %	50.0 %
1	0.000	0.000	0.000	0.001	0.004	0.016	0.025	0.064	0.102	0.186	0.455
2	0.002	0.010	0.020	0.051	0.103	0.211	0.267	0.446	0.575	0.811	1.386
3	0.024	0.072	0.115	0.216	0.352	0.584	0.692	1.005	1.213	1.568	2.366
4	0.091	0.207	0.297	0.484	0.711	1.064	1.219	1.649	1.923	2.378	3.357
5	0.210	0.412	0.554	0.831	1.145	1.610	1.808	2.343	2.675	3.216	4.351
6	0.381	0.676	0.872	1.237	1.635	2.204	2.441	3.070	3.455	4.074	5.348
7	0.598	0.989	1.239	1.690	2.167	2.833	3.106	3.822	4.255	4.945	6.346
8	0.857	1.344	1.646	2.180	2.733	3.490	3.797	4.594	5.071	5.826	7.344
9	1.152	1.735	2.088	2.700	3.325	4.168	4.507	5.380	5.899	6.716	8.343
10	1.479	2.156	2.558	3.247	3.940	4.865	5.234	6.179	6.737	7.612	9.342
11	1.834	2.603	3.053	3.816	4.575	5.578	5.975	6.989	7.584	8.514	10.341
12	2.214	3.074	3.571	4.404	5.226	6.304	6.729	7.807	8.438	9.420	11.340
13	2.617	3.565	4.107	5.009	5.892	7.042	7.493	8.634	9.299	10.331	12.340
14	3.041	4.075	4.660	5.629	6.571	7.790	8.266	9.467	10.165	11.245	13.339
15	3.483	4.601	5.229	6.262	7.261	8.547	9.048	10.307	11.037	12.163	14.339
16	3.942	5.142	5.812	6.908	7.962	9.312	9.837	11.152	11.912	13.083	15.338
17	4.416	5.697	6.408	7.564	8.672	10.085	10.633	12.002	12.792	14.006	16.338
18	4.905	6.265	7.015	8.231	9.390	10.865	11.435	12.857	13.675	14.931	17.338
19	5.407	6.844	7.633	8.907	10.117	11.651	12.242	13.716	14.562	15.859	18.338
20	5.921	7.434	8.260	9.591	10.851	12.443	13.055	14.578	15.452	16.788	19.337
21	6.447	8.034	8.897	10.283	11.591	13.240	13.873	15.445	16.344	17.720	20.337
22	6.983	8.643	9.542	10.982	12.338	14.041	14.695	16.314	17.240	18.653	21.337
23	7.529	9.260	10.196	11.689	13.091	14.848	15.521	17.187	18.137	19.587	22.337
24	8.085	9.886	10.856	12.401	13.848	15.659	16.351	18.062	19.037	20.523	23.337
25	8.649	10.520	11.524	13.120	14.611	16.473	17.184	18.940	19.939	21.461	24.337
26	9.222	11.160	12.198	13.844	15.379	17.292	18.021	19.820	20.843	22.399	25.336
27	9.803	11.808	12.879	14.573	16.151	18.114	18.861	20.703	21.749	23.339	26.336
28	10.391	12.461	13.565	15.308	16.928	18.939	19.704	21.588	22.657	24.280	27.336
29	10.986	13.121	14.256	16.047	17.708	19.768	20.550	22.475	23.567	25.222	28.336
30	11.588	13.787	14.953	16.791	18.493	20.599	21.399	23.364	24.478	26.165	29.336
35	14.688	17.192	18.509	20.569	22.465	24.797	25.678	27.836	29.054	30.894	34.336
40	17.916	20.707	22.164	24.433	26.509	29.051	30.008	32.345	33.660	35.643	39.335
45	21.251	24.311	25.901	28.366	30.612	33.350	34.379	36.884	38.291	40.407	44.335
50	24.674	27.991	29.707	32.357	34.764	37.689	38.785	41.449	42.942	45.184	49.335
55	28.173	31.735	33.570	36.398	38.958	42.060	43.220	46.036	47.610	49.972	54.335
60	31.738	35.534	37.485	40.482	43.188	46.459	47.680	50.641	52.294	54.770	59.335

Tabla A.2: Distribución χ^2 (Parte 2)

t	40.0 %	33.3 %	25.0 %	20.0 %	12.5 %	10.0 %	5.0 %	2.5 %	1.0 %	0.5 %	00.1 %
1	0.708	0.936	1.323	1.642	2.354	2.706	3.841	5.024	6.635	7.879	10.828
2	1.833	2.197	2.773	3.219	4.159	4.605	5.991	7.378	9.210	10.597	13.816
3	2.946	3.405	4.108	4.642	5.739	6.251	7.815	9.348	11.345	12.838	16.266
4	4.045	4.579	5.385	5.989	7.214	7.779	9.488	11.143	13.277	14.860	18.467
5	5.132	5.730	6.626	7.289	8.625	9.236	11.070	12.833	15.086	16.750	20.515
6	6.211	6.867	7.841	8.558	9.992	10.645	12.592	14.449	16.812	18.548	22.458
7	7.283	7.992	9.037	9.803	11.326	12.017	14.067	16.013	18.475	20.278	24.322
8	8.351	9.107	10.219	11.030	12.636	13.362	15.507	17.535	20.090	21.955	26.125
9	9.414	10.215	11.389	12.242	13.926	14.684	16.919	19.023	21.666	23.589	27.877
10	10.473	11.317	12.549	13.442	15.198	15.987	18.307	20.483	23.209	25.188	29.588
11	11.530	12.414	13.701	14.631	16.457	17.275	19.675	21.920	24.725	26.757	31.264
12	12.584	13.506	14.845	15.812	17.703	18.549	21.026	23.337	26.217	28.300	32.910
13	13.636	14.595	15.984	16.985	18.939	19.812	22.362	24.736	27.688	29.819	34.528
14	14.685	15.680	17.117	18.151	20.166	21.064	23.685	26.119	29.141	31.319	36.123
15	15.733	16.761	18.245	19.311	21.384	22.307	24.996	27.488	30.578	32.801	37.697
16	16.780	17.840	19.369	20.465	22.595	23.542	26.296	28.845	32.000	34.267	39.252
17	17.824	18.917	20.489	21.615	23.799	24.769	27.587	30.191	33.409	35.718	40.790
18	18.868	19.991	21.605	22.760	24.997	25.989	28.869	31.526	34.805	37.156	42.312
19	19.910	21.063	22.718	23.900	26.189	27.204	30.144	32.852	36.191	38.582	43.820
20	20.951	22.133	23.828	25.038	27.376	28.412	31.410	34.170	37.566	39.997	45.315
21	21.991	23.201	24.935	26.171	28.559	29.615	32.671	35.479	38.932	41.401	46.797
22	23.031	24.268	26.039	27.301	29.737	30.813	33.924	36.781	40.289	42.796	48.268
23	24.069	25.333	27.141	28.429	30.911	32.007	35.172	38.076	41.638	44.181	49.728
24	25.106	26.397	28.241	29.553	32.081	33.196	36.415	39.364	42.980	45.559	51.179
25	26.143	27.459	29.339	30.675	33.247	34.382	37.652	40.646	44.314	46.928	52.620
26	27.179	28.520	30.435	31.795	34.410	35.563	38.885	41.923	45.642	48.290	54.052
27	28.214	29.580	31.528	32.912	35.570	36.741	40.113	43.195	46.963	49.645	55.476
28	29.249	30.639	32.620	34.027	36.727	37.916	41.337	44.461	48.278	50.993	56.892
29	30.283	31.697	33.711	35.139	37.881	39.087	42.557	45.722	49.588	52.336	58.301
30	31.316	32.754	34.800	36.250	39.033	40.256	43.773	46.979	50.892	53.672	59.703
35	36.475	38.024	40.223	41.778	44.753	46.059	49.802	53.203	57.342	60.275	66.619
40	41.622	43.275	45.616	47.269	50.424	51.805	55.758	59.342	63.691	66.766	73.402
45	46.761	48.510	50.985	52.729	56.052	57.505	61.656	65.410	69.957	73.166	80.077
50	51.892	53.733	56.334	58.164	61.647	63.167	67.505	71.420	76.154	79.49	86.661
55	57.016	58.945	61.665	63.577	67.211	68.796	73.311	77.380	82.292	85.749	93.168
60	62.135	64.147	66.981	68.972	72.751	74.397	79.082	83.298	88.379	91.952	99.607

Apéndice B

Funciones de optimización del experimento 1

En esta sección se describen las funciones que se programaron para el Experimento 1 (Ver 6.1)

B.1. Función de Langermann

La Función de Langermann es una función multimodal, esto es, que tiene más de un valor que se considera como el valor global de optimización.

En este caso son mínimos globales, estos están distribuidos de manera no uniforme, lo que convierte a esta función es adecuada para probar el comportamiento de los algoritmos en el caso de que se encuentre con varias soluciones optimas que no se encuentran distribuidas de manera uniforme.

La función se define de la siguiente manera:

$$f(x_1, ..., x_n) = \sum_{i=1}^{m} c_i \exp[-\frac{1}{\pi} \sum_{j=1}^{n} (x_j - a_{ij})^2] \cos[\pi \sum_{j=1}^{n} (x_j - a_{ij})^2] \qquad \text{(B.1)}$$

Donde c_i y a_{ij} son constantes (con $i = 1, ..., m$ y $j = 1, ..., n$), por lo general se toma el valor de $m = 5$ y n es el número de dimensiones menos una de la función.

Para la implementación se usaran los siguientes valores propuestos por Molga y Smutnicki:

$$\begin{array}{l} m = 5 \\ n = 2 \\ c = \left\langle \begin{array}{ccccc} 1 & 2 & 5 & 2 & 3 \end{array} \right\rangle \end{array} \qquad a = \begin{pmatrix} 3 & 5 \\ 5 & 2 \\ 2 & 1 \\ 1 & 4 \\ 7 & 9 \end{pmatrix}$$

Esto hace que la ecuación B.1 quede para estos valores de la siguiente manera:

$$f(x_1, x_2) = \sum_{i=1}^{5} c_i \exp[-\frac{1}{\pi}\sum_{j=1}^{2}(x_j - a_{ij})^2] \cos[\pi \sum_{j=1}^{2}(x_j - a_{ij})^2] \tag{B.2}$$

En la figura B.1 puede observarse el comportamiento de la ecuación B.2 en un intervalo de $0 \leq x_j \leq 10$.

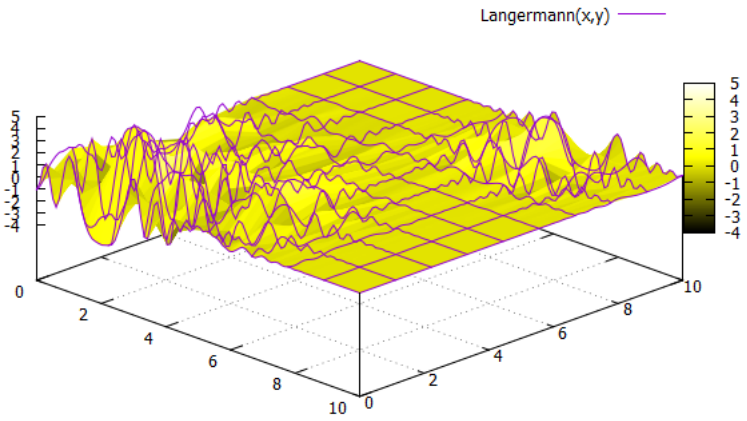

Figura B.1: Función de Langermann en 2 dimensiones

En general podemos escribir la ecuación B.2 en forma de algoritmo para su

implementaación como se muestra en el algoritmo 5.

Algoritmo 5: Pseudocódigo de la Función de Langermann en dos dimensiones

Entrada:
x_1: Valor en la coordenada x
x_2: Valor en la coordenada y
Salida:
f : Valor de la Función de Langermann en el punto (x, y)

```
1  c ← [ 1  2  5  2  3 ];
2  a ← [ 3  5 ]
       [ 5  2 ]
       [ 2  1 ]
       [ 1  4 ]
       [ 7  9 ];
3  g ← 0,0 ;
4  f ← 0,0 ;
5  i ← 1;
6  j ← 1;
7  mientras i ≤ 5 hacer
8  |   mientras j ≤ 2 hacer
9  |   |   g ← g + (x_j − a_ij)^2 ;
10 |   |   j ← j + 1;
11 |   f ← f + c_i exp(−(1/π) g) cos(πg);
12 |   i ← i + 1;
13 devolver f;
```

Esta función nos permite probar el caso en el que las soluciones de un problema estén distribuidas de manera totalmente aleatoria en el espacio de búsqueda.

B.2. Función de Griewangk

La función de Griewangk es una función multimodal, que a diferencia de la anterior función de Langermann, tiene sus mínimos globales repartidos de una manera uniforme por todo el espacio de búsqueda.

La función de Griewangk n-dimensional se define como sigue:

$$f(x_1, ..., x_n)\frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos(\frac{x_i}{\sqrt{i}}) + 1 \tag{B.3}$$

Esto con todas las x_i en un intervalo de $[-600, 600]$, tiene su un mínimo global en el el punto $x_i = 0, i = 1, ..., n$ donde la función vale 0.

La figura B.2 muestra esta función para una $n = 2$, y nos permite apreciar que aunque pareciera una función convexa en realidad al realizar un acercamiento de ella en la figura B.3 se observa que esta repleta de mínimos locales.

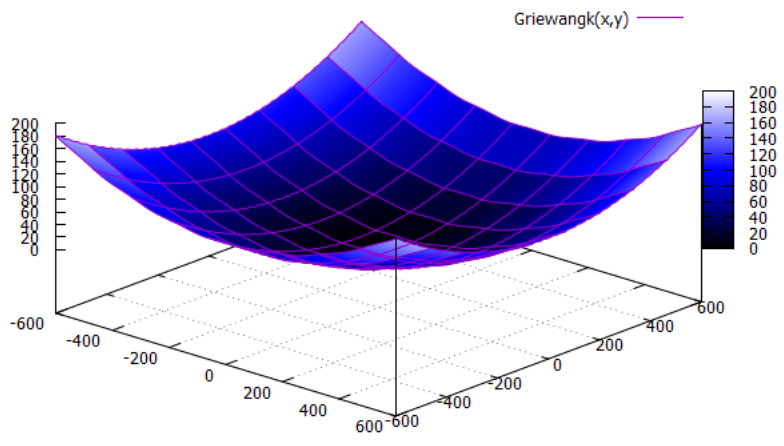

Figura B.2: Función de Grienwangk

El pseudocódigo que describe la función para n dimensiones esta dado por

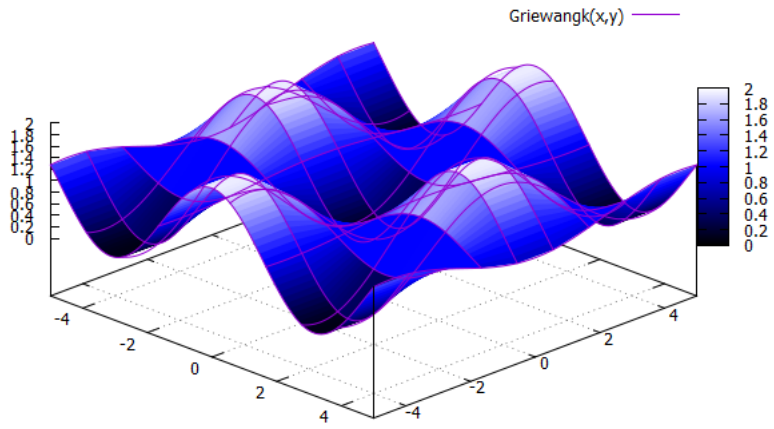

Figura B.3: Acercamiento a la Función de Grienwangk

el algoritmo 6 para su implementación.

Algoritmo 6: Pseudocódigo de la Función de Griewangk en n dimensiones

Entrada:
x: Arreglo de valores de entrada
Salida:
f : Valor de la Función de Griewangk en el punto dado por el vector de entrada x

```
1  g ← 0,0 ;
2  h ← 0,0 ;
3  f ← 0,0 ;
4  i ← 1;
5  mientras i ≤ longitud(x) hacer
6  |   g ← x_i^2 + g;
7  |   i ← i + 1;
8  i ← 1;
9  mientras i ≤ longitud(x) hacer
10 |   h ← cos(x_i / √i) * h;
11 |   i ← i + 1;
12 f ← 1/4000 * g − h + 1;
13 devolver f;
```

B.3. Función de Schwefel

La función de Shwefel es una función con un único optimo global, su dificultad en problemas de optimización radica en que los óptimos locales van mejorando conforme se alejan de este optimo global, lo que significa que la generación de la primera generación en los algoritmos evolutivos es un punto importante a tomar en este tipo de problemas.

La ecuación B.4 nos define la función de Schwefel.

$$f(x_1, ..., x_n) = 418{,}9828 * n - \sum_{i=1}^{n} [x_i \sin(\sqrt{|x_i|})] \tag{B.4}$$

La función definida en un intervalo $[-500, 500]$ en x_i tiene su optimo global en $x_i = 420{,}9687$ donde la función tiene un valor de 0.

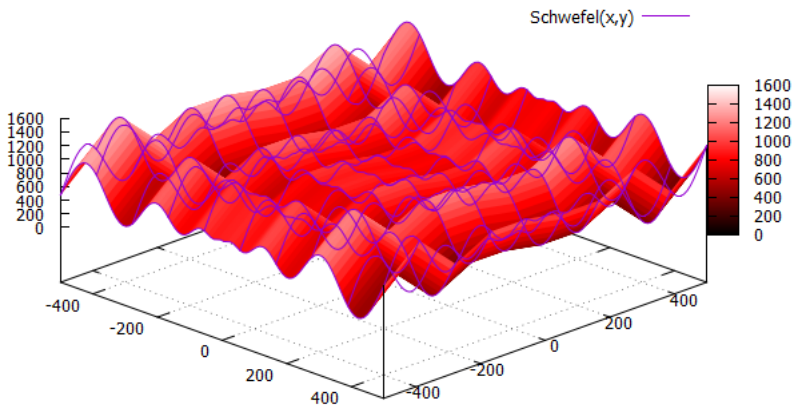

Figura B.4: Función de Schwefel para 2 dimensiones

El pseudocódigo para la implementación de esta función se muestra en el

algoritmo 7.

Algoritmo 7: Pseudocódigo de la Función de Schwefel en n dimensiones

Entrada:
x: Arreglo de valores de entrada
Salida:
f : Valor de la Función de Schwefel en el punto dado por el vector de entrada x

```
1 g ← 0,0 ;
2 f ← 0,0 ;
3 i ← 1;
4 mientras i ≤ longitud(x) hacer
5 |   g ← x_i * sin(√|x_i|) + g;
6 |   i ← i + 1;
7 f ← 418,9829 * n − g;
8 devolver f;
```

B.4. Función de Rosenbrock

La función de Rosenbrock, también conocida como función plátano, tiene un único optimo global localizado en un valle extenso, donde se pierde fácilmente entre otros valores, lo que nos permite explorar el caso donde es difícil vislumbrar la solución optima del problema.

La función en n dimensiones se define a continuación:

$$f(x_1, ..., x_n) = \sum_{i=1}^{n-} [(1 - x_i)^2 + 100 * (x_{i+1} - x_i^2)^2] \tag{B.5}$$

La figura nos B.5 muestra la función en dos dimensiones en un intervalo para $-2{,}048 \leq x_i \leq 2{,}048$ y su optimo en $x_i = 1$ con $f(x, y) = 0$.

El algoritmo 8 muestra el pseudocódigo de la función para su implementa-

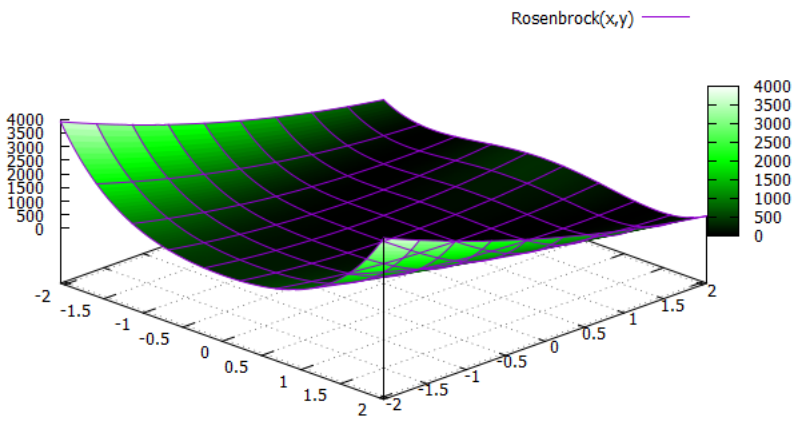

Figura B.5: Función de Rosenbrock en 2 dimensiones

ción.

Algoritmo 8: Pseudocódigo de la Función de Rosenbrock en n dimensiones

Entrada:
x: Arreglo de valores de entrada
Salida:
f : Valor de la Función de Rosenbrock en el punto dado por el vector de entrada x

```
1 f ← 0,0 ;
2 i ← 1;
3 mientras i ≤ longitud(x) − 1 hacer
4 |   f ← (1 − x_i)^2 + 100 ∗ (x_{i+1} − x_i^2)^2 + f;
5 |   i ← i + 1;
6 devolver f;
```

B.5. Función de Shubert

La función de Shubert es una función definida en el espacio tridimensional con una gran cantidad de mínimos locales y globales. Esta definida como:

$$f(x,y) = (\sum_{i=1}^{5} i\cos((i+1)x+i))(\sum_{i=1}^{5} i\cos((i+1)y+i)) \tag{B.6}$$

En la figura B.6 puede verse el comportamiento de la función con x y y en un intervalo de $[-10, 10]$, y sus mínimos globales tienen un valor de $f(x,y) = -186{,}7309$.

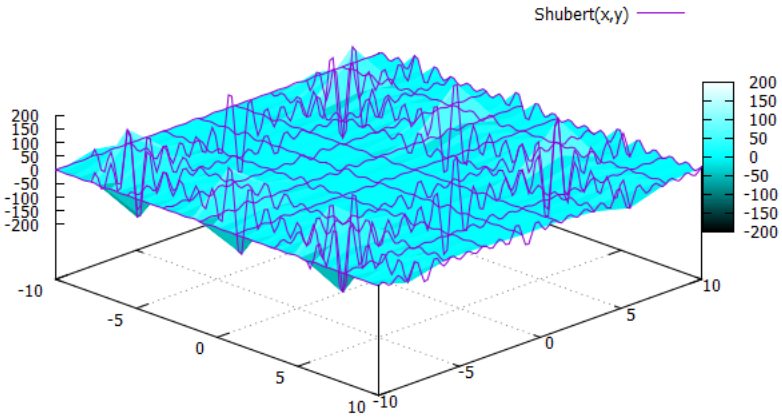

Figura B.6: Función de Shubert

Su implementación va de acuerdo a lo que se especifica en el algoritmo 9.

Algoritmo 9: Pseudocódigo de la Función de Shubert

Entrada:
x
y
Salida:
f : Valor de la Función de Shubert en el punto (x, y)

```
g ← 0,0 ;
h ← 0,0 ;
f ← 0,0 ;
i ← 1;
mientras i ≤ 5 hacer
    g ← i cos((i + 1)x + i) + g;
    i ← i + 1;
i ← 1;
mientras i ≤ 5 hacer
    h ← i cos((i + 1)y + i) + h;
    i ← i + 1;
f ← g ∗ h;
devolver f;
```

Apéndice C

Distribuciones generadas para la experimentación

Para los experimentos 2, 3 y 4 (Ver 6.2, 6.3 y 6.4 respectivamente) se utilizaron diferentes distribuciones de ocurrencias de fenómenos, con los códigos D.3 y D.4 se generaron los siguientes doce grupos de distribuciones.

Los grupos de distribuciones 1 al 3 fueron generadas con 100 ocurrencias del fenómeno con 5 posibles resultados, los grupos 4 al 5 tienen 1000 ocurrencias del fenómeno con 10 posibles resultados y los últimos 3 tienen 10000 ocurrencias con 25 resultados posibles.

Los grupos 1, 4 y 7 tienen 3 conjuntos de distribuciones, los grupos 2, 5 y 8 tienen 6 conjuntos y los grupos 3, 6 y 9 tienen 12.

La información de los diferentes grupos de distribuciones se resume en la tabla C.1 y estos se presentan en las tablas siguientes.

Tabla C.1: Información de los grupos de distribuciones generados

Grupo	Número de Conjuntos	Ocurrencias del fenomeno	Posibles resultados
1ro	3	100	5
2do	6	100	5
3ro	9	100	5
4to	3	1000	10
5to	6	1000	10
6to	9	1000	10
7mo	3	10000	25
8vo	6	10000	25
9no	9	10000	25

Tabla C.2: Primer grupo de distribuciones

Probabilidades	Distribuciones		
	1	2	3
24 %	15	22	27
10 %	13	8	14
31 %	38	33	25
25 %	17	27	24
10 %	17	10	10

Tabla C.3: Segundo grupo de distribuciones

Probabilidad	Distribuciones					
	1	2	3	4	5	6
20 %	17	21	27	21	21	25
29 %	12	9	9	13	13	10
3 %	31	37	31	31	29	33
32 %	29	23	22	26	23	19
16 %	11	10	11	9	14	13

Tabla C.4: Tercer grupo de distribuciones

Probabilidad	Distribuciones											
	1	2	3	4	5	6	7	8	9	10	11	12
16 %	22	22	18	30	21	26	25	26	29	26	20	25
28 %	12	9	13	11	12	6	14	6	14	14	14	8
20 %	33	28	32	28	24	29	32	36	27	28	25	35
19 %	26	28	25	18	37	29	24	24	20	22	33	24
17 %	7	13	12	13	6	10	5	8	10	10	8	8

Tabla C.5: Cuarto grupo de distribuciones

Probabilidad	Distribuciones		
	1	2	3
5 %	35	52	45
14 %	135	163	150
12 %	122	123	99
5 %	47	46	57
16 %	174	147	152
14 %	127	139	139
8 %	78	85	98
5 %	48	45	59
11 %	124	104	95
11 %	110	96	106

Tabla C.6: Quinto grupo de distribuciones

Probabilidad	Distribuciones					
	1	2	3	4	5	6
13 %	116	101	107	122	109	124
5 %	40	48	54	52	47	56
19 %	146	148	146	121	146	161
11 %	117	112	97	109	124	111
5 %	46	50	49	61	56	39
1 %	160	166	139	157	150	147
22 %	147	151	146	152	149	138
10 %	81	78	85	74	73	71
9 %	41	50	52	44	45	36
5 %	106	96	125	108	101	117

Tabla C.7: Sexto grupo de distribuciones

Probabilidad	Distribuciones											
	1	2	3	4	5	6	7	8	9	10	11	12
16 %	112	118	117	121	105	113	115	114	114	120	99	101
15 %	50	57	54	67	49	49	58	52	51	52	52	44
4 %	147	145	156	146	156	128	132	154	167	168	133	152
6 %	121	116	117	101	123	139	139	117	113	114	127	124
17 %	38	39	42	45	48	48	45	47	39	47	51	39
9 %	172	175	149	150	151	156	167	173	151	147	156	180
12 %	121	128	134	149	159	136	134	136	130	116	151	144
6 %	81	67	83	76	67	72	71	61	72	76	90	69
7 %	56	50	46	44	46	51	40	40	56	41	44	43
8 %	102	105	102	101	96	108	96	106	107	119	97	104

Tabla C.8: Séptimo grupo de distribuciones

Probabilidad	Distribuciones		
	1	2	3
4 %	407	426	415
1 %	178	194	194
5 %	533	537	518
4 %	434	486	453
2 %	176	151	157
6 %	651	643	618
5 %	556	594	540
3 %	295	275	305
2 %	153	191	173
4 %	411	417	396
1 %	107	145	142
1 %	68	71	70
4 %	359	361	358
2 %	242	248	248
1 %	90	65	57
6 %	615	609	668
6 %	557	553	642
6 %	616	646	647
6 %	609	593	601
6 %	591	564	614
2 %	203	224	185
3 %	337	265	302
6 %	614	579	592
6 %	600	607	563
6 %	598	556	542

Tabla C.9: Octavo grupo de distribuciones

Probabilidad	Distribuciones					
	1	2	3	4	5	6
8 %	394	407	427	455	388	447
2 %	171	173	160	170	165	169
1 %	526	536	513	534	508	510
5 %	417	417	447	433	412	451
5 %	160	170	164	155	179	145
4 %	625	610	648	618	617	611
1 %	530	526	527	484	560	524
3 %	269	306	293	309	293	285
1 %	178	165	189	165	204	152
8 %	408	388	400	419	386	390
8 %	136	132	127	132	125	120
7 %	68	76	75	71	68	72
5 %	373	351	362	325	350	338
3 %	247	252	241	241	255	235
8 %	74	77	52	54	65	66
1 %	647	658	647	599	608	636
1 %	581	611	600	621	594	586
3 %	610	602	669	629	662	711
5 %	607	606	553	638	634	583
4 %	662	616	626	634	617	617
1 %	217	221	201	208	215	191
4 %	302	296	292	292	314	323
8 %	604	609	607	610	608	582
3 %	632	588	620	620	622	623
1 %	562	607	560	584	551	633

Tabla C.10: Noveno grupo de distribuciones

Probabilidad	Distribuciones											
	1	2	3	4	5	6	7	8	9	10	11	12
6 %	418	405	410	407	431	420	421	430	430	385	407	406
6 %	189	166	168	184	172	162	204	180	169	179	184	178
8 %	532	534	521	544	539	497	569	497	536	523	514	498
3 %	445	420	436	444	437	439	474	450	403	433	428	439
5 %	168	150	192	167	183	161	179	186	178	173	156	166
3 %	604	636	626	599	601	577	610	599	662	571	623	586
3 %	519	526	502	526	549	541	506	532	556	568	511	521
3 %	294	280	320	290	297	306	315	277	294	269	285	274
1 %	181	177	166	154	175	199	173	183	162	161	192	167
7 %	397	405	409	390	419	394	400	402	400	390	386	423
2 %	160	127	128	134	119	127	126	117	125	121	118	103
3 %	84	62	61	66	64	68	56	64	78	65	71	70
2 %	346	371	334	339	361	343	333	438	357	406	367	353
3 %	228	281	247	250	278	234	242	220	257	242	257	227
6 %	62	77	61	59	52	78	56	60	74	74	73	70
2 %	678	619	652	679	616	673	646	636	633	640	656	681
1 %	559	575	622	566	606	597	620	572	601	626	609	622
3 %	627	631	581	648	637	652	672	682	623	640	654	628
6 %	556	621	597	619	586	617	571	583	576	580	584	629
5 %	613	612	610	601	596	591	610	614	627	656	579	633
6 %	199	196	221	198	220	209	206	225	198	216	216	216
4 %	286	305	281	296	311	300	273	283	281	293	303	284
6 %	609	621	643	628	597	620	567	573	604	593	583	604
1 %	654	606	640	632	579	608	610	586	586	636	619	630
5 %	592	597	572	580	575	587	561	611	590	560	629	592

Apéndice D

Código Fuente de Algoritmos

En este Apéndice se puede consultar el código fuente de los Algoritmos Evolutivos en sus diferentes prototipos, las funciones de optimización del experimento 1, la Prueba χ^2 de Pearson, la función generadora de distribuciones para los experimentos 2 al 5 y la función de lectura de archivos.

