Optimización de cadenas de adición

Fernando Oscar Aquino

Entre Ríos, Argentina
Junio de 2018

UNIVERSIDAD TECNOLÓGICA NACIONAL FACULTAD REGIONAL
CONCEPCIÓN DEL URUGUAY
DEPTO. INGENIERÍA EN SISTEMAS DE INFORMACIÓN

Optimización de cadenas de adición

Fernando Oscar Aquino

Tesis para optar al grado de Magister en Ciencias de la Computación con orientación Bases de Datos

---

Asesor Científico: Dr. Mario Guillermo Leguizamón
Universidad Nacional de San Luis

Resumen

El presente estudio aborda el problema de la computación óptima de cadenas de adición, ampliamente tratado con diferentes métodos y enfoques (tanto deterministas como estocásticos) y de interés en el ámbito de la criptografía. En este trabajo, se propone el uso del algoritmo de lobos grises o GWO (por sus siglas en inglés: Grey Wolf Optimizer) para hacer frente a este problema a fin de comparar los resultados obtenidos con otros enfoques del estado del arte. Si bien el problema en cuestión ha sido tratado mediante diferentes estrategias y para distintos tipos de exponentes, particularmente esta propuesta se centra en la optimización de cadenas de adición, asociadas a exponentes de tamaño moderado.

Agradecimientos

A la Facultad Regional de Concepción del Uruguay de la Universidad Tecnológica Nacional por haberme permitido concretar esta etapa de mi formación académica.

Mi sincero agradecimiento a mi director Dr. Mario Guillermo Leguizamón por su apoyo, paciencia y por brindarme todos sus conocimientos y asesoría para alcanzar este logro.

Quiero dedicar este trabajo a mi familia y amigos que siempre me han acompañado y me han brindado el apoyo necesario para concretar este desafío y a todos aquellos que de una u otra manera apoyaron y acompañaron en cada momento.

Índice General

Lista de Tablas

Lista de Figuras

Capítulo 1

Introducción

Existen diferentes tipos de problemas relacionados al cómputo, dentro de éstos, algunos nunca tendrán una solución porque no existe un algoritmo para resolverlos, llamados *"indecidibles"* [13], tal es el caso de demostrar que la intersección de dos LICs (Lenguajes Independientes del Contexto) es vacía. Por otro lado, existen otros problemas que sí pueden resolverse mediante algoritmos pero el tiempo necesario para obtener la solución los convierten en *"intratables"*, salvo mediante la modificación de algunos de sus requisitos y el tratamiento mediante una heurística [13]. Los problemas que tienen solución y pueden resolver las computadoras en un tiempo razonable se clasifican dentro de una clase denominada P (*"polinomiales"*), también existe una clase llamada NP ("polinómicos no deterministas") de modo que: P $\subseteq$ NP, para esta última clase de problemas los algoritmos que se conocen tienen un elevado costo computacional al igual que los intratables, pero no se ha demostrado aún que no existe un algoritmo polinomial capaz de resolverlos, tampoco que sean intratables, ejemplo: *problema del viajante de comercio*. Si se demostrara que existe para éste u otro de los problemas de esta clase un algoritmo polinomial, todos los problemas de dicha clase tendrían una solución de este tipo y se resolvería la pregunta: ¿Es P = NP? (problema abierto más importante de la Teoría de la Computación). A este tipo especial de problemas de la clase de complejidad NP se los ha denominado NP-Completos [13].
La criptografía actualmente constituye un problema de la clase NP, debido a que está basada en la factorización de números para lo que emplea operaciones como la *exponenciación modular*. Los algoritmos asimétricos de criptografía utilizan la exponenciación modular para cifrar y descifrar datos. El costo computacional producto de la gran cantidad de operaciones de multiplicación, que implica resolver potencias de número de orden considerable, se puede evitar mediante el uso del concepto de "cadenas de adición" [2]. Sin embargo, un exponente no posee un única cadena de adición válida asociada y por tanto hallar una cadena válida y cuya longitud resulte mínima constituye un problema NP-Completo. Se denomina exponenciación modular [8] a la operación cuya finalidad es obtener un valor b (entero, positivo) para

satisfacer la siguiente ecuación:

$$b = A^e mod P \tag{1.1}$$

Dado un valor A entero, arbitrario en el rango [1,P −1] y un número e entero, positivo y arbitrario, se define exponenciación modular al problema de encontrar un valor único y entero positivo $b \in$ [1,P − 1], capaz de satisfacer la Ecuación (1.1) siendo para el problema estudiado aquí, la función objetivo a optimizar para poder obtener un valor b mínimo (i.e., una cadena de adición de longitud mínima). El problema radica en que a medida que se incrementa el exponente e, aumenta el número de operaciones a realizar para resolver la Ecuación (1.1), con lo cual se requiere una alternativa para optimizar el proceso: la alternativa es emplear cadenas de adición para poder reducir el número de multiplicaciones para resolver una potencia. Ejemplo: dado el exponente $e = 91$, en vez de multiplicar la base 91 veces es factible contar con una cadena asociada al número 91, como ser: C = 1,2,4,6,8,14,22,36,58,80,88,90,91 y entonces resolver la potencia consistiría en multiplicar la base por cada uno de los exponentes de C, es decir que para este ejemplo se realizan sólo 13 operaciones. Sin embargo y teniendo en cuenta que no existe una única cadena para un exponente, hallar una cadena de longitud mínima, constituye un problema de optimización combinatoria y puede abordarse empleando una metaheurística para evitar las pruebas por fuerza bruta para hallar un valor óptimo (cadena de longitud mínima).

