GBHS-LEM: Hibridación del algoritmo GBHS con Modelos LEM

DARIO FRANCISCO ESTUPIÑAN

JOSE PEREZ HERNANDEZ

Director: Ph.D. (c) MSc. CARLOS ALBERTO COBOS LOZADA

UNIVERSIDAD DEL CAUCA

FACULTAD DE INGENIERÍA ELECTRÓNICA Y TELECOMUNICACIONES

DEPARTAMENTO DE SISTEMAS

GRUPO DE I+D EN TECNOLOGÍAS DE LA INFORMACIÓN

GESTIÓN DE LA INFORMACIÓN

POPAYÁN, MARZO DE 2011

Agradecimientos

A Dios, por ser nuestro creador, por permitirnos seguir nuestro camino y darnos la fuerza para continuar cada día.

A nuestras familias, que sin esperar nada a cambio han acompañado cada momento de nuestras vidas, con su esfuerzo y aliento para que lleguemos a nuestras metas propuestas.

Al Ph.D. (c). Carlos Alberto Cobos Lozada por su dedicación, tiempo, apoyo y enorme conocimiento para guiarnos en este reto.

A nuestros amigos y educadores, fieles compañeros en este proceso con su ánimo, colaboración y palabras de aliento.

Para finalizar, nuestros agradecimientos a la Universidad del Cauca, institución que nos forjó como personas, brindándonos la oportunidad a través del programa de Ingeniería de Sistemas de realizar nuestros estudios de pregrado.

Tabla de Contenido

LISTA DE TABLAS

LISTA DE FIGURAS

Presentación

La optimización es un proceso matemático y estadístico que intenta dar respuesta a un tipo general de problemas en los cuales se desea elegir la mejor solución dentro de un conjunto de posibles soluciones. Puede entenderse como un conjunto de métodos que sirven para resolver problemas matemáticos complejos.

En la actualidad, la optimización ha sido un campo en el cual se han producido grandes avances científicos. Infinidad de nuevas propuestas (métodos, algoritmos, etc.) que incluyen estrategias de inteligencia artificial han sido presentadas como alternativas para resolver problemas en los cuales las matemáticas tradicionales no son la mejor alternativa.

Entre los diversos métodos y algoritmos propuestos en el campo de la optimización se destaca el algoritmo de la búsqueda armónica (*Harmony Search*). Este algoritmo basado en el proceso de la improvisación musical ha demostrado ser más eficiente que otras técnicas empleadas (algoritmos genéticos) en la solución de diversos problemas de optimización (diseño estructural, redes hidráulicas, robótica, genética, etc.) y ha sido objeto de infinidad de aplicaciones y mejoras entre ellas la búsqueda armónica mejorada, (*Improved Harmony Search*) y la mejor búsqueda armónica global (*Global-best Harmony Search*), las cuales buscan aumentar su precisión y/o disminuir el tiempo de respuesta, sin embargo, todavía hay muchas mejoras por hacer.

Con el fin de mejorar la precisión del algoritmo de la búsqueda armónica, en la presente monografía de trabajo de grado se describe un nuevo algoritmo de

búsqueda armónica, basado en el algoritmo de la mejor búsqueda armónica y modelos evolutivos que aprenden (*Learnable Evolutionary Models*).

En este documento se encuentran diferentes secciones que contienen la descripción de los conceptos teóricos además de la metodología utilizada para el desarrollo del proyecto. A continuación se describe de manera general el contenido de esta monografía y su organización.

Parte 1 – Introducción: Este capítulo presenta el planteamiento del problema que dio origen al proyecto, la justificación del mismo, los objetivos propuestos para la solución y los resultados obtenidos durante el desarrollo del proyecto.

Parte 2 – Marco Teórico: Este capítulo enmarca las bases teóricas utilizadas para el desarrollo del proyecto, las cuales comprenden: el algoritmo de búsqueda armónica (HS) y sus mejoras (IHS, GHS y NGHS); Modelos Evolutivos que aprenden (LEM) y el algoritmo PRISM.

Parte 3 – Descripción del Survey: Aquí se describe la metodología utilizada para construir el Survey propuesto y parte de su contenido.

Parte 4 – Algoritmo GHS+LEM: En este capítulo se presenta en detalle los aspectos relacionados con el algoritmo propuesto en este proyecto. Se describe la etapa de pre-procesamiento realizada, los parámetros y los pasos del algoritmo.

Parte 5 – Experimentación: En este capítulo se presenta la metodología utilizada para realizar las pruebas de algoritmo, análisis y resultados finales.

Parte 6 – Conclusiones, recomendaciones y trabajo futuro: Aquí se describen las conclusiones generadas después de terminar el proyecto, además se presentan posibles mejoras o elementos adicionales que se puedan incluir en futuros trabajos de investigación.

Parte 7 – Glosario y Bibliografía: Este capítulo contiene el catálogo de palabras importantes en la investigación, con su respectiva definición y la bibliografía empleada para soportar el proyecto.

Parte 1 – Introducción

1 PLANTEAMIENTO DEL PROBLEMA

La optimización es un proceso matemático y estadístico cuya finalidad es obtener la mejor de todas las soluciones posibles a un problema. Las primeras técnicas de optimización se remontan a Gauss y el método de los mínimos cuadrados, en el que dado un conjunto de pares (o ternas), se intentaba encontrar la función que mejor se aproximaba a los datos de acuerdo con el criterio del mínimo error cuadrático [1]. Después, con la aparición de los computadores se introdujo el término programación lineal, propuesto por George Dantzig en 1940 [2]. La programación lineal estudia el caso en el cual se busca optimizar una función f lineal teniendo en cuenta el conjunto de restricciones que la definen y la describen usando únicamente expresiones lineales [3]. La programación lineal surgió en un principio con fines militares, evolucionó a través de estudios netamente académicos los cuales buscaban resolver diversos problemas matemáticos que involucran optimización (programación de convergencia, estocástica, heurística, etc.) [4].

Las ideas de la programación lineal han inspirado muchos de los conceptos centrales de la teoría de la optimización y se usan para resolver problemas en los cuales las variables pueden representar datos de cualquier magnitud. En el área de la optimización se destacan George Dantzig, Albert Tucker y Phil Wolfe entre otros autores, quienes fundaron la MPS (Sociedad de Programación Matemática) la cual es en la actualidad el organismo más importante en el estudio de la optimización.

Hoy en día, la investigación en optimización es ampliamente difundida como un conjunto de métodos que sirven para resolver problemas matemáticos complejos. Dentro de este conjunto de métodos nace un término denominado meta heurística[1], derivado del campo de la optimización estocástica, donde las variables de un problema son aleatorias [5]. La optimización estocástica está compuesta de un conjunto de algoritmos y técnicas que se usan para resolver problemas en los cuales no se conoce la solución óptima, no se posee un número de criterios bien definidos y un algoritmo de fuerza bruta no es práctico debido al tamaño de la muestra [6-7].

En el marco de las meta heurísticas surgen los algoritmos de búsqueda armónicos (Harmony Search), los cuales se basan en la observación del proceso de improvisación musical [8]. Su principal innovación con respecto a los demás métodos de búsqueda de esta rama, es que plantea una *Derivativa Estocástica* [9-10]. Esta derivativa es un proceso que incorpora elementos aleatorios probabilísticos tanto en los datos del problema como en el algoritmo en sí, a diferencia de otros métodos los cuales emplean el método de las gradientes, en el cual se encuentra un mínimo o máximo local a través de tomar puntos más cercanos al valor del gradiente en un punto hasta localizar el mínimo o máximo buscado, ejemplos de estos métodos son la búsqueda Tabú, Convolución Gaussiana y Búsqueda local iterada [10]. Usando la Derivativa Estocástica, los algoritmos de búsqueda armónica logran efectuar un muestreo del espacio de solución del problema lo suficientemente representativo sin emplear demasiado tiempo en la búsqueda. Estos métodos han sido empleados con éxito en el clustering (agrupación) de documentos web [11], resumen de textos [12], enrutamiento en internet [13], entre otros.

[1] Un marco de trabajo o enfoque algorítmico de alto nivel que se especializa en resolver problemas de optimización. También se puede definir como una estrategia de alto nivel que guía otras heurísticas en la búsqueda de soluciones factibles (National Institute of Standards and Technology ver http://www.itl.nist.gov/div897/sqg/dads/HTML/metaheuristic.html).

A partir de la versión inicial de la búsqueda armónica propuesta por Geem [6], se han propuesto varias mejoras entre las que se destacan el algoritmo de la mejor búsqueda armónica global (Global-Best Harmony Search, GBHS) [5], el cual utiliza técnicas de optimización de enjambre (Particle Swarm Optimization, PSO) y la búsqueda armónica mejorada (IHS) [14] que realiza la adaptación (cambio en tiempo de ejecución) de dos parámetros del algoritmo original propuesto por Geem. Aunque estas mejoras han demostrado ser eficientes en la resolución de diversos problemas de optimización, todavía no se logra un cambio significativo en el tiempo de respuesta frente a los resultados de la búsqueda armónica original. IHS es efectivo pero su rendimiento no presenta un avance significativo en cuanto al tiempo de respuesta [14]. Por otro lado el GBHS trabaja muy bien en problemas discretos y continuos, mejorando la calidad de las soluciones cuando aumenta la dimensionalidad (número de atributos usados para estimar, clasificar o predecir la variable objetivo) del problema, pero con un rendimiento que no es significativo en problemas de baja dimensionalidad [5].

Por otro lado, entre las técnicas evolutivas de optimización una nueva clase de procesos se presenta con el nombre de Modelos Evolutivos que Aprenden (Learnable Evolution Model, LEM). En contraste con el método darwiniano, aplicado a la computación evolutiva, el cual se basa en mutación, combinación y selección natural, LEM adicionalmente emplea técnicas de aprendizaje de máquina para generar nuevas poblaciones. Al usar el modo de aprendizaje de máquina, LEM puede determinar cuáles individuos de una población (o un conjunto de individuos de poblaciones antiguas) son superiores a otros en la realización de ciertas tareas. Estas razones, expresadas como hipótesis inductivas, se usan para la generación de nuevas poblaciones. Luego, cuando el algoritmo se ejecuta en modo de evolución darwiniana usa operaciones aleatorias o semi-aleatorias para la generación de nuevos individuos (usando técnicas de mutación y/o recombinación tradicionales) [15].

En los algoritmos de búsqueda armónica, el proceso de creación de poblaciones se basa en la aleatoriedad, es decir las poblaciones son mejoradas basándose en el ajuste de ciertas variables que utilizan características exitosas en antiguas poblaciones para crear las nuevas. El GBHS ha presentado un rendimiento más significativo que depende de la dimensionalidad del problema, dándole una enorme ventaja sobre los otros algoritmos de búsqueda armónica. La propuesta de LEM implica una etapa más elaborada en la cual se decide si las poblaciones deben ser generadas aleatoriamente o utilizando el modelo darwiniano, haciendo que las poblaciones generadas sean mejores en cada paso, aumentando la velocidad de convergencia. Por lo anterior se planteo la siguiente pregunta de investigación ¿Es posible aumentar la precisión[2] y/o disminuir el tiempo de convergencia[3] a la solución optima encontrada por el algoritmo de la Mejor Búsqueda Armónica Global (GBHS) utilizando modelos de aprendizaje evolutivo (LEM)?

2 JUSTIFICACIÓN

Partiendo de la pregunta de investigación previamente planteada y teniendo en cuenta los algoritmos armónicos propuestos hasta la fecha, aun existe la necesidad de superar ciertas deficiencias importantes, como lo son el aumento de la precisión y la disminución del tiempo de convergencia a la solución optima. Este proyecto presenta una alternativa de solución mediante la hibridación del algoritmo de la Mejor Búsqueda Armónica Global (GBHS)[5] y los modelos de aprendizaje evolutivo (LEM) [15], así como del uso de una variación del algoritmo PRISM [16], los cuales permiten obtener una mejora notable en la generación de soluciones optimas.

[2] La precisión hace referencia a si existe un acuerdo entre una medición efectuada sobre un objeto y su verdadero valor (objetivo o de referencia). factibles (National Institute of Standards and Technology ver http://www.itl.nist.gov/div897/sqg/dads/HTML/metaheuristic.html).

[3] El tiempo de convergencia de un algoritmo de optimización hace referencia al tiempo invertido en aproximarse a un valor definido o a un punto definido.
http://buscon.rae.es/draeI/SrvltConsulta?TIPO_BUS=3&LEMA=convergencia

Este proyecto fue muy conveniente puesto que buscó mejorar el rendimiento del algoritmo GBHS frente a los diversos problemas de optimización que se presentan en problemas reales, haciendo uso de LEM con el objetivo de mejorar la calidad de las soluciones con base en las características detectadas en cada generación.

Por otra parte, el proyecto fue un aporte investigativo en el campo de la optimización a nivel internacional, con lo cual se buscó proyectar la investigación de los algoritmos armónicos aplicando diversos métodos evolutivos, entre ellos LEM, el cual hace parte de nuestra propuesta y que no ha sido implementado en investigaciones anteriores. En las pruebas de laboratorio se utilizaron los problemas de optimización usados en este campo para medir el rendimiento de este tipo de algoritmos con otros resultados obtenidos en diferentes propuestas de investigación presentadas ante la comunidad científica.

Las herramientas tecnológicas (Microsoft: Microsoft Visual Studio 2010 y Microsoft Project 2007) necesarias para la realización de este proyecto fueron las adecuadas; ellas fueron inicialmente seleccionadas con base en la experiencia que el GTI ha obtenido en los últimos siete (7) años de trabajo, específicamente a las experiencias exitosas en el desarrollo de proyectos relacionados con optimización (como por ejemplo: Buscador Inteligente Basado en Minería de Datos, Hibridación De La Mejor Búsqueda Armónica Global Y El Algoritmo K-Means Para El Clustering De Documentos Web) y finalmente a la disponibilidad del software, documentación y materiales de aprendizaje que se tiene en la Universidad del Cauca, gracias al programa MSDN Academic Alliance[4].

Durante el desarrollo de este proyecto se aplicaron varios conceptos aprendidos en el transcurso de la carrera de ingeniería de sistemas, y se ha realizado una investigación muy profunda sobre los nuevos conceptos necesarios para la

[4] Acuerdo que vincula a Microsoft con entidades educativas universitarias, en la cual se permite tener acceso a software de desarrollo con propósitos académicos.

consecución del producto final. Con lo anterior, los autores han podido demostrar su capacidad para plantear y resolver los problemas relacionados con su formación ya como profesionales y como investigadores en formación de su alma mater.

3 OBJETIVOS

A continuación se muestran los objetivos del proyecto, conforme fueron aprobados por el Comité de Investigaciones de la Facultad de Ingeniería Electrónica y Telecomunicaciones en el documento de anteproyecto.

3.1 OBJETIVO GENERAL

Proponer un algoritmo que hibride la Mejor Búsqueda Armónica Global (GBHS) con modelos LEM (Learnable Evolutionary Models) y comparar sus resultados con las tres variaciones más conocidas de la búsqueda armónica en la solución de problemas de optimización.

3.2 OBJETIVOS ESPECÍFICOS

- Obtener un survey[5] que refleje los más recientes (últimos diez años, 2001-2010) aportes de investigación en la Búsqueda Armónica (Harmony Search, HS).

- Modelar un algoritmo de optimización que basado en la meta heurística de la Mejor Búsqueda Armónica Global (GBHS) incorpore una etapa de aprendizaje basada en modelos LEM (Learnable Evolutionary Models).

- Diseñar e implementar un prototipo software que permita realizar comparaciones entre el algoritmo propuesto, el algoritmo de búsqueda

[5] En ciencias de la computación, es un documento de investigación que presenta una visión general de los puntos de vista, modelos, estrategias y/o algoritmos usados por la comunidad científica para enfrentar y solucionar un problema o tema de investigación.

armónica original propuesto por Geem, el algoritmo de la búsqueda armónica mejorada y el algoritmo de la mejor búsqueda armónica global (GBHS) con el fin de comparar la velocidad de convergencia y la precisión de la solución en 10 funciones clásicas usadas en optimización (Sphere, Schwefel, Step, Rosenbrock, Rotated hyper-ellipsoid, Generalized Schwefel, Rastrigin, Ackley, Griewank, Six-Hump Camel-back).

4 RESULTADOS OBTENIDOS

- Artículo: a survey of harmony search (Una revisión de la búsqueda armónica) en proceso de evaluación en la revista Avances en Sistemas e Informática de la Universidad Nacional de Colombia sede Medellín (RASI). Esta revista está actualmente en categoría C según el PUBLINDEX de COLCIENCIAS. Sitio Web: http://pisis.unalmed.edu.co/avances.

- Aplicación Software de Laboratorio. Utilizada en las pruebas de laboratorio del proyecto, código fuente e instaladores.

- Aplicación Software de Animación. Utilizada para presentar los resultados de la investigación y el proceso de búsqueda del algoritmo GHS-LEM frente a los otros algoritmos de búsqueda armónica, código fuente e instaladores.

- Artículo: *GHS+LEM: Global-best Harmony Search using Learnable Evolution Models (GHS+LEM: Global-best Harmony Search usando modelos evolutivos que aprenden)* en evaluación en la revista Matemáticas Aplicadas y Computación (Applied Mathematics and Computation) Sitio web: http://www.elsevier.com/wps/find/journaldescription.cws_home/522482/description.

- Monografía del trabajo de grado. Corresponde al presente documento, donde se describe el proceso seguido en el desarrollo del proyecto, los aportes más

sobresalientes, las conclusiones y las recomendaciones para el desarrollo de futuras investigaciones en el área.

Parte 2 – Contexto Teórico

5 BÚSQUEDA ARMÓNICA ORIGINAL

El algoritmo de la búsqueda armónica, propuesto por Zong Woo Geem y Kang Seo Lee [6] en el 2001, es un algoritmo meta heurístico, es decir, un algoritmo aproximado de propósito general que consiste en procedimientos iterativos que guían una heurística combinando de forma inteligente distintos conceptos para explorar y explotar adecuadamente el espacio de búsqueda [6]. HS simula el proceso de improvisación musical, en el cual los músicos buscan producir una armonía agradable determinada por el estándar estético auditivo [17]. Cuando un músico esta improvisando, el realiza una de las siguientes acciones:

1. Toca alguna melodía conocida que ha aprendido anteriormente.
2. Toca algo parecido a la melodía anteriormente mencionada, ajustándola poco a poco al tono deseado.
3. Compone una nueva melodía basándose en sus conocimientos musicales a partir de nuevas notas seleccionadas aleatoriamente.

Estas tres opciones formalizadas en [6], corresponden a los componentes del algoritmo: Uso de la memoria armónica, ajuste de tono y aleatoriedad, respectivamente.

En la improvisación musical, cada músico toca una nota dentro de un posible rango, de tal manera que forman un vector armónico. Si el conjunto de notas tocadas por los músicos son consideradas una buena armonía, esta es guardada en la memoria de cada músico, incrementando la posibilidad de hacer una buena

armonía la próxima vez. Del mismo modo en el proceso de optimización en ingeniería, cada variable de decisión inicialmente toma valores aleatorios dentro del rango posible, formando un vector solución. Si dicho vector, es decir, dicho conjunto de valores que lo conforman son una buena solución, esta es almacenada en la memoria de cada variable, aumentando la posibilidad de encontrar mejores soluciones en la siguiente iteración.

HS es un algoritmo meta heurístico basado en población, lo cual indica que un grupo de múltiples armonías pueden ser usadas en paralelo. Este recurso usado apropiadamente tiende a obtener un mejor rendimiento con una alta eficiencia. El proceso de improvisación que lleva a cabo el algoritmo HS para optimizar una función hace de este una poderosa herramienta de programación. Al no realizar cálculos matemáticos complejos, el procesamiento es mucho más rápido, lo cual hace que el tiempo de convergencia del algoritmo marque la diferencia entre los diferentes algoritmos meta heurísticos existentes [17].

Los algoritmos de búsqueda armónica han sido utilizados para una gran variedad de problemas de optimización reales, tales como la composición musical, enrutamiento de viajes, diseño estructural, rutas de vehículos, diseño de tuberías de calor para satélites, entre otros [18]. La ventaja de los algoritmos armónicos frente a otros algoritmos evolutivos se basa en sus características, las cuales lo identifican y hacen de él una poderosa herramienta de optimización. Entre estas se cuentan: no requiere cálculos complejos, obvia óptimos locales y puede manejar variables discretas y continuas [17, 19]. Estas características permiten diferenciarlo de los algoritmos genéticos y hacen de él una poderosa herramienta basada en cálculos matemáticos simples e improvisación.

5.1 DESCRIPCIÓN DEL ALGORITMO HS

Los pasos que indican el funcionamiento del algoritmo HS son los siguientes:

Paso 1. **Inicializar los parámetros del problema y los parámetros de HS**: El problema de optimización se define como minimizar (o maximizar) $f(x)$ tal que $LB_i < x_i < UB_i$ donde, $f(x)$ es la función objetivo, *x* es una solución candidata que consiste de N variables de decisión (x_i), y LB_i y UB_i son el límite de decisión más bajo y el más alto de cada variable, respectivamente. Los parámetros de HS se especifican en este paso. Estos parámetros son el tamaño de la memoria armónica (HMS), la tasa de consideración de la memoria armónica (HMCR), la tasa de ajuste del tono (PAR), el ancho de banda de ajuste del tono (BW) y el número de improvisaciones (NI).

Paso 2. **Inicializar la memoria armónica**: La memoria armónica inicial es generada desde una distribución uniforme en los rangos [LB_i, UB_i], donde $1 \leq i \leq N$. Esto se realiza de la siguiente manera: $x_i' = LB_i + r \times (UB_i - LB_i)$, donde j = *1,2,…, HMS* y $r \sim U(0,1)$. La variable r hace referencia a un número aleatorio y $U(0,1)$ a la función que genera el número aleatorio uniforme.

Paso 3. **Improvisar una nueva armonía**: El proceso de generación de una nueva armonía es llamado *improvisación*. El nuevo vector armónico, $x' = (x'_1, x'_2, \ldots, x'_N)$, se genera utilizando las siguientes reglas: consideración de la memoria, ajuste del tono y selección aleatoria. Este procedimiento se muestra en la Figura 1. En la línea 006, r es un número aleatorio uniforme entre 0 y 1, y el valor BW es un ancho de banda arbitrario de la distancia para variables de diseño continuas [8].

```
001 para cada i ∈ [1, N] hacer
002     si U(0,1) < HMCR entonces /*consideración de la memoria*/
003         inicio
004             x_i' = x_i^j, donde j ~ U(1, ..., HMS)
005             si U(0,1) ≤ PAR entonces /*ajuste del tono*/
006                 x_i' = x_i' + r × bw
007             fin_si
008         fin
009     sino /*selección aleatoria*/
010         x_i' = LB_i + r × (UB_i − LB_i)
```

```
011      fin_si
012  continuar_para
```

Figura 1. Improvisación de una nueva armonía.

Paso 4. **Actualizar la memoria armónica**: El vector armónico generado, $x' = (x'_1, x'_2, \ldots, x'_N)$, reemplaza la peor armonía almacenada en la memoria armónica si el *fitness* (o valor de aptitud del vector armónico actual, medido en términos de la función objetivo) es mejor que el de la peor armonía.

Paso 5. **Verificar el criterio de parada**: Terminar cuando el número máximo de improvisaciones (NI) se alcanza.

En general HS es poco sensible a los parámetros [17, 19], por esto, el algoritmo no requiere de una afinación exhaustiva de los parámetros para obtener buenas soluciones.

A pesar de lo anterior, es preciso destacar que los parámetros HMCR y PAR ayudan al método en la búsqueda de mejores soluciones globales y locales, respectivamente. PAR y BW tienen un profundo efecto en el desempeño del algoritmo y es por esto que el ajuste de estos dos parámetros es muy importante.

6 MEJORAS REALIZADAS AL ALGORITMO HS

HS es un algoritmo que ha sido aplicado en diversos problemas reales y sobre el cual se han realizado gran variedad de transformaciones, obteniendo resultados satisfactorios y motivando que la investigación sobre este algoritmo meta heurístico se haya incrementado en los últimos años [17, 19]. Entre las trasformaciones, o hibridaciones que ha sufrido HS se destacan las siguientes: la búsqueda armónica mejorada (IHS, Improve Harmony Search) [14] , la búsqueda armónica global (GBHS, Global-Best Harmony Search) [5]) y la nueva búsqueda armónica global (NGHS) [20]. Estos nuevos algoritmos han tenido en cuenta para su desarrollo las diversas técnicas de optimización existentes (Particle Swarm Optimization, PSO y mutación genética) y el ajuste dinámico de parámetros, para

disminuir el ruido en el algoritmo original, soportar mayor dimensionalidad y aumentar la precisión y el grado de convergencia en las soluciones, entre otras.

6.1 BÚSQUEDA ARMÓNICA MEJORADA (IHS)

La búsqueda armónica mejorada (*Improved Harmony Search* o IHS por sus sigla en inglés), es un algoritmo armónico propuesto en el año 2007 por Mahdavi et al [14], que emplea un novedoso método para la generación de nuevos vectores solución basado en el ajuste dinámico de los parámetros PAR (tasa de ajuste del tono) y BW (ancho de banda de la distancia), logrando con esto mejorar la precisión y la velocidad de convergencia. Ha sido probado en problemas de optimización en ingeniería y evaluado frente a otros algoritmos con éxito. Esta variante de la búsqueda armónica modifica el paso 3 del algoritmo original, el paso en el que se crea un nuevo armónico. PAR y BW cambian dinámicamente con el número de generaciones y se calculan con las fórmulas de la Figura 2.

$$PAR(gn) = PAR_{min} + \frac{(PAR_{max} - PAR_{min})}{NI} \times gn$$

Donde,

PAR	Tasa de ajuste del tono para cada generación
PAR_{min}	Tasa mínima de ajuste del tono
PAR_{max}	Tasa máxima de ajuste del tono
NI	Número de generaciones de vectores solución
gn	Número de generaciones

y

$$bw(gn) = bw_{max} \exp(c \times gn)$$

$$c = \frac{Ln\left(\frac{bw_{min}}{bw_{max}}\right)}{NI}$$

Donde,

$bw(gn)$	Ancho de banda de la distancia para cada generación
bw_{min}	Ancho de banda de la distancia mínimo
bw_{max}	Ancho de banda de la distancia máximo

Figura 2. Ecuaciones que rigen el cambio de los parámetros PAR y BW en el algoritmo IHS.

El parámetro PAR crece linealmente con el número de generaciones, mientras que bw decrece exponencialmente, como se puede apreciar en la Figura 3. Con este cambio en los parámetros, IHS logra mejorar el rendimiento de HS, ya que encuentra mejores soluciones tanto a nivel global como local.

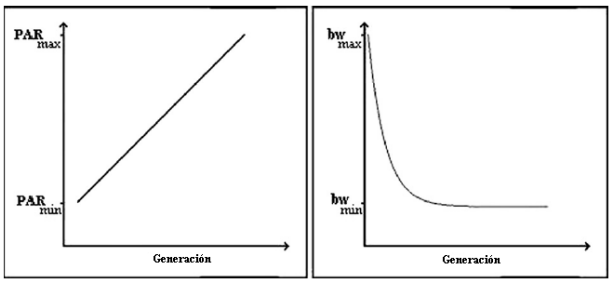


Figura 3. Variación de PAR y BW versus el número de generaciones (Tomada de [14]).

6.2 MEJOR BÚSQUEDA ARMÓNICA GLOBAL (GBHS)

Global-Best Harmony Search (GBHS) es un algoritmo de optimización estocástica propuesto en el año 2008 por Mahamed G.H. Omran y Mehrdad Mahdavi [5], el cual hibrida la búsqueda armónica original con el concepto de inteligencia de enjambre propuesto en PSO [5], donde un enjambre de individuos (llamados partículas) vuela a través del espacio de búsqueda. Cada partícula representa un candidato a la solución del problema de optimización. La posición de una partícula está influenciada por la mejor posición visitada por sí misma (es decir, su propia experiencia) y la posición de las mejores partículas en el enjambre (es decir, la experiencia de enjambre). GBHS modifica el paso de ajuste del tono en HS de modo que el nuevo armónico puede imitar el mejor armónico en la memoria armónica. Esto permite a GBHS trabajar eficientemente en problemas continuos y discretos. En general GBHS es mejor que IHS y HS, por ejemplo cuando se aplica a problemas de gran dimensionalidad y cuando hay presencia de ruido [5].

El GBHS tiene exactamente los mismos pasos que IHS con la salvedad de la modificación del paso 3 que corresponde a la improvisación de un nuevo armónico, el cual se modifica conforme a la Figura 4.

001 **para cada** $i \in [1, \mathrm{N}]$ **hacer**
002 **si** $U(0,1) < HMCR$ **entonces** */*consideración de la memoria*/*
003 **inicio**
004 $x_i^{'} = x_i^j$, donde $j \sim U(1, \ldots, HMS)$
005 **si** $U(0,1) \leq PAR(t)$ **entonces** */*ajuste del tono para la generación t (basado en IHS)*/*
006 $x_i^{'} = x_k^{best}$, *donde* $best$ *es el índice de la mejor armonía en* HM *y* $k \sim U(1, N)$
007 **fin_si**
008 **fin**

```
009       sino /*selección aleatoria*/
010             x_i' = LB_i + r × (UB_i − LB_i)
011       fin_si
012 continuar_para
```

Figura 4. Improvisación en el algoritmo de la mejor búsqueda armónica global (GBHS).

6.3 NUEVA BÚSQUEDA ARMÓNICA GLOBAL (NGHS)

La nueva búsqueda armónica global (novel global harmony search, NGHS) [20] propone una hibridación entre la búsqueda armónica, la optimización basada en enjambre y la mutación genética. Este algoritmo se propuso para resolver problemas de fiabilidad, término usado para indicar la probabilidad de que un sistema funcione correctamente (sea seguro y eficiente) sin violar ninguna restricción. De los diversos problemas de fiabilidad que existen se tomaron: los sistemas complejos (sistemas de programación no lineal entera y mixta), un sistema de protección de exceso de velocidad para una turbina de gas y un problema de fiabilidad de un sistema a gran escala.

NGHS y HS son diferentes en tres aspectos: en el paso 1, la tasa de consideración de la memoria armónica (HMCR) y el rango de ajuste de tono (PAR) son excluidos, y se incluye la probabilidad de mutación genética (p_m); en el paso 3, se modifica el paso de la improvisación por completo de modo que la nueva armonía imite a la mejor armonía global en la memoria armónica (HM) como se muestra en la Figura 5; y en el paso 4, se reemplaza la peor armonía x^{worst} con la nueva armonía $x^{'}$ aun sí $x^{'}$ es peor que x^{worst} [20].

```
001 para cada i ϵ [1, N] hacer
002       x_R = 2 × x_i^best − x_i^worst
003       si x_R > x_iU entonces
004             inicio
005                   x_R = x_iU
006             fin
007       sino
008             si x_R < x_iL entonces
009                   x_R < x_iL
010             fin_si
011       fin_si
012       x_i' = x_i^worst + r × (x_R − x_i^worst) // % actualización de la posición
013       si rand( ) ≤ p_m entonces
014             x_i' = x_iL + rand( ) × (x_iU − x_iL) // % mutación genética
```

```
015        fin_si
016  fin_para
```

Figura 5. Nueva Improvisación en el algoritmo de la nueva búsqueda armónica global (NGHS).

Esta propuesta se ha probado en optimización de problemas discretos y muestra una mejor convergencia y una mayor capacidad de exploración del espacio de solución que el algoritmo de búsqueda armónica original (HS) y que la búsqueda armónica mejorada (IHS).

7 MODELO EVOLUTIVO QUE APRENDE (LEM)

LEM (Learnable Evolution Model), o Modelo Evolutivo que Aprende es un modelo no darwiniano (no se basan únicamente en la teoría de evolución darwiniana) propuesto por Ryszard S. Michalski [21] en el año 2000, que hace parte de la computación evolutiva y que emplea aprendizaje de máquina con el objetivo de determinar la generación de nuevos individuos (soluciones candidatas al problema). Al usar el modo de aprendizaje de máquina, LEM puede determinar cuáles individuos de una población (o un conjunto de individuos de poblaciones antiguas) son superiores a otros en la realización de ciertas tareas designadas (dichas tareas suelen verse como las características primordiales que deben tener los individuos de la población). Estas razones, expresadas como hipótesis inductivas (es decir, características identificadas y consideradas como esenciales), se usan para la generación de nuevas poblaciones [22]. En el modo de evolución darwiniana, el algoritmo usa operaciones aleatorias o semi-aleatorias para la generación de nuevos individuos (usando técnicas de mutación y/o recombinación de la teoría de evolución darwiniana). Una característica notable de LEM es su capacidad para dar saltos cuánticos (saltos perspicaces que pueden realizar un ajuste) en la función objetivo [22]. El algoritmo de LEM actúa conforme al siguiente diagrama de flujo:

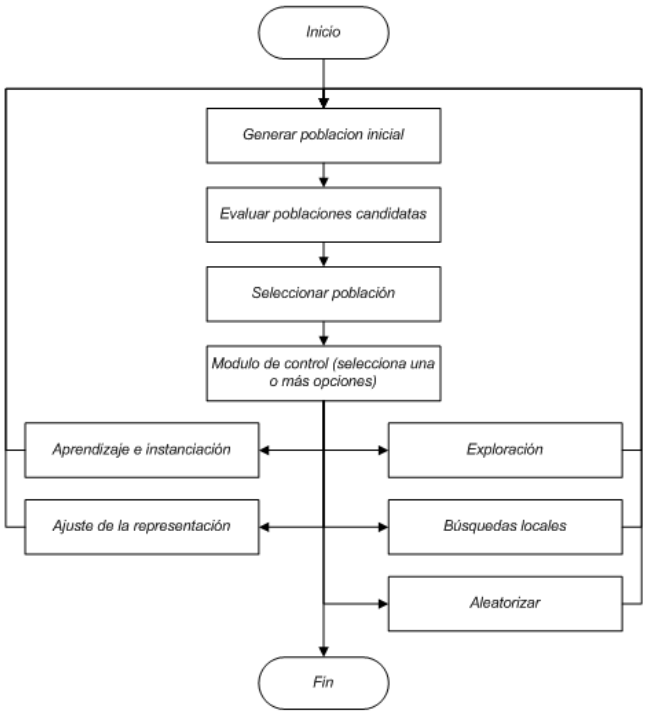


Figura 6. Diagrama de flujo de LEM3 (Tomada de http://www.mli.gmu.edu/projects/lem.html).

El Modulo de Control se encarga de realizar la selección de la acción o acciones que se ejecutarán para crear la nueva población. LEM utiliza aprendizaje de máquina o evolución darwiniana para crear las poblaciones según lo considere necesario. El módulo de aprendizaje e instanciación determina como se crearan las poblaciones de acuerdo a las hipótesis inductivas tomadas como referencia. LEM ha sido probado y en sus primeros estudios experimentales superó con éxito otros métodos de programación evolutiva, alcanzando en algunas ocasiones la velocidad en dos o más órdenes de magnitud el número de pasos evolutivos [22].

LEM tiene potencial para una amplia gama de aplicaciones, en particular, en ámbitos complejos o en problemas de optimización de búsqueda, diseño de ingeniería, diseño de fármacos, hardware evolutivo, ingeniería de software, la economía, la minería de datos, y la programación automática [22]. Algunas variaciones de LEM han sido desarrolladas con el objetivo de mejorar el rendimiento de la función objetivo, implementando diversos algoritmos genéticos

con el fin de suplir esta necesidad. Entre estas tenemos LEM1 y LEM2 desarrolladas en el año 2000 y LEM3 [23], desarrollada en el año 2006.

LEM1 se creó con el objetivo de probar algunas de las ideas propuestas para LEM en la generación de nuevas poblaciones de individuos. Fue concebido como un sistema preliminar, que emplea en modo de aprendizaje de máquina, aprendizaje tipo AQ15 (AQ15 es un programa que gradualmente aprende reglas de decisión a partir de ejemplos y contraejemplos de las decisiones, y, posiblemente, mediante reglas previamente aprendidas) [24], el cual se basa específicamente en el uso del cálculo atribucional[6] para facilitar la descripción del aprendizaje de objetos cuyos componentes se caracterizan por diferentes subconjuntos de atributos. En modo de evolución darwiniana, usa un algoritmo genético simple que se compone de operaciones como representación de valor real, mutación uniforme, selección proporcional y combinación uniforme para la creación de nuevas poblaciones de individuos.

LEM2 se creó con el objetivo de hacer mejoras al algoritmo LEM1. En modo de aprendizaje de máquina, LEM2 usa el aprendizaje AQ18[7], utilizado para datos

[6] Es una lógica y un sistema de representación definido por Ryszard S. Michalski. Es simple de implementar, aunque es un lenguaje de descripción altamente expresivo. Consta de una semántica y una sintaxis bien definidas que se asemejan al lenguaje natural. Algunos ejemplos son: [tamaño = mediano], condición atribucional que significa que el tamaño del objeto es considerado mediano; [color = rojo V blanco V azul], condición atribucional que significa que el color puede ser rojo o blanco o azul 21. Ryszard, S.M., *LEARNABLE EVOLUTION MODEL: Evolutionary Processes Guided by Machine Learning.* Mach. Learn., 2000. **38**(1-2): p. 9-40.

[7] AQ18 es un entorno que soporta la inducción natural, el aprendizaje de máquina y el descubrimiento del conocimiento. Por inducción natural se entiende una forma de inferencia inductiva que se usa para hacer las descripciones de los datos más naturales y comprensibles para las personas. Esta función se consigue mediante el empleo de un lenguaje de descripción de gran fuerza expresiva (el cálculo atribucional) Junto con el aprendizaje para la determinación de conjuntos de reglas de atribución de los ejemplos, o para mejorar progresivamente los conjuntos de reglas previamente aprendidas a través de nuevos ejemplos, AQ18 también incorpora un módulo de pruebas de conjunto de reglas (ATEST) y un módulo para la selección de los mejores

inconsistentes y ruidosos. En el modo de evolución darwiniana usa un algoritmo evolutivo [21].

Seis años después, en el 2006, Janusz Wojtusiak and Ryszard S. Michalski implementaron LEM3 [22], el cual, propone mejoras a LEM2 tomando como objetivo guiar a la computación evolutiva a través del aprendizaje de máquina. Esto se logra seleccionando en cada paso de la evolución individuos con el más alto y el más bajo rendimiento, categorizados en dos grupos H y L, con los cuales, luego se emplea el método de aprendizaje AQ21[8] para generar una hipótesis de la diferencia entre los dos grupos. La hipótesis es entonces instanciada de varias maneras para la generación de nuevos individuos de la población.

Los avances logrados en el campo de la programación evolutiva han sido de gran ayuda para la solución de problemas de optimización, puesto que, en el mundo real la toma de decisiones frente a los problemas no solo se basa en los cálculos matemáticos, estadísticos y físicos que se aplican con el uso de las ciencias sino también parte de la experiencia propia que se obtiene al realizar estas labores una y otra vez.

atributos de un determinado problema de aprendizaje (PROMISE) 25. Kenneth, A.K. and S.M. Ryszard, *An Adjustable Description Quality Measure for Pattern Discovery Usingthe AQ Methodology.* J. Intell. Inf. Syst. **14**(2-3): p. 199-216.

[8] Este método busca diferentes tipos de patrones en los datos y los representa en formas humano-orientadas semejantes a descripciones en lenguaje natural. Debido a esta última característica es conocido como un programa de inducción natural. Esta funcionalidad se consigue mediante el empleo de un lenguaje de representación de gran fuerza expresiva, Cálculo Atribucional, que combina los aspectos de la lógica proposicional, la lógica de predicados y la lógica de múltiples valores con el fin de apoyar el descubrimiento de patrones y de aprendizaje inductivo 22. Janusz, W. and S.M. Ryszard, *The LEM3 implementation of learnable evolution model and its testing on complex function optimization problems*, in *Proceedings of the 8th annual conference on Genetic and evolutionary computation*. 2006, ACM: Seattle, Washington, USA.

8 PRISM

Conocido como PRISM (prisma, en español), este algoritmo de aprendizaje de reglas fue introducido por Jadzia Cendrowska [16, 26] y aplicado como solución alternativa al problema de los lentes de contacto. En este problema se evaluó PRISM frente al algoritmo ID3 (algoritmo de aprendizaje de reglas basado en árboles de decisión, predecesor de C4.5).

PRISM tiene como objetivo inducir reglas de clasificación directamente en el conjunto de entrenamiento, mediante la selección de una combinación emparejada de atributo-valor para maximizar la probabilidad de cada clase de resultados de destino a su vez [27].