D.1. Funciones de Optimización

Código D.1: Funciones de Minimización

```
1  ;
       ..........................................................................................................
2  ;
3  ;
4  ;Funciones de para realizar la implementación y pruebas de los algoritmos
       evolutivos
5  ;
6  ;
7  ;
       ..........................................................................................................
8
9  ;NOTA: Todas las funciones se esribieron para el caso de 2 dimensiones
       solamente.
10
11 (defparameter *f* 0.0)          ;Salida de todas las funciones
12
13 ;
       ..........................................................................................................
14 ;
15 ;Función de Langermann
```

```
16 ;
17 ;
     ..........................................................................................

18 ;
19 ;Función multimodal con mínimos locales distribuidos de manera no pareja.
20 ; 0 <= x,y <= 10
21 ;Los valores para m, a, b y c fueron recomendados por Molga y Smutnicki
22 ;http://www.sfu.ca/~ssurjano/langer
23 ;
24 ;
     ..........................................................................................

25
26 (defun Langermann (x y)
27   (setf *f* 0.0)
28   (let
29     (
30       (m 5)
31       (a '(3 5 2 1 7))
32       (b '(5 2 1 4 9))
33       (c '(1 2 5 2 3))
34     )
35     (loop for i from 0 to (- m 1) do
36       (setf *f*
37         (+ *f*
38           (* (nth i c)
39             (exp (+ (* (/ -1 pi) (expt (- x (nth i a)) 2)) (* (/ -1 pi) (expt
                    (- y (nth i b)) 2))))
40             (cos (+ (* pi (expt (- x (nth i a)) 2)) (* pi (expt (- y (nth i b
                  )) 2))))
41           )
42         )
43       )
44     )
45   )
46   *f*
47 )
48
49 ;
     ..........................................................................................

50 ;
51 ;Función de Griewangk
52 ;
53 ;
     ..........................................................................................

54 ;
55 ;Función multimodal con mínimos locales distribuidos de manera regular,
       conforme se acerca al mínimo global la función pasa a ser una serie de
       saltos continuos a mínimos locales.
56 ; -600 <= x,y <= 600
57 ;http://www.sfu.ca/~ssurjano/griewank.html
58 ;http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/
       TestGO_files/Page1905.htm
59 ;
60 ;
     ..........................................................................................

61
62 (defun Griewangk (x y)
63   (setf *f*
64     (+
65       (* (/ 1 4000) (+ (expt x 2) (expt y 2)))
66       (* -1 (cos x) (cos (/ y (sqrt 2))))
67       1
68     )
```

```
69    )
70    *f*
71  )
72
73  ;
        ..........................................................................................................

74  ;
75  ;Función de Schwefel
76  ;
77  ;
        ..........................................................................................................

78  ;
79  ;Función multimodal con mínimos locales distribuidos de manera que el
        siguiente mínimo local más pequeño esta más lejos que el anterior al mí
        nimo global
80  ; -500 <= x,y <= 500
81  ;http://www.sfu.ca/~ssurjano/schwef.html
82  ;http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/
        TestGO_files/Page2530.htm
83  ;
84  ;
        ..........................................................................................................

85
86  (defun Schwefel (x y)
87    (setf *f*
88      (-
89        (* 418.9829 2)
90        (+
91          (* x (sin (sqrt (abs x))))
92          (* y (sin (sqrt (abs y))))
93        )
94      )
95    )
96    *f*
97  )
98
99  ;
        ..........................................................................................................

100 ;
101 ;Función de Rosenbrock
102 ;
103 ;
        ..........................................................................................................

104 ;
105 ;Función con el mínimo global en una zona casi homogénea lo que dificulta su
        búsqueda.
106 ; -2.048 <= x,y <= 2.048
107 ;http://www.sfu.ca/~ssurjano/rosen.html
108 ;http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/
        TestGO_files/Page2537.htm
109 ;
110 ;
        ..........................................................................................................

111
112 (defun Rosenbrock (x y)
113   (setf *f*
114     (+
115       (expt (- 1 x) 2)
116       (*
117         100
118         (expt (- y (expt x 2)) 2)
119       )
```

```
120     )
121   )
122   *f*
123 )
124
125 ;
        ..........................................................................................
126 ;
127 ;Función de Shubert
128 ;
129 ;
        ..........................................................................................
130 ;
131 ;Función con el mínimo global en una zona casi homogénea lo que dificulta su
        búsqueda.
132 ; -5.12 <= x,y <= 5.12
133 ; Min f(x,y) = -186.7309
134 ;http://www.sfu.ca/~ssurjano/shubert.html
135 ;http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/
        TestGO_files/Page1882.htm
136 ;
137 ;
        ..........................................................................................
138
139 (defun Shubert (x y)
140   (let
141     (
142       (m 5)
143       (f1 0.0)
144       (f2 0.0)
145     )
146     (loop for i from 1 to m do
147       (setf f1 (+ f1 (* i (cos (+ i (* x (+ i 1)))))))
148       (setf f2 (+ f2 (* i (cos (+ i (* y (+ i 1)))))))
149     )
150     (setf *f* (* f1 f2))
151   )
152   *f*
153 )
```

D.2. Prueba χ^2 de Pearson

Código D.2: Función de la Prueba χ^2 de Pearson

```
1 ;Código de la función de distribución de la prueba Chi Cuadrada de Pearson
      para n variables
2
3 (defun chi-square-pearson (Ob Th)
4   (apply #'+ (loop for a in Ob for b in Th collect (/ (expt (- a b) 2) b)))
5 )
```

D.3. Función generadora de distribuciones

Código D.3: Código de Generación de Distribuciones

```
1 ;;;;Código para crear las distribuciones de prueba
2
3 ;;; Variables globales
```

```
4
5  (defvar *file* (string "distribucion_"))
6  (defvar *disPro* nil)
7  (defvar *disProAcu* nil)
8
9  ;;; Funciones
10
11 ;;Función para crear la probabilidad de las distribuciones
12 ;; sizeD := Tamaño de las distribuciones
13
14 (defun make-prob (sizeD)
15   (setf *disPro*
16     (loop for i from 1 to sizeD collect
17       (1+ (random 100.0))
18     )
19   )
20   (setf *disPro*
21     (loop for i in *disPro* collect
22       (/ i (apply #'+ *disPro*))
23     )
24   )
25   (loop for i from 1 to sizeD do
26     (setf
27       *disProAcu*
28       (append
29         *disProAcu*
30         (list (apply #'+ (subseq *disPro* 0 i)))
31       )
32     )
33   )
34 )
35
36 ;;Función para crear las distribuciones
37 ;; eventos := Cantidad de ocurrencias ; n := Número de la distribución ;
       sizeD := Tamaño de las distribuciones
38
39
40 (defun make-distribution (eventos n sizeD id)
41   (let
42     (
43       (c 0.0)
44       (u (make-array sizeD :initial-element 0))
45       (r 0)
46       (file "")
47     )
48     (loop for i from 1 to eventos do
49       (setf c (random 1.0))
50       (loop for j from 0 to (1- sizeD) do
51         (setf r j)
52         (when (<= c (nth j *disProAcu*))
53           (setf j sizeD)
54         )
55       )
56       (setf (aref u r) (1+ (aref u r)))
57       (setf r 0)
58     )
59     (setf file (concatenate 'string id *file* (write-to-string n) ".data"))
60     (with-open-file
61       (stream file :direction :output
62               :if-exists :supersede
63               :if-does-not-exist :create
64       )
65       (format stream (concatenate 'string (write-to-string sizeD) "␣" (
             write-to-string eventos) "~%"))
66       (loop for j from 1 to sizeD do
67         (format stream (concatenate 'string (write-to-string (aref u (1- j)))
                 "~%"))
68       )
```

```
    )
  )
)

;;Función para crear grupos de distribuciones
;; sizeC := Cantidad de distribuciones ; sizeD := Tamaño de las
     distribuciones ; eventos := Número total de eventos en la distribución
(defun make-cluster (sizeC sizeD eventos id)
  (let
    (
      (file "")
      (path (write-to-string id))
    )
    (make-prob sizeD)
    (setf path (concatenate 'string path "/"))
    (ensure-directories-exist path)
    (setf file (concatenate 'string path *file* "Info.data"))
    (with-open-file
      (stream file :direction :output
              :if-exists :supersede
              :if-does-not-exist :create
      )
      (format stream (concatenate 'string "Distribuciones␣generadas:␣" (
           write-to-string sizeC) "␣~%Probabilidades␣de␣estas:␣" (
           write-to-string *disPro*) "~%"))
    )
    (loop for i from 1 to sizeC do
      (make-distribution eventos i sizeD path)
    )
  )
)
```

Código D.4: Experimento de Generación de Funciones

```
;;;;Experimento para la generación de distribuciones

;;Archivos importados

(load "distribuciones.lisp")

;;Creación de distribuciones

(let
  (
    (c 1)
  )
  (loop for i in '(5 10 25 30) do
    (loop for j in '(3 6 12) do
      (make-cluster j i 100 c)
      (setf c (1+ c))
    )
  )
)
```

D.4. Lectura de Archivos de Distribuciones

Código D.5: Función de Lectura de archivos con Distribuciones

```
;Código para lectura de las BD en texto plano con las distribuciones
     observadas de crimenes

(defun read-db (DB)
  (let
    (
```

```
6       (n 0)
7       (l 0)
8     )
9     (with-open-file (stream DB)
10      (setf n (read stream nil nil))
11      (setf l (read stream nil nil))
12      (list (list n l)  (loop for i from 1 to n collect (read stream nil nil)
            ))
13    )
14  )
15 )
```

D.5. Algoritmos Evolutivos

A continuación, se presenta el código de cada uno de los prototipos programados de los algoritmos evolutivos utilizados.

D.5.1. Optimización Por Enjambre de Partículas

Código D.6: Algoritmo OEP. Prototipo 1

```
1  ;;;;ALgoritmo PSO
2
3  ;-------------------------------------------------------------------
4  ;;;Archivos importados
5
6  (load "funciones.lisp")
7
8  ;-------------------------------------------------------------------
9
10 ;-------------------------------------------------------------------
11 ;;Variables Globales
12
13 (defparameter *p* (make-array 1 :element-type 'list
14                 :initial-element NIL
15                 :adjustable T
16                 :fill-pointer 0))  ; Población
17 (defparameter *g* 0)                  ; Mejor Particula
18 (defparameter *a* 1.0)                  ; Parámetro de escala
19 (defparameter *q1* 1.5)                 ; Valor para la distribución uniforme
       para la influencia propia
20 (defparameter *q2* 1.5)                 ; Valor para la distribución uniforme
       para la influencia grupal
21 (defparameter *path* (string "./"))          ; Directorio para Guartar
       Experimento
22 (defparameter *file* (string "."))           ; Archivo con información el
       experimento
23 (defparameter *mejores-individuos* (string ""))    ; Cadena con mejores
       individuos
24
25 ;-------------------------------------------------------------------
26
27 ;-------------------------------------------------------------------
28 ;;Funciones para crear un experimento de PSO y llevar un recuento de las
       variables
29
30 (defun crea-experimento (f n g li ls)
31   (setf *mejores-individuos* (string ""))
32   (setf *path* (concatenate 'string "./PSO/" (write-to-string (
         get-universal-time)) "/"))
```

```
33   (ensure-directories-exist *path*)
34   (ensure-directories-exist (concatenate 'string *path* "Generaciones/
         Posiciones/"))
35   (ensure-directories-exist (concatenate 'string *path* "Generaciones/
         Velocidades/"))
36   (ensure-directories-exist (concatenate 'string *path* "Generaciones/Memoria
         /"))
37   (setf *file* (concatenate 'string *path* "DatosExperimento.txt"))
38   (with-open-file
39     (stream *file* :direction :output
40             :if-exists :supersede
41             :if-does-not-exist :create
42     )
43     (format stream
44       (concatenate 'string "Optimización␣por␣Enjambre␣de␣Partículas~2%~3TTama
           ño␣de␣Población:␣" (write-to-string n) "~%~3TNúmero␣de␣
           Generaciones:␣" (write-to-string g) "~2%~3TFunción␣de␣optimización
           :␣" (remove #\> (subseq (write-to-string f) 11)) "~%~6TEspacio␣de␣
           búsqueda:␣[" (write-to-string li) "," (write-to-string ls) "]")
45     )
46   )
47 )
48
49 (defun guarda-generacion (g)
50   (setf *file* (concatenate 'string *path* "Generaciones/Posiciones/" "g_" (
         write-to-string g) ".pos.data"))
51   (with-open-file
52     (stream *file* :direction :output
53             :if-exists :supersede
54             :if-does-not-exist :create
55     )
56     (loop for i from 0 to (1- (array-total-size *p*)) do
57       (format stream
58         (concatenate 'string (write-to-string (nth 0 (nth 0 (aref *p* i)))) "
               ␣" (write-to-string (nth 1 (nth 0 (aref *p* i)))) "~%")
59       )
60     )
61   )
62   (setf *file* (concatenate 'string *path* "Generaciones/Velocidades/" "g_" (
         write-to-string g) ".vel.data"))
63   (with-open-file
64     (stream *file* :direction :output
65             :if-exists :supersede
66             :if-does-not-exist :create
67     )
68     (loop for i from 0 to (1- (array-total-size *p*)) do
69       (format stream
70         (concatenate 'string (write-to-string (nth 0 (nth 2 (aref *p* i)))) "
               ␣" (write-to-string (nth 1 (nth 2 (aref *p* i)))) "~%")
71       )
72     )
73   )
74   (setf *file* (concatenate 'string *path* "Generaciones/Memoria/" "g_" (
         write-to-string g) ".mem.data"))
75   (with-open-file
76     (stream *file* :direction :output
77             :if-exists :supersede
78             :if-does-not-exist :create
79     )
80     (loop for i from 0 to (1- (array-total-size *p*)) do
81       (format stream
82         (concatenate 'string (write-to-string (nth 0 (nth 1 (aref *p* i)))) "
               ␣" (write-to-string (nth 1 (nth 1 (aref *p* i)))) "~%")
83       )
84     )
85   )
86 )
87
```

```
88 (defun guarda-mejores-individuos ()
89   (setf *mejores-individuos* (concatenate 'string *mejores-individuos* (
         write-to-string (nth 0 (nth 1 (aref *p* *g*)))) "␣" (write-to-string (
         nth 1 (nth 1 (aref *p* *g*)))) "~%"))
90 )
91
92 (defun guarda-resultados ()
93   (setf *file* (concatenate 'string *path* "mejoresIndividuos.data"))
94   (with-open-file
95     (stream *file* :direction :output
96             :if-exists :supersede
97             :if-does-not-exist :create
98     )
99     (format stream *mejores-individuos*)
100  )
101  (setf *file* (concatenate 'string *path* "mejorIndividuo.data"))
102  (with-open-file
103    (stream *file* :direction :output
104            :if-exists :supersede
105            :if-does-not-exist :create
106    )
107    (format stream
108        (concatenate 'string (write-to-string (nth 0 (nth 1 (aref *p* *g*))))
               "␣" (write-to-string (nth 1 (nth 1 (aref *p* *g*)))) "~%")
109    )
110  )
111 )
112
113 ;---------------------------------------------------------------------
114
115 ;---------------------------------------------------------------------
116 ;;Función de generación de enjambre
117 ;; n := Tamaño de Enjambre ; li := Limite inferior ; ls := Limite Superior
118
119 (defun inicializaEnjambre (n li ls)
120   (let
121     (
122       (p NIL)
123       (v NIL)
124     )
125     (adjust-array *p* (list n))
126     (loop for i from 0 to (1- n) do
127       (setf p (list (+ (random (- ls li)) li) (+ (random (- ls li)) li)))   ;
             Posición inicial
128       (setf v (list (+ (random (/ (- ls li) 5.0)) (/ li 5.0)) (+ (random (/
             (- ls li) 5.0)) (/ li 5.0)))) ;Velocidad Inicial
129       ;;La estructura de una particula sera una lista de tres listas, esta
             dada como: (Posición Memoria Velocidad)
130       (setf (aref *p* i) (list p p v))
131     )
132   )
133 )
134
135 ;---------------------------------------------------------------------
136
137 ;---------------------------------------------------------------------
138 ;;Función para encontrar la mejor particula
139 ;; f := Función objetivo :: #'f
140
141 (defun mejorParticula (f)
142   (setf *g* 0)
143   (loop for i from 1 to (1- (array-total-size *p*)) do
144     (when (< (funcall f (nth 0 (nth 1 (aref *p* i))) (nth 1 (nth 1 (aref *p*
           i)))) (funcall f (nth 0 (nth 1 (aref *p* *g*))) (nth 1 (nth 1 (aref
           *p* *g*)))))
145       (setf *g* i)
146     )
147   )
```

```
148
149 )
150 ;---------------------------------------------------------------------
151
152 ;---------------------------------------------------------------------
153 ;;Función para actualizar las velocidades de las particulas
154
155 (defun nuevaVelocidad ()
156   (let
157     (
158       (v1 0.0)
159       (v2 0.0)
160     )
161     (loop for i from 0 to (1- (array-total-size *p*)) do
162       (setf v1
163         (+
164           (* *a* (nth 0 (nth 2 (aref *p* i))))
165           (* (random *q1*) (- (nth 0 (nth 1 (aref *p* i))) (nth 0 (nth 0 (
                aref *p* i)))))
166           (* (random *q2*) (- (nth 0 (nth 0 (aref *p* *g*))) (nth 0 (nth 0 (
                aref *p* i)))))
167         )
168       )
169       (setf v2
170         (+
171           (* *a* (nth 1 (nth 2 (aref *p* i))))
172           (* (random *q1*) (- (nth 1 (nth 1 (aref *p* i))) (nth 1 (nth 0 (
                aref *p* i)))))
173           (* (random *q2*) (- (nth 1 (nth 0 (aref *p* *g*))) (nth 1 (nth 0 (
                aref *p* i)))))
174         )
175       )
176       (setf (aref *p* i) (list (nth 0 (aref *p* i)) (nth 1 (aref *p* i)) (
            list v1 v2)))
177     )
178   )
179 )
180
181 ;---------------------------------------------------------------------
182 ;; Función para actulizar la posición de las particulas
183 ;; f := Función objetivo :: #'f ; li := Limite inferior ; ls := Limite
      superior
184 (defun nuevaPosicion (f li ls)
185   (let
186     (
187       (p NIL)
188       (b NIL)
189       (v NIL)
190       (p1 0.0)
191       (p2 0.0)
192     )
193     (loop for i from 0 to (1- (array-total-size *p*)) do
194       (setf p (nth 0 (aref *p* i)))
195       (setf b (nth 1 (aref *p* i)))
196       (setf v (nth 2 (aref *p* i)))
197       (setf p1 (+ (nth 0 p) (nth 0 v)))
198       (if (< p1 li)
199         (setf p1 (- ls (mod (- ls p1) (- ls li))))
200       )
201       (if (> p1 ls)
202         (setf p1 (mod p1 ls))
203       )
204       (setf p2 (+ (nth 1 p) (nth 1 v)))
205       (if (< p2 li)
206         (setf p2 (- ls (mod (- ls p2) (- ls li))))
207       )
208       (if (> p2 ls)
209         (setf p2 (mod p2 ls))
```

```
      )
      (setf p (list p1 p2))
      (if (< (funcall f p1 p2) (funcall f (nth 0 b) (nth 1 b)))
        (setf b (list p1 p2))
      )
      (setf (aref *p* i) (list p b v))
    )
  )
)

;---------------------------------------------------------------------
;;Algoritmo de Optimización por Enjambre de Particulas
;; n := Tamaño del enjambre ; g := Número de iteraciones ; f := Función
    objetivo  :: #'f ; li := Limite inferior ; ls := Limite superior

(defun PSO (f n g li ls)
  (crea-experimento f g n li ls)
  (setf *a* 1.0)
  (inicializaEnjambre n li ls)
  (mejorParticula f)
  (guarda-generacion 0)
  (guarda-mejores-individuos)
  (loop for i from 1 to g do
    (nuevaVelocidad)
    (nuevaPosicion f li ls)
    (mejorParticula f)
    (guarda-generacion i)
    (guarda-mejores-individuos)
    (setf *a* (- *a* (/ 1.0 g)))
  )
  (guarda-resultados)
)

;---------------------------------------------------------------------
```

Código D.7: Algoritmo OEP. Prototipo 2

```
;;;;ALgoritmo PSO

;;---------------------------------------------------------------------
;;Archivos importados

(load "lecturaBD.lisp")
(load "PCCP.lisp")

;;---------------------------------------------------------------------

;;---------------------------------------------------------------------
;;Variables Globales

(defparameter *db* nil)                ;Información de la Base de Datos
(defparameter *p* (make-array 1 :element-type 'list
                :initial-element NIL
                :adjustable T
                :fill-pointer 0))  ; Población
(defparameter *g* 0)                   ; Mejor Particula
(defparameter *-g* 0)                  ; Peor Particula
(defparameter *a* 1.0)                   ; Parámetro de escala
(defparameter *q1* 1.5)                ; Valor para la distribución uniforme
    para la influencia propia
(defparameter *q2* 1.5)                ; Valor para la distribución uniforme
    para la influencia grupal
(defparameter *path* (string "./"))          ; Directorio para Guartar
    Experimento
(defparameter *file* (string "."))           ; Archivo con información el
    experimento
(defparameter *mejores-individuos* (string ""))    ; Cadena con mejores
    individuos
```

```
27 (defparameter *mejores-aptitudes* (string ""))     ; Cadena con mejores
       aptitudes
28 (defparameter *peores-aptitudes* (string ""))    ; Cadena con peores
       aptitudes
29 (defparameter *promedio-aptitudes* (string ""))    ; Cadena con promedio de
       aptitudes
30
31 ;;-------------------------------------------------------------------
32
33 ;;-------------------------------------------------------------------
34 ;;Funciones para crear un experimento de PSO y llevar un recuento de las
       variables
35
36 (defun crea-experimento (n g)
37   (setf *mejores-individuos* (string ""))
38   (setf *mejores-aptitudes* (string ""))
39   (setf *peores-aptitudes* (string ""))
40   (setf *promedio-aptitudes* (string ""))
41   (setf *path* (concatenate 'string "./PSO/" (write-to-string (
         get-universal-time)) "/"))
42   (ensure-directories-exist *path*)
43   (ensure-directories-exist (concatenate 'string *path* "Generaciones/
         Posiciones/"))
44   (ensure-directories-exist (concatenate 'string *path* "Generaciones/
         Velocidades/"))
45   (ensure-directories-exist (concatenate 'string *path* "Generaciones/Memoria
         /"))
46   (setf *file* (concatenate 'string *path* "DatosExperimento.txt"))
47   (with-open-file
48     (stream *file* :direction :output
49             :if-exists :supersede
50             :if-does-not-exist :create
51     )
52     (format stream
53       (concatenate 'string "Optimización␣por␣Enjambre␣de␣Partículas~2%~3TTama
           ño␣de␣Población:␣" (write-to-string n) "~%~3TNúmero␣de␣
           Generaciones:␣" (write-to-string g) "~2%~3TFunción␣de␣optimización
           :␣Prueba␣Chi␣Cuadrada␣de␣Pearson␣~%~6TEspacio␣de␣búsqueda:␣
           (0.0,1.0)")
54     )
55   )
56 )
57
58 (defun guarda-generacion (g)
59   (setf *file* (concatenate 'string *path* "Generaciones/Posiciones/" "g_" (
         write-to-string g) ".pos.data"))
60   (with-open-file
61     (stream *file* :direction :output
62             :if-exists :supersede
63             :if-does-not-exist :create
64     )
65     (loop for i from 0 to (1- (array-total-size *p*)) do
66       (format stream
67         (concatenate 'string (remove #\( (remove #\) (write-to-string (nth 0
               (aref *p* i))))) "~%")
68       )
69     )
70   )
71   (setf *file* (concatenate 'string *path* "Generaciones/Velocidades/" "g_" (
         write-to-string g) ".vel.data"))
72   (with-open-file
73     (stream *file* :direction :output
74             :if-exists :supersede
75             :if-does-not-exist :create
76     )
77     (loop for i from 0 to (1- (array-total-size *p*)) do
78       (format stream
79         (concatenate 'string (remove #\( (remove #\) (write-to-string (nth 2
```

```
                (aref *p* i))))) "~%")
80        )
81      )
82    )
83    (setf *file* (concatenate 'string *path* "Generaciones/Memoria/" "g_" (
          write-to-string g) ".mem.data"))
84    (with-open-file
85      (stream *file* :direction :output
86              :if-exists :supersede
87              :if-does-not-exist :create
88      )
89      (loop for i from 0 to (1- (array-total-size *p*)) do
90        (format stream
91          (concatenate 'string (remove #\( (remove #\) (write-to-string (nth 1
                (aref *p* i))))) "~%")
92        )
93      )
94    )
95  )
96
97  (defun guarda-mejores-individuos ()
98    (setf *mejores-individuos* (concatenate 'string *mejores-individuos* (
          remove #\( (remove #\) (write-to-string (nth 1 (aref *p* *g*))))) "~%"
          ))
99  )
100
101 (defun guarda-mejores-aptitudes ()
102   (setf *mejores-aptitudes* (concatenate 'string *mejores-aptitudes* (
          write-to-string (chi-square-pearson (nth 1 *db*) (loop for j in (nth 1
           (aref *p* *g*)) collect (* j (apply #'+ (nth 1 *db*)))))) "~%"))
103 )
104
105 (defun guarda-peores-aptitudes ()
106   (setf *peores-aptitudes* (concatenate 'string *peores-aptitudes* (
          write-to-string (chi-square-pearson (nth 1 *db*) (loop for j in (nth 1
           (aref *p* *-g*)) collect (* j (apply #'+ (nth 1 *db*)))))) "~%"))
107 )
108
109 (defun guarda-promedio-aptitudes (n)
110   (setf *promedio-aptitudes*
111     (concatenate 'string *promedio-aptitudes*
112       (write-to-string
113         (/
114           (apply #'+
115             (loop for i from 0 to (1- n) collect
116               (chi-square-pearson (nth 1 *db*) (loop for j in (nth 1 (aref *p
                      * i)) collect (* j (apply #'+ (nth 1 *db*)))))
117             )
118           )
119           n
120         )
121       )
122       "~%"
123     )
124   )
125 )
126
127 (defun guarda-resultados ()
128   (setf *file* (concatenate 'string *path* "mejoresIndividuos.data"))
129   (with-open-file
130     (stream *file* :direction :output
131             :if-exists :supersede
132             :if-does-not-exist :create
133     )
134     (format stream *mejores-individuos*)
135   )
136   (setf *file* (concatenate 'string *path* "mejorIndividuo.data"))
137   (with-open-file
```

```
138     (stream *file* :direction :output
139              :if-exists :supersede
140              :if-does-not-exist :create
141     )
142     (format stream
143         (concatenate 'string (remove #\( (remove #\) (write-to-string (nth 1
                (aref *p* *g*))))) "~%")
144     )
145   )
146   (setf *file* (concatenate 'string *path* "mejoresAptitudes.data"))
147   (with-open-file
148     (stream *file* :direction :output
149              :if-exists :supersede
150              :if-does-not-exist :create
151     )
152     (format stream *mejores-aptitudes*)
153   )
154   (setf *file* (concatenate 'string *path* "peoresAptitudes.data"))
155   (with-open-file
156     (stream *file* :direction :output
157              :if-exists :supersede
158              :if-does-not-exist :create
159     )
160     (format stream *peores-aptitudes*)
161   )
162   (setf *file* (concatenate 'string *path* "promedioAptitudes.data"))
163   (with-open-file
164     (stream *file* :direction :output
165              :if-exists :supersede
166              :if-does-not-exist :create
167     )
168     (format stream *promedio-aptitudes*)
169   )
170 )
171
172 ;;-------------------------------------------------------------------------
173
174 ;;-------------------------------------------------------------------------
175 ;;Función de generación de enjambre
176 ;; n := Tamaño de Enjambre ; li := Limite inferior ; ls := Limite Superior
177
178 (defun inicializaEnjambre (n f)
179   (let
180     (
181       (p NIL)
182       (v NIL)
183     )
184     (adjust-array *p* (list n))
185     (setf *db* (read-db f))
186     (loop for i from 0 to (1- n) do
187       (setf p (loop for j from 1 to (nth 0 (nth 0 *db*)) collect (random 1.0)
              ))
188       (setf p (loop for j in p collect (/ j (apply #'+ p))))
189       (setf v (loop for j from 1 to (nth 0 (nth 0 *db*)) collect (- 0.5 (
              random 1.0))))
190       ;;La estructura de una particula sera una lista de tres listas, esta
              dada como: (Posición Memoria Velocidad)
191       (setf (aref *p* i) (list p p v))
192     )
193   )
194 )
195
196 ;;-------------------------------------------------------------------------
197
198 ;;-------------------------------------------------------------------------
199 ;;Función para encontrar la mejor particula
200
201 (defun mejorParticula nil
```

```
202   (setf *g* 0)
203   (loop for i from 1 to (1- (array-total-size *p*)) do
204     (when
205       (<
206         (chi-square-pearson (nth 1 *db*) (loop for j in (nth 1 (aref *p* i))
                collect (* j (apply #'+ (nth 1 *db*)))))
207         (chi-square-pearson (nth 1 *db*) (loop for j in (nth 1 (aref *p* *g*)
                ) collect (* j (apply #'+ (nth 1 *db*)))))
208       )
209       (setf *g* i)
210     )
211   )
212 )
213
214 ;;---------------------------------------------------------------------
215
216 ;;---------------------------------------------------------------------
217 ;;Función para encontrar la peor particula
218
219 (defun peorParticula nil
220   (setf *-g* 0)
221   (loop for i from 1 to (1- (array-total-size *p*)) do
222     (when
223       (>
224         (chi-square-pearson (nth 1 *db*) (loop for j in (nth 1 (aref *p* i))
                collect (* j (apply #'+ (nth 1 *db*)))))
225         (chi-square-pearson (nth 1 *db*) (loop for j in (nth 1 (aref *p* *-g
                *)) collect (* j (apply #'+ (nth 1 *db*)))))
226       )
227       (setf *-g* i)
228     )
229   )
230 )
231
232 ;;---------------------------------------------------------------------
233
234 ;;---------------------------------------------------------------------
235 ;;Función para actualizar las velocidades de las particulas
236
237 (defun nuevaVelocidad nil
238   (loop for i from 0 to (1- (array-total-size *p*)) do
239     (setf (aref *p* i)
240       (list
241         (nth 0 (aref *p* i))
242         (nth 1 (aref *p* i))
243         (loop for j in (nth 2 (aref *p* i))
244             for k in (nth 1 (aref *p* i))
245             for l in (nth 0 (aref *p* i))
246             for m in (nth 0 (aref *p* *g*)) collect
247           (+
248             (* *a* j)
249             (* (random *q1*) (- k l))
250             (* (random *q2*) (- m l))
251           )
252         )
253       )
254     )
255   )
256 )
257
258 ;;---------------------------------------------------------------------
259 ;; Función para actulizar la posición de las particulas
260 ;; f := Función objetivo :: #'f ; li := Limite inferior ; ls := Limite
      superior
261
262 (defun nuevaPosicion nil
263   (let
264     (
```

```
265       (p nil)
266       (b NIL)
267       (v NIL)
268     )
269     (loop for i from 0 to (1- (array-total-size *p*)) do
270       (setf p
271         (loop for j in (nth 0 (aref *p* i)) for k in (nth 2 (aref *p* i))
                collect
272           (mod (+ j k) 1.0)
273         )
274       )
275       (if (> (apply #'+ p) 1.0)
276         (setf p (loop for j in p collect (/ j (apply #'+ p))))
277       )
278       (setf b (nth 1 (aref *p* i)))
279       (setf v (nth 2 (aref *p* i)))
280       (if
281         (<
282           (chi-square-pearson (nth 1 *db*) (loop for j in p collect (* j (
                apply #'+ (nth 1 *db*)))))
283           (chi-square-pearson (nth 1 *db*) (loop for j in b collect (* j (
                apply #'+ (nth 1 *db*)))))
284         )
285         (setf b p)
286       )
287       (setf (aref *p* i) (list p b v))
288     )
289   )
290 )
291 ;;-------------------------------------------------------------------
292 ;;Algoritmo de Optimización por Enjambre de Particulas
293 ;; n := Tamaño del enjambre ; g := Número de iteraciones; f:= Archivo de la
      Base de Datos
294
295 (defun PSO (n g f)
296   (crea-experimento g n)
297   (setf *a* 1.0)
298   (inicializaEnjambre n f)
299   (mejorParticula)
300   (peorParticula)
301   (guarda-generacion 0)
302   (guarda-mejores-individuos)
303   (guarda-mejores-aptitudes)
304   (guarda-peores-aptitudes)
305   (guarda-promedio-aptitudes n)
306   (loop for i from 1 to g do
307     (nuevaVelocidad)
308     (nuevaPosicion)
309     (mejorParticula)
310     (peorParticula)
311     (guarda-generacion i)
312     (guarda-mejores-individuos)
313     (guarda-mejores-aptitudes)
314     (guarda-peores-aptitudes)
315     (guarda-promedio-aptitudes n)
316     (setf *a* (- *a* (/ 0.5 g)))
317   )
318   (guarda-resultados)
319 )
320
321 ;;-------------------------------------------------------------------
```