1.1 Objetivo General

El objetivo principal de este trabajo de tesis es adaptar una metaheurística no explorada, según el estado del arte, como el caso de Grey Wolf Optimizer, se usará de ahora en más sus siglas GWO para referirnos al algoritmo, para abordar el problema de generación de cadenas de adición de longitud mínima, incluso a exponentes del rango de 128-bits o cercanos.

1.1.1 Objetivos específicos

- Obtener cadenas de adición de longitud igual o menor, para exponentes pequeños, utilizados en las pruebas de otros abordajes del estado de arte, a fin de verificar que la metaheurística propuesta logra resultados competitivos.
- Adaptar la metaheurística elegida para obtener cadenas de adición de longitud mínima para exponentes pequeños y del orden de los 128-bits o cercanos, obteniendo resultados superiores o equivalentes a otras propuestas.

- Realizar pruebas estadísticas paramétricas y no paramétricas, según corresponda, para cotejar rigurosamente los resultados obtenidos con la metaheurística propuesta respecto a los algoritmos del estado del arte.

1.2 Hipótesis

La adaptación del algoritmo basado en manada de lobos grises o GWO, adecuada al problema, permitirá obtener cadenas de adición de longitud mínima, incluso para exponentes del orden de los 128-bits, con resultados competitivos respecto de otras propuestas del estado del arte. El uso de metaheurísticas ha dado buenos resultados y GWO (ejemplo de otra metaheurística) puede ser un candidato para obtener resultados competitivos.

1.3 Justificación

Puesto que los sistemas de criptografía asimétricos utilizan cadenas de adición, dotar a los mismos de un mecanismo para hallar cadenas de longitud mínima sería un beneficio para ellos, debido a la reducción del costo computacional inherente al proceso de cifrado y descrifrado de datos.

1.4 Aportes de la tesis

Durante el proceso de elaboración de esta tesis se publicó el siguiente artículo:

- F. Aquino, G. Leguizamón. *Optimización de Cadenas de Adición.* XVII Workshop Agentes y Sistemas Inteligentes (WASI). En Actas del XXII Congreso Argentino en Ciencias de la Computacion (CACIC), páginas 114-126. San Luis, Argentina, 2016.
- Además se está preparando actualmente un nuevo artículo para revista llamado "Optimization of Addition Chains Using Gray Wolf Optimizer".

1.5 Organización del informe

El resto del documento de tesis está estructurado de la siguiente manera. En el Capítulo 2 se desarrolla el marco teórico de referencia de la temática, se describen distintos tipos de cadenas de adición y los algoritmos de exponenciación modular

presentes en la literatura del tema. El Capítulo 3 cita y describe las distintas metaheurísticas bioinspiradas que han sido empleadas como instrumento de abordaje en estudios anteriores. En el Capítulo 4 se describe el trabajo relacionado con la generación y optimización de cadenas de adición de longitud mínima, teniendo en cuenta tanto los métodos deterministas como estocásticos. En el Capítulo 5 se describe la propuesta objeto de este trabajo que consta de la adaptación de la metaheurística: optimización basada en manada de lobos grises GWO para el problema de generación óptima de cadenas de adición (GWOc), los experimentos propuestos y discusión de los resultados. En el Capítulo 6 se presenta una propuesta alternativa de GWO con mecanismo de *ventana deslizante* para tratamiento de exponentes de mayor rango numérico (GWOc_SWM), los experimentos asociados a la propuesta y discusión de los resultados obtenidos. El Capítulo 7 plantea una propuesta alternativa de mejora de GWOc mediante combinación con *Búsqueda Local* (LGWOc), con sus respectivos experimentos y la discusión de los resultados. Finalmente en el Capítulo 8 se exponen las conclusiones y trabajo futuro propuesto.

Capítulo 2

Marco Teórico

2.1 Concepto de criptografía

Criptografía o Criptología (del griego *"krypto"* y *"logos"*, significa el estudio de lo oculto o escondido). Más precisamente, es la ciencia que trata los problemas teóricos relacionados con la seguridad en el intercambio de mensajes (codificados) entre un emisor y un receptor a través de un canal de comunicaciones, por ejemplo: una red de computadoras. La criptografía es una evolución tecnológica de lo que en la antigüedad se conocía como *Esteganografía* (encubrimiento) [9] la cual era un procedimiento para ocultar mensajes y poder enviarlos al receptor sin ser descubiertos, de tales procedimientos se encuentran detalles desde el siglo V a.C. descriptos por el célebre historiador griego Heródoto [9]. Mucho más tarde también se conocieron ciertos procedimientos, durante la segunda guerra mundial, por ejemplo: uno que básicamente trataba de ocultar un mensaje (microfilmado), reduciendo el mismo hasta el extremo de un pequeño punto de manera que el mismo podía pasar a simple vista como un signo de puntuación o un caracter dentro de otro texto, sin revelarse su contenido más que para quién sabía que dicho caracter albergaba un mensaje [9]. De la mano de los avances de la ciencia la *Esteganografía* evolucionó al grado de dar lugar a diversas técnicas que se conocen con el nombre de Criptografía pero ésta, a diferencia de la *Esteganografía* no intenta ocultar la existencia de un mensaje sino más bien de ocultar su significado, mediante algún mecanismo de codificación.