Los pasos básicos del algoritmo PRISM [27] para la inducción de reglas de clasificación se describen a continuación, asumiendo n ($n > 1$) posibles clases:

Para cada clase i desde 1 hasta n:

Paso 1. Calcular la probabilidad de la clase i para cada par atributo-valor

Paso 2. Seleccionar el par atributo-valor con la máxima probabilidad y cree un subconjunto del conjunto de entrenamiento que comprenda todas las instancias con la combinación seleccionada (esto se hace para todas las clases)

Paso 3. Repetir los pasos 1 y 2 para cada subconjunto hasta que se contenga solo instancias de la clase i. La regla inducida es entonces la conjunción de todos los pares atributo-valor seleccionados para la creación de este subconjunto.

PRISM es basado en técnicas empleadas por ID3, con la diferencia principal de que PRISM se concentra en encontrar únicamente los valores relevantes de los atributos, mientras que ID3 se preocupa por encontrar el atributo más relevante del conjunto, a pesar de que algunos valores de este atributo podrían ser

irrelevantes. ID3 divide un conjunto de entrenamiento en subconjuntos homogéneos sin hacer referencia a la clase de este subconjunto, mientras que PRISM debe identificar subconjuntos de una clase específica. Esto tiene como desventaja un ligero aumento del esfuerzo de cómputo, pero con la ventaja de una salida en forma de normas modulares en lugar de un árbol de decisión[16].

En el problema de los lentes de contacto, Cendrowska propuso un conjunto de 108 casos que indicaban las características de pacientes que habían sido diagnosticados con problemas en la vista que requerían el uso de lentes de contacto o pacientes con problemas en la vista para los cuales era inadecuado usarlos. Los atributos *age, specRx, astig* y *tears* corresponden a la edad del paciente, la prescripción de los lentes (miopía, hipermetropía), si el paciente posee astigmatismo y su tasa de producción de lágrimas, respectivamente. Las tres posibles clasificaciones para el conjunto de datos corresponden a lentes de contacto duros, lentes de contacto suaves y no adecuado para usar lentes de contacto. Dichos casos fueron usados como el conjunto de entrada del algoritmo PRISM para la generación de las reglas que permitían diagnosticar el estado de un nuevo paciente según fuera su caso. En [27] se muestra el proceso de este experimento realizado con un subconjunto de 24 reglas del conjunto usado por Cendrowska.

8.1 EJEMPLO DEL FUNCIONAMIENTO DE PRISM

El siguiente ejemplo muestra en detalle la operación del algoritmo básico de PRISM cuando es aplicado a un conjunto de valores que contiene 12 instancias. El conjunto de datos se presenta en la Tabla 1.

A	B	C	Clase
1	1	1	1
1	1	2	2
1	2	1	3
1	2	2	2
2	1	1	1

2	1	2	3
2	2	1	2
2	2	2	3
3	1	1	3
3	1	2	1
3	2	1	2
3	2	2	2

Tabla 1. Tabla de datos de prueba para el entrenamiento de PRISM (12 instancias).

En este ejemplo, PRISM es usado para generar las reglas de clasificación correspondientes a la clase 1 únicamente. Este proceso se realiza en forma cíclica haciendo uso de la probabilidad emparejada entre atributo-valor.

Clase 1: Primera regla

La Tabla 2 muestra la probabilidad de cada par atributo-valor para el primer conjunto de valores de entrenamiento (12 instancias).

Par atributo-valor	Frecuencia para la clase 1	Total frecuencia (12 instancias)	probabilidad
A = 1	1	4	0,25
A = 2	1	4	0,25
A = 3	1	4	0,25
B = 1	3	6	0,5
B = 2	0	6	0
C = 1	2	6	0,333333333
C = 2	1	6	0,166666667

Tabla 2. Probabilidad de cada par atributo-valor para el primer conjunto de valores de entrenamiento.

El par atributo-valor que maximiza la probabilidad de la clase 1 es *B = 1*. Por tal motivo, este es escogido. En el caso de que existan dos o más pares atributos-valor que tengan la misma probabilidad, se escoge uno de ellos aleatoriamente.

La regla inducida (incompleta) hasta ahora es: **if B = 1 then Clase = ?**

El subconjunto de entrenamiento generado a partir de esta regla es la Tabla 3.

A	B	C	Clase
1	1	1	1
1	1	2	2
2	1	1	1
2	1	2	3
3	1	1	3

3	1	2	1

Tabla 3. Subconjunto de entrenamiento a partir de la regla incompleta que se ha construido.

La Tabla 4 muestra la probabilidad de cada par atributo-valor para el anterior subconjunto de valores de entrenamiento.

Par atributo-valor	Frecuencia para la clase 1	Total frecuencia (12 instancias)	probabilidad
A = 1	1	2	0,5
A = 2	1	2	0,5
A = 3	1	2	0,5
C = 1	2	3	0,666666667
C = 2	1	3	0,333333333

Tabla 4. Probabilidad de cada par atributo-valor para el segundo conjunto de valores de entrenamiento.

La máxima probabilidad *(0,6666667)* ocurre cuando *C = 1.*

Regla inducida hasta este momento es: **if B = 1 and C = 1 then Clase = ?**

El subconjunto de entrenamiento generado por esta regla está en la Tabla 5.

A	B	C	Clase
1	1	1	1
2	1	1	1
3	1	1	3

Tabla 5. Subconjunto de entrenamiento a partir de la regla con dos condiciones.

La Tabla 6 muestra la probabilidad de cada par atributo-valor para el anterior subconjunto de valores de entrenamiento.

Par atributo-valor	Frecuencia para la clase 1	Total frecuencia (3 instancias)	Probabilidad
A = 1	1	1	1
A = 2	1	1	1
A = 3	0	1	0

Tabla 6. Probabilidad de cada par atributo-valor para el tercer conjunto de valores de entrenamiento.

La máxima probabilidad ocurre cuando *A = 1* y *A = 2.* Escogemos *A = 1* aleatoriamente. Hasta el momento podemos ver que la única variable por determinar es A. En este caso la primera regla que PRISM genera para el conjunto inicial es **if B = 1 and C = 1 and A = 1 then Clase = 1.**

Clase 1: Segunda regla.

Removemos todas las instancias que cumplen con la regla 1 del conjunto de entrenamiento. Este nuevo conjunto es representado en la Tabla 7.

A	B	C	Clase
1	1	2	2
1	2	1	3
1	2	2	2
2	1	1	1
2	1	2	3
2	2	1	2
2	2	2	3
3	1	1	3
3	1	2	1
3	2	1	2
3	2	2	2

Tabla 7. Tabla de Datos de Prueba para el entrenamiento de PRISM para la segunda regla (11 instancias).

La Tabla 8 muestra la probabilidad de cada par atributo-valor para el anterior conjunto de valores de entrenamiento (11 instancias).

Par atributo-valor	Frecuencia para la clase 1	Total frecuencia (11 instancias)	probabilidad
A = 1	0	3	0
A = 2	1	4	0,25
A = 3	1	4	0,25
B = 1	2	5	0,4
B = 2	0	6	0
C = 1	1	5	0,2
C = 2	1	6	0,166666667

Tabla 8. Probabilidad de cada par atributo-valor para el primer conjunto de valores de entrenamiento de la segunda regla.

La máxima probabilidad ocurre cuando *B = 1.*

Regla inducida hasta ahora: **if B = 1 then Clase = ?**

El subconjunto de entrenamiento generado por esta regla es la Tabla 9.

A	B	C	Clase
1	1	2	2
2	1	1	1
2	1	2	3
3	1	1	3
3	1	2	1

Tabla 9. Subconjunto de entrenamiento a partir de la segunda regla incompleta que se ha construido.

La Tabla 10 muestra la probabilidad de cada par atributo-valor para el anterior subconjunto de valores de entrenamiento.

Par atributo-valor	Frecuencia para la clase 1	Total frecuencia (5 instancias)	probabilidad
A = 1	0	1	0
A = 2	1	2	0,5
A = 3	1	2	0,5
C = 1	1	2	0,5
C = 2	1	3	0,333333333

Tabla 10. Probabilidad de cada par atributo-valor para el segundo conjunto de valores de entrenamiento de la segunda regla.

La máxima probabilidad ocurre cuando *A = 2, A = 3* y *C = 1.* Escogemos *A = 2* aleatoriamente.

Regla inducida hasta ahora: **if B = 1 and A = 2 then Clase = ?**

El subconjunto de entrenamiento generado por esta regla es la Tabla 11.

A	B	C	Clase
2	1	1	1
2	1	2	3

Tabla 11. Subconjunto de entrenamiento a partir de la segunda regla con dos condiciones.

La Tabla 12 muestra la probabilidad de cada par atributo-valor para el anterior subconjunto de valores de entrenamiento.

Par atributo-valor	Frecuencia para la clase 1	Total frecuencia (2 instancias)	probabilidad
C = 1	1	1	1
C = 2	0	1	0

Tabla 12. Probabilidad de cada par atributo-valor para el tercer conjunto de valores de entrenamiento de la segunda regla.

La máxima probabilidad ocurre cuando *C = 1*. Hasta el momento podemos ver que la única variable por determinar es C. En este caso la segunda regla que PRISM genera para el conjunto inicial es **if B = 1 and A = 2 and C = 1 then Clase = 1.**

Clase 1: Tercera regla.

Removemos todas las instancias que cumplen con la regla 2 del conjunto de entrenamiento. Este nuevo conjunto es representado en la Tabla 13.

A	B	C	Clase
1	1	2	2
1	2	1	3
1	2	2	2
2	1	2	3
2	2	1	2
2	2	2	3
3	1	1	3
3	1	2	1
3	2	1	2
3	2	2	2

Tabla 13. Tabla de Datos de Prueba para el entrenamiento de PRISM para la tercera regla (10 instancias).

La Tabla 14 muestra la probabilidad de cada par atributo-valor para el anterior conjunto de valores de entrenamiento (10 instancias).

Par atributo-valor	Frecuencia para la clase 1	Total frecuencia (10 instancias)	probabilidad
A = 1	0	3	0
A = 2	0	3	0
A = 3	1	4	0,25
B = 1	1	4	0,25
B = 2	0	6	0
C = 1	0	4	0
C = 2	1	6	0,16666667

Tabla 14. Probabilidad de cada par atributo-valor para el primer conjunto de valores de entrenamiento de la tercera regla.

La máxima probabilidad ocurre cuando *A = 3* y *B = 1.* Escogemos aleatoriamente *A = 3.*

Regla inducida hasta ahora: **if A = 3 then Clase = ?**

El subconjunto de entrenamiento generado por esta regla es la Tabla 15.

A	B	C	Clase
3	1	1	3
3	1	2	1
3	2	1	2
3	2	2	2

Tabla 15. Subconjunto de entrenamiento a partir de la tercera regla incompleta que se ha construido.

La Tabla 16 muestra la probabilidad de cada par atributo-valor para el anterior subconjunto de valores de entrenamiento.

Par atributo-valor	Frecuencia para la clase 1	Total frecuencia (4 instancias)	probabilidad
B = 1	1	2	0,5
B = 2	0	2	0
C = 1	0	2	0
C = 2	1	2	0,5

Tabla 16. Probabilidad de cada par atributo-valor para el segundo conjunto de valores de entrenamiento de la segunda regla.

La máxima probabilidad ocurre cuando *B = 1y C = 2.* Escogemos aleatoriamente *B = 1.*

Regla inducida hasta ahora: **if A = 3 and B = 1 then Clase = ?**

El subconjunto de entrenamiento generado por esta regla es la Tabla 17.

A	B	C	Clase
3	1	1	3
3	1	2	1

Tabla 17. Subconjunto de entrenamiento a partir de la tercera regla con dos condiciones.

La Tabla 18 muestra la probabilidad de cada par atributo-valor para el anterior subconjunto de valores de entrenamiento.

Par atributo-valor	Frecuencia para la clase 1	Total frecuencia (2 instancias)	probabilidad
C = 1	0	1	0
C = 2	1	1	1

Tabla 18. Probabilidad de cada par atributo-valor para el tercer conjunto de valores de entrenamiento de la tercera regla.

La máxima probabilidad ocurre cuando *C = 2.* Hasta el momento podemos ver que la única variable por determinar es C. En este caso la tercera regla que PRISM genera para el conjunto inicial **if A = 3 and B = 1 and C = 2 then Clase = 1**.

Al remover las instancias que cumplen con la regla 3 del conjunto de entrenamiento obtenemos un subconjunto de entrenamiento con 9 instancias para el cual todas las instancias de la clase 1 han sido removidas. En este caso, el algoritmo de PRISM ha terminado para la clase 1.

Entonces las reglas inducidas por el algoritmo PRISM para la clase 1 son:

if B = 1 and C = 1 and A = 1 then Clase = 1

if B = 1 and A = 2 and C = 1 then Clase = 1

if A = 3 and B = 1 and C = 2 then Clase = 1

Parte 3 – Survey de Algoritmos Armónicos

Entre las primeras metas del proyecto, se elaboró una revisión (Survey) sobre los Algoritmos Armónicos. Desde sus inicios en 2001, HS ha sido un tema investigado y aplicado activamente por la comunidad científica en diversas áreas [18]. El objetivo del survey, se centra en realizar una revisión de las investigaciones realizadas sobre él en los últimos 10 años con el fin de establecer similitudes, diferencias y encontrar ciertas características presentes en las diversas aplicaciones y mejoras de HS que sirvieron de referencia para la realización del algoritmo planteado en este proyecto. A continuación se da a conocer la metodología utilizada y el proceso de creación del survey.

9 METODOLOGÍA UTILIZADA PARA REALIZAR EL SURVEY

El proceso de elaboración del Survey se baso en la metodología documental propuesta por Carlos Serrano [28]. Esta metodología propuesta para la investigación documental expone actividades y técnicas que se adaptaron para el desarrollo del Survey.

En el inicio del desarrollo del Survey fue necesario recopilar información que permitiera construir el estado del arte, en la cual se tuvo en cuenta las diferentes fuentes bibliográficas disponibles (artículos y libros), las cuales fueron recopiladas y organizadas mediante el uso de EndNote, un software de gestión bibliográfica.

Realizada la recopilación de referencias, el paso a seguir fue la clasificación e identificación de los temas referentes al tema central del Survey (Algoritmo HS),

con el fin de organizar el contenido del mismo y fijar el tiempo que este tomaría en su desarrollo. La base conceptual del Survey se definió en cuatro límites de la siguiente manera:

- Relación con el tiempo: Para el desarrollo del Survey se determinó un cronograma de actividades en el cual se definieron los tiempos de ejecución de cada fase de construcción (ver en la sección 9.1 las fases que se llevaron a cabo) y el orden de cada una de ellas. El cronograma para la realización del Survey, se planteo desde el anteproyecto con una duración de tres (3) meses.

- Relación con el material bibliográfico: Para el desarrollo del Survey se tuvo en cuenta las referencias que se publicaron entre los años 2001-2010. No fue definida una cantidad exacta de referencias y algunas referencias de años anteriores (1984, 1994 y 1995) fueron incluidas como referentes de algunas investigaciones planteadas recientemente.

- Relación con el espacio: El espacio definido para realizar la investigación fue la internet puesto a que el acceso a las diferentes referencias se hizo a través de las bases de datos de Science Direct, IEEE, ACM, Springer Link, y mediante el uso de los buscadores web como Google, Yahoo! y Bing.

- Relación con los integrantes: El número de integrantes que participaron en esta investigación corresponde a tres personas: dos estudiantes, los cuales desarrollaron la investigación y un director que tuvo la tarea de dirigir y revisar dicho proceso.

9.1 FASES PARA CONSTRUIR EL ESTADO DEL ARTE

En una investigación documental se definen 5 fases [28] para la construcción del estado del arte. A continuación se realiza una breve descripción de cada una de ellas y su aplicación en la elaboración del Survey.

- Fase Preparatoria: Inicia con la recopilación de las referencias y su organización. En el proyecto se uso el software de gestión bibliográfica EndNote, que tiene entre su funcionalidad, el almacenamiento de referencias bibliográficas (incluido el documento de la referencia) y la creación de librerías personalizadas. EndNote soporta gran cantidad de formatos de citas bibliográficas, los cuales pueden ser usados gracias a su integración con el programa Microsoft Office Word.

- Fase Descriptiva: Comprende la revisión en detalle de cada referencia y la creación de la ficha descriptiva correspondiente. La ficha tiene información sobre el tema, el autor, entre otros.

- Fase Interpretativa por Núcleos Temáticos: Se realizó el análisis de las referencias según el núcleo temático con el fin de obtener información relevante para el desarrollo del Survey.

- Fase de Construcción Teórica Global: En esta fase se realiza un balance del conjunto de resultados del estudio partiendo de la interpretación por núcleos temáticos, esto con el fin de identificar vacios, limitaciones y logros obtenidos en la temática estudiada. Esta fase se realizó mediante un sistema de entregas de cada subtema del Survey, los cuales eran revisados por el director de la investigación y refinados hasta obtener una versión final para incluirlo en el Survey.

- Fase de Extensión y Publicación: La investigación representada por el Survey fue presentada de manera escrita en la revista Avances en Sistemas e Informática de la Universidad Nacional de Colombia sede Medellín (RASI). Se espera su publicación en el mes de marzo de 2011.

9.2 FACTORES E INDICADORES PARA EL ANÁLISIS DE DOCUMENTOS

Para realizar la revisión adecuada de cada referencia se definió un conjunto de factores e indicadores adecuados para esta investigación. Estos fueron seleccionados de [28] y se presentan en la Tabla 19.

FACTORES	INDICADORES
1. Aspectos Formales (Características del autor y del documento)	1.1. Autor 1.2. Titulo del Documento 1.3. Tipo de Material
2. Asunto Investigado (Objeto, fenómeno o proceso de estudio)	2.1. Tema Central 2.2. Núcleo Temático 2.3. Problema
4. Propósito (Fin buscado por el autor con los resultados de su investigación)	4.1. Objetivo General 4.2. Objetivos Específicos
5. Enfoque (Referente disciplinar y conceptual desde el cual se analiza el objeto de estudio)	5.1. Referentes teóricos 5.2. Conceptos principales
6. Metodología (Conjunto de procedimientos y estrategias utilizadas para la formulación, el diseño y la ejecución del proceso de investigación)	6.1. Técnicas
7. Resultados (Los señalados por el autor en el documento como producto de su investigación)	7.1. Conclusiones
8.- Observaciones (Las que hace quien efectuó el análisis del documento)	8.1. Comentarios

Tabla 19. Factores e Indicadores de referencia para el análisis de documentos. (Tomada de [28]).

9.3 FICHAS DE REFERENCIA

El modelo de investigación documental de Carlos Serrano [28] plantea el uso de cinco tipos de fichas utilizadas para el desarrollo del estado del arte. Las fichas de referencia utilizadas en el desarrollo del Survey fueron:

- Ficha descriptiva: Contiene los aspectos formales del artículo, y la información relevante de la referencia asociada. Fueron muy útiles en la formación del estado del arte y la base conceptual del Survey. Los componentes de esta ficha se presentan en la Tabla 19.

- Reseña Bibliográfica: Contienen los aspectos formales del documento y resumen del documento ("abstract"/síntesis).}

- Ficha de interpretación por núcleo temático: Construidas a partir de las fichas descriptivas, estas fichas contienen la identificación del núcleo temático, síntesis y observaciones.

Para la construcción de las fichas anteriores, se inició con la creación de la librería mediante el uso del software de gestión bibliográfica EndNote. La librería llamada Harmony Search (Búsqueda armónica) contiene la clasificación de las referencias en 10 subtemas: Algoritmos Genéticos, Algoritmos HS, Aplicaciones en Ciencias de la Computación, Otras aplicaciones, Aplicaciones en Ingeniería Civil, Aplicaciones en Ingeniería Eléctrica, Aplicaciones en Ingeniería Mecánica, Aplicaciones en Medicina, Aplicaciones en Economía y Aplicaciones en Transporte. En la Tabla 20 se relaciona el número de las referencias recopiladas por cada subtema.

SUBTEMA	NUMERO DE REFERENCIAS	LIBROS	TESIS	ARTÍCULOS	OTROS
Algoritmos Genéticos	19	7	0	11	1
Algoritmos HS	16	0	0	16	0

Aplicaciones en Ciencias de la Computación,	19	0	0	19	0
Aplicaciones en Ingeniería Civil	36	0	0	36	0
Aplicaciones en Ingeniería Eléctrica	11	0	0	11	0
Aplicaciones en Ingeniería Mecánica	19	0	0	19	0
Aplicaciones en Medicina	5	0	0	5	0
Aplicaciones en Economía	15	0	0	15	0
Aplicaciones en Transporte	4	0	0	4	0
Otras aplicaciones	10	0	0	10	0
TOTALES	154	7	0	146	1

Tabla 20. Clasificación de referencias recopiladas para realizar el Survey.

Se recopilaron un total de 154 referencias, para las cuales se realizó una revisión general con base en el titulo y en su contenido. Se clasificaron teniendo en cuenta los criterios descritos en las fichas de referencia y a continuación se realizó una revisión por referencia teniendo en cuenta las siguientes instrucciones tomadas de [28]:

- Realizar una revisión general de la literatura, con el fin de detectar la bibliografía que podría ser útil para conformar el banco de documentos, identificar los autores y grupos de investigación más importantes en la temática respectiva.
- Ubicar preferiblemente documentos resultantes de trabajos de investigación.
- Identificar las más importantes fuentes secundarias de información relacionadas con el área temática respectiva, ya que contienen compilaciones, resúmenes y listados de fuentes primarias.

- Detectar y ubicar la mayor cantidad posible de fuentes primarias, teniendo en cuenta la fecha de publicación y que hayan sido escritas o editadas por un experto en el tema.

10 DESCRIPCIÓN DEL SURVEY

Con la identificación de los núcleos temáticos, se realizó la organización estructural de la base conceptual o cuerpo del contenido. Los contenidos definidos para el Survey se muestran en la Figura 7.

Siguiendo la metodología planteada para el desarrollo del Survey, se tomaron 74 unidades de análisis, las cuales fueron escogidas debido al aporte investigativo que representaban para el tema central y los subtemas del Survey.

10.1 REFERENCIAS POR AÑO DE PUBLICACIÓN

La clasificación de la referencia se hizo de la siguiente manera:

- Se clasificaron las referencias utilizadas en el desarrollo del Survey por año de publicación. En la Figura 8 observamos que el Survey está compuesto en su gran mayoría de referencias actuales, que corresponden a los años 2008, 2009 y 2010 lo cual equivale al 72% del total de las referencias utilizadas.

Adicionalmente se realizó la clasificación de referencias por número de publicaciones de cada autor. En las figura 8 y 9 se muestra la cantidad de referencias por autor, en la cual se destaca Zong Woo Geem, autor del algoritmo HS (tema central del -Survey) y que ha dedicado gran parte de su trabajo de investigación a mejorarlo.

Contenido

Figura 7. Tabla de Contenido del Survey “Una revisión de la Búsqueda Armónica”.

También se analizó el número de autores por país de origen, con el objetivo de identificar los países que más desarrollo investigativo han producido con respecto al tema central. Podemos ver en la figura 11 que los países más interesados en el tema son Estados Unidos con 26 publicaciones (32.1%) e Irán con 16 publicaciones (19.8% del 100% del total de publicaciones por país, 81 publicaciones). Es necesario entender que un articulo puede tener varios autores de distintos países, por lo cual, dicho artículo se registra en varias países y cuenta una vez para cada autor.

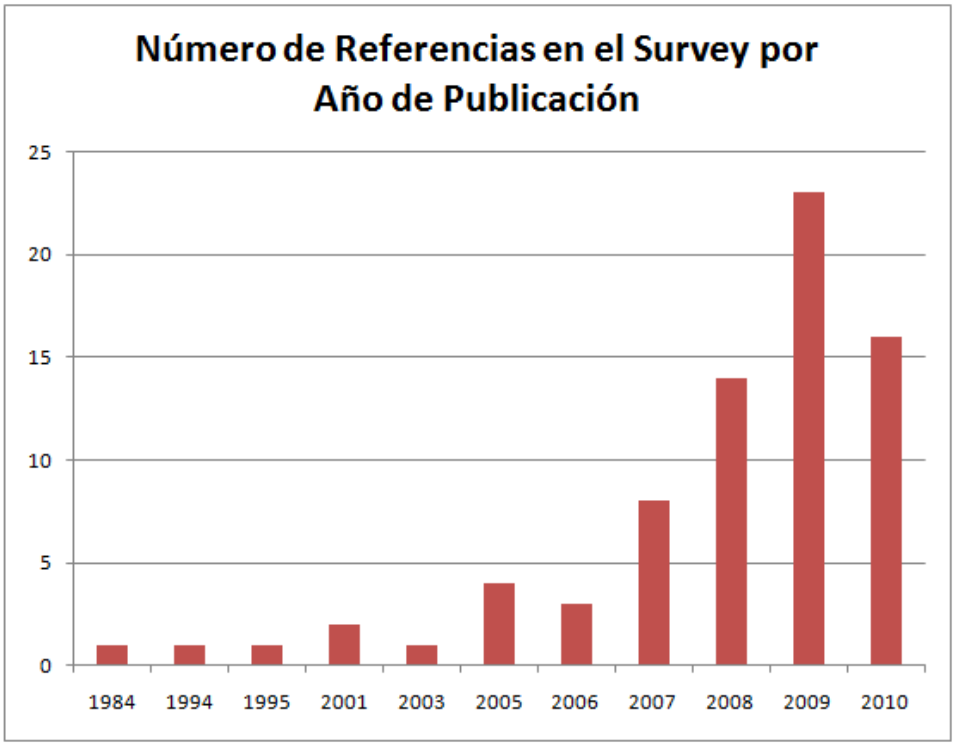


Figura 8. Número de referencias en el Survey por año de publicación.

Figura 9. Número de publicaciones por autor (parte 1).

Figura 10. Número de publicaciones por autor (parte 2).

Figura 11. Número de publicaciones por país.

Parte 4 – Algoritmo GHS+LEM

En este proyecto se propuso como segundo objetivo especifico el modelado y desarrollo del algoritmo denominado inicialmente con la sigla GBHS-LEM (Global Best Harmony Search – Learnable Evolutionary Models), y que en adelante denominaremos GHS+LEM.

11 ALGORITMO PROPUESTO: GHS+LEM

Inspirados en el concepto de Learnable Evolution Model (LEM) propuesto por Michalsky [29], en este artículo se propone una nueva variación del algoritmo GBHS. En LEM se emplean técnicas de aprendizaje de máquina para generar nuevas poblaciones adicionalmente al método darwiniano, aplicado en la computación evolutiva, el cual se basa en mutación, combinación y selección natural. Este método puede determinar cuáles individuos de una población (o un conjunto de individuos de poblaciones anteriores) son mejores a otros en la realización de ciertas tareas. Estas razones, expresadas como hipótesis inductivas, se usan para la generación de nuevas poblaciones. Luego, cuando el algoritmo se ejecuta en modo de evolución darwiniana, usa operaciones aleatorias o semi-aleatorias para la generación de nuevos individuos (usando técnicas de mutación y/o recombinación tradicionales).

11.1 PASOS DEL PROCESO LEM

El proceso LEM puede resumirse en los siguientes pasos:

1. Generar una población. Este paso se ejecuta para crear la población inicial, ya sea aleatoriamente o de acuerdo a un método.

2. Ejecutar el modo de aprendizaje de Máquina. Consta de los siguientes pasos:

   A. Derive extrema: seleccionar dos grupos de la población actual, uno de Alto rendimiento (H-Group) y otro de bajo rendimiento (L-Group), basados en la evaluación de la función objetivo.

   B. Crear una hipótesis: aplicar un método de aprendizaje de máquina para crear una descripción del H-group que lo diferencie del L-group, opcionalmente se puede hacer la consideración de poblaciones pasadas.

   C. Generar la nueva población: Generar nuevos individuos a través de las reglas aprendidas de la descripción de los dos grupos.

   D. Ir al paso A y repetir hasta que se alcance el criterio de terminación del proceso de aprendizaje de máquina.

3. Ejecutar el modo de aprendizaje Darwiniano. Se aplica un método evolutivo de tipo darwiniano, es decir, se aplica algún tipo de mutación, cruce (opcional) y los operadores de selección para generar una nueva población. Continuar este modo hasta que la condición de terminación evolución darwiniana modo se cumple.

4. Alternar de modo hasta que el criterio de finalización sea alcanzado.

El modo de aprendizaje de maquina propuesto en GHS+LEM hace uso de una variación del algoritmo de inferencia de reglas PRISM. A continuación se explica su funcionamiento.

11.2 PROCESO DE APRENDIZAJE DE MAQUINA CON PRISM

El proceso de aprendizaje de máquina en el nuevo algoritmo emplea un variación del algoritmo PRISM propuesto por Cendrowska [16, 30]. PRISM toma como

entrada un conjunto de entrenamiento dado como un archivo de conjuntos ordenados de valores de atributos, cada uno determinado por una clasificación. La información sobre los atributos y clasificaciones (nombre, número de valores posibles, lista de posibles valores, etc.) es la entrada de un archivo por separado al inicio del programa, y los resultados se emiten como reglas individuales para cada una de las clasificaciones que figuran en términos de los atributos descritos.

La aproximación que se utiliza del algoritmo PRISM en la propuesta, está destinada a imitar las decisiones simples que manejan los algoritmos de la familia armónica, y puede trabajar tanto con variables continuas como discretas. Para tal fin se cuenta con un conjunto de reglas conjuntivas ($P \leftarrow R_1 \wedge R_2 \wedge \ldots \wedge R_n$), que delimitan las regiones alrededor de las cuales hay una mayor posibilidad de encontrar un mejor valor para cada x_i (por ejemplo, $LV_{x_i} \leq x_i \leq HV_{x_i}$, donde *LV* y *HV* son los límites inferior y superior de las reglas para el valor x_i). Con la conjunción de las reglas (R) para cada dimensión se limita el espacio de búsqueda a las regiones con más posibilidades de generar un óptimo global. El algoritmo de inferencia de reglas se ejecuta por primera vez inmediatamente después de creada la memoria armónica inicial. Los pasos del algoritmo de inferencia de reglas, se resumen a continuación:

A. Se escoge de la memoria armónica actual la población de alto rendimiento y la población de bajo rendimiento siguiendo la siguiente fórmula

$$Hgroup = P_{actual}(1, \ldots, i),$$
$$Lgroup = P_{actual}(HMS - i, \ldots, HMS),$$

donde i = HLGS, HMS es el tamaño de la memoria armónica (HM) y HLGS es el tamaño de los grupos de alto y bajo rendimiento

B. Se genera una matriz para cada clase *i* perteneciente a cada grupo. En esta matriz se almacena el valor del atributo en esa posición y el *fitness* correspondiente de la siguiente forma, si es *Hgroup* se asigna 1 y en caso contrario se asigna 0, después se ordena con respecto al valor de su atributo de menor a mayor.

C. Se calcula la probabilidad de la ocurrencia de cada atributo.

D. Se selecciona el atributo con la mayor probabilidad de ocurrencia y se agrega a la lista de reglas *P*.

E. Se repiten los pasos B, C y D hasta que se evalúen todos los atributos de cada grupo, la regla resultante es de tipo $P \rightarrow Q$, donde *P* es una conjunción de las reglas que tienen la mayor probabilidad para cada atributo y Q corresponde a la clase 1 (grupo de alto rendimiento).

12 PASOS DEL ALGORITMO GHS+LEM

La nueva propuesta denominada Mejor búsqueda armónica global usando modelos evolutivos que aprenden (GHS+LEM) se define en los siguientes pasos:

a) *Inicializar los parámetros del problema y los parámetros de GHS*

Este paso es similar al propuesto en GHS y agrega tres parámetros que se explican más adelante, a saber: la tasa de actualización de las reglas (RRU), el tamaño de los grupos de alto y bajo rendimiento (HLGS) y la tasa de consideración de reglas (RCR).

b) *Inicializar la memoria armónica*

Se ejecuta el proceso de inicialización propuesto en HS sin ningún cambio.

c) *Ejecutar el proceso de inferencia de reglas por primera vez*

Se ejecuta por primera vez el proceso de inferencia de reglas con base en la memoria armónica inicial y el tamaño de los grupos de alto y bajo rendimiento (HLGS). Este parámetro (HLGS) indica el tamaño de los grupos de alto desempeño y los grupos de bajo desempeño, este valor debe ser $\leq \lfloor HMS/2 \rfloor$.

d) *Improvisar la nueva armonía*

En este paso se introduce el uso de las reglas generadas en el paso anterior para la definición de los valores de la dimensión del nuevo improviso (ver Figura 12). Lo anterior se ejecuta basado en el parámetro denominada tasa de consideración de reglas (*RCR*). Este parámetro (RCR) decide en que porcentaje de las veces se utilizará las reglas, de lo contrario se ejecutará el método tradicional de generación aleatoria basado en el espacio general de búsqueda (original en HS).

for each $i \in [1, \mathrm{N}]$ **do**
  if $U(0,1) < HMCR$ **then** */*memory consideration*/*
    begin
      $x_i^{'} = x_i^j$, where $j \sim U(1, \ldots, HMS)$
      if $U(0,1) \leq PAR(t)$ **then** */*pitch adjustment with PSO*/*
        $x_i^{'} = x_k^{best}$, where $best$ is the index of the best harmony in HM and $k \sim U(1, N)$
      end_if
    end
  else
    if $U(0,1) < RCR$ **then** /*rule consideration rate*/
      $x_i^{'} = U(LB_i^{best}, UB_i^{best})$, where *best* is the best set of rules for x_n
    else /* random selection */
      $x_i^{'} = LB_i + r \times (UB_i - LB_i)$
    end_if
  end_if
done

Figura 12. Improvisación en el algoritmo GHS+LEM.

e) Actualizar la memoria armónica

La nueva armonía generada, $x' = (x'_1, x'_2, \ldots, x'_N)$ reemplaza la peor armonía almacenada en la memoria armónica sólo si el *fitness* (o valor de aptitud de la nueva armonía, medido en términos de la función objetivo) es mejor que el de la peor armonía.

f) Verificar el criterio de actualización de reglas

Se realiza a través del parámetro RRU, el cual especifica en que porcentaje de las veces se deben actualizar las reglas. Si un número aleatorio generado uniformemente entre 0 y 1 es menor que el valor de RRU, se ejecuta el proceso de inferencia de reglas nuevamente (ver Figura 13).

```
if U(0,1) < RRU then /*rule update*/
    Ejecutar el proceso de inferencia de reglas
end_if
```

Figura 13. Proceso de actualización de reglas en GHS+LEM.

g) *Verificar el criterio de parada*:

Termina la ejecución del algoritmo cuando el número máximo de improvisaciones (NI) se alcanza, de lo contrario los pasos 5.4, 5.5 y 5.6 se repiten. Este paso no sufre cambios con respecto al original.

13 CODIGO FUENTE DEL ALGORITMO EN C#

En las Figuras 14 y 15 se muestra el código fuente del proceso que realiza el algoritmo GHS+LEM; en las Figuras 16 y 17 se muestra el código fuente de la clase Harmony Solution, la cual almacena la memoria armónica del algoritmo; en las figuras 18, 19 y 20 se muestra el código de la clase LEM, que contiene el proceso de inferencia de reglas y en la Figura 21 se muestra el código fuente de la clase Rule la cual contiene la definición de una regla.

```
using System.Collections.Generic;
using System.IO;
using Funciones;
using HarmonySearch.HarmonyFamily;
using prism;

namespace HarmonySearch.GlobalBestHarmonySearchLEM
{
    /// <summary>
    /// Implementa la funcionalidad del algoritmo Global-best Harmony Search + Learnable Evolution Models
    /// </summary>
    public class Gbhslem : HarmonyFamily.HarmonyFamily
    {
        /// <summary>
        /// Procedimiento principal del algoritmo Global-best Harmony Search + Learnable Evolution Models
        /// </summary>
        /// <param name="nombreFuncion">The nombre de la funcion.</param>
        /// <param name="args">Argumentos opcionales para la funcion.</param>
        public override void Run(Function nombreFuncion, params object[] args)
        {
            // Se crea una nueva memoria armonica
            Hm = new HarmonySolution[Constants.Hms];
            // Se crea un vector de tipo memoria armonica
            NewHarmony = new HarmonySolution();
            // PASO 1: Coloca las soluciones iniciales en la Memoria Armónica(Hm)
            InitializeHarmonyMemory(nombreFuncion);
            // Se encuentran las reglas de acuerdo a la memoria armonica inicial
            var rules = EncontrarReglas(Constants.Hlgs);
            // Se inicializa el ciclo
            var ciclo = 1;
            // Mientras no se alcance el numero total de iteraciones
            while (ciclo <= Constants.Ni)
            {
                // PASO 2: Improvisar un nuevo vector armónico
                NewHarmony.Improvise(Hm, ciclo, rules, nombreFuncion, args);
                // PASO 3: Actualizar la Memoria Armónica
                UpdateHm();
                // Se crea un factor aleatorio que decide el criterio de actualizacion de reglas
                var ranLem = Constants.Rand.NextDouble();
                // Si el valor es menor que el factor
                if (ranLem <= Constants.Rru)
                {
                    // Se actualizan las reglas
                    rules = EncontrarReglas(Constants.Hlgs);
                }
                // Se aumenta el ciclo
                ciclo++;
                .
                {
                    Hm[i] = new HarmonySolution();
                    Hm[i].AllDimensionsRandomlyGenerated(function);
                }
            }
            /// <summary>
            /// Procedimiento para encontrar las reglas de acuerdo a la memoria armonica
            /// </summary>
            /// <param name="tamanoMuestra">El tamaño de la muestra para generar las reglas</param>
            /// <returns>La lista de reglas</returns>
            public List<Rule> EncontrarReglas(int tamanoMuestra)
            {
                var lemGenerator = new LEM(Constants.LimiteInferior, Constants.LimiteSuperior, Constants.Bw);
                var reglaFinal = new List<Rule>();
                var reglas = ConvertirHm(Hm, tamanoMuestra, Constants.LimiteInferior, Constants.LimiteSuperior,
                                         Constants.Bw);
                var posicion = new List<int>();
                var p = 0;
                for (var i = 0; i < Constants.NumberOfDimensions; i++)
                {
                    posicion.Add(p++);
                }
                lemGenerator.GenerateFullRule(reglas, posicion, reglaFinal);
                reglaFinal.Sort();

                return reglaFinal;
            }
```

Figura 14. Código Fuente de la clase GHS+LEM (Parte 1).

```
84          /// <summary>
85          /// Convierte un vector tipo HarmonySolutionFamily en uno de tipo List para
86          /// poder trabajar en la creacion de reglas.
87          /// </summary>
88          /// <param name="hm">El vector con las solucion armonica</param>
89          /// <param name="tamanoMuestra">El tamaño de la muestra para seleccionar</param>
90          /// <param name="limiteInferior">El limite inferior de la funcion actual</param>
91          /// <param name="limiteSuperior">El limite superior de la funcion actual</param>
92          /// <param name="bw">El bandwidth por defecto para la generacion de reglas</param>
93          /// <returns>La lista ordenada de tipo HV</returns>
92          /// <param name="bw">El bandwidth por defecto para la generacion de reglas</param>
93          /// <returns>La lista ordenada de tipo HV</returns>
94          public List<HV> ConvertirHm(HarmonySolutionArray[] hm, int tamanoMuestra, double limiteInferior,
95                                      double limiteSuperior, double bw)
96          {
97              var convertedValues = new List<HV>();
98              foreach (HarmonySolutionArray harmonySolutionFamily in hm)
99              {
100                 var hvTemp = new HV();
101                 double[] arrayTemp = harmonySolutionFamily.Variables;
102                 foreach (double d in arrayTemp)
103                 {
104                     hvTemp.atributos.Add(d);
105                 }
106                 hvTemp.atributos.Add(harmonySolutionFamily.Fitness);
107                 convertedValues.Add(hvTemp);
108             }
109             convertedValues.Sort();
110             foreach (var convertedValue in convertedValues)
111             {
112                 convertedValue.atributos.RemoveAt(convertedValue.atributos.Count - 1);
113             }
114             var reducedArray = new List<HV>();
115             for (var i = 0; i < tamanoMuestra; i++)
116             {
117                 var hvTemp = new HV {atributos = convertedValues[i].atributos, fitness = 1};
118                 reducedArray.Add(hvTemp);
119             }
120             for (var i = convertedValues.Count - tamanoMuestra; i < convertedValues.Count; i++)
121             {
122                 var hvTemp = new HV {atributos = convertedValues[i].atributos, fitness = 0};
123                 reducedArray.Add(hvTemp);
124             }
125             return reducedArray;
126         }
127     }
128 }
```

Figura 15. Código Fuente de la clase GHS+LEM (Parte 2).

```
1   using System;
2   using System.Collections.Generic;
3   using System.IO;
4   using System.Linq;
5   using System.Text;
6   using System.Xml;
7   using HarmonySearch.HarmonyFamily;
8   using prism;
9   using Funciones;
10
11  namespace HarmonySearch.GlobalBestHarmonySearchLEM
12  {
13      /// <summary>
14      /// La memoria armonica para GHS+LEM
15      /// </summary>
16      internal class HarmonySolution : HarmonySolutionArray
17      {
18          #region Improvisacion
19          /// <summary>
20          /// Proceso de improvisacion del algoritmo GHS+LEM,
21          /// </summary>
22          /// <param name="memoriaArmonica">La memoria armonica</param>
23          /// <param name="ciclo">El numero del ciclo</param>
24          /// <param name="reglas">Las reglas para la generacion de reglas</param>
25          /// <param name="function">La funcion de optimizacion con la que se prueba el algoritmo</param>
26          /// <param name="args">Datos para las pruebas de variacion de RCR</param>
27          public override void Improvise(HarmonySolutionArray[] memoriaArmonica,
28                                         int ciclo,
29                                         List<Rule> reglas,
30                                         Function function,
31                                         params object[] args)
32          {
33              // La posicion de la mejor armonia en la memoria armonica
37              // Para cada una de las dimensiones del problema
```

Figura 16. Código Fuente de la clase Harmony Solution (Parte 1).

```
            for (int i = 0; i < Constants.NumberOfDimensions; (i)++)
            {
                //Valor aleatorio para el manejo de PAR, BW y HCMR segun el algoritmo
                var ran = Constants.Rand.NextDouble();
                // operador para consideración de memoria
                if (ran <= Constants.Hmcr)
                {
                    //Se genera un valor aleatorio para la memoria Armonica
                    int randHmv = Constants.Rand.Next() % Constants.Hms;
                    //Se obtiene el valor de la memoria armonica en la posicion i
                    Variables[i] = memoriaArmonica[randHmv].GetVariables(i);
                    //Se compara el operador del ajuste del Pitch
                    if (Constants.Par < parGn)
                    {
                        //Se selecciona el mejor valor en la posicion i de la memoria armonica .
                        Variables[i] =
                            memoriaArmonica[mejorPosicion].GetVariables(Constants.Rand.Next(0, Constants.NumberOfDimensions));
                        // Controlo que los datos no se salgan del limite
                        if (Variables[i] < Constants.LimiteInferior) Variables[i] = Constants.LimiteInferior;
                        if (Variables[i] > Constants.LimiteSuperior) Variables[i] = Constants.LimiteSuperior;
                    }
                }
                else
                {
                    // Se crea un nuevo randomico que manejara el valor RCR
                    var rcrRandomic = Constants.Rand.NextDouble();
                    // Si el valor generado es menor que la constante RCR
                    if (rcrRandomic <= Constants.Rcr)
                                                                Constants.Rand.Next(0, Constants.NumberOfDimensions)));
                    else
                        // Se genera un valor aleatorio
                        OneDimensionRandomlyGenerated(i);
                }
            }
            //Evalua la funcion para encontrar el fitness
            EvaluateFunction(function);
        }
        #endregion

        #region Methods
        /// <summary>
        /// Genera un valor entre los valores ruleMaxValue y ruleMinValue de la regla
        /// </summary>
        /// <param name="regla">La regla que se implementa</param>
        /// <returns>El valor entre los limites de la regla</returns>
        public double GenerarConReglas(Rule regla)
        {
            var newValue = regla.ruleMinValue +
                           ((regla.ruleMaxValue - regla.ruleMinValue) *
                            Constants.Rand.NextDouble());
            return newValue;
        }
        /// <summary>
        /// Encuentra la regla en la posicion real en el vector
        /// </summary>
        /// <param name="reglas">La lista de reglas generadas en memoria actualmente</param>
        /// <param name="posBuscada">La posicion que se busca en las reglas</param>
        {
            var best = memoriaArmonica[0].Fitness;
            var bestLocInHm = 0;
            for (var i = 1; i < Constants.Hms; (i)++)
            {
                if (best <= memoriaArmonica[i].Fitness) continue;
                best = memoriaArmonica[i].Fitness;
                bestLocInHm = i;
            }
            return bestLocInHm;
        }
        /// <summary>
        /// Calcula el par correspondiente de acuerdo al numero de iteraciones
        /// </summary>
        /// <param name="ciclo">El numero de iteracion actual del algoritmo</param>
        /// <returns>El valor correspondiente al PAR según el numero de iteracion</returns>
        private static double ParGn(int ciclo)
        {
            var parGn = Constants.ParMin + (((Constants.ParMax - Constants.ParMin) / Constants.Ni) * ciclo);
            return parGn;
        }
    }
```

Figura 17. Código Fuente de la clase Harmony Solution (Parte 2).