Código D.8: Algoritmo OEP. Prototipo 3

```
1 ;;;;ALgoritmo PSO
2
3 ;-------------------------------------------------------------------
4 ;;Archivos importados
```

```
5
6  (load "lecturaBD.lisp")
7  (load "PCCP.lisp")
8
9  ;;---------------------------------------------------------------------
10
11 ;;---------------------------------------------------------------------
12 ;;Variables Globales
13
14 (defparameter *o* (make-array 1 :element-type 'list
15                 :initial-element NIL
16                 :adjustable T
17                 :fill-pointer 0))  ; Distribuciones Observadas
18 (defparameter *p* (make-array 1 :element-type 'list
19                 :initial-element NIL
20                 :adjustable T
21                 :fill-pointer 0))  ; Población
22 (defparameter *g* 0)                  ; Mejor Particula
23 (defparameter *-g* 0)                 ; Peor Particula
24 (defparameter *a* 1.0)                  ; Parámetro de escala
25 (defparameter *q1* 1.5)                 ; Valor para la distribución uniforme
       para la influencia propia
26 (defparameter *q2* 1.5)                 ; Valor para la distribución uniforme
       para la influencia grupal
27 (defparameter *path* (string "./"))          ; Directorio para Guartar
       Experimento
28 (defparameter *file* (string "."))           ; Archivo con información el
       experimento
29 (defparameter *mejores-individuos* (string ""))    ; Cadena con mejores
       individuos
30 (defparameter *mejores-aptitudes* (string ""))     ; Cadena con mejores
       aptitudes
31 (defparameter *peores-aptitudes* (string ""))    ; Cadena con peores
       aptitudes
32 (defparameter *promedio-aptitudes* (string ""))    ; Cadena con promedio de
       aptitudes
33
34
35 ;;---------------------------------------------------------------------
36
37 ;;---------------------------------------------------------------------
38 ;;Funciones para crear un experimento de PSO y llevar un recuento de las
       variables
39
40 (defun crea-experimento (n g)
41   (setf *mejores-individuos* (string ""))
42   (setf *mejores-aptitudes* (string ""))
43   (setf *peores-aptitudes* (string ""))
44   (setf *promedio-aptitudes* (string ""))
45   (setf *path* (concatenate 'string "./PSO/" (write-to-string (
         get-universal-time)) "/"))
46   (ensure-directories-exist *path*)
47   (ensure-directories-exist (concatenate 'string *path* "Generaciones/
         Posiciones/"))
48   (ensure-directories-exist (concatenate 'string *path* "Generaciones/
         Velocidades/"))
49   (ensure-directories-exist (concatenate 'string *path* "Generaciones/Memoria
         /"))
50   (setf *file* (concatenate 'string *path* "DatosExperimento.txt"))
51   (with-open-file
52     (stream *file* :direction :output
53           :if-exists :supersede
54           :if-does-not-exist :create
55     )
56     (format stream
57       (concatenate 'string "Optimización␣por␣Enjambre␣de␣Partículas~2%~3TTama
           ño␣de␣Población:␣" (write-to-string n) "~%~3TNúmero␣de␣
           Generaciones:␣" (write-to-string g) "~2%~3TFunción␣de␣optimización
```

```
            :␣Prueba␣Chi␣Cuadrada␣de␣Pearson␣~%~6TEspacio␣de␣búsqueda:␣
            [0.0,1.0]")
58      )
59    )
60  )
61
62  (defun guarda-generacion (g)
63    (setf *file* (concatenate 'string *path* "Generaciones/Posiciones/" "g_" (
          write-to-string g) ".pos.data"))
64    (with-open-file
65      (stream *file* :direction :output
66              :if-exists :supersede
67              :if-does-not-exist :create
68      )
69      (loop for i from 0 to (1- (array-total-size *p*)) do
70        (format stream
71          (concatenate 'string (remove #\( (remove #\) (write-to-string (nth 0
                (aref *p* i))))) "~%")
72        )
73      )
74    )
75    (setf *file* (concatenate 'string *path* "Generaciones/Velocidades/" "g_" (
          write-to-string g) ".vel.data"))
76    (with-open-file
77      (stream *file* :direction :output
78              :if-exists :supersede
79              :if-does-not-exist :create
80      )
81      (loop for i from 0 to (1- (array-total-size *p*)) do
82        (format stream
83          (concatenate 'string (remove #\( (remove #\) (write-to-string (nth 2
                (aref *p* i))))) "~%")
84        )
85      )
86    )
87    (setf *file* (concatenate 'string *path* "Generaciones/Memoria/" "g_" (
          write-to-string g) ".mem.data"))
88    (with-open-file
89      (stream *file* :direction :output
90              :if-exists :supersede
91              :if-does-not-exist :create
92      )
93      (loop for i from 0 to (1- (array-total-size *p*)) do
94        (format stream
95          (concatenate 'string (remove #\( (remove #\) (write-to-string (nth 1
                (aref *p* i))))) "~%")
96        )
97      )
98    )
99  )
100
101 (defun guarda-mejores-individuos nil
102   (setf *mejores-individuos* (concatenate 'string *mejores-individuos* (
          remove #\( (remove #\) (write-to-string (nth 1 (aref *p* *g*))))) "~%"
          ))
103 )
104
105 (defun guarda-mejores-aptitudes ()
106   (setf *mejores-aptitudes* (concatenate 'string *mejores-aptitudes* (
          write-to-string (funcionObjetivo (nth 1 (aref *p* *g*))))) "~%"))
107 )
108
109 (defun guarda-peores-aptitudes ()
110   (setf *peores-aptitudes* (concatenate 'string *peores-aptitudes* (
          write-to-string (funcionObjetivo (nth 1 (aref *p* *-g*))))) "~%"))
111 )
112
113 (defun guarda-promedio-aptitudes (n)
```

```
114   (setf *promedio-aptitudes*
115     (concatenate 'string *promedio-aptitudes*
116       (write-to-string
117         (/
118           (apply #'+
119             (loop for i from 0 to (1- n) collect
120               (funcionObjetivo (nth 1 (aref *p* i)))
121             )
122           )
123           n
124         )
125       )
126       "~%"
127     )
128   )
129 )
130
131 (defun guarda-resultados nil
132   (setf *file* (concatenate 'string *path* "mejoresIndividuos.data"))
133   (with-open-file
134     (stream *file* :direction :output
135              :if-exists :supersede
136              :if-does-not-exist :create
137     )
138     (format stream *mejores-individuos*)
139   )
140   (setf *file* (concatenate 'string *path* "mejorIndividuo.data"))
141   (with-open-file
142     (stream *file* :direction :output
143              :if-exists :supersede
144              :if-does-not-exist :create
145     )
146     (format stream
147         (concatenate 'string (remove #\( (remove #\) (write-to-string (nth 1
                (aref *p* *g*))))) "~%")
148     )
149   )
150   (setf *file* (concatenate 'string *path* "mejoresAptitudes.data"))
151   (with-open-file
152     (stream *file* :direction :output
153              :if-exists :supersede
154              :if-does-not-exist :create
155     )
156     (format stream *mejores-aptitudes*)
157   )
158   (setf *file* (concatenate 'string *path* "peoresAptitudes.data"))
159   (with-open-file
160     (stream *file* :direction :output
161              :if-exists :supersede
162              :if-does-not-exist :create
163     )
164     (format stream *peores-aptitudes*)
165   )
166   (setf *file* (concatenate 'string *path* "promedioAptitudes.data"))
167   (with-open-file
168     (stream *file* :direction :output
169              :if-exists :supersede
170              :if-does-not-exist :create
171     )
172     (format stream *promedio-aptitudes*)
173   )
174 )
175
176 ;;-------------------------------------------------------------------
177 ;;Función de lectura de las distribuciones observadas
178 ;; f := Lista de Archivos con distribuciones
179 (defun leerDistribuciones (f)
180   (adjust-array *o* (list (length f)))
```

```
181   (loop for i in f for j from 1 to (length f) do
182     (setf (aref *o* (1- j)) (read-db i))
183   )
184 )
185
186 ;;---------------------------------------------------------------------
187
188 ;;---------------------------------------------------------------------
189 ;;Funcion de optimización
190 ;; Distribución Esperada
191 (defun funcionObjetivo (E)
192   (/
193     (apply #'+
194       (loop for i from 0 to (1- (array-total-size *o*)) collect
195         (chi-square-pearson
196           (nth 1 (aref *o* i))
197           (loop for j in E collect (* j (apply #'+ (nth 1 (aref *o* i)))))
198         )
199       )
200     )
201     (length E)
202   )
203 )
204
205 ;;---------------------------------------------------------------------
206
207 ;;---------------------------------------------------------------------
208 ;;Función de generación de enjambre
209 ;; n := Tamaño de Enjambre
210
211 (defun inicializaEnjambre (n f)
212   (let
213     (
214       (p NIL)
215       (v NIL)
216     )
217     (leerDistribuciones f)
218     (adjust-array *p* (list n))
219     (loop for i from 0 to (1- n) do
220       (setf p (loop for j from 1 to (nth 0 (nth 0 (aref *o* 0))) collect (
              random 1.0)))
221       (setf p (loop for j in p collect (/ j (apply #'+ p))))
222       (setf v (loop for j from 1 to (nth 0 (nth 0 (aref *o* 0))) collect (-
              0.5 (random 1.0))))
223       ;;La estructura de una particula sera una lista de tres listas, esta
              dada como: (Posición Memoria Velocidad)
224       (setf (aref *p* i) (list p p v))
225     )
226   )
227 )
228
229 ;;---------------------------------------------------------------------
230
231 ;;---------------------------------------------------------------------
232 ;;Función para encontrar la mejor particula
233 ;; f := Función objetivo :: #'f
234
235 (defun mejorParticula nil
236   (setf *g* 0)
237   (loop for i from 1 to (1- (array-total-size *p*)) do
238     (when
239       (and
240         (compruebaFactor1 (nth 1 (aref *p* *g*)) (nth 1 (aref *p* i)))
241         (compruebaFactor2 (nth 1 (aref *p* *g*)) (nth 1 (aref *p* i)))
242       )
243       (setf *g* i)
244     )
245   )
```

```
246 )
247
248 (defun compruebaFactor1 (a b)
249   (let
250     (
251       (r T)
252     )
253     (loop for i from 0 to (1- (array-total-size *o*)) do
254       (unless
255         (<=
256           (chi-square-pearson
257             (nth 1 (aref *o* i))
258             (loop for j in b collect (* j (apply #'+ (nth 1 (aref *o* i)))))
259           )
260           (chi-square-pearson
261             (nth 1 (aref *o* i))
262             (loop for j in a collect (* j (apply #'+ (nth 1 (aref *o* i)))))
263           )
264         )
265         (setf r nil)
266       )
267     )
268     r
269   )
270 )
271
272 (defun compruebaFactor2 (a b)
273   (let
274     (
275       (r nil)
276     )
277     (loop for i from 0 to (1- (array-total-size *o*)) do
278       (when
279         (<
280           (chi-square-pearson
281             (nth 1 (aref *o* i))
282             (loop for j in b collect (* j (apply #'+ (nth 1 (aref *o* i)))))
283           )
284           (chi-square-pearson
285             (nth 1 (aref *o* i))
286             (loop for j in a collect (* j (apply #'+ (nth 1 (aref *o* i)))))
287           )
288         )
289         (setf r T)
290       )
291     )
292     r
293   )
294 )
295
296 ;;---------------------------------------------------------------------
297
298 ;;---------------------------------------------------------------------
299 ;;Función para encontrar el peor individuo
300
301 (defun peorParticula nil
302   (setf *-g* 0)
303   (loop for i from 1 to (1- (array-total-size *p*)) do
304     (when
305       (and
306         (compruebaFactorA (nth 1 (aref *p* *-g*)) (nth 1 (aref *p* i)))
307         (compruebaFactorB (nth 1 (aref *p* *-g*)) (nth 1 (aref *p* i)))
308       )
309       (setf *-g* i)
310     )
311   )
312 )
313
```

```
(defun compruebaFactorA (a b)
  (let
    (
      (r T)
    )
    (loop for i from 0 to (1- (array-total-size *o*)) do
      (unless
        (>=
          (chi-square-pearson
            (nth 1 (aref *o* i))
            (loop for j in b collect (* j (apply #'+ (nth 1 (aref *o* i)))))
          )
          (chi-square-pearson
            (nth 1 (aref *o* i))
            (loop for j in a collect (* j (apply #'+ (nth 1 (aref *o* i)))))
          )
        )
        (setf r nil)
      )
    )
    r
  )
)

(defun compruebaFactorB (a b)
  (let
    (
      (r nil)
    )
    (loop for i from 0 to (1- (array-total-size *o*)) do
      (when
        (>
          (chi-square-pearson
            (nth 1 (aref *o* i))
            (loop for j in b collect (* j (apply #'+ (nth 1 (aref *o* i)))))
          )
          (chi-square-pearson
            (nth 1 (aref *o* i))
            (loop for j in a collect (* j (apply #'+ (nth 1 (aref *o* i)))))
          )
        )
        (setf r T)
      )
    )
    r
  )
)
;;-------------------------------------------------------------------------

;;-------------------------------------------------------------------------
;;Función para actualizar las velocidades de las particulas

(defun nuevaVelocidad nil
  (loop for i from 0 to (1- (array-total-size *p*)) do
    (setf (aref *p* i)
      (list
        (nth 0 (aref *p* i))
        (nth 1 (aref *p* i))
        (loop for j in (nth 2 (aref *p* i))
            for k in (nth 1 (aref *p* i))
            for l in (nth 0 (aref *p* i))
            for m in (nth 0 (aref *p* *g*)) collect
          (+
            (* *a* j)
            (* (random *q1*) (- k l))
            (* (random *q2*) (- m l))
          )
        )
```

```
382       )
383     )
384   )
385 )
386
387 ;;---------------------------------------------------------------------
388 ;; Función para actulizar la posición de las particulas
389 ;; f := Función objetivo :: #'f ; li := Limite inferior ; ls := Limite
        superior
390
391 (defun nuevaPosicion nil
392   (let
393     (
394       (p nil)
395       (b NIL)
396       (v NIL)
397     )
398     (loop for i from 0 to (1- (array-total-size *p*)) do
399       (setf p
400         (loop for j in (nth 0 (aref *p* i)) for k in (nth 2 (aref *p* i))
                collect
401           (mod (+ j k) 1.0)
402         )
403       )
404       (if (> (apply #'+ p) 1.0)
405         (setf p (loop for j in p collect (/ j (apply #'+ p))))
406       )
407       (setf p
408         (loop for j in p collect
409           (if (<= j 0.0)
410             (+ j 0.000001)
411             j
412           )
413         )
414       )
415       (setf b (nth 1 (aref *p* i)))
416       (setf v (nth 2 (aref *p* i)))
417       (if
418         (<
419           (funcionObjetivo p)
420           (funcionObjetivo b)
421         )
422         (setf b p)
423       )
424       (setf (aref *p* i) (list p b v))
425     )
426   )
427 )
428 ;;---------------------------------------------------------------------
429 ;;Algoritmo de Optimización por Enjambre de Particulas
430 ;; n := Tamaño del enjambre ; g := Número de iteraciones; f:= Archivo de la
        Base de Datos
431
432 (defun PSO (n g f)
433   (crea-experimento g n)
434   (setf *a* 1.0)
435   (inicializaEnjambre n f)
436   (mejorParticula)
437   (peorParticula)
438   (guarda-generacion 0)
439   (guarda-mejores-individuos)
440   (guarda-mejores-aptitudes)
441   (guarda-peores-aptitudes)
442   (guarda-promedio-aptitudes n)
443   (loop for i from 1 to g do
444     (nuevaVelocidad)
445     (nuevaPosicion)
446     (mejorParticula)
```

```
447     (peorParticula)
448     (guarda-generacion i)
449     (guarda-mejores-individuos)
450     (guarda-mejores-aptitudes)
451     (guarda-peores-aptitudes)
452     (guarda-promedio-aptitudes n)
453     (setf *a* (- *a* (/ 0.5 g)))
454   )
455   (guarda-resultados)
456 )
457
458 ;;-------------------------------------------------------------------------
```

Código D.9: Algoritmo OEP. Prototipo 4

```
1  ;;;;ALgoritmo PSO
2
3  ;;-------------------------------------------------------------------------
4  ;;Archivos importados
5
6  (load "lecturaBD.lisp")
7  (load "PCCP.lisp")
8  ;;-------------------------------------------------------------------------
9
10 ;;-------------------------------------------------------------------------
11 ;;Variables Globales
12
13 (defparameter *o* (make-array 1 :element-type 'list
14                  :initial-element NIL
15                  :adjustable T
16                  :fill-pointer 0))  ; Distribuciones Observadas
17 (defparameter *p* (make-array 1 :element-type 'list
18                  :initial-element NIL
19                  :adjustable T
20                  :fill-pointer 0))  ; Población
21 (defparameter *g* 0)                  ; Mejor Particula
22 (defparameter *-g* 0)                 ; Peor Particula
23 (defparameter *a* 1.0)                  ; Parámetro de escala
24 (defparameter *q1* 1.5)                 ; Valor para la distribución uniforme
       para la influencia propia
25 (defparameter *q2* 1.5)                 ; Valor para la distribución uniforme
       para la influencia grupal
26 (defparameter *path* (string "./"))          ; Directorio para Guartar
       Experimento
27 (defparameter *file* (string "."))           ; Archivo con información el
       experimento
28 (defparameter *mejores-individuos* (string ""))    ; Cadena con mejores
       individuos
29 (defparameter *mejores-aptitudes* (string ""))     ; Cadena con mejores
       aptitudes
30 (defparameter *peores-aptitudes* (string ""))    ; Cadena con peores
       aptitudes
31 (defparameter *promedio-aptitudes* (string ""))    ; Cadena con promedio de
       aptitudes
32
33 ;;-------------------------------------------------------------------------
34
35 ;;-------------------------------------------------------------------------
36 ;;Funciones para crear un experimento de PSO y llevar un recuento de las
       variables
37
38 (defun crea-experimento (n g a)
39   (setf *mejores-individuos* (string ""))
40   (setf *mejores-aptitudes* (string ""))
41   (setf *peores-aptitudes* (string ""))
42   (setf *promedio-aptitudes* (string ""))
43   (setf *path* (concatenate 'string "./PSO/" (write-to-string (
         get-universal-time)) "/"))
```

```
44    (ensure-directories-exist *path*)
45    (ensure-directories-exist (concatenate 'string *path* "Generaciones/
          Posiciones/"))
46    (ensure-directories-exist (concatenate 'string *path* "Generaciones/
          Velocidades/"))
47    (ensure-directories-exist (concatenate 'string *path* "Generaciones/Memoria
          /"))
48    (setf *file* (concatenate 'string *path* "DatosExperimento.txt"))
49    (with-open-file
50      (stream *file* :direction :output
51              :if-exists :supersede
52              :if-does-not-exist :create
53      )
54      (format stream
55        (concatenate 'string "Optimización␣por␣Enjambre␣de␣Partículas~2%~3TTama
              ño␣de␣Población:␣" (write-to-string n) "~%~3TNúmero␣de␣
              Generaciones:␣" (write-to-string g) "~2%~3TFunción␣de␣optimización
              :␣Prueba␣Chi␣Cuadrada␣de␣Pearson␣~%~6TEspacio␣de␣búsqueda:␣
              [0.0,1.0]~%~6TValor␣minimo␣de␣paro:␣" (write-to-string a))
56      )
57    )
58  )
59
60  (defun guarda-generacion (g)
61    (setf *file* (concatenate 'string *path* "Generaciones/Posiciones/" "g_" (
          write-to-string g) ".pos.data"))
62    (with-open-file
63      (stream *file* :direction :output
64              :if-exists :supersede
65              :if-does-not-exist :create
66      )
67      (loop for i from 0 to (1- (array-total-size *p*)) do
68        (format stream
69          (concatenate 'string (remove #\( (remove #\) (write-to-string (nth 0
                (aref *p* i))))) "~%")
70        )
71      )
72    )
73    (setf *file* (concatenate 'string *path* "Generaciones/Velocidades/" "g_" (
          write-to-string g) ".vel.data"))
74    (with-open-file
75      (stream *file* :direction :output
76              :if-exists :supersede
77              :if-does-not-exist :create
78      )
79      (loop for i from 0 to (1- (array-total-size *p*)) do
80        (format stream
81          (concatenate 'string (remove #\( (remove #\) (write-to-string (nth 2
                (aref *p* i))))) "~%")
82        )
83      )
84    )
85    (setf *file* (concatenate 'string *path* "Generaciones/Memoria/" "g_" (
          write-to-string g) ".mem.data"))
86    (with-open-file
87      (stream *file* :direction :output
88              :if-exists :supersede
89              :if-does-not-exist :create
90      )
91      (loop for i from 0 to (1- (array-total-size *p*)) do
92        (format stream
93          (concatenate 'string (remove #\( (remove #\) (write-to-string (nth 1
                (aref *p* i))))) "~%")
94        )
95      )
96    )
97  )
98
```

```
99  (defun guarda-mejores-individuos nil
100   (setf *mejores-individuos* (concatenate 'string *mejores-individuos* (
          remove #\( (remove #\) (write-to-string (nth 1 (aref *p* *g*))))) "~%"
          ))
101 )
102
103 (defun guarda-mejores-aptitudes ()
104   (setf *mejores-aptitudes* (concatenate 'string *mejores-aptitudes* (
          write-to-string (funcionObjetivo (nth 1 (aref *p* *g*)))) "~%"))
105 )
106
107 (defun guarda-peores-aptitudes ()
108   (setf *peores-aptitudes* (concatenate 'string *peores-aptitudes* (
          write-to-string (funcionObjetivo (nth 1 (aref *p* *-g*)))) "~%"))
109 )
110
111 (defun guarda-promedio-aptitudes (n)
112   (setf *promedio-aptitudes*
113     (concatenate 'string *promedio-aptitudes*
114       (write-to-string
115         (/
116           (apply #'+
117             (loop for i from 0 to (1- n) collect
118               (funcionObjetivo (nth 1 (aref *p* i)))
119             )
120           )
121           n
122         )
123       )
124       "~%"
125     )
126   )
127 )
128
129 (defun guarda-resultados nil
130   (setf *file* (concatenate 'string *path* "mejoresIndividuos.data"))
131   (with-open-file
132     (stream *file* :direction :output
133             :if-exists :supersede
134             :if-does-not-exist :create
135     )
136     (format stream *mejores-individuos*)
137   )
138   (setf *file* (concatenate 'string *path* "mejorIndividuo.data"))
139   (with-open-file
140     (stream *file* :direction :output
141             :if-exists :supersede
142             :if-does-not-exist :create
143     )
144     (format stream
145         (concatenate 'string (remove #\( (remove #\) (write-to-string (nth 1
              (aref *p* *g*))))) "~%")
146     )
147   )
148   (setf *file* (concatenate 'string *path* "mejoresAptitudes.data"))
149   (with-open-file
150     (stream *file* :direction :output
151             :if-exists :supersede
152             :if-does-not-exist :create
153     )
154     (format stream *mejores-aptitudes*)
155   )
156   (setf *file* (concatenate 'string *path* "peoresAptitudes.data"))
157   (with-open-file
158     (stream *file* :direction :output
159             :if-exists :supersede
160             :if-does-not-exist :create
161     )
```

```
162     (format stream *peores-aptitudes*)
163   )
164   (setf *file* (concatenate 'string *path* "promedioAptitudes.data"))
165   (with-open-file
166     (stream *file* :direction :output
167             :if-exists :supersede
168             :if-does-not-exist :create
169     )
170     (format stream *promedio-aptitudes*)
171   )
172 )
173 ;;---------------------------------------------------------------------
174
175 ;;---------------------------------------------------------------------
176 ;;Función de lectura de las distribuciones observadas
177 ;; f := Lista de Archivos con distribuciones
178 (defun leerDistribuciones (f)
179   (adjust-array *o* (list (length f)))
180   (loop for i in f for j from 1 to (length f) do
181     (setf (aref *o* (1- j)) (read-db i))
182   )
183 )
184 ;;---------------------------------------------------------------------
185
186 ;;---------------------------------------------------------------------
187 ;;Funcion de optimización
188 ;; Distribución Esperada
189 (defun funcionObjetivo (E)
190   (/
191     (apply #'+
192       (loop for i from 0 to (1- (array-total-size *o*)) collect
193         (chi-square-pearson
194           (nth 1 (aref *o* i))
195           (loop for j in E collect (* j (apply #'+ (nth 1 (aref *o* i)))))
196         )
197       )
198     )
199     (length E)
200   )
201 )
202 ;;---------------------------------------------------------------------
203
204 ;;---------------------------------------------------------------------
205 ;;Función de generación de enjambre
206 ;; n := Tamaño de Enjambre
207
208 (defun inicializaEnjambre (n f)
209   (let
210     (
211       (p NIL)
212       (v NIL)
213     )
214     (leerDistribuciones f)
215     (adjust-array *p* (list n))
216     (loop for i from 0 to (1- n) do
217       (setf p (loop for j from 1 to (nth 0 (nth 0 (aref *o* 0))) collect (
              random 1.0)))
218       (setf p (loop for j in p collect (/ j (apply #'+ p))))
219       (setf v (loop for j from 1 to (nth 0 (nth 0 (aref *o* 0))) collect (-
              0.5 (random 1.0))))
220       ;;La estructura de una particula sera una lista de tres listas, esta
              dada como: (Posición Memoria Velocidad)
221       (setf (aref *p* i) (list p p v))
222     )
223   )
224 )
225 ;;---------------------------------------------------------------------
226
```

```
227 ;;---------------------------------------------------------------------
228 ;;Función para encontrar la mejor particula
229
230 (defun mejorParticula nil
231   (setf *g* 0)
232   (loop for i from 1 to (1- (array-total-size *p*)) do
233     (when
234       (and
235         (compruebaFactor1 (nth 1 (aref *p* *g*)) (nth 1 (aref *p* i)))
236         (compruebaFactor2 (nth 1 (aref *p* *g*)) (nth 1 (aref *p* i)))
237       )
238       (setf *g* i)
239     )
240   )
241 )
242
243 (defun compruebaFactor1 (a b)
244   (let
245     (
246       (r T)
247     )
248     (loop for i from 0 to (1- (array-total-size *o*)) do
249       (unless
250         (<=
251           (chi-square-pearson
252             (nth 1 (aref *o* i))
253             (loop for j in b collect (* j (apply #'+ (nth 1 (aref *o* i)))))
254           )
255           (chi-square-pearson
256             (nth 1 (aref *o* i))
257             (loop for j in a collect (* j (apply #'+ (nth 1 (aref *o* i)))))
258           )
259         )
260         (setf r nil)
261       )
262     )
263     r
264   )
265 )
266
267 (defun compruebaFactor2 (a b)
268   (let
269     (
270       (r nil)
271     )
272     (loop for i from 0 to (1- (array-total-size *o*)) do
273       (when
274         (<
275           (chi-square-pearson
276             (nth 1 (aref *o* i))
277             (loop for j in b collect (* j (apply #'+ (nth 1 (aref *o* i)))))
278           )
279           (chi-square-pearson
280             (nth 1 (aref *o* i))
281             (loop for j in a collect (* j (apply #'+ (nth 1 (aref *o* i)))))
282           )
283         )
284         (setf r T)
285       )
286     )
287     r
288   )
289 )
290 ;;---------------------------------------------------------------------
291
292 ;;---------------------------------------------------------------------
293 ;;Función para encontrar el peor individuo
294
```

```
295 (defun peorParticula nil
296   (setf *-g* 0)
297   (loop for i from 1 to (1- (array-total-size *p*)) do
298     (when
299       (and
300         (compruebaFactorA (nth 1 (aref *p* *-g*)) (nth 1 (aref *p* i)))
301         (compruebaFactorB (nth 1 (aref *p* *-g*)) (nth 1 (aref *p* i)))
302       )
303       (setf *-g* i)
304     )
305   )
306 )
307
308 (defun compruebaFactorA (a b)
309   (let
310     (
311       (r T)
312     )
313     (loop for i from 0 to (1- (array-total-size *o*)) do
314       (unless
315         (>=
316           (chi-square-pearson
317             (nth 1 (aref *o* i))
318             (loop for j in b collect (* j (apply #'+ (nth 1 (aref *o* i)))))
319           )
320           (chi-square-pearson
321             (nth 1 (aref *o* i))
322             (loop for j in a collect (* j (apply #'+ (nth 1 (aref *o* i)))))
323           )
324         )
325         (setf r nil)
326       )
327     )
328     r
329   )
330 )
331
332 (defun compruebaFactorB (a b)
333   (let
334     (
335       (r nil)
336     )
337     (loop for i from 0 to (1- (array-total-size *o*)) do
338       (when
339         (>
340           (chi-square-pearson
341             (nth 1 (aref *o* i))
342             (loop for j in b collect (* j (apply #'+ (nth 1 (aref *o* i)))))
343           )
344           (chi-square-pearson
345             (nth 1 (aref *o* i))
346             (loop for j in a collect (* j (apply #'+ (nth 1 (aref *o* i)))))
347           )
348         )
349         (setf r T)
350       )
351     )
352     r
353   )
354 )
355 ;;-------------------------------------------------------------------------
356
357 ;;-------------------------------------------------------------------------
358 ;;Función para actualizar las velocidades de las particulas
359
360 (defun nuevaVelocidad nil
361   (loop for i from 0 to (1- (array-total-size *p*)) do
362     (setf (aref *p* i)
```

```
363       (list
364         (nth 0 (aref *p* i))
365         (nth 1 (aref *p* i))
366         (loop for j in (nth 2 (aref *p* i))
367             for k in (nth 1 (aref *p* i))
368             for l in (nth 0 (aref *p* i))
369             for m in (nth 0 (aref *p* *g*)) collect
370           (+
371             (* *a* j)
372             (* (random *q1*) (- k l))
373             (* (random *q2*) (- m l))
374           )
375         )
376       )
377     )
378   )
379 )
380 ;;---------------------------------------------------------------------
381
382 ;;---------------------------------------------------------------------
383 ;; Función para actulizar la posición de las particulas
384 ;; f := Función objetivo :: #'f ; li := Limite inferior ; ls := Limite
      superior
385
386 (defun nuevaPosicion nil
387   (let
388     (
389       (p nil)
390       (b NIL)
391       (v NIL)
392     )
393     (loop for i from 0 to (1- (array-total-size *p*)) do
394       (setf p
395         (loop for j in (nth 0 (aref *p* i)) for k in (nth 2 (aref *p* i))
              collect
396           (mod (+ j k) 1.0)
397         )
398       )
399       (if (> (apply #'+ p) 1.0)
400         (setf p (loop for j in p collect (/ j (apply #'+ p))))
401       )
402       (setf p
403         (loop for j in p collect
404           (if (<= j 0.0)
405             (+ j 0.000001)
406             j
407           )
408         )
409       )
410       (setf b (nth 1 (aref *p* i)))
411       (setf v (nth 2 (aref *p* i)))
412       (if
413         (<
414           (funcionObjetivo p)
415           (funcionObjetivo b)
416         )
417         (setf b p)
418       )
419       (setf (aref *p* i) (list p b v))
420     )
421   )
422 )
423 ;;---------------------------------------------------------------------
424
425 ;;---------------------------------------------------------------------
426 ;;Función para revisar la diversidad en la población
427
428 (defun diversidadPoblacion nil
```

```
429   (
430     (lambda (p) (/ (apply #'+ p) (length p)))
431     (apply #'append
432       (loop for j from 0 to (- (array-total-size *p*) 2) collect
433         (loop for k from (1+ j) to (1- (array-total-size *p*)) collect
434           (sqrt
435             (apply #'+
436               (loop for n in (nth 1 (aref *p* j)) for m in (nth 1 (aref *p* k
                      )) collect
437                 (expt (- (* n 100) (* m 100)) 2)
438               )
439             )
440           )
441         )
442       )
443     )
444   )
445 )
446 ;;---------------------------------------------------------------------
447
448 ;;---------------------------------------------------------------------
449 ;;Función de verificación de aptitud aceptable
450 ;; c := Limite de aceptación
451
452 (defun revisarAptitud (c)
453   (let
454     (
455       (r T)
456     )
457     (loop for i from 0 to (1- (array-total-size *o*)) do
458       (unless
459         (<
460           (chi-square-pearson
461             (nth 1 (aref *o* i))
462             (loop for j in (nth 1 (aref *p* *g*)) collect (* j (apply #'+ (
                      nth 1 (aref *o* i)))))
463           )
464           (* c (1- (nth 0 (nth 0 (aref *o* i)))))
465         )
466         (setf r nil)
467       )
468     )
469     r
470   )
471 )
472 ;;---------------------------------------------------------------------
473
474 ;;---------------------------------------------------------------------
475 ;;Algoritmo de Optimización por Enjambre de Particulas
476 ;; n := Tamaño del enjambre ; g := Número de iteraciones; f:= Archivo de la
      Base de Datos ; a := Valor minimo de aprovación
477
478 (defun PSO (n g f a)
479   (crea-experimento n g a)
480   (setf *a* 1.0)
481   (inicializaEnjambre n f)
482   (mejorParticula)
483   (peorParticula)
484   (guarda-generacion 0)
485   (guarda-mejores-individuos)
486   (guarda-mejores-aptitudes)
487   (guarda-peores-aptitudes)
488   (guarda-promedio-aptitudes n)
489   (loop for i from 1 to g do
490     (nuevaVelocidad)
491     (nuevaPosicion)
492     (mejorParticula)
493     (peorParticula)
```

```
494     (guarda-generacion i)
495     (guarda-mejores-individuos)
496     (guarda-mejores-aptitudes)
497     (guarda-peores-aptitudes)
498     (guarda-promedio-aptitudes n)
499     (setf *a* (- *a* (/ 0.5 g)))
500     (if (revisarAptitud a)
501       (setf i (1+ g))
502     )
503   )
504   (guarda-resultados)
505 )
506 ;;-------------------------------------------------------------------------
507
508 (PSO 100 1000 '("prueba.data" "prueba2.data" "prueba3.data") 0.0)
```