Esta ciencia está dividida en dos grandes ramas: la criptografía, ocupada del diseño de los artefactos capaces de lograr una codificación robusta para ocultar el significado de los mensajes (criptosistemas) y el criptoanálisis que trata de descifrar los mensajes codificados en clave, intentando vulnerar los criptosistemas [9].

2.2 Introducción a la criptografía

El criptoanálisis es uno de los componentes o ramas de la criptografía y puede definirse como la disciplina que se centra en el diseño y estudio de las técnicas tendientes a descifrar comunicaciones o mensajes codificados sin conocer las claves correctas para haber generado dicha codificación. Para esto existen diversas técnicas, dentro de las más comunes y descriptas en [2]:

- Ataques sólo a texto cifrado: en estos casos el atacante (criptoanalista) o sistema de criptoanálisis no conoce ninguna parte del mensaje original codificado o en clave y sólo debe trabajar a partir de texto o mensaje cifrado.
- Ataque a texto sin cifrar conocido: el atacante o sistema de criptoanálisis conoce o puede inferir el texto sin cifrar para algunas partes del texto cifrado. Esta tarea básicamente consiste en la posibilidad de descifrar el resto de los bloques de texto cifrado a partir de esta información. Esto es factible de realizar determinando la clave utilizada para cifrar los datos.
- Ataque a texto sin cifrar elegido: el atacante o sistema de criptoanálisis puede obtener alguna porción del texto cifrado, a elección. Al igual que el caso anterior, la tarea consiste en determinar la clave utilizada para el cifrado, para decodificar el bloque elegido. Algunos métodos de cifrado (por ejemplo, el algoritmo de clave pública o asimétrico RSA [16]) suelen ser vulnerables a los ataques de texto sin cifrar elegido.
- Ataque "man-in-the-middle" (hombre en el medio): en este tipo de ataque el atacante o sistema de criptoanálisis mediante intromisión, se coloca en medio de las partes legítimas que se comunican (emisor y receptor), el principal objetivo de este tipo de ataques son los protocolos de intercambio de claves en las comunicaciones. Este tipo de ataque radica en que cuando dos partes se intercambian claves (mediante un mecanismo seguro de comunicaciones) el atacante se coloca entre las partes, en la línea de comunicaciones, ejecutando un intercambio de clave por separado con cada parte. Las partes finalizarán utilizando una clave diferente cada una, de manera que ambas son conocidas por el atacante, descifrando en consecuencia dicha comunicación. Las partes (emisor y receptor) creerán que se están comunicando de forma segura, pero el atacante del sistema está "escuchando" y capturando los mensajes.
- Ataque de tiempo: este tipo es más novedoso que los anteriores y se basa en la medición repetida de los tiempos de ejecución (exacta) de las operaciones de *exponenciación modular* [16], algunos de los objetivos de este tipo de ataques son los métodos criptográficos: RSA, Diffie-Hellman y las técnicas basadas en Curvas Elípticas. Generalmente los algoritmos en su implementación realizan cálculos de tiempo de ejecución no constante, debido al proceso de optimización del rendimiento. Si un sistema de criptoanálisis logra instrumentar

un mecanismo capaz de recuperar información estadística de los tiempos de ejecución de las operaciones o variaciones de dichos tiempos, puede recuperar a partir de esta información el valor de ciertos parámetros secretos o revelar detalles de la implementación del algoritmo.

- Criptoanálisis Diferencial, Lineal y Lineal-Diferencial: el criptoanálisis diferencial es un tipo de ataque que puede aplicarse a codificadores de bloque iterativos. Dentro de este tipo tenemos los algoritmos simétricos o de clave secreta: DES (por sus siglas en inglés: Data Encryption Standard), 3DES (por sus siglas en inglés: Three Data Encryption Standard, es decir DES utilizando 3 claves), etc., de hecho estas técnicas fueron introducidas y posteriormente mejoradas y se utilizaron para atacar el DES. El criptoanálisis diferencial es un ataque aplicado sobre textos sin cifrar, escogidos previamente y se basa en un análisis de la evolución de las diferencias entre dos textos sin cifrar relacionados cuando se cifran bajo la misma clave. Mediante un cuidadoso análisis de los datos disponibles, las probabilidades pueden asignarse a cada una de las posibles claves y eventualmente la clave más probable se identifica como la correcta. El segundo caso de este grupo es el criptoanálisis lineal que fue ideado para un ataque sobre FEAL (por sus siglas en inglés: Fast data Encipherment Algorithm). Fue extendido por Matsui para atacar al DES. Es un ataque dirigido a texto no cifrado conocido y emplea una aproximación lineal para describir el comportamiento del cifrador de bloque. En función de esto, al tener suficientes pares de texto sin cifrar y los correspondientes textos cifrados, es posible obtener los bits de información acerca de la clave de cifrado y a medida que aumenta la cantidad de pares (texto no cifrado con sus correspondientes cifrados) dan lugar a una elevada probabilidad de éxito en el ataque. Posteriormente, han surgido una serie de mejoras de las cuales ha resultado un tipo denominado lineal – diferencial que combina elementos de ambos tipos de criptoanálisis.