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace prism
{
    /// <summary>
    /// Clase que implementa los metodos para la inferencia de reglas
    /// siguiendo el algoritmo PRISM modificado
    /// </summary>
    public class LEM
    {
        #region Members
        /// <summary>
        /// El ancho de la regla.
        /// </summary>
        private readonly double _bandwith = double.NaN;
        /// <summary>
        /// El limite superior del espacio dentro del cual se generan las reglas.
        /// </summary>
        private readonly double _maximo = double.NaN;
        /// <summary>
        /// El limite inferior del espacio dentro del cual se generan las reglas.
        /// </summary>
        private readonly double _minimo = double.NaN;
        #endregion

        #region Constructors
        /// <summary>
        /// Crea una nueva instancia de la clase <see cref="LEM"/>.
        /// </summary>
        /// <param name="li">El limite inferior del espacio donde se encuentran las reglas.</param>
        /// <param name="ls">El limite superior del espacio donde se encuentran las reglas.</param>
        /// <summary>
        /// Genera un vector de reglas a partir de una memoria armónica.
        /// </summary>
        /// <param name="hmReducido">La memoria armónica reducida.</param>
        /// <param name="vectorPosiciones">Un vector con las posiciones del algoritmo.</param>
        /// <param name="reglaConjuntiva">La lista de reglas conjuntivas.</param>
        public void GenerateFullRule(List<HV> hmReducido, List<int> vectorPosiciones, List<Rule> reglaConjuntiva)
        {
            // Mientras que no se hallan recorrido todas las posiciones
            while (vectorPosiciones.Count != 0)
            {
                // 1. Genera una regla con datos disponibles
                Rule reglaActual = GenerateSingleRule(hmReducido, vectorPosiciones);

                // Se añade la regla generada al conjunt de reglas.
                reglaConjuntiva.Add(reglaActual);

                // 2. Filtrar filas de datos que no cumplen con la regla
                // Encuentra la posicion real del atributo en la memoria original
                // esto se realiza por que con cada iteración se reduce el tamaño de
                // columnas del vector con el que se trabaja y necesitamos la posicion
                // original en la cual se encontraba en la memoria armónica.
                // original en la cual se encontraba en la memoria armónica.
                int posicionReal = vectorPosiciones.FindIndex(
                    delegate(int posicion)
                    {
                        if (posicion == reglaActual.originalPositionOfAttribute)
                            return true;
                        return false;
                    }
                );

                // Crea una nueva lista con las filas que se van a eliminar
                var filasAEliminar = new List<int>();
                // Se recorre el vector de memoria armónica reducido con el fin
                // de determinar que filas se van a eliminar del hm, ya que estan contenidas en alguna
                // de las reglas que ya se han generado
                for (int posicionVectorHMReducido = 0; posicionVectorHMReducido < hmReducido.Count; posicionVectorHMReducido++)
                {
                    // Si el atributo en la posicion esta fuera de los limites de la regla actual
                    if (hmReducido[posicionVectorHMReducido].atributos[posicionReal] >= reglaActual.ruleMinValue)
                        if (hmReducido[posicionVectorHMReducido].atributos[posicionReal] <= reglaActual.ruleMaxValue)
                            continue;
                    // En el caso que ya se encuentre dentro de los limites lo añadimos a las filas a borrar
                    filasAEliminar.Add(posicionVectorHMReducido);
                }

                // Ordenamos de forma descendente el vector de filas a eliminar
                IEnumerable<int> sortDescending =
                    from w in filasAEliminar
                    orderby w descending
                    select w;

                // Eliminamos los elementos encontrados en el vector ordenado anteriormente
                foreach (int c in sortDescending)
                vectorPosiciones.RemoveAt(posicionReal);
            }
        }
```

Figura 18. Código Fuente de la clase LEM (Parte 1).

```
111
112         /// <summary>
113         /// Genera una regla a partir de una lista de atributos y una posicion especifica
114         /// </summary>
115         /// <param name="atributosActuales">La lista a partir de la cual se creara la regla</param>
116         /// <param name="vectorPosiciones">El vector de posiciones de la regla</param>
117         /// <returns>La regla con la más alta probabilidad</returns>
118         public Rule GenerateSingleRule(List<HV> atributosActuales, List<int> vectorPosiciones)
119         {
120             var miRegla = new Rule();
121             // La lista de probabilidades de ocurrencia de una regla
122             var listaProbabilidades = new List<Probabilidadrule>();
123             // La posicion relativa de la regla
124             var posicionRelativa = 0;
125
126             // Para cada uno de los valores en el vector de posiciones
127             foreach (int actual in vectorPosiciones)
128             {
129                 // Crea la lista ordenada de valores con su cantidad de altos y bajos
129                 // Crea la lista ordenada de valores con su cantidad de altos y bajos
130                 // y los ordena de menor a mayor valor
131                 var ordenados = new List<Orden>();
132                 foreach (HV current in atributosActuales)
133                 {
134                     //Encuentra la posicion real del atributo partiendo de su posicion relativa
135                     int posicionReal = ordenados.FindIndex(
136                         px => px.Valor == current.atributos[posicionRelativa]
137                         );
138                     // Cataloga el vector ordenado y adiciona una ocurrencia a altos
139                     // o bajos dependiendo del fitness
140                     if (posicionReal >= 0)
141                     {
142                         if (current.fitness == 1)
143                             ordenados[posicionReal].Altos++;
144                         else
145                             ordenados[posicionReal].Bajos++;
146                     }
147                     else
148                     {
149                         // Crea un nuevo orden en la posicion relativa y le adiciona su ocurrencia
150                         // de altos o de bajos
151                         var nuevo = new Orden {Valor = current.atributos[posicionRelativa]};
152                         if (current.fitness == 1)
153                             nuevo.Altos++;
154                         else
155                             nuevo.Bajos++;
156                         // lo adiciona al vector de ordenados
157                         ordenados.Add(nuevo);
158                     }
159                 }
160                 //Se organiza el vector de ordenados de mayor a menor
161                 ordenados.Sort();
172                     // Si es la primera vez se escoje el menor entre el valor que se tiene
173                     // como limite inferior o se escoje el menor - ancho de banda como el menor
174                     if (posicionOrdenados == 0)
175                     {
176                         menor = ordenados[posicionOrdenados].Valor - _bandwith;
177                         if (menor < _minimo) menor = _minimo;
178                     }
179                     else
180                     {
181                         // Se verifica que el menor este definido y se define el menor
182                         // como el valor que se encuentra en la posicion actual del para
183                         // despues se verifican limites para el menor
184                         if (double.IsNaN(menor))
185                         {
186                             menor = ordenados[posicionOrdenados].Valor - _bandwith;
187                             if (menor < _minimo) menor = _minimo;
188                             double mitad = (ordenados[posicionOrdenados].Valor + ordenados[posicionOrdenados - 1].Valor)/2.0;
189                             if (mitad > menor) menor = mitad;
190                         }
191                     }
192                     // Almacena el valor del mayor
193                     double mayor;
203                     {
204                         //Si la posicion no es la ultima del vector de ordenados asigna al mayor al
205                         //actual de la lista de ordenados y comprueba limites
206                         mayor = ordenados[posicionOrdenados].Valor + _bandwith;
207                         if (mayor > _maximo) mayor = _maximo;
208                         double mitad = (ordenados[posicionOrdenados].Valor + ordenados[posicionOrdenados + 1].Valor)/2.0;
209                         if (mitad < mayor) mayor = mitad;
210                     }
211                     // Se aumenta el rango de altos con respecto a la posicion.
212                     altosrango += ordenados[posicionOrdenados].Altos;
213                     // Se ejecuta el proceso de creacion de probabilidades y manejo de adyacencias
214                     // 1. EL nodo actual tienen tanto altos como bajos
215                     // 2. Se llego al final de la lista de nodos con datos
216                     // 3. si hay cambios de un alto a bajo o de un bajo a alto
217                     if ((ordenados[posicionOrdenados].Altos != 0 && ordenados[posicionOrdenados].Bajos != 0) ||
218                         (posicionOrdenados + 1 == ordenados.Count) ||
219                         ((ordenados[posicionOrdenados].Altos != 0 && ordenados[posicionOrdenados + 1].Altos == 0) ||
220                          (ordenados[posicionOrdenados].Altos == 0 && ordenados[posicionOrdenados + 1].Altos != 0)))
221                     {
```

Figura 19. Código Fuente de la clase LEM (Parte 2).

```
                    //Se crea una nueva nodo que almacena los datos basicos de la regla
                    //incluyendo la probabilidad de ocurrencia de la misma
                    var nuevaProbabilidad = new Probabilidadrule
                                            {
                    altosrango = 0;
                    continue;
                }
            }
            // Se ordena el vector ordenados con probabilidad
            ordenadosConProbabilidad.Sort();
            // Se añade a la lista de probabilidades la regla con la probabilidad mas alta
            listaProbabilidades.Add(ordenadosConProbabilidad[0]);
            // Se continua con la siguiente posicion
            posicionRelativa++;
        }
        // Al final se organizan las probabilidades encontradas
        listaProbabilidades.Sort();
        // Se asignan los valores de la regla con la mas alta probabilidad
        miRegla.ruleMinValue = listaProbabilidades[0].Min;
        miRegla.ruleMaxValue = listaProbabilidades[0].Max;
        miRegla.originalPositionOfAttribute = listaProbabilidades[0].Posatr;
        //Se retorna la regla
        return miRegla;
    }

    #endregion
    #endregion

    #region Nested type: Orden
    /// <summary>
    /// Clase utilitaria diseñada para guardar la probabilidad de ocurrencia de altos y bajos de
    /// acuerdo a un valor especifico
    /// </summary>
    private class Orden : IComparable<Orden>
    {
        /// <summary>
        /// El numero de ocurrencia dentro del tipo de alta confiabilidad
        /// </summary>
        public int Altos;
        /// <summary>
        /// El numero de ocurrencia dentro del tipo de baja confiabilidad
        /// </summary>
        public int Bajos;
        /// <summary>
        /// El valor del atributo
        /// </summary>
        public double Valor;

        #region IComparable<Orden> Members
        /// <summary>
        /// Compara los atributos para especificar si son iguales o no
        /// </summary>
        /// <param name="other">El otro elemento contra el cual se compara</param>
        /// <returns>Cero si son iguales, otro valor si no lo son</returns>
        public int CompareTo(Orden other)
        {
            return Valor.CompareTo(other.Valor);
        }

    private class Probabilidadrule : IComparable<Probabilidadrule>
    {
        /// <summary>
        /// El valor maximo de la regla
        /// </summary>
        public double Max;
        /// <summary>
        /// El valor minimo de la regla
        /// </summary>
        public double Min;
        /// <summary>
        /// La posicion original del atributo
        /// </summary>
        public int Posatr;
        /// <summary>
        /// La probabilidad de ocurrencia del atributo
        /// </summary>
        public double Probabilidad;
        /// <summary>
        /// El ancho de la regla
        /// </summary>
        public double Wide;

        /// <summary>
        /// Compara los atributos para especificar si son iguales o no, compara primero por probabilidad
        /// si son iguales en cuanto a probabilidad compara por ancho de regla.
        /// </summary>
        /// <param name="other">El otro elemento contra el cual se compara</param>
        /// <returns>Cero si son iguales, otro valor si no lo son</returns>
        public int CompareTo(Probabilidadrule other)
        {
            int x = -1*Probabilidad.CompareTo(other.Probabilidad);
            if (x == 0)
            {
                x = Wide.CompareTo(other.Wide);
            }
            return x;
        }

        #endregion
    }

    #endregion
}
```

Figura 20. Código Fuente de la clase LEM (Parte 3).

```
using System;

namespace prism
{

    /// <summary>
    /// Clase Rule, Maneja los valores que toma una regla
    /// </summary>
    public class Rule : IComparable<Rule>
    {
        #region Members
        /// <summary>
        /// El valor máximo que puede tomar la regla
        /// </summary>
        public double ruleMaxValue;
        /// <summary>
        /// El valor mínimo que puede tomar la regla
        /// </summary>
        public double ruleMinValue;
        /// <summary>
        /// El valor de la posición original del atributo
        /// </summary>
        public int originalPositionOfAttribute;
        #endregion

        #region IComparable<rule> Members
        /// <summary>
        /// Compara la regla actual contra otra del mismo tipo.
        /// </summary>
        /// <param name="other">La regla contra la cual se compara.</param>
        /// <returns>Cero si las reglas son iguales, en caso distinto retorna otro tipo de entero</returns>
        public int CompareTo(Rule other)
        {
        /// <param name="other">La regla contra la cual se compara.</param>
        /// <returns>Cero si las reglas son iguales, en caso distinto retorna otro tipo de entero</returns>
        public int CompareTo(Rule other)
        {
            // return originalPositionOfAttribute.CompareTo(other.originalPositionOfAttribute);
            return originalPositionOfAttribute.CompareTo(other.originalPositionOfAttribute);
        }
        #endregion