D.5.2. Estrategias Evolutivas

Código D.10: Algoritmo EE. Prototipo 1

```
1  ;;;;ALgoritmo EE
2
3  ;-------------------------------------------------------------------------
4  ;;Archivos importados
5
6  (load "funciones.lisp")
7
8  ;-------------------------------------------------------------------------
9
10 ;-------------------------------------------------------------------------
11 ;;Variables Globales
12
13 (defparameter *u* (make-array 1 :element-type 'list
14                 :initial-element NIL
15                 :adjustable T
16                 :fill-pointer 0))  ; Población
17 (defparameter *h* (make-array 1 :element-type 'list
18                 :initial-element NIL
19                 :adjustable T
20                 :fill-pointer 0))  ; Hijos generados
21 (defparameter *g* 0)                  ; Mejor individuo
22 (defparameter *path* (string "./"))          ;Directorio para Guartar
       Experimento
23 (defparameter *file* (string "."))           ;Archivo con información el
       experimento
24 (defparameter *version* (string "()"))         ;Version del algoritmo
25 (defparameter *mejores-individuos* (string ""))    ;Cadena con mejores
       individuos
26
27 ;-------------------------------------------------------------------------
28
29 ;-------------------------------------------------------------------------
30 ;;Funciones para crear un experimento de ED y llevar un recuento de las
       variables
31
32 (defun crea-experimento (f n g li ls v c)
33   (setf *mejores-individuos* (string ""))
34   (setf *path* (concatenate 'string "./EE/" (remove #\> (subseq (
         write-to-string v) 11)) "/" (write-to-string (get-universal-time)) "/"
         ))
35   (ensure-directories-exist *path*)
36   (ensure-directories-exist (concatenate 'string *path* "Generaciones/
         Individuos/"))
37   (ensure-directories-exist (concatenate 'string *path* "Generaciones/
         Desviaciones/"))
38   (setf *file* (concatenate 'string *path* "DatosExperimento.txt"))
```

```
39    (with-open-file
40      (stream *file* :direction :output
41              :if-exists :supersede
42              :if-does-not-exist :create
43      )
44      (if (string= "SINSOBREVIVIENTES" (remove #\> (subseq (write-to-string v)
            11)))
45        (setf *version* (string "(u,h)"))
46      )
47      (if (string= "CONSOBREVIVIENTES" (remove #\> (subseq (write-to-string v)
            11)))
48        (setf *version* (string "(u+h)"))
49      )
50      (format stream
51        (concatenate 'string "Estrategia␣Evolutiva~2%~3TVersión:␣" *version* "-
              EE~%~3TTamaño␣de␣Población:␣" (write-to-string n) "~%~3TNúmero␣de␣
              Generaciones:␣" (write-to-string g) "~%~3THijos␣por␣Padre:␣" (
              write-to-string c) "~2%~3TFunción␣de␣optimización:␣" (remove #\> (
              subseq (write-to-string f) 11)) "~%~6TEspacio␣de␣búsqueda:␣[" (
              write-to-string li) "," (write-to-string ls) "]")
52      )
53    )
54  )
55
56  (defun guarda-generacion (g)
57    (setf *file* (concatenate 'string *path* "Generaciones/Individuos/" "g." (
          write-to-string g) ".data"))
58    (with-open-file
59      (stream *file* :direction :output
60              :if-exists :supersede
61              :if-does-not-exist :create
62      )
63      (loop for i from 0 to (1- (array-total-size *u*)) do
64        (format stream
65          (concatenate 'string (write-to-string (nth 0 (nth 0 (aref *u* i)))) "
                ␣" (write-to-string (nth 1 (nth 0 (aref *u* i)))) "~%")
66        )
67      )
68    )
69  )
70
71  (defun guarda-desviaciones (g)
72    (setf *file* (concatenate 'string *path* "Generaciones/Desviaciones/" "d."
          (write-to-string g) ".data"))
73    (with-open-file
74      (stream *file* :direction :output
75              :if-exists :supersede
76              :if-does-not-exist :create
77      )
78      (loop for i from 0 to (1- (array-total-size *u*)) do
79        (format stream
80          (concatenate 'string (write-to-string (nth 0 (nth 1 (aref *u* i)))) "
                ␣" (write-to-string (nth 1 (nth 1 (aref *u* i)))) "~%")
81        )
82      )
83    )
84  )
85
86  (defun guarda-mejores-individuos ()
87    (setf *mejores-individuos* (concatenate 'string *mejores-individuos* (
          write-to-string (nth 0 (nth 0 (aref *u* *g*)))) "␣" (write-to-string (
          nth 1 (nth 0 (aref *u* *g*)))) "~%"))
88  )
89
90  (defun guarda-resultados ()
91    (setf *file* (concatenate 'string *path* "mejoresIndividuos.data"))
92    (with-open-file
93      (stream *file* :direction :output
```

```
94              :if-exists :supersede
95              :if-does-not-exist :create
96      )
97      (format stream *mejores-individuos*)
98    )
99    (setf *file* (concatenate 'string *path* "mejorIndividuo.data"))
100   (with-open-file
101     (stream *file* :direction :output
102             :if-exists :supersede
103             :if-does-not-exist :create
104     )
105     (format stream
106         (concatenate 'string (write-to-string (nth 0 (nth 0 (aref *u* *g*))))
                "␣" (write-to-string (nth 1 (nth 0 (aref *u* *g*)))) "~%")
107     )
108   )
109 )
110
111 ;---------------------------------------------------------------------
112
113 ;---------------------------------------------------------------------
114 ;;Función de generación de problación
115 ;; n := Tamaño de porblación ; li := Limite inferior ; ls := Limite Superior
116
117 (defun inicializaPoblacion (n li ls c)
118   (adjust-array *u* (list n))
119   (adjust-array *h* (list (* n c)))
120   (loop for i from 0 to (1- n) do
121     ;;El individuo es un lista de dos listas, la primera es su cromosoma y la
            segunda su vector de desviación estandar
122     (setf (aref *u* i) (list (list (+ (random (- ls li)) li) (+ (random (- ls
            li)) li)) (list (/ (random (- ls li)) 2.0) (/ (random (- ls li))
          2.0))))
123   )
124 )
125
126 ;---------------------------------------------------------------------
127
128 ;---------------------------------------------------------------------
129 ;;Función para encontrar el mejor individuo
130 ;; f := Función objetivo :: #'f
131
132 (defun mejorIndividuo (f)
133   (setf *g* 0)
134   (loop for i from 1 to (1- (array-total-size *u*)) do
135     (if (< (funcall f (nth 0 (nth 0 (aref *u* i)))) (nth 1 (nth 0 (aref *u* i)
            ))) (funcall f (nth 0 (nth 0 (aref *u* *g*))) (nth 1 (nth 0 (aref *u
            * *g*)))))
136       (setf *g* i)
137     )
138   )
139 )
140 ;---------------------------------------------------------------------
141
142 ;---------------------------------------------------------------------
143 ;;Función para generar los hijos
144 ;; c := Cantidad de hijos por padre
145
146 (defun mutarPadres (li ls c)
147   (let
148     (
149       (k 0)
150       (v nil)
151     )
152     (loop for i from 0 to (1- (array-total-size *u*)) do
153       (loop for j from 1 to c do
154         (setf v (mapcar #'(lambda (u d) (+ u (/ (exp (* -0.5 (expt (/ (random
              (abs u)) d) 2))) (* d (sqrt (* 2 pi)))))) (nth 0 (aref *u* i))
```

```
              (nth 1 (aref *u* i))))
155         (if (< (nth 0 v) li)
156           (setf v (list li (nth 1 v)))
157         )
158         (if (> (nth 0 v) ls)
159           (setf v (list ls (nth 1 v)))
160         )
161         (if (< (nth 1 v) li)
162           (setf v (list (nth 0 v) li))
163         )
164         (if (> (nth 1 v) ls)
165           (setf v (list (nth 0 v) ls))
166         )
167         (setf (aref *h* k) (list v (nth 1 (aref *u* i))))
168         (setf k (1+ k))
169       )
170     )
171   )
172 )
173
174 ;---------------------------------------------------------------------
175
176 ;---------------------------------------------------------------------
177 ;;Función para seleccionar la nueva población
178 ;; f := Función objetivo :: #'f ;  v := Versión de selección :: #'f
179
180 (defun sinSobrevivientes (f)
181   (let
182     (
183       (l NIL)
184       (r 0)
185     )
186     (loop for i from 0 to (1- (array-total-size *u*)) do
187       (loop for j from 0 to (1- (array-total-size *h*)) do
188         (unless (find j l)
189           (if (< (funcall f (nth 0 (nth 0 (aref *h* j))) (nth 1 (nth 0 (aref
                  *h* j)))) (funcall f (nth 0 (nth 0 (aref *h* r))) (nth 1 (nth
                  0 (aref *h* r)))))
190             (setf r j)
191           )
192         )
193       )
194       (setf l (append l (list r)))
195       (setf r 0)
196     )
197     (loop for i from 0 to (1- (array-total-size *u*)) do
198       (setf (aref *u* i) (aref *h* (nth i l)))
199     )
200   )
201 )
202
203 (defun conSobrevivientes (f)
204   (let
205     (
206       (h (make-array (+ (array-total-size *u*) (array-total-size *h*)) :
              element-type 'list
207                                       :initial-element NIL
208                                       :adjustable T
209                                         :fill-pointer 0))
210       (l NIL)
211       (r 0)
212     )
213     (loop for i from 0 to (1- (array-total-size *u*)) do
214       (setf (aref h i) (aref *u* i))
215     )
216     (loop for i from 0 to (1- (array-total-size *h*)) do
217       (setf (aref h (+ i (array-total-size *u*))) (aref *h* i))
218     )
```

```
219     (loop for i from 0 to (1- (array-total-size *u*)) do
220       (loop for j from 0 to (1- (array-total-size h)) do
221         (unless (find j l)
222           (if (< (funcall f (nth 0 (nth 0 (aref h j))) (nth 1 (nth 0 (aref h
                  j)))) (funcall f (nth 0 (nth 0 (aref h r))) (nth 1 (nth 0 (
                  aref h r)))))
223             (setf r j)
224           )
225         )
226       )
227       (setf l (append l (list r)))
228       (setf r 0)
229     )
230     (loop for i from 0 to (1- (array-total-size *u*)) do
231       (setf (aref *u* i) (aref h (nth i l)))
232     )
233   )
234 )
235
236 (defun nuevaPoblacion (f v)
237   (funcall v f)
238 )
239
240 ;---------------------------------------------------------------------
241
242 ;---------------------------------------------------------------------
243 ;;Algoritmo de Evolución Estrategica
244 ;; n := Tamaño del población ; g := Número de generaciones ; f := Función
       objetivo  :: #'f ; li := Limite inferior ; ls := Limite superior ; c :=
       Cantidad de hijos por padre ; v := Versión del algoritmo :: T para (u+h)
       -EE y NIL para (u,h)-EE
245
246 (defun Estrategia-Evolutiva (f n g li ls c v)
247   (crea-experimento f n g li ls v c)
248   (inicializaPoblacion n li ls c)
249   (guarda-generacion 0)
250   (guarda-desviaciones 0)
251   (mejorIndividuo f)
252   (guarda-mejores-individuos)
253   (loop for i from 1 to g do
254     (mutarPadres li ls c)
255     (nuevaPoblacion f v)
256     (guarda-generacion i)
257     (guarda-desviaciones i)
258     (mejorIndividuo f)
259     (guarda-mejores-individuos)
260   )
261   (guarda-resultados)
262 )
263
264 ;---------------------------------------------------------------------
```

Código D.11: Algoritmo EE. Prototipo 2

```
1  ;;;;ALgoritmo EE
2
3  ;;-------------------------------------------------------------------------
4  ;;;Archivos importados
5
6  (load "lecturaBD.lisp")
7  (load "PCCP.lisp")
8
9  ;;-------------------------------------------------------------------------
10
11 ;;-------------------------------------------------------------------------
12 ;;;Variables Globales
13
14 (defparameter *db* nil)                ;Información de la Base de Datos
```

```
15 (defparameter *u* (make-array 1 :element-type 'list
16                 :initial-element NIL
17                 :adjustable T
18                 :fill-pointer 0))  ;Población
19 (defparameter *h* (make-array 1 :element-type 'list
20                 :initial-element NIL
21                 :adjustable T
22                 :fill-pointer 0))  ;Hijos generados
23 (defparameter *g* 0)                  ;Mejor individuo
24 (defparameter *-g* 0)                 ;Peor individuo
25 (defparameter *path* (string "./"))          ;Directorio para Guartar
       Experimento
26 (defparameter *file* (string "."))           ;Archivo con información el
       experimento
27 (defparameter *version* (string "()"))         ;Version del algoritmo
28 (defparameter *mejores-individuos* (string ""))    ;Cadena con mejores
       individuos
29 (defparameter *mejores-aptitudes* (string ""))     ; Cadena con mejores
       aptitudes
30 (defparameter *peores-aptitudes* (string ""))    ; Cadena con peores
       aptitudes
31 (defparameter *promedio-aptitudes* (string ""))    ; Cadena con promedio de
       aptitudes
32
33 ;;---------------------------------------------------------------------
34
35 ;;---------------------------------------------------------------------
36 ;;Funciones para crear un experimento de ED y llevar un recuento de las
       variables
37
38 (defun crea-experimento (n g v c)
39   (setf *mejores-individuos* (string ""))
40   (setf *mejores-aptitudes* (string ""))
41   (setf *peores-aptitudes* (string ""))
42   (setf *promedio-aptitudes* (string ""))
43   (setf *path* (concatenate 'string "./EE/" (remove #\> (subseq (
         write-to-string v) 11)) "/" (write-to-string (get-universal-time)) "/"
         ))
44   (ensure-directories-exist *path*)
45   (ensure-directories-exist (concatenate 'string *path* "Generaciones/
         Individuos/"))
46   (ensure-directories-exist (concatenate 'string *path* "Generaciones/
         Desviaciones/"))
47   (setf *file* (concatenate 'string *path* "DatosExperimento.txt"))
48   (with-open-file
49     (stream *file* :direction :output
50           :if-exists :supersede
51           :if-does-not-exist :create
52     )
53     (if (string= "SINSOBREVIVIENTES" (remove #\> (subseq (write-to-string v)
           11)))
54       (setf *version* (string "(u,h)"))
55     )
56     (if (string= "CONSOBREVIVIENTES" (remove #\> (subseq (write-to-string v)
           11)))
57       (setf *version* (string "(u+h)"))
58     )
59     (format stream
60       (concatenate 'string "Estrategia␣Evolutiva~2%~3TVersión:␣" *version* "-
           EE~%~3TTamaño␣de␣Población:␣" (write-to-string n) "~%~3TNúmero␣de␣
           Generaciones:␣" (write-to-string g) "~%~3THijos␣por␣Padre:␣" (
           write-to-string c) "~2%~3TFunción␣de␣optimización:␣Prueba␣Chi␣
           Cuadrada␣de␣Pearson~%~6TEspacio␣de␣búsqueda:␣(0.0,1.0)")
61     )
62   )
63 )
64
65 (defun guarda-generacion (g)
```

```
66    (setf *file* (concatenate 'string *path* "Generaciones/Individuos/" "g." (
          write-to-string g) ".data"))
67    (with-open-file
68      (stream *file* :direction :output
69              :if-exists :supersede
70              :if-does-not-exist :create
71      )
72      (loop for i from 0 to (1- (array-total-size *u*)) do
73        (format stream
74          (concatenate 'string (remove #\( (remove #\) (write-to-string (nth 0
                (aref *u* i))))) "~%")
75        )
76      )
77    )
78  )
79
80  (defun guarda-desviaciones (g)
81    (setf *file* (concatenate 'string *path* "Generaciones/Desviaciones/" "d."
          (write-to-string g) ".data"))
82    (with-open-file
83      (stream *file* :direction :output
84              :if-exists :supersede
85              :if-does-not-exist :create
86      )
87      (loop for i from 0 to (1- (array-total-size *u*)) do
88        (format stream
89          (concatenate 'string (remove #\( (remove #\) (write-to-string (nth 1
                (aref *u* i))))) "~%")
90        )
91      )
92    )
93  )
94
95  (defun guarda-mejores-individuos ()
96    (setf *mejores-individuos* (concatenate 'string *mejores-individuos* (
          remove #\( (remove #\) (write-to-string (nth 0 (aref *u* *g*))))) "~%"
          ))
97  )
98
99  (defun guarda-mejores-aptitudes ()
100   (setf *mejores-aptitudes* (concatenate 'string *mejores-aptitudes* (
          write-to-string (chi-square-pearson (nth 1 *db*) (loop for j in (nth 0
           (aref *u* *g*)) collect (* j (apply #'+ (nth 1 *db*)))))) "~%"))
101 )
102
103 (defun guarda-peores-aptitudes ()
104   (setf *peores-aptitudes* (concatenate 'string *peores-aptitudes* (
          write-to-string (chi-square-pearson (nth 1 *db*) (loop for j in (nth 0
           (aref *u* *-g*)) collect (* j (apply #'+ (nth 1 *db*)))))) "~%"))
105 )
106
107 (defun guarda-promedio-aptitudes (n)
108   (setf *promedio-aptitudes*
109     (concatenate 'string *promedio-aptitudes*
110       (write-to-string
111         (/
112           (apply #'+
113             (loop for i from 0 to (1- n) collect
114               (chi-square-pearson (nth 1 *db*) (loop for j in (nth 0 (aref *u
                      * i)) collect (* j (apply #'+ (nth 1 *db*)))))
115             )
116           )
117           n
118         )
119       )
120       "~%"
121     )
122   )
```

```
123 )
124
125 (defun guarda-resultados ()
126   (setf *file* (concatenate 'string *path* "mejoresIndividuos.data"))
127   (with-open-file
128     (stream *file* :direction :output
129             :if-exists :supersede
130             :if-does-not-exist :create
131     )
132     (format stream *mejores-individuos*)
133   )
134   (setf *file* (concatenate 'string *path* "mejorIndividuo.data"))
135   (with-open-file
136     (stream *file* :direction :output
137             :if-exists :supersede
138             :if-does-not-exist :create
139     )
140     (format stream
141         (concatenate 'string (remove #\( (remove #\) (write-to-string (nth 0
                (aref *u* *g*))))) "~%")
142     )
143   )
144   (setf *file* (concatenate 'string *path* "mejoresAptitudes.data"))
145   (with-open-file
146     (stream *file* :direction :output
147             :if-exists :supersede
148             :if-does-not-exist :create
149     )
150     (format stream *mejores-aptitudes*)
151   )
152   (setf *file* (concatenate 'string *path* "peoresAptitudes.data"))
153   (with-open-file
154     (stream *file* :direction :output
155             :if-exists :supersede
156             :if-does-not-exist :create
157     )
158     (format stream *peores-aptitudes*)
159   )
160   (setf *file* (concatenate 'string *path* "promedioAptitudes.data"))
161   (with-open-file
162     (stream *file* :direction :output
163             :if-exists :supersede
164             :if-does-not-exist :create
165     )
166     (format stream *promedio-aptitudes*)
167   )
168 )
169
170 ;;---------------------------------------------------------------------
171
172 ;;---------------------------------------------------------------------
173 ;;Función de generación de problación
174 ;; n := Tamaño de porblación ; c := Cantidad de Hijos por Padre ; f :=
        Archivo de la Base de Datos
175
176 (defun inicializaPoblacion (n c f)
177   (let
178     (
179       (u nil)
180       (d nil)
181     )
182     (setf *db* (read-db f))
183     (adjust-array *u* (list n))
184     (adjust-array *h* (list (* n c)))
185     (loop for i from 0 to (1- n) do
186       ;;El individuo es un lista de dos listas, la primera es su cromosoma y
              la segunda su vector de desviación estandar
187       (setf u (loop for j from 1 to (nth 0 (nth 0 *db*)) collect (random 1.0)
```

```
          ))
188       (setf u (loop for j in u collect (/ j (apply #'+ u))))
189       (setf d (loop for j from 1 to (nth 0 (nth 0 *db*)) collect (random 0.5)
              ))
190       (setf (aref *u* i) (list u d))
191     )
192   )
193 )
194 ;;---------------------------------------------------------------------
195
196 ;;---------------------------------------------------------------------
197 ;;Función para encontrar el mejor individuo
198
199 (defun mejorIndividuo nil
200   (setf *g* 0)
201   (loop for i from 1 to (1- (array-total-size *u*)) do
202     (if (<
203         (chi-square-pearson (nth 1 *db*) (loop for j in (nth 0 (aref *u* i))
              collect (* j (apply #'+ (nth 1 *db*)))))
204         (chi-square-pearson (nth 1 *db*) (loop for j in (nth 0 (aref *u* *g*)
              ) collect (* j (apply #'+ (nth 1 *db*)))))
205       )
206       (setf *g* i)
207     )
208   )
209 )
210 ;;---------------------------------------------------------------------
211
212 ;;---------------------------------------------------------------------
213 ;;Función para encontrar el peor individuo
214
215 (defun peorIndividuo nil
216   (setf *-g* 0)
217   (loop for i from 1 to (1- (array-total-size *u*)) do
218     (if (>
219         (chi-square-pearson (nth 1 *db*) (loop for j in (nth 0 (aref *u* i))
              collect (* j (apply #'+ (nth 1 *db*)))))
220         (chi-square-pearson (nth 1 *db*) (loop for j in (nth 0 (aref *u* *-g
              *)) collect (* j (apply #'+ (nth 1 *db*)))))
221       )
222       (setf *-g* i)
223     )
224   )
225 )
226 ;;---------------------------------------------------------------------
227
228 ;;---------------------------------------------------------------------
229 ;;Función para generar los hijos
230 ;; c := Cantidad de hijos por padre
231
232 (defun mutarPadres (c)
233   (let
234     (
235       (k 0)
236       (v nil)
237     )
238     (loop for i from 0 to (1- (array-total-size *u*)) do
239       (loop for j from 1 to c do
240         (setf v
241           (mapcar
242             #'(lambda
243                 (u d)
244                 (+ u (/ (exp (* -0.5 (expt (/ u d) 2))) (* d (sqrt (* 2 pi)))))
245             )
246             (nth 0 (aref *u* i)) (nth 1 (aref *u* i))
247           )
248         )
249         (setf v (loop for j in v collect (/ j (apply #'+ v))))
```

```
250           (setf (aref *h* k) (list v (nth 1 (aref *u* i))))
251           (setf k (1+ k))
252         )
253       )
254     )
255   )
256
257   ;;-------------------------------------------------------------------
258
259   ;;-------------------------------------------------------------------
260   ;;Función para seleccionar la nueva población
261   ;; v := Versión de selección :: #'v
262
263   (defun sinSobrevivientes nil
264     (let
265       (
266         (l NIL)
267         (r 0)
268       )
269       (loop for i from 0 to (1- (array-total-size *u*)) do
270         (loop for j from 0 to (1- (array-total-size *h*)) do
271           (unless (find j l)
272             (if
273               (<
274                 (chi-square-pearson (nth 1 *db*) (loop for k in (nth 0 (aref *h
                        * j)) collect (* k (apply #'+ (nth 1 *db*)))))
275                 (chi-square-pearson (nth 1 *db*) (loop for k in (nth 0 (aref *h
                        * r)) collect (* k (apply #'+ (nth 1 *db*)))))
276               )
277               (setf r j)
278             )
279           )
280         )
281         (setf l (append l (list r)))
282         (setf r 0)
283       )
284       (loop for i from 0 to (1- (array-total-size *u*)) do
285         (setf (aref *u* i) (aref *h* (nth i l)))
286       )
287     )
288   )
289
290   (defun conSobrevivientes nil
291     (let
292       (
293         (h (make-array (+ (array-total-size *u*) (array-total-size *h*)) :
                element-type 'list
294                                             :initial-element NIL
295                                             :adjustable T
296                                               :fill-pointer 0))
297         (l NIL)
298         (r 0)
299       )
300       (loop for i from 0 to (1- (array-total-size *u*)) do
301         (setf (aref h i) (aref *u* i))
302       )
303       (loop for i from 0 to (1- (array-total-size *h*)) do
304         (setf (aref h (+ i (array-total-size *u*))) (aref *h* i))
305       )
306       (loop for i from 0 to (1- (array-total-size *u*)) do
307         (loop for j from 0 to (1- (array-total-size h)) do
308           (unless (find j l)
309             (if
310               (<
311                 (chi-square-pearson (nth 1 *db*) (loop for k in (nth 0 (aref h
                        j)) collect (* k (apply #'+ (nth 1 *db*)))))
312                 (chi-square-pearson (nth 1 *db*) (loop for k in (nth 0 (aref h
                        r)) collect (* k (apply #'+ (nth 1 *db*)))))
```

```
313             )
314             (setf r j)
315           )
316         )
317       )
318       (setf l (append l (list r)))
319       (setf r 0)
320     )
321     (loop for i from 0 to (1- (array-total-size *u*)) do
322       (setf (aref *u* i) (aref h (nth i l)))
323     )
324   )
325 )
326
327 (defun nuevaPoblacion (v)
328   (funcall v)
329 )
330
331 ;;---------------------------------------------------------------------
332
333 ;;---------------------------------------------------------------------
334 ;; Función de auto-adaptación de las desviaciones estandar
335 (defun autoAdaptacion nil
336   (loop for i from 0 to (1- (array-total-size *u*)) do
337     (setf
338       (aref *u* i)
339       (list
340         (nth 0 (aref *u* i))
341         (loop for j in (nth 1 (aref *u* i)) collect
342           (*
343             j
344             (exp
345               (*
346                 ((lambda (u) (/ (exp (* -0.5 (expt u 2))) (sqrt (* 2 pi)))) j
                        )
347                 (/ 1 (sqrt (length (nth 0 (aref *u* i)))))
348               )
349             )
350           )
351         )
352       )
353     )
354   )
355 )
356 ;;---------------------------------------------------------------------
357
358 ;;---------------------------------------------------------------------
359 ;;Algoritmo de Evolución Estrategica
360 ;; n := Tamaño del población ; g := Número de generaciones ; f := Archivo de
        la Base de Datos ; c := Cantidad de hijos por padre ; v := Versión del
        algoritmo :: T para (u+h)-EE y NIL para (u,h)-EE
361
362 (defun Estrategia-Evolutiva (f n g c v)
363   (crea-experimento n g v c)
364   (inicializaPoblacion n c f)
365   (guarda-generacion 0)
366   (guarda-desviaciones 0)
367   (mejorIndividuo)
368   (peorIndividuo)
369   (guarda-mejores-individuos)
370   (guarda-mejores-aptitudes)
371   (guarda-peores-aptitudes)
372   (guarda-promedio-aptitudes n)
373   (loop for i from 1 to g do
374     (mutarPadres c)
375     (nuevaPoblacion v)
376     (autoAdaptacion)
377     (guarda-generacion i)
```

```
    (guarda-desviaciones i)
    (mejorIndividuo)
    (peorIndividuo)
    (guarda-mejores-aptitudes)
    (guarda-peores-aptitudes)
    (guarda-promedio-aptitudes n)
    (guarda-mejores-individuos)
  )
  (guarda-resultados)
)

;;-------------------------------------------------------------------
```

Código D.12: Algoritmo EE. Prototipo 3

```
;;;;ALgoritmo EE

;;-------------------------------------------------------------------
;;;Archivos importados

(load "lecturaBD.lisp")
(load "PCCP.lisp")

;;;;-------------------------------------------------------------------

;;;;-------------------------------------------------------------------
;;;;Variables Globales

(defparameter *o* (make-array 1 :element-type 'list
                :initial-element NIL
                :adjustable T
                :fill-pointer 0))  ; Distribuciones Observadas
(defparameter *u* (make-array 1 :element-type 'list
                :initial-element NIL
                :adjustable T
                :fill-pointer 0))  ;Población
(defparameter *h* (make-array 1 :element-type 'list
                :initial-element NIL
                :adjustable T
                :fill-pointer 0))  ;Hijos generados
(defparameter *g* 0)                ;Mejor individuo
(defparameter *-g* 0)               ;Peor individuo
(defparameter *path* (string "./"))          ;Directorio para Guartar
    Experimento
(defparameter *file* (string "."))           ;Archivo con información el
    experimento
(defparameter *version* (string "()"))         ;Version del algoritmo
(defparameter *mejores-individuos* (string ""))    ;Cadena con mejores
    individuos
(defparameter *mejores-aptitudes* (string ""))     ; Cadena con mejores
    aptitudes
(defparameter *peores-aptitudes* (string ""))    ; Cadena con peores
    aptitudes
(defparameter *promedio-aptitudes* (string ""))    ; Cadena con promedio de
    aptitudes

;;;-------------------------------------------------------------------

;;;-------------------------------------------------------------------
;;;Funciones para crear un experimento de ED y llevar un recuento de las
    variables

(defun crea-experimento (n g v c)
  (setf *mejores-individuos* (string ""))
  (setf *mejores-aptitudes* (string ""))
  (setf *peores-aptitudes* (string ""))
  (setf *promedio-aptitudes* (string ""))
```

```
46    (setf *path* (concatenate 'string "./EE/" (remove #\> (subseq (
          write-to-string v) 11)) "/" (write-to-string (get-universal-time)) "/"
          ))
47    (ensure-directories-exist *path*)
48    (ensure-directories-exist (concatenate 'string *path* "Generaciones/
          Individuos/"))
49    (ensure-directories-exist (concatenate 'string *path* "Generaciones/
          Desviaciones/"))
50    (setf *file* (concatenate 'string *path* "DatosExperimento.txt"))
51    (with-open-file
52      (stream *file* :direction :output
53            :if-exists :supersede
54            :if-does-not-exist :create
55      )
56      (if (string= "SINSOBREVIVIENTES" (remove #\> (subseq (write-to-string v)
              11)))
57        (setf *version* (string "(u,h)"))
58      )
59      (if (string= "CONSOBREVIVIENTES" (remove #\> (subseq (write-to-string v)
              11)))
60        (setf *version* (string "(u+h)"))
61      )
62      (format stream
63        (concatenate 'string "Estrategia␣Evolutiva~2%~3TVersión:␣" *version* "-
              EE~%~3TTamaño␣de␣Población:␣" (write-to-string n) "~%~3TNúmero␣de␣
              Generaciones:␣" (write-to-string g) "~%~3THijos␣por␣Padre:␣" (
              write-to-string c) "~2%~3TFunción␣de␣optimización:␣Prueba␣Chi␣
              Cuadrada␣de␣Pearson~%~6TEspacio␣de␣búsqueda:␣[0.0,1.0]")
64      )
65    )
66  )
67
68  (defun guarda-generacion (g)
69    (setf *file* (concatenate 'string *path* "Generaciones/Individuos/" "g." (
          write-to-string g) ".data"))
70    (with-open-file
71      (stream *file* :direction :output
72                :if-exists :supersede
73                :if-does-not-exist :create
74      )
75      (loop for i from 0 to (1- (array-total-size *u*)) do
76        (format stream
77          (concatenate 'string (remove #\( (remove #\) (write-to-string (nth 0
                (aref *u* i))))) "~%")
78        )
79      )
80    )
81  )
82
83  (defun guarda-desviaciones (g)
84    (setf *file* (concatenate 'string *path* "Generaciones/Desviaciones/" "d."
          (write-to-string g) ".data"))
85    (with-open-file
86      (stream *file* :direction :output
87                :if-exists :supersede
88                :if-does-not-exist :create
89      )
90      (loop for i from 0 to (1- (array-total-size *u*)) do
91        (format stream
92          (concatenate 'string (remove #\( (remove #\) (write-to-string (nth 1
                (aref *u* i))))) "~%")
93        )
94      )
95    )
96  )
97
98  (defun guarda-mejores-individuos ()
99    (setf *mejores-individuos* (concatenate 'string *mejores-individuos* (
```

```
        remove #\( (remove #\) (write-to-string (nth 0 (aref *u* *g*))))) "~%"
        ))
100 )
101
102 (defun guarda-mejores-aptitudes ()
103   (setf *mejores-aptitudes* (concatenate 'string *mejores-aptitudes* (
        write-to-string (funcionObjetivo (nth 0 (aref *u* *g*)))) "~%"))
104 )
105
106 (defun guarda-peores-aptitudes ()
107   (setf *peores-aptitudes* (concatenate 'string *peores-aptitudes* (
        write-to-string (funcionObjetivo (nth 0 (aref *u* *-g*)))) "~%"))
108 )
109
110 (defun guarda-promedio-aptitudes (n)
111   (setf *promedio-aptitudes*
112     (concatenate 'string *promedio-aptitudes*
113       (write-to-string
114         (/
115           (apply #'+
116             (loop for i from 0 to (1- n) collect
117               (funcionObjetivo (nth 0 (aref *u* i)))
118             )
119           )
120           n
121         )
122       )
123       "~%"
124     )
125   )
126 )
127
128 (defun guarda-resultados ()
129   (setf *file* (concatenate 'string *path* "mejoresIndividuos.data"))
130   (with-open-file
131     (stream *file* :direction :output
132             :if-exists :supersede
133             :if-does-not-exist :create
134     )
135     (format stream *mejores-individuos*)
136   )
137   (setf *file* (concatenate 'string *path* "mejorIndividuo.data"))
138   (with-open-file
139     (stream *file* :direction :output
140             :if-exists :supersede
141             :if-does-not-exist :create
142     )
143     (format stream
144         (concatenate 'string (remove #\( (remove #\) (write-to-string (nth 0
              (aref *u* *g*))))) "~%")
145     )
146   )
147   (setf *file* (concatenate 'string *path* "mejoresAptitudes.data"))
148   (with-open-file
149     (stream *file* :direction :output
150             :if-exists :supersede
151             :if-does-not-exist :create
152     )
153     (format stream *mejores-aptitudes*)
154   )
155   (setf *file* (concatenate 'string *path* "peoresAptitudes.data"))
156   (with-open-file
157     (stream *file* :direction :output
158             :if-exists :supersede
159             :if-does-not-exist :create
160     )
161     (format stream *peores-aptitudes*)
162   )
```

```
163   (setf *file* (concatenate 'string *path* "promedioAptitudes.data"))
164   (with-open-file
165     (stream *file* :direction :output
166             :if-exists :supersede
167             :if-does-not-exist :create
168     )
169     (format stream *promedio-aptitudes*)
170   )
171 )
172
173 ;;---------------------------------------------------------------------
174
175 ;;---------------------------------------------------------------------
176 ;;Función de lectura de las distribuciones observadas
177 ;; f := Lista de Archivos con distribuciones
178 (defun leerDistribuciones (f)
179   (adjust-array *o* (list (length f)))
180   (loop for i in f for j from 1 to (length f) do
181     (setf (aref *o* (1- j)) (read-db i))
182   )
183 )
184
185 ;;---------------------------------------------------------------------
186
187 ;;---------------------------------------------------------------------
188 ;;Funcion de optimización
189 ;; Distribución Esperada
190 (defun funcionObjetivo (E)
191   (/
192     (apply #'+
193       (loop for i from 0 to (1- (array-total-size *o*)) collect
194         (chi-square-pearson
195           (nth 1 (aref *o* i))
196           (loop for j in E collect (* j (apply #'+ (nth 1 (aref *o* i)))))
197         )
198       )
199     )
200     (length E)
201   )
202 )
203
204 ;;---------------------------------------------------------------------
205 ;;Función de generación de problación
206 ;; n := Tamaño de porblación ; c := Cantidad de Hijos por Padre ; f :=
      Archivos de la Base de Datos
207
208 (defun inicializaPoblacion (n c f)
209   (let
210     (
211       (u nil)
212       (d nil)
213     )
214     (leerDistribuciones f)
215     (adjust-array *u* (list n))
216     (adjust-array *h* (list (* n c)))
217     (loop for i from 0 to (1- n) do
218       ;;El individuo es un lista de dos listas, la primera es su cromosoma y
            la segunda su vector de desviación estandar
219       (setf u (loop for j from 1 to (nth 0 (nth 0 (aref *o* 0))) collect (
            random 1.0)))
220       (setf u (loop for j in u collect (/ j (apply #'+ u))))
221       (setf d (loop for j from 1 to (nth 0 (nth 0 (aref *o* 0))) collect (
            random 0.5)))
222       (setf (aref *u* i) (list u d))
223     )
224   )
225 )
226
```

```
227 ;;---------------------------------------------------------------------
228
229 ;;---------------------------------------------------------------------
230 ;;Función para encontrar el mejor individuo
231
232 (defun mejorIndividuo nil
233   (setf *g* 0)
234   (loop for i from 1 to (1- (array-total-size *u*)) do
235     (when
236       (and
237         (compruebaFactor1 (nth 0 (aref *u* *g*)) (nth 0 (aref *u* i)))
238         (compruebaFactor2 (nth 0 (aref *u* *g*)) (nth 0 (aref *u* i)))
239       )
240       (setf *g* i)
241     )
242   )
243 )
244
245 (defun compruebaFactor1 (a b)
246   (let
247     (
248       (r T)
249     )
250     (loop for i from 0 to (1- (array-total-size *o*)) do
251       (unless
252         (<=
253           (chi-square-pearson
254             (nth 1 (aref *o* i))
255             (loop for j in b collect (* j (apply #'+ (nth 1 (aref *o* i)))))
256           )
257           (chi-square-pearson
258             (nth 1 (aref *o* i))
259             (loop for j in a collect (* j (apply #'+ (nth 1 (aref *o* i)))))
260           )
261         )
262         (setf r nil)
263       )
264     )
265     r
266   )
267 )
268
269 (defun compruebaFactor2 (a b)
270   (let
271     (
272       (r nil)
273     )
274     (loop for i from 0 to (1- (array-total-size *o*)) do
275       (when
276         (<
277           (chi-square-pearson
278             (nth 1 (aref *o* i))
279             (loop for j in b collect (* j (apply #'+ (nth 1 (aref *o* i)))))
280           )
281           (chi-square-pearson
282             (nth 1 (aref *o* i))
283             (loop for j in a collect (* j (apply #'+ (nth 1 (aref *o* i)))))
284           )
285         )
286         (setf r T)
287       )
288     )
289     r
290   )
291 )
292 ;;---------------------------------------------------------------------
293
294 ;;---------------------------------------------------------------------
```

```
295 ;;Función para encontrar el peor individuo
296
297 (defun peorIndividuo nil
298   (setf *-g* 0)
299   (loop for i from 1 to (1- (array-total-size *u*)) do
300     (when
301       (and
302         (compruebaFactorA (nth 0 (aref *u* *-g*)) (nth 0 (aref *u* i)))
303         (compruebaFactorB (nth 0 (aref *u* *-g*)) (nth 0 (aref *u* i)))
304       )
305       (setf *-g* i)
306     )
307   )
308 )
309
310 (defun compruebaFactorA (a b)
311   (let
312     (
313       (r T)
314     )
315     (loop for i from 0 to (1- (array-total-size *o*)) do
316       (unless
317         (>=
318           (chi-square-pearson
319             (nth 1 (aref *o* i))
320             (loop for j in b collect (* j (apply #'+ (nth 1 (aref *o* i)))))
321           )
322           (chi-square-pearson
323             (nth 1 (aref *o* i))
324             (loop for j in a collect (* j (apply #'+ (nth 1 (aref *o* i)))))
325           )
326         )
327         (setf r nil)
328       )
329     )
330     r
331   )
332 )
333
334 (defun compruebaFactorB (a b)
335   (let
336     (
337       (r nil)
338     )
339     (loop for i from 0 to (1- (array-total-size *o*)) do
340       (when
341         (>
342           (chi-square-pearson
343             (nth 1 (aref *o* i))
344             (loop for j in b collect (* j (apply #'+ (nth 1 (aref *o* i)))))
345           )
346           (chi-square-pearson
347             (nth 1 (aref *o* i))
348             (loop for j in a collect (* j (apply #'+ (nth 1 (aref *o* i)))))
349           )
350         )
351         (setf r T)
352       )
353     )
354     r
355   )
356 )
357 ;;-----------------------------------------------------------------------
358
359 ;;-----------------------------------------------------------------------
360 ;;Función para generar los hijos
361 ;; c := Cantidad de hijos por padre
362
```

```
(defun mutarPadres (c)
  (let
    (
      (k 0)
      (v nil)
    )
    (loop for i from 0 to (1- (array-total-size *u*)) do
      (loop for j from 1 to c do
        (setf v
          (mapcar
            #'(lambda
              (u d)
              (+ u (/ (exp (* -0.5 (expt (/ u d) 2))) (* d (sqrt (* 2 pi)))))
            )
            (nth 0 (aref *u* i)) (nth 1 (aref *u* i))
          )
        )
        (setf v (loop for j in v collect (/ j (apply #'+ v))))
        (setf v
          (loop for j in v collect
            (if (<= j 0.0)
              (+ j 0.000001)
              j
            )
          )
        )
        (setf (aref *h* k) (list v (nth 1 (aref *u* i))))
        (setf k (1+ k))
      )
    )
  )
)

;;---------------------------------------------------------------------

;;---------------------------------------------------------------------
;;Función para seleccionar la nueva población
;; v := Versión de selección :: #'v

(defun sinSobrevivientes nil
  (let
    (
      (l NIL)
      (r 0)
    )
    (loop for i from 0 to (1- (array-total-size *u*)) do
      (loop for j from 0 to (1- (array-total-size *h*)) do
        (unless (find j l)
          (if
            (<
              (funcionObjetivo (nth 0 (aref *h* j)))
              (funcionObjetivo (nth 0 (aref *h* r)))
            )
            (setf r j)
          )
        )
      )
      (setf l (append l (list r)))
      (setf r 0)
    )
    (loop for i from 0 to (1- (array-total-size *u*)) do
      (setf (aref *u* i) (aref *h* (nth i l)))
    )
  )
)

(defun conSobrevivientes nil
  (let
```

```
431     (
432       (h (make-array (+ (array-total-size *u*) (array-total-size *h*)) :
              element-type 'list
433                                          :initial-element NIL
434                                          :adjustable T
435                                            :fill-pointer 0))
436       (l NIL)
437       (r 0)
438     )
439     (loop for i from 0 to (1- (array-total-size *u*)) do
440       (setf (aref h i) (aref *u* i))
441     )
442     (loop for i from 0 to (1- (array-total-size *h*)) do
443       (setf (aref h (+ i (array-total-size *u*))) (aref *h* i))
444     )
445     (loop for i from 0 to (1- (array-total-size *u*)) do
446       (loop for j from 0 to (1- (array-total-size h)) do
447         (unless (find j l)
448           (if
449             (<
450               (funcionObjetivo (nth 0 (aref h j)))
451               (funcionObjetivo (nth 0 (aref h r)))
452             )
453             (setf r j)
454           )
455         )
456       )
457       (setf l (append l (list r)))
458       (setf r 0)
459     )
460     (loop for i from 0 to (1- (array-total-size *u*)) do
461       (setf (aref *u* i) (aref h (nth i l)))
462     )
463   )
464 )
465
466 (defun nuevaPoblacion (v)
467   (funcall v)
468 )
469
470 ;;---------------------------------------------------------------------
471
472 ;;---------------------------------------------------------------------
473 ;; Función de auto-adaptación de las desviaciones estandar
474 (defun autoAdaptacion nil
475   (loop for i from 0 to (1- (array-total-size *u*)) do
476     (setf
477       (aref *u* i)
478       (list
479         (nth 0 (aref *u* i))
480         (loop for j in (nth 1 (aref *u* i)) collect
481           (*
482             j
483             (exp
484               (*
485                 ((lambda (u) (/ (exp (* -0.5 (expt u 2))) (sqrt (* 2 pi)))) j
                      )
486                 (/ 1 (sqrt (length (nth 0 (aref *u* i)))))
487               )
488             )
489           )
490         )
491       )
492     )
493   )
494 )
495 ;;---------------------------------------------------------------------
496
```

```
;;---------------------------------------------------------------------
;;Algoritmo de Evolución Estrategica
;; n := Tamaño del población ; g := Número de generaciones ; f := Archivo de
    la Base de Datos ; c := Cantidad de hijos por padre ; v := Versión del
    algoritmo :: T para (u+h)-EE y NIL para (u,h)-EE

(defun Estrategia-Evolutiva (f n g c v)
  (crea-experimento n g v c)
  (inicializaPoblacion n c f)
  (guarda-generacion 0)
  (guarda-desviaciones 0)
  (mejorIndividuo)
  (peorIndividuo)
  (guarda-mejores-individuos)
  (guarda-mejores-aptitudes)
  (guarda-peores-aptitudes)
  (guarda-promedio-aptitudes n)
  (loop for i from 1 to g do
    (mutarPadres c)
    (nuevaPoblacion v)
    (autoAdaptacion)
    (guarda-generacion i)
    (guarda-desviaciones i)
    (mejorIndividuo)
    (peorIndividuo)
    (guarda-mejores-individuos)
    (guarda-mejores-aptitudes)
    (guarda-peores-aptitudes)
    (guarda-promedio-aptitudes n)
  )
  (guarda-resultados)
)

;;---------------------------------------------------------------------
```