- Ataques de diccionario: a diferencia de los casos anteriores, el objetivo de este tipo de criptoanálisis no es obtener la clave, sino directamente el dato concreto que se pretende ocultar, ya que el método de cifrado es público y aunque no se disponga de la clave se puede reproducir. Este es el caso que suele aplicarse en los sistemas de cifrado de claves de acceso, en sistemas operativos, para vulnerar claves de usuario, teniendo en cuenta que cuando un usuario se da de alta, introduce su código y clave de acceso. Luego se cifra dicha clave y posteriormente se compara el resultado con la clave cifrada (almacenada en el fichero de claves), al momento de un intento de inicio de sesión con estas credenciales y si son iguales, el sistema considera que el usuario es quien dice ser, permitiendo el acceso. Teniendo en cuenta lo anterior, los sistemas de criptoanálisis dedicados a vulnerar claves, parten del hecho de obtener una copia del archivo de claves, a fin de comprobar si existe alguna cuenta de usuario sin clave asociada, para valerse de la misma. Estos sistemas utilizan un diccionario con grandes cantidades de palabras y combinaciones comunes y válidas como claves. Dicho diccionario es cifrado empleando la utilidad de cifrado del sistema

operativo en cuestión o una copia de dicha herramienta y a continuación se ejecuta un proceso de comparación del archivo de claves del sistema con el resultado del cifrado, de modo que si se producen coincidencias para al menos un caso, se puede inferir cuál es la palabra o combinación de caracteres oculta bajo una clave.

- Ataques por búsqueda exhaustiva o fuerza bruta: este tipo de ataque se realiza generando de forma aleatoria todos los posibles valores de las claves de acceso de un sistema y transformándolas, de manera que se pueda elegir la clave de acceso cuya transformada coincida con la interceptada. Este criptoanálisis resulta útil cuando la clave es de tamaño reducido, ya que en caso contrario el número de combinaciones sería impracticable, sin embargo para claves de gran longitud es factible agilizar este tipo de ataque mediante técnicas de hardware, pero a un costo muy elevado.

2.3 Clasificación de la criptografía

En los sistemas de criptografía se contemplan tres aspectos fundamentales [30]:

1. El tipo de operación empleada para la transformación del texto original en texto cifrado. Estas operaciones se clasifican de forma general en: sustitución y transposición o permutación.

    - Sustitución: este mecanismo básicamente consiste en reemplazar los caracteres del mensaje original por otros diferentes, siendo los nuevos caracteres que reemplazan a los del mensaje original de cualquier tipo: letras, números, símbolos o caracteres especiales.
    - Transposición o permutación: mecanismo de cifrado que consiste en cambiar el orden de los caracteres dentro del mismo texto original. Básicamente se trata de colocar el texto original en una tabla o matriz de "n" columnas y una clave de cifrado "c" que consta del número "n" junto con el orden en el que deben ser leídas las columnas del mensaje original para dar origen al mensaje cifrado.

2. El método de procesamiento del texto a ser cifrado, este puede ser: mediante bloques, bits, bytes, etc.

3. El número de claves utilizadas para el mecanismo de cifrado y descifrado de textos, siendo este número: uno o más y de éstos dos tipos es que surje la clasificación según dicho aspecto.

Los sistemas criptográficos se pueden clasificar en dos grandes tipos, dependiendo del número de claves que utilizan [30]:

- Sistemas de clave privada: también llamados de criptografía convencional o simétrica.
- Sistemas de clave pública: también llamados de criptografía asimétrica.

Criptografía de clave privada o simétrica

Es la utilizada por aquellos sistemas o mecanismos de criptografía que emplean la misma clave para cifrar y descifrar mensajes. Estos mecanismos toman como entrada el texto original y obtienen otra versión diferente del mismo mediante la clave simétrica de cifrado. Poseen un complejo intercambio de claves, siendo esto su mayor desventaja: Requieren $n(n-1)/2$ claves por cada "n" puntos que se comunican entre sí (emisor y receptor) y dichas claves no vuelven a emplearse una vez utilizadas. Por ejemplo: para 1000 puntos de comunicación que interactúan entre sí, se requieren 499.500 claves por cada uno.

Criptografía de clave pública o asimétrica

Es la utilizada por aquellos sistemas o mecanismos de criptografía que utilizan un par de claves para el envío de mensajes entre emisor y receptor: una clave privada y una pública. De modo tal, que el remitente usa la clave pública del destinatario para cifrar el mensaje, entonces una vez cifrado, sólo la clave privada del destinatario podrá descifrar el mensaje original. La principal ventaja que aportan frente a los mecanismos simétricos es que evitan el inconveniente del intercambio de claves. Requieren sólo "n" pares de claves para comunicar mensajes, por cada "n" puntos que se comunican entre sí (emisor y receptor).