        #region Methods
        /// <summary>
        /// Clase que exporta las regla a una cadena de caracteres,
        /// para propósitos de monitoreo de la regla
        /// </summary>
        /// <returns>Una cadena de caracteres con la regla</returns>
        public override string ToString()
        {
            return originalPositionOfAttribute + "(" + ruleMinValue.ToString("0.000") + " - " + ruleMaxValue.ToString("0.000") + ")";
        }
        #endregion
    }
}
```

Figura 21. Código Fuente de la clase Rule.

14 COMPLEJIDAD DEL ALGORITMO GHS+LEM

La complejidad del algoritmo GHS+LEM está dada por:

- El proceso de inicialización de la memoria armónica, el cual asigna valores aleatorios a las dimensiones de cada vector solución y el cálculo de la función de aptitud para cada vector solución de la memoria armónica (HM). Este proceso tiene un orden de complejidad limitado por *O(Nd*HMS)*.

- El proceso de inferencia de reglas tiene un orden de complejidad limitado por $O(Nd^3 * HMS)$, el cual se consigue de analizar los tres pasos principales de esta función:

- La transformación de la memoria armónica (HM) para el manejo de las dimensiones en la creación de reglas. Este proceso se realiza (HMS*Nd) veces.

- La creación del vector de posiciones, el cual almacena el número de la dimensión a evaluar. Este proceso se realiza Nd veces.

- La realización del proceso de creación de la regla conjuntiva y de cada regla que la compone. Para este proceso se realizan Nd veces los siguientes pasos:

    - Generar una regla, lo cual toma Nd(HMS*Nd + HMS) veces, debido a que se ejecuta Nd veces la creación del vector de valores ordenados (lo cual toma HMS*Nd veces) y la creación de la regla que a su vez toma Tordenados veces (donde Tordenados es el tamaño del vector de valores ordenados y es un valor menor que HMS).

    - Evaluar el vector reducido con la regla generada y marcar las filas del vector a borrar, lo cual toma HMS veces.

    - Eliminar las filas marcadas del vector reducido, lo cual toma Nfilas veces (donde Nfilas es el número de filas a borrar). Nfilas es un valor menor que HMS.

    - Eliminar la columna del vector reducido usada para la creación de la regla, lo cual toma HMS veces.

- El proceso de improvisación de las NI nuevas armonías. Este proceso se ejecuta NI veces, y para cada vector solución se necesitan asignar Nd dimensiones, por lo anterior tiene un orden de complejidad limitado por *O(Nd*NI)*.

- El proceso de inferencia de reglas que se ejecuta RRU veces el número de improvisaciones. Es decir tiene un orden de complejidad limitado principalmente por *O(RRU * Nd3 * HMS)*.

Como resultado final se obtiene que el orden de complejidad del algoritmo propuesto es de ***O(Nd * (HMS + NI + RRU * Nd^2* HMS))*** que resulta de sumar el *O(Nd*HMS*) de la generación inicial de la memoria armónica, *O(Nd*NI)* de la generación de los improvisos y *O(RRU * Nd^3 * HMS)* del proceso de generación de reglas.

A pesar de que el algoritmo tiene una complejidad cúbica en relación con el número de dimensiones del problema, el factor más costoso lo aporta el número de improvisaciones, en primera instancia porque es el valor que usualmente es más alto, por ejemplo: 5.000 o 50.000 iteraciones. Además porque el proceso de generación de reglas se ejecuta un porcentaje del número de improvisaciones (valor definido por RRU). Si el valor RRU es bajo, se controla el factor cubico del algoritmo.

Parte 5 – Experimentación

En el desarrollo de este proyecto se propuso como tercer objetivo el desarrollo de un prototipo software que permitiera realizar las comparaciones entre el algoritmo propuesto y los algoritmos armónicos más destacado (HS, IHS, GBHS) mediante una serie de pruebas experimentales sobre 10 funciones clásicas usadas en optimización. El proceso del desarrollo del Sistema y los resultados del experimento se detallan a continuación.

15 SOFTWARE DE LABORATORIO

Para el desarrollo de las pruebas del proyecto se realizó un software que permitió efectuar las comparaciones del algoritmo GHS+LEM con los principales algoritmos armónicos propuestos en la literatura. Esta labor requirió el uso de una metodología de desarrollo de software ágil basada en UP que se describe a continuación.

15.1 DESCRIPCIÓN GENERAL DE LA METODOLOGÍA

La metodología que se utilizó en este proyecto es una instanciación del Proceso Unificado, la cual tuvo en cuenta las siguientes fases:

15.1.1 INICIACIÓN

En esta fase se realizó el diseño preliminar del algoritmo y de la arquitectura general del sistema teniendo en cuenta los requisitos relacionados con la evaluación. Como resultado se obtuvo el Diagrama de Casos de Uso del Sistema.

15.1.2 ELABORACIÓN

En esta fase se definieron los requerimientos necesarios para el modelado y diseño del algoritmo y el afinamiento de la arquitectura general del sistema. En

esta etapa se obtuvieron los siguientes artefactos de manera preliminar: Casos de uso de alto nivel, Diagrama de Clases y la Arquitectura Base.

15.1.3 CONSTRUCCIÓN

Una vez realizadas las fases de Iniciación y Elaboración, se obtuvo un prototipo funcional del algoritmo a través de las siguientes actividades:

- **Análisis**: Se profundizó sobre los artefactos generados durante la fase de elaboración para la creación del sistema.
- **Diseño**: Se realizaron los casos de uso reales que sirvieron como guía para la construcción de las diferentes funcionalidades.
- **Implementación**: Se implementó el sistema (en los primeros ciclos el algoritmo) con los artefactos obtenidos en las anteriores actividades (Análisis y Diseño), posteriormente se realizaron pruebas alfa para garantizar su funcionalidad.
- **Pruebas**: Se realizaron las validaciones pertinentes sobre el software para verificar el resultado de esta actividad.

15.1.3.1 CICLOS DE DESARROLLO

Los ciclos de desarrollo permitieron dividir la funcionalidad del sistema en funciones más pequeñas que facilitaron la labor de construcción del sistema cumpliendo con cada una de las fases mencionadas anteriormente. Los ciclos desarrollados fueron:

- Ciclo 1. Algoritmo de hibridación inicial: En este ciclo se modeló e implementó el algoritmo inicial utilizando Microsoft Visual Studio 2010 y C# como lenguaje de programación.

- Ciclo 2. Algoritmo de hibridación final: En este ciclo se modeló e implementó el algoritmo final teniendo en cuenta el proceso de inferencia de reglas derivado de PRISM.

- Ciclo 3. Mecanismo de comparación entre algoritmos: En este ciclo se definieron e implementaron las 10 funciones que posteriormente se usaron para realizar la comparación de los algoritmos.

- Ciclo 4. Implementación de algoritmos de comparación: En este ciclo se implementaron los algoritmos HS, IHS y GHS.

- Ciclo 5. Tabulación y reporte de resultados de las comparativas: en este ciclo se desarrollo el mecanismo para obtener los resultados de la comparativa de los algoritmos a través de un archivo Excel.

15.1.4 TRANSICIÓN

Después de finalizar la fase de construcción del sistema, se verificó la funcionalidad del mismo y se realizó la evaluación del algoritmo con los resultados obtenidos mediante el uso del sistema. En nuestro caso, se usaron funciones (las definidas en el tercer objetivo especifico) ya definidas por la comunidad de optimización, a saber: Sphere, Schwefel, Step, Rosenbrock, Rotated hyper-ellipsoid, Generalized Schwefel, Rastrigin, Ackley, Griewank, Six-Hump Camel-back.

15.1.5 DOCUMENTACIÓN Y DIVULGACIÓN

A lo largo de cada fase se desarrolló la documentación respectiva. Finalmente la divulgación se termina con la realización de la presente monografía y la sustentación de la tesis ante los jurados definidos por la Facultad de Ingeniería Electrónica y Telecomunicaciones, FIET. La documentación presentó los resultados obtenidos de las comparativas del algoritmo y las recomendaciones sugeridas.

15.2 ARQUITECTURA DEL SISTEMA

Para el sistema se definió una arquitectura multinivel que consta de 3 niveles, lógica de presentación, lógica de negocio y lógica de servicios. Entre las ventajas que se obtienen mediante el uso de este tipo de arquitectura están la flexibilidad,

la escalabilidad y el mantenimiento eficiente del sistema. En la figura 22 se muestra la arquitectura del sistema y sus componentes.

A continuación se hace una breve descripción de las funciones que se realizan en cada uno de los niveles de la arquitectura.

- Nivel de Presentación: En este nivel se incluyen los componentes de la interfaz del usuario que permiten seleccionar las opciones necesarias para la ejecución de las pruebas. Este nivel se comunica únicamente con la capa de negocio por medio de las interfaces del servicio.

- Nivel de Lógica de Negocio: En este nivel se implementan por medio de servicios, las reglas del negocio y los procesos relacionados con las funcionalidades que ofrece el sistema. Está dividido en interfaces de servicio y en tres módulos:

    - Interfaces del Servicio: dan soporte a la implementación de servicios para que el cliente pueda acceder a los métodos suministrados por los módulos en el nivel de lógica del negocio.

    - Módulo de Funciones: Contiene las 10 funciones de optimización que pueden ser evaluadas en la aplicación.

    - Módulo Harmony Search: Contiene los algoritmos armónicos que realizaran la evaluación de las funciones de optimización.

    - Módulo Prism: Contiene el proceso de inferencia de reglas necesario para la ejecución del algoritmo armónico propuesto en el proyecto.

- Nivel de Persistencia: En este nivel es donde residen los datos. Tiene como objetivo almacenar los resultados de la ejecución de las pruebas en un

conjunto de directorios y archivos para su posterior revisión por parte del usuario final. Este nivel contiene el siguiente subnivel:

- Lógica de servicios: Implementa la persistencia de la información obtenida por el programa ocultando los detalles de los repositorios de datos a los niveles superiores. Este subnivel comunica la capa de negocio con el conjunto de datos (directorio y archivos) que realiza el almacenamiento de la información.

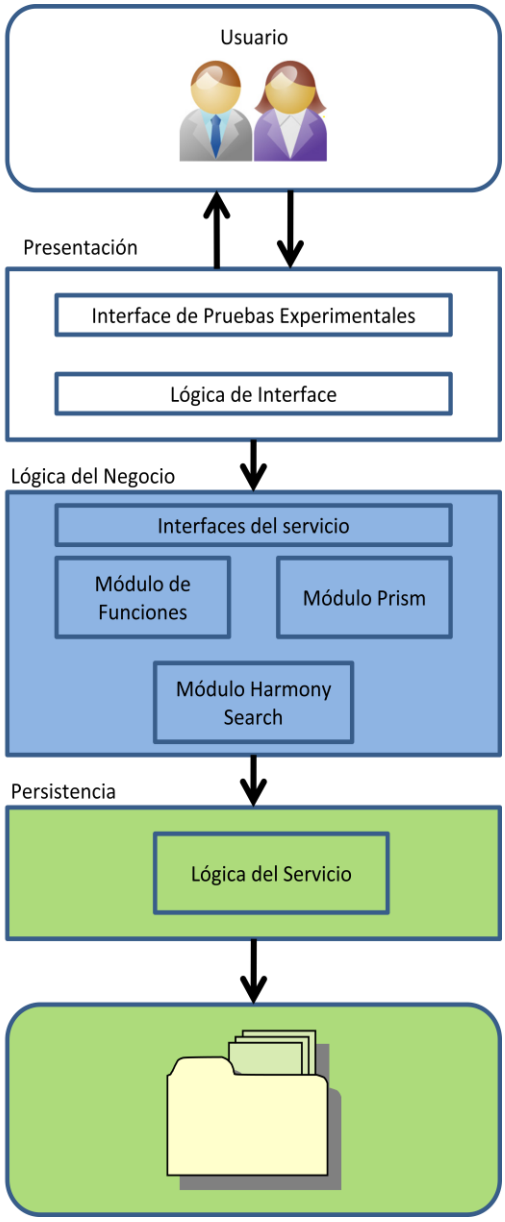


Figura 22. Arquitectura del Sistema.

15.3 ANÁLISIS Y DISEÑO

A continuación se muestran algunos resultados generales del sistema, resultado del análisis, diseño, implementación y pruebas en los diferentes ciclos, llevados a cabo para el desarrollo del software de laboratorio.

15.3.1 CASO DE USO DE ALTO NIVEL

En la figura 23 se muestra la única operación que el usuario puede realizar.

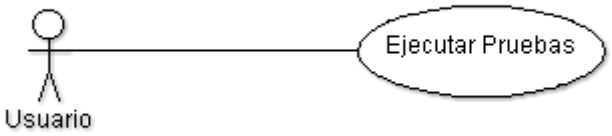


Figura 23. Diagrama de casos de uso para los usuarios del Sistema.

15.3.2 CASO DE USO REAL

A continuación se muestra el caso de uso real del sistema.

CASO DE USO REAL: EJECUTAR PRUEBAS	
Actores: Usuario.	
Propósito: Realizar la ejecución de las pruebas del experimento.	
Resumen: El usuario selecciona la opción ejecutar pruebas y el sistema despliega la interfaz correspondiente a la opción seleccionada. El usuario digita la ubicación donde se almacenaran los archivos generados. El sistema ejecuta las opciones seleccionadas y regresa al anterior caso de uso.	
Tipo: Primario.	
CURSO NORMAL DE LOS EVENTOS	
Acción del actor	**Respuesta del sistema**
1. El usuario ejecuta la opción que da inicio a este caso de uso.	
	2. El sistema presenta al usuario una interfaz con las opciones Directorio de destino, Pruebas Programación entera, Comparativa GHS+LEM,

	Pruebas Variación HMS, Pruebas Variación HCMR, Pruebas Variación RCR y Pruebas Variación PAR.
3. El usuario puede modificar el directorio de destino, en caso de que lo requiera.	
4. El usuario escoge la opción deseada.	5. El sistema procesa la opción: a. Si es “Pruebas Programación entera”, el sistema realiza las pruebas de programación entera de la experimentación y las almacena en un conjunto de archivos de Excel y directorios en el directorio de destino. b. Si es “Comparativa GHS+LEM”, el sistema realiza las pruebas comparativas entre los algoritmos armónicos HS, IHS, GHS y GHS+LEM las almacena en un conjunto de archivos de Excel y directorios en el directorio de destino. c. Si es “Pruebas Variación HMS”, el sistema realiza las pruebas correspondientes a la variación de HMS y las almacena en un conjunto de archivos de Excel y directorios en el directorio de destino. d. Si es “Pruebas Variación HCMR”, el sistema realiza las pruebas correspondientes a la variación de HCMR y las almacena en un conjunto de archivos de Excel y

	directorios en el directorio de destino. e. Si es “Pruebas Variación RCR”, el sistema realiza las pruebas correspondientes a la variación de RCR y las almacena en un conjunto de archivos de Excel y directorios en el directorio de destino. f. Si es “Pruebas Variación PAR”, el sistema realiza las pruebas correspondientes a la variación de PAR y las almacena en un conjunto de archivos de Excel y directorios en el directorio de destino.
	6. Una vez terminada la ejecución de la opción deseada, el sistema finaliza.

Tabla 21. Caso de Uso Real Ejecutar Pruebas.

15.3.3 DIAGRAMA DE CLASES

En las Figuras 24, 25, 26 y 27 se muestra el diagrama de clases del sistema representado por paquetes. A continuación en la Tabla 22 se describe la funcionalidad de cada clase.

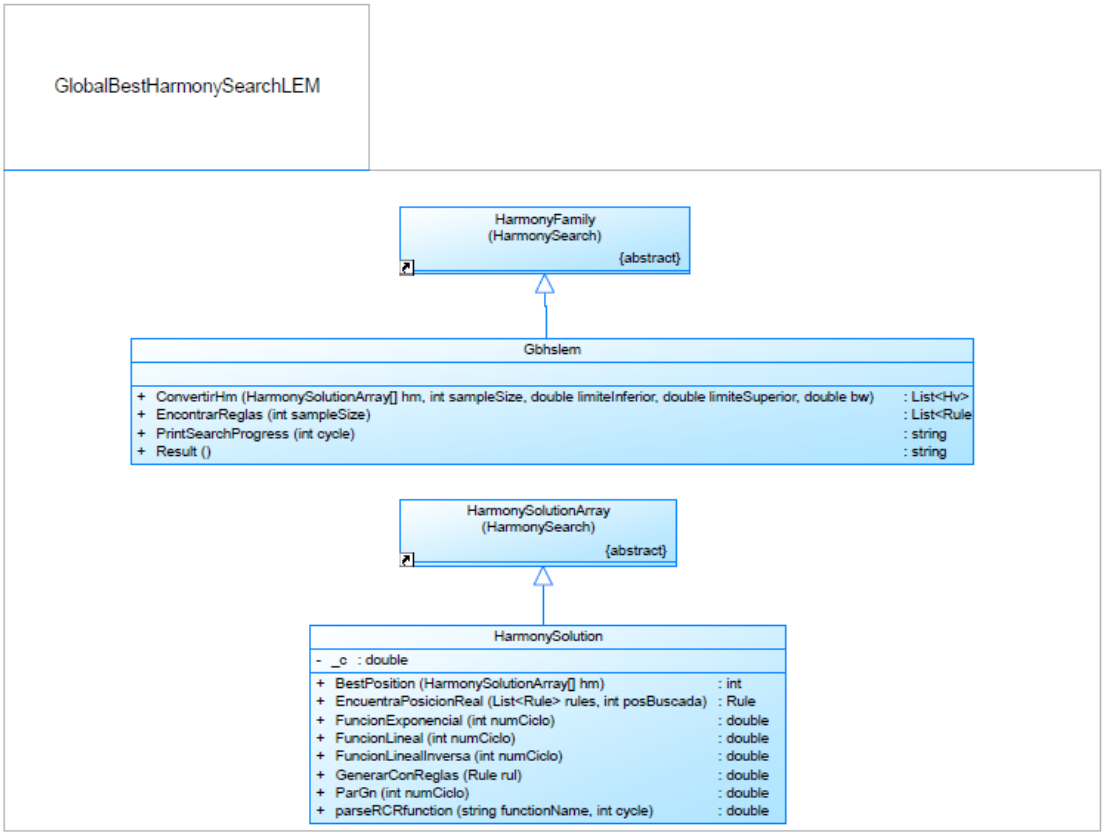


Figura 24. Diagrama de Clases del Sistema (Parte 1).

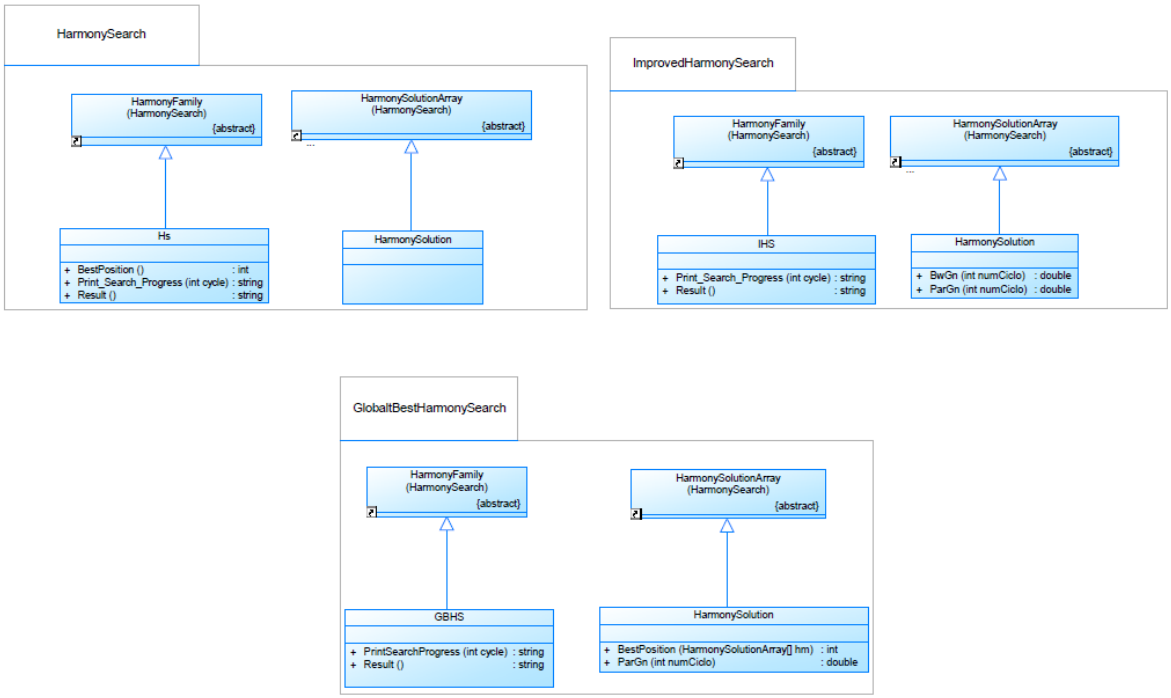


Figura 25. Diagrama de Clases del Sistema (Parte 2).

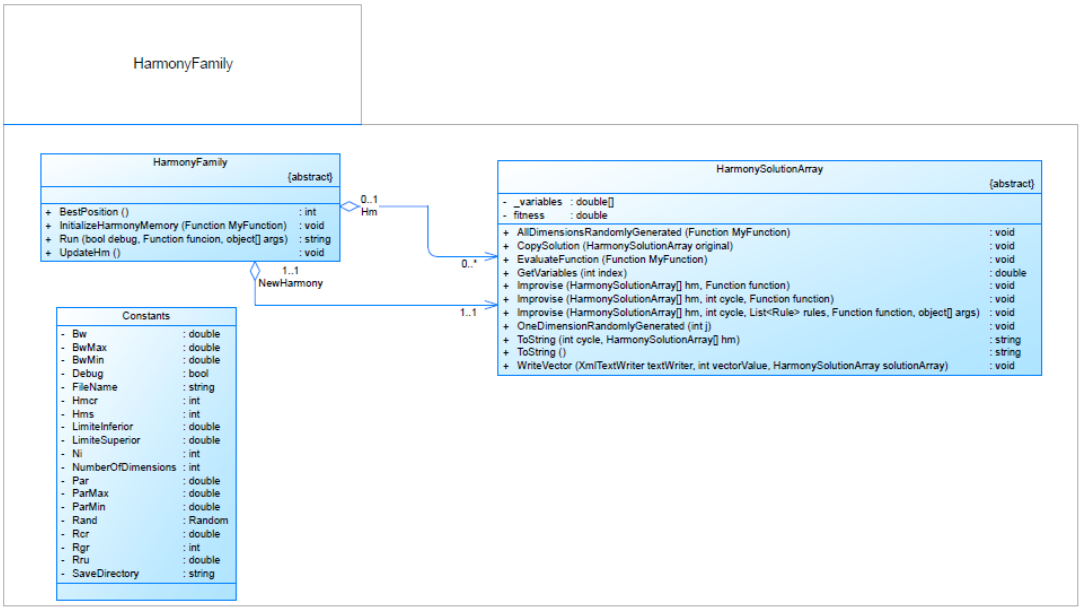


Figura 26. Diagrama de Clases del Sistema (Parte 3).

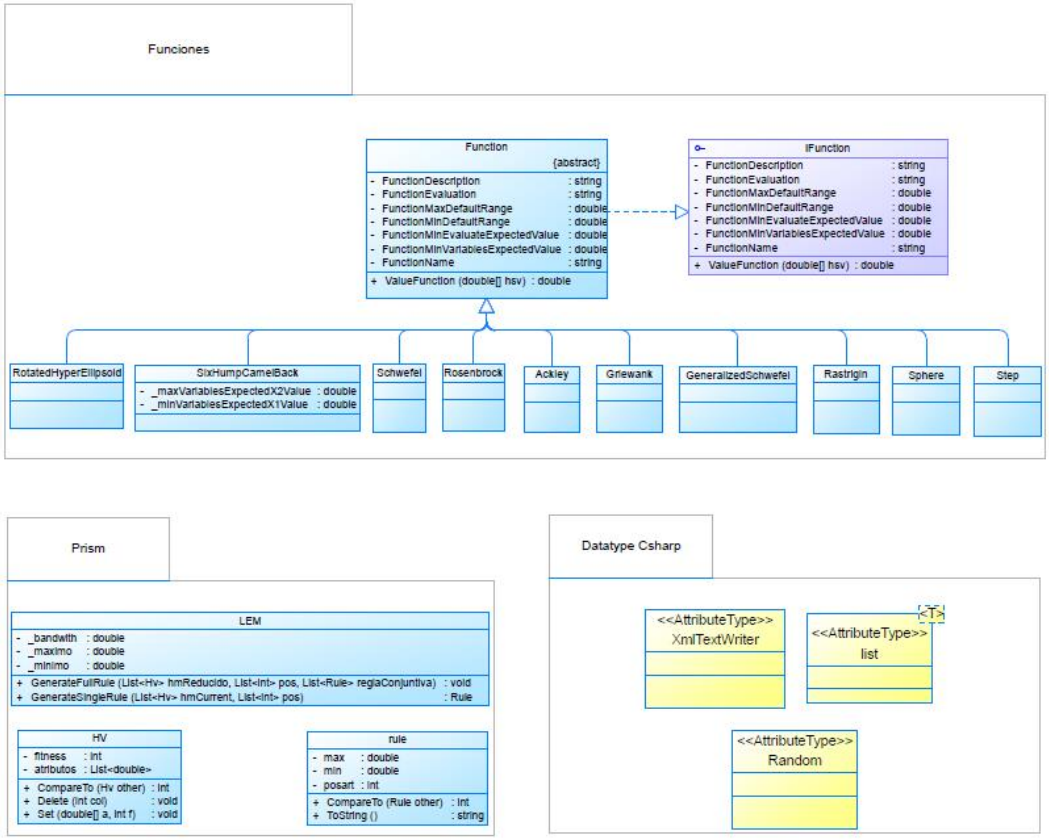


Figura 27. Diagrama de Clases del Sistema (Parte 4).

CLASE	FUNCION
Ackley	Clase derivada de Función que contiene las características de la función Ackley.
Constants	Contiene las variables necesarias para la ejecución de HS.
Function	Representa las características básicas de una función.
GBHS	Representa la funcionalidad (pasos) del algoritmo GBHS
GBHSLEM	Representa la funcionalidad (pasos) del algoritmo GBHSLEM
GeneralizedSchwefel	Clase derivada de Función que contiene las características de la función GeneralizedSchwefel.
Griewank	Clase derivada de Función que contiene las características de la función Griewank.
HarmonyFamily	Representa la funcionalidad básica (pasos) del algoritmo HS.
HarmonySolution (paquete GlobalBestHarmonySearch)	Representa la funcionalidad de HM (memoria armónica) para el algoritmo GBHS.
HarmonySolution (paquete GlobalBestHarmonySearchLEM)	Representa la funcionalidad de HM (memoria armónica) para el algoritmo GBHSLEM.
HarmonySolution (paquete HarmonySearch)	Representa la funcionalidad de HM (memoria armónica) para el algoritmo HS.
HarmonySolution (paquete ImprovedHarmonySearch)	Representa la funcionalidad de HM (memoria armónica) para el algoritmo IHS.
HarmonySolutionArray	Representa la funcionalidad básica de la HM (memoria armónica).
HS	Representa la funcionalidad (pasos) del algoritmo

	HS
HV	Especifica los atributos y el fitness de una dimensión contenida en HM (memoria armónica).
IFunction	Interfaz de la clase Función.
IHS	Representa la funcionalidad (pasos) del algoritmo IHS
LEM	Contiene la funcionalidad necesaria para la ejecución de LEM (Modelo Evolutivo que Aprende).
list	Representa el tipo de dato list que hace parte del lenguaje Csharp.
Random	Representa el tipo de dato Random que hace parte del lenguaje Csharp.
Rastring	Clase derivada de Función que contiene las características de la función Rastring.
Rosenbrock	Clase derivada de Función que contiene las características de la función Rosenbrock.
RotatedHyperEllipsoid	Clase derivada de Función que contiene las características de la función RotatedHyperEllipsoid.
Rule	Especifica las características de una regla.
Schwefel	Clase derivada de Función que contiene las características de la función Schwefel.
SixHumpCamelBack	Clase derivada de Función que contiene las características de la función SixHumpCamelBack.
Sphere	Clase derivada de Función que contiene las características de la función Sphere.
Step	Clase derivada de Función que contiene las características de la función Step.

XMLTextWriter	Representa el tipo de dato XMLTextWriter que hace parte del lenguaje Csharp.

Tabla 22. Descripción de las Clases del Sistema.

16 RESULTADOS DEL EXPERIMENTO

Con el desarrollo del software de laboratorio, se realizaron las pruebas del algoritmo GHS+LEM. Este proceso tuvo como objetivo medir el desempeño del algoritmo frente a los otros algoritmos de búsqueda armónica (HS, IHS, GBHS). En la Tabla 23 se muestra la configuración de parámetros generalmente usada.

Variable	HS	IHS	GHS	GHSLEM
HMS	5	5	5	5
HCMR	0.9	0.9	0.9	0.9
PAR	0.3	N.A.	N.A.	N.A.
PAR_{min}	N.A.	0.01	0.01	0.01
PAR_{max}	N.A.	0.99	0.99	0.99
bw	0.01	N.A.	N.A.	N.A.
bw_{min}	N.A.	0.0001	N.A.	N.A.
bw_{max}	N.A.	$1/(20\ x\ (UB\ -\ LB)\ )$	N.A.	N.A.
HLGS	N.A.	N.A.	N.A.	$\lfloor HMS/2 \rfloor$
RCR	N.A.	N.A.	N.A.	0.9
RRU	N.A.	N.A.	N.A.	0.2

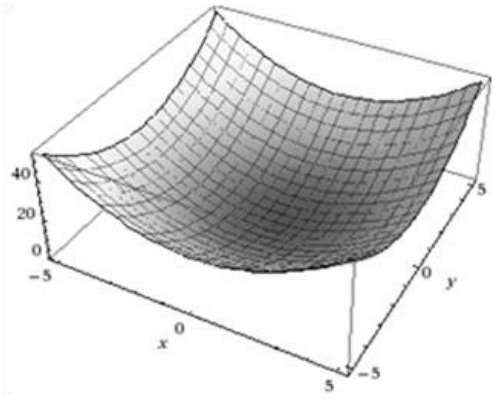
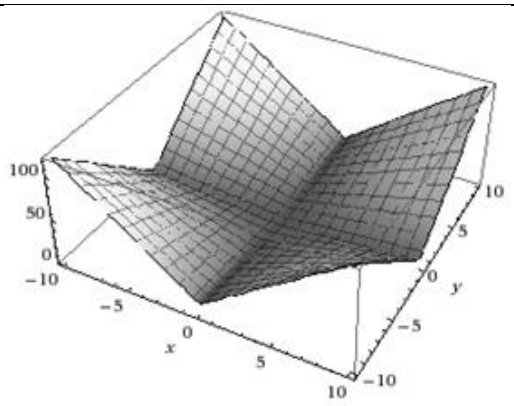
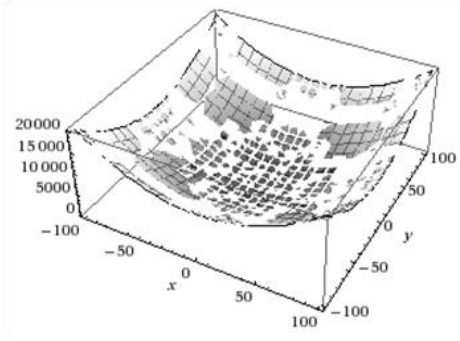
Tabla 23. Configuraciones generales de las pruebas.

Todas las funciones, excepto la Six-Hump Camel-Back que es bidimensional, fueron implementadas para 2, 3, 5, 10, 15, 20, 30, 40 y 50 dimensiones. Para cada una de las dimensiones se utilizaron 50, 500, 5.000 y 50.000 iteraciones. En cada caso se especifica cuántas dimensiones se utilizaron en la prueba específica, además del número de iteraciones específico. La memoria armónica inicial se genera de forma aleatoria dentro de los rangos específicos de cada función.

16.1 FUNCIONES DE PRUEBA USADAS EN LA EVALUACIÓN

Para la comparativa se utilizaron las funciones que aparecen en la Tabla 24 de funciones unimodales y en la Tabla 25 de funciones multimodales. Estas se basan en las funciones propuestas en el artículo de GHS [5] que proporciona un balance adecuado entre funciones unimodales y multimodales. Para cada una de las funciones se busca encontrar el mínimo global definido como:

Dado $f = \Re^{N_d} \rightarrow \Re$ encontrar $x^* \in \Re^{N_d}$ tal que $f(x^*) \leq f(x), \forall x \in \Re^{N_d}$, donde $N_d, es\ el\ número\ de\ dimensiones$

A. Sphere (Primera función de De Jong´s)[31] $$f(x) = \sum_{i=1}^{N_d} {x_i}^2$$ Donde $x^* = 0$ y $f(x^*) = 0$ para $-5.12 \leq x_i \leq 5.12$	
B. Schwefel's Problem 2.22[32] $$f(x) = \sum_{i=1}^{N_d} \lvert x_i \rvert + \prod_{i=1}^{N_d} \lvert x_i \rvert$$ Donde $x^* = 0$ y $f(x^*) = 0$ para $-10 \leq x_i \leq 10$	
C. Step $$f(x) = \sum_{i=1}^{N_d} (\lfloor x_i + 0.5 \rfloor)^2$$ Donde $x^* = 0$ y $f(x^*) = 0$ para $-100 \leq x_i \leq 100$	

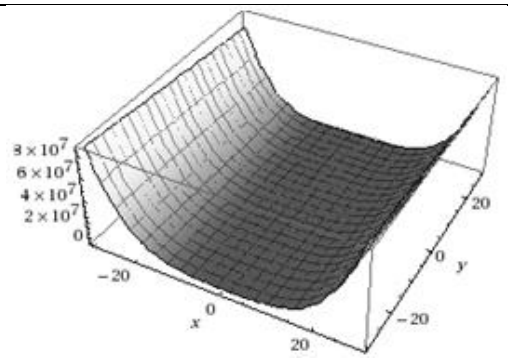
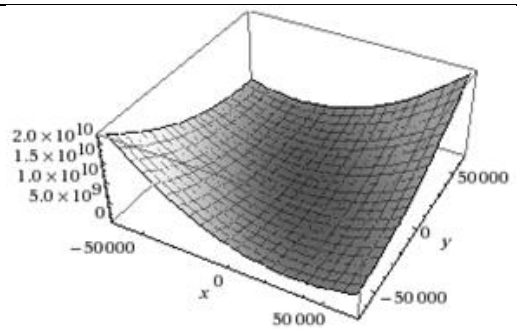
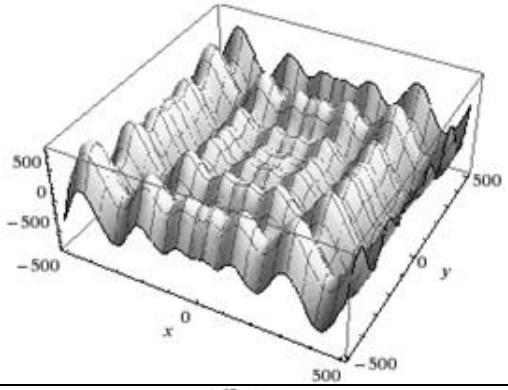
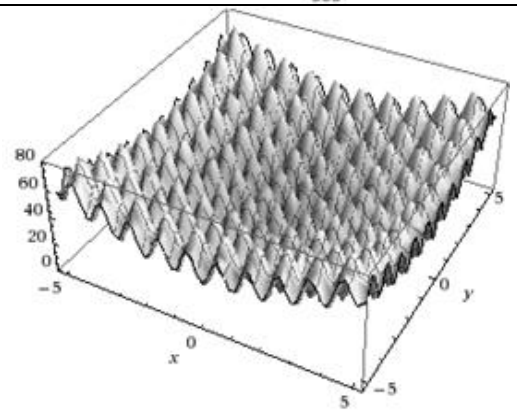
D. Rosenbrock [31] $f(x) = \sum_{i=1}^{N_d-1} \left[100(x_{i+1} - x_i^2)^2 + (1 - x_{i)}{}^2\right]$ Donde $x^* = (1,1,\dots,1)$ y $f(x^*) = 0$ para $-30 \leq x_i \leq 30$	
E. Rotated Hyper-ellipsoid [5] $f(x) = \sum_{i=1}^{N_d} \left(\sum_{j=1}^{i} x_j \right)^2$ Donde $x^* = 0$ y $f(x^*) = 0$ para $-65536 \leq x_i \leq 65536$	

Tabla 24. Funciones Unimodales.

F. Generalized Schwefel 2.26[32] $f(x) = -\sum_{i=1}^{N_d} (x_i sin(\sqrt{x_i}))$ Donde $x^* = (420.9687, \dots, 420.9687)$ y $f(x^*) = -12569.5$ para $-500 \leq x_i \leq 500$	
G. Rastrigin[31] $f(x) = \sum_{i=1}^{N_d} ({x_i}^2 - 10cos(2\pi x_i) + 10N_d)$ Donde $x^* = 0$ y $f(x^*) = 0$ para $-5.12 \leq x_i \leq 5.12$	

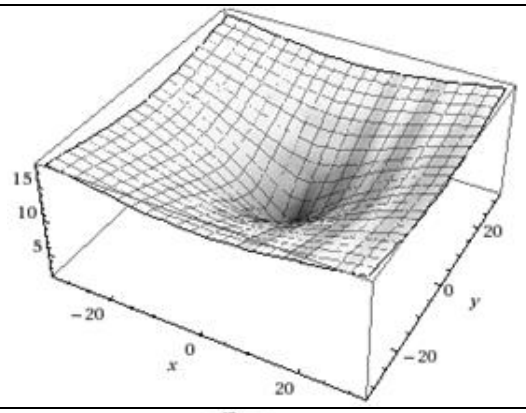
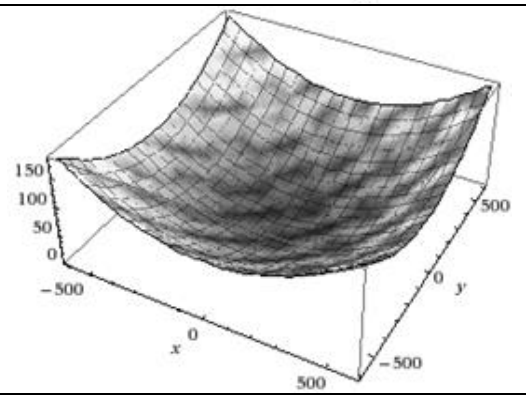
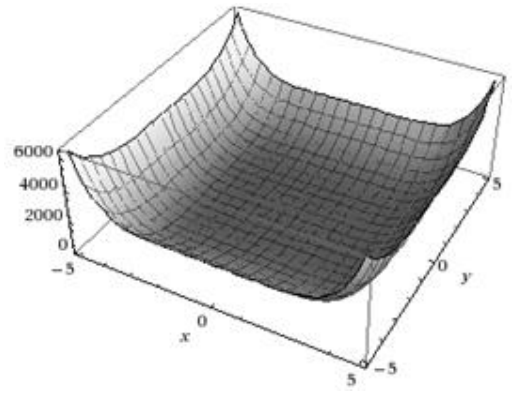
H. Ackley[31] $f(x) = -20e^{\left(-0.2\sqrt{\frac{1}{30}\sum_{i=1}^{N_d} x_i^2}\right)} - e^{\left(\frac{1}{30}\sum_{i=1}^{N_d}\cos 2\pi x_i\right)} + 20 + e$ Donde $x^* = 0$ y $f(x^*) = 0$ para $-32 \leq x_i \leq 32$	
I. Griewank[31] $f(x) = \frac{1}{4000}\sum_{i=1}^{N_d} x_i^2 - \prod_{i=1}^{N_d}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$ Donde $x^* = 0$ y $f(x^*) = 0$ para $-600 \leq x_i \leq 600$	
J. Six-Hump Camel-Back [5] Función de baja dimensionalidad con pocos mínimos locales. $f(x) = 4x_1^2 + 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$ Donde $x^* = (-0.08983, 0.7126), (-0.08983, 0.7126)$ y $f(x^*) = -1.0316285$ para $-5 \leq x_i \leq 5$	

Tabla 25. Funciones Multimodales.

16.2 RESULTADOS DE LA COMPARATIVA

La Tabla 26 presenta los resultados de las pruebas comparativas aplicadas al algoritmo GHS+LEM con el objeto de medir su precisión contra los otros algoritmos de la familia armónica. El número de iteraciones para la prueba se definió en 50.000 y el número de dimensiones se estableció en 30 para todas las funciones excepto para Six-hump Camel-Back la cual está definida en dos

dimensiones. Los algoritmos fueron ejecutados 100 veces para garantizar una desviación media confiable.

		HMS	IHS	GHS	GHS+LEM
Unimodales					
Sphere	Media	0.000684005	0.017838978	4.0457E-05	**2.09647E-10**
	(±SD)	*9.67781E-05*	*0.00710319*	*7.29366E-05*	***3.60168E-10***
Schwefel Problem 2.22	Media	0.143656975	0.997096357	0.040860755	**5.70121E-05**
	(±SD)	*0.047911784*	*0.200329207*	*0.037067055*	***4.49754E-05***
Rosenbrock	Media	312.2431152	423.9427774	72.47196696	**15.77537882**
	(±SD)	*486.5124844*	*330.6943507*	*103.3253058*	***22.43722786***
Step	Media	11.56	11.22	**0**	**0**
	(±SD)	*4.608555943*	*3.945538332*	***0***	***0***
Rotated Hyper-Ellipsoid	Media	4419.904558	4238.371416	5427.433309	**686.306798**
	(±SD)	*1306.038774*	***1196.967797***	*6641.408049*	*2049.883544*
Multimodales					
Schwefel Problem 2.26	Media	-12545.01282	-12540.34846	-12569.46257	**-12569.48662**
	(±SD)	*9.274118296*	*10.54344883*	*0.03971048*	***3.40336E-11***
Rastrigin	Media	1.266797341	2.722732645	0.009457309	**3.81653E-08**
	(±SD)	*1.023021844*	*1.130249802*	*0.014012005*	***6.06791E-08***
Ackley	Media	0.981392208	1.584674315	0.024746761	**5.83147E-06**
	(±SD)	*0.485630315*	*0.331393069*	*0.026603311*	***4.86194E-06***
Griewank	Media	1.085396028	1.087082117	0.091022469	**2.1722E-11**
	(±SD)	*0.035098647*	*0.031926489*	*0.192952247*	***4.5967E-11***
Six-Hump Camel- Back	Media	-1.031600318	-1.031628428	-1.031568182	**-1.031628452**
	(±SD)	*3.48248E-05*	*5.53445E-09*	*8.34751E-05*	***1.45999E-09***

Tabla 26. Valor Medio y desviación estándar ($\pm \boldsymbol{SD}$) de las pruebas (Nd = 30, NI=50.000).

Los resultados de las pruebas aplicadas indican que GHS+LEM supera ampliamente la precisión obtenida por HS, IHS y GHS en todas las funciones de optimización realizadas. La desviación estándar de las pruebas también es menor para todas las funciones en las cuales se aplicó el algoritmo a excepción de Rotated Hyper-ellipsoid en la cual IHS es menor. Sin embargo aún en el peor caso, el resultado obtenido con GHS+LEM (2736.190342) supera el resultado del mejor obtenido por IHS (5435.339213).

La Tabla 27 presenta los resultados de las pruebas de escalabilidad a las que fue sometido el algoritmo GHS+LEM. El número de iteraciones se definió en 50.000, el número de dimensiones en 50 y el número de ejecuciones de cada algoritmo fue fijado en 100. No se incluyen los resultados de la función Six-Hump Camel-back los cuales se presentan en la **Tabla 39**. GHS+LEM mejora la precisión en cada una de las funciones de optimización usadas, probando ser mejor que HS, IHS y GHS en condiciones de alta dimensionalidad. Además, en funciones discontinuas como Step, GHS+LEM mantiene su resultado óptimo a diferencia de las demás propuestas, las cuales sufren en condiciones de mayor dimensionalidad.

		HMS	IHS	GHS	GHS+LEM
Unimodales					
Sphere	Media	1.231713634	1.36689254	0.005550663	**2.58528E-08**
	(±SD)	*0.287760963*	*0.308892735*	*0.00776301*	***5.5786E-08***
Schwefel Problem 2.22	Media	9.594968369	10.03102019	0.411417885	**0.000441435**
	(±SD)	*1.080214642*	*1.361876185*	*0.397066313*	***0.000376589***
Rosenbrock	Media	28119.05942	27416.72832	357.7255365	**39.49848422**
	(±SD)	*10535.26342*	*9607.338888*	*726.1644056*	***59.9930843***
Step	Media	513.92	535.07	0.09	**0**
	(±SD)	*101.4288505*	*112.1655752*	*0.637149587*	***0***
Rotated Hyper-Ellipsoid	Media	28751.33713	28878.96444	59867.79886	**27492.62734**
	(±SD)	*5722.795639*	*6583.833253*	***21857.53114***	*30492.1086*
Multimodales					
Schwefel Problem 2.26	Media	-20065.84929	-20055.73906	-20944.1766	**-20949.14436**
	(±SD)	*183.3728874*	*187.2603852*	*8.953405915*	***3.8441E-09***
Rastrigin	Media	35.35722669	45.97323267	0.407654111	**4.13742E-06**
	(±SD)	*4.955395824*	*5.414365656*	*0.622184354*	***8.94894E-06***
Ackley	Media	5.259876609	5.382419569	0.324365569	**6.09717E-05**
	(±SD)	*0.384154298*	*0.38808497*	*0.444545366*	***5.45021E-05***
Griewank	Media	5.701261605	5.887341775	0.700857354	**5.35254E-09**
	(±SD)	*1.080435525*	*1.108359613*	*0.368310151*	***1.198E-08***
Six-Hump Camel- Back	Media	-1.031600318	-1.031628428	-1.031568182	**-1.031628452**
	(±SD)	*3.48248E-05*	*5.53445E-09*	*8.34751E-05*	***1.45999E-09***

Tabla 27. Valor Medio y desviación estándar ($\pm \boldsymbol{SD}$) de las pruebas de optimización (Nd = 50, NI=50.000).

16.3 EFECTOS DE LOS PARÁMETROS HCMR, HMS, PAR Y RCR

Para estudiar la variación de los parámetros HCMR, HMS, PAR y RCR se usaron los valores especificados en la **Tabla 36**. El número de dimensiones es 30, el número de iteraciones es 50.000 y el número de ejecuciones para cada prueba es 30.

La Tabla 28 muestra los efectos de la variación del parámetro HCMR en el algoritmo propuesto. Se observa que la precisión del algoritmo mejora con un HCMR más alto. Un HCMR alto (≥ 0.9) favorece la convergencia. En funciones de baja dimensionalidad como Six-Hump Camel-back se requiere un menor valor de HCMR para aumentar su capacidad de exploración. Es decir, un HCMR más bajo (0.7 por ejemplo) favorece la búsqueda del óptimo en funciones en las cuales se necesita una mayor exploración.

HCMR		**0.5**	**0.7**	**0.9**	**0.95**
Unimodales					
Sphere	Media	0.000197381	2.16632E-05	2.46402E-10	**3.33277E-11**
	(±SD)	*3.46488E-05*	*8.90181E-06*	*4.85473E-10*	***6.94329E-11***
Schwefel Problem 2.22	Media	0.046732628	0.008995544	4.78127E-05	**1.67511E-05**
	(±SD)	*0.00547272*	*0.003419191*	*3.78919E-05*	***1.41968E-05***
Rosenbrock	Media	23.02466444	**16.02993218**	26.47111615	32.26301449
	(±SD)	*32.03192831*	***14.24846561***	*32.66147183*	*108.4442821*
Step	Media	0	0	0	0
	(±SD)	***0***	***0***	***0***	***0***
Rotated Hyper-Ellipsoid	Media	2638.719935	**0.832511519**	380044196.6	1066841975
	(±SD)	*9217.946483*	***4.557631147***	*898797929.4*	*1890003832*
Multimodales					
Schwefel Problem 2.26	Media	-12569.48659	-12569.48662	**-12569.48662**	**-12569.48662**
	(±SD)	*5.24351E-06*	*1.07255E-06*	*4.07439E-11*	***1.62063E-11***
Rastrigin	Media	0.677007539	0.00421244	3.39398E-08	**4.2067E-09**
	(±SD)	*3.47513999*	*0.001785797*	*7.24178E-08*	***7.68495E-09***
Ackley	Media	0.010563105	0.003265899	8.99544E-06	**3.70719E-06**
	(±SD)	*0.001000305*	*0.000749123*	*9.99327E-06*	***3.20833E-06***
Griewank	Media	9.62872E-06	9.92445E-07	2.79283E-11	**2.95602E-12**

	(±SD)	*2.24672E-06*	*4.86263E-07*	*3.46684E-11*	***4.3954E-12***
Six-Hump Camel- Back	Media	-1.031628453	**-1.031628453**	-1.031628453	-1.031628448
	(±SD)	*3.15616E-10*	***2.0307E-10***	*1.36426E-09*	*1.88262E-08*

Tabla 28. Valor Medio y desviación estándar ($\pm \boldsymbol{SD}$) con variación de HCMR (Nd = 30, NI=50.000).

La Tabla 29 presenta los resultados de la variación del parámetro HMS en el algoritmo propuesto. Se observa que los mejores resultados para el algoritmo propuesto se encuentran ubicados entre las dimensiones 5 y 10 (80%). Le siguen las dimensiones mayores de 10 (20%). El algoritmo propuesto encuentra mejores resultados con tamaños de memoria armónica pequeños tal como se recomienda en el algoritmo HS original. La función Step, al ser discontinua, parece no ser afectada por este parámetro. Un trabajo futuro debe comprobar si al utilizar un histórico de las reglas se pueda mejorar la precisión del algoritmo aun utilizando un tamaño de memoria armónica menor.

HMS		**5**	**10**	**20**	**50**
Unimodales					
Sphere	Media	1.09049E-08	**9.42334E-09**	4.56347E-08	6.10931E-08
	(±SD)	*1.97455E-08*	***1.21108E-08***	*8.98288E-08*	*1.00668E-07*
Schwefel Problem 2.22	Media	**0.00058248**	0.000613678	0.000721842	0.000857497
	(±SD)	***0.000568687***	*0.000648931*	*0.00074417*	*0.000758539*
Rosenbrock	Media	**31.0074059**	48.32958186	39.0007948	31.84389228
	(±SD)	*51.12983659*	*57.16531525*	*47.41503805*	***35.13859472***
Step	Media	**0**	**0**	**0**	**0**
	(±SD)	***0***	***0***	***0***	***0***
Rotated Hyper-Ellipsoid	Media	30170.55111	**23222.86045**	33483.26924	30271.62452
	(±SD)	*53524.86969*	***42045.26373***	*89766.40829*	*103520.6057*
Multimodales					
Schwefel Problem 2.26	Media	**-12569.48662**	-12569.48662	-12569.48662	-12569.48596
	(±SD)	***4.53706E-08***	*1.32036E-07*	*7.77143E-08*	*0.00362494*
Rastrigin	Media	**2.95253E-06**	5.09339E-06	2.9867E-06	1.61558E-05
	(±SD)	***3.63668E-06***	*1.01325E-05*	*4.88117E-06*	*2.98689E-05*
Ackley	Media	**7.03661E-05**	8.6753E-05	7.74663E-05	0.000163219
	(±SD)	***5.66978E-05***	*0.000139645*	*6.64779E-05*	*0.000179311*
Griewank	Media	0.030470641	0.010369382	**0.002829622**	0.021107642

	(±SD)	*0.157625199*	*0.037402693*	***0.014132011***	*0.050745481*
Six-Hump Camel- Back	Media	-1.031628349	-1.031628331	-1.031628381	**-1.031628382**
	(±SD)	*1.22241E-07*	*1.37004E-07*	***7.88387E-08***	*1.01315E-07*

Tabla 29. Valor Medio y desviación estándar ($\pm SD$) con variación de HMS (Nd = 30, NI=50.000).

La Tabla 30 se ocupa de los resultados de la variación del parámetro PAR en el algoritmo propuesto. En la propuesta original de GHS [5] el PAR se ajusta dinámicamente con respecto al número de iteraciones [5, 14]. El algoritmo HS propuesto por Geem [6] establece que el parámetro PAR debe ser fijo y se recomienda 0.3. En este grupo de pruebas se establece un valor PAR constante de 0.1, 0.3, 0.5, 0.7, 0.9 y uno dinámico, como en IHS. Se observa que los mejores desempeños del algoritmo propuesto se obtienen cuando el par es dinámico para todas las funciones de optimización. En funciones no continuas el parámetro PAR parece no afectar la precisión del algoritmo. Aun en funciones cuya convergencia es lenta (Rotated Hyper-Ellipsoid) la mejor elección es un PAR dinámico. La diferencia entre los resultados de Schwefel Problem 2.26 para el PAR escogido y el PAR dinámico es de 0.003%, lo cual lo hace estadísticamente irrelevante.