Código D.13: Algoritmo EE. Prototipo 4

```
;;;;ALgoritmo EE

;;---------------------------------------------------------------------
;;Archivos importados

(load "lecturaBD.lisp")
(load "PCCP.lisp")

;;---------------------------------------------------------------------

;;---------------------------------------------------------------------
;;Variables Globales

(defparameter *o* (make-array 1 :element-type 'list
                :initial-element NIL
                :adjustable T
                :fill-pointer 0))  ; Distribuciones Observadas
(defparameter *u* (make-array 1 :element-type 'list
                :initial-element NIL
                :adjustable T
                :fill-pointer 0))  ;Población
(defparameter *h* (make-array 1 :element-type 'list
                :initial-element NIL
                :adjustable T
                :fill-pointer 0))  ;Hijos generados
(defparameter *g* 0)                ;Mejor individuo
(defparameter *-g* 0)               ;Peor individuo
(defparameter *path* (string "./"))          ;Directorio para Guartar
    Experimento
(defparameter *file* (string "."))           ;Archivo con información el
    experimento
```

```
30 (defparameter *version* (string "()"))          ;Version del algoritmo
31 (defparameter *mejores-individuos* (string ""))    ;Cadena con mejores
       individuos
32 (defparameter *mejores-aptitudes* (string ""))     ; Cadena con mejores
       aptitudes
33 (defparameter *peores-aptitudes* (string ""))    ; Cadena con peores
       aptitudes
34 (defparameter *promedio-aptitudes* (string ""))    ; Cadena con promedio de
       aptitudes
35
36 ;;---------------------------------------------------------------------
37
38 ;;---------------------------------------------------------------------
39 ;;Funciones para crear un experimento de ED y llevar un recuento de las
       variables
40
41 (defun crea-experimento (n g v c a)
42   (setf *mejores-individuos* (string ""))
43   (setf *mejores-aptitudes* (string ""))
44   (setf *peores-aptitudes* (string ""))
45   (setf *promedio-aptitudes* (string ""))
46   (setf *path* (concatenate 'string "./EE/" (remove #\> (subseq (
        write-to-string v) 11)) "/" (write-to-string (get-universal-time)) "/"
        ))
47   (ensure-directories-exist *path*)
48   (ensure-directories-exist (concatenate 'string *path* "Generaciones/
        Individuos/"))
49   (ensure-directories-exist (concatenate 'string *path* "Generaciones/
        Desviaciones/"))
50   (setf *file* (concatenate 'string *path* "DatosExperimento.txt"))
51   (with-open-file
52     (stream *file* :direction :output
53           :if-exists :supersede
54           :if-does-not-exist :create
55     )
56     (if (string= "SINSOBREVIVIENTES" (remove #\> (subseq (write-to-string v)
           11)))
57       (setf *version* (string "(u,h)"))
58     )
59     (if (string= "CONSOBREVIVIENTES" (remove #\> (subseq (write-to-string v)
           11)))
60       (setf *version* (string "(u+h)"))
61     )
62     (format stream
63       (concatenate 'string "Estrategia␣Evolutiva~2%~3TVersión:␣" *version* "-
            EE~%~3TTamaño␣de␣Población:␣" (write-to-string n) "~%~3TNúmero␣de␣
            Generaciones:␣" (write-to-string g) "~%~3THijos␣por␣Padre:␣" (
            write-to-string c) "~2%~3TFunción␣de␣optimización:␣Prueba␣Chi␣
            Cuadrada␣de␣Pearson~%~6TEspacio␣de␣búsqueda:␣[0.0,1.0]~%~6TValor␣
            minimo␣de␣paro:␣" (write-to-string a))
64     )
65   )
66 )
67
68 (defun guarda-generacion (g)
69   (setf *file* (concatenate 'string *path* "Generaciones/Individuos/" "g." (
        write-to-string g) ".data"))
70   (with-open-file
71     (stream *file* :direction :output
72             :if-exists :supersede
73             :if-does-not-exist :create
74     )
75     (loop for i from 0 to (1- (array-total-size *u*)) do
76       (format stream
77         (concatenate 'string (remove #\( (remove #\) (write-to-string (nth 0
              (aref *u* i))))) "~%")
78       )
79     )
```

```
80    )
81  )
82
83  (defun guarda-desviaciones (g)
84    (setf *file* (concatenate 'string *path* "Generaciones/Desviaciones/" "d."
          (write-to-string g) ".data"))
85    (with-open-file
86      (stream *file* :direction :output
87              :if-exists :supersede
88              :if-does-not-exist :create
89      )
90      (loop for i from 0 to (1- (array-total-size *u*)) do
91        (format stream
92          (concatenate 'string (remove #\( (remove #\) (write-to-string (nth 1
                (aref *u* i))))) "~%")
93        )
94      )
95    )
96  )
97
98  (defun guarda-mejores-individuos nil
99    (setf *mejores-individuos* (concatenate 'string *mejores-individuos* (
          remove #\( (remove #\) (write-to-string (nth 0 (aref *u* *g*))))) "~%"
          ))
100 )
101
102 (defun guarda-mejores-aptitudes ()
103   (setf *mejores-aptitudes* (concatenate 'string *mejores-aptitudes* (
          write-to-string (funcionObjetivo (nth 0 (aref *u* *g*)))) "~%"))
104 )
105
106 (defun guarda-peores-aptitudes ()
107   (setf *peores-aptitudes* (concatenate 'string *peores-aptitudes* (
          write-to-string (funcionObjetivo (nth 0 (aref *u* *-g*)))) "~%"))
108 )
109
110 (defun guarda-promedio-aptitudes (n)
111   (setf *promedio-aptitudes*
112     (concatenate 'string *promedio-aptitudes*
113       (write-to-string
114         (/
115           (apply #'+
116             (loop for i from 0 to (1- n) collect
117               (funcionObjetivo (nth 0 (aref *u* i)))
118             )
119           )
120           n
121         )
122       )
123       "~%"
124     )
125   )
126 )
127
128 (defun guarda-resultados nil
129   (setf *file* (concatenate 'string *path* "mejoresIndividuos.data"))
130   (with-open-file
131     (stream *file* :direction :output
132             :if-exists :supersede
133             :if-does-not-exist :create
134     )
135     (format stream *mejores-individuos*)
136   )
137   (setf *file* (concatenate 'string *path* "mejorIndividuo.data"))
138   (with-open-file
139     (stream *file* :direction :output
140             :if-exists :supersede
141             :if-does-not-exist :create
```

```
142     )
143     (format stream
144         (concatenate 'string (remove #\( (remove #\) (write-to-string (nth 0
                (aref *u* *g*))))) "~%")
145     )
146   )
147   (setf *file* (concatenate 'string *path* "mejoresAptitudes.data"))
148   (with-open-file
149     (stream *file* :direction :output
150             :if-exists :supersede
151             :if-does-not-exist :create
152     )
153     (format stream *mejores-aptitudes*)
154   )
155   (setf *file* (concatenate 'string *path* "peoresAptitudes.data"))
156   (with-open-file
157     (stream *file* :direction :output
158             :if-exists :supersede
159             :if-does-not-exist :create
160     )
161     (format stream *peores-aptitudes*)
162   )
163   (setf *file* (concatenate 'string *path* "promedioAptitudes.data"))
164   (with-open-file
165     (stream *file* :direction :output
166             :if-exists :supersede
167             :if-does-not-exist :create
168     )
169     (format stream *promedio-aptitudes*)
170   )
171 )
172 ;;---------------------------------------------------------------------
173
174 ;;---------------------------------------------------------------------
175 ;;Función de lectura de las distribuciones observadas
176 ;; f := Lista de Archivos con distribuciones
177 (defun leerDistribuciones (f)
178   (adjust-array *o* (list (length f)))
179   (loop for i in f for j from 1 to (length f) do
180     (setf (aref *o* (1- j)) (read-db i))
181   )
182 )
183
184 ;;---------------------------------------------------------------------
185
186 ;;---------------------------------------------------------------------
187 ;;Funcion de optimización
188 ;; Distribución Esperada
189 (defun funcionObjetivo (E)
190   (/
191     (apply #'+
192       (loop for i from 0 to (1- (array-total-size *o*)) collect
193         (chi-square-pearson
194           (nth 1 (aref *o* i))
195           (loop for j in E collect (* j (apply #'+ (nth 1 (aref *o* i)))))
196         )
197       )
198     )
199     (length E)
200   )
201 )
202 ;;---------------------------------------------------------------------
203
204 ;;---------------------------------------------------------------------
205 ;;Función de generación de problación
206 ;; n := Tamaño de porblación ; c := Cantidad de Hijos por Padre ; f :=
      Archivos de la Base de Datos
207
```

```
208 (defun inicializaPoblacion (n c f)
209   (let
210     (
211       (u nil)
212       (d nil)
213     )
214     (leerDistribuciones f)
215     (adjust-array *u* (list n))
216     (adjust-array *h* (list (* n c)))
217     (loop for i from 0 to (1- n) do
218       ;El individuo es un lista de dos listas, la primera es su cromosoma y
              la segunda su vector de desviación estandar
219       (setf u (loop for j from 1 to (nth 0 (nth 0 (aref *o* 0))) collect (
              random 1.0)))
220       (setf u (loop for j in u collect (/ j (apply #'+ u))))
221       (setf d (loop for j from 1 to (nth 0 (nth 0 (aref *o* 0))) collect (
              random 1.0)))
222       (setf (aref *u* i) (list u d))
223     )
224   )
225 )
226
227 ;;---------------------------------------------------------------------
228
229 ;;---------------------------------------------------------------------
230 ;;Función para encontrar el mejor individuo
231
232 (defun mejorIndividuo nil
233   (setf *g* 0)
234   (loop for i from 1 to (1- (array-total-size *u*)) do
235     (when
236       (and
237         (compruebaFactor1 (nth 0 (aref *u* *g*)) (nth 0 (aref *u* i)))
238         (compruebaFactor2 (nth 0 (aref *u* *g*)) (nth 0 (aref *u* i)))
239       )
240       (setf *g* i)
241     )
242   )
243 )
244
245 (defun compruebaFactor1 (a b)
246   (let
247     (
248       (r T)
249     )
250     (loop for i from 0 to (1- (array-total-size *o*)) do
251       (unless
252         (<=
253           (chi-square-pearson
254             (nth 1 (aref *o* i))
255             (loop for j in b collect (* j (apply #'+ (nth 1 (aref *o* i)))))
256           )
257           (chi-square-pearson
258             (nth 1 (aref *o* i))
259             (loop for j in a collect (* j (apply #'+ (nth 1 (aref *o* i)))))
260           )
261         )
262         (setf r nil)
263       )
264     )
265     r
266   )
267 )
268
269 (defun compruebaFactor2 (a b)
270   (let
271     (
272       (r nil)
```

```
273     )
274     (loop for i from 0 to (1- (array-total-size *o*)) do
275       (when
276         (<
277           (chi-square-pearson
278             (nth 1 (aref *o* i))
279             (loop for j in b collect (* j (apply #'+ (nth 1 (aref *o* i)))))
280           )
281           (chi-square-pearson
282             (nth 1 (aref *o* i))
283             (loop for j in a collect (* j (apply #'+ (nth 1 (aref *o* i)))))
284           )
285         )
286         (setf r T)
287       )
288     )
289     r
290   )
291 )
292 ;;---------------------------------------------------------------------
293
294 ;;---------------------------------------------------------------------
295 ;;Función para encontrar el peor individuo
296
297 (defun peorIndividuo nil
298   (setf *-g* 0)
299   (loop for i from 1 to (1- (array-total-size *u*)) do
300     (when
301       (and
302         (compruebaFactorA (nth 0 (aref *u* *-g*)) (nth 0 (aref *u* i)))
303         (compruebaFactorB (nth 0 (aref *u* *-g*)) (nth 0 (aref *u* i)))
304       )
305       (setf *-g* i)
306     )
307   )
308 )
309
310 (defun compruebaFactorA (a b)
311   (let
312     (
313       (r T)
314     )
315     (loop for i from 0 to (1- (array-total-size *o*)) do
316       (unless
317         (>=
318           (chi-square-pearson
319             (nth 1 (aref *o* i))
320             (loop for j in b collect (* j (apply #'+ (nth 1 (aref *o* i)))))
321           )
322           (chi-square-pearson
323             (nth 1 (aref *o* i))
324             (loop for j in a collect (* j (apply #'+ (nth 1 (aref *o* i)))))
325           )
326         )
327         (setf r nil)
328       )
329     )
330     r
331   )
332 )
333
334 (defun compruebaFactorB (a b)
335   (let
336     (
337       (r nil)
338     )
339     (loop for i from 0 to (1- (array-total-size *o*)) do
340       (when
```

```
341           (>
342             (chi-square-pearson
343               (nth 1 (aref *o* i))
344               (loop for j in b collect (* j (apply #'+ (nth 1 (aref *o* i)))))
345             )
346             (chi-square-pearson
347               (nth 1 (aref *o* i))
348               (loop for j in a collect (* j (apply #'+ (nth 1 (aref *o* i)))))
349             )
350           )
351           (setf r T)
352         )
353       )
354       r
355     )
356 )
357 ;;---------------------------------------------------------------------
358
359 ;;---------------------------------------------------------------------
360 ;;Función para generar los hijos
361 ;; c := Cantidad de hijos por padre
362
363 (defun mutarPadres (c)
364   (let
365     (
366       (k 0)
367       (v nil)
368     )
369     (loop for i from 0 to (1- (array-total-size *u*)) do
370       (loop for j from 1 to c do
371         (setf v
372           (mapcar
373             #'(lambda
374               (u d)
375               (+ u (/ (exp (* -0.5 (expt (/ u d) 2))) (* d (sqrt (* 2 pi)))))
376             )
377             (nth 0 (aref *u* i)) (nth 1 (aref *u* i))
378           )
379         )
380         (setf v (loop for j in v collect (/ j (apply #'+ v))))
381         (setf v
382           (loop for j in v collect
383             (if (<= j 0.0)
384               (+ j 0.000001)
385               j
386             )
387           )
388         )
389         (setf (aref *h* k) (list v (nth 1 (aref *u* i))))
390         (setf k (1+ k))
391       )
392     )
393   )
394 )
395
396 ;;---------------------------------------------------------------------
397
398 ;;---------------------------------------------------------------------
399 ;;Función para seleccionar la nueva población
400 ;; v := Versión de selección :: #'v
401
402 (defun sinSobrevivientes nil
403   (let
404     (
405       (l NIL)
406       (r 0)
407     )
408     (loop for i from 0 to (1- (array-total-size *u*)) do
```

```
409       (loop for j from 0 to (1- (array-total-size *h*)) do
410         (unless (find j l)
411           (if
412             (<
413               (funcionObjetivo (nth 0 (aref *h* j)))
414               (funcionObjetivo (nth 0 (aref *h* r)))
415             )
416             (setf r j)
417           )
418         )
419       )
420       (setf l (append l (list r)))
421       (setf r 0)
422     )
423     (loop for i from 0 to (1- (array-total-size *u*)) do
424       (setf (aref *u* i) (aref *h* (nth i l)))
425     )
426   )
427 )
428
429 (defun conSobrevivientes nil
430   (let
431     (
432       (h (make-array (+ (array-total-size *u*) (array-total-size *h*)) :
              element-type 'list
433                                         :initial-element NIL
434                                         :adjustable T
435                                           :fill-pointer 0))
436       (l NIL)
437       (r 0)
438     )
439     (loop for i from 0 to (1- (array-total-size *u*)) do
440       (setf (aref h i) (aref *u* i))
441     )
442     (loop for i from 0 to (1- (array-total-size *h*)) do
443       (setf (aref h (+ i (array-total-size *u*))) (aref *h* i))
444     )
445     (loop for i from 0 to (1- (array-total-size *u*)) do
446       (loop for j from 0 to (1- (array-total-size h)) do
447         (unless (find j l)
448           (if
449             (<
450               (funcionObjetivo (nth 0 (aref h j)))
451               (funcionObjetivo (nth 0 (aref h r)))
452             )
453             (setf r j)
454           )
455         )
456       )
457       (setf l (append l (list r)))
458       (setf r 0)
459     )
460     (loop for i from 0 to (1- (array-total-size *u*)) do
461       (setf (aref *u* i) (aref h (nth i l)))
462     )
463   )
464 )
465
466 (defun nuevaPoblacion (v)
467   (funcall v)
468 )
469
470 ;;---------------------------------------------------------------------
471
472 ;;---------------------------------------------------------------------
473 ;; Función de auto-adaptación de las desviaciones estandar
474 (defun autoAdaptacion nil
475   (loop for i from 0 to (1- (array-total-size *u*)) do
```

```
476     (setf
477       (aref *u* i)
478       (list
479         (nth 0 (aref *u* i))
480         (loop for j in (nth 1 (aref *u* i)) collect
481           (*
482             j
483             (exp
484               (*
485                 ((lambda (u) (/ (exp (* -0.5 (expt u 2))) (sqrt (* 2 pi)))) j
                        )
486                 (/ 1 (sqrt (length (nth 0 (aref *u* i)))))
487               )
488             )
489           )
490         )
491       )
492     )
493   )
494 )
495 ;;---------------------------------------------------------------------
496
497 ;;---------------------------------------------------------------------
498 ;;Función para revisar la diversidad en la población
499
500 (defun diversidadPoblacion nil
501   (
502     (lambda (l) (/ (apply #'+ l) (length l)))
503     (apply #'append
504       (loop for j from 0 to (- (array-total-size *u*) 2) collect
505         (loop for k from (1+ j) to (1- (array-total-size *u*)) collect
506           (sqrt
507             (apply #'+
508               (loop for n in (nth 0 (aref *u* j)) for m in (nth 0 (aref *u* k
                      )) collect
509                 (expt (- (* n 100) (* m 100)) 2)
510               )
511             )
512           )
513         )
514       )
515     )
516   )
517 )
518 ;;---------------------------------------------------------------------
519
520 ;;---------------------------------------------------------------------
521 ;;Función de verificación de aptitud aceptable
522 ;; c := Limite de aceptación
523 (defun revisarAptitud (c)
524   (let
525     (
526       (r T)
527     )
528     (loop for i from 0 to (1- (array-total-size *o*)) do
529       (unless
530         (<=
531           (chi-square-pearson
532             (nth 1 (aref *o* i))
533             (loop for j in (nth 0 (aref *u* *g*)) collect (* j (apply #'+ (
                    nth 1 (aref *o* i)))))
534           )
535           (* c (1- (nth 0 (nth 0 (aref *o* i)))))
536         )
537         (setf r nil)
538       )
539     )
540     r
```

```
  )
)
;;-------------------------------------------------------------------

;;-------------------------------------------------------------------
;;Algoritmo de Evolución Estrategica
;; n := Tamaño del población ; g := Número de generaciones ; f := Archivo de
    la Base de Datos ; c := Cantidad de hijos por padre ; v := Versión del
    algoritmo :: T para (u+h)-EE y NIL para (u,h)-EE

(defun Estrategia-Evolutiva (f n g c v a)
  (crea-experimento n g v c a)
  (inicializaPoblacion n c f)
  (guarda-generacion 0)
  (guarda-desviaciones 0)
  (mejorIndividuo)
  (peorIndividuo)
  (guarda-mejores-individuos)
  (guarda-mejores-aptitudes)
  (guarda-peores-aptitudes)
  (guarda-promedio-aptitudes n)
  (loop for i from 1 to g do
    (mutarPadres c)
    (nuevaPoblacion v)
    (autoAdaptacion)
    (guarda-generacion i)
    (guarda-desviaciones i)
    (mejorIndividuo)
    (peorIndividuo)
    (guarda-mejores-individuos)
    (guarda-mejores-aptitudes)
    (guarda-peores-aptitudes)
    (guarda-promedio-aptitudes n)
    ;(print (diversidadPoblacion))
    (if (revisarAptitud a)
      (setf i (1+ g))
    )
    ;(print (funcionObjetivo (nth 0 (aref *u* *g*))))
  )
  (guarda-resultados)
)
;;-------------------------------------------------------------------
(Estrategia-Evolutiva '("prueba.data" "prueba2.data" "prueba3.data") 10 1000
    5 #'conSobrevivientes 0.554)
(Estrategia-Evolutiva '("prueba.data" "prueba2.data" "prueba3.data") 10 1000
    5 #'sinSobrevivientes 0.554)

;(Estrategia-Evolutiva '("prueba.data" "prueba4.data" "prueba5.data") 10 1000
    5 #'conSobrevivientes 0.554)
;(Estrategia-Evolutiva '("prueba.data" "prueba4.data" "prueba5.data") 10 1000
    5 #'sinSobrevivientes 0.554)
```

D.5.3. Evolución Diferencial

Código D.14: Algoritmo ED. Prototipo 1

```
;;;;ALgoritmo ED

;-------------------------------------------------------------------
;;;Archivos importados

(load "funciones.lisp")

;-------------------------------------------------------------------

;-------------------------------------------------------------------
```

```
11 ;;Variables Globales
12
13 (defparameter *p* (make-array 1 :element-type 'list
14                 :initial-element NIL
15                 :adjustable T
16                 :fill-pointer 0))  ; Población
17 (defparameter *h* (make-array 1 :element-type 'list
18                 :initial-element NIL
19                 :adjustable T
20                 :fill-pointer 0))  ; Hijos generados
21 (defparameter *d* (make-array 1 :element-type 'list
22                 :initial-element NIL
23                 :adjustable T
24                 :fill-pointer 0))  ;Parejas de Padres
25 (defparameter *g* 0)                  ; Mejor individuo
26 (defparameter *path* (string "./"))          ;Directorio para Guartar
       Experimento
27 (defparameter *file* (string "."))           ;Archivo con información el
       experimento
28 (defparameter *mejores-individuos* (string ""))    ;Cadena con mejores
       individuos
29
30 ;---------------------------------------------------------------------
31
32 ;---------------------------------------------------------------------
33 ;;Funciones para crear un experimento de ED y llevar un recuento de las
       variables
34
35 (defun crea-experimento (f n g Fp Fc li ls)
36   (setf *mejores-individuos* (string ""))
37   (setf *path* (concatenate 'string "./ED/1/" (remove #\> (subseq (
         write-to-string Fp) 11)) "/" (remove #\> (subseq (write-to-string Fc)
         11)) "/" (write-to-string (get-universal-time)) "/"))
38   (ensure-directories-exist *path*)
39   (ensure-directories-exist (concatenate 'string *path* "Generaciones/"))
40   (ensure-directories-exist (concatenate 'string *path* "Padres/"))
41   (setf *file* (concatenate 'string *path* "DatosExperimento.txt"))
42   (with-open-file
43     (stream *file* :direction :output
44             :if-exists :supersede
45             :if-does-not-exist :create
46     )
47     (format stream
48       (concatenate 'string "Evolución␣Diferencial~2%~3TVersión:␣ED/1/" (
             remove #\> (subseq (write-to-string Fp) 11)) "/" (remove #\> (
             subseq (write-to-string Fc) 11)) "~%~3TTamaño␣de␣Población:␣" (
             write-to-string n) "~%~3TNúmero␣de␣Generaciones:␣" (
             write-to-string g) "~2%~3TFunción␣de␣optimización:␣" (remove #\> (
             subseq (write-to-string f) 11)) "~%~6TEspacio␣de␣búsqueda:␣[" (
             write-to-string li) "," (write-to-string ls) "]")
49     )
50   )
51 )
52
53 (defun guarda-generacion (g)
54   (setf *file* (concatenate 'string *path* "Generaciones/" "g." (
         write-to-string g) ".data"))
55   (with-open-file
56     (stream *file* :direction :output
57             :if-exists :supersede
58             :if-does-not-exist :create
59     )
60     (loop for i from 0 to (1- (array-total-size *p*)) do
61       (format stream
62         (concatenate 'string (write-to-string (nth 0 (aref *p* i))) "␣" (
               write-to-string (nth 1 (aref *p* i))) "~%")
63       )
64     )
```

```
65    )
66  )
67
68  (defun guarda-padres (g)
69    (setf *file* (concatenate 'string *path* "Padres/" "p." (write-to-string g)
          ".data"))
70    (with-open-file
71      (stream *file* :direction :output
72              :if-exists :supersede
73              :if-does-not-exist :create
74      )
75      (loop for i from 0 to (1- (array-total-size *d*)) do
76        (format stream
77          (concatenate 'string (write-to-string (nth 0 (aref *d* i))) "␣" (
                write-to-string (nth 1 (aref *d* i))) "␣" (write-to-string (nth
                2 (aref *d* i))) "~%")
78        )
79      )
80    )
81  )
82
83  (defun guarda-mejores-individuos ()
84    (setf *mejores-individuos* (concatenate 'string *mejores-individuos* (
          write-to-string (nth 0 (aref *p* *g*))) "␣" (write-to-string (nth 1 (
          aref *p* *g*))) "~%"))
85  )
86
87  (defun guarda-resultados ()
88    (setf *file* (concatenate 'string *path* "mejoresIndividuos.data"))
89    (with-open-file
90      (stream *file* :direction :output
91              :if-exists :supersede
92              :if-does-not-exist :create
93      )
94      (format stream *mejores-individuos*)
95    )
96    (setf *file* (concatenate 'string *path* "mejorIndividuo.data"))
97    (with-open-file
98      (stream *file* :direction :output
99              :if-exists :supersede
100             :if-does-not-exist :create
101     )
102     (format stream
103       (concatenate 'string (write-to-string (nth 0 (aref *p* *g*))) "␣" (
              write-to-string (nth 1 (aref *p* *g*))) "~%")
104     )
105   )
106 )
107
108 ;---------------------------------------------------------------------
109
110
111 ;---------------------------------------------------------------------
112 ;;Función de generación de problación
113 ;; n := Tamaño de porblación ; li := Limite inferior ; ls := Limite Superior
114
115 (defun inicializaPoblacion (n li ls)
116   (adjust-array *p* (list n))
117   (loop for i from 0 to (1- n) do
118     (setf (aref *p* i) (list (+ (random (- ls li)) li) (+ (random (- ls li))
          li)))
119   )
120 )
121
122 ;---------------------------------------------------------------------
123
124 ;---------------------------------------------------------------------
125 ;;Función para encontrar el mejor individuo
```

```
126 ;; f := Función objetivo :: #'f
127
128 (defun mejorIndividuo (f)
129   (setf *g* 0)
130   (loop for i from 1 to (1- (array-total-size *p*)) do
131     (if (< (funcall f (nth 0 (aref *p* i)) (nth 1 (aref *p* i))) (funcall f (
          nth 0 (aref *p* *g*)) (nth 1 (aref *p* *g*))))
132       (setf *g* i)
133     )
134   )
135 )
136
137 ;---------------------------------------------------------------------
138
139 ;---------------------------------------------------------------------
140 ;;función para generar los grupos de cruza
141 ;;f := Metodo de selección ya sea Aleatoria o Mejor :: #'f
142
143
144 (defun Aleatorio ()
145   (let
146     (
147       (a 0)
148       (b 0)
149       (c 0)
150     )
151     (loop for i from 0 to (1- (/ (- (array-total-size *p*) (mod (
          array-total-size *p*) 3)) 3)) do
152       (setf a (random (array-total-size *p*)))
153       (setf b (random (array-total-size *p*)))
154       (loop for j from 1 to 1 do
155         (when (= a b)
156           (setf b (random (array-total-size *p*)))
157           (setf j 1)
158         )
159       )
160       (setf c (random (array-total-size *p*)))
161       (loop for j from 1 to 1 do
162         (when (or (= a c) (= b c))
163           (setf c (random (array-total-size *p*)))
164           (setf j 1)
165         )
166       )
167       (setf (aref *d* i) (list a b c))
168     )
169   )
170 )
171
172 (defun Mejor ()
173   (let
174     (
175       (a 0)
176       (b 0)
177     )
178     (loop for i from 0 to (1- (/ (- (array-total-size *p*) (mod (
          array-total-size *p*) 3)) 3)) do
179       (setf a (random (array-total-size *p*)))
180       (loop for j from 1 to 1 do
181         (when (= a *g*)
182           (setf a (random (array-total-size *p*)))
183           (setf j 1)
184         )
185       )
186       (setf b (random (array-total-size *p*)))
187       (loop for j from 1 to 1 do
188         (when (or (= b *g*) (= b a))
189           (setf b (random (array-total-size *p*)))
190           (setf j 1)
```

```
191         )
192       )
193       (setf (aref *d* i) (list *g* a b))
194     )
195   )
196 )
197
198 (defun elegirPadres (f)
199   (adjust-array *d* (list (/ (- (array-total-size *p*) (mod (array-total-size
          *p*) 3)) 3)))
200   (funcall f)
201 )
202
203 ;---------------------------------------------------------------------
204
205 ;---------------------------------------------------------------------
206 ;;Función para la mutación de individuos
207
208 (defun mutacion (li ls)
209   (adjust-array *h* (array-total-size *d*))
210   (loop for i from 0 to (1- (array-total-size *d*)) do
211     (setf (aref *h* i) (mapcar #'(lambda (a b c) (+ a (* (random 2.0) (- b c)
            ))) (aref *p* (nth 0 (aref *d* i))) (aref *p* (nth 1 (aref *d* i)))
            (aref *p* (nth 2 (aref *d* i)))))
212     (setf (aref *h* i)
213       (loop for j in (aref *h* i) collect
214         (if (< j li) (mod j li))
215         (if (> j ls) (mod j ls))
216       )
217     )
218   )
219 )
220
221 ;---------------------------------------------------------------------
222
223 ;---------------------------------------------------------------------
224 ;;Función para la cruza de individuos
225 ;;C := Tasa de recombinación :: f := Función de cruza Binomial(bin) o
        Exponencial(expo) :: #'f
226
227 (defun bin (C lis ls)
228   (loop for i from 0 to (1- (array-total-size *h*)) do
229     (setf (aref *h* i) (mapcar #'(lambda (p h) (if (< (random 1.0) C) h p)) (
            aref *p* (nth 0 (aref *d* i))) (aref *h* i)))
230     (setf (aref *h* i)
231       (loop for j in (aref *h* i) collect
232         (if (< j li) (mod j li))
233         (if (> j ls) (mod j ls))
234       )
235     )
236   )
237 )
238
239 (defun expo (C li ls)
240   (let
241     (
242       (p NIL)
243       (i 0)
244     )
245     (loop for j from 0 to (1- (array-total-size *h*)) do
246       (setf i 0)
247       (setf p NIL)
248       (loop for k from 0 to (1- (length (aref *h* j))) do
249         (if (< (random 1.0) C)
250           (setf i k)
251           (setf k (length (aref *h* j)))
252         )
253       )
```

```
254       (loop for k from 0 to (1- (length (aref *h* j))) do
255         (if (<= k i)
256           (setf p (append p (list (nth k (aref *h* j)))))
257           (setf p (append p (list (nth k (aref *p* (nth 0 (aref *d* j)))))))
258         )
259       )
260       (setf (aref *h* j) p)
261       (setf (aref *h* j)
262         (loop for l in (aref *h* j) collect
263           (if (< l li) (mod l li))
264           (if (> l ls) (mod l ls))
265         )
266       )
267     )
268   )
269 )
270
271 (defun cruza (C f li ls)
272   (funcall f C li ls)
273 )
274 ;-------------------------------------------------------------------------
275
276 ;-------------------------------------------------------------------------
277 ;;Función para la selección de nueva población
278 ;;f := Función objetivo :: #'f
279
280 (defun seleccion (f)
281   (loop for i from 0 to (1- (array-total-size *h*)) do
282     (when (< (funcall f (nth 0 (aref *h* i)) (nth 1 (aref *h* i))) (funcall f
            (nth 0 (aref *p* (nth 0 (aref *d* i)))) (nth 1 (aref *p* (nth 0 (
           aref *d* i))))))
283       (setf (aref *p* (nth 0 (aref *d* i))) (aref *h* i))
284     )
285   )
286 )
287
288 ;-------------------------------------------------------------------------
289
290 ;-------------------------------------------------------------------------
291 ;;Algoritmo de Evolución Diferencial
292 ;;f := Función objetivo ; n := tamaño de población ; g := Generaciones ; Fp
       := Función de selección de padres ; C := Taza de cruza ; Fc := Función
       de Cruza; li := Limite inferior del espacio de búsqueda; ls := Limite
       superior del espacio de búsqueda
293
294 (defun Evolucion-Diferencial (f n g Fp C Fc li ls)
295   (crea-experimento f n g Fp Fc li ls)
296   (inicializaPoblacion n li ls)
297   (guarda-generacion 0)
298   (mejorIndividuo f)
299   (guarda-mejores-individuos)
300   (loop for i from 1 to g do
301     (elegirPadres Fp)
302     (guarda-padres (1- i))
303     (mutacion li ls)
304     (cruza C Fc li ls)
305     (seleccion f)
306     (guarda-generacion i)
307     (mejorIndividuo f)
308     (guarda-mejores-individuos)
309   )
310   (guarda-resultados)
311 )
312
313 ;-------------------------------------------------------------------------
```