2.3.1 Las firmas digitales

Con el auge de las comunicaciones y la posibilidad de compartir datos entre sistemas, muchas veces a través de un medio inseguro como sería la red amplia mundial (Internet), los Criptosistemas como RSA han tenido que implementar mecanismos a fin de asegurar que quién recibe un mensaje de un remitente (emisor) tenga seguridad que dicho mensaje proviene de quién dice ser dicho remitente, y a su vez, que el emisor tenga certeza de que su mensaje llego sin alteraciones al destinatario (receptor). Un ejemplo de estos mecanismos son las *Firmas Digitales* [16], que constituyen un caso de aplicación de la Criptografía. Dichas firmas se aplican a mensajes de longitud arbitraria y se generan calculando primeramente un valor *hash* (resumen) del mensaje original y luego firmando el mismo con la firma digital obtenida, tal como se ejemplifica en la Figura 2.1.

Una firma digital es una transformación que vincula de forma única un documento con la clave privada del firmante del mismo (ver Figura 2.1), mediante el uso de una función para firmar y otra para verificar la firma, tal como se expone en [16]. A continuación se describen dichos procesos.

- Firma
  Esta operación puede describirse de la siguiente manera:
  - Considerar un emisor A, un receptor B, un mensaje M y una firma digital S.
  - El firmante F genera una firma digital S para un mensaje M.
  - Se ejecuta la firma S= S_a (M), siendo S_a la función que aplica la firma de A sobre el mensaje M.
  - Se envía al receptor B el par (M,S).
- Verificación
  - Una vez cumplida la secuencia anterior, B verifica que la firma S aplicada al mensaje M, haya sido generada por A, para esto:
  - B aplicará una función de verificación V_a, sobre el par (M,S) recibidos de A.
  - De este modo B calcula: V = V_a (M,S)
  - El resultado de lo anterior será verdadero o falso, según sea aceptado o rechazado el resultado de V.

Las firmas digitales trabajan y operan bajo esquemas de clave pública, en donde la clave privada se utiliza para firmar, y la pública para verificar la firma. Con el fin de asegurar tanto la integridad del documento como la identidad del remitente. Es aconsejable que se utilicen diferentes claves para el cifrado y para la firma digital. Este tipo de criptosistemas utilizan la exponenciación modular para las operaciones de cifrado y descifrado de datos.
Las firmas digitales entre otros sistemas de cifrado de clave pública han cobrado importancia en la actualidad producto del desarrollo de los sistemas de cómputo e información y la importancia de proteger los activos de información de organizaciones de diversos rubros y sectores de la sociedad, como las empresas del rubro bancario que deben realizar operaciones y movimientos económicos, intercambiando información sensible como datos personales, estados de cuenta, etc.; entre entidad y clientes o bien entre dichas entidades y terceros, asegurando el resguardo de dicha información.

Figura 2.1: Esquema de uso de firma digital.

2.4 Criptografía de curvas elípticas (Eliptic Curve Cryptography)

Este tipo de criptografía está basado en la teoría matemática de las curvas elípticas sobre la cual sólo se hace una breve introducción conceptual a continuación, debido a la complejidad del tema, sin embargo para un análisis y revisión detallada se puede consultar [29].

Curvas elípticas sobre números reales

Las curvas elípticas son ecuaciones cúbicas de dos variables, cuya forma general es como se ejemplifica en ecuación 2.1. Son curvas sobre números reales.

$$Y^2 = X^3 + aX + b \tag{2.1}$$

En principio podemos pensar en una curva elíptica como el conjunto de soluciones de una ecuación de la forma anterior. Para el empleo de las mismas en criptografía nos resulta de interés contemplar dichas curvas pero sobre un campo numérico finito (diferente al de los números reales). En este sentido, el uso más frecuente resulta ser el campo de los números primos enteros, ya que se aplica aritmética modular de un número primo p.

Una curva elíptica sobre el campo numérico Zp (conjunto de números enteros primos p), con un determinado valor de p, es el conjunto de todos los pares ordenados $(x, y) \in Zp$ que satisfacen:

$$Y^2 = X^3 + aX + b \mod p \tag{2.2}$$

Así como mediante los sistemas criptográficos tales como el algoritmo RSA (Rivest, Shamir y Adleman) [16], la finalidad es factorizar números, cuando se trabaja con curvas elípticas el objetivo es tratar de obtener logaritmos o en realidad un grupo de logaritmos que mientras más complejo sea, más seguro resulte el criptosistema o el esquema de cifrado.

En álgebra, un grupo es un conjunto de elementos unidos a una operación matemática, dicha operación que podríamos denominar ($\otimes$), debe cumplir con ciertas propiedades a saber:

- La operación debe ser cerrada: esto significa que si dos elementos, por ejemplo: a, b, pertenecen al grupo, el resultado de $a \otimes b$ también deberá pertenecer al grupo.
- La operación debe ser asociativa.
- Debe cumplir con existencia de elemento neutro, de modo que: $a \otimes e = e \otimes a$, donde e es el elemento neutro respecto de $\otimes$.
- Debe cumplir con existencia de elemento opuesto.

Éstas son las propiedades que definen a un grupo, en particular para este tipo de criptosistemas, los que son de interés son los grupos "cíclicos", es decir, aquellos que mediante un elemento llamado generador, son capaces de generar todos los elementos que lo componen [29].

El algoritmo básico de un mecanismo de curvas elípticas tendría por ejemplo, los siguientes pasos:

1. Elegir una curva elíptica. La curva elíptica tiene un conjunto de soluciones (x,y).
2. Si los valores (x,y) pertenecen a un campo finito, entonces los puntos (x,y) de la curva forman un grupo.
3. Tomamos un elemento del grupo, y hallamos su logaritmo discreto para una base dada.
4. Eso nos servirá de base para establecer algoritmos criptográficos de intercambio de claves y de firma digital.