PAR		**0.1**	**0.3**	**0.5**	**0.7**	**0.9**	**Dinámico**
Unimodales							
Sphere	Media	3.16E-09	2.43E-09	1.06E-08	1.55E-08	2.05E-08	**2.10E-10**
	(±SD)	*7.21E-09*	*6.07E-09*	*2.73E-08*	*2.94E-08*	*3.23E-08*	***3.60E-10***
Schwefel Problem 2.22	Media	1.45E-04	2.19E-04	2.41E-04	4.29E-04	5.84E-04	**5.70E-05**
	(±SD)	*1.05E-04*	*1.94E-04*	*2.80E-04*	*3.92E-04*	*5.55E-04*	***4.50E-05***
Rosenbrock	Media	2.67E+01	4.97E+01	5.35E+01	5.82E+01	2.55E+01	**1.58E+01**
	(±SD)	*4.16E+01*	*7.67E+01*	*7.03E+01*	*6.29E+01*	*4.82E+01*	***2.24E+01***
Step	Media	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
	(±SD)	***0.0***	***0.0***	***0.0***	***0.0***	***0.0***	***0.0***
Rotated Hyper-Ellipsoid	Media	3.37E+03	4.14E+03	1.16E+04	4.02E+03	1.77E+04	**6.86E+02**
	(±SD)	*1.12E+04*	*7.65E+03*	*2.13E+04*	*1.08E+04*	*5.43E+04*	***2.05E+03***
Multimodales							
Schwefel Problem 2.26	Media	-1.26E+04	-1.26E+04	-1.26E+04	-1.26E+04	-1.26E+04	**-1.26E+04**
	(±SD)	*1.16E-09*	*1.08E-09*	*8.46E-10*	*5.13E-09*	*6.67E-09*	***3.40E-11***
Rastrigin	Media	5.75E-07	7.07E-07	1.23E-06	1.41E-06	2.71E-06	**3.82E-08**
	(±SD)	*8.04E-07*	*1.20E-06*	*2.06E-06*	*2.43E-06*	*5.04E-06*	***6.07E-08***
Ackley	Media	2.37E-05	2.87E-05	3.51E-05	5.17E-05	9.03E-05	**5.83E-06**
	(±SD)	*2.40E-05*	*2.43E-05*	*3.35E-05*	*5.68E-05*	*7.93E-05*	***4.86E-06***

Griewank	Media	4.79E-02	3.85E-10	9.14E-10	4.92E-09	3.46E-09	**2.17E-11**
	(±SD)	*1.62E-01*	*5.39E-10*	*1.80E-09*	*1.94E-08*	*5.65E-09*	***4.60E-11***
Six-Hump Camel- Back	Media	-1.03162744E+00	-1.03162754E+00	-1.03162813E+00	-1.03162814E+00	-1.03162830E+00	**-1.03162845E+00**
	(±SD)	*1.30410791E-06*	*1.75798819E-06*	*3.98521037E-07*	*4.35232361E-07*	*1.90219051E-07*	***1.45999395E-09***

Tabla 30. Valor Medio y desviación estándar ($\pm SD$) con variación de PAR (Nd = 30, NI=50.000).

Los resultados de la variación del parámetro RCR se muestran en la Tabla 31. El parámetro RCR determina en que porcentaje se utilizarán las reglas inferidas del algoritmo adaptado PRISM, en la generación de una nueva armonía. Se observa una tendencia clara a usar un factor de uso de reglas, RCR, grande ($0.7 \leq RCR \leq 1.0$). El parámetro recomendado para RCR por defecto en el algoritmo propuesto es 0.9, el cual muestra mayor nivel de efectividad. Queda abierta la posibilidad de un estudio con un factor RCR variable entre 0.7 y 1.0.

LEMCR	**1.0**	**0.9**	**0.8**	**0.7**	**0.6**	**0.5**	**0.4**	**0.3**	**0.2**	**0.1**
Unimodales										
Sphere	**2.161E-10**	2.352E-10	2.453E-10	3.132E-10	2.729E-10	4.625E-10	1.399E-09	2.230E-09	4.838E-09	1.011E-08
	5.040E-10	*4.509E-10*	*4.745E-10*	*5.985E-10*	***4.207E-10***	*7.560E-10*	*2.929E-09*	*4.419E-09*	*9.146E-09*	*1.561E-08*
Schwefel Problem 2.22	**3.607E-05**	4.470E-05	5.970E-05	6.989E-05	7.995E-05	9.629E-05	1.086E-04	1.732E-04	2.777E-04	5.493E-04
	4.350E-05	***3.848E-05***	*7.632E-05*	*7.082E-05*	*8.713E-05*	*7.975E-05*	*1.053E-04*	*1.367E-04*	*3.002E-04*	*4.912E-04*
Rosenbrock	2.101E+01	1.861E+01	**1.476E+01**	3.032E+01	1.994E+01	2.617E+01	2.719E+01	7.098E+01	4.208E+01	6.856E+01
	3.121E+01	*3.042E+01*	***2.344E+01***	*3.910E+01*	*3.451E+01*	*3.375E+01*	*4.439E+01*	*1.682E+02*	*8.332E+01*	*1.333E+02*
Step	1.20	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
	5.94	***0.00***	***0.00***	***0.00***	***0.00***	***0.00***	***0.00***	***0.00***	***0.00***	***0.00***
Rotated Hyper-Ellipsoid	**1.514E+03**	1.858E+03	1.537E+03	2.253E+03	2.350E+03	3.560E+03	2.497E+03	3.740E+03	6.115E+03	6.009E+03
	3.680E+03	*4.963E+03*	***3.238E+03***	*5.492E+03*	*5.929E+03*	*7.180E+03*	*4.506E+03*	*6.954E+03*	*7.886E+03*	*6.796E+03*
Multimodales										
Schwefel Problem 2.26	-1.250E+04	**-1.257E+04**	**-1.257E+04**	**-1.257E+04**	**-1.257E+04**	-1.257E+04	-1.257E+04	-1.257E+04	-1.257E+04	-1.257E+04
	5.025E+02	***3.855E-11***	*5.829E-11*	*5.530E-11*	*6.038E-11*	*2.142E-10*	*7.002E-10*	*3.531E-10*	*5.424E-10*	*5.673E-04*
Rastrigin	1.194E+00	6.038E-08	5.753E-08	**5.499E-08**	8.028E-08	1.356E-07	2.565E-07	2.105E-07	7.543E-07	5.055E-06
	5.909E+00	*1.366E-07*	*1.132E-07*	***9.368E-08***	*1.141E-07*	*3.275E-07*	*3.396E-07*	*4.158E-07*	*1.073E-06*	*1.152E-05*
Ackley	1.430E-01	7.612E-06	**7.243E-06**	9.316E-06	1.307E-05	9.818E-06	1.325E-05	2.711E-05	3.759E-05	5.917E-05
	7.076E-01	***6.617E-06***	*6.937E-06*	*1.132E-05*	*1.028E-05*	*1.205E-05*	*1.479E-05*	*2.417E-05*	*3.438E-05*	*6.534E-05*
Griewank	2.272E-11	**1.208E-11**	1.400E-11	2.154E-11	8.018E-11	5.236E-11	1.672E-10	1.272E-02	2.786E-03	5.095E-02
	3.142E-11	***1.542E-11***	*2.240E-11*	*3.105E-11*	*2.327E-10*	*8.748E-11*	*4.402E-10*	*8.994E-02*	*1.957E-02*	*1.551E-01*

Six-Hump Camel- Back	-9.500E-01	**-1.032E+00**	-1.032E+00	-1.032E+00	-1.032E+00	-1.032E+00	-1.032E+00	-1.032E+00	-1.032E+00	-1.032E+00
	2.473E-01	***1.645E-09***	*2.202E-09*	*3.530E-09*	*3.106E-09*	*6.871E-09*	*1.025E-08*	*8.118E-09*	*4.938E-08*	*9.653E-08*

Tabla 31. Valor Medio y desviación estándar ($\pm \boldsymbol{SD}$) con variación de RCR (Nd = 30, NI=50.000).

Se realizaron otras pruebas variando el número de iteraciones como se muestra en la Tabla 32. El número de iteraciones para esta prueba es de 5.000. Se observa que el algoritmo GHS+LEM supera a las demás propuestas cuando el número de iteraciones es bajo. El algoritmo propuesto conserva un nivel de aproximación a la solución óptima aceptable a pesar del bajo número de iteraciones propuesto. Las diferencias en la función Six-hump Camel-back no son estadísticamente significativas.

		HMS	IHS	GHS	GHS+LEM
Unimodales					
Sphere	Media	1.391113338	1.492336778	0.008490508	**5.93279E-08**
	(±SD)	*0.442516349*	*0.429375922*	*0.015014868*	***9.80204E-08***
Schwefel Problem 2.22	Media	7.343708198	7.942415139	0.38849092	**0.000681506**
	(±SD)	*1.298620467*	*1.376467735*	*0.392227894*	***0.000587781***
Rosenbrock	Media	44801.42361	44335.30705	365.8780006	**35.91286497**
	(±SD)	*27087.38846*	*25827.36583*	*988.1136926*	***68.65861969***
Step	Media	577.17	599.34	3.25	**0**
	(±SD)	*178.0053373*	*181.6569761*	*8.62738768*	***0***
Rotated Hyper-Ellipsoid	Media	17744.96362	17590.00622	26959.75359	**11283.9062**
	(±SD)	***3863.897243***	*4062.189579*	*14265.59047*	*15864.62194*
Multimodales					
Schwefel Problem 2.26	Media	-11723.5473	-11734.68475	-12561.42156	**-12569.4597**
	(±SD)	*194.1725169*	*182.3846445*	*16.70991705*	***0.248116333***
Rastrigin	Media	29.46075697	36.89200558	0.881408659	**1.17383E-05**
	(±SD)	*5.461732931*	*6.233664508*	*1.594135534*	***1.89959E-05***
Ackley	Media	6.335464888	6.480145971	0.535233079	**0.000125684**
	(±SD)	*0.620693868*	*0.686288345*	*0.701961543*	***0.000103427***
Griewank	Media	6.277931588	6.370436585	0.795212512	**0.006467333**
	(±SD)	*1.494900647*	*1.816294084*	*0.350711873*	***0.054165186***
Six-Hump Camel- Back	Media	-1.03155243	**-1.031628431**	-1.026283458	-1.030628257
	(±SD)	*5.65151E-05*	***7.41828E-09***	*0.008075092*	*0.004327667*

Tabla 32. Valor Medio y desviación estándar ($\pm SD$) con variación de número de iteraciones (Nd = 30, NI=5.000).

Se compararon los resultados de GHS+LEM con 5.000 iteraciones con los resultados obtenidos por los demás métodos (HS, IHS y GHS) con 50.000 iteraciones (ver Tabla 33). Se observa que aún con un número de iteraciones bajo (5.000) el algoritmo propuesto mejora los resultados obtenidos en casi todas las funciones de optimización empleadas. En casos como Schwefel Problem 2.26 la diferencia no es estadísticamente significativa (0.000023%). Para funciones con convergencia lenta, como Rotated Hyper-Ellipsoid, se hace necesario un mayor número de iteraciones para mejorar la precisión del algoritmo propuesto.

		HMS	IHS	GHS	GHS+LEM (5000 NI)
Unimodales					
Sphere	Media	0.000684005	0.017838978	4.0457E-05	**5.93279E-08**
	(±SD)	*9.67781E-05*	*0.00710319*	*7.29366E-05*	***9.80204E-08***
Schwefel Problem 2.22	Media	0.143656975	0.997096357	0.040860755	**0.000681506**
	(±SD)	*0.047911784*	*0.200329207*	*0.037067055*	***0.000587781***
Rosenbrock	Media	312.2431152	423.9427774	72.47196696	**35.91286497**
	(±SD)	*486.5124844*	*330.6943507*	*103.3253058*	***68.65861969***
Step	Media	11.56	11.22	**0**	**0**
	(±SD)	*4.608555943*	*3.945538332*	***0***	***0***
Rotated Hyper-Ellipsoid	Media	4419.904558	**4238.371416**	5427.433309	11283.9062
	(±SD)	*1306.038774*	***1196.967797***	*6641.408049*	*15864.62194*
Multimodales					
Schwefel Problem 2.26	Media	-12545.01282	-12540.34846	**-12569.46257**	-12569.4597
	(±SD)	*9.274118296*	*10.54344883*	***0.03971048***	*0.248116333*
Rastrigin	Media	1.266797341	2.722732645	0.009457309	**1.17383E-05**
	(±SD)	*1.023021844*	*1.130249802*	*0.014012005*	***1.89959E-05***
Ackley	Media	0.981392208	1.584674315	0.024746761	**0.000125684**
	(±SD)	*0.485630315*	*0.331393069*	*0.026603311*	***0.000103427***
Griewank	Media	1.085396028	1.087082117	0.091022469	**0.006467333**
	(±SD)	*0.035098647*	***0.031926489***	*0.192952247*	*0.054165186*
Six-Hump Camel- Back	Media	-1.031600318	**-1.031628428**	-1.031568182	-1.030628257
	(±SD)	*3.48248E-05*	***5.53445E-09***	*8.34751E-05*	*0.004327667*

Tabla 33. Valor Medio y desviación estándar ($\pm SD$) comparación entre precisión con diferentes números de iteraciones (Nd = 30; GHS+LEM con NI=5.000; HS, IHS y GHS con NI=50.000).

Adicionalmente se realizaron pruebas para evaluar el desempeño del algoritmo en problemas de programación entera. Las pruebas implementadas fueron definidas en el artículo de GHS [5] y corresponden a cinco problemas definidos como F1,

F2, F3, F4 y F5, los resultados se observan en la **Tabla 47**. En los mismos se aprecia que GHS+LEM mejora la precisión para la función F1 en alta dimensionalidad con respecto a las demás propuestas. Para todas las demás funciones implementadas se observa que GHS+LEM se desempeña de forma similar a los otros algoritmos, mejorando los resultados de GHS en F2, F3 y F5.

		HS	**IHS**	**GHS**	**GHS+LEM**
F1 (N=5)	Media	**0**	**0**	**0**	**0**
	(±SD)	***0***	***0***	***0***	***0***
F1 (N=15)	Media	**0**	0.833333333	**0**	**0**
	(±SD)	***0***	0.461133037	***0***	***0***
F1 (N=30)	Media	6.266666667	11.26666667	0.433333333	**0**
	(±SD)	1.387961376	1.964044619	0.504006933	***0***
F2	Media	**0**	**0**	0.3	**0**
	(±SD)	***0***	***0***	0.70221325	***0***
F3	Media	**0**	9	0.133333333	**0**
	(±SD)	***0***	16.29258346	0.434172485	***0***
F4	Media	**-7**	**-7**	-6.933333333	-6.933333333
	(±SD)	***0***	***0***	0.253708132	0.253708132
F5	Media	**-3880**	**-3880**	-3879.633333	**-3880**
	(±SD)	***0***	***0***	0.556053417	***0***

Tabla 34. Problemas de programación entera.

17 SOFTWARE PARA PRESENTACIÓN GRÁFICA DEL EXPERIMENTO

Adicionalmente se desarrolló un software que permite representar gráficamente el proceso de optimización llevado a cabo por el algoritmo propuesto y los algoritmos HS, IHS y GBHS para cada una de las funciones de prueba. En el desarrollo del software se utilizó el componente de software TeeChart[9] para .Net, el cual permite generar gráficos matemáticos y estadísticos con base en series de datos.

[9] TeeChart es un componente de software desarrollado por Steema Software (http://www.steema.com/) que permite la presentación gráfica de datos matemáticos y estadísticos. Esta disponible para diferentes entornos y lenguajes de programación (PHP, JAVA, AX, .NET, AX, VCL, entre otros).

El funcionamiento del software se resume en tres pasos:

- Al ejecutar la aplicación el programa muestra una interfaz de usuario (Figura 28) la cual permite escoger la función que se va a optimizar, ajustar los valores para los parámetros PAR, HMS, HCMR, Número de Dimensiones, Número de Iteraciones, RCR y ejecutar la prueba respectiva.

- Al realizar la ejecución el sistema despliega un mensaje de confirmación (Figura 29) para indicar que se ha realizado la ejecución de las pruebas.

- Luego de recibir el mensaje, el sistema despliega una interfaz (Figura 30) que muestra gráficamente el proceso realizado por cada uno de los algoritmos hasta ejecutarse el número de iteraciones permitidas. Adicionalmente se puede rotar cada una de las gráficas que representan el proceso de optimización de los algoritmos GHS+LEM, GBHS, IHS y HS respectivamente y se muestra un control común que permite visualizar la gráfica correspondiente al número de iteración de cada algoritmo, ya sea manualmente o automáticamente.

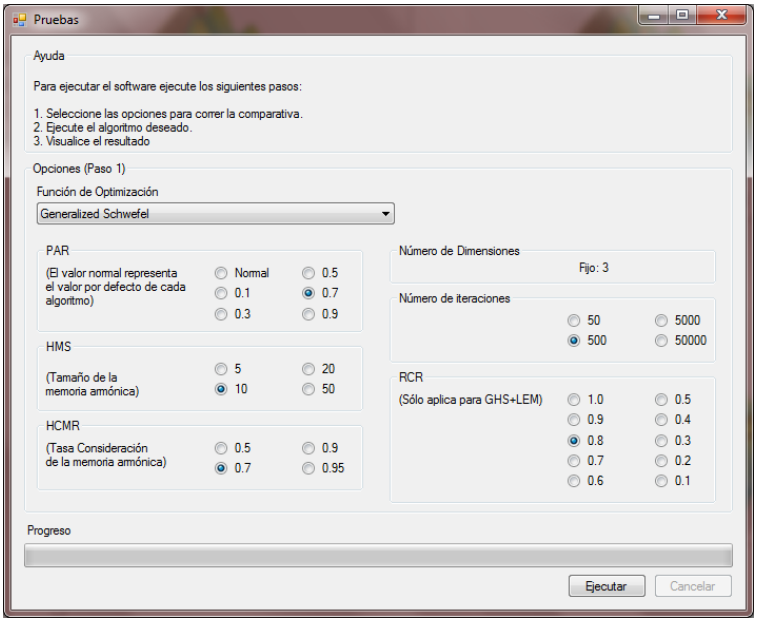


Figura 28. Interfaz del Software de Representación gráfica.

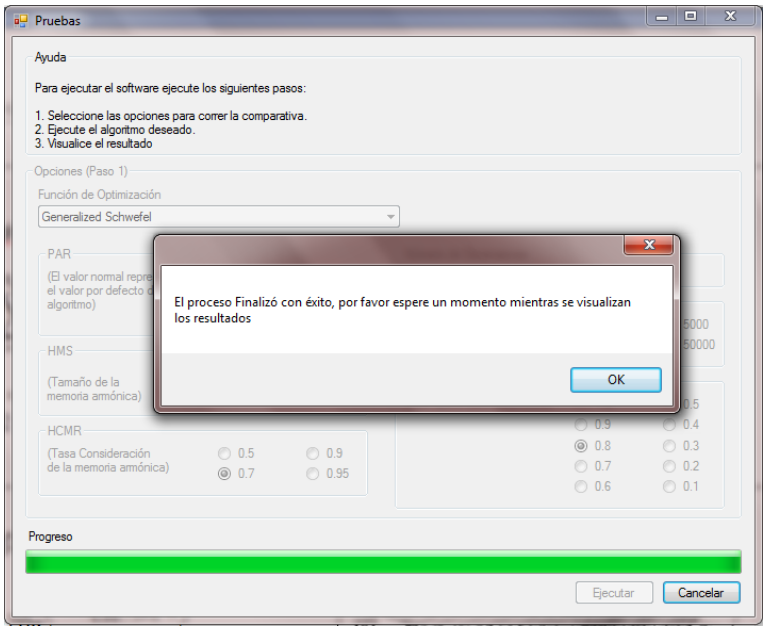


Figura 29. Mensaje de finalización de pruebas sobre la función seleccionada.

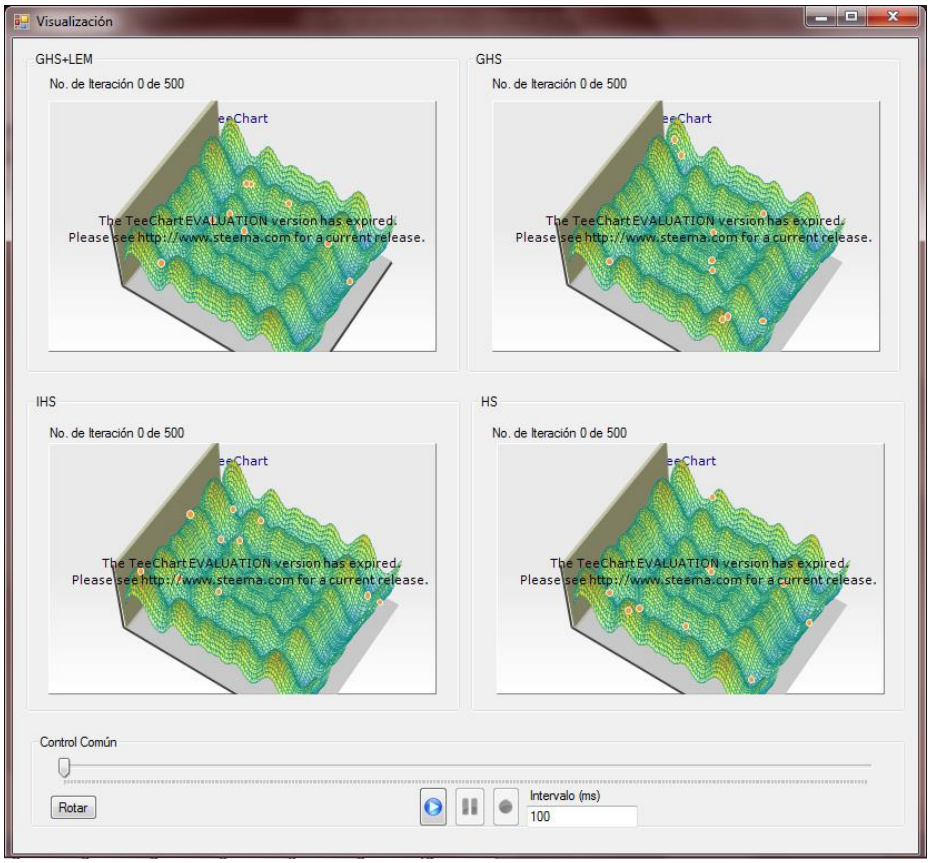


Figura 30. Resultados de forma gráfica de las pruebas realizadas sobre la función seleccionada.

Parte 6 – Conclusiones, Recomendaciones y Trabajo Futuro

18 CONCLUSIONES

Se diseñó, implementó y evaluó el algoritmo GHS+LEM el cual utiliza técnicas de LEM para crear un conjunto de reglas que permita inferir nuevos candidatos en la población que no surjan solamente a partir de la exploración aleatoria. La modificación permite que el nuevo algoritmo se desempeñe eficientemente tanto en funciones discretas como continuas. Se sometió el algoritmo a diez funciones de optimización clásicas y en la mayoría mejora los resultados obtenidos contra los demás métodos (HS, IHS, GHS). Los resultados obtenidos a través de una prueba de escalabilidad muestran que el algoritmo mantiene su precisión inclusive en altas dimensiones (≥30).

Los efectos de los parámetros HCMR, HMS, PAR y RCR en el desempeño del algoritmo propuesto muestran los siguientes resultados:

- Cuando se toman valores para HCMR ≥ 0.9, el algoritmo generalmente mejora su eficacia.
- En cuanto al tamaño de la memoria armónica (HMS) las pruebas muestran que el algoritmo propuesto generalmente se desempeña mejor cuando el tamaño está entre 5 y 10, lo cual es consecuente con las recomendaciones del algoritmo HS original.
- El valor de PAR obtiene mejores resultados cuando este es dinámico, coincidiendo con la propuesta de IHS.

- Con respecto a la variación del parámetro RCR se observó que se tiene un mejor desempeño general del algoritmo cuando el proceso de aplicación de las reglas es ejecutado con una probabilidad comprendida entre 0.7 y 1.

El algoritmo demuestra obtener una mejor precisión que las otras propuestas incluso cuando el número de iteraciones es 10 veces menor al usado por las otras propuestas armónicas.

El algoritmo HS ha demostrado ser una poderosa herramienta de optimización, la cual no necesita de cálculos matemáticos complejos para obtener soluciones óptimas a un problema determinado. Su éxito en las diversas aplicaciones en ingeniería civil, ingeniería mecánica y la medicina, así como el creciente desarrollo continuo de mejoras al algoritmo (GBHS, IHS y NGHS, entre otras) hacen de HS un algoritmo clave para la optimización general que permite adaptar su estructura y parámetros dependiendo del campo de aplicación.

19 RECOMENDACIONES Y TRABAJO FUTURO

Como trabajo futuro el grupo de investigación se propone estudiar el desempeño del algoritmo en problemas del mundo real; introducir modificaciones en el algoritmo de inferencia de reglas que permitan tener en cuenta históricos de la memoria armónica y modificar el proceso de generación de reglas para que se actualicen cada vez que se ejecute un cambio en la memoria armónica con el fin de evaluar el desempeño del algoritmo en estas nuevas condiciones.

Parte 7 – Glosario y Bibliografía

20 GLOSARIO

- **Optimización**: Es un proceso conocido también como programación matemática cuyo objetivo es intentar dar respuesta a un tipo general de problemas donde se desea elegir la mejor solución dentro de un conjunto de soluciones.

- **Dimensionalidad**: Hace referencia al número de grados de libertad para realizar un movimiento en el espacio.

- **Método Darwiniano**: Proceso que hace referencia a la teoría de la evolución de las especies propuesta por Charles Darwin, más conocida como Darwinismo.

- **Computación Evolutiva**: Es una rama de la inteligencia Artificial inspirada en los mecanismos de la evolución biológica. Esta compuesta por tres corrientes independientes conocidas como Programación Evolutiva, Estrategias Evolutivas y Algoritmos genéticos.

- **Función Unimodal**: Es una función matemática para la cual existe una única solución optima (expresada como un mínimo o un máximo).

- **Funciones Multimodal**: Es una función matemática para la cual existe más de una solución optima.

21 REFERENCIAS

1. Gauss, C.F., *Disquisitiones arithmeticae*. 1966, New Haven,: Yale University Press. xx, 472 p.
2. Dantzig, G.B., *Linear programming and extensions*. Reprinted with corrections. ed. 1968, Princeton, N.J: Princeton University Press. xvi, 632 p.
3. Dantzig, G.B., M.N. Thapa, and NetLibrary Inc., *Linear programming. 2, Theory and extensions*. 2003, Springer: New York.
4. Vanderbei, R.J. and SpringerLink (Online service), *Linear Programming Foundations and Extensions*. 2008, Robert J.Vanderbei: Boston, MA.
5. Omran, M.G.H. and M. Mahdavi, *Global-best harmony search.* Applied Mathematics and Computation, 2008. **198**(2): p. 643-656.
6. Geem, Z., J. Kim, and G.V. Loganathan, *A New Heuristic Optimization Algorithm: Harmony Search.* SIMULATION, 2001. **76**(2): p. 60-68.
7. Luke, S., *Essentials of Metaheuristics*, S. Luke, Editor. 2009, George Mason University.
8. Lee, K. and Z. Geem, *A new meta-heuristic algorithm for continuous engineering optimization: harmony search theory and practice.* Computer Methods in Applied Mechanics and Engineering, 2005. **194**(36-38): p. 3902-3933.
9. Glover, F., G.A. Kochenberger, and ebrary Inc., *Handbook of metaheuristics*, in *International series in operations research & management science 57*. 2003, Kluwer Academic Publishers: Boston. p. xii, 556 p.
10. Petrowski, J.D., et al., *Metaheuristics for Hard Optimization Simulated Annealing, Tabu Search, Evolutionary and Genetic Algorithms, Ant Colonies, Methods and Case Studies*. 2006, Springer-Verlag Berlin Heidelberg: Berlin, Heidelberg.
11. Rana, F., et al., *Hybridization of K-Means and Harmony Search Methods for Web Page Clustering*, in *Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology - Volume 01*. 2008, IEEE Computer Society.

12. Ehsan, S. and H. Leila Sharif, *Text summarization with harmony search algorithm-based sentence extraction*, in *Proceedings of the 5th international conference on Soft computing as transdisciplinary science and technology*. 2008, ACM: Cergy-Pontoise, France.
13. Forsati, R., A.T. Haghighat, and M. Mahdavi, *Harmony search based algorithms for bandwidth-delay-constrained least-cost multicast routing.* Comput. Commun., 2008. **31**(10): p. 2505-2519.
14. Mahdavi, M., M. Fesanghary, and E. Damangir, *An improved harmony search algorithm for solving optimization problems.* Applied Mathematics and Computation, 2007. **188**(2): p. 1567-1579.
15. Michalsky, R.S., *Learnable Evolution Model.* Machine Learning, 2000(38): p. 9-40.
16. Cendrowska, J., *PRISM: An algorithm for inducing modular rules.* International Journal of Man-Machine Studies, 1987. **27**(4): p. 349-370.
17. Geem, Z. and X.-S. Yang, *Harmony Search as a Metaheuristic Algorithm*, in *Music-Inspired Harmony Search Algorithm*. 2009, Springer Berlin / Heidelberg. p. 1-14.
18. Geem, Z., *State-of-the-Art in the Structure of Harmony Search Algorithm*, in *Recent Advances In Harmony Search Algorithm*, Springer Berlin / Heidelberg. p. 1-10.
19. Geem, Z.W., *Music-Inspired Harmony Search Algorithm: Theory and Applications.* 2009: p. 206.
20. Zou, D., et al., *A novel global harmony search algorithm for reliability problems.* Computers & Industrial Engineering, 2009. **58**(2): p. 307-316.
21. Ryszard, S.M., *LEARNABLE EVOLUTION MODEL: Evolutionary Processes Guided by Machine Learning.* Mach. Learn., 2000. **38**(1-2): p. 9-40.
22. Janusz, W. and S.M. Ryszard, *The LEM3 implementation of learnable evolution model and its testing on complex function optimization problems*, in *Proceedings of the 8th annual conference on Genetic and evolutionary computation*. 2006, ACM: Seattle, Washington, USA.

23. Wojtusiak, J. and R.S. Michalski, *The LEM3 implementation of learnable evolution model and its testing on complex function optimization problems*, in *Proceedings of the 8th annual conference on Genetic and evolutionary computation*. 2006, ACM: Seattle, Washington, USA. p. 1281-1288.
24. Eric, B. and S.M. Ryszard, *Data-Driven Constructive Induction.* IEEE Intelligent Systems, 1998. **13**(2): p. 30-37.
25. Kenneth, A.K. and S.M. Ryszard, *An Adjustable Description Quality Measure for Pattern Discovery Usingthe AQ Methodology.* J. Intell. Inf. Syst. **14**(2-3): p. 199-216.
26. Witten, I. and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems)*. 2005: Morgan Kaufmann.
27. Bramer, M., *Automatic Induction of Classification Rules from Examples Using N-Prism.* p. 23.
28. Serrano, C., *Modelo para la Investigación Documental.* **2**.
29. Michalski, R.S., *LEARNABLE EVOLUTION MODEL: Evolutionary Processes Guided by Machine Learning.* Mach. Learn., 2000. **38**(1-2): p. 9-40.
30. Witten, I.H. and E. Frank, *Data mining: practical machine learning tools and techniques with Java implementations.* SIGMOD Rec., 2002. **31**(1): p. 76-77.
31. M. Molga, C.S., *Test functions for optimization needs.* available at www.zsd.ict.pwr.wroc.pl/files/docs/functions.pdf 2005.
32. Xin, Y., L. Yong, and L. Guangming, *Evolutionary programming made faster.* Evolutionary Computation, IEEE Transactions on, 1999. **3**(2): p. 82-102.

Anexo A: Artículo “A Survey of Harmony Search” (Una Revisión de la Búsqueda Armónica)

A SURVEY OF HARMONY SEARCH
UNA REVISIÓN DE LA BÚSQUEDA ARMÓNICA

Carlos Cobos, José Pérez, Darío Estupiñan
Grupo de I+D en Tecnologías de la Información, Facultad de Ingeniería electrónica y Telecomunicaciones, Universidad del Cauca
Popayán (Cauca) - Colombia / Sector Tulcán Edificio FIET Oficina 422 - Tel. +57 (2) 8209800 extensión 2119 – Fax. +57 (2) 8209810
coboscarlos@gmail.com, j.perezhdez@hotmail.com, dario.estupinan@gmail.com

Abstract

The Harmony Search Algorithm is a meta-heuristic algorithm which bases its operation on the musical improvisation process. HS has been applied to many optimization problems, showing its efficiency compared to other meta-heuristics and mathematical optimization techniques. This paper reviews the development and evolution of the algorithm over the last 10 years, outlining the different areas of application and the most important improvements along the way

Keywords: Harmony Search Algorithm, Meta-heuristics, Optimization, Hybrid Methods, Harmony Search Applications.

Resumen

El algoritmo de búsqueda armónica (Harmony Search Algorithm) es un algoritmo meta heurístico que basa su funcionamiento en el proceso de la improvisación musical. La búsqueda armónica ha sido aplicado a infinidad de problemas de optimización, mostrando su eficiencia frente a otras meta heurísticas y otras técnicas matemáticas de optimización. Este artículo revisa el desarrollo y la evolución del algoritmo en los últimos 10 años, mostrando los diferentes campos de aplicación y sus mejoras más importantes.

Palabras Clave: Algoritmo de Búsqueda Armónica, Meta heurísticas, Optimización, Métodos Híbridos, Aplicaciones de Búsqueda Armónica.

1 INTRODUCCIÓN

El algoritmo de Búsqueda Armónica, conocido como HS por sus siglas en inglés (Harmony Search), es un algoritmo meta heurístico que ha basado su funcionamiento en el proceso de improvisación musical [1, 2]. La improvisación musical es una característica musical que no todo músico posee. Es un proceso en el cual la experiencia y el conocimiento previo de las armonías aportan a la calidad de la pieza que se está tocando. El éxito del HS frente a otros algoritmos evolutivos han hecho de este algoritmo una herramienta esencial para múltiples problemas de optimización, entre los cuales se encuentran el diseño de una red de distribución de agua [3, 4], el diseño Estructural [5], problemas relacionados con el transporte [1], la solución al juego del sudoku [6], entre otros; además, ha sido objeto de diversas variaciones con otras técnicas de optimización, como lo son GBHS [7], IHS [8] y NGHS [9].

La optimización es uno de los campos más importantes en ciencia e ingeniería. Cuando hablamos de minimización de tiempo, costos, materiales y espacio, nos referimos al objetivo principal de todo proyecto: optimizar. Este aspecto tan importante hoy en día es objeto de investigación y desarrollo en el campo científico, con lo cual se busca mejorar aspectos tan importantes en la industria como lo son la productividad y el costo; y el tiempo de procesamiento y la reducción de espacio en disco, en el campo computacional. HS ha cumplido con este objetivo a cabalidad, lo cual ha sido demostrado en diversas aplicaciones [10, 11].

A continuación, en la sección 2 se presenta en detalle el algoritmo HS, mostrando sus ventajas frente a otros algoritmos meta heurísticos; en la sección 3 se muestran las mejoras más importantes realizadas al algoritmo HS; luego en la sección 4 y 5 se presentan las diferentes aplicaciones y transformaciones que HS

ha tenido en los últimos 10 años (2000 - 2010), mostrando la tendencia en investigación y aplicación de esta meta heurística, y finalmente en la sección 6, se presentan las conclusiones generales del presente artículo.

2 BÚSQUEDA ARMÓNICA ORIGINAL

El algoritmo de la búsqueda armónica, propuesto por Zong Woo Geem y Kang Seo Lee [12] en el 2001, es un algoritmo meta heurístico, es decir, un algoritmo aproximado de propósito general que consiste en procedimientos iterativos que guían una heurística combinando de forma inteligente distintos conceptos para explorar y explotar adecuadamente el espacio de búsqueda [12]. HS simula el proceso de improvisación musical, en el cual los músicos buscan producir una armonía agradable determinada por el estándar estético auditivo [2]. Cuando un músico esta improvisando, el realiza una de las siguientes acciones:

4. Toca alguna melodía conocida que ha aprendido anteriormente.
5. Toca algo parecido a la melodía anteriormente mencionada, ajustándola poco a poco al tono deseado.
6. Compone una nueva melodía basándose en sus conocimientos musicales para seleccionar nuevas notas aleatoriamente.

Estas tres opciones formalizadas en [12], corresponden a los componentes del algoritmo: Uso de la memoria armónica, ajuste de tono y aleatoriedad, respectivamente.

En la improvisación musical, cada músico toca una nota dentro de un posible rango, de tal manera que forman un vector armónico. Si el conjunto de notas tocadas por los músicos son consideradas una buena armonía, esta es guardada en la memoria de cada músico, incrementando la posibilidad de hacer una buena armonía la próxima vez. Del mismo modo en el proceso de optimización en ingeniería, cada variable de decisión inicialmente toma valores aleatorios dentro del rango posible, formando un vector solución. Si dicho vector, es decir, dicho conjunto de valores que lo conforman son una buena solución, esta es

almacenada en la memoria de cada variable, aumentando la posibilidad de encontrar mejores soluciones en la siguiente iteración.

HS es un algoritmo meta heurístico basado en la población, lo cual indica que un grupo de múltiples armonías pueden ser usadas en paralelo. Este recurso usado apropiadamente tiende a obtener un mejor rendimiento con una alta eficiencia. El proceso de improvisación que lleva a cabo el algoritmo HS para optimizar una función hace de este una poderosa herramienta de programación. Al no realizar cálculos matemáticos complejos, el procesamiento es mucho más rápido, lo cual hace que el tiempo de convergencia del algoritmo marque la diferencia entre los diferentes algoritmos meta heurísticos existentes [2].

Los algoritmos de búsqueda armónica han sido utilizados para una gran variedad de problemas de optimización reales, tales como la composición musical, enrutamiento de viajes, diseño estructural, rutas de vehículos, diseño de tuberías de calor para satélites, entre otros [11]. La ventaja de los algoritmos armónicos frente a otros algoritmos evolutivos se basa en sus características, las cuales lo identifican y hacen de él una poderosa herramienta de optimización. Entre estas se cuentan: no requiere cálculos complejos, obvia óptimos locales y puede manejar variables discretas y continuas [1, 2]. Estas características permiten diferenciarlo de los algoritmos genéticos y hacen de él una poderosa herramienta basada en cálculos matemáticos simples e improvisación.

2.1 Descripción del Algoritmo HS

Los pasos que indican el funcionamiento del algoritmo HS son los siguientes:

Paso 1. **Inicializar los parámetros del problema y los parámetros de HS**: El problema de optimización se define como minimizar (o maximizar) $f(x)$ tal que $LB_i < x_i < UB_i$ donde, $f(x)$ es la función objetivo, x es una solución candidata que consiste de N variables de decisión (x_i), y LB_i y UB_i son el límite de decisión más bajo y el más alto de cada variable, respectivamente. Los parámetros de HS

se especifican en este paso. Estos parámetros son el tamaño de la memoria armónica (HMS), la tasa de consideración de la memoria armónica (HMCR), la tasa de ajuste del tono (PAR), el ancho de banda de ajuste del tono (BW) y el número de improvisaciones (NI).

Paso 2. **Inicializar la memoria armónica**: La memoria armónica inicial es generada desde una distribución uniforme en los rangos [LB_i, UB_i], donde $1 \leq i \leq N$. Esto se realiza de la siguiente manera: $x_i' = LB_i + r \times (UB_i - LB_i)$, donde j = *1,2,…, HMS* y $r \sim U(0,1)$. La variable r hace referencia a un número aleatorio y $U(0,1)$ a la función que genera el número aleatorio uniforme.

Paso 3. **Improvisar la nueva armonía**: El proceso de generación de una nueva armonía es llamado *improvisación*. El nuevo vector armónico, $x' = (x'_1, x'_2, \ldots, x'_N)$, se genera utilizando las siguientes reglas: consideración de la memoria, ajuste del tono y selección aleatoria. Este procedimiento se muestra en la Figura 1. En la línea 006, r es un número aleatorio uniforme entre 0 y 1, y el valor BW es un ancho de banda arbitrario de la distancia para variables de diseño continuas [13].