Código D.15: Algoritmo ED. Prototipo 2

```
;;;;ALgoritmo ED

;;-------------------------------------------------------------------------
;;;Archivos importados

(load "lecturaBD.lisp")
(load "PCCP.lisp")

;;-------------------------------------------------------------------------

;;-------------------------------------------------------------------------
;;Variables Globales

(defparameter *db* nil)                ;Información de la Base de Datos
(defparameter *p* (make-array 1 :element-type 'list
                  :initial-element NIL
                  :adjustable T
                  :fill-pointer 0))  ;Población
(defparameter *h* (make-array 1 :element-type 'list
                  :initial-element NIL
                  :adjustable T
                  :fill-pointer 0))  ;Hijos generados
(defparameter *d* (make-array 1 :element-type 'list
                  :initial-element NIL
                  :adjustable T
                  :fill-pointer 0))  ;Parejas de Padres
(defparameter *g* 0)                  ;Mejor individuo
(defparameter *-g* 0)                 ;Peor individuo
(defparameter *path* (string "./"))          ;Directorio para Guartar
    Experimento
(defparameter *file* (string "."))           ;Archivo con información el
    experimento
(defparameter *mejores-individuos* (string ""))    ;Cadena con mejores
    individuos
(defparameter *mejores-aptitudes* (string ""))     ; Cadena con mejores
    aptitudes
(defparameter *peores-aptitudes* (string ""))    ; Cadena con peores
    aptitudes
(defparameter *promedio-aptitudes* (string ""))    ; Cadena con promedio de
    aptitudes

;;-------------------------------------------------------------------------

;;-------------------------------------------------------------------------
;;Funciones para crear un experimento de ED y llevar un recuento de las
    variables

(defun crea-experimento (n g Fp Fc)
  (setf *mejores-individuos* (string ""))
  (setf *mejores-aptitudes* (string ""))
  (setf *peores-aptitudes* (string ""))
  (setf *promedio-aptitudes* (string ""))
  (setf *path* (concatenate 'string "./ED/1/" (remove #\> (subseq (
      write-to-string Fp) 11)) "/" (remove #\> (subseq (write-to-string Fc)
      11)) "/" (write-to-string (get-universal-time)) "/"))
  (ensure-directories-exist *path*)
  (ensure-directories-exist (concatenate 'string *path* "Generaciones/"))
  (ensure-directories-exist (concatenate 'string *path* "Padres/"))
  (setf *file* (concatenate 'string *path* "DatosExperimento.txt"))
  (with-open-file
    (stream *file* :direction :output
          :if-exists :supersede
          :if-does-not-exist :create
    )
    (format stream
      (concatenate 'string "Evolución␣Diferencial~2%~3TVersión:␣ED/1/" (
          remove #\> (subseq (write-to-string Fp) 11)) "/" (remove #\> (
```

```
            subseq (write-to-string Fc) 11)) "~%~3TTamaño␣de␣Población:␣" (
            write-to-string n) "~%~3TNúmero␣de␣Generaciones:␣" (
            write-to-string g) "~2%~3TFunción␣de␣optimización:␣Prueba␣Chi␣
            Cuadrada␣de␣Pearson~%~6TEspacio␣de␣búsqueda:␣(0.0,1.0)")
58      )
59    )
60  )
61
62  (defun guarda-generacion (g)
63    (setf *file* (concatenate 'string *path* "Generaciones/" "g." (
          write-to-string g) ".data"))
64    (with-open-file
65      (stream *file* :direction :output
66              :if-exists :supersede
67              :if-does-not-exist :create
68      )
69      (loop for i from 0 to (1- (array-total-size *p*)) do
70        (format stream
71          (concatenate 'string (remove #\( (remove #\) (write-to-string (aref *
              p* i)))) "~%")
72        )
73      )
74    )
75  )
76
77  (defun guarda-padres (g)
78    (setf *file* (concatenate 'string *path* "Padres/" "p." (write-to-string g)
           ".data"))
79    (with-open-file
80      (stream *file* :direction :output
81              :if-exists :supersede
82              :if-does-not-exist :create
83      )
84      (loop for i from 0 to (1- (array-total-size *d*)) do
85        (format stream
86          (concatenate 'string (remove #\( (remove #\) (write-to-string (aref *
              d* i)))) "~%")
87        )
88      )
89    )
90  )
91
92  (defun guarda-mejores-individuos ()
93    (setf *mejores-individuos* (concatenate 'string *mejores-individuos* (
          remove #\( (remove #\) (write-to-string (aref *p* *g*)))) "~%"))
94  )
95
96  (defun guarda-mejores-aptitudes ()
97    (setf *mejores-aptitudes* (concatenate 'string *mejores-aptitudes* (
          write-to-string (chi-square-pearson (nth 1 *db*) (loop for j in (aref
          *p* *g*) collect (* j (apply #'+ (nth 1 *db*)))))) "~%"))
98  )
99
100 (defun guarda-peores-aptitudes ()
101   (setf *peores-aptitudes* (concatenate 'string *peores-aptitudes* (
          write-to-string (chi-square-pearson (nth 1 *db*) (loop for j in (aref
          *p* *-g*) collect (* j (apply #'+ (nth 1 *db*)))))) "~%"))
102 )
103
104 (defun guarda-promedio-aptitudes (n)
105   (setf *promedio-aptitudes*
106     (concatenate 'string *promedio-aptitudes*
107       (write-to-string
108         (/
109           (apply #'+
110             (loop for i from 0 to (1- n) collect
111               (chi-square-pearson (nth 1 *db*) (loop for j in (aref *p* i)
                      collect (* j (apply #'+ (nth 1 *db*))))))
```

```
            )
          )
          n
        )
      )
      "~%"
    )
  )
)

(defun guarda-resultados ()
  (setf *file* (concatenate 'string *path* "mejoresIndividuos.data"))
  (with-open-file
    (stream *file* :direction :output
             :if-exists :supersede
             :if-does-not-exist :create
    )
    (format stream *mejores-individuos*)
  )
  (setf *file* (concatenate 'string *path* "mejorIndividuo.data"))
  (with-open-file
    (stream *file* :direction :output
             :if-exists :supersede
             :if-does-not-exist :create
    )
    (format stream
        (concatenate 'string (remove #\( (remove #\) (write-to-string (aref *
            p* *g*)))) "~%")
    )
  )
  (setf *file* (concatenate 'string *path* "mejoresAptitudes.data"))
  (with-open-file
    (stream *file* :direction :output
             :if-exists :supersede
             :if-does-not-exist :create
    )
    (format stream *mejores-aptitudes*)
  )
  (setf *file* (concatenate 'string *path* "peoresAptitudes.data"))
  (with-open-file
    (stream *file* :direction :output
             :if-exists :supersede
             :if-does-not-exist :create
    )
    (format stream *peores-aptitudes*)
  )
  (setf *file* (concatenate 'string *path* "promedioAptitudes.data"))
  (with-open-file
    (stream *file* :direction :output
             :if-exists :supersede
             :if-does-not-exist :create
    )
    (format stream *promedio-aptitudes*)
  )
)

;;---------------------------------------------------------------------


;;---------------------------------------------------------------------
;;Función de generación de problación
;; n := Tamaño de porblación ; f := Archivo de la Base de Fatos

(defun inicializaPoblacion (n f)
  (adjust-array *p* (list n))
  (setf *db* (read-db f))
  (loop for i from 0 to (1- n) do
    (setf (aref *p* i) (loop for j from 1 to (nth 0 (nth 0 *db*)) collect (
```

```
          random 1.0)))
179     (setf (aref *p* i) (loop for j in (aref *p* i) collect (/ j (apply #'+ (
          aref *p* i)))))
180   )
181 )
182
183 ;;---------------------------------------------------------------------
184
185 ;;---------------------------------------------------------------------
186 ;;Función para encontrar el mejor individuo
187
188 (defun mejorIndividuo nil
189   (setf *g* 0)
190   (loop for i from 1 to (1- (array-total-size *p*)) do
191     (when
192       (<
193         (chi-square-pearson (nth 1 *db*) (loop for j in (aref *p* i) collect
              (* j (apply #'+ (nth 1 *db*)))))
194         (chi-square-pearson (nth 1 *db*) (loop for j in (aref *p* *g*)
              collect (* j (apply #'+ (nth 1 *db*)))))
195       )
196       (setf *g* i)
197     )
198   )
199 )
200
201 ;;---------------------------------------------------------------------
202
203 ;;---------------------------------------------------------------------
204 ;;Función para encontrar el peor individuo
205
206 (defun peorIndividuo nil
207   (setf *-g* 0)
208   (loop for i from 1 to (1- (array-total-size *p*)) do
209     (when
210       (>
211         (chi-square-pearson (nth 1 *db*) (loop for j in (aref *p* i) collect
              (* j (apply #'+ (nth 1 *db*)))))
212         (chi-square-pearson (nth 1 *db*) (loop for j in (aref *p* *-g*)
              collect (* j (apply #'+ (nth 1 *db*)))))
213       )
214       (setf *-g* i)
215     )
216   )
217 )
218
219 ;;---------------------------------------------------------------------
220
221 ;;---------------------------------------------------------------------
222 ;;función para generar los grupos de cruza
223 ;;f := Metodo de selección ya sea Aleatoria o Mejor :: #'f
224
225
226 (defun Aleatorio nil
227   (let
228     (
229       (a 0)
230       (b 0)
231       (c 0)
232     )
233     (loop for i from 0 to (1- (/ (- (array-total-size *p*) (mod (
          array-total-size *p*) 3)) 3)) do
234       (setf a (random (array-total-size *p*)))
235       (setf b (random (array-total-size *p*)))
236       (loop for j from 1 to 1 do
237         (when (= a b)
238           (setf b (random (array-total-size *p*)))
239           (setf j 1)
```

```
240         )
241       )
242       (setf c (random (array-total-size *p*)))
243       (loop for j from 1 to 1 do
244         (when (or (= a c) (= b c))
245           (setf c (random (array-total-size *p*)))
246           (setf j 1)
247         )
248       )
249       (setf (aref *d* i) (list a b c))
250     )
251   )
252 )
253
254 (defun Mejor nil
255   (let
256     (
257       (a 0)
258       (b 0)
259     )
260     (loop for i from 0 to (1- (/ (- (array-total-size *p*) (mod (
          array-total-size *p*) 3)) 3)) do
261       (setf a (random (array-total-size *p*)))
262       (loop for j from 1 to 1 do
263         (when (= a *g*)
264           (setf a (random (array-total-size *p*)))
265           (setf j 1)
266         )
267       )
268       (setf b (random (array-total-size *p*)))
269       (loop for j from 1 to 1 do
270         (when (or (= b *g*) (= b a))
271           (setf b (random (array-total-size *p*)))
272           (setf j 1)
273         )
274       )
275       (setf (aref *d* i) (list *g* a b))
276     )
277   )
278 )
279
280 (defun elegirPadres (f)
281   (adjust-array *d* (list (/ (- (array-total-size *p*) (mod (array-total-size
        *p*) 3)) 3)))
282   (funcall f)
283 )
284
285 ;;-------------------------------------------------------------------------
286
287 ;;-------------------------------------------------------------------------
288 ;;Función para la mutación de individuos
289
290 (defun mutacion nil
291   (adjust-array *h* (array-total-size *d*))
292   (loop for i from 0 to (1- (array-total-size *d*)) do
293     (setf (aref *h* i) (mapcar #'(lambda (a b c) (mod (+ a (* (random 2.0) (-
            b c))) 1.0)) (aref *p* (nth 0 (aref *d* i))) (aref *p* (nth 1 (aref
           *d* i))) (aref *p* (nth 2 (aref *d* i)))))
294     (setf (aref *h* i)
295       (loop for j in (aref *h* i) collect
296         (/ j (apply #'+ (aref *h* i)))
297       )
298     )
299   )
300 )
301
302 ;;-------------------------------------------------------------------------
303
```

```
304 ;;---------------------------------------------------------------------
305 ;;Función para la cruza de individuos
306 ;;C := Tasa de recombinación :: f := Función de cruza Binomial(bin) o
        Exponencial(expo) :: #'f
307
308 (defun bin (C)
309   (loop for i from 0 to (1- (array-total-size *h*)) do
310     (setf (aref *h* i) (mapcar #'(lambda (p h) (if (< (random 1.0) C) h p)) (
          aref *p* (nth 0 (aref *d* i))) (aref *h* i)))
311     (setf (aref *h* i)
312       (loop for j in (aref *h* i) collect
313         (/ j (apply #'+ (aref *h* i)))
314       )
315     )
316   )
317 )
318
319 (defun expo (C)
320   (let
321     (
322       (p NIL)
323       (i 0)
324     )
325     (loop for j from 0 to (1- (array-total-size *h*)) do
326       (setf i 0)
327       (setf p NIL)
328       (loop for k from 0 to (1- (length (aref *h* j))) do
329         (if (< (random 1.0) C)
330           (setf i k)
331           (setf k (length (aref *h* j)))
332         )
333       )
334       (loop for k from 0 to (1- (length (aref *h* j))) do
335         (if (<= k i)
336           (setf p (append p (list (nth k (aref *h* j)))))
337           (setf p (append p (list (nth k (aref *p* (nth 0 (aref *d* j)))))))
338         )
339       )
340       (setf (aref *h* j) p)
341       (setf (aref *h* j)
342         (loop for k in (aref *h* j) collect
343           (/ k (apply #'+ (aref *h* j)))
344         )
345       )
346     )
347   )
348 )
349
350 (defun cruza (C f)
351   (funcall f C)
352 )
353 ;;---------------------------------------------------------------------
354
355 ;;---------------------------------------------------------------------
356 ;;Función para la selección de nueva población
357
358 (defun seleccion nil
359   (loop for i from 0 to (1- (array-total-size *h*)) do
360     (setf (aref *h* i) (loop for j in (aref *h* i) collect (/ j (apply #'+ (
          aref *h* i)))))
361     (when
362       (<
363         (chi-square-pearson (nth 1 *db*) (loop for j in (aref *h* i) collect
              (* j (apply #'+ (nth 1 *db*)))))
364         (chi-square-pearson (nth 1 *db*) (loop for j in (aref *p* (nth 0 (
              aref *d* i))) collect (* j (apply #'+ (nth 1 *db*)))))
365       )
366       (setf (aref *p* (nth 0 (aref *d* i))) (aref *h* i))
```

```
    )
  )
)

;;-------------------------------------------------------------------------

;;-------------------------------------------------------------------------
;;Algoritmo de Evolución Diferencial
;; n := tamaño de población ; g := Generaciones ; Fp := Función de selección
     de padres ; C := Taza de cruza ; Fc := Función de Cruza ; f := Archivo
     de la Base de Datos

(defun Evolucion-Diferencial (n g Fp C Fc f)
  (crea-experimento n g Fp Fc)
  (inicializaPoblacion n f)
  (guarda-generacion 0)
  (mejorIndividuo)
  (peorIndividuo)
  (guarda-mejores-individuos)
  (guarda-mejores-aptitudes)
  (guarda-peores-aptitudes)
  (guarda-promedio-aptitudes n)
  (loop for i from 1 to g do
    (elegirPadres Fp)
    (guarda-padres (1- i))
    (mutacion)
    (cruza C Fc)
    (seleccion)
    (guarda-generacion i)
    (mejorIndividuo)
    (peorIndividuo)
    (guarda-mejores-individuos)
    (guarda-mejores-aptitudes)
    (guarda-peores-aptitudes)
    (guarda-promedio-aptitudes n)
  )
  (guarda-resultados)
)

;;-------------------------------------------------------------------------
```

Código D.16: Algoritmo ED. Prototipo 3

```
;;;;ALgoritmo ED

;;-------------------------------------------------------------------------
;;;Archivos importados

(load "lecturaBD.lisp")
(load "PCCP.lisp")

;;-------------------------------------------------------------------------

;;-------------------------------------------------------------------------
;;;Variables Globales

(defparameter *o* (make-array 1 :element-type 'list
                  :initial-element NIL
                  :adjustable T
                  :fill-pointer 0))  ; Distribuciones Observadas
(defparameter *p* (make-array 1 :element-type 'list
                  :initial-element NIL
                  :adjustable T
                  :fill-pointer 0))  ;Población
(defparameter *h* (make-array 1 :element-type 'list
                  :initial-element NIL
                  :adjustable T
                  :fill-pointer 0))  ;Hijos generados
```

```
26 (defparameter *d* (make-array 1 :element-type 'list
27                 :initial-element NIL
28                 :adjustable T
29                 :fill-pointer 0))  ;Parejas de Padres
30 (defparameter *g* 0)                  ;Mejor individuo
31 (defparameter *-g* 0)                 ;Peor individuo
32 (defparameter *path* (string "./"))          ;Directorio para Guartar
       Experimento
33 (defparameter *file* (string "."))           ;Archivo con información el
       experimento
34 (defparameter *mejores-individuos* (string ""))    ;Cadena con mejores
       individuos
35 (defparameter *mejores-aptitudes* (string ""))     ; Cadena con mejores
       aptitudes
36 (defparameter *peores-aptitudes* (string ""))    ; Cadena con peores
       aptitudes
37 (defparameter *promedio-aptitudes* (string ""))    ; Cadena con promedio de
       aptitudes
38
39 ;;---------------------------------------------------------------------
40
41 ;;---------------------------------------------------------------------
42 ;;Funciones para crear un experimento de ED y llevar un recuento de las
       variables
43
44 (defun crea-experimento (n g Fp Fc)
45   (setf *mejores-individuos* (string ""))
46   (setf *mejores-aptitudes* (string ""))
47   (setf *peores-aptitudes* (string ""))
48   (setf *promedio-aptitudes* (string ""))
49   (setf *path* (concatenate 'string "./ED/1/" (remove #\> (subseq (
         write-to-string Fp) 11)) "/" (remove #\> (subseq (write-to-string Fc)
         11)) "/" (write-to-string (get-universal-time)) "/"))
50   (ensure-directories-exist *path*)
51   (ensure-directories-exist (concatenate 'string *path* "Generaciones/"))
52   (ensure-directories-exist (concatenate 'string *path* "Padres/"))
53   (setf *file* (concatenate 'string *path* "DatosExperimento.txt"))
54   (with-open-file
55     (stream *file* :direction :output
56             :if-exists :supersede
57             :if-does-not-exist :create
58     )
59     (format stream
60       (concatenate 'string "Evolución␣Diferencial~2%~3TVersión:␣ED/1/" (
             remove #\> (subseq (write-to-string Fp) 11)) "/" (remove #\> (
             subseq (write-to-string Fc) 11)) "~%~3TTamaño␣de␣Población:␣" (
             write-to-string n) "~%~3TNúmero␣de␣Generaciones:␣" (
             write-to-string g) "~2%~3TFunción␣de␣optimización:␣Prueba␣Chi␣
             Cuadrada␣de␣Pearson~%~6TEspacio␣de␣búsqueda:␣[0.0,1.0]")
61     )
62   )
63 )
64
65 (defun guarda-generacion (g)
66   (setf *file* (concatenate 'string *path* "Generaciones/" "g." (
         write-to-string g) ".data"))
67   (with-open-file
68     (stream *file* :direction :output
69             :if-exists :supersede
70             :if-does-not-exist :create
71     )
72     (loop for i from 0 to (1- (array-total-size *p*)) do
73       (format stream
74         (concatenate 'string (remove #\( (remove #\) (write-to-string (aref *
               p* i)))) "~%")
75       )
76     )
77   )
```

```
78 )
79
80 (defun guarda-padres (g)
81   (setf *file* (concatenate 'string *path* "Padres/" "p." (write-to-string g)
         ".data"))
82   (with-open-file
83     (stream *file* :direction :output
84             :if-exists :supersede
85             :if-does-not-exist :create
86     )
87     (loop for i from 0 to (1- (array-total-size *d*)) do
88       (format stream
89         (concatenate 'string (remove #\( (remove #\) (write-to-string (aref *
              d* i)))) "~%")
90       )
91     )
92   )
93 )
94
95 (defun guarda-mejores-individuos ()
96   (setf *mejores-individuos* (concatenate 'string *mejores-individuos* (
         remove #\( (remove #\) (write-to-string (aref *p* *g*)))) "~%"))
97 )
98
99 (defun guarda-mejores-aptitudes ()
100   (setf *mejores-aptitudes* (concatenate 'string *mejores-aptitudes* (
         write-to-string (funcionObjetivo (aref *p* *g*))) "~%"))
101 )
102
103 (defun guarda-peores-aptitudes ()
104   (setf *peores-aptitudes* (concatenate 'string *peores-aptitudes* (
         write-to-string (funcionObjetivo (aref *p* *-g*))) "~%"))
105 )
106
107 (defun guarda-promedio-aptitudes (n)
108   (setf *promedio-aptitudes*
109     (concatenate 'string *promedio-aptitudes*
110       (write-to-string
111         (/
112           (apply #'+
113             (loop for i from 0 to (1- n) collect
114               (funcionObjetivo (aref *p* i))
115             )
116           )
117           n
118         )
119       )
120       "~%"
121     )
122   )
123 )
124
125 (defun guarda-resultados ()
126   (setf *file* (concatenate 'string *path* "mejoresIndividuos.data"))
127   (with-open-file
128     (stream *file* :direction :output
129             :if-exists :supersede
130             :if-does-not-exist :create
131     )
132     (format stream *mejores-individuos*)
133   )
134   (setf *file* (concatenate 'string *path* "mejorIndividuo.data"))
135   (with-open-file
136     (stream *file* :direction :output
137             :if-exists :supersede
138             :if-does-not-exist :create
139     )
140     (format stream
```

```
141         (concatenate 'string (remove #\( (remove #\) (write-to-string (aref *
                p* *g*)))) "~%")
142     )
143   )
144   (setf *file* (concatenate 'string *path* "mejoresAptitudes.data"))
145   (with-open-file
146     (stream *file* :direction :output
147             :if-exists :supersede
148             :if-does-not-exist :create
149     )
150     (format stream *mejores-aptitudes*)
151   )
152   (setf *file* (concatenate 'string *path* "peoresAptitudes.data"))
153   (with-open-file
154     (stream *file* :direction :output
155             :if-exists :supersede
156             :if-does-not-exist :create
157     )
158     (format stream *peores-aptitudes*)
159   )
160   (setf *file* (concatenate 'string *path* "promedioAptitudes.data"))
161   (with-open-file
162     (stream *file* :direction :output
163             :if-exists :supersede
164             :if-does-not-exist :create
165     )
166     (format stream *promedio-aptitudes*)
167   )
168 )
169
170 ;;---------------------------------------------------------------------
171
172 ;;---------------------------------------------------------------------
173 ;;Función de lectura de las distribuciones observadas
174 ;; f := Lista de Archivos con distribuciones
175 (defun leerDistribuciones (f)
176   (adjust-array *o* (list (length f)))
177   (loop for i in f for j from 1 to (length f) do
178     (setf (aref *o* (1- j)) (read-db i))
179   )
180 )
181
182 ;;---------------------------------------------------------------------
183
184 ;;---------------------------------------------------------------------
185 ;;Funcion de optimización
186 ;; Distribución Esperada
187 (defun funcionObjetivo (E)
188   (/
189     (apply #'+
190       (loop for i from 0 to (1- (array-total-size *o*)) collect
191         (chi-square-pearson
192           (nth 1 (aref *o* i))
193           (loop for j in E collect (* j (apply #'+ (nth 1 (aref *o* i)))))
194         )
195       )
196     )
197     (length E)
198   )
199 )
200
201 ;---------------------------------------------------------------------
202 ;Función de generación de problación
203 ; n := Tamaño de porblación ; f := Archivos de las distribuciones observadas
204
205 (defun inicializaPoblacion (n f)
206   (leerDistribuciones f)
207   (adjust-array *p* (list n))
```

```
208   (loop for i from 0 to (1- n) do
209     (setf (aref *p* i) (loop for j from 1 to (nth 0 (nth 0 (aref *o* 0)))
          collect (random 1.0)))
210     (setf (aref *p* i) (loop for j in (aref *p* i) collect (/ j (apply #'+ (
          aref *p* i)))))
211   )
212 )
213
214 ;---------------------------------------------------------------------
215
216 ;---------------------------------------------------------------------
217 ;Función para encontrar el mejor individuo
218
219 (defun mejorIndividuo nil
220   (setf *g* 0)
221   (loop for i from 1 to (1- (array-total-size *p*)) do
222     (when
223       (and
224         (compruebaFactor1 (aref *p* *g*) (aref *p* i))
225         (compruebaFactor2 (aref *p* *g*) (aref *p* i))
226       )
227       (setf *g* i)
228     )
229   )
230 )
231
232 (defun compruebaFactor1 (a b)
233   (let
234     (
235       (r T)
236     )
237     (loop for i from 0 to (1- (array-total-size *o*)) do
238       (unless
239         (<=
240           (chi-square-pearson
241             (nth 1 (aref *o* i))
242             (loop for j in b collect (* j (apply #'+ (nth 1 (aref *o* i)))))
243           )
244           (chi-square-pearson
245             (nth 1 (aref *o* i))
246             (loop for j in a collect (* j (apply #'+ (nth 1 (aref *o* i)))))
247           )
248         )
249         (setf r nil)
250       )
251     )
252     r
253   )
254 )
255
256 (defun compruebaFactor2 (a b)
257   (let
258     (
259       (r nil)
260     )
261     (loop for i from 0 to (1- (array-total-size *o*)) do
262       (when
263         (<
264           (chi-square-pearson
265             (nth 1 (aref *o* i))
266             (loop for j in b collect (* j (apply #'+ (nth 1 (aref *o* i)))))
267           )
268           (chi-square-pearson
269             (nth 1 (aref *o* i))
270             (loop for j in a collect (* j (apply #'+ (nth 1 (aref *o* i)))))
271           )
272         )
273         (setf r T)
```

```
274       )
275     )
276     r
277   )
278 )
279 ;;---------------------------------------------------------------------
280
281 ;;---------------------------------------------------------------------
282 ;;Función para encontrar el peor individuo
283
284 (defun peorIndividuo nil
285   (setf *-g* 0)
286   (loop for i from 1 to (1- (array-total-size *p*)) do
287     (when
288       (and
289         (compruebaFactorA (aref *p* *-g*) (aref *p* i))
290         (compruebaFactorB (aref *p* *-g*) (aref *p* i))
291       )
292       (setf *-g* i)
293     )
294   )
295 )
296
297 (defun compruebaFactorA (a b)
298   (let
299     (
300       (r T)
301     )
302     (loop for i from 0 to (1- (array-total-size *o*)) do
303       (unless
304         (>=
305           (chi-square-pearson
306             (nth 1 (aref *o* i))
307             (loop for j in b collect (* j (apply #'+ (nth 1 (aref *o* i)))))
308           )
309           (chi-square-pearson
310             (nth 1 (aref *o* i))
311             (loop for j in a collect (* j (apply #'+ (nth 1 (aref *o* i)))))
312           )
313         )
314         (setf r nil)
315       )
316     )
317     r
318   )
319 )
320
321 (defun compruebaFactorB (a b)
322   (let
323     (
324       (r nil)
325     )
326     (loop for i from 0 to (1- (array-total-size *o*)) do
327       (when
328         (>
329           (chi-square-pearson
330             (nth 1 (aref *o* i))
331             (loop for j in b collect (* j (apply #'+ (nth 1 (aref *o* i)))))
332           )
333           (chi-square-pearson
334             (nth 1 (aref *o* i))
335             (loop for j in a collect (* j (apply #'+ (nth 1 (aref *o* i)))))
336           )
337         )
338         (setf r T)
339       )
340     )
341     r
```

```
342   )
343 )
344 ;;-------------------------------------------------------------------
345
346 ;;-------------------------------------------------------------------
347 ;;función para generar los grupos de cruza
348 ;;f := Metodo de selección ya sea Aleatoria o Mejor :: #'f
349
350
351 (defun Aleatorio nil
352   (let
353     (
354       (a 0)
355       (b 0)
356       (c 0)
357     )
358     (loop for i from 0 to (1- (/ (- (array-total-size *p*) (mod (
          array-total-size *p*) 3)) 3)) do
359       (setf a (random (array-total-size *p*)))
360       (setf b (random (array-total-size *p*)))
361       (loop for j from 1 to 1 do
362         (when (= a b)
363           (setf b (random (array-total-size *p*)))
364           (setf j 1)
365         )
366       )
367       (setf c (random (array-total-size *p*)))
368       (loop for j from 1 to 1 do
369         (when (or (= a c) (= b c))
370           (setf c (random (array-total-size *p*)))
371           (setf j 1)
372         )
373       )
374       (setf (aref *d* i) (list a b c))
375     )
376   )
377 )
378
379 (defun Mejor nil
380   (let
381     (
382       (a 0)
383       (b 0)
384     )
385     (loop for i from 0 to (1- (/ (- (array-total-size *p*) (mod (
          array-total-size *p*) 3)) 3)) do
386       (setf a (random (array-total-size *p*)))
387       (loop for j from 1 to 1 do
388         (when (= a *g*)
389           (setf a (random (array-total-size *p*)))
390           (setf j 1)
391         )
392       )
393       (setf b (random (array-total-size *p*)))
394       (loop for j from 1 to 1 do
395         (when (or (= b *g*) (= b a))
396           (setf b (random (array-total-size *p*)))
397           (setf j 1)
398         )
399       )
400       (setf (aref *d* i) (list *g* a b))
401     )
402   )
403 )
404
405 (defun elegirPadres (f)
406   (adjust-array *d* (list (/ (- (array-total-size *p*) (mod (array-total-size
        *p*) 3)) 3)))
```

```
407   (funcall f)
408 )
409
410 ;;---------------------------------------------------------------------
411
412 ;;---------------------------------------------------------------------
413 ;;Función para la mutación de individuos
414
415 (defun mutacion nil
416   (adjust-array *h* (array-total-size *d*))
417   (loop for i from 0 to (1- (array-total-size *d*)) do
418     (setf (aref *h* i) (mapcar #'(lambda (a b c) (mod (+ a (* (random 2.0) (-
            b c))) 1.0)) (aref *p* (nth 0 (aref *d* i))) (aref *p* (nth 1 (aref
            *d* i))) (aref *p* (nth 2 (aref *d* i)))))
419     (setf (aref *h* i)
420       (loop for j in (aref *h* i) collect
421         (/ j (apply #'+ (aref *h* i)))
422       )
423     )
424     (setf (aref *h* i)
425       (loop for j in (aref *h* i) collect
426         (if (<= j 0.0)
427           (+ j 0.000001)
428           j
429         )
430       )
431     )
432   )
433 )
434
435 ;;---------------------------------------------------------------------
436
437 ;;---------------------------------------------------------------------
438 ;;Función para la cruza de individuos
439 ;;C := Tasa de recombinación :: f := Función de cruza Binomial(bin) o
      Exponencial(expo) :: #'f
440
441 (defun bin (C)
442   (loop for i from 0 to (1- (array-total-size *h*)) do
443     (setf (aref *h* i) (mapcar #'(lambda (p h) (if (< (random 1.0) C) h p)) (
          aref *p* (nth 0 (aref *d* i))) (aref *h* i)))
444     (setf (aref *h* i)
445       (loop for j in (aref *h* i) collect
446         (/ j (apply #'+ (aref *h* i)))
447       )
448     )
449     (setf (aref *h* i)
450       (loop for j in (aref *h* i) collect
451         (if (<= j 0.0)
452           (+ j 0.000001)
453           j
454         )
455       )
456     )
457   )
458 )
459
460 (defun expo (C)
461   (let
462     (
463       (p NIL)
464       (i 0)
465     )
466     (loop for j from 0 to (1- (array-total-size *h*)) do
467       (setf i 0)
468       (setf p NIL)
469       (loop for k from 0 to (1- (length (aref *h* j))) do
470         (if (< (random 1.0) C)
```

```
471             (setf i k)
472             (setf k (length (aref *h* j)))
473           )
474         )
475         (loop for k from 0 to (1- (length (aref *h* j))) do
476           (if (<= k i)
477             (setf p (append p (list (nth k (aref *h* j)))))
478             (setf p (append p (list (nth k (aref *p* (nth 0 (aref *d* j)))))))
479           )
480         )
481         (setf (aref *h* j) p)
482         (setf (aref *h* j)
483           (loop for l in (aref *h* j) collect
484             (/ l (apply #'+ (aref *h* j)))
485           )
486         )
487         (setf (aref *h* i)
488           (loop for j in (aref *h* i) collect
489             (if (<= j 0.0)
490               (+ j 0.000001)
491               j
492             )
493           )
494         )
495       )
496     )
497 )
498
499 (defun cruza (C f)
500   (funcall f C)
501 )
502 ;;-------------------------------------------------------------------------
503
504 ;;-------------------------------------------------------------------------
505 ;;Función para la selección de nueva población
506
507 (defun seleccion nil
508   (loop for i from 0 to (1- (array-total-size *h*)) do
509     (setf (aref *h* i) (loop for j in (aref *h* i) collect (/ j (apply #'+ (
          aref *h* i)))))
510     (when
511       (<
512         (funcionObjetivo (aref *h* i))
513         (funcionObjetivo (aref *p* (nth 0 (aref *d* i))))
514       )
515       (setf (aref *p* (nth 0 (aref *d* i))) (aref *h* i))
516     )
517   )
518 )
519
520 ;;-------------------------------------------------------------------------
521
522 ;;-------------------------------------------------------------------------
523 ;;Algoritmo de Evolución Diferencial
524 ;; n := tamaño de población ; g := Generaciones ; Fp := Función de selección
       de padres ; C := Taza de cruza ; Fc := Función de Cruza ; f := Archivo
       de la Base de Datos
525
526 (defun Evolucion-Diferencial (n g Fp C Fc f)
527   (crea-experimento n g Fp Fc)
528   (inicializaPoblacion n f)
529   (guarda-generacion 0)
530   (mejorIndividuo)
531   (peorIndividuo)
532   (guarda-mejores-individuos)
533   (guarda-mejores-aptitudes)
534   (guarda-peores-aptitudes)
535   (guarda-promedio-aptitudes n)
```

```
536   (loop for i from 1 to g do
537     (elegirPadres Fp)
538     (guarda-padres (1- i))
539     (mutacion)
540     (cruza C Fc)
541     (seleccion)
542     (guarda-generacion i)
543     (mejorIndividuo)
544     (peorIndividuo)
545     (guarda-mejores-individuos)
546     (guarda-mejores-aptitudes)
547     (guarda-peores-aptitudes)
548     (guarda-promedio-aptitudes n)
549   )
550   (guarda-resultados)
551 )
552
553 ;;---------------------------------------------------------------------
```