El empleo de los sistemas basados en curvas elípticas surge como una alternativa a los sistemas criptográficos clásicos de clave pública, como el RSA, debido a la ventaja de poder utilizar claves más reducidas en tamaño con costos computacionales menores tal como se describe en [29].

2.5 Cadenas de adición

Una cadena de adición [31] C cuya longitud podemos expresarla como Lc, es una secuencia de números enteros positivos C = $[c_1; c_2; c_3; ...; c_i; ...; c_{L_c}]$, con $c_1 = 1, c_2 = 2$ y $c_{L_c} = e$ y $c_{i-1} < c_i < c_{i+1} < c_{L_c}$, donde cada elemento c_i se obtiene de la suma de dos elementos previos, no siempre distintos y e resulta ser el número asociado de dicha cadena, en otras palabras una cadena de adición C para un número dado e, es una secuencia de números que debe cumplir las siguientes propiedades:

- El primer elemento de la cadena es el número uno.
- Cada número es la suma de dos números anteriores.
- El número dado e se ubica al final de la cadena.

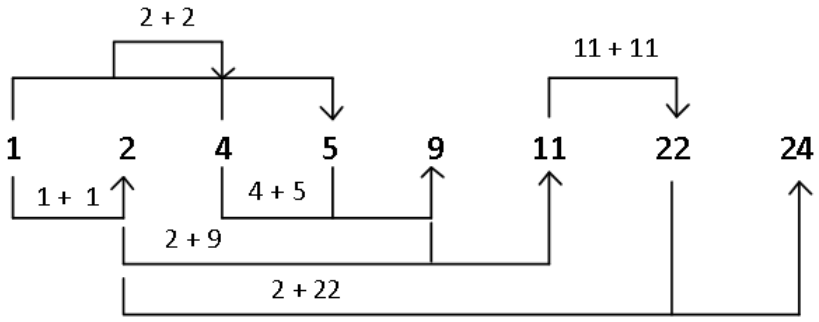

Figura 2.2: Obtención de una cadena de adición para e=24.

A continuación se presentan tres ejemplos de cadenas de adición, para $e = 91$, e=177 y e=24. Para este último se muestra en la Figura 2.2 la obtención de cada componente de la cadena de adicion asociada.

C_1 = [1; 2; 3; 5; 6; 11; 22; 33; 36; 58; 80; 91].
C_2 = [1; 2; 4; 5; 10; 11; 22; 44; 88; 176; 177].
C_3 = [1; 2; 4; 5; 9; 11; 22; 24].

Entonces, una cadena de adición se puede ver como la secuencia numérica asociada a e, siendo el valor entero Lc la longitud de la cadena de adición C. Vale mencionar que la longitud es igual al número de elementos de la cadena menos uno.

2.6 Clases de Cadenas

Existen diversos tipos de cadenas de adición, las cuales han sido definidas en el transcurso del tiempo de acuerdo con las necesidades que involucraban su uso.

2.6.1 Cadenas de adición euclidianas

Son una variante de las cadenas de adición definidas anteriormente. La única diferencia que presenta una cadena Euclidiana respecto de otra que no lo es, radica en la restricción de no permitir el "doblado" de números, es decir, un número que conforma la cadena proviene de la suma de dos números previos, distintos.

Dicho lo anterior podemos denotar a una cadena euclidiana Ce con longitud L_{Ce} a un secuencia de número enteros positivos:

$$Ce = [c_1;\ c_2;\ c_3;...;\ c_i;\ ...\ c_{L_{Ce}}]$$

Donde:

- c_1= 1, c_2=2, $c_{L_{Ce}} = e$,
- $c_{i-1} < c_i < c_{i+1} < c_{L_{Ce}}$,
- c_i se obtiene de la suma de dos elementos previos pero siempre distintos.
  i < j ∧ k < j
  ∃ j, k / j ≠ k ∧ $c_i = c_j + c_k$

Por su naturaleza, la longitud L_{Ce} de una cadena de adición euclidiana mínima sería de mayor o igual longitud a una cadena de adición tradicional.

El uso de este tipo de cadenas en los criptosistemas se justifica en ciertos casos ya que reducen el riesgo de diversos tipos de ataques criptográficos que se realizan durante el proceso de cifrado y descifrado.

2.6.2 Cadenas de adición vectoriales

Son un tipo de cadenas [17] que se caracterizan por estar conformadas por vectores, donde cada vector es producto de la suma de dos vectores pertenecientes a la cadena. Una cadena vectorial Cv cumple las siguientes propiedades:

- Los vectores iniciales (elementos iniciales) son los vectores unitarios:
  [1, 0,..., 0]; [0, 1, 0,..., 0]; ... ; [0 ,..., 0, 1].
- Cada vector es la suma de dos vectores anteriores.
- El último vector de la cadena es igual al vector dado.