```
001 para cada i ∈ [1, N] hacer
002     si U(0,1) < HMCR entonces /*consideración de la memoria*/
003         inicio
004             x_i' = x_i^j, donde j ~ U(1, ..., HMS)
005             si U(0,1) ≤ PAR entonces /*ajuste del tono*/
006                 x_i' = x_i' + r × bw
007             fin_si
008         fin
009     sino /*selección aleatoria*/
010         x_i' = LB_i + r × (UB_i − LB_i)
011     fin_si
012 continuar_para
```

Figura 31. Improvisación de una nueva armonía

Paso 4. **Actualizar la memoria armónica**: El vector armónico generado, $x' = (x'_1, x'_2, \dots, x'_N)$, reemplaza la peor armonía almacenada en la memoria armónica si el *fitness* (o valor de aptitud del vector armónico actual, medido en términos de la función objetivo) es mejor que el de la peor armonía.

Paso 5. **Verificar el criterio de parada**: Terminar cuando el número máximo de improvisaciones (NI) se alcanza.

En general HS es poco sensible a los parámetros [1, 2], por esto, el algoritmo no requiere de una afinación exhaustiva de los parámetros para obtener buenas soluciones.

A pesar de lo anterior, es preciso destacar que los parámetros HMCR y PAR ayudan al método en la búsqueda de mejores soluciones globales y locales, respectivamente. PAR y BW tienen un profundo efecto en el desempeño del algoritmo y es por esto que el ajuste de estos dos parámetros es muy importante.

2.2 HS Frente a Otras Estrategias de Optimización

Un algoritmo meta heurístico tiende a ser más exitoso si emplea mecanismos que no requieran de un conocimiento complejo para que un programador o desarrollador lo entienda. La ventaja de muchos algoritmos meta heurísticos radica en que su simplicidad puede ayudar a encontrar errores lógicos y solucionarlos de forma exitosa. La simulación de procesos naturales es uno de los primeros responsables del éxito de dichos algoritmos frente a algoritmos que emplean cálculos y operaciones matemáticas [2].

Los algoritmos meta heurísticos son algoritmos heurísticos avanzados, los cuales hacen uso de técnicas de alto nivel y algunos procesos de ensayo y error, para encontrar la solución a un problema determinado. Las meta heurísticas son consideradas técnicas o estrategias de alto nivel que intentan combinar técnicas

de bajo nivel, y tácticas para exploración y explotación del espacio de búsqueda [1, 2].

Entre las más conocidas meta heurísticas se incluyen: los algoritmos evolutivos [14], entre los cuales se encuentran los algoritmos genéticos (Genetic Algorithm, GA) [14] y los algoritmos Meméticos (Memetic algorithm, MA) [14], el Recocido Simulado (Simulated Anealing) [15], la Búsqueda Tabú [16], la optimización basada en colonia de hormigas (Ant Colony Optimization) [17, 18], la Optimización basada en enjambre de partículas (Particle Swarm Optimization) [19], la optimización basada en el algoritmo de la abeja (bee algorithm) [20, 21], la optimización basada en el algoritmos de la luciérnaga (firefly algorithm) [21, 22] y los algoritmos armónicos (Harmony Search) [21].

Entre los algoritmos que usan meta heurísticas se destacan dos características principales: la diversificación y la intensificación. También son conocidos como exploración y explotación, los cuales son contradictorios entre sí, pero un adecuado balance entre los dos es determinante para el éxito de un algoritmo meta heurístico.

La diversificación o exploración, es el proceso por el cual el algoritmo explora en muchos lugares y regiones del espacio de soluciones, tanto como sea posible de una manera eficiente y efectiva. Esta característica pretende que el algoritmo no caiga en óptimos locales sin tener que recorrer todo el espacio de búsqueda.

La intensificación o explotación pretende aprovechar la historia y la experiencia del proceso de búsqueda, asegurando la velocidad de convergencia, cuando sea necesaria haciendo uso de la reducción de la aleatoriedad y limitando la diversificación. Este proceso consiste en volver a explorar zonas del espacio prometedoras (en las cuales se hallaron buenas soluciones), ya exploradas parcialmente.

El balance de estos dos componentes es una tarea propia de cada problema, pues lo que se busca es adecuar (afinar) los parámetros mismos de cada algoritmo para mejorar su funcionamiento. Por ejemplo, en la HS esta tarea ha sido llevada a cabo por los propios investigadores que han usado HS en diferentes campos de aplicación [11] y han determinado ciertos estándares sobre los cuales HS puede determinar mejores soluciones para un determinado problema.

En HS, la diversificación es controlada por la tasa de ajuste de tono y la aleatoriedad, con lo cual se busca refinar los valores obtenidos anteriormente, es decir encontrar nuevos valores cercanos a los mejores valores ya obtenidos. Esto asegura que las buenas soluciones locales ya obtenidas sean guardadas en memoria mientras que la aleatoriedad hace que el algoritmo explore el espacio de búsqueda global de manera efectiva. Por otro lado, la intensificación en HS es representada por el rango de aceptación de la memoria armónica. Un alto rango de valor de aceptación indica que mejores soluciones serán almacenadas en memoria. Si es demasiado bajo, la convergencia se tornara demasiado lenta. Teniendo en cuenta estas características, se puede observar que la interacción entre estos dos componentes asegura el éxito del algoritmo HS frente a otros algoritmos, como ya ha sido probado en [5, 7, 23-27].

3 MEJORAS REALIZADAS AL ALGORITMO HS

HS es un algoritmo que ha sido aplicado en diversos problemas reales y sobre el cual se han realizado gran variedad de transformaciones, obteniendo resultados satisfactorios y motivando que la investigación sobre este algoritmo meta heurístico se haya incrementado en los últimos años [1, 2]. Entre las trasformaciones, o hibridaciones que ha sufrido HS se destacan las siguientes: la búsqueda armónica mejorada (IHS, Improve Harmony Search) [8] , la búsqueda armónica global (GBHS, Global-Best Harmony Search) [7]) y la nueva búsqueda armónica global (NGHS) [28]. Estos nuevos algoritmos han tenido en cuenta para su desarrollo las diversas técnicas de optimización existentes (Particle Swarm Optimization, PSO y mutación genética) y el ajuste dinámico de parámetros, para

disminuir el ruido en el algoritmo original, soportar mayor dimensionalidad y aumentar la precisión y el grado de convergencia en las soluciones, entre otras.

3.1 Búsqueda Armónica Mejorada (IHS)

La búsqueda armónica mejorada (*Improved Harmony Search* o IHS por sus sigla en inglés), es un algoritmo armónico propuesto en el año 2007 por Mahdavi et al [8], que emplea un novedoso método para la generación de nuevos vectores solución basado en el ajuste dinámico de los parámetros PAR (tasa de ajuste del tono) y BW (ancho de banda de la distancia), logrando con esto mejorar la precisión y la velocidad de convergencia. Ha sido probado en problemas de optimización en ingeniería y evaluado frente a otros algoritmos con éxito. Esta variante de la búsqueda armónica modifica el paso 3 del algoritmo original, el paso en el que se crea un nuevo armónico. PAR y BW cambian dinámicamente con el número de generaciones y se calculan con las fórmulas de la Figura 2.

$$PAR(gn) = PAR_{min} + \frac{(PAR_{max} - PAR_{min})}{NI} \times gn$$

Donde,

PAR Tasa de ajuste del tono para cada generación

PAR_{min} Tasa mínima de ajuste del tono

PAR_{max} Tasa máxima de ajuste del tono

NI Número de generaciones de vectores solución

gn Número de generaciones

y

$$bw(gn) = bw_{max} \exp(c \times gn)$$

$$c = \frac{Ln\left(\frac{bw_{min}}{bw_{max}}\right)}{NI}$$

Donde,

$bw(gn)$	Ancho de banda de la distancia para cada generación
bw_{min}	Ancho de banda de la distancia mínimo
bw_{max}	Ancho de banda de la distancia máximo

Figura 32. Ecuaciones que rigen el cambio de los parámetros PAR y BW en el algoritmo IHS

El parámetro PAR crece linealmente con el número de generaciones, mientras que bw decrece exponencialmente, como se puede apreciar en la Figura 3. Con este cambio en los parámetros, IHS logra mejorar el rendimiento de HS, ya que encuentra mejores soluciones tanto a nivel global como local.

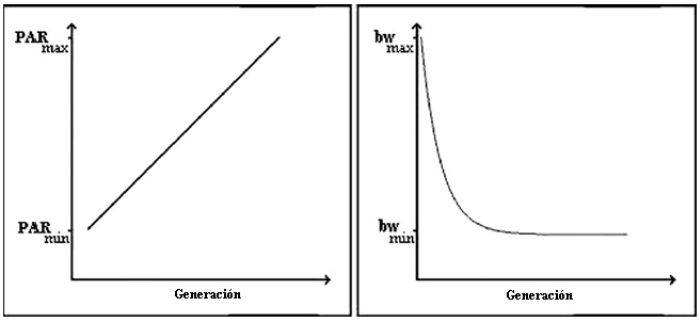


Figura 33. Variación de PAR y BW versus el número de generaciones (Tomada de [8]).

3.2 Mejor Búsqueda Armónica Global (GBHS)

Global-Best Harmony Search (GBHS) es un algoritmo de optimización estocástica propuesto en el año 2008 por Mahamed G.H. Omran y Mehrdad Mahdavi [7], el cual hibrida la búsqueda armónica original con el concepto de inteligencia de enjambre propuesto en PSO [7], donde un enjambre de individuos (llamados partículas) vuela a través del espacio de búsqueda. Cada partícula representa un candidato a la solución del problema de optimización. La posición de una partícula está influenciada por la mejor posición visitada por sí misma (es decir, su propia experiencia) y la posición de las mejores partículas en el enjambre (es decir, la experiencia de enjambre). GBHS modifica el paso de ajuste del tono en HS de modo que el nuevo armónico puede imitar el mejor armónico en la memoria

armónica. Esto permite a GBHS trabajar eficientemente en problemas continuos y discretos. En general GBHS es mejor que IHS y HS, por ejemplo cuando se aplica a problemas de gran dimensionalidad y cuando hay presencia de ruido [7].

El GBHS tiene exactamente los mismos pasos que IHS con la salvedad de la modificación del paso 3 que corresponde a la improvisación de un nuevo armónico, el cual se modifica conforme a la Figura 4.

001 **para cada** $i \in [1, \mathrm{N}]$ **hacer**
002 **si** $U(0,1) < HMCR$ **entonces** */*consideración de la memoria*/*
003 **inicio**
004 $x_i' = x_i^j$, donde $j \sim U(1, \ldots, HMS)$
005 **si** $U(0,1) \leq PAR(t)$ **entonces** */*ajuste del tono*/*
006 $x_i' = x_k^{best}$, donde $best$ es el índice de la mejor armonía en la HM y $k \sim U(1, N)$
007 **fin_si**
008 **fin**
009 **sino** */*selección aleatoria*/*
010 $x_i' = LB_i + r \times (UB_i - LB_i)$
011 **fin_si**
012 **continuar_para**

Figura 34. Improvisación en el algoritmo de la mejor búsqueda armónica global (GBHS)

3.3 Nueva Búsqueda Armónica Global (NGHS)

La nueva búsqueda armónica global (novel global harmony search, NGHS) [28] propone una hibridación entre la búsqueda armónica, la optimización basada en enjambre y la mutación genética. Este algoritmo se propuso para resolver problemas de fiabilidad, término usado para indicar la probabilidad de que un sistema funcione correctamente (sea seguro y eficiente) sin violar ninguna restricción. De los diversos problemas de fiabilidad que existen se tomaron: los

sistemas complejos (sistemas de programación no lineal entera y mixta), un sistema de protección de exceso de velocidad para una turbina de gas y un problema de fiabilidad de un sistema a gran escala.

NGHS y HS son diferentes en tres aspectos: en el paso 1, la tasa de consideración de la memoria armónica (HMCR) y el rango de ajuste de tono (PAR) son excluidos, y se incluye la probabilidad de mutación genética (p_m); en el paso 3, se modifica el paso de la improvisación por completo de modo que la nueva armonía imite a la mejor armonía global en la memoria armónica (HM) como se muestra en la Figura 5; y en el paso 4, se reemplaza la peor armonía x^{worst} con la nueva armonía x' aun sí x' es peor que x^{worst} [28].

001 **para cada** $i \in [1, N]$ **hacer**
002 $x_R = 2 \times x_i^{best} - x_i^{worst}$
003 **si** $x_R > x_{iU}$ **entonces**
004 **inicio**
005 $x_R = x_{iU}$
006 **fin**
007 **sino**
008 **si** $x_R < x_{iL}$ **entonces**
009 $x_R < x_{iL}$
010 **fin_si**
011 **fin_si**
012 $x_i' = x_i^{worst} + r \times (x_R - x_i^{worst})$ *// % actualización de la posición*
013 **si** $rand(\ ) \leq p_m$ **entonces**
014 $x_i' = x_{iL} + rand(\ ) \times (x_{iU} - x_{iL})$ *// % mutación genética*
015 **fin_si**
016 **fin_para**

Figura 35. Nueva Improvisación en el algoritmo de la nueva búsqueda armónica global (NGHS).

Esta propuesta se ha probado en optimización de problemas discretos y muestra una mejor convergencia y una mayor capacidad de exploración del espacio de solución que el algoritmo de búsqueda armónica original (HS) y que la búsqueda armónica mejorada (IHS).

4 USOS DE HS Y SUS VARIACIONES

HS ha sido considerado por muchos autores como un algoritmo muy exitoso, compitiendo con otras meta heurísticas como lo son PSO, Tabu Search –TS- y GA. En los últimos años, varias investigaciones se han desarrollado alrededor de HS, y se ha aplicado a diversos problemas de optimización en ciencia e ingeniería, entre los cuales se incluyen: aplicaciones cotidianas, aplicaciones en ciencias de la computación, aplicaciones en ingeniería eléctrica, aplicaciones en ingeniería civil, aplicaciones en ingeniería mecánica, aplicaciones biomédicas, aplicaciones en economía, aplicaciones en transporte, entre otras [11, 29, 30].

4.1 Aplicaciones Cotidianas

Cuando se habla de problemas cotidianos, se hace referencia a la aplicación de HS en la solución de problemas comunes en el desarrollo humano y para las cuales no hay una ciencia o campo de investigación adecuado que lo identifique. Algunas de estos problemas son la composición musical, resolver un sudoku y la planeación de un tour. Todas las anteriores, son tareas para las cuales se necesita de experiencia y de conocimiento previo de diversas herramientas o técnicas que permitan llevarlas a cabo con un grado de satisfacción adecuado.

La composición musical no solo depende de la experiencia y de los conocimientos en música, sino también del buen gusto y de la improvisación. Cuando se aplicó HS para componer piezas musicales en el estilo medieval, se logró teniendo en cuenta que una línea armónica (vox organalis) se compone para acompañar a la melodía del canto gregoriano (vox principalis). El órgano, el cual es el instrumento usado para acompañar este canto, tiene las siguientes reglas de composición: la línea armónica avanza en paralelo; para el movimiento paralelo, el intervalo de la

cuarta perfecta es referido y en orden de distinguir la vox principalis de la vox organalis, el primero siempre debe estar situado por encima del último [10]. Aprovechando las características y analogías del algoritmo con el proceso de improvisación musical, se modificaron los parámetros conforme a las reglas de composición y se adecuó el algoritmo para componer exitosamente líneas sobre la base de la armonía original de las líneas de canto gregoriano [31].

El juego del sudoku puede verse como un problema de optimización con numerosas penalidades únicas. Varios métodos han sido utilizadas para la solución de este problema, entre las cuales se encuentran: la teoría de grafos, la inteligencia artificial y los algoritmos genéticos [6]. El algoritmo de HS fue probado con éxito para encontrar una solución óptima sin ninguna violación de las tres reglas del juego en 285 problemas de sudoku con diferente nivel de dificultad.

La programación o planeación de actividades también son un problema común en la vida de las personas. Muchos de estos problemas tienden a tener una serie de restricciones (tiempo, espacio, orden, etc.) que son casi imposibles de controlar, lo que implica que no es posible tener una solución optima en un tiempo determinado. Este tipo de problemas son conocidos como NP completos. Los problemas NP completos son problemas en los cuales ninguna solución dada puede ser rápidamente verificada, y son también problemas NP-hard, lo cual indica que ningún problema NP puede ser convertido en único si se transforman su entradas en tiempo polinomial.

En el caso de la programación de una agenda o un horario, HS ha sido aplicado a la programación de cursos universitarios [26], el cual es un problema NP completo, que consta de 3 restricciones fuertes y 3 restricciones suaves. Las restricciones fuertes deben ser cumplidas para que la solución sea usable, mientras que las restricciones suaves son deseables pero no absolutamente esenciales. Para resolver este problema, se realizó la adaptación de HS a las restricciones del problema, y se realizó la comparación del algoritmo HS propuesto frente a otros

algoritmos que también fueron aplicados a este problema. El algoritmo planteado en [26] se destaca por ser capaz de encontrar un equilibrio entre la exploración a través HMCR en la consideración la memoria y explotación a través de PAR en el procedimiento de ajuste de tono.

4.2 Aplicaciones en Ciencias de la Computación

El algoritmo HS ha sido aplicado recientemente en muchas aplicaciones en ciencias de la computación e ingeniería. El clustering o agrupación de páginas web, el resumen o la sumarización de texto, el enrutamiento en internet y la robótica son algunas de las áreas en las cuales HS ha sido aplicado obteniendo resultados satisfactorios.

El clustering es un problema de gran importancia práctica que ha sido el foco de investigación sustancial en varios dominios por décadas [23]. Teniendo en cuenta las dimensiones y las propiedades de los documentos, el clustering tiende a ser una tarea mucho más difícil cuando se trata de documentos web. HS ha sido aplicado con éxito al problema del clustering de páginas web, tanto para una representación continua de datos [32] como para una representación discreta de datos [25]. HS basado en clustering ha sido hibridado con el algoritmo de k-means obteniendo como resultado HSCLUST [33]. Los resultados muestran que el clustering de documentos web basado en HS es una buena elección cuando se trata de particionar grandes cantidades de documentos [23].

IGBHSK [34] es un algoritmo resultante de la hibridación de GBHS y K-means para resolver el problema del clustering de documentos web. IGBHSK tiene la capacidad de definir automáticamente el número de clusters para un problema dado. Este algoritmo muestra mejores resultados cuando se usa con una matriz de términos frecuentes por documento, en el marco de un modelo vectorial de representación de documentos, cuando el criterio de información bayesiana es usado como función de aptitud y cuando se usa la similitud de cosenos para comparar los documentos [34].

El seguimiento visual (visual tracking) es usado comúnmente en sistemas para identificar correctamente un objetivo arbitrario en una secuencia de video. Dicho objetivo tiende a sufrir constantes cambios, para lo cual el sistema debe estar preparado. En este tipo de problemas se debe tener en cuenta la posibilidad de que el objetivo aparezca y desaparezca, cambie de tamaño, sea cubierto por otro objeto en la secuencia de video, con el objetivo de ubicarlo rápida y eficientemente. Los métodos más populares usados en seguimiento visual son el filtro Kalman y el filtro de partículas. HS e IHS han sido usados para crear el Harmony Filter [24], el cual usa el coeficiente de Bhattacharyya para comparar los histogramas de color en la secuencia de video. HF fue probado y comparado en diversas situaciones con el filtro Kalman y el filtro de partículas, obteniendo mayor exactitud en el proceso de seguimiento y recobro del objetivo en situaciones en las que los otros dos algoritmos no lo lograron.

La robótica es una de las ciencias en las cuales HS ha sido aplicado con éxito frente a otros algoritmos evolutivos y otras técnicas matemáticas. El cálculo de las coordenadas de los movimientos de los robots es el fuerte en este tipo de aplicaciones, entre las cuales se encuentran el cálculo de trayectorias optimas [35] y una aplicación para un prototipo de robot móvil reconfigurable [36].

En el cálculo de trayectorias óptimas [35], HS se usó para minimizar el tiempo de duración del movimiento, que es la restricción principal de este tipo de problemas que tienen como objetivo principal aumentar la productividad, para robots usados en el campo de la industria, en este caso, de un robot manipulador con 6 grados de libertad. HS es hibridado con la programación cuadrática secuencial (SQP, siglas de Sequential Quadratic Programming), dando como resultado HHSA con el fin de mejorar las soluciones para trayectorias calculadas, haciendo uso de la aleatoriedad con el fin de encontrar los valores iníciales óptimos que han de ser usados en SQP para calcular los vectores en el paso 3 del algoritmo. HS, SQP y

HHSA fueron comparados demostrando que HHSA es más eficiente que HS y que SQP en el cálculo de las trayectorias.

HS fue usado exitosamente en un prototipo de robot móvil reconfigurable [36]. HSMOO (Optimización Multi-objetivo con Búsqueda Armónica) se desarrolló para reconfigurar el diseño del robot con base a las restricciones planteadas y a la mecánica del suelo. HSMOO utiliza dos memorias armónicas, la primera para la evolución de los miembros armónicos, y la otra es considerada como un repositorio de almacenamiento externo para las soluciones optimas de pareto que han sido encontradas. Se hace uso de la optimización de Pareto como criterio de rango para ordenar las soluciones optimas en la primera memoria armónica.

La llegada de varias aplicaciones multimedia en tiempo real en redes de alta velocidad crea una necesidad de calidad de servicio (QoS) de enrutamiento basado en multidifusión. Dos de las restricciones más importantes en QoS son la restricción de ancho de banda y la restricción de retardo de extremo a extremo. El problema de la calidad del servicio en el enrutamiento basado en multidifusión es conocido como un problema NP completo que depende de: la demora limitada de extremo a extremo, el ancho de banda de enlace a lo largo de los caminos de la fuente a cada destino y el costo mínimo del árbol de multidifusión. En [37] se presentan dos nuevos algoritmos centralizados para resolver el problema de multidifusión de menor costo con restricción de retardo en el ancho de banda basado en HS. HSPR y HSNPI [37], los cuales fueron evaluados en rendimiento y eficiencia frente a un algoritmo basado en GA y una versión modificada del algoritmo BSMA. Los resultados de la simulación de redes generadas aleatoriamente y topologías reales indican que el algoritmo HSNPI propuesto ha superado a los otros tres algoritmos.

4.3 Aplicaciones en Ingeniería Eléctrica

HS ha sido ampliamente utilizado en problemas de optimización compleja aplicados a la ingeniería eléctrica, puesto que ha demostrado ser una alternativa

viable frente a otras técnicas de optimización tradicional aplicadas normalmente en esta área.

El método que permite conocer la forma más eficiente, con el menor costo y que mantiene operando de manera confiable un sistema de alimentación eléctrica, respetando las limitaciones operacionales de los recursos de generación disponibles es denominado Despacho Económico (Economic Dispatch) [1]. Varios métodos de programación matemática han sido empleados para resolver problemas de Despacho económico, tales como programación lineal, programación no lineal y programación homogénea [1, 38, 39]. Sin embargo, para que estos métodos sean realmente eficientes las variables y la función de costo necesitan ser continuas y tener un buen punto de inicio. Considerando el uso de varios tipos de combustibles, en la implementación se pueden encontrar funciones no convergentes, no suaves, zonas operativas prohibitivas, entre otras; lo cual hace que este tipo de problemas no sean resueltos por modelos tradicionales de programación matemática, y se apliquen modelos de programación dinámica y programación mixta no lineal entera, aunque estos métodos tienden a crear una expansión de las dimensiones del problema [39]. HS ha sido usado en esta área como una alternativa de solución a los métodos tradicionales, debido a su capacidad de exploración para encontrar las regiones de alto rendimiento dentro del espacio en un tiempo razonable. Su aplicación es una alternativa a uno de los problemas más difíciles de Despacho Económico que surge en Sistemas de Calor y Energia (CHP Systems) [38].

Dentro del diseño de sistemas eléctricos, se busca encontrar el mejor y seguro punto de operación correspondiente a cada situación de carga [29]. En general, los problemas de diseño de sistemas eléctricos son problema altamente no lineales con restricciones y de gran escala. HS ha sido usado para la solución de este tipo de problemas eléctricos en una versión modificada llamada Búsqueda Armónica con Varianza de la Población (PVHS) [40], en donde “el parámetro de control conocido como ancho de banda (bw) ha sido igualado a la desviación

estándar de la población actual" (traducción libre) [40]. Los resultados del algoritmo han demostrado su efectividad tras ser sometidos al sistema de pruebas IEEE 30 [29, 40].

En el campo de la detección foto-electrónica, una versión llamada búsqueda armónica adaptable (Adaptative Harmoy Search, AHS) ha sido usada para resolver el problema de Onda de densidad de fotón (Photondensity wave), dentro de la cual, "la memoria armónica se ajusta durante todo el proceso de búsqueda teniendo en cuenta la tasa HCMR y el ajuste de tono (PAR)" (traducción libre) [41]. Este método es usado para localizar anormalidades incrustadas en la muestra de acuerdo con los parámetros de la ubicación a diagnosticar [41].

En la optimización de inversores multinivel, se presenta un nuevo método basado en HS [42] para optimizar la forma de onda armónica escalonada para este tipo de inversores. "El método propuesto tiene como ventaja la alta tasa de convergencia y precisión en comparación con otros métodos convencionales de optimización" (traducción libre) [42]. La técnica propuesta se puede aplicar a convertidores multinivel con cualquier número de niveles. Como caso de estudio, el método es aplicado y probado en un inversor de 13 niveles [42]. Los resultados de la simulación muestran la efectividad y la flexibilidad del método propuesto [42].

En el campo de las redes móviles, se propone un nuevo método de optimización guiado aleatoriamente para la detección multiusuario en DS-CDMA (Direct-Sequence Code Division Multiple Access), la cual es una técnica de modulación usada en el ámbito de las telecomunicaciones. HS se ha usado en conjunto con el algoritmo de detección multiusuario en un entorno de trabajo de decodificación de canal. Los resultados muestran que se puede alcanzar soluciones muy cercanas a las esperadas sin el empleo de una búsqueda optima de detección multiusuario incluso en sistemas con alto nivel de carga [43].

4.4 Aplicaciones en Ingeniería Civil

Las aplicaciones de HS en ingeniería civil cubren el diseño estructural, diseño de redes hidráulicas, calibración de un modelo de inundación, administración de aguas subterráneas, análisis de estabilidad de suelos, conservación ecológica, enrutamiento de vehículos, entre otros.

En diseño estructural, la minimización de costos, materiales y espacio, son unas de las características más importantes en el diseño, desarrollo e implementación de una estructura o construcción. En el campo de la ingeniería civil, la optimización de estos recursos juega un papel importante, que no sólo depende de los conocimientos y estudios obtenidos por medio de la profesión, sino también de la experiencia que se obtenga a lo largo de la misma. HS ha sido objeto de múltiples aplicaciones relacionadas con la construcción, el diseño de redes y la optimización de recursos, obteniendo excelentes resultados frente a otros algoritmos en la minimización del costo total de la construcción (producción), que para los ingenieros civiles, es el factor principal y lo que toda empresa busca, no dejando de lado la calidad de los materiales y de la construcción en sí misma.

Cuando se habla de diseño estructural, se hace referencia a la estructura o marco de la construcción, es decir, a las vigas y columnas de acero que soportaran el peso de la edificación [44, 45]. En esta parte, se tienen en cuenta la toma de decisiones acerca de las dimensiones de sección trasversal de los miembros que constituyen la estructura, y algunas veces, la geometría y topología de la propia estructura. Cuando se diseña un marco de acero, el proceso de toma de decisiones implica la escogencia de una sección W (wide flange) [46] para las secciones transversales y longitudinales de las vigas o algún otro tipo de sección de acero que esté disponible en la tabla de secciones de acero de tal manera que la respuesta a cargas externas este dentro de los límites impuestos en el código de diseño de acero [44].

En el diseño de redes hidráulicas, el objetivo principal del uso del HS, es optimizar los costos teniendo en cuenta las dimensiones de los tubos empleados en la red. La configuración de una red hidráulica depende de la selección óptima de los diámetros de las tuberías. HS se adecúo para realizar esta labor, en la cual el algoritmo selecciona el diámetro para cada segmento de la tubería con el fin de minimizar el costo total de la red. Para esto se debe tener en cuenta las restricciones impuestas por la mecánica de fluidos, es decir: ecuaciones de masa y conservación de la energía, con lo cual se determina la presión mínima a la que debe estar sometida cada sección (segmento de la tubería) [3, 4]. Para probar las soluciones en [4], se usó EPANET [47], un paquete de software que permite evaluar la calidad del agua en sistemas de distribución de agua potable. El sistema fue diseñado para realizar la simulación de las diferentes redes que se diseñan teniendo en cuenta aspectos como la presión en cada tubo, simulación del flujo y la calidad del agua en la red, la presión en la conexión de los tubos, la altura del agua en cada tanque y la concentración de una sustancia disuelta en cada cruce durante uno o varios periodos de simulación. Además, la concentración, la edad del agua y la fuente de seguimiento también pueden ser simuladas.

Una represa es una estructura de barrera para retención de agua con fines de riego, abastecimiento urbano de agua, navegación, usos industriales, generación de energía hidroeléctrica, usos recreativos, la creación de un hábitat de vida silvestre y el control de inundaciones [48]. En este tipo de problemas, muchos investigadores han utilizado técnicas de programación dinámica en sus soluciones, pero este enfoque tiende a tener problemas de dimensionalidad, es decir, mayores requerimientos de memoria y mayores tiempos de computación. “Esto ha hecho que en los últimos años los algoritmos meta heurísticos como GA, SA y HS hayan sido considerados como el nuevo enfoque de solución a este tipo de problemas” (traducción libre) [48]. El objetivo de optimizar la programación de una represa esta dado por la maximización de los beneficios para la generación de energía hidroeléctrica y la irrigación, satisfaciendo las restricciones propias del

problema, entre estas: el rango del flujo de agua, el rango de presas almacenadas y la conservación de masa entre el flujo de entrada y salida [48]. HS mostró ser mucho más eficiente que otros algoritmos al ser evaluado en un sistema de 4 presas [48].

Una inundación es un desbordamiento de agua que sumerge una porción de tierra. “Son el resultado de situaciones en las cuales el volumen de un cuerpo de agua (rio, lago, laguna) excede el límite de su capacidad total” (traducción libre) [48]. Las inundaciones son causantes de muchos problemas, entre estos: los daños a infraestructura, las epidemias, la contaminación del agua, escasez de cosechas y dificultades económicas. “Por esto es importante predecir la cantidad de inundaciones en un intervalo de tiempo” (traducción libre). HS fue aplicado en [49], al igual que varias técnicas [48], para la calibración de tres parámetros del modelo más usado en este tipo de problemas, conocido como el modelo de Muskingum. El objetivo de este modelo es minimizar la diferencia entre las inundaciones reales y las inundaciones simuladas por el computador. Los resultados mostraron que HS fueron eficientes y se acercaron al óptimo global encontrado por otra técnica, conocida como BFGS [48].

En la vida moderna, industrial y urbana, la conservación de los ecosistemas y sus especies es muy importante. Con el fin de lograrlo, diferentes técnicas de optimización cuantitativa han sido desarrolladas y utilizadas para el problema de selección de sitios de reserva natural. En el campo de la optimización, 5 clases de problemas de selección de reserva han sido identificados: problema de recuperación de especies (Species Set Covering Problem, SSCP), problema de recuperación máxima de especies (Maximal Covering Species Problem, MCSP), problema de la máxima representación de múltiples especies (Maximal Multiple-Representation Species Problem, MMRSP), problema de la restricción de oportunidad de preservación (chance constrained covering problem) y problema de la preservación esperada (expected covering problem) [48, 50]. De los cinco problemas de optimización propuestos, MCSP ha sido el modelo más usado

comúnmente. MCSP maximiza el número de especies conservadas, mientras limita el número de reservas operadas [48, 50, 51]. HS fue modificado para adaptarse a varias funcionalidades específicas del problema como lo son: selección escasa (sparse selection), selección de la primera gran reserva (big-reserve-first selection) y selección de la primera diversidad (diversity-first selection). Aunque HS no alcanzó el optimo global para el problema planteado con el modelo MCSP [48], obtuvo muchos resultados cercanos a este. Este aspecto es importante pues la identificación de soluciones óptimas alternativas es valiosa para la planeación y la toma de decisiones, dado el caso de la posible ausencia (no disponibilidad) de alguna reserva [48].

Las aguas subterráneas son un importante recurso hidráulico en todo el mundo [52]. Por tal motivo, el desarrollo de estrategias sostenibles de planificación y gestión óptimas son necesarias para operar los recursos del agua subterránea. Modelos de simulación matemáticos son usados generalmente para predecir la futura respuesta de los sistemas de aguas subterráneas para diferentes flujos y condiciones de transporte de masa. Estos modelos se basan en la solución de ecuaciones diferenciales parciales que requieren algunos parámetros del modelo de distribución espacial hidrogeológicos. Dichos parámetros suelen ser desconocidos, debido a la complejidad de este tipo de sistemas. Esto hace que la identificación de estos sea una tarea importante para poder realizar las diferentes simulaciones en modelos de gestión utilizados para modelar las aguas subterráneas [52]. Modelos de simulación/optimización (S/O) han sido usados para la solución de muchos problemas de modelamiento de aguas subterráneas. Estos problemas se pueden clasificar en tres grupos principales: problemas de estimación de parámetros de aguas subterráneas, los problemas de gestión hidráulica de aguas subterráneas, y los problemas de gestión de calidad de aguas subterráneas. La estimación de parámetros se refiere a determinar algunos valores de los parámetros hidrogeológicos a través de enfoque de modelamiento inverso [52]. HS fue aplicado para solucionar el problema de la identificación de los parámetros [53] y los resultados fueron comparados con los obtenidos por la

aplicación de GA, mostrando que HS requiere menos simulaciones que GA y las soluciones son iguales o mejores que las obtenidas por GA [52].

Las pendientes de suelo son comunes en ingeniería civil y la evaluación de su estabilidad es de gran importancia en esta rama. Para el análisis de este tipo de problemas es comúnmente usado el método del límite de equilibrio. Utilizando el método del límite de equilibrio, “un valor de F, también llamado el factor de seguridad se puede estimar sin el conocimiento de las condiciones de estrés inicial y un problema puede ser definido y resuelto en un plazo relativamente corto” (traducción libre) [42]. El método del límite de equilibrio es un problema estáticamente indeterminado y diferentes hipótesis sobre la distribución de fuerzas internas son adoptadas por diferentes métodos de análisis. En la actualidad, el método propuesto por Morgenstern y Price (1965) se utiliza para dar el factor de seguridad para la superficie de deslizamiento especificada [10]. El factor mínimo de seguridad de una pendiente y su correspondiente superficie de fallo crítico son fundamentales para el diseño adecuado de las medidas de estabilización del deslizamiento. HS fue empleado para localizar la superficie de fallo crítico en el análisis de estabilidad de pendientes [54]. Los resultados muestran que HS es eficiente y eficaz para la minimización de los factores de seguridad para varios problemas difíciles y el método de generación de superficies fallidas de prueba puede ser importante en el proceso de minimización [54].

4.5 Aplicaciones en Ingeniería Mecánica

Las aplicaciones de HS en ingeniería mecánica cubren el diseño de intercambiadores de calor, diseño de redes de intercambiadores, diseño satelital, el diseño de amarres de plataformas marinas, entre otros.

Los intercambiadores de calor de carcasas y tubos (Shell and Tube Heat Exchangers - STHXs) son generalmente usados en procesos industriales, debido a su relativa sencilla fabricación y su adaptabilidad a diferentes condiciones de operación. “El diseño de STHXs, incluyendo el diseño termodinámico y dinámico

del fluido y la estimación de costos representa un proceso complejo que contiene un conjunto integrado de normas de diseño y el conocimiento empírico de diversos campos" (traducción libre) [55]. En [55] se explora el uso del HS y el análisis de sensibilidad global (GSA) para la optimización del diseño STHXs desde el punto de vista económico. Para reducir el tamaño del problema de optimización, los parámetros geométricos que tienen el menor efecto en el costo total de STHXs se identifican con la GSA y HS se aplica para optimizar los parámetros geométricos influyentes. Comparando los resultados de HS con los obtenidos por un algoritmo genético (GA) se obtiene que HS converge a la solución óptima con mayor precisión [55].

Para el diseño de redes de intercambiadores de calor, en [56] se presenta una nueva metodología hibrida para la síntesis de redes de intercambiadores de calor de costo optimo (heat Exchange networks - HENs). Dicho problema se resuelve en dos niveles: El nivel superior genera la estructura del HENs usando HS. En el nivel inferior para evaluar el mínimo costo de cada estructura se utiliza una combinación de HS y SQP. Basado en el costo obtenido para cada estructura, HS clasifica los HENs y produce nuevas estructuras hasta que el algoritmo converge a una solución óptima [56]. Para propósitos de validación, se examinaron tres problemas de referencia y también un problema del mundo real de tamaño industrial. Los resultados de este estudio muestran que el nuevo enfoque es capaz de encontrar redes más económicas que las generadas por otros métodos.

Un diseño satelital requiere de la optimización de múltiples objetivos, tales como rendimiento, confiabilidad y peso. Para considerar estos objetivos se debe considerar una optimización multi-objetivo. En este caso HS se ha usado para considerar la conductancia térmica y la masa del tubo de calor. El método consta de dos pasos: en la primera etapa cada función es optimizada por aparte, después se minimiza la función multi-objetivo, que es la suma del error individual entre el valor de la función actual y el valor óptimo. El estudio demostró que los resultados