Código D.17: Algoritmo ED. Prototipo 4

```
1  ;;;;ALgoritmo ED
2
3  ;;---------------------------------------------------------------------
4  ;;Archivos importados
5
6  (load "lecturaBD.lisp")
7  (load "PCCP.lisp")
8  ;;---------------------------------------------------------------------
9
10 ;;---------------------------------------------------------------------
11 ;;Variables Globales
12
13 (defparameter *o* (make-array 1 :element-type 'list
14                  :initial-element NIL
15                  :adjustable T
16                  :fill-pointer 0))  ;Distribuciones Observadas
17 (defparameter *p* (make-array 1 :element-type 'list
18                  :initial-element NIL
19                  :adjustable T
20                  :fill-pointer 0))  ;Población
21 (defparameter *h* (make-array 1 :element-type 'list
22                  :initial-element NIL
23                  :adjustable T
24                  :fill-pointer 0))  ;Hijos generados
25 (defparameter *d* (make-array 1 :element-type 'list
26                  :initial-element NIL
27                  :adjustable T
28                  :fill-pointer 0))  ;Parejas de Padres
29 (defparameter *g* 0)                  ;Mejor individuo
30 (defparameter *-g* 0)                 ;Peor individuo
31 (defparameter *path* (string "./"))          ;Directorio para Guartar
       Experimento
32 (defparameter *file* (string "."))           ;Archivo con información el
       experimento
33 (defparameter *mejores-individuos* (string ""))    ;Cadena con mejores
       individuos
34 (defparameter *mejores-aptitudes* (string ""))     ; Cadena con mejores
       aptitudes
35 (defparameter *peores-aptitudes* (string ""))    ; Cadena con peores
       aptitudes
36 (defparameter *promedio-aptitudes* (string ""))    ; Cadena con promedio de
       aptitudes
37 ;;---------------------------------------------------------------------
38
39 ;;---------------------------------------------------------------------
40 ;;Funciones para crear un experimento de ED y llevar un recuento de las
       variables
```

```
41
42 (defun crea-experimento (n g Fp Fc a)
43   (setf *mejores-individuos* (string ""))
44   (setf *mejores-aptitudes* (string ""))
45   (setf *peores-aptitudes* (string ""))
46   (setf *promedio-aptitudes* (string ""))
47   (setf *path* (concatenate 'string "./ED/1/" (remove #\> (subseq (
         write-to-string Fp) 11)) "/" (remove #\> (subseq (write-to-string Fc)
         11)) "/" (write-to-string (get-universal-time)) "/"))
48   (ensure-directories-exist *path*)
49   (ensure-directories-exist (concatenate 'string *path* "Generaciones/"))
50   (ensure-directories-exist (concatenate 'string *path* "Padres/"))
51   (setf *file* (concatenate 'string *path* "DatosExperimento.txt"))
52   (with-open-file
53     (stream *file* :direction :output
54             :if-exists :supersede
55             :if-does-not-exist :create
56     )
57     (format stream
58       (concatenate 'string "Evolución␣Diferencial~2%~3TVersión:␣ED/1/" (
             remove #\> (subseq (write-to-string Fp) 11)) "/" (remove #\> (
             subseq (write-to-string Fc) 11)) "~%~3TTamaño␣de␣Población:␣" (
             write-to-string n) "~%~3TNúmero␣de␣Generaciones:␣" (
             write-to-string g) "~2%~3TFunción␣de␣optimización:␣Prueba␣Chi␣
             Cuadrada␣de␣Pearson~%~6TEspacio␣de␣búsqueda:␣[0.0,1.0]~%~6TValor␣
             minimo␣de␣paro:␣" (write-to-string a))
59     )
60   )
61 )
62
63 (defun guarda-generacion (g)
64   (setf *file* (concatenate 'string *path* "Generaciones/" "g." (
         write-to-string g) ".data"))
65   (with-open-file
66     (stream *file* :direction :output
67             :if-exists :supersede
68             :if-does-not-exist :create
69     )
70     (loop for i from 0 to (1- (array-total-size *p*)) do
71       (format stream
72         (concatenate 'string (remove #\( (remove #\) (write-to-string (aref *
               p* i)))) "~%")
73       )
74     )
75   )
76 )
77
78 (defun guarda-padres (g)
79   (setf *file* (concatenate 'string *path* "Padres/" "p." (write-to-string g)
         ".data"))
80   (with-open-file
81     (stream *file* :direction :output
82             :if-exists :supersede
83             :if-does-not-exist :create
84     )
85     (loop for i from 0 to (1- (array-total-size *d*)) do
86       (format stream
87         (concatenate 'string (remove #\( (remove #\) (write-to-string (aref *
               d* i)))) "~%")
88       )
89     )
90   )
91 )
92
93 (defun guarda-mejores-individuos nil
94   (setf *mejores-individuos* (concatenate 'string *mejores-individuos* (
         remove #\( (remove #\) (write-to-string (aref *p* *g*)))) "~%"))
95 )
```

```
96
97 (defun guarda-mejores-aptitudes ()
98   (setf *mejores-aptitudes* (concatenate 'string *mejores-aptitudes* (
       write-to-string (funcionObjetivo (aref *p* *g*))) "~%"))
99 )
100
101 (defun guarda-peores-aptitudes ()
102   (setf *peores-aptitudes* (concatenate 'string *peores-aptitudes* (
       write-to-string (funcionObjetivo (aref *p* *-g*))) "~%"))
103 )
104
105 (defun guarda-promedio-aptitudes (n)
106   (setf *promedio-aptitudes*
107     (concatenate 'string *promedio-aptitudes*
108       (write-to-string
109         (/
110           (apply #'+
111             (loop for i from 0 to (1- n) collect
112               (funcionObjetivo (aref *p* i))
113             )
114           )
115           n
116         )
117       )
118       "~%"
119     )
120   )
121 )
122
123 (defun guarda-resultados nil
124   (setf *file* (concatenate 'string *path* "mejoresIndividuos.data"))
125   (with-open-file
126     (stream *file* :direction :output
127              :if-exists :supersede
128              :if-does-not-exist :create
129     )
130     (format stream *mejores-individuos*)
131   )
132   (setf *file* (concatenate 'string *path* "mejorIndividuo.data"))
133   (with-open-file
134     (stream *file* :direction :output
135              :if-exists :supersede
136              :if-does-not-exist :create
137     )
138     (format stream
139         (concatenate 'string (remove #\( (remove #\) (write-to-string (aref *
             p* *g*)))) "~%")
140     )
141   )
142   (setf *file* (concatenate 'string *path* "mejoresAptitudes.data"))
143   (with-open-file
144     (stream *file* :direction :output
145              :if-exists :supersede
146              :if-does-not-exist :create
147     )
148     (format stream *mejores-aptitudes*)
149   )
150   (setf *file* (concatenate 'string *path* "peoresAptitudes.data"))
151   (with-open-file
152     (stream *file* :direction :output
153              :if-exists :supersede
154              :if-does-not-exist :create
155     )
156     (format stream *peores-aptitudes*)
157   )
158   (setf *file* (concatenate 'string *path* "promedioAptitudes.data"))
159   (with-open-file
160     (stream *file* :direction :output
```

```
161             :if-exists :supersede
162             :if-does-not-exist :create
163     )
164     (format stream *promedio-aptitudes*)
165   )
166 )
167 ;;---------------------------------------------------------------------
168
169 ;;---------------------------------------------------------------------
170 ;;Función de lectura de las distribuciones observadas
171 ;; f := Lista de Archivos con distribuciones
172 (defun leerDistribuciones (f)
173   (adjust-array *o* (list (length f)))
174   (loop for i in f for j from 1 to (length f) do
175     (setf (aref *o* (1- j)) (read-db i))
176   )
177 )
178 ;;---------------------------------------------------------------------
179
180 ;;---------------------------------------------------------------------
181 ;;Funcion de optimización
182 ;; Distribución Esperada
183 (defun funcionObjetivo (E)
184   (/
185     (apply #'+
186       (loop for i from 0 to (1- (array-total-size *o*)) collect
187         (chi-square-pearson
188           (nth 1 (aref *o* i))
189           (loop for j in E collect (* j (apply #'+ (nth 1 (aref *o* i)))))
190         )
191       )
192     )
193     (length E)
194   )
195 )
196 ;;---------------------------------------------------------------------
197
198 ;;---------------------------------------------------------------------
199 ;;Función de generación de problación
200 ;; n := Tamaño de porblación ; f := Archivos de las distribuciones observadas
201
202 (defun inicializaPoblacion (n f)
203   (leerDistribuciones f)
204   (adjust-array *p* (list n))
205   (loop for i from 0 to (1- n) do
206     (setf (aref *p* i) (loop for j from 1 to (nth 0 (nth 0 (aref *o* 0)))
          collect (random 1.0)))
207     (setf (aref *p* i) (loop for j in (aref *p* i) collect (/ j (apply #'+ (
          aref *p* i)))))
208   )
209 )
210 ;;---------------------------------------------------------------------
211
212 ;;---------------------------------------------------------------------
213 ;;Función para encontrar el mejor individuo
214
215 (defun mejorIndividuo nil
216   (setf *g* 0)
217   (loop for i from 1 to (1- (array-total-size *p*)) do
218     (when
219       (and
220         (compruebaFactor1 (aref *p* *g*) (aref *p* i))
221         (compruebaFactor2 (aref *p* *g*) (aref *p* i))
222       )
223       (setf *g* i)
224     )
225   )
226 )
```

```
227
228 (defun compruebaFactor1 (a b)
229   (let
230     (
231       (r T)
232     )
233     (loop for i from 0 to (1- (array-total-size *o*)) do
234       (unless
235         (<=
236           (chi-square-pearson
237             (nth 1 (aref *o* i))
238             (loop for j in b collect (* j (apply #'+ (nth 1 (aref *o* i)))))
239           )
240           (chi-square-pearson
241             (nth 1 (aref *o* i))
242             (loop for j in a collect (* j (apply #'+ (nth 1 (aref *o* i)))))
243           )
244         )
245         (setf r nil)
246       )
247     )
248     r
249   )
250 )
251
252 (defun compruebaFactor2 (a b)
253   (let
254     (
255       (r nil)
256     )
257     (loop for i from 0 to (1- (array-total-size *o*)) do
258       (when
259         (<
260           (chi-square-pearson
261             (nth 1 (aref *o* i))
262             (loop for j in b collect (* j (apply #'+ (nth 1 (aref *o* i)))))
263           )
264           (chi-square-pearson
265             (nth 1 (aref *o* i))
266             (loop for j in a collect (* j (apply #'+ (nth 1 (aref *o* i)))))
267           )
268         )
269         (setf r T)
270       )
271     )
272     r
273   )
274 )
275 ;;-------------------------------------------------------------------------
276
277 ;;-------------------------------------------------------------------------
278 ;;Función para encontrar el peor individuo
279
280 (defun peorIndividuo nil
281   (setf *-g* 0)
282   (loop for i from 1 to (1- (array-total-size *p*)) do
283     (when
284       (and
285         (compruebaFactorA (aref *p* *-g*) (aref *p* i))
286         (compruebaFactorB (aref *p* *-g*) (aref *p* i))
287       )
288       (setf *-g* i)
289     )
290   )
291 )
292
293 (defun compruebaFactorA (a b)
294   (let
```

```
295     (
296       (r T)
297     )
298     (loop for i from 0 to (1- (array-total-size *o*)) do
299       (unless
300         (>=
301           (chi-square-pearson
302             (nth 1 (aref *o* i))
303             (loop for j in b collect (* j (apply #'+ (nth 1 (aref *o* i)))))
304           )
305           (chi-square-pearson
306             (nth 1 (aref *o* i))
307             (loop for j in a collect (* j (apply #'+ (nth 1 (aref *o* i)))))
308           )
309         )
310         (setf r nil)
311       )
312     )
313     r
314   )
315 )
316
317 (defun compruebaFactorB (a b)
318   (let
319     (
320       (r nil)
321     )
322     (loop for i from 0 to (1- (array-total-size *o*)) do
323       (when
324         (>
325           (chi-square-pearson
326             (nth 1 (aref *o* i))
327             (loop for j in b collect (* j (apply #'+ (nth 1 (aref *o* i)))))
328           )
329           (chi-square-pearson
330             (nth 1 (aref *o* i))
331             (loop for j in a collect (* j (apply #'+ (nth 1 (aref *o* i)))))
332           )
333         )
334         (setf r T)
335       )
336     )
337     r
338   )
339 )
340 ;;-------------------------------------------------------------------------
341
342 ;;-------------------------------------------------------------------------
343 ;;función para generar los grupos de cruza
344 ;;f := Metodo de selección ya sea Aleatoria o Mejor :: #'f
345
346
347 (defun Aleatorio nil
348   (let
349     (
350       (a 0)
351       (b 0)
352       (c 0)
353     )
354     (loop for i from 0 to (1- (/ (- (array-total-size *p*) (mod (
          array-total-size *p*) 3)) 3)) do
355       (setf a (random (array-total-size *p*)))
356       (setf b (random (array-total-size *p*)))
357       (loop for j from 1 to 1 do
358         (when (= a b)
359           (setf b (random (array-total-size *p*)))
360           (setf j 1)
361         )
```

```
362       )
363       (setf c (random (array-total-size *p*)))
364       (loop for j from 1 to 1 do
365         (when (or (= a c) (= b c))
366           (setf c (random (array-total-size *p*)))
367           (setf j 1)
368         )
369       )
370       (setf (aref *d* i) (list a b c))
371     )
372   )
373 )
374
375 (defun Mejor nil
376   (let
377     (
378       (a 0)
379       (b 0)
380     )
381     (loop for i from 0 to (1- (/ (- (array-total-size *p*) (mod (
          array-total-size *p*) 3)) 3)) do
382       (setf a (random (array-total-size *p*)))
383       (loop for j from 1 to 1 do
384         (when (= a *g*)
385           (setf a (random (array-total-size *p*)))
386           (setf j 1)
387         )
388       )
389       (setf b (random (array-total-size *p*)))
390       (loop for j from 1 to 1 do
391         (when (or (= b *g*) (= b a))
392           (setf b (random (array-total-size *p*)))
393           (setf j 1)
394         )
395       )
396       (setf (aref *d* i) (list *g* a b))
397     )
398   )
399 )
400
401 (defun elegirPadres (f)
402   (adjust-array *d* (list (/ (- (array-total-size *p*) (mod (array-total-size
        *p*) 3)) 3)))
403   (funcall f)
404 )
405 ;;-------------------------------------------------------------------------
406
407 ;;-------------------------------------------------------------------------
408 ;;Función para la mutación de individuos
409
410 (defun mutacion nil
411   (adjust-array *h* (array-total-size *d*))
412   (loop for i from 0 to (1- (array-total-size *d*)) do
413     (setf (aref *h* i) (mapcar #'(lambda (a b c) (mod (+ a (* (random 2.0) (-
            b c))) 1.0)) (aref *p* (nth 0 (aref *d* i))) (aref *p* (nth 1 (aref
            *d* i))) (aref *p* (nth 2 (aref *d* i)))))
414     (setf (aref *h* i)
415       (loop for j in (aref *h* i) collect
416         (/ j (apply #'+ (aref *h* i)))
417       )
418     )
419     (setf (aref *h* i)
420       (loop for j in (aref *h* i) collect
421         (if (<= j 0.0)
422           (+ j 0.000001)
423           j
424         )
425       )
```

```
426     )
427   )
428 )
429
430 ;;---------------------------------------------------------------------
431
432 ;:---------------------------------------------------------------------
433 ;;Función para la cruza de individuos
434 ;;C := Tasa de recombinación :: f := Función de cruza Binomial(bin) o
        Exponencial(expo) :: #'f
435
436 (defun bin (C)
437   (loop for i from 0 to (1- (array-total-size *h*)) do
438     (setf (aref *h* i) (mapcar #'(lambda (p h) (if (< (random 1.0) C) h p)) (
          aref *p* (nth 0 (aref *d* i))) (aref *h* i)))
439     (setf (aref *h* i)
440       (loop for j in (aref *h* i) collect
441         (/ j (apply #'+ (aref *h* i)))
442       )
443     )
444     (setf (aref *h* i)
445       (loop for j in (aref *h* i) collect
446         (if (<= j 0.0)
447           (+ j 0.000001)
448           j
449         )
450       )
451     )
452   )
453 )
454
455 (defun expo (C)
456   (let
457     (
458       (p NIL)
459       (i 0)
460     )
461     (loop for j from 0 to (1- (array-total-size *h*)) do
462       (setf i 0)
463       (setf p NIL)
464       (loop for k from 0 to (1- (length (aref *h* j))) do
465         (if (< (random 1.0) C)
466           (setf i k)
467           (setf k (length (aref *h* j)))
468         )
469       )
470       (loop for k from 0 to (1- (length (aref *h* j))) do
471         (if (<= k i)
472           (setf p (append p (list (nth k (aref *h* j)))))
473           (setf p (append p (list (nth k (aref *p* (nth 0 (aref *d* j)))))))
474         )
475       )
476       (setf (aref *h* j) p)
477       (setf (aref *h* j)
478         (loop for l in (aref *h* j) collect
479           (/ l (apply #'+ (aref *h* j)))
480         )
481       )
482       (setf (aref *h* i)
483         (loop for j in (aref *h* i) collect
484           (if (<= j 0.0)
485             (+ j 0.000001)
486             j
487           )
488         )
489       )
490     )
491   )
```

```
492 )
493
494 (defun cruza (C f)
495   (funcall f C)
496 )
497 ;;---------------------------------------------------------------------
498
499 ;;---------------------------------------------------------------------
500 ;;Función para la selección de nueva población
501
502 (defun seleccion nil
503   (loop for i from 0 to (1- (array-total-size *h*)) do
504     (setf (aref *h* i) (loop for j in (aref *h* i) collect (/ j (apply #'+ (
          aref *h* i)))))
505     (when
506       (<
507         (funcionObjetivo (aref *h* i))
508         (funcionObjetivo (aref *p* (nth 0 (aref *d* i))))
509       )
510       (setf (aref *p* (nth 0 (aref *d* i))) (aref *h* i))
511     )
512   )
513 )
514 ;;---------------------------------------------------------------------
515
516 ;;---------------------------------------------------------------------
517 ;;Función para revisar la diversidad en la población
518
519 (defun diversidadPoblacion nil
520   (
521     (lambda (p) (/ (apply #'+ p) (length p)))
522     (apply #'append
523       (loop for j from 0 to (- (array-total-size *p*) 2) collect
524         (loop for k from (1+ j) to (1- (array-total-size *p*)) collect
525           (sqrt
526             (apply #'+
527               (loop for n in (aref *p* j) for m in (aref *p* k) collect
528                 (expt (- (* n 100) (* m 100)) 2)
529               )
530             )
531           )
532         )
533       )
534     )
535   )
536 )
537 ;;---------------------------------------------------------------------
538
539 ;;---------------------------------------------------------------------
540 ;;Función de verificación de aptitud aceptable
541 ;; c := Limite de aceptación
542 (defun revisarAptitud (c)
543   (let
544     (
545       (r T)
546     )
547     (loop for i from 0 to (1- (array-total-size *o*)) do
548       (unless
549         (<=
550           (chi-square-pearson
551             (nth 1 (aref *o* i))
552             (loop for j in (aref *p* *g*) collect (* j (apply #'+ (nth 1 (
                  aref *o* i)))))
553           )
554           (* c (1- (nth 0 (nth 0 (aref *o* i)))))
555         )
556         (setf r nil)
557       )
```

```
558     )
559     r
560   )
561 )
562 ;;---------------------------------------------------------------------
563
564 ;;---------------------------------------------------------------------
565 ;;Algoritmo de Evolución Diferencial
566 ;; n := tamaño de población ; g := Generaciones ; Fp := Función de selección
        de padres ; C := Taza de cruza ; Fc := Función de Cruza ; f := Archivo
        de la Base de Datos
567
568 (defun Evolucion-Diferencial (n g Fp C Fc f a)
569   (crea-experimento n g Fp Fc a)
570   (inicializaPoblacion n f)
571   (guarda-generacion 0)
572   (mejorIndividuo)
573   (peorIndividuo)
574   (guarda-mejores-individuos)
575   (guarda-mejores-aptitudes)
576   (guarda-peores-aptitudes)
577   (guarda-promedio-aptitudes n)
578   (loop for i from 1 to g do
579     (elegirPadres Fp)
580     (guarda-padres (1- i))
581     (mutacion)
582     (cruza C Fc)
583     (seleccion)
584     (guarda-generacion i)
585     (mejorIndividuo)
586     (peorIndividuo)
587     (guarda-mejores-individuos)
588     (guarda-mejores-aptitudes)
589     (guarda-peores-aptitudes)
590     (guarda-promedio-aptitudes n)
591     (if (revisarAptitud a)
592       (setf i (1+ g))
593     )
594   )
595   (guarda-resultados)
596 )
597 ;;---------------------------------------------------------------------
598
599 (Evolucion-Diferencial 10 1000 #'Mejor 0.25 #'expo '("prueba.data" "prueba2.
        data" "prueba3.data") 0.554)
600 ;(Evolucion-Diferencial 10 1000 #'Aleatorio 0.25 #'bin '("prueba.data" "
        prueba2.data" "prueba3.data") 0.554)
601 ;(Evolucion-Diferencial 10 1000 #'Aleatorio 0.25 #'expo '("prueba.data" "
        prueba2.data" "prueba3.data") 0.554)
602
603 ;(Evolucion-Diferencial 10 1000 #'Mejor 0.25 #'bin '("prueba.data" "prueba4.
        data" "prueba5.data") 0.554)
604 ;(Evolucion-Diferencial 10 1000 #'Mejor 0.25 #'expo '("prueba.data" "prueba5.
        data" "prueba4.data") 0.554)
605 ;(Evolucion-Diferencial 10 1000 #'Aleatorio 0.25 #'bin '("prueba.data" "
        prueba5.data" "prueba4.data") 0.554)
606 ;(Evolucion-Diferencial 100 1000 #'Aleatorio 0.25 #'expo '("prueba.data" "
        prueba5.data" "prueba4.data") 0.554)
```

D.6. Código de los Experimentos

En esta sección se encuentran los códigos de los experimentos, estos se ejecutan en terminal como se mostró en 6.1.

D.6.1. Experimento 1

Código D.18: Experimento 1. Algoritmo OEP

```
;;;;Experimento 1

(load "PSO.lisp")

;;;Función de Langerman

(PSO #'Langermann 10 100 0.0 10.0)
(print (Langermann (nth 0 (nth 1 (aref *p* *g*))) (nth 1 (nth 1 (aref *p* *g
    *)))))
(PSO #'Langermann 10 500 0.0 10.0)
(print (Langermann (nth 0 (nth 1 (aref *p* *g*))) (nth 1 (nth 1 (aref *p* *g
    *)))))
(PSO #'Langermann 10 1000 0.0 10.0)
(print (Langermann (nth 0 (nth 1 (aref *p* *g*))) (nth 1 (nth 1 (aref *p* *g
    *)))))

(PSO #'Langermann 100 100 0.0 10.0)
(print (Langermann (nth 0 (nth 1 (aref *p* *g*))) (nth 1 (nth 1 (aref *p* *g
    *)))))
(PSO #'Langermann 100 500 0.0 10.0)
(print (Langermann (nth 0 (nth 1 (aref *p* *g*))) (nth 1 (nth 1 (aref *p* *g
    *)))))
(PSO #'Langermann 100 1000 0.0 10.0)
(print (Langermann (nth 0 (nth 1 (aref *p* *g*))) (nth 1 (nth 1 (aref *p* *g
    *)))))

(PSO #'Langermann 1000 100 0.0 10.0)
(print (Langermann (nth 0 (nth 1 (aref *p* *g*))) (nth 1 (nth 1 (aref *p* *g
    *)))))
(PSO #'Langermann 1000 500 0.0 10.0)
(print (Langermann (nth 0 (nth 1 (aref *p* *g*))) (nth 1 (nth 1 (aref *p* *g
    *)))))
(PSO #'Langermann 1000 1000 0.0 10.0)
(print (Langermann (nth 0 (nth 1 (aref *p* *g*))) (nth 1 (nth 1 (aref *p* *g
    *)))))

;;;Función de Griewangk
(PSO #'Griewangk 10 100 -600.0 600.0)
(print (Griewangk (nth 0 (nth 1 (aref *p* *g*))) (nth 1 (nth 1 (aref *p* *g*)
    ))))
(PSO #'Griewangk 10 500 -600.0 600.0)
(print (Griewangk (nth 0 (nth 1 (aref *p* *g*))) (nth 1 (nth 1 (aref *p* *g*)
    ))))
(PSO #'Griewangk 10 1000 -600.0 600.0)
(print (Griewangk (nth 0 (nth 1 (aref *p* *g*))) (nth 1 (nth 1 (aref *p* *g*)
    ))))

(PSO #'Griewangk 100 100 -600.0 600.0)
(print (Griewangk (nth 0 (nth 1 (aref *p* *g*))) (nth 1 (nth 1 (aref *p* *g*)
    ))))
(PSO #'Griewangk 100 500 -600.0 600.0)
(print (Griewangk (nth 0 (nth 1 (aref *p* *g*))) (nth 1 (nth 1 (aref *p* *g*)
    ))))
(PSO #'Griewangk 100 1000 -600.0 600.0)
(print (Griewangk (nth 0 (nth 1 (aref *p* *g*))) (nth 1 (nth 1 (aref *p* *g*)
    ))))

(PSO #'Griewangk 1000 100 -600.0 600.0)
(print (Griewangk (nth 0 (nth 1 (aref *p* *g*))) (nth 1 (nth 1 (aref *p* *g*)
    ))))
(PSO #'Griewangk 1000 500 -600.0 600.0)
(print (Griewangk (nth 0 (nth 1 (aref *p* *g*))) (nth 1 (nth 1 (aref *p* *g*)
    ))))
(PSO #'Griewangk 1000 1000 -600.0 600.0)
```

```
48 (print (Griewangk (nth 0 (nth 1 (aref *p* *g*))) (nth 1 (nth 1 (aref *p* *g*)
       ))))
49
50
51 ;;;Función de Schwefel
52 (PSO #'Schwefel 10 100 -500.0 500.0)
53 (print (Schwefel (nth 0 (nth 1 (aref *p* *g*))) (nth 1 (nth 1 (aref *p* *g*))
       )))
54 (PSO #'Schwefel 10 500 -500.0 500.0)
55 (print (Schwefel (nth 0 (nth 1 (aref *p* *g*))) (nth 1 (nth 1 (aref *p* *g*))
       )))
56 (PSO #'Schwefel 10 1000 -500.0 500.0)
57 (print (Schwefel (nth 0 (nth 1 (aref *p* *g*))) (nth 1 (nth 1 (aref *p* *g*))
       )))
58
59 (PSO #'Schwefel 100 100 -500.0 500.0)
60 (print (Schwefel (nth 0 (nth 1 (aref *p* *g*))) (nth 1 (nth 1 (aref *p* *g*))
       )))
61 (PSO #'Schwefel 100 500 -500.0 500.0)
62 (print (Schwefel (nth 0 (nth 1 (aref *p* *g*))) (nth 1 (nth 1 (aref *p* *g*))
       )))
63 (PSO #'Schwefel 100 1000 -500.0 500.0)
64 (print (Schwefel (nth 0 (nth 1 (aref *p* *g*))) (nth 1 (nth 1 (aref *p* *g*))
       )))
65
66 (PSO #'Schwefel 1000 100 -500.0 500.0)
67 (print (Schwefel (nth 0 (nth 1 (aref *p* *g*))) (nth 1 (nth 1 (aref *p* *g*))
       )))
68 (PSO #'Schwefel 1000 500 -500.0 500.0)
69 (print (Schwefel (nth 0 (nth 1 (aref *p* *g*))) (nth 1 (nth 1 (aref *p* *g*))
       )))
70 (PSO #'Schwefel 1000 1000 -500.0 500.0)
71 (print (Schwefel (nth 0 (nth 1 (aref *p* *g*))) (nth 1 (nth 1 (aref *p* *g*))
       )))
72
73
74 ;;;Función de Rosenbrock
75 (PSO #'Rosenbrock 10 100 -2.048 2.048)
76 (print (Rosenbrock (nth 0 (nth 1 (aref *p* *g*))) (nth 1 (nth 1 (aref *p* *g
       *)))))
77 (PSO #'Rosenbrock 10 500 -2.048 2.048)
78 (print (Rosenbrock (nth 0 (nth 1 (aref *p* *g*))) (nth 1 (nth 1 (aref *p* *g
       *)))))
79 (PSO #'Rosenbrock 10 1000 -2.048 2.048)
80 (print (Rosenbrock (nth 0 (nth 1 (aref *p* *g*))) (nth 1 (nth 1 (aref *p* *g
       *)))))
81
82 (PSO #'Rosenbrock 100 100 -2.048 2.048)
83 (print (Rosenbrock (nth 0 (nth 1 (aref *p* *g*))) (nth 1 (nth 1 (aref *p* *g
       *)))))
84 (PSO #'Rosenbrock 100 500 -2.048 2.048)
85 (print (Rosenbrock (nth 0 (nth 1 (aref *p* *g*))) (nth 1 (nth 1 (aref *p* *g
       *)))))
86 (PSO #'Rosenbrock 100 1000 -2.048 2.048)
87 (print (Rosenbrock (nth 0 (nth 1 (aref *p* *g*))) (nth 1 (nth 1 (aref *p* *g
       *)))))
88
89 (PSO #'Rosenbrock 1000 100 -2.048 2.048)
90 (print (Rosenbrock (nth 0 (nth 1 (aref *p* *g*))) (nth 1 (nth 1 (aref *p* *g
       *)))))
91 (PSO #'Rosenbrock 1000 500 -2.048 2.048)
92 (print (Rosenbrock (nth 0 (nth 1 (aref *p* *g*))) (nth 1 (nth 1 (aref *p* *g
       *)))))
93 (PSO #'Rosenbrock 1000 1000 -2.048 2.048)
94 (print (Rosenbrock (nth 0 (nth 1 (aref *p* *g*))) (nth 1 (nth 1 (aref *p* *g
       *)))))
95
96
```

```
;;;Función de Shubert
(PSO #'Shubert 10 100 -5.12 5.12)
(print (Shubert (nth 0 (nth 1 (aref *p* *g*))) (nth 1 (nth 1 (aref *p* *g*)))
    ))
(PSO #'Shubert 10 500 -5.12 5.12)
(print (Shubert (nth 0 (nth 1 (aref *p* *g*))) (nth 1 (nth 1 (aref *p* *g*)))
    ))
(PSO #'Shubert 10 1000 -5.12 5.12)
(print (Shubert (nth 0 (nth 1 (aref *p* *g*))) (nth 1 (nth 1 (aref *p* *g*)))
    ))

(PSO #'Shubert 100 100 -5.12 5.12)
(print (Shubert (nth 0 (nth 1 (aref *p* *g*))) (nth 1 (nth 1 (aref *p* *g*)))
    ))
(PSO #'Shubert 100 500 -5.12 5.12)
(print (Shubert (nth 0 (nth 1 (aref *p* *g*))) (nth 1 (nth 1 (aref *p* *g*)))
    ))
(PSO #'Shubert 100 1000 -5.12 5.12)
(print (Shubert (nth 0 (nth 1 (aref *p* *g*))) (nth 1 (nth 1 (aref *p* *g*)))
    ))

(PSO #'Shubert 1000 100 -5.12 5.12)
(print (Shubert (nth 0 (nth 1 (aref *p* *g*))) (nth 1 (nth 1 (aref *p* *g*)))
    ))
(PSO #'Shubert 1000 500 -5.12 5.12)
(print (Shubert (nth 0 (nth 1 (aref *p* *g*))) (nth 1 (nth 1 (aref *p* *g*)))
    ))
(PSO #'Shubert 1000 1000 -5.12 5.12)
(print (Shubert (nth 0 (nth 1 (aref *p* *g*))) (nth 1 (nth 1 (aref *p* *g*)))
    ))
```