En [16] se puede ver el siguiente ejemplo de una cadena de adición vectorial:

$[1; 0; 0]$
$[0; 1; 0] \rightarrow [[0; 1; 1][1; 1; 1][0; 1; 2][1; 2; 3][1; 3; 5][2; 4; 6][3; 7; 11][4; 8; 12]\underline{[7; 15; 23]}]$
$[0; 0; 1]$

En el ejemplo anterior vemos que el vector [0; 1; 1] se conforma de la suma de los vectores unitarios: [0; 1; 0], [0; 0; 1], luego el vector [1; 1; 1] se conforma de la suma del primer vector unitario [1; 0; 0] y el elemento número 1 de la cadena: [0; 1; 1] y luego continua el proceso de generación tomando dos vectores anteriores para dar origen a un nuevo elemento hasta llegar a [7; 15; 23], obteniéndose una cadena vectorial Cv cuya longitud $L_{Cv} = 9$.

Este tipo de cadenas de adición pueden emplearse para encontrar las secuencias mínimas de adición o en exponenciación con vectores.

2.6.3 Cadenas de adición – sustracción

Este tipo de cadenas [31] son similares a las cadenas de adición tradicionales pero éstas se denotan como se expresa a continuación.

Una cadena de adición – sustracción, en adelante: AS, con longitud L_{as} se puede definir como aquella secuencia de números enteros positivos de la forma:

$$AS = [as_1; as_2; as_3;...; as_i; ... as_{L_{as}}]$$

Donde:

- $as_1 = 1$; $as_2 = 2$; $as_{L_{as}} = e$
- $as_i = as_j +- as_k \wedge j; k < i$

Este tipo de cadena fue desarrollado, al igual que las cadenas vectoriales, debido a necesidades puntuales que no eran satisfechas por las cadenas de adición tradicionales, por tanto no son aplicables a todos los tipos de operaciones.

El caso puntual de las cadenas AS tinen aplicación en los algoritmos de criptografía basados en curvas elípticas [29] y especialmente han demostrado robustes frente a una técnica específica de criptoanálisis denominada ataque de canal lateral, tal como se describe en el trabajo [11].

2.7 Aplicaciones de las cadenas de adición

Las cadenas de adición tienen aplicación en el cifrado y descifrado de datos en algoritmos asimétricos o de llave pública, con el objetivo de reducir el número de multiplicaciones que supone la operación utilizada para estos fines: *exponenciación modular* [16]. Estas operaciones implican un gran consumo de recursos computacionales, tales como el uso de memoria y procesador. Este tipo de exponenciación es una operación que pertenece al ámbito de la *aritmética modular* y a continuación se describe en que consiste.

2.7.1 Aritmética modular

A través de la historia la aritmética modular ha tenido incidencia en diversas actividades o disciplinas, tales como: la teoría de números, el álgebra abstracta, las artes, pero sobre todo, en la criptografía y seguridad de la información. En la actualidad ha cobrado relevancia, debido entre otros factores, a la necesidad de las grandes empresas de mantener sus activos de información de forma segura sin impactar en el acceso y disponibilidad; esto se ha podido lograr mediante algunas estrategias como el uso de códigos, en los que se aplican operaciones como la factorización de grandes números. Si bien los primeros estudios sobre *aritmética modular* se sitúan en una época remota (Siglo I d.C) y fueron los chinos los primeros en abordar el tema, posteriormente fue Gauss el primero en hacer un estudio sistemático y metódico de *aritmética modular* como parte de su obra [12].

Definición de congruencia

Dado $m \in \mathbb{Z}$ (conjunto de enteros), con $m > 1$, se dice que $a, b \in \mathbb{Z}$ son congruentes módulo m si y sólo si $m|(a - b)$, es decir si m es divisible por la diferencia de dichos números. Se denota esta relación como $a \equiv b \pmod{m}$, donde m se denomina módulo de congruencia.

Cabe mencionar que si m divide al resultado de la operación $(a - b)$, esto supone que a y b tienen ambos el mismo resto, al ser divididos por el módulo m.

Ejemplo: $23 \equiv 2$ módulo 7, se lee: "23 es congruente módulo 7 con 2", pues $23 = 7 * 3 + 2$

La congruencia módulo m es una relación de equivalencia para todo $m \in \mathbb{Z}$, por lo tanto, cumple con: reflexividad, simetría y transitividad. Puesto que es una relación de equivalencia, es posible definir el conjunto cociente de las clases de equivalencia

resultantes de la relación de congruencia, de este modo, la relación clasifica a cualquier entero según el resto obtenido al dividirlo por el módulo m correspondiente [12].

2.7.2 La exponenciación en aritmética modular

Dentro de la aritmética modular y sus operaciones, para el caso de los algoritmos de criptografía, nos interesa conocer particularmente la exponenciación.

Las operaciones aritméticas que hacen la mayoría de las computadoras son de este tipo, donde el modulo es 2^b siendo b el número de bits de los valores sobre los que operamos.

La exponenciación clásica es aquella operación que consiste en multiplicar la base por sí misma tantas veces como indica el exponente; lo que produce que el número de multiplicaciones se incrementa en función del exponente. Por otro lado la *exponenciación modular* es una técnica de amplio uso en algoritmos criptográficos de cifrado asimétrico y cuya diferencia con la exponenciación clásica, radica en que la primera es menos costosa de realizar que su operación homóloga en la aritmética entera. Esto se debe a que después de cada multiplicación se calcula el modulo del resultado para que el resultado temporal no se incremente en exceso.