al aplicar HS encuentra una mejor solución que la tradicional basada en cálculo de BFGS [57].

El diseño de amarres de plataformas marinas requiere una cantidad relativamente importante de ciclos de diseño, ya que una solución deseada deberá cumplir restricciones de diseño complejo y ser económicamente competitiva. La complejidad de estas restricciones de diseño en el puerto puede ser consecuencia de acoplamiento entre el movimiento de la plataforma y el puerto (sistema de bandas), restricción de desplazamiento máximo del sistema de bandas, varios números de parámetros de diseño que definen los componentes del sistema del anclaje y la singularidad de las condiciones ambientales dependen del sitio, incluyendo la profundidad del agua , condición actual del viento y la ola, estado del fondo marino, entre otras [10]. El proceso de diseño se torna más complejo si se solicita el costo óptimo del diseño. El diseño de amarres busca encontrar una rigidez adecuada que sea la suficientemente rígida y suave a la vez para satisfacer principalmente 2 restricciones de diseño: desplazamiento máximo horizontal requerido y la reducción de fuerzas extremas que actúan sobre la plataforma causadas por las interacciones entre el medio ambiente y las respuestas de la plataforma. Muchos sistemas han sido implementados para modelar este tipo de problemas, entre los cuales se destaca el sistema FPSO (Floating, Production, Storage and Offloading system). HS fue aplicado a este problema con el fin de determinar la longitud y el diámetro de cada componente del amarre [58]. En este diseño, sólo se aplicaron tres restricciones de diseño: la plataforma de desplazamiento máximo, el Factor de seguridad (FS) para el caso de tensión de tope intacta, y la no elevación de la cadena inferior. La función objetivo es el costo total del sistema de amarre. Un total de 2.000 iteraciones se realizaron para encontrar diseños óptimos en el puerto. HS fue usado para crear una herramienta de optimización de diseño y se propuso una herramienta de análisis de frecuencia de dominio global para minimizar el costo de los sistemas de amarres. Los resultados obtenidos muestran que la herramienta de

optimización de amarres basadas en HS tienen potencial para encontrar rápidamente el costo optimo de sistemas [10].

4.6 Aplicaciones en Medicina

Las aplicaciones de HS en medicina cubren la determinación de la función de las moléculas de ARN, el diseño de audífonos que optimicen el tiempo de la batería, la física medica, entre otros.

La determinación de la función de moléculas de ARN se basa en gran medida en su estructura secundaria. Los actuales métodos físicos para la determinación de la estructura del ARN consumen demasiado tiempo y son costosos. “Esto ha hecho que los métodos de predicción computacional de la estructura sean una mejor alternativa para la solución de este tipo de problemas” (traducción libre) [59]. Varios algoritmos se han utilizado para la predicción de la estructura del ARN, incluyendo programación dinámica y algoritmos meta heurísticos. En [59] se propone el algoritmo HSRNAFold para encontrar la estructura secundaria del ARN con la mínima energía disponible y la similitud de la estructura nativa. HSRNAFold es comparado con otras técnicas de programación dinámica: RNAfold y Mfold como punto de referencia. Los resultados muestran que HSRNAFold es comparable a la programación dinámica en la búsqueda de las mínimas energías disponibles en todas las secuencias de prueba de ARN. El método propuesto es eficiente y prometedor en la predicción de la estructura secundaria del RNA basada en la mínima energía disponible [59].

La aplicación de algoritmos de clasificación de sonido incorporados en los audífonos es una tarea difícil de llevar a cabo. Los audífonos tienen que trabajar con una frecuencia de reloj muy baja para reducir al mínimo el consumo de energía y así maximizar la vida útil de la batería [60]. Esto requiere la reducción de la carga computacional, mientras se mantiene una baja probabilidad de error. Dado que el proceso de extracción de características es una de las tareas que más tiempo consume, la selección de un número reducido de características

apropiadas es esencial, lo que requiere bajo coste computacional sin degradar la operación de este. HS permite una efectiva búsqueda de soluciones adecuadas a este problema fuertemente restringido. Al comenzar con un conjunto inicial de 74 diferentes características descriptivas de sonido, una serie de experimentos se llevaron a cabo para probar la eficacia del método propuesto. Los resultados del algoritmo HS son comparados con los alcanzados por otros métodos ampliamente utilizados [60]. HS puede reducir el número de combinaciones de las características a ser evaluadas en el proceso de selección de características de un sistema de clasificación de sonido. Los resultados muestran que HS es una elección viable y muy prometedora en este tipo de problemas [60].

La física médica es una rama de la física que se refiere a la aplicación de radiación para uso terapéutico y de diagnóstico en medicina. En la física medica terapéutica, la radiación ionizante se utiliza para tratar a los pacientes afectados por cáncer. El objetivo principal de la radioterapia es entregar una gran cantidad de dosis de radiación a las células cancerosas sin afectar los órganos circundantes. El proceso comienza con la planificación de la radioterapia, lo que requiere la optimización de la colocación de radioisótopos o la intensidad del haz de radiación. Anteriormente, algoritmos de optimización tales como algoritmos genéticos (GA) y recocido simulado (SA), han sido ampliamente usados [61]. Sin embargo, HS ha demostrado que es un algoritmo de optimización superior basado en los resultados en otros campos científicos. Por lo tanto, en [61] se investigó la optimización de la alta tasa de dosis de la braquiterapia de próstata con HS. Los resultados mostraron que el algoritmo es significativamente más rápido que el GA y que la rápida planificación permite mejorar el cuidado de paciente en física médica.

4.7 Aplicaciones en Economía

La economía es la ciencia social en la cual el objetivo principal es establecer la relación entre los recursos y la asignación de estos al hombre. Diversos problemas han sido identificados en esta ciencia, la mayoría como resultado a las cuestiones

sociales a las que se enfrenta un individuo en la sociedad, o un grupo de estos. Es el caso del problema de la suma de radios.

La suma de radios hace parte de la programación fraccional, la cual tiene como objetivo la optimización de uno o varios radios de diversas funciones. Los problemas de suma de radios tienen numerosas aplicaciones en economía e ingeniería. Son considerados muy difíciles, porque son funciones no convexas y multimodales. Entre las aplicaciones de los problemas de suma de radios se encuentran: la contratación gubernamental, la ciencia de transporte, las finanzas, la economía, ingeniería, etc. Además, en su parte investigativa, estos problemas presentan grandes desafíos teóricos y computacionales.

El algoritmo HS en [30], tuvo un importante aporte en la solución de este tipo de problemas. Los parámetros HMCR, PAR y BW son de gran eficiencia para mejorar la convergencia del HS a las soluciones óptimas. En este caso, la diferencia clave entre el HS y el PHS (Proposed Harmony Search, Búsqueda Armónica Propuesta) se encuentra en la manera de ajustar el BW, en la cual, se utiliza la diferencia central finita para aproximar la derivada. Los resultados obtenidos optimizando la función Himmelblau, muestran que el PHS obtuvo un mejor rendimiento frente al HS inicial, en cuanto al número de iteraciones. Al aplicar el PHS en la solución de diversos problemas de suma de radios, obtuvieron mejores resultados en el valor objetivo frente a otros métodos.

4.8 Aplicaciones en Transporte

HS ha sido exitoso en infinidad de problemas de transporte, entre ellos el enrutamiento de vehículos [27, 62]. El enrutamiento es uno de los diversos problemas que se presentan en el campo de la optimización, al cual se le han aplicado diversas técnicas para su solución, como es el caso de GA y diversos métodos que se han programado con este. El algoritmo HS fue modificado y adaptado en sus parámetros, para los cuales se tuvo en cuenta las restricciones típicas en este tipo de problemas: costo fijo por bus, costo de ruta por tiempo de

movimiento, penalidad de costo por violación de la capacidad del bus y penalidad de costos por violación de tiempo. El HS demostró ser más eficiente a la hora de minimizar el costo total del proceso de transporte de un bus escolar que el algoritmo GA teniendo en cuenta el numero de iteraciones para alcanzar el optimo global, el costo promedio de múltiples ejecuciones y el tiempo de computación. En resumen, puede decirse que el HS tiene potencial para ser aplicado en el campo de la ingeniería de tráfico.

5 TENDENCIAS

La optimización se ha convertido en un campo importante en el cual, el desarrollo de nuevos algoritmos meta heurísticos que utilizan novedosas técnicas ha sido un punto fuerte y se ha convertido últimamente en el objetivo a seguir para muchos algoritmos evolutivos. La inclusión de nuevas técnicas, como la simulación de procesos naturales y la hibridación de las mismas con técnicas utilizadas anteriormente, ha permitido corregir debilidades en los algoritmos e identificar características a mejorar para muchos de ellos.

A raíz de este fenómeno, se han tenido en cuenta algunos problemas de optimización que se han vuelto un factor estándar para realizar análisis comparativos entre los diferentes métodos y algoritmos de optimización propuestos. El objetivo de este campo de investigación es identificar las características y el campo de aplicación de un nuevo algoritmo frente a otros desarrollados anteriormente, buscando aislar los diferentes factores que hacen fuerte y débil a dicho proceso o técnica utilizada en él en comparación a otros.

Diversos algoritmos han nacido desde que el HS fue desarrollado y probado con éxito en 10 funciones que representan problemas clásicos de optimización (Sphere, Schwefel, Step, Rosenbrock, Rotated hyper-ellipsoid, Generalized Schwefel, Rastrigin, Ackley, Griewank, Six-Hump Camel-back) [7], lo cual ha permitido mejorar el rendimiento en ciertas características importantes para el

algoritmo. Entre estas se tiene el ruido, la dimensionalidad del problema, el ajuste de parámetros y el tipo de función (continua, discreta).

GBHS, IHS y NGHS son los desarrollos científicos más destacados que han solucionado algunas de las deficiencias de la versión original de la HS, mejorando el rendimiento y las propiedades de exploración y explotación del algoritmo. Entre los diversos desarrollos que involucran al algoritmo HS se encuentran: HS + GA [63], Harmony-Tabu (HS+TS) [64], HS + DE [65], HS + PSO + Ant [66], HS + SA [67], HS + Chaos [68], HS + ANN [69], HSSA (HS + SQP) [70], HS + Solver [71], HS + Google Map [72], HS + Simplex [73], HS + Taguchi [74], entre otros.

En este sentido, se espera que en el futuro cercano se realicen nuevos proceso de hibridación del HS o sus variaciones con otras meta heurísticas para resolver problemas en diversas áreas del desarrollo humano. También se espera que HS sea usado en entornos híper heurísticos para la solución de diversos problemas discretos y continuos. Finalmente, la búsqueda de reglas más específicas para definir los valores de los parámetros del algoritmo HS de acuerdo a necesidades especificas del espacio de solución y las restricciones del problema.

6 CONCLUSIONES

El algoritmo de búsqueda armónica (HS) ha demostrado ser una poderosa herramienta de optimización, la cual no necesita de cálculos matemáticos complejos para obtener soluciones optimas a un problema determinado.

En los últimos años, HS fue aplicado a infinidad de problemas, demostrando su eficiencia frente a otros algoritmos meta heurísticos y otras técnicas matemáticas de optimización. El desarrollo continuo de mejoras al algoritmo (GBHS, IHS y NGHS, entre otras) y las diversas aplicaciones a nuevos tipos de problemas (economía, ciencias de la computación, ingeniería eléctrica, entre otros), indican que HS es una buena elección si lo que se busca es una poderosa herramienta de programación que obtenga un alto desempeño en la búsqueda de soluciones, un

bajo consumo de tiempo, energía y costos computacionales. Su éxito en las diversas aplicaciones en ingeniería civil, ingeniería mecánica y la medicina, hacen de HS un algoritmo clave para la optimización general que permite adaptar su estructura y parámetros dependiendo del campo de aplicación.

7 REFERENCIAS

1. Geem, Z.W., *Music-Inspired Harmony Search Algorithm: Theory and Applications.* 2009: p. 206.
2. Yang, X.-S., *Harmony Search as a Metaheuristic Algorithm*, in *Music-Inspired Harmony Search Algorithm*. 2009, Springer Berlin / Heidelberg. p. 1-14.
3. Geem, Z.W., *Optimal Design of Water Distribution Networks Using Harmony Search.* 2009: p. 112.
4. Geem, Z.W., *Optimal cost design of water distribution networks using harmony search.* Engineering Optimization, 2006. **38**.
5. Geem, Z.W., *Harmony search algorithms for structural design optimization*, in *Studies in computational intelligence, v.239.* 2009, Springer Berlin Heidelberg: Berlin, Heidelberg. p. 228.
6. Geem, Z.W., *Harmony search algorithm for solving Sudoku*, in *Proceedings of the 11th international conference, KES 2007 and XVII Italian workshop on neural networks conference on Knowledge-based intelligent information and engineering systems: Part I*. 2007, Springer-Verlag: Vietri sul Mare, Italy.
7. Omran, M.G.H. and M. Mahdavi, *Global-best harmony search.* Applied Mathematics and Computation, 2008. **198**(2): p. 643-656.
8. Mahdavi, M., M. Fesanghary, and E. Damangir, *An improved harmony search algorithm for solving optimization problems.* Applied Mathematics and Computation, 2007. **188**(2): p. 1567-1579.
9. Geem, Z.W., *Novel derivative of harmony search algorithm for discrete design variables.* Applied Mathematics and Computation, 2008. **199**(1): p. 223-230.
10. Geem, Z.W., et al., *Recent Advances in Harmony Search.* Advances in Evolutionary Algorithms, 2008: p. 16.

11. Geem, Z., *State-of-the-Art in the Structure of Harmony Search Algorithm*, in *Recent Advances In Harmony Search Algorithm*. 2010, Springer Berlin / Heidelberg. p. 1-10.
12. Geem, Z., J. Kim, and G.V. Loganathan, *A New Heuristic Optimization Algorithm: Harmony Search.* SIMULATION, 2001. **76**(2): p. 60-68.
13. Lee, K. and Z. Geem, *A new meta-heuristic algorithm for continuous engineering optimization: harmony search theory and practice.* Computer Methods in Applied Mechanics and Engineering, 2005. **194**(36-38): p. 3902-3933.
14. Bäck, T., *Evolutionary Algorithms in Theory and Practice.* 1995: p. 328.
15. Kirkpatrick, S., *Optimization by simulated annealing: Quantitative studies.* Journal of Statistical Physics, 1984. **34**(5-6): p. 12.
16. Enrique Alba, R.M., *Tabu Search* 2006. **36**: p. 13.
17. Glover, F., et al., *The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances*, in *Handbook of Metaheuristics*. 2003, Springer New York. p. 250-285.
18. Vázquez, K., *Ant Colony Optimization.* Genetic Programming and Evolvable Machines, 2005. **6**(4): p. 459-460.
19. Poli, R., J. Kennedy, and T. Blackwell, *Particle swarm optimization.* Swarm Intelligence, 2007. **1**(1): p. 33-57.
20. Mira, J., J. Álvarez, and X.-S. Yang, *Engineering Optimizations via Nature-Inspired Virtual Bee Algorithms*, in *Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach*. 2005, Springer Berlin / Heidelberg. p. 317-323.
21. Yang, X.-S., *Nature-Inspired Metaheuristic Algorithms.* 2008: p. 128.
22. Watanabe, O., T. Zeugmann, and X.-S. Yang, *Firefly Algorithms for Multimodal Optimization*, in *Stochastic Algorithms: Foundations and Applications*. 2009, Springer Berlin / Heidelberg. p. 169-178.
23. Forsati, R. and M. Mahdavi, *Web Text Mining Using Harmony Search*, in *Recent Advances In Harmony Search Algorithm*. 2010, Springer Berlin / Heidelberg. p. 51-64.

24. Fourie, J., S. Mills, and R. Green, *Visual Tracking Using Harmony Search*, in *Recent Advances In Harmony Search Algorithm*. 2010, Springer Berlin / Heidelberg. p. 37-50.
25. Mahdavi, M. and H. Abolhassani, *Harmony K-means algorithm for document clustering.* Data Mining and Knowledge Discovery, 2009. **18**(3): p. 370-391.
26. Al-Betar, M.A., A.T. Khader, and T.A. Gani, *A harmony search for university course timetabling.* 2008: p. 12.
27. Mahdavi, M., *Solving NP-Complete Problems by Harmony Search*, in *Music-Inspired Harmony Search Algorithm*. 2009, Springer Berlin / Heidelberg. p. 53-70.
28. Zou, D., et al., *A novel global harmony search algorithm for reliability problems.* Computers & Industrial Engineering, 2009. **58**(2): p. 307-316.
29. Geem, Z.W., *Recent Advances in Harmony Search Algorithm.* 2010: p. 176.
30. Jaberipour, M. and E. Khorram, *Solving the sum-of-ratios problems by a harmony search algorithm.* Journal of Computational and Applied Mathematics, 2010. **234**(3): p. 733-742.
31. Geem, Z.W. and J.-Y. Choi, *Music Composition Using Harmony Search Algorithm.* Applications of Evolutionary Computing, 2007. **4448/2007**: p. 7.
32. Mahdavi, M., et al., *Novel meta-heuristic algorithms for clustering web documents.* Applied Mathematics and Computation, 2008. **201**(1-2): p. 441-451.
33. Forsati, R., et al. *Hybridization of K-Means and Harmony Search Methods for Web Page Clustering*. in *Web Intelligence and Intelligent Agent Technology, 2008. WI-IAT '08. IEEE/WIC/ACM International Conference on*. 2008.
34. Cobos, C., et al. *Web document clustering based on Global-Best Harmony Search, K-means, Frequent Term Sets and Bayesian Information Criterion*. in *IEEE Congress on Evolutionary Computation (IEEE CEC)*. 2010. Barcelona, Spain: IEEE.
35. Tangpattanakul, P., A. Meesomboon, and P. Artrit, *Optimal Trajectory of Robot Manipulator Using Harmony Search Algorithms*, in *Recent Advances In Harmony Search Algorithm*. 2010, Springer Berlin / Heidelberg. p. 23-36.

36. Xu, H., et al., *Harmony Search Optimization Algorithm: Application to a Reconfigurable Mobile Robot Prototype*, in *Recent Advances In Harmony Search Algorithm*. 2010, Springer Berlin / Heidelberg. p. 11-22.
37. Forsati, R., A.T. Haghighat, and M. Mahdavi, *Harmony search based algorithms for bandwidth-delay-constrained least-cost multicast routing.* Computer Communications, 2008. **31**(10): p. 2505-2519.
38. Fesanghary, M., *An Introduction to the Hybrid HS-SQP Method and Its Applications*, in *Recent Advances In Harmony Search Algorithm*. 2010, Springer Berlin / Heidelberg. p. 99-109.
39. Fesanghary, M., *Harmony Search Applications in Mechanical, Chemical and Electrical Engineering*, in *Music-Inspired Harmony Search Algorithm*. 2009, Springer Berlin / Heidelberg. p. 71-86.
40. Panigrahi, B., et al., *Population Variance Harmony Search Algorithm to Solve Optimal Power Flow with Non-Smooth Cost Function*, in *Recent Advances In Harmony Search Algorithm*. 2010, Springer Berlin / Heidelberg. p. 65-75.
41. Dong, H., et al. *Improved harmony search for detection with photon density wave*. in *International Symposium on Photoelectronic Detection and Imaging 2007: Related Technologies and Applications*. 2007. Beijing, China: SPIE.
42. Majidi, B., et al. *Harmonic optimization in multi-level inverters using harmony search algorithm*. in *Power and Energy Conference, 2008. PECon 2008. IEEE 2nd International*. 2008.
43. Rong, Z. and L. Hanzo, *Iterative Multiuser Detection and Channel Decoding for DS-CDMA Using Harmony Search.* Signal Processing Letters, IEEE, 2009. **16**(10): p. 917-920.
44. Saka, M.P., *Optimum design of steel sway frames to BS5950 using harmony search algorithm.* Journal of Constructional Steel Research, 2009. **65**(1): p. 36-43.
45. Geem, Z. and S. Degertekin, *Optimum Design of Steel Frames via Harmony Search Algorithm*, in *Harmony Search Algorithms for Structural Design Optimization*. 2009, Springer Berlin / Heidelberg. p. 51-78.

46. Saka, M. and F. Erdal, *Harmony search based algorithm for the optimum design of grillage systems to LRFD-AISC.* Structural and Multidisciplinary Optimization, 2009. **38**(1): p. 25-41.
47. Rossman, L.A., *EPANET- Users Manual.* 1994: p. 7.
48. Geem, Z., C.-L. Tseng, and J. Williams, *Harmony Search Algorithms for Water and Environmental Systems*, in *Music-Inspired Harmony Search Algorithm*. 2009, Springer Berlin / Heidelberg. p. 113-127.
49. Kim, J.H., Z.W. Geem, and E.S. Kim, *Parameter Estimation Of The Nonlinear Muskingum Model Using Harmony Search1.* JAWRA Journal of the American Water Resources Association, 2001. **37**(5): p. 1131-1138.
50. Geem, Z.W. and J.C. Williams, *Ecological optimization using harmony search*, in *Proceedings of the American Conference on Applied Mathematics*. 2008, World Scientific and Engineering Academy and Society (WSEAS): Cambridge, Massachusetts.
51. Geem, Z.W., Williams, J.C., *Harmony search and ecological optimization.* International Journal of Energy and Enviroment 1, 2007: p. 150 - 154.
52. Ayvaz, M., *Identification of Groundwater Parameter Structure Using Harmony Search Algorithm*, in *Music-Inspired Harmony Search Algorithm*. 2009, Springer Berlin / Heidelberg. p. 129-140.
53. Ayvaz, M.T., *Simultaneous determination of aquifer parameters and zone structures with fuzzy c-means clustering and meta-heuristic harmony search algorithm.* Advances in Water Resources, 2007. **30**(11): p. 2326-2338.
54. Cheng, Y.M., et al., *An improved harmony search minimization algorithm using different slip surface generation methods for slope stability analysis.* Engineering Optimization, 2008. **40**(2): p. 95 - 115.
55. Fesanghary, M., E. Damangir, and I. Soleimani, *Design optimization of shell and tube heat exchangers using global sensitivity analysis and harmony search algorithm.* Applied Thermal Engineering, 2009. **29**(5-6): p. 1026-1031.
56. Khorasany, R.M. and M. Fesanghary, *A novel approach for synthesis of cost-optimal heat exchanger networks.* Computers & Chemical Engineering, 2009. **33**(8): p. 1363-1370.

57. Geem, Z.W. and H. Hwangbo, *Application of Harmony Search to Multi-Objective Optimization for Satellite Heat Pipe Design.* 2006.
58. Sam Ryu, A.S.D., Caspar N. Heyl, Zong Woo Geem, *Mooring Cost Optimization Via Harmony Search.* Proceedings of OMAE07 - 26thh International Conference on Offshore Mechanics and Arctic Engineering, 2007: p. 8.
59. Mohsen, A., A. Khader, and D. Ramachandram, *An Optimization Algorithm Based on Harmony Search for RNA Secondary Structure Prediction*, in *Recent Advances In Harmony Search Algorithm*. 2010, Springer Berlin / Heidelberg. p. 163-174.
60. Alexandre, E., L. Cuadra, and R. Gil-Pita, *Sound Classification in Hearing Aids by the Harmony Search Algorithm*, in *Music-Inspired Harmony Search Algorithm*. 2009, Springer Berlin / Heidelberg. p. 173-188.
61. Panchal, A., *Harmony Search in Therapeutic Medical Physics*, in *Music-Inspired Harmony Search Algorithm*. 2009, Springer Berlin / Heidelberg. p. 189-203.
62. Geem, Z.W., *Application of Harmony Search to Vehicle Routing.* American Journal of Applied Sciences, 2005: p. 6.
63. Farhad, N., K. Ahamad Tajudin, and A.-B. Mohammed Azmi, *Adaptive genetic algorithm using harmony search*, in *Proceedings of the 12th annual conference on Genetic and evolutionary computation*. 2010, ACM: Portland, Oregon, USA.
64. Das, V.V., et al., *Music-Inspired Optimization Algorithm: Harmony-Tabu for Document Retrieval Using Relevance Feedback*, in *Information Processing and Management*. 2010, Springer Berlin Heidelberg. p. 385-387.
65. Liao, T.W., *Two hybrid differential evolution algorithms for engineering design optimization.* Applied Soft Computing, 2010. **10**(4): p. 1188-1199.
66. Kaveh, A. and S. Talatahari, *Particle swarm optimizer, ant colony strategy and harmony search scheme hybridized for optimization of truss structures.* Computers & Structures, 2009. **87**(5-6): p. 267-283.

67. Tan, Y., et al., *Design and Simulation of Simulated Annealing Algorithm with Harmony Search*, in *Advances in Swarm Intelligence*. 2010, Springer Berlin / Heidelberg. p. 454-460.
68. Santos Coelho, L.d. and D.L. de Andrade Bernert, *An improved harmony search algorithm for synchronization of discrete-time chaotic systems.* Chaos, Solitons & Fractals, 2009. **41**(5): p. 2526-2532.
69. Lee, J.-H. and Y.-S. Yoon, *Modified Harmony Search Algorithm and Neural Networks for Concrete Mix Proportion Design.* Journal of Computing in Civil Engineering, 2009. **23**(1): p. 57-61.
70. Fesanghary, M.M., M. Minary-Jolandan, M. Alizadeh, Y., *Hybridizing harmony search algorithm with sequential quadratic programming for engineering optimization problems.* Computer Methods in Applied Mechanics and Engineering, 2008. **197**(33-40): p. 3080-3091.
71. Ayvaz, M.T., et al., *Hybridizing the harmony search algorithm with a spreadsheet 'Solver' for solving continuous engineering optimization problems.* Engineering Optimization, 2009. **41**(12): p. 1119 - 1144.
72. Geem, Z.W. and W.E. Roper, *Various continuous harmony search algorithms for web-based hydrologic parameter optimisation.* International Journal of Mathematical Modelling and Numerical Optimisation, 2010. **1**: p. 14.
73. Woo Seok Jang, H.I.K., Byung Hee Lee, *Hybrid Simplex-Harmony search method for optimization problems.* Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on 2008: p. 8.
74. Yildiz, A.R., *Hybrid Taguchi-Harmony Search Algorithm for Solving Engineering Optimization Problems.* International Journal of Industrial Engineering - Theory, Applications and Practice, 2008. **15**.

Anexo B: Artículo “GHS+LEM: Global-best Harmony Search using Learnable Evolution Models” (GHS+LEM: Global-best Harmony Search usando Modelos Evolutivos que Aprenden)

Available online at www.sciencedirect.com
ScienceDirect
Applied Mathematics and Computation
APPLIED MATHEMATICS AND COMPUTATION
www.elsevier.com/locate/amc

GHS+LEM: Global-best Harmony Search usando modelos evolutivos que aprenden

Carlos Cobos [a], Dario Estupiñán [a], José Pérez [a]

[a]*University of Cauca (GTI), Sector Tulcán Office 422 FIET, Popayán, Colombia*

Abstract

Este artículo presenta un nuevo algoritmo de optimización denominado GHS+LEM, el cual se basa en el algoritmo Global-best Harmony Search (GHS) y técnicas de Learnable Evolution Models (LEM) para mejorar la convergencia y exactitud del algoritmo. El desempeño del algoritmo es evaluado con diez funciones de optimización comúnmente usadas por la comunidad de optimización. Además, los resultados obtenidos son comparados contra el algoritmo de búsqueda armónica original, la búsqueda armónica mejorada y la mejor búsqueda armónica global. La evaluación muestra que el algoritmo propuesto (GHS+LEM) mejora la precisión de los resultados obtenidos con relación a las otras propuestas, presentando mejores resultados en la mayoría de situaciones, pero más específicamente en problemas con alta dimensionalidad, donde ofrece una convergencia más rápida con un número reducido de iteraciones.

Keywords: Harmony search; Meta-heuristics; Evolutionary algorithms; Optimization, Learnable evolution models

1. Introducción

Las meta-heurísticas se definen como estrategias de alto nivel que guían otras heurísticas en la búsqueda de soluciones factibles y son generalmente usadas en problemas para los cuales no es posible obtener una solución óptima mediante métodos matemáticos complejos [1, 2]. En el marco de las meta heurísticas se encuentran los algoritmos de búsqueda armónica (Harmony Search), los cuales se basan en la observación del proceso de improvisación musical [3, 4]. HS ha sido aplicado exitosamente en múltiples problemas de optimización [5-26] y ha sido objeto de diversas variaciones con otras técnicas de optimización, entre los cuales se destacan el algoritmo de la búsqueda armónica mejorada (IHS) [27] y la mejor búsqueda armónica global (GHS) [28].

En el 2000, se presentan los modelos evolutivos que aprenden (Learnable Evolution Model, LEM) como una nueva técnica de optimización. En el método darwiniano de computación evolutiva, las poblaciones evolucionan basadas en procesos de selección, combinación y mutación. En LEM adicionalmente se emplean técnicas de aprendizaje de máquina para generar nuevas poblaciones. Al usar el modo de aprendizaje de máquina, LEM puede determinar cuáles individuos de una población (o un conjunto de individuos de poblaciones antiguas) son superiores a otros en la realización de ciertas tareas. Estas razones, expresadas como hipótesis inductivas, se usan para la generación de nuevas poblaciones. Luego, cuando el algoritmo se ejecuta en modo de evolución darwiniana usa operaciones aleatorias o semi-aleatorias para la generación de nuevos individuos (usando técnicas de combinación y/o mutación tradicionales) [29].

Este artículo propone una nueva versión de GHS denominada GHS+LEM, en la cual se hace uso de los conceptos de LEM, con el fin de mejorar la precisión del algoritmo GHS original. LEM fue adaptado para que funcione de una forma simple, busque zonas promisorias donde se encuentre el óptimo global y funcione con variables discretas y continuas. El desempeño de GHS+LEM se compara con diez funciones de optimización ampliamente conocidas y las cuales son usadas originalmente en las pruebas del algoritmo GHS [28], posteriormente se hace un estudio del desempeño del algoritmo propuesto modificando el número de dimensiones de los problemas propuestos y la variación de los parámetros del algoritmo.

El artículo está organizado de la siguiente forma: La secciones 2, 3 y 4 hacen un resumen de los algoritmos HS, IHS y GHS. La sección 5 presenta el nuevo algoritmo, GHS+LEM. La sección 6 presenta los resultados del algoritmo frente a las funciones de optimización usadas y el impacto de la variación de las dimensiones y los parámetros del algoritmo. Finalmente la sección 7 presenta las conclusiones y el trabajo futuro que el grupo de investigación espera desarrollar.

2. Harmony Search

El algoritmo de la búsqueda armónica, propuesto por Zong Woo Geem y Kang Seo Lee [1] en el 2001, es un algoritmo meta heurístico, es decir, un algoritmo de propósito general que consiste en procedimientos iterativos que guían una heurística, combinando de forma inteligente distintos conceptos para explorar y explotar adecuadamente el espacio de búsqueda [1]. HS simula el proceso de improvisación musical, en el cual los músicos buscan producir una armonía agradable determinada por el estándar estético auditivo [4]. La Tabla 35 presenta las acciones realizadas por un músico cuando está improvisando y su correspondiente representación (formalizadas en [1]) en el algoritmo HS.

Tabla 35
Relación de los componentes del algoritmo HS con las acciones de improvisación musical

Acciones	Componentes
Toca alguna melodía aprendida anteriormente.	Uso de la memoria armónica
Toca algo parecido a la melodía anteriormente mencionada, ajustándola poco a poco al tono deseado.	Ajuste de tono
Compone una nueva melodía basándose en sus conocimientos musicales a partir de notas seleccionadas aleatoriamente.	Aleatoriedad

"En la improvisación musical, cada músico toca una nota dentro de un posible rango, de tal manera que forman un vector de armonías. Si el conjunto de notas tocadas por los músicos son consideradas una buena armonía, esta es guardada en la memoria de cada músico, incrementando la posibilidad de hacer una buena armonía la próxima vez. Del mismo modo en el proceso de optimización en ingeniería, cada variable de decisión inicialmente toma valores aleatorios dentro del rango posible, formando un vector solución. Si dicho conjunto de valores que conforman el vector, son una buena solución, esta es almacenada en la memoria de cada variable, aumentando la posibilidad de encontrar mejores soluciones en la siguiente iteración" (traducción libre) [30].

Los pasos que indican el funcionamiento del algoritmo HS y su descripción son descritos a continuación:

a) Inicializar los parámetros del problema y de HS

El problema de optimización se define como $Minimizar\ (o\ maximizar)\ f(x)\ tal\ que\ LB_i < x_i < UB_i$. Donde $f(x)$ es la función objetivo, *x* es una solución candidata que consta de N variables de decisión (x_i), y LB_i y UB_i son el límite de decisión más bajo y el más alto de cada variable, respectivamente. Los parámetros de HS se especifican en este paso y son: El tamaño de la memoria armónica (HMS), la tasa de consideración de la memoria armónica (HMCR), la tasa de ajuste del tono (PAR), el ancho de banda de ajuste del tono (bw) y el número de improvisaciones (NI).

b) Inicializar la memoria armónica

La memoria armónica inicial es generada desde una distribución uniforme en los rangos $[LB_i, UB_i]$, donde $1 \leq i \leq N$. Esto se realiza de la siguiente manera: $x_i^j = LB_i + r \times (UB_i - LB_i), donde\ i = 1,2,\dots,HMS\ y\ r \sim U(0,1)$. La variable r hace referencia a un número aleatorio y $U(0,1)$ a la función que genera el número aleatorio uniforme entre 0 y 1.

c) Improvisar una nueva armonía

El proceso de generación de una nueva armonía es llamado *improvisación*. El nuevo vector de armonías, $x' = (x'_1, x'_2, \dots, x'_N)$, se genera utilizando las siguientes reglas: consideración de la memoria, ajuste del tono y selección aleatoria. El valor BW es un ancho de banda arbitrario de la distancia para variables continuas y r es un número aleatorio uniforme entre 0 y 1 [30]. Este procedimiento se muestra a continuación en la Fig. 1.

foreach $i \in [1, \mathrm{N}]$ **do**
  if $U(0,1) < HMCR$ **then** */*memory consideration*/*

begin
$x_i' = x_i^j$, where $j \sim U(1, \ldots, HMS)$
if $U(0,1) \leq PAR$ **then** */*pitch adjustement*/*
$x_i' = x_i' \pm r \times bw$
end_if
end
else */*random selection*/*
$x_i' = LB_i + r \times (UB_i - LB_i)$
end_if
done

Fig. 1. Improvisación de una nueva armonía en HS

d) Actualizar la memoria armónica

La nueva armonía generado, $x' = (x'_1, x'_2, \ldots, x'_N)$, reemplaza la peor armonía almacenada en la memoria armónica (HM) sólo si su *fitness* (o valor de aptitud de la nueva armonía, medido en términos de la función objetivo) es mejor que el de la peor armonía.

e) Verificar el criterio de parada

Termina la ejecución del algoritmo cuando el número máximo de improvisaciones (NI) se alcanza, de lo contrario los pasos 2.3 y 2.4 se repiten.

En general HS es poco sensible a los parámetros [3, 4], por esto, el algoritmo no requiere de una afinación exhaustiva de los mismos para obtener buenas soluciones. A pesar de lo anterior, es preciso destacar que los parámetros HMCR y PAR ayudan al método en la búsqueda de mejores soluciones globales y locales, respectivamente. PAR y bw tienen un profundo efecto en el desempeño del algoritmo y es por esto que el ajuste de estos dos parámetros es muy importante.

3. Improved Harmony Search

La búsqueda armónica mejorada (*Improved Harmony Search*) es un algoritmo armónico propuesto en el año 2007 por Mahdavi et al [27], emplea un método para la generación de nuevos vectores solución basado en el ajuste dinámico de los parámetros PAR (tasa de ajuste del tono) y bw (ancho de banda de la ajuste del tono), logrando con esto mejorar la precisión y la velocidad de convergencia. En esta variante sólo se modifica el paso que crea una nueva armonía. PAR y bw cambian dinámicamente con el número de generaciones y se calculan con las siguientes fórmulas.

$$PAR(gn) = PAR_{min} + \frac{(PAR_{max} - PAR_{min})}{NI} \times gn$$

Donde,

$PAR(gn)$	Tasa de ajuste del tono para cada generación.
PAR_{min}	Tasa mínima de ajuste del tono.
PAR_{max}	Tasa máxima de ajuste del tono.
NI	Número de generaciones de vectores solución.
gn	Número de generación.

y

$$bw(gn) = bw_{max}\, e^{(c \times gn)},\ donde\ c = \frac{\ln\left(\frac{bw_{min}}{bw_{max}}\right)}{NI}$$

Donde,

$bw(gn)$	Ancho de banda de la distancia para cada generación
bw_{min}	Ancho de banda de la distancia mínimo
bw_{max}	Ancho de banda de la distancia máximo

El parámetro PAR crece linealmente con el número de generaciones, mientras que *bw* decrece exponencialmente. Con este cambio en los parámetros, IHS logra mejorar el rendimiento de HS, ya que encuentra mejores soluciones tanto a nivel global

como local. “Una desventaja importante de IHS es que el usuario tiene que especificar los valores mínimo y máximo de bw, ya que estos valores son difíciles de definir y dependen del problema” [28].

4. Global-best Harmony Search

Global-best Harmony Search (GHS) es un algoritmo de optimización estocástica propuesto en el año 2008 por Mahamed G.H. Omran y Mehrdad Mahdavi [28], el cual hibrida la búsqueda armónica original con el concepto de inteligencia de enjambre propuesto en PSO (Particle Swarm Optimization) [28], donde un enjambre de individuos (llamados partículas) vuela a través del espacio de búsqueda. Cada partícula representa un candidato a la solución del problema de optimización. La posición de una partícula está influenciada por la mejor posición visitada por sí misma (su propia experiencia) y la posición de las mejores partículas en el enjambre (la experiencia de enjambre). GHS modifica el paso de ajuste del tono en HS de modo que la nueva armonía puede imitar la mejor armonía en la memoria armónica. Esto permite a GHS trabajar eficientemente en problemas continuos y discretos. En general GHS es mejor que IHS y HS cuando se aplica a problemas de alta dimensionalidad y cuando hay presencia de ruido [28].

GHS tiene exactamente los mismos pasos que IHS y HS con la salvedad de la modificación del paso 3 que corresponde a la improvisación de una nueva armonía, el cual se modifica conforme a la Fig. 2.

foreach $i \in [1, \mathrm{N}]$ **do**
  if $U(0,1) < HMCR$ **then** */*memory consideration*/*
    begin
      $x_i' = x_i^j$, where $j \sim U(1, \ldots, HMS)$
      if $U(0,1) \leq PAR(t)$ **then** */*pitch adjustment with PSO*/*
        $x_i' = x_k^{best}$, where $best$ is the index of the best harmony in HM and $k \sim U(1, N)$
      end_if
    end
  else */*random selection*/*
    $x_i' = LB_i + r \times (UB_i - LB_i)$
  end_if
done

Fig. **2**. Improvisación en el algoritmo de la mejor búsqueda armónica global (GHS)

5. Algoritmo propuesto: GHS+LEM

Inspirados en el concepto de Learnable Evolution Model (LEM) propuesto por Michalsky [29], en este artículo se propone una nueva variación del algoritmo GHS. En LEM se emplean técnicas de aprendizaje de máquina para generar nuevas poblaciones adicionalmente al método darwiniano, aplicado en la computación evolutiva, el cual se basa en mutación, combinación y selección natural. Este método puede determinar cuáles individuos de una población (o un conjunto de individuos de poblaciones anteriores) son mejores a otros en la realización de ciertas tareas. Estas razones, expresadas como hipótesis inductivas, se usan para la generación de nuevas poblaciones. Luego, cuando el algoritmo se ejecuta en modo de evolución darwiniana, usa operaciones aleatorias o semi-aleatorias para la generación de nuevos individuos (usando técnicas de mutación y/o recombinación tradicionales). El proceso LEM puede resumirse en los siguientes pasos:

1. Generar una población
2. Ejecutar el modo de aprendizaje de Máquina.
3. Ejecutar el modo de aprendizaje Darwiniano.
4. Alternar de modo hasta que el criterio de finalización sea alcanzado.

El modo de aprendizaje de Máquina (ítem 2 del anterior listado) a su vez consta de los siguientes pasos [29].

A. Derive extrema: seleccionar dos grupos de la población actual, uno de Alto rendimiento (H-Group) y otro de bajo rendimiento (L-Group), basados en la evaluación de la función objetivo.
B. Crear una hipótesis: aplicar un método de aprendizaje de máquina para crear una descripción del H-group que lo diferencie del L-group, opcionalmente se puede hacer la consideración de poblaciones pasadas.

C. Generar la nueva población: Generar nuevos individuos a través de las reglas aprendidas de la descripción de los dos grupos.
D. Ir al paso A y repetir hasta que se alcance el criterio de terminación del proceso de aprendizaje de máquina.

El proceso de aprendizaje de máquina en el nuevo algoritmo emplea un variación del algoritmo PRISM propuesto por Cendrowska [31, 32]. PRISM toma como entrada un conjunto de entrenamiento dado como un archivo de conjuntos ordenados de valores de atributos, cada uno determinado por una clasificación. La información sobre los atributos y clasificaciones (nombre, número de valores posibles, lista de posibles valores, etc.) es la entrada de un archivo por separado al inicio del programa, y los resultados se emiten como reglas individuales para cada una de las clasificaciones que figuran en términos de los atributos descritos.

La aproximación que se utiliza del algoritmo PRISM en la propuesta, está destinada a imitar las decisiones simples que manejan los algoritmos de la familia armónica, y puede trabajar tanto con variables continuas como discretas. Para tal fin se cuenta con un conjunto de reglas conjuntivas ($P \leftarrow R_1 \wedge R_2 \wedge ... \wedge R_n$), que delimitan las regiones alrededor de las cuales hay una mayor posibilidad de encontrar un mejor valor para cada x_i (por ejemplo, $LV_{x_i} \leq x_i \leq HV_{x_i}$, donde *LV* y *HV* son los límites inferior y superior de las reglas para el valor x_i). Con la conjunción de las reglas (R) para cada dimensión se limita el espacio de búsqueda a las regiones con más posibilidades de generar un óptimo global. El algoritmo de inferencia de reglas se ejecuta por primera vez inmediatamente después de creada la memoria armónica inicial. Los pasos del algoritmo de inferencia de reglas, se resumen a continuación:

A. Se escoge de la memoria armónica actual la población de alto rendimiento y la población de bajo rendimiento siguiendo la siguiente fórmula

$$Hgroup = P_{actual}(1, \dots, i),$$
$$Lgroup = P_{actual}(HMS - i, \dots, HMS),$$

donde i = HLGS, HMS es el tamaño de la memoria armónica (HM) y HLGS es la rata de generación de reglas

B. Se genera una matriz para cada clase *i* perteneciente a cada grupo. En esta matriz se almacena el valor del atributo en esa posición y el *fitness* correspondiente de la siguiente forma, si es *Hgroup* se asigna 1 y en caso contrario se asigna 0, después se ordena con respecto al valor de su atributo de menor a mayor.
C. Se calcula la probabilidad de la ocurrencia de cada atributo.
D. Se selecciona el atributo con la mayor probabilidad de ocurrencia y se agrega a la lista de reglas *P*
E. Se repiten los pasos B, C y D hasta que se evalúen todos los atributos de cada grupo, la regla resultante es de tipo $P \rightarrow Q$, donde *P* es una conjunción de las reglas que tienen la mayor probabilidad para cada atributo y Q corresponde a la clase 1 (grupo de alto rendimiento).

La nueva propuesta se denomina Mejor búsqueda armónica global usando modelos evolutivos que aprenden (GHS+LEM) y los pasos del algoritmo se presentan a continuación:

a) Inicializar los parámetros del problema y los parámetros de GHS+LEM
Este paso es similar al propuesto en GHS y agrega tres parámetros que se explican más adelante, a saber: la tasa de actualización de las reglas (RRU), el tamaño de los grupos de alto y bajo rendimiento (HLGS) y la tasa de consideración de reglas (RCR).

b) Inicializar la memoria armónica
Se ejecuta el proceso de inicialización propuesto en HS sin ningún cambio.

c) Ejecutar el proceso de inferencia de reglas por primera vez
Se ejecuta por primera vez el proceso de inferencia de reglas con base en la memoria armónica inicial y el tamaño de los grupos de alto y bajo rendimiento (HLGS). Este parámetro (HLGS) indica el tamaño de los grupos de alto desempeño y los grupos de bajo desempeño, este valor debe ser $\leq \lfloor HMS/2 \rfloor$.

d) Improvisar la nueva armonía
En este paso se introduce el uso de las reglas generadas en el paso anterior para la definición de los valores de la dimensión del nuevo improviso (ver Fig. 3). Lo anterior se ejecuta basado en el parámetro denominada tasa de consideración de reglas (*RCR*). Este parámetro (RCR) decide en que porcentaje de las veces se utilizará las reglas, de lo contrario se ejecutará el método tradicional de generación aleatoria basado en el espacio general de búsqueda (original en HS).

for each $i \in [1, \mathrm{N}]$ **do**

if $U(0,1) < HMCR$ **then** */*memory consideration*/*
  begin
    $x_i' = x_i^j$, where $j \sim U(1, \ldots, HMS)$
    if $U(0,1) \leq PAR(t)$ **then** */*pitch adjustment with PSO*/*
      $x'_i = x_k^{best}$, where *best* is the index of the best harmony in HM and $k \sim U(1, N)$
    end_if
  end
else
  if $U(0,1) < RCR$ **then** /*rule consideration rate*/
    $x'_i = U(LB_i^{best}, UB_i^{best})$, where *best* is the best set of rules for x_n
  else /* random selection */
    $x_i' = LB_i + r \times (UB_i - LB_i)$
  end_if
end_if
done

Fig. 3. Improvisación en el algoritmo GHS+LEM

e) Actualizar la memoria armónica
La nueva armonía generado, $x' = (x'_1, x'_2, \ldots, x'_N)$ reemplaza la peor armonía almacenada en la memoria armónica si el *fitness* (o valor de aptitud de la nueva armonía, medido en términos de la función objetivo) es mejor que el de la peor armonía.

f) Verificar el criterio de actualización de reglas
Se realiza a través del parámetro RRU, el cual especifica en que porcentaje de las veces se deben actualizar las reglas. Si un número aleatorio generado uniformemente entre 0 y 1 es menor que el valor de RRU, se ejecuta el proceso de inferencia de reglas nuevamente (ver Fig. 4).

if $U(0,1) < RRU$ **then** */*rule update*/*
  Ejecutar el proceso de inferencia de reglas
end_if

Fig. 4. Proceso de actualización de reglas en GHS+LEM

g) Verificar el criterio de parada:
Termina la ejecución del algoritmo cuando el número máximo de improvisaciones (NI) se alcanza, de lo contrario los pasos 5.4, 5.5 y 5.6 se repiten. Este paso no sufre cambios con respecto al original.

6. Resultados Experimentales

En esta sección se muestra el desempeño del algoritmo GHS+LEM frente a la búsqueda armónica original (HS), la búsqueda armónica mejorada (IHS) y la Mejor búsqueda armónica global (GHS). Los parámetros usados para la ejecución de los algoritmos en cada experimento se presentan antes de cada tabla de resultados, pero en la Tabla 36 se muestra la configuración de parámetros generalmente usada.

Tabla 36
Configuraciones generales de las pruebas

Variable	HS	IHS	GHS	GHSLEM
HMS	5	5	5	5
HCMR	0.9	0.9	0.9	0.9
PAR	0.3	N.A.	N.A.	N.A.
PAR_{min}	N.A.	0.01	0.01	0.01
PAR_{max}	N.A.	0.99	0.99	0.99
bw	0.01	N.A.	N.A.	N.A.
bw_{min}	N.A.	0.0001	N.A.	N.A.
bw_{max}	N.A.	$1/(20\ x\ (UB - LB))$	N.A.	N.A.
HLGS	N.A.	N.A.	N.A.	$\lfloor HMS/2 \rfloor$
RCR	N.A.	N.A.	N.A.	0.9

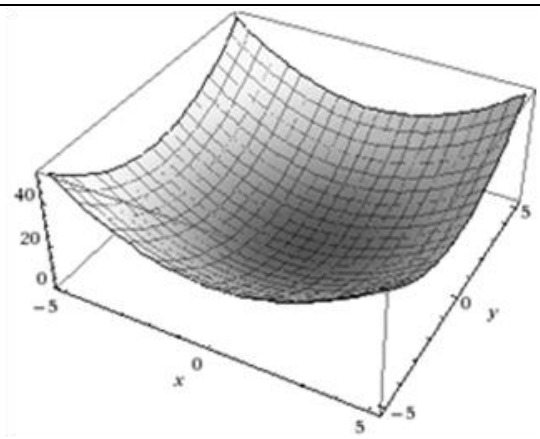
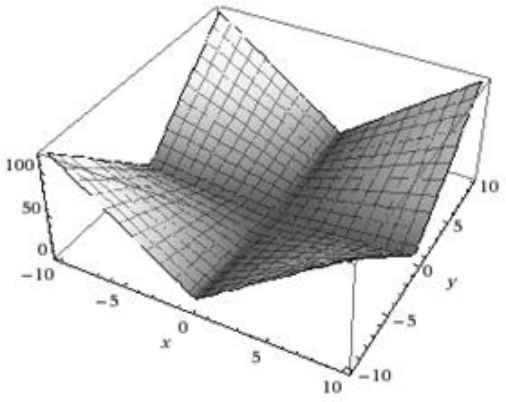
RRU	N.A.	N.A.	N.A.	0.2

Todas las funciones, excepto la Six-Hump Camel-Back que es bi-dimensional, fueron implementadas para 2, 3, 5, 10, 15, 20, 30, 40 y 50 dimensiones. Para cada una de las dimensiones se utilizaron 50, 500, 5.000 y 50.000 iteraciones. En cada caso se especifica cuántas dimensiones se utilizaron en esa prueba específica y el número de iteraciones. La memoria armónica inicial se genera de forma aleatoria dentro de los rangos específicos de cada función.

a) Funciones de prueba usadas en la evaluación

Para la comparativa se utilizaron las funciones que aparecen en la Tabla 37 de funciones unimodales y en la Tabla 38 de funciones multimodales. Estas se basan en las funciones propuestas en el artículo de GHS [28] que proporciona un balance adecuado entre funciones unimodales y multimodales. Para cada una de las funciones se busca encontrar el mínimo global definido como:
Dado $f = \Re^{N_d} \rightarrow \Re$
Encontrar $x^* \in \Re^{N_d}$ tal que $f(x^*) \leq f(x), \forall x \in \Re^{N_d}$, donde $N_d, es\ el\ número\ de\ dimensiones$

Tabla 37
Funciones Unimodales

A. Sphere (Primera función de De Jong´s)[33] $f(x) = \sum_{i=1}^{N_d} {x_i}^2$ Donde $x^* = 0$ y $f(x^*) = 0$ para $-5.12 \leq x_i \leq 5.12$	
B. Schwefel's Problem 2.22[34] $f(x) = \sum_{i=1}^{N_d} \lvert x_i \rvert + \prod_{i=1}^{N_d} \lvert x_i \rvert$ Donde $x^* = 0$ y $f(x^*) = 0$ para $-10 \leq x_i \leq 10$	

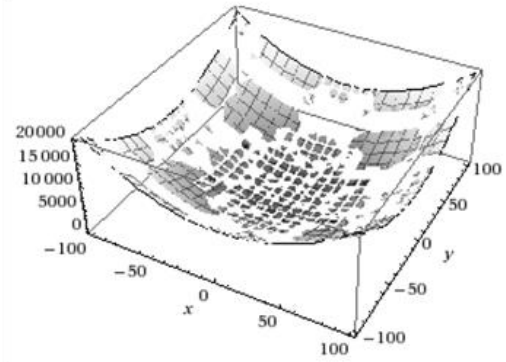
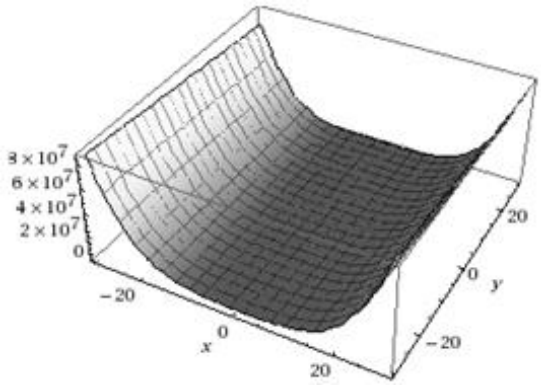
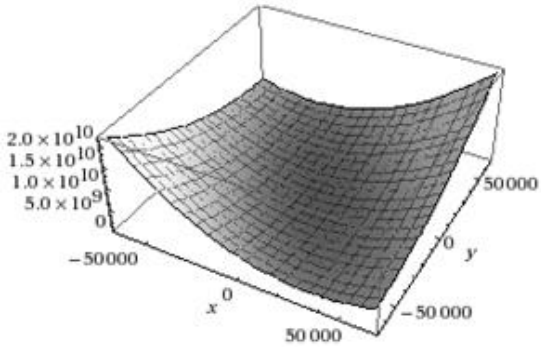
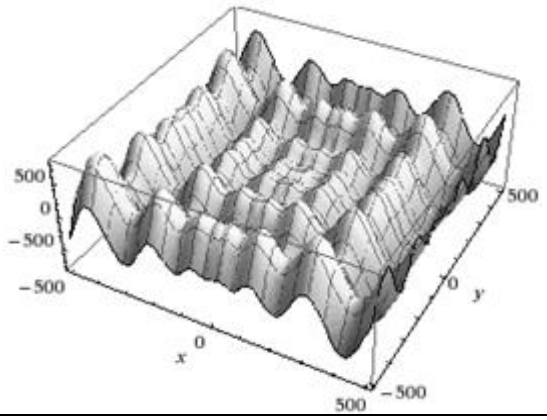
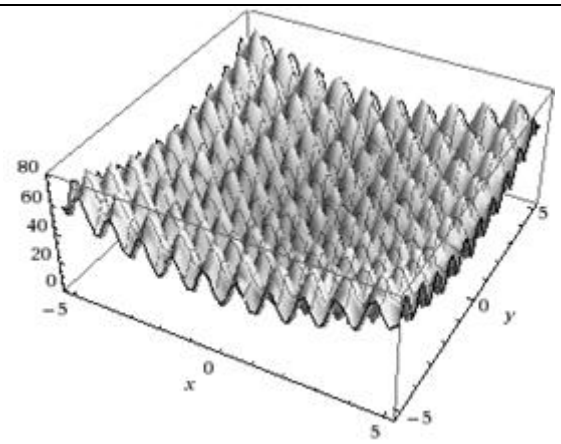
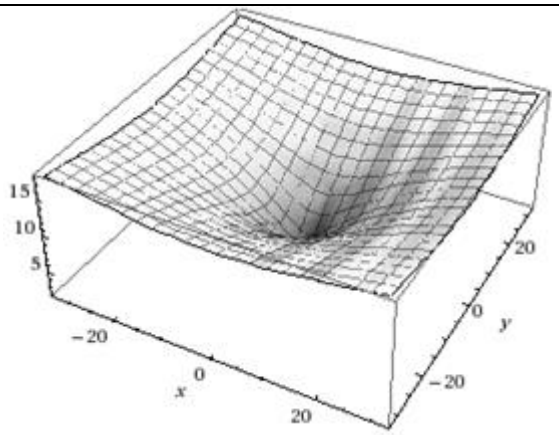
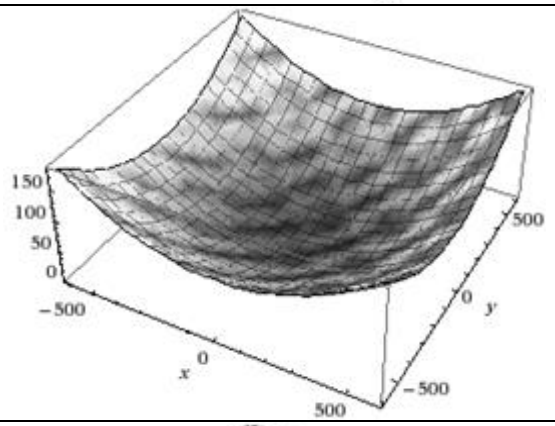
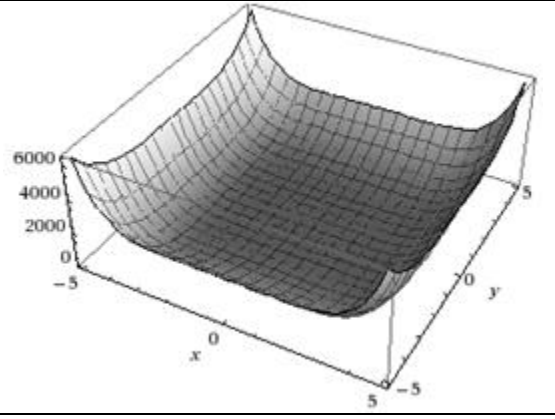
C. Step $$f(x) = \sum_{i=1}^{N_d} (\lfloor x_i + 0.5 \rfloor)^2$$ Donde $x^* = 0$ y $f(x^*) = 0$ para $-100 \leq x_i \leq 100$	
D. Rosenbrock [33] $$f(x) = \sum_{i=1}^{N_d-1} \left[100(x_{i+1} - x_i^2)^2 + (1 - x_{i)}{}^2\right]$$ Donde $x^* = (1,1,\ldots,1)$ y $f(x^*) = 0$ para $-30 \leq x_i \leq 30$	
E. Rotated Hyper-ellipsoid [28] $$f(x) = \sum_{i=1}^{N_d} \left(\sum_{j=1}^{i} x_j \right)^2$$ Donde $x^* = 0$ y $f(x^*) = 0$ para $-65536 \leq x_i \leq 65536$	

Tabla 38
Funciones Multimodales

F. Generalized Schwefel 2.26[34] $$f(x) = -\sum_{i=1}^{N_d} (x_i sin(\sqrt{x_i}))$$ Donde $x^* = (420.9687, \ldots, 420.9687)$ y $f(x^*) = -12569.5$ para $-500 \leq x_i \leq 500$	

G. Rastrigin[33] $$f(x) = \sum_{i=1}^{N_d} ({x_i}^2 - 10cos(2\pi x_i) + 10N_d)$$ Donde $x^* = 0$ y $f(x^*) = 0$ para $-5.12 \leq x_i \leq 5.12$	
H. Ackley[33] $$f(x) = -20e^{\left(-0.2\sqrt{\frac{1}{30}\sum_{i=1}^{N_d} x_i^2}\right)} - e^{\left(\frac{1}{30}\sum_{i=1}^{N_d}\cos 2\pi x_i\right)} + 20 + e$$ Donde $x^* = 0$ y $f(x^*) = 0$ para $-32 \leq x_i \leq 32$	
I. Griewank[33] $$f(x) = \frac{1}{4000}\sum_{i=1}^{N_d} x_i^2 - \prod_{i=1}^{N_d}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$$ Donde $x^* = 0$ y $f(x^*) = 0$ para $-600 \leq x_i \leq 600$	
J. Six-Hump Camel-Back [28] Función de baja dimensionalidad con pocos mínimos locales. $$f(x) = 4x_1^2 + 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$$ Donde $x^* = (-0.08983, 0.7126), (-0.08983, 0.7126)$ y $f(x^*) = -1.0316285$ para $-5 \leq x_i \leq 5$	

b) Resultados de la comparativa

La Tabla 39 presenta los resultados de las pruebas comparativas aplicadas al algoritmo GHS+LEM con el objeto de medir su precisión contra los otros algoritmos de la familia armónica. El número de iteraciones para la prueba se definió en 50.000 y el número de dimensiones se estableció en 30 para todas las funciones excepto para Six-hump Camel-Back la cual está definida en dos dimensiones. Los algoritmos fueron ejecutados 100 veces para garantizar una desviación media confiable.

Tabla 39
Valor Medio y desviación estándar (**$\pm SD$**) de las pruebas de optimización (N_d = 30, NI=50.000).

		HMS	**IHS**	**GHS**	**GHS+LEM**
Unimodales					
Sphere	Media	0.000684005	0.017838978	4.0457E-05	**2.09647E-10**
	(±SD)	*9.67781E-05*	*0.00710319*	*7.29366E-05*	***3.60168E-10***
Schwefel Problem 2.22	Media	0.143656975	0.997096357	0.040860755	**5.70121E-05**
	(±SD)	*0.047911784*	*0.200329207*	*0.037067055*	***4.49754E-05***
Rosenbrock	Media	312.2431152	423.9427774	72.47196696	**15.77537882**
	(±SD)	*486.5124844*	*330.6943507*	*103.3253058*	***22.43722786***
Step	Media	11.56	11.22	**0**	**0**
	(±SD)	*4.608555943*	*3.945538332*	***0***	***0***
Rotated Hyper-Ellipsoid	Media	4419.904558	4238.371416	5427.433309	**686.306798**
	(±SD)	*1306.038774*	***1196.967797***	*6641.408049*	*2049.883544*
Multimodales					
Schwefel Problem 2.26	Media	-12545.01282	-12540.34846	-12569.46257	**-12569.48662**
	(±SD)	*9.274118296*	*10.54344883*	*0.03971048*	***3.40336E-11***
Rastrigin	Media	1.266797341	2.722732645	0.009457309	**3.81653E-08**
	(±SD)	*1.023021844*	*1.130249802*	*0.014012005*	***6.06791E-08***
Ackley	Media	0.981392208	1.584674315	0.024746761	**5.83147E-06**
	(±SD)	*0.485630315*	*0.331393069*	*0.026603311*	***4.86194E-06***
Griewank	Media	1.085396028	1.087082117	0.091022469	**2.1722E-11**
	(±SD)	*0.035098647*	*0.031926489*	*0.192952247*	***4.5967E-11***
Six-Hump Camel- Back	Media	-1.031600318	-1.031628428	-1.031568182	**-1.031628452**
	(±SD)	*3.48248E-05*	*5.53445E-09*	*8.34751E-05*	***1.45999E-09***

Los resultados de las pruebas aplicadas indican que GHS+LEM supera ampliamente la precisión obtenida por HS, IHS y GHS en todas las funciones de optimización realizadas. La desviación estándar de las pruebas también es menor para todas las funciones en las cuales se aplicó el algoritmo a excepción de Rotated Hyper-ellipsoid en la cual IHS es menor. Sin embargo aún en el peor caso, el resultado obtenido con GHS+LEM (2736.190342) supera el resultado del mejor obtenido por IHS (5435.339213).

La Tabla 40 presenta los resultados de las pruebas de escalabilidad a las que fue sometido el algoritmo GHS+LEM. El número de iteraciones se definió en 50.000, el número de dimensiones en 50 y el número de ejecuciones de cada algoritmo fue fijado en 100. No se incluyen los resultados de la función Six-Hump Camel-back los cuales se presentan en la Tabla 39. GHS+LEM mejora la precisión en cada una de las funciones de optimización usadas, probando ser mejor que HS, IHS y GHS en condiciones de alta dimensionalidad. Además, en funciones discontinuas como Step, GHS+LEM mantiene su resultado óptimo a diferencia de las demás propuestas, las cuales sufren en condiciones de mayor dimensionalidad.

Tabla 40
Valor Medio y desviación estándar (**$\pm SD$**) de las pruebas de optimización (N_d = 50, NI=50.000).

		HMS	**IHS**	**GHS**	**GHS+LEM**
Unimodales					
Sphere	Media	1.231713634	1.36689254	0.005550663	**2.58528E-08**
	(±SD)	*0.287760963*	*0.308892735*	*0.00776301*	***5.5786E-08***
Schwefel Problem 2.22	Media	9.594968369	10.03102019	0.411417885	**0.000441435**
	(±SD)	*1.080214642*	*1.361876185*	*0.397066313*	***0.000376589***

Rosenbrock	Media	28119.05942	27416.72832	357.7255365	**39.49848422**
	(±SD)	*10535.26342*	*9607.338888*	*726.1644056*	***59.9930843***
Step	Media	513.92	535.07	0.09	**0**
	(±SD)	*101.4288505*	*112.1655752*	*0.637149587*	***0***
Rotated Hyper-Ellipsoid	Media	28751.33713	28878.96444	59867.79886	**27492.62734**
	(±SD)	*5722.795639*	*6583.833253*	***21857.53114***	*30492.1086*
Multimodales					
Schwefel Problem 2.26	Media	-20065.84929	-20055.73906	-20944.1766	**-20949.14436**
	(±SD)	*183.3728874*	*187.2603852*	*8.953405915*	***3.8441E-09***
Rastrigin	Media	35.35722669	45.97323267	0.407654111	**4.13742E-06**
	(±SD)	*4.955395824*	*5.414365656*	*0.622184354*	***8.94894E-06***
Ackley	Media	5.259876609	5.382419569	0.324365569	**6.09717E-05**
	(±SD)	*0.384154298*	*0.38808497*	*0.444545366*	***5.45021E-05***
Griewank	Media	5.701261605	5.887341775	0.700857354	**5.35254E-09**
	(±SD)	*1.080435525*	*1.108359613*	*0.368310151*	***1.198E-08***
Six-Hump Camel- Back	Media	-1.031600318	-1.031628428	-1.031568182	**-1.031628452**
	(±SD)	*3.48248E-05*	*5.53445E-09*	*8.34751E-05*	***1.45999E-09***

c) Efectos de la modificación de los parámetros HCMR, HMS, PAR y RCR

Para estudiar la variación de los parámetros HCMR, HMS, PAR y RCR se usaron los valores especificados en la Tabla 36. El número de dimensiones es 30, el número de iteraciones es 50.000 y el número de ejecuciones para cada prueba es 30.

La Tabla 41 muestra los efectos de la variación del parámetro HCMR en el algoritmo propuesto. Se observa que la precisión del algoritmo mejora con un HCMR más alto. Un HCMR alto (≥ 0.9) favorece la convergencia. En funciones de baja dimensionalidad como Six-Hump Camel-back se requiere un menor valor de HCMR para aumentar su capacidad de exploración. Es decir, un HCMR más bajo (0.7 por ejemplo) favorece la búsqueda del óptimo en funciones en las cuales se necesita una mayor exploración.

Tabla 41
Valor Medio y desviación estándar (**$\pm SD$**) con variación de HCMR (N_d = 30, NI=50.000)

HCMR		**0.5**	**0.7**	**0.9**	**0.95**
Unimodales					
Sphere	Media	0.000197381	2.16632E-05	2.46402E-10	**3.33277E-11**
	(±SD)	*3.46488E-05*	*8.90181E-06*	*4.85473E-10*	***6.94329E-11***
Schwefel Problem 2.22	Media	0.046732628	0.008995544	4.78127E-05	**1.67511E-05**
	(±SD)	*0.00547272*	*0.003419191*	*3.78919E-05*	***1.41968E-05***
Rosenbrock	Media	23.02466444	**16.02993218**	26.47111615	32.26301449
	(±SD)	*32.03192831*	***14.24846561***	*32.66147183*	*108.4442821*
Step	Media	0	0	0	0
	(±SD)	***0***	***0***	***0***	***0***
Rotated Hyper-Ellipsoid	Media	2638.719935	**0.832511519**	380044196.6	1066841975
	(±SD)	*9217.946483*	***4.557631147***	*898797929.4*	*1890003832*
Multimodales					
Schwefel Problem 2.26	Media	-12569.48659	-12569.48662	**-12569.48662**	**-12569.48662**
	(±SD)	*5.24351E-06*	*1.07255E-06*	*4.07439E-11*	***1.62063E-11***
Rastrigin	Media	0.677007539	0.00421244	3.39398E-08	**4.2067E-09**
	(±SD)	*3.47513999*	*0.001785797*	*7.24178E-08*	***7.68495E-09***
Ackley	Media	0.010563105	0.003265899	8.99544E-06	**3.70719E-06**
	(±SD)	*0.001000305*	*0.000749123*	*9.99327E-06*	***3.20833E-06***
Griewank	Media	9.62872E-06	9.92445E-07	2.79283E-11	**2.95602E-12**

	(±SD)	*2.24672E-06*	*4.86263E-07*	*3.46684E-11*	***4.3954E-12***
Six-Hump Camel- Back	Media	-1.031628453	**-1.031628453**	-1.031628453	-1.031628448
	(±SD)	*3.15616E-10*	***2.0307E-10***	*1.36426E-09*	*1.88262E-08*

La Tabla 42 presenta los resultados de la variación del parámetro HMS en el algoritmo propuesto. Se observa que los mejores resultados para el algoritmo propuesto se encuentran ubicados entre los valores 5 y 10 (80%). Le siguen los valores mayores de 10 (20%). El algoritmo propuesto encuentra mejores resultados en tamaños de memoria armónica pequeños tal como se recomienda en el algoritmo HS original. La función Step, al ser discontinua, parece no ser afectada por este parámetro. Un trabajo futuro debe comprobar si al utilizar un histórico de las reglas se pueda mejorar la precisión del algoritmo aun utilizando un tamaño de memoria armónica menor.

Tabla 42
Valor Medio y desviación estándar (**±*SD***) con variación de HMS (N_d = 30, NI=50.000)

HMS		**5**	**10**	**20**	**50**
Unimodales					
Sphere	Media	1.09049E-08	**9.42334E-09**	4.56347E-08	6.10931E-08
	(±SD)	*1.97455E-08*	***1.21108E-08***	*8.98288E-08*	*1.00668E-07*
Schwefel Problem 2.22	Media	**0.00058248**	0.000613678	0.000721842	0.000857497
	(±SD)	***0.000568687***	*0.000648931*	*0.00074417*	*0.000758539*
Rosenbrock	Media	**31.0074059**	48.32958186	39.0007948	31.84389228
	(±SD)	*51.12983659*	*57.16531525*	*47.41503805*	***35.13859472***
Step	Media	**0**	**0**	**0**	**0**
	(±SD)	***0***	***0***	***0***	***0***
Rotated Hyper-Ellipsoid	Media	30170.55111	**23222.86045**	33483.26924	30271.62452
	(±SD)	*53524.86969*	***42045.26373***	*89766.40829*	*103520.6057*
Multimodales					
Schwefel Problem 2.26	Media	**-12569.48662**	-12569.48662	-12569.48662	-12569.48596
	(±SD)	***4.53706E-08***	*1.32036E-07*	*7.77143E-08*	*0.00362494*
Rastrigin	Media	**2.95253E-06**	5.09339E-06	2.9867E-06	1.61558E-05
	(±SD)	***3.63668E-06***	*1.01325E-05*	*4.88117E-06*	*2.98689E-05*
Ackley	Media	**7.03661E-05**	8.6753E-05	7.74663E-05	0.000163219
	(±SD)	***5.66978E-05***	*0.000139645*	*6.64779E-05*	*0.000179311*
Griewank	Media	0.030470641	0.010369382	**0.002829622**	0.021107642
	(±SD)	*0.157625199*	*0.037402693*	***0.014132011***	*0.050745481*
Six-Hump Camel- Back	Media	-1.031628349	-1.031628331	-1.031628381	**-1.031628382**
	(±SD)	*1.22241E-07*	*1.37004E-07*	***7.88387E-08***	*1.01315E-07*

La Tabla 43 se ocupa de los resultados de la variación del parámetro PAR en el algoritmo propuesto. En la propuesta original de GHS [28] el PAR se ajusta dinámicamente con respecto al número de iteraciones [27, 28]. El algoritmo HS propuesto por Geem [1] establece que el parámetro PAR debe ser fijo y se recomienda 0.3. En este grupo de pruebas se establece un valor PAR constante de 0.1, 0.3, 0.5, 0.7, 0.9 y uno dinámico, como en IHS. Se observa que los mejores desempeños del algoritmo propuesto se obtienen cuando el par es dinámico para todas las funciones de optimización. En funciones no continuas el parámetro PAR parece no afectar la precisión del algoritmo. Aun en a funciones cuya convergencia es lenta (Rotated Hyper-Ellipsoid) la mejor elección es un PAR dinámico. La diferencia entre los resultados de Schwefel Problem 2.26 para el PAR escogido y el PAR dinámico es de 0.003%, lo cual lo hace estadísticamente irrelevante.

Tabla 43
Valor Medio y desviación estándar (**±*SD***) con variación de PAR (N_d = 30, NI=50.000)

PAR		**0.1**	**0.3**	**0.5**	**0.7**	**0.9**	**Dinámico**
Unimodales							
Sphere	Media	3.16E-09	2.43E-09	1.06E-08	1.55E-08	2.05E-08	**2.10E-10**
	(±SD)	*7.21E-09*	*6.07E-09*	*2.73E-08*	*2.94E-08*	*3.23E-08*	***3.60E-10***

Schwefel Problem 2.22	Media	1.45E-04	2.19E-04	2.41E-04	4.29E-04	5.84E-04	**5.70E-05**
	(±SD)	*1.05E-04*	*1.94E-04*	*2.80E-04*	*3.92E-04*	*5.55E-04*	***4.50E-05***
Rosenbrock	Media	2.67E+01	4.97E+01	5.35E+01	5.82E+01	2.55E+01	**1.58E+01**
	(±SD)	*4.16E+01*	*7.67E+01*	*7.03E+01*	*6.29E+01*	*4.82E+01*	***2.24E+01***
Step	Media	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
	(±SD)	***0.0***	***0.0***	***0.0***	***0.0***	***0.0***	***0.0***
Rotated Hyper-Ellipsoid	Media	3.37E+03	4.14E+03	1.16E+04	4.02E+03	1.77E+04	**6.86E+02**
	(±SD)	*1.12E+04*	*7.65E+03*	*2.13E+04*	*1.08E+04*	*5.43E+04*	***2.05E+03***
Multimodales							
Schwefel Problem 2.26	Media	-1.26E+04	-1.26E+04	-1.26E+04	-1.26E+04	-1.26E+04	**-1.26E+04**
	(±SD)	*1.16E-09*	*1.08E-09*	*8.46E-10*	*5.13E-09*	*6.67E-09*	***3.40E-11***
Rastrigin	Media	5.75E-07	7.07E-07	1.23E-06	1.41E-06	2.71E-06	**3.82E-08**
	(±SD)	*8.04E-07*	*1.20E-06*	*2.06E-06*	*2.43E-06*	*5.04E-06*	***6.07E-08***
Ackley	Media	2.37E-05	2.87E-05	3.51E-05	5.17E-05	9.03E-05	**5.83E-06**
	(±SD)	*2.40E-05*	*2.43E-05*	*3.35E-05*	*5.68E-05*	*7.93E-05*	***4.86E-06***
Griewank	Media	4.79E-02	3.85E-10	9.14E-10	4.92E-09	3.46E-09	**2.17E-11**
	(±SD)	*1.62E-01*	*5.39E-10*	*1.80E-09*	*1.94E-08*	*5.65E-09*	***4.60E-11***
Six-Hump Camel-Back	Media	-1.03162744E+00	-1.03162754E+00	-1.03162813E+00	-1.03162814E+00	-1.03162830E+00	**-1.03162845E+00**
	(±SD)	*1.30410791E-06*	*1.75798819E-06*	*3.98521037E-07*	*4.35232361E-07*	*1.90219051E-07*	***1.45999395E-09***

Los resultados de la variación del parámetro RCR se muestran en la Tabla 44. El parámetro RCR determina en que porcentaje se utilizarán las reglas inferidas del algoritmo adaptado PRISM, en la generación de una nueva armonía. Se observa una tendencia clara a usar un factor de uso de reglas, RCR, grande ($0.7 \leq RCR \leq 1.0$). El parámetro recomendado para RCR por defecto en el algoritmo propuesto es 0.9, el cual muestra mayor nivel de efectividad. Queda abierta la posibilidad de un estudio con un factor RCR variable entre 0.7 y 1.0.

Tabla 44
Valor Medio y desviación estándar **($\pm SD$)** con variación de RCR (N_d = 30, NI=50.000)

LEMCR	**1.0**	**0.9**	**0.8**	**0.7**	**0.6**	**0.5**	**0.4**	**0.3**	**0.2**	**0.1**
Unimodales										
Sphere	**2.161E-10**	2.352E-10	2.453E-10	3.132E-10	2.729E-10	4.625E-10	1.399E-09	2.230E-09	4.838E-09	1.011E-08
	5.040E-10	*4.509E-10*	*4.745E-10*	*5.985E-10*	***4.207E-10***	*7.560E-10*	*2.929E-09*	*4.419E-09*	*9.146E-09*	*1.561E-08*
Schwefel Problem 2.22	**3.607E-05**	4.470E-05	5.970E-05	6.989E-05	7.995E-05	9.629E-05	1.086E-04	1.732E-04	2.777E-04	5.493E-04
	4.350E-05	***3.848E-05***	*7.632E-05*	*7.082E-05*	*8.713E-05*	*7.975E-05*	*1.053E-04*	*1.367E-04*	*3.002E-04*	*4.912E-04*
Rosenbrock	2.101E+01	1.861E+01	**1.476E+01**	3.032E+01	1.994E+01	2.617E+01	2.719E+01	7.098E+01	4.208E+01	6.856E+01
	3.121E+01	*3.042E+01*	***2.344E+01***	*3.910E+01*	*3.451E+01*	*3.375E+01*	*4.439E+01*	*1.682E+02*	*8.332E+01*	*1.333E+02*
Step	1.20	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
	5.94	***0.00***	***0.00***	***0.00***	***0.00***	***0.00***	***0.00***	***0.00***	***0.00***	***0.00***
Rotated Hyper-Ellipsoid	**1.514E+03**	1.858E+03	1.537E+03	2.253E+03	2.350E+03	3.560E+03	2.497E+03	3.740E+03	6.115E+03	6.009E+03
	3.680E+03	*4.963E+03*	***3.238E+03***	*5.492E+03*	*5.929E+03*	*7.180E+03*	*4.506E+03*	*6.954E+03*	*7.886E+03*	*6.796E+03*
Multimodales										
Schwefel Problem 2.26	-1.250E+04	**-1.257E+04**	**-1.257E+04**	**-1.257E+04**	**-1.257E+04**	-1.257E+04	-1.257E+04	-1.257E+04	-1.257E+04	-1.257E+04
	5.025E+02	***3.855E-11***	*5.829E-11*	*5.530E-11*	*6.038E-11*	*2.142E-10*	*7.002E-10*	*3.531E-10*	*5.424E-10*	*5.673E-04*

Rastrigin	1.194E+00	6.038E-08	5.753E-08	**5.499E-08**	8.028E-08	1.356E-07	2.565E-07	2.105E-07	7.543E-07	5.055E-06
	5.909E+00	*1.366E-07*	*1.132E-07*	***9.368E-08***	*1.141E-07*	*3.275E-07*	*3.396E-07*	*4.158E-07*	*1.073E-06*	*1.152E-05*
Ackley	1.430E-01	7.612E-06	**7.243E-06**	9.316E-06	1.307E-05	9.818E-06	1.325E-05	2.711E-05	3.759E-05	5.917E-05
	7.076E-01	***6.617E-06***	*6.937E-06*	*1.132E-05*	*1.028E-05*	*1.205E-05*	*1.479E-05*	*2.417E-05*	*3.438E-05*	*6.534E-05*
Griewank	2.272E-11	**1.208E-11**	1.400E-11	2.154E-11	8.018E-11	5.236E-11	1.672E-10	1.272E-02	2.786E-03	5.095E-02
	3.142E-11	***1.542E-11***	*2.240E-11*	*3.105E-11*	*2.327E-10*	*8.748E-11*	*4.402E-10*	*8.994E-02*	*1.957E-02*	*1.551E-01*
Six-Hump Camel- Back	-9.500E-01	**-1.032E+00**	-1.032E+00	-1.032E+00	-1.032E+00	-1.032E+00	-1.032E+00	-1.032E+00	-1.032E+00	-1.032E+00
	2.473E-01	***1.645E-09***	*2.202E-09*	*3.530E-09*	*3.106E-09*	*6.871E-09*	*1.025E-08*	*8.118E-09*	*4.938E-08*	*9.653E-08*

Se realizaron otras pruebas variando el número de iteraciones como se muestra en la Tabla 45. El número de iteraciones para esta prueba es de 5.000. Se observa que el algoritmo GHS+LEM supera a las demás propuestas cuando el número de iteraciones es bajo. El algoritmo propuesto conserva un nivel de aproximación a la solución óptima aceptable a pesar del bajo número de iteraciones propuesto. Las diferencias en la función Six-hump Camel-back no son estadísticamente significativas.

Tabla 45
Valor Medio y desviación estándar **($\pm SD$)** con variación de número de iteraciones (N_d = 30, NI=5.000)

		HMS	**IHS**	**GHS**	**GHS+LEM**
Unimodales					
Sphere	Media	1.391113338	1.492336778	0.008490508	**5.93279E-08**
	(±SD)	*0.442516349*	*0.429375922*	*0.015014868*	***9.80204E-08***
Schwefel Problem 2.22	Media	7.343708198	7.942415139	0.38849092	**0.000681506**
	(±SD)	*1.298620467*	*1.376467735*	*0.392227894*	***0.000587781***
Rosenbrock	Media	44801.42361	44335.30705	365.8780006	**35.91286497**
	(±SD)	*27087.38846*	*25827.36583*	*988.1136926*	***68.65861969***
Step	Media	577.17	599.34	3.25	**0**
	(±SD)	*178.0053373*	*181.6569761*	*8.62738768*	***0***
Rotated Hyper-Ellipsoid	Media	17744.96362	17590.00622	26959.75359	**11283.9062**
	(±SD)	***3863.897243***	*4062.189579*	*14265.59047*	*15864.62194*
Multimodales					
Schwefel Problem 2.26	Media	-11723.5473	-11734.68475	-12561.42156	**-12569.4597**
	(±SD)	*194.1725169*	*182.3846445*	*16.70991705*	***0.248116333***
Rastrigin	Media	29.46075697	36.89200558	0.881408659	**1.17383E-05**
	(±SD)	*5.461732931*	*6.233664508*	*1.594135534*	***1.89959E-05***
Ackley	Media	6.335464888	6.480145971	0.535233079	**0.000125684**
	(±SD)	*0.620693868*	*0.686288345*	*0.701961543*	***0.000103427***
Griewank	Media	6.277931588	6.370436585	0.795212512	**0.006467333**
	(±SD)	*1.494900647*	*1.816294084*	*0.350711873*	***0.054165186***
Six-Hump Camel- Back	Media	-1.03155243	**-1.031628431**	-1.026283458	-1.030628257
	(±SD)	*5.65151E-05*	***7.41828E-09***	*0.008075092*	*0.004327667*

Se compararon los resultados de GHS+LEM con 5.000 iteraciones con los resultados obtenidos por los demás métodos (HS, IHS y GHS) con 50.000 iteraciones (ver Tabla 46). Se observa que aún con un número de iteraciones bajo (5.000) el algoritmo propuesto mejora los resultados obtenidos en casi todas las funciones de optimización empleadas. En casos como Schwefel Problem 2.26 la diferencia no es estadísticamente significativa (0.000023%). Para funciones con convergencia

lenta, como Rotated Hyper-Ellipsoid, se hace necesario un mayor número de iteraciones para mejorar la precisión del algoritmo propuesto.

Tabla 46
Valor Medio y desviación estándar (**$\pm SD$**) comparación entre precisión con diferentes números de iteraciones (N_d = 30; GHS+LEM con NI=5.000; HS, IHS y GHS con NI=50.000)

		HMS	**IHS**	**GHS**	**GHS+LEM (5000 NI)**
Unimodales					
Sphere	Media	0.000684005	0.017838978	4.0457E-05	**5.93279E-08**
	(±SD)	*9.67781E-05*	*0.00710319*	*7.29366E-05*	***9.80204E-08***
Schwefel Problem 2.22	Media	0.143656975	0.997096357	0.040860755	**0.000681506**
	(±SD)	*0.047911784*	*0.200329207*	*0.037067055*	***0.000587781***
Rosenbrock	Media	312.2431152	423.9427774	72.47196696	**35.91286497**
	(±SD)	*486.5124844*	*330.6943507*	*103.3253058*	***68.65861969***
Step	Media	11.56	11.22	**0**	**0**
	(±SD)	*4.608555943*	*3.945538332*	***0***	***0***
Rotated Hyper-Ellipsoid	Media	4419.904558	**4238.371416**	5427.433309	11283.9062
	(±SD)	*1306.038774*	***1196.967797***	*6641.408049*	*15864.62194*
Multimodales					
Schwefel Problem 2.26	Media	-12545.01282	-12540.34846	**-12569.46257**	-12569.4597
	(±SD)	*9.274118296*	*10.54344883*	***0.03971048***	*0.248116333*
Rastrigin	Media	1.266797341	2.722732645	0.009457309	**1.17383E-05**
	(±SD)	*1.023021844*	*1.130249802*	*0.014012005*	***1.89959E-05***
Ackley	Media	0.981392208	1.584674315	0.024746761	**0.000125684**
	(±SD)	*0.485630315*	*0.331393069*	*0.026603311*	***0.000103427***
Griewank	Media	1.085396028	1.087082117	0.091022469	**0.006467333**
	(±SD)	*0.035098647*	***0.031926489***	*0.192952247*	*0.054165186*
Six-Hump Camel- Back	Media	-1.031600318	**-1.031628428**	-1.031568182	-1.030628257
	(±SD)	*3.48248E-05*	***5.53445E-09***	*8.34751E-05*	*0.004327667*

Además se realizaron pruebas para evaluar el desempeño del algoritmo en problemas de programación entera. Las pruebas implementadas fueron definidas en el artículo de GHS [28] y corresponden a cinco problemas definidos como F1, F2, F3, F4 y F5, los resultados se observan en la Tabla 47. En los mismos se aprecia que GHS+LEM mejora la precisión para la función F1 en alta dimensionalidad con respecto a las demás propuestas. Para todas las demás funciones implementadas se observa que GHS+LEM se desempeña de forma similar a los otros algoritmos, mejorando los resultados de GHS en F2, F3 y F5.

Tabla 47
Problemas de programación entera

		HS	**IHS**	**GHS**	**GHS+LEM**
F1 (N=5)	Media	**0**	**0**	**0**	**0**
	(±SD)	***0***	***0***	***0***	***0***
F1 (N=15)	Media	**0**	0.833333333	**0**	**0**
	(±SD)	***0***	0.461133037	***0***	***0***
F1	Media	6.266666667	11.26666667	0.433333333	**0**

(N=30)					
	(±SD)	1.387961376	1.964044619	0.504006933	***0***
F2	Media	**0**	**0**	0.3	**0**
	(±SD)	***0***	***0***	0.70221325	***0***
F3	Media	**0**	9	0.133333333	**0**
	(±SD)	***0***	16.29258346	0.434172485	***0***
F4	Media	**-7**	**-7**	-6.933333333	-6.933333333
	(±SD)	***0***	***0***	0.253708132	0.253708132
F5	Media	**-3880**	**-3880**	-3879.633333	**-3880**
	(±SD)	***0***	***0***	0.556053417	***0***

7. **Conclusiones**

Este artículo presenta una nueva versión del algoritmo GHS llamada GHS+LEM. El algoritmo propuesto utiliza técnicas de LEM para crear un conjunto de reglas que permita inferir nuevos candidatos en la población que no surjan solamente a partir de la exploración aleatoria. La modificación permite que el nuevo algoritmo se desempeñe eficientemente tanto en funciones discretas como continuas. Se sometió el algoritmo a diez funciones de optimización clásicas y en la mayoría mejora los resultados obtenidos contra los demás métodos (HS, IHS, GHS). Se concluye también a través de una prueba de escalabilidad que el algoritmo mantiene su precisión inclusive en altas dimensiones (≥ 30). Se investigaron los efectos de los parámetros HCMR, HMS, PAR y RCR en el desempeño del algoritmo propuesto, dando como resultado que en HCMR ≥ 0.9 el algoritmo generalmente mejora su eficacia, por otra parte, en cuanto al tamaño de la memoria armónica las pruebas muestran que el algoritmo propuesto generalmente se desempeña mejor cuando el tamaño está entre 5 y 10, lo cual es consecuente con las recomendaciones del algoritmo HS original. Se observa también que el valor de PAR obtiene mejores resultados cuando este es dinámico, como lo propuesto en IHS. Con respecto a la variación del parámetro RCR se observó que se tiene un mejor desempeño general del algoritmo cuando el proceso de aplicación de las reglas es ejecutado con una probabilidad comprendida entre 0.7 y 1. El algoritmo también demuestra mantener una precisión mayor que las otras propuestas incluso cuando el número de iteraciones es 10 veces menor al usado por las otras propuestas armónicas. Como trabajo futuro el grupo de investigación se propone estudiar el desempeño del algoritmo en problemas del mundo real; introducir modificaciones en el algoritmo de inferencia de reglas que permitan tener en cuenta históricos de la memoria armónica y modificar el proceso de generación de reglas para que se actualicen cada vez que se ejecute un cambio en la memoria armónica con el fin de evaluar el desempeño del algoritmo en estas nuevas condiciones.

9. **Referencias**

1. Geem, Z., J. Kim, and G.V. Loganathan, *A New Heuristic Optimization Algorithm: Harmony Search.* SIMULATION, 2001. **76**(2): p. 60-68.
2. Luke, S., *Essentials of Metaheuristics*, S. Luke, Editor. 2009, George Mason University.
3. Yang, X.-S., *Harmony Search as a Metaheuristic Algorithm*, in *Music-Inspired Harmony Search Algorithm*. 2009, Springer Berlin / Heidelberg. p. 1-14.
4. Geem, Z.W., *Music-Inspired Harmony Search Algorithm: Theory and Applications.* 2009: p. 206.
5. Geem, Z.W., et al., *Recent Advances in Harmony Search.* Advances in Evolutionary Algorithms, 2008: p. 16.
6. Tangpattanakul, P., A. Meesomboon, and P. Artrit, *Optimal Trajectory of Robot Manipulator Using Harmony Search Algorithms*, in *Recent Advances In Harmony Search Algorithm*. 2010, Springer Berlin / Heidelberg. p. 23-36.
7. Panigrahi, B., et al., *Population Variance Harmony Search Algorithm to Solve Optimal Power Flow with Non-Smooth Cost Function*, in *Recent Advances In Harmony Search Algorithm*. 2010, Springer Berlin / Heidelberg. p. 65-75.
8. Mohsen, A., A. Khader, and D. Ramachandram, *An Optimization Algorithm Based on Harmony Search for RNA Secondary Structure Prediction*, in *Recent Advances In Harmony Search Algorithm*. 2010, Springer Berlin / Heidelberg. p. 163-174.
9. Fourie, J., S. Mills, and R. Green, *Visual Tracking Using Harmony Search*, in *Recent Advances In Harmony Search Algorithm*. 2010, Springer Berlin / Heidelberg. p. 37-50.

10. Forsati, R. and M. Mahdavi, *Web Text Mining Using Harmony Search*, in *Recent Advances In Harmony Search Algorithm*. 2010, Springer Berlin / Heidelberg. p. 51-64.
11. dos Santos Coelho, L. and D. de A. Bernert, *A Harmony Search Approach Using Exponential Probability Distribution Applied to Fuzzy Logic Control Optimization*, in *Recent Advances In Harmony Search Algorithm*. 2010, Springer Berlin / Heidelberg. p. 77-88.
12. Ayvaz, M., *Solution of Groundwater Management Problems Using Harmony Search Algorithm*, in *Recent Advances In Harmony Search Algorithm*. 2010, Springer Berlin / Heidelberg. p. 111-122.
13. Geem, Z., *State-of-the-Art in the Structure of Harmony Search Algorithm*, in *Recent Advances In Harmony Search Algorithm*. 2010, Springer Berlin / Heidelberg. p. 1-10.
14. Jaberipour, M. and E. Khorram, *Solving the sum-of-ratios problems by a harmony search algorithm.* Journal of Computational and Applied Mathematics, 2010. **234**(3): p. 733-742.
15. Cobos, C., et al. *Web document clustering based on Global-Best Harmony Search, K-means, Frequent Term Sets and Bayesian Information Criterion*. in *IEEE Congress on Evolutionary Computation (IEEE CEC)*. 2010. Barcelona, Spain: IEEE.
16. Zou, D., et al., *A novel global harmony search algorithm for reliability problems.* Computers & Industrial Engineering, 2009. **58**(2): p. 307-316.
17. Zong Woo, G., *Harmony Search Algorithms for Structural Design Optimization.* 2009: p. 228.
18. Geem, Z.W., *Optimal Design of Water Distribution Networks Using Harmony Search.* 2009: p. 112.
19. Saka, M., *Optimum Design of Steel Skeleton Structures*, in *Music-Inspired Harmony Search Algorithm*. 2009, Springer Berlin / Heidelberg. p. 87-112.
20. Panchal, A., *Harmony Search in Therapeutic Medical Physics*, in *Music-Inspired Harmony Search Algorithm*. 2009, Springer Berlin / Heidelberg. p. 189-203.
21. Mahdavi, M., *Solving NP-Complete Problems by Harmony Search*, in *Music-Inspired Harmony Search Algorithm*. 2009, Springer Berlin / Heidelberg. p. 53-70.
22. Fesanghary, M., *Harmony Search Applications in Mechanical, Chemical and Electrical Engineering*, in *Music-Inspired Harmony Search Algorithm*. 2009, Springer Berlin / Heidelberg. p. 71-86.
23. Geem, Z.W. and J.C. Williams, *Ecological optimization using harmony search*, in *Proceedings of the American Conference on Applied Mathematics*. 2008, World Scientific and Engineering Academy and Society (WSEAS): Cambridge, Massachusetts.
24. Prasad, B. and Z. Geem, *Harmony Search Applications in Industry*, in *Soft Computing Applications in Industry*. 2008, Springer Berlin / Heidelberg. p. 117-134.
25. Geem, Z.W. and J.-Y. Choi, *Music Composition Using Harmony Search Algorithm.* Applications of Evolutionary Computing, 2007. **4448/2007**: p. 7.
26. Geem, Z.W., *Optimal cost design of water distribution networks using harmony search.* Engineering Optimization, 2006. **38**.
27. Mahdavi, M., M. Fesanghary, and E. Damangir, *An improved harmony search algorithm for solving optimization problems.* Applied Mathematics and Computation, 2007. **188**(2): p. 1567-1579.
28. Omran, M.G.H. and M. Mahdavi, *Global-best harmony search.* Applied Mathematics and Computation, 2008. **198**(2): p. 643-656.
29. Michalski, R.S., *LEARNABLE EVOLUTION MODEL: Evolutionary Processes Guided by Machine Learning.* Machine Learning, 2000-01-01. **38**(1): p. 9-40.
30. Lee, K. and Z. Geem, *A new meta-heuristic algorithm for continuous engineering optimization: harmony search theory and practice.* Computer Methods in Applied Mechanics and Engineering, 2005. **194**(36-38): p. 3902-3933.
31. Cendrowska, J., *PRISM: An algorithm for inducing modular rules.* International Journal of Man-Machine Studies, 1987. **27**(4): p. 349-370.
32. Witten, I.H. and E. Frank, *Data mining: practical machine learning tools and techniques with Java implementations.* SIGMOD Rec., 2002. **31**(1): p. 76-77.
33. M. Molga, C.S., *Test functions for optimization needs.* available at www.zsd.ict.pwr.wroc.pl/files/docs/functions.pdf 2005.
34. Xin, Y., L. Yong, and L. Guangming, *Evolutionary programming made faster*. Evolutionary Computation, IEEE Transactions on, 1999. **3**(2): p. 82-102.