Código D.19: Experimento 1. Algoritmo EE

```
;;;;Experimento 1 Algoritmo EE

(load "EE.lisp")

(loop for i in '(1 5 10) do
  (print i)
  (loop for j in '(1 5) do
    (print j)
    (loop for k in '(100 500) do
      (print k)
      (Estrategia-Evolutiva #'Langermann i k 0.0 10.0 j #'conSobrevivientes)
      (print (Langermann (nth 0 (nth 0 (aref *u* *g*))) (nth 1 (nth 0 (aref *
          u* *g*)))))
      (Estrategia-Evolutiva #'Langermann i k 0.0 10.0 j #'sinSobrevivientes)
      (print (Langermann (nth 0 (nth 0 (aref *u* *g*))) (nth 1 (nth 0 (aref *
          u* *g*)))))
      (Estrategia-Evolutiva #'Griewangk i k -600.0 600.0 j #'
          conSobrevivientes)
      (print (Griewangk (nth 0 (nth 0 (aref *u* *g*))) (nth 1 (nth 0 (aref *u
          * *g*)))))
      (Estrategia-Evolutiva #'Griewangk i k -600.0 600.0 j #'
          sinSobrevivientes)
      (print (Griewangk (nth 0 (nth 0 (aref *u* *g*))) (nth 1 (nth 0 (aref *u
          * *g*)))))
      (Estrategia-Evolutiva #'Schwefel i k -500.0 500.0 j #'conSobrevivientes
          )
      (print (Schwefel (nth 0 (nth 0 (aref *u* *g*))) (nth 1 (nth 0 (aref *u*
           *g*)))))
      (Estrategia-Evolutiva #'Schwefel i k -500.0 500.0 j #'sinSobrevivientes
          )
      (print (Schwefel (nth 0 (nth 0 (aref *u* *g*))) (nth 1 (nth 0 (aref *u*
           *g*)))))
      (Estrategia-Evolutiva #'Rosenbrock i k -2.048 2.048 j #'
          conSobrevivientes)
```

```
24        (print (Rosenbrock (nth 0 (nth 0 (aref *u* *g*))) (nth 1 (nth 0 (aref *
              u* *g*)))))
25        (Estrategia-Evolutiva #'Rosenbrock i k -2.048 2.048 j #'
              sinSobrevivientes)
26        (print (Rosenbrock (nth 0 (nth 0 (aref *u* *g*))) (nth 1 (nth 0 (aref *
              u* *g*)))))
27        (Estrategia-Evolutiva #'Shubert i k -5.12 5.12 j #'conSobrevivientes)
28        (print (Shubert (nth 0 (nth 0 (aref *u* *g*))) (nth 1 (nth 0 (aref *u*
              *g*)))))
29        (Estrategia-Evolutiva #'Shubert i k -5.12 5.12 j #'sinSobrevivientes)
30        (print (Shubert (nth 0 (nth 0 (aref *u* *g*))) (nth 1 (nth 0 (aref *u*
              *g*)))))
31      )
32    )
33  )
```

Código D.20: Experimento 1. Algoritmo ED

```
1  ;;;;Experimento 1 Algoritmo ED
2
3  (load "ED.lisp")
4
5  (loop for i in '(10 100 1000) do
6    (print i)
7    (loop for j in '(100 500 1000) do
8      (print j)
9      (Evolucion-Diferencial #'Langermann i j #'Mejor 0.25 #'bin 0 10.0)
10     (print (Langermann (nth 0 (aref *p* *g*)) (nth 1 (aref *p* *g*))))
11     (Evolucion-Diferencial #'Langermann i j #'Mejor 0.25 #'expo 0 10.0)
12     (print (Langermann (nth 0 (aref *p* *g*)) (nth 1 (aref *p* *g*))))
13     (Evolucion-Diferencial #'Langermann i j #'Aleatorio 0.25 #'bin 0 10.0)
14     (print (Langermann (nth 0 (aref *p* *g*)) (nth 1 (aref *p* *g*))))
15     (Evolucion-Diferencial #'Langermann i j #'Aleatorio 0.25 #'expo 0 10.0)
16     (print (Langermann (nth 0 (aref *p* *g*)) (nth 1 (aref *p* *g*))))
17     (Evolucion-Diferencial #'Griewangk i j #'Mejor 0.25 #'bin -600.0 600.0)
18     (print (Griewangk (nth 0 (aref *p* *g*)) (nth 1 (aref *p* *g*))))
19     (Evolucion-Diferencial #'Griewangk i j #'Mejor 0.25 #'expo -600.0 600.0)
20     (print (Griewangk (nth 0 (aref *p* *g*)) (nth 1 (aref *p* *g*))))
21     (Evolucion-Diferencial #'Griewangk i j #'Aleatorio 0.25 #'bin -600.0
           600.0)
22     (print (Griewangk (nth 0 (aref *p* *g*)) (nth 1 (aref *p* *g*))))
23     (Evolucion-Diferencial #'Griewangk i j #'Aleatorio 0.25 #'expo -600.0
           600.0)
24     (print (Griewangk (nth 0 (aref *p* *g*)) (nth 1 (aref *p* *g*))))
25     (Evolucion-Diferencial #'Schwefel i j #'Mejor 0.25 #'bin -500.0 500.0)
26     (print (Schwefel (nth 0 (aref *p* *g*)) (nth 1 (aref *p* *g*))))
27     (Evolucion-Diferencial #'Schwefel i j #'Mejor 0.25 #'expo -500.0 500.0)
28     (print (Schwefel (nth 0 (aref *p* *g*)) (nth 1 (aref *p* *g*))))
29     (Evolucion-Diferencial #'Schwefel i j #'Aleatorio 0.25 #'bin -500.0
           500.0)
30     (print (Schwefel (nth 0 (aref *p* *g*)) (nth 1 (aref *p* *g*))))
31     (Evolucion-Diferencial #'Schwefel i j #'Aleatorio 0.25 #'expo -500.0
           500.0)
32     (print (Schwefel (nth 0 (aref *p* *g*)) (nth 1 (aref *p* *g*))))
33     (Evolucion-Diferencial #'Rosenbrock i j #'Mejor 0.25 #'bin -2.048 2.048)
34     (print (Rosenbrock (nth 0 (aref *p* *g*)) (nth 1 (aref *p* *g*))))
35     (Evolucion-Diferencial #'Rosenbrock i j #'Mejor 0.25 #'expo -2.048 2.048)
36     (print (Rosenbrock (nth 0 (aref *p* *g*)) (nth 1 (aref *p* *g*))))
37     (Evolucion-Diferencial #'Rosenbrock i j #'Aleatorio 0.25 #'bin -2.048
           2.048)
38     (print (Rosenbrock (nth 0 (aref *p* *g*)) (nth 1 (aref *p* *g*))))
39     (Evolucion-Diferencial #'Rosenbrock i j #'Aleatorio 0.25 #'expo -2.048
           2.048)
40     (print (Rosenbrock (nth 0 (aref *p* *g*)) (nth 1 (aref *p* *g*))))
41     (Evolucion-Diferencial #'Shubert i j #'Mejor 0.25 #'bin -5.12 5.12)
42     (print (Shubert (nth 0 (aref *p* *g*)) (nth 1 (aref *p* *g*))))
43     (Evolucion-Diferencial #'Shubert i j #'Mejor 0.25 #'expo -5.12 5.12)
44     (print (Shubert (nth 0 (aref *p* *g*)) (nth 1 (aref *p* *g*))))
```

```
45      (Evolucion-Diferencial #'Shubert i j #'Aleatorio 0.25 #'bin -5.12 5.12)
46      (print (Shubert (nth 0 (aref *p* *g*)) (nth 1 (aref *p* *g*))))
47      (Evolucion-Diferencial #'Shubert i j #'Aleatorio 0.25 #'expo -5.12 5.12)
48      (print (Shubert (nth 0 (aref *p* *g*)) (nth 1 (aref *p* *g*))))
49    )
50  )
```

D.6.2. Experimento 2

Código D.21: Experimento 2. Algoritmo OEP

```
1  ;;;;Código del segundo experimento del algoritmo PSO
2
3  ;;Archivos
4
5  (load "PSO.lisp")
6
7  ;;Experimentos
8
9  (loop for n in '(10 50 100) do
10   (loop for m in '(100 500 1000) do
11     (loop for i in '(1 4 7) do
12       (PSO n m (concatenate 'string (write-to-string i) "/distribucion_1.data
             "))
13     )
14   )
15 )
```

Código D.22: Experimento 2. Algoritmo EE

```
1  ;;;;Código del segundo experimento del algoritmo EE
2
3  ;;Archivos
4
5  (load "EE.lisp")
6
7  ;;Experimentos
8
9  (loop for n in '(10 25 50) do
10   (loop for m in '(100 500 1000) do
11     (loop for l in (3 5 7 9) do
12       (loop for i in '(1 4 7) do
13         (Estrategia-Evolutiva (concatenate 'string (write-to-string i) "/
               distribucion_1.data") n m l #'conSobrevivientes)
14         (Estrategia-Evolutiva (concatenate 'string (write-to-string i) "/
               distribucion_1.data") n m l #'sinSobrevivientes)
15       )
16     )
17   )
18 )
```

Código D.23: Experimento 2. Algoritmo ED

```
1  ;;;;Código del segundo experimento del algoritmo ED
2
3  ;;Archivos
4
5  (load "ED.lisp")
6
7  ;;Experimentos
8
9  (loop for n in '(10 50 100) do
10   (loop for m in '(100 500 1000) do
11     (loop for i in '(1 4 7) do
```

```
12        (Evolucion-Diferencial n m #'Mejor 0.25 #'bin  (concatenate 'string (
              write-to-string i) "/distribucion_1.data"))
13        (Evolucion-Diferencial n m #'Mejor 0.25 #'expo  (concatenate 'string (
              write-to-string i) "/distribucion_1.data"))
14        (Evolucion-Diferencial n m #'Aleatorio 0.25 #'bin  (concatenate 'string
               (write-to-string i) "/distribucion_1.data"))
15        (Evolucion-Diferencial n m #'Aleatorio 0.25 #'expo  (concatenate '
              string (write-to-string i) "/distribucion_1.data"))
16      )
17    )
18  )
```

D.6.3. Experimento 3

Código D.24: Experimento 3. Algoritmo OEP

```
1  ;;;;Código del segundo experimento del algoritmo PSO
2
3  ;;Archivos
4
5  (load "PSO.lisp")
6
7  ;;Experimentos
8
9  (defparameter *distribuciones* '(3 6 12))
10
11 (loop for n in '(100 250) do
12   (loop for m in '(1000 2000 3000) do
13     (loop for i from 0 to 8 do
14       (PSO n m
15         (loop for j from 1 to (nth (mod i 3) *distribuciones*) collect
16           (concatenate 'string (write-to-string (1+ i)) "/" "distribucion_" (
                 write-to-string j) ".data")
17         )
18       )
19     )
20   )
21 )
```

Código D.25: Experimento 3. Algoritmo EE

```
1  ;;;;Código del terver experimento del algoritmo EE
2
3  ;;Archivos
4
5  (load "EE.lisp")
6
7  ;;Experimentos
8
9  (defparameter *distribuciones* '(3 6 12))
10
11 (loop for n in '(20 50) do
12   (loop for m in '(1000 2000 3000) do
13     (loop for i from 8 to 8 do
14       #| (Estrategia-Evolutiva
15         (loop for j from 1 to (nth (mod i 3) *distribuciones*) collect
16           (concatenate 'string (write-to-string (1+ i)) "/" "distribucion_" (
                 write-to-string j) ".data")
17         )
18         n m 7 #'conSobrevivientes
19       )
20       (Estrategia-Evolutiva
21         (loop for j from 1 to (nth (mod i 3) *distribuciones*) collect
22           (concatenate 'string (write-to-string (1+ i)) "/" "distribucion_" (
                 write-to-string j) ".data")
```

```
        )
        n m 7 #'sinSobrevivientes
      )
    )
  )
)
```

Código D.26: Experimento 3. Algoritmo ED

```
;;;;Código del terver experimento del algoritmo ED

;;Archivos

(load "ED.lisp")

;;Experimentos

(defparameter *distribuciones* '(3 6 12))

(loop for n in '(100 250) do
  (loop for m in '(1000 2000 3000) do
    (loop for i from 0 to 8 do
      (Evolucion-Diferencial n m #'Mejor 0.25 #'bin
        (loop for j from 1 to (nth (mod i 3) *distribuciones*) collect
          (concatenate 'string (write-to-string (1+ i)) "/" "distribucion_" (
              write-to-string j) ".data")
        )
      )
      (Evolucion-Diferencial n m #'Mejor 0.25 #'expo
        (loop for j from 1 to (nth (mod i 3) *distribuciones*) collect
          (concatenate 'string (write-to-string (1+ i)) "/" "distribucion_" (
              write-to-string j) ".data")
        )
      )
      (Evolucion-Diferencial n m #'Aleatorio 0.25 #'bin
        (loop for j from 1 to (nth (mod i 3) *distribuciones*) collect
          (concatenate 'string (write-to-string (1+ i)) "/" "distribucion_" (
              write-to-string j) ".data")
        )
      )
      (Evolucion-Diferencial n m #'Aleatorio 0.25 #'expo
        (loop for j from 1 to (nth (mod i 3) *distribuciones*) collect
          (concatenate 'string (write-to-string (1+ i)) "/" "distribucion_" (
              write-to-string j) ".data")
        )
      )
    )
  )
)
```

D.6.4. Experimento 4

Código D.27: Experimento 4. Algoritmo OEP

```
(load "PSO.lisp")

(defparameter *distribuciones* '(3 6 12))
(defparameter *porcentajes* '(1.064 1.064 1.064 4.168 4.168 4.168 15.659
    15.659 15.659)) ;Todos al 90%

(loop for n in '(250) do
  (loop for m in '(1500) do
    (loop for i from 0 to 8 do
      (loop for j from 0 to 4 do
        (PSO n m
```

```
11            (loop for j from 1 to (nth (mod i 3) *distribuciones*) collect
12              (concatenate 'string (write-to-string (1+ i)) "/" "distribucion_"
                      (write-to-string j) ".data")
13            )
14            (nth i *porcentajes*)
15          )
16        )
17      )
18    )
19  )
```

Código D.28: Experimento 4. Algoritmo EE

```
1  (load "EE.lisp")
2
3  (defparameter *distribuciones* '(3 6 12))
4  (defparameter *porcentajes* '(1.064 1.064 1.064 4.168 4.168 4.168 15.659
       15.659 15.659)) ;Todos al 90%
5
6  (loop for n in '(30) do
7    (loop for m in '(2000) do
8      (loop for i from 0 to 8 do
9        (loop for j from 0 to 4 do
10         (Estrategia-Evolutiva
11           (loop for j from 1 to (nth (mod i 3) *distribuciones*) collect
12             (concatenate 'string (write-to-string (1+ i)) "/" "distribucion_"
                      (write-to-string j) ".data")
13           )
14           n m 7 #'conSobrevivientes (nth i *porcentajes*)
15         )
16         (Estrategia-Evolutiva
17           (loop for j from 1 to (nth (mod i 3) *distribuciones*) collect
18             (concatenate 'string (write-to-string (1+ i)) "/" "distribucion_"
                      (write-to-string j) ".data")
19           )
20           n m 7 #'sinSobrevivientes (nth i *porcentajes*)
21         )
22       )
23     )
24   )
25 )
```

Código D.29: Experimento 4. Algoritmo ED

```
1  (load "ED.lisp")
2
3  (defparameter *distribuciones* '(3 6 12))
4  (defparameter *porcentajes* '(1.064 1.064 1.064 4.168 4.168 4.168 15.659
       15.659 15.659)) ;Todos al 90%
5
6  (loop for n in '(300) do
7    (loop for m in '(2000) do
8      (loop for i from 0 to 8 do
9        (loop for j from 0 to 4 do
10         (Evolucion-Diferencial n m #'Mejor 0.25 #'bin
11           (loop for j from 1 to (nth (mod i 3) *distribuciones*) collect
12             (concatenate 'string (write-to-string (1+ i)) "/" "distribucion_"
                      (write-to-string j) ".data")
13           )
14           (nth i *porcentajes*)
15         )
16         (Evolucion-Diferencial n m #'Mejor 0.25 #'expo
17           (loop for j from 1 to (nth (mod i 3) *distribuciones*) collect
18             (concatenate 'string (write-to-string (1+ i)) "/" "distribucion_"
                      (write-to-string j) ".data")
19           )
20           (nth i *porcentajes*)
```

```
21          )
22          (Evolucion-Diferencial n m #'Aleatorio 0.25 #'bin
23            (loop for j from 1 to (nth (mod i 3) *distribuciones*) collect
24              (concatenate 'string (write-to-string (1+ i)) "/" "distribucion_"
                      (write-to-string j) ".data")
25            )
26            (nth i *porcentajes*)
27          )
28          (Evolucion-Diferencial n m #'Aleatorio 0.25 #'expo
29            (loop for j from 1 to (nth (mod i 3) *distribuciones*) collect
30              (concatenate 'string (write-to-string (1+ i)) "/" "distribucion_"
                      (write-to-string j) ".data")
31            )
32            (nth i *porcentajes*)
33          )
34        )
35      )
36    )
37  )
```

D.6.5. Experimento 5

Código D.30: Experimento 5. Algoritmo OEP

```
1  (load "PSO.lisp")
2
3  (loop for i from 0 to 9 do
4    (PSO 300 3000
5      (loop for j in '("Ene" "Feb" "Mar" "Abr") collect
6        (concatenate 'string "BC/dis" j ".data")
7      )
8      1.064
9    )
10 )
11
12 (loop for i from 0 to 9 do
13   (PSO 300 3000
14     (loop for j in '("Ene" "Feb" "Mar" "Abr") collect
15       (concatenate 'string "COL/dis" j ".data")
16     )
17     4.168
18   )
19 )
20
21 (loop for i from 0 to 9 do
22   (PSO 300 3000
23     (loop for j in '("Ene" "Feb" "Mar" "Abr") collect
24       (concatenate 'string "CDMX/dis" j ".data")
25     )
26     8.547
27   )
28 )
```

Código D.31: Experimento 5. Algoritmo EE

```
1  (load "EE.lisp")
2
3  (loop for i from 0 to 4 do
4    (Estrategia-Evolutiva
5      (loop for j in '("Ene" "Feb" "Mar" "Abr") collect
6        (concatenate 'string "BC/dis" j ".data")
7      )
8      50 3000 7 #'conSobrevivientes 1.064
9    )
10 )
```

```
11
12 (loop for i from 0 to 4 do
13   (Estrategia-Evolutiva
14     (loop for j in '("Ene" "Feb" "Mar" "Abr") collect
15       (concatenate 'string "COL/dis" j ".data")
16     )
17     50 3000 7 #'conSobrevivientes 4.168
18   )
19 )
20
21 (loop for i from 0 to 4 do
22   (Estrategia-Evolutiva
23     (loop for j in '("Ene" "Feb" "Mar" "Abr") collect
24       (concatenate 'string "CDMX/dis" j ".data")
25     )
26     50 3000 7 #'conSobrevivientes 15.659
27   )
28 )
29
30 (loop for i from 0 to 4 do
31   (Estrategia-Evolutiva
32     (loop for j in '("Ene" "Feb" "Mar" "Abr") collect
33       (concatenate 'string "BC/dis" j ".data")
34     )
35     50 3000 7 #'sinSobrevivientes 1.064
36   )
37 )
38
39 (loop for i from 0 to 4 do
40   (Estrategia-Evolutiva
41     (loop for j in '("Ene" "Feb" "Mar" "Abr") collect
42       (concatenate 'string "COL/dis" j ".data")
43     )
44     50 3000 7 #'sinSobrevivientes 4.168
45   )
46 )
47
48 (loop for i from 0 to 4 do
49   (Estrategia-Evolutiva
50     (loop for j in '("Ene" "Feb" "Mar" "Abr") collect
51       (concatenate 'string "CDMX/dis" j ".data")
52     )
53     50 3000 7 #'sinSobrevivientes 15.659
54   )
55 )
```

Código D.32: Experimento 5. Algoritmo ED

```
1 (load "ED.lisp")
2
3 (loop for i from 0 to 9 do
4   (Evolucion-Diferencial 300 3000 #'Mejor 0.25 #'bin
5     (loop for j in '("Ene" "Feb" "Mar" "Abr") collect
6       (concatenate 'string "BC/dis" j ".data")
7     )
8     1.064
9   )
10
11 )
12
13 (loop for i from 0 to 9 do
14   (Evolucion-Diferencial 300 3000 #'Mejor 0.25 #'bin
15     (loop for j in '("Ene" "Feb" "Mar" "Abr") collect
16       (concatenate 'string "COL/dis" j ".data")
17     )
18     4.168
19   )
20
```

```
21 )
22
23 (loop for i from 0 to 9 do
24   (Evolucion-Diferencial 300 3000 #'Mejor 0.25 #'bin
25     (loop for j in '("Ene" "Feb" "Mar" "Abr") collect
26       (concatenate 'string "CDMX/dis" j ".data")
27     )
28     8.547
29   )
30
31 )
32
33 (loop for i from 0 to 9 do
34   (Evolucion-Diferencial 300 3000 #'Mejor 0.25 #'exp
35     (loop for j in '("Ene" "Feb" "Mar" "Abr") collect
36       (concatenate 'string "BC/dis" j ".data")
37     )
38     1.064
39   )
40
41 )
42
43 (loop for i from 0 to 9 do
44   (Evolucion-Diferencial 300 3000 #'Mejor 0.25 #'exp
45     (loop for j in '("Ene" "Feb" "Mar" "Abr") collect
46       (concatenate 'string "COL/dis" j ".data")
47     )
48     4.168
49   )
50
51 )
52
53 (loop for i from 0 to 9 do
54   (Evolucion-Diferencial 300 3000 #'Mejor 0.25 #'exp
55     (loop for j in '("Ene" "Feb" "Mar" "Abr") collect
56       (concatenate 'string "CDMX/dis" j ".data")
57     )
58     8.547
59   )
60
61 )
62
63 (loop for i from 0 to 9 do
64   (Evolucion-Diferencial 300 3000 #'Aleatorio 0.25 #'bin
65     (loop for j in '("Ene" "Feb" "Mar" "Abr") collect
66       (concatenate 'string "BC/dis" j ".data")
67     )
68     1.064
69   )
70
71 )
72
73 (loop for i from 0 to 9 do
74   (Evolucion-Diferencial 300 3000 #'Aleatorio 0.25 #'bin
75     (loop for j in '("Ene" "Feb" "Mar" "Abr") collect
76       (concatenate 'string "COL/dis" j ".data")
77     )
78     4.168
79   )
80
81 )
82
83 (loop for i from 0 to 9 do
84   (Evolucion-Diferencial 300 3000 #'Aleatorio 0.25 #'bin
85     (loop for j in '("Ene" "Feb" "Mar" "Abr") collect
86       (concatenate 'string "CDMX/dis" j ".data")
87     )
88     8.547
```

```
  )

)

(loop for i from 0 to 9 do
  (Evolucion-Diferencial 300 3000 #'Aleatorio 0.25 #'exp
    (loop for j in '("Ene" "Feb" "Mar" "Abr") collect
      (concatenate 'string "BC/dis" j ".data")
    )
    1.064
  )

)

(loop for i from 0 to 9 do
  (Evolucion-Diferencial 300 3000 #'Aleatorio 0.25 #'exp
    (loop for j in '("Ene" "Feb" "Mar" "Abr") collect
      (concatenate 'string "COL/dis" j ".data")
    )
    4.168
  )

)

(loop for i from 0 to 9 do
  (Evolucion-Diferencial 300 3000 #'Aleatorio 0.25 #'exp
    (loop for j in '("Ene" "Feb" "Mar" "Abr") collect
      (concatenate 'string "CDMX/dis" j ".data")
    )
    8.547
  )

)
```

Bibliografía

[1] V. Furtado, A. Melo, A. L. Coelho, R. Menezes, and R. Perrone, "A bio-inspired crime simulation model," *Decision Support Systems*, vol. 48, no. 1, pp. 282–292, 2009.

[2] M. Han and M. Fan, "Application of neural networks on multivariate time series modeling and prediction," in *American Control Conference, 2006*, pp. 6–pp, IEEE, 2006.

[3] S. Mukherjee, E. Osuna, and F. Girosi, "Nonlinear prediction of chaotic time series using support vector machines," in *Neural Networks for Signal Processing [1997] VII. Proceedings of the 1997 IEEE Workshop*, pp. 511–520, IEEE, 1997.

[4] G. Chakraborty, H. Watanabe, and B. Chakraborty, "Prediction in dynamic system-a divide and conquer approach," in *Soft Computing in Industrial Applications, 2005. SMCia/05. Proceedings of the 2005 IEEE Mid-Summer Workshop on*, pp. 196–201, IEEE, 2005.

[5] H. Yanbin, G. Xianwen, W. Mingshun, L. Xiangyu, L. Yu, and W. Bing, "Application of dynamic liquid level prediction model based on improved svr in sucker rod pump oil wells," in *Control Conference (CCC), 2013 32nd Chinese*, pp. 7826–7830, IEEE, 2013.

[6] S. de los Cobos Silva, J. G. Close, M. G. Andrade, and A. M. Licona, "Búsqueda y exploración estocástica," *Universidad Autónoma Metropolitana, México*, 2010.

[7] L. Araujo and C. Cervigón, *Algoritmos evolutivos: un enfoque práctico*. Alfaomega, 2009.

[8] D. Floreano and C. Mattiussi, *Bio-inspired artificial intelligence: theories, methods, and technologies*. MIT press, 2008.

[9] C. A. C. Coello, "Introducción a la computación evolutiva," *Notas del curso. Departamento de Ingeniería Eléctrica, Sección de Computación, Instituto Politécnico Nacional, México*, 2004.

[10] G. Polya, *How to solve it: A new aspect of mathematical method*. Princeton university press, 2014.

[11] W. B. Cannon, "The wisdom of the body.," 1932.

[12] A. M. Turing, "Computing machinery and intelligence," *Mind*, pp. 433–460, 1950.

[13] C. Coello, "La computación evolutiva en el contexto de la inteligencia artificial," *LANIA, AC, Méxic o*, 2000.

[14] C. A. C. Coello, "Introducción a la computación evolutiva (notas de curso)," *CINVESTAV-IPN, Departamento de Ingeniería Eléctrica, Sección de Computación. México, DF*, 2004.

[15] G. E. Box, "Evolutionary operation: A method for increasing industrial productivity," *Applied Statistics*, pp. 81–101, 1957.

[16] R. M. Friedberg, "A learning machine: Part i," *IBM Journal of Research and Development*, vol. 2, no. 1, pp. 2–13, 1958.

[17] H. J. Bremermann, *The evolution of intelligence: The nervous system as a model of its environment*. University of Washington, Department of Mathematics, 1958.

[18] H. J. Bremermann, "Optimization through evolution and recombination," *Self-organizing systems*, pp. 93–106, 1962.

[19] H. J. Bremermann, "Numerical optimization procedures derived from biological evolution processes," *Cybernetic problems in bionics*, vol. 1, no. 9, p. 6, 1968.

[20] H. J. Bremermann and M. Rogson, "An evolution-type search method for convex sets.," tech. rep., DTIC Document, 1964.

[21] N. A. Barricelli *et al.*, "Esempi numerici di processi di evoluzione," *Methodos*, vol. 6, no. 21-22, pp. 45–68, 1954.

[22] N. A. Barricelli, "Symbiogenetic evolution processes realized by artificial methods," *Methodos*, vol. 9, no. 35-36, pp. 143–182, 1957.

[23] J. Reed, R. Toombs, and N. A. Barricelli, "Simulation of biological evolution and machine learning: I. selection of self-reproducing numeric patterns by data processing machines, effects of hereditary control, mutation type and crossing," *Journal of theoretical biology*, vol. 17, no. 3, pp. 319–342, 1967.

[24] L. J. Fogel, A. J. Owens, and M. J. Walsh, "Artificial intelligence through simulated evolution," 1966.

[25] J. Heitkotter and D. Beasley, "The hitch-hiker's guide to evolutionary computation (faq for comp. ai. genetic) issue 8.2, 2000."

[26] T. Back, D. B. Fogel, and Z. Michalewicz, *Handbook of evolutionary computation*. IOP Publishing Ltd., 1997.

[27] K. Pearson, "X. on the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 50, no. 302, pp. 157–175, 1900.

[28] R. Johnson and P. Kuby, "Estadística elemental: Lo esencial méxico," 2004.

[29] N. William, "Estadística para ingenieros y científicos," *México. McGraw-Hill*, 2006.

[30] M. L. Cachero, *Fundamentos y métodos de estadística*. 1996.

[31] J. A. Moyne, "Lisp. primer lenguaje de para programadores de computadoras," 1994.

[32] J. G. García, "Lenguaje lisp - inteligencia artificial," 2009. https://sites.google.com/site/proyectointeligenciaartificial/indice/lenguajes-de-programacin/lenguaje-lisp.

[33] "About - steel bank common lisp." http://www.sbcl.org.

[34] "gnuplot homepage." http://www.gnuplot.info.

[35] "Sublime text: The text editor you will fall in love with." http://www.sublimetext.com/.

[36] L. V. S. Quintero, *Un Algoritmo Basado en Evolución Diferencial para Resolver Problemas Multiobjetivo*. PhD thesis, Tesis de Maestría CINVESTAV-IPN, 2004.

[37] R. Storn and K. Price, *Differential evolution-a simple and efficient adaptive scheme for global optimization over continuous spaces*, vol. 3. ICSI Berkeley, 1995.

[38] F. Vitaliy, "Differential evolution–in search of solutions," 2006.

[39] L. C. Echevarría, O. L. Santiago, and A. S. Neto, "Aplicación de los algoritmos evolución diferencial y colisión de partículas al diagnóstico de fallos en sistemas industriales," *Revista Investigación Operacional*, vol. 33, no. 2, pp. 160–172, 2012.

[40] K. Price, R. M. Storn, and J. A. Lampinen, *Differential evolution: a practical approach to global optimization*. Springer Science & Business Media, 2006.

[41] M. Muñoz, J. López, and E. F. Caicedo, "Inteligencia de enjambres: sociedades para la solución de problemas (una revisión)," *Ingeniería e Investigación*, vol. 28, no. 2, pp. 119–130, 2008.

[42] M. del Carmen Chávez, G. Casas, J. Moreira, E. González, R. B. Pérez, and R. Grau, "Uso de redes bayesianas obtenidas mediante optimización de enjambre de partículas para el diagnóstico de la hipertensión arterial," *Investigación Operacional*, vol. 30, no. 1, pp. 52–60, 2009.

[43] S. de los Cobos Silva, M. A. Gutiérrez-Andrade, E. A. Rincón-Garcıa, P. Lara-Velázquez, and M. Aguilar-Cornejo, "Colonia de abejas artificiales y optimización por enjambre de partículas para la estimación de parámetros de regresión no lineal," *Revista de Matemática: Teoría y Aplicaciones*, vol. 21, no. 1, pp. 107–126, 2014.

[44] E. A. RINCÓN-GARCÍA, M. A. GUTIÉRREZ-ANDRADE, S. G. DE-LOS-COBOS-SILVA, P. LARA-VELÁZQUEZ, R. A. MORA-GUTIÉRREZ, and A. S. PONSICH, "A discrete particle swarm optimization algorithm for designing electoral zones," in *Methods for Decision Making in an Uncertain Environment-Proceedings of the Xvii Sigef Congress*, vol. 6, p. 174, World Scientific, 2012.

[45] G. W. Greenwood and Q. J. Zhu, "Convergence in evolutionary programs with self-adaptation," *Evolutionary Computation*, vol. 9, no. 2, pp. 147–157, 2001.

[46] G. de Finanzas Computacionales, "Una comparación entre estrategias evolutivas y rprop para la estimación de redes neuronales a comparison between evolutionary strategies and rprop for estimating neural networks," *Revista Avances en Sistemas e Informática*, vol. 4, no. 2, 2007.

[47] "Secretariado ejecutivo - datos abiertos de incidencia delictiva." http://secretariadoejecutivo.gob.mx/incidencia-delictiva/incidencia-delictiva-datos-abiertos.php.