Nótese que $a * b \mod m = [(a \mod m)(b \mod m)] \mod m$, y que por grande que sea el exponente, nunca es necesario multiplicar por números enteros mayores que el módulo m.

2.7.3 Algoritmo básico de exponenciación modular

En el Algoritmo 1 la cantidad de operaciones (multiplicaciones y operaciones de módulo) estará determinada por el valor del exponente, es decir que si dicho valor es 100.000, se ejecutarán 100.000 multiplicaciones y 100.000 operaciones de módulo (divisiones donde nos quedamos con el residuo). En resumen si bien la exponenciación modular es menos costosa que la clásica, vemos que por más que utilicemos esta variante, cuando el exponente se incrementa a valores elevados, no se reduce la cantidad de operaciones y en términos de costo computacional el método resulta ser poco eficiente. Una manera de resolver esta situación es mediante el empleo de cadenas de adición tal como describe [7] y otros estudios. El Algoritmo 1 resume la operación de exponenciación modular.

Algoritmo 1 exponenciación modular

Entrada: a, e y m.
Salida: $exp = a^e \bmod m$.

```
1: exp := a; i := 0;
2: mientras (i < e); i++ hacer
3:     exp := (exp × a) mod m
4: fin mientras
5: devolver exp
```

Donde:

a : base (número entero)
e : exponente (número entero)
m : módulo (número entero)
exp : exponente (número entero)

Por ejemplo, si se desea calcular $a^{17} \pmod 9$, una manera predecible de realizarlo, es multiplicar la base a, 16 veces y de manera análoga calcular el módulo. No obstante, una manera más eficiente de calcularlo es empleando cadenas de adición. Dada la cadena de adición C, donde C = [1; 2; 4; 6; 10; 16; 17], valiéndonos de las propiedades de la potencia, el cálculo se reduce a 6 multiplicaciones:

$a^1 = \text{a} \pmod 9$
$a^2 = a^1 \times a^1 \pmod 9$
$a^4 = a^2 \times a^2 \pmod 9$
$a^6 = a^2 \times a^4 \pmod 9$
$a^{10} = a^4 \times a^6 \pmod 9$
$a^{16} = a^6 \times a^{10} \pmod 9$
$a^{17} = a^1 \times a^{16} \pmod 9$

El mecanismo de cómputo descripto anteriormente es el que emplean los algoritmos criptográficos de clave pública como RSA [16]. Cable destacar que el algoritmo de clave pública RSA, es uno de los sistemas criptográficos asimétricos más conocidos y usados. Sus creadores [Rivest - Shamir - Adlman] idearon este algoritmo y fundaron la empresa RSA Data Security Inc., una de las más prestigiosas en el ámbito de la protección de datos. El fundamento de RSA es la dificultad de factorizar números muy grandes, ya que para factorizar un número, el método más lógico consiste en dividir sucesivamente dicho número entre 2, 3, 4, y así sucesivamente, buscando siempre que el resultado de la operación sea exacto (cuyo resto sea igual a cero), obteniéndose un divisor del número en cuestión.

Valiéndose de la exponenciación modular de exponente y módulo fijos, RSA crea sus claves tal como describe el Algoritmo 2.

Algoritmo 2 algoritmo criptográfico RSA

Entrada: p, q (aleatorios)
Salida: d, n

1: obtener los números:
   $n = p \times q$
   $\phi = (p-1) \times (q-1)$;
2: obtener e
3: Se calcula $d = e^{-1} \mod \phi$, con mod = resto de la división de números enteros.

   Y ya con estos números obtenidos, n es la clave pública y d es la clave privada. Los números p, q y ϕ se destruyen. También se hace público el número e, necesario para alimentar el algoritmo.

Donde:

p, q : números primos grandes, por ejemplo del orden de los 100 o hasta 300 dígitos.
n : clave pública.
d : clave privada.
e : número sin múltiplos comunes con ϕ, capaz de satisfacer la ecuación 2.3.

$$mcd(d; (p-1) \times (q-1)) = 1 \tag{2.3}$$

En [16] se muestra un ejemplo sencillo de dicho proceso, construyendo un criptosistema RSA con la siguiente información.
Dados un mensaje $M = 50$ (mensaje en texto plano), $p = 11$ y $q = 13$. En el contexto de este ejemplo, llamaremos M a un mensaje en texto plano (no cifrado) y $\hat{M}$ a su equivalente cifrado.

El método opera con exponente público y módulo $(e; n) = (13; 143)$, y conserva como privado, los siguientes datos: $d = 113$, $p = 11$, $q = 13$. De este modo el proceso típico de cifrado / descifrado se ejecuta según los pasos del Algoritmo 2:

$n = p \times q = 143$
$\phi = (p-1) \times (q-1) = (11-1) \times (13-1) = 120$
$e = 17$, arbitrario pero que no tenga múltiplos en común con ϕ.
$d = e^{-1} \mod \phi = 17^{-1} \mod 120 = 113$

En este ejemplo M representa el mensaje en texto plano y $\hat{M}$ al mensaje cifrado.

$M = 50$
$\hat{M} = M^e \pmod{n}$
$\hat{M} = 50^{17} \pmod{143}$
$\hat{M} = 85$

$\hat{M} = 85$
$M = M^d \pmod{n}$
$M = 85^{113} \pmod{120}$
$M = 50$

Ejemplo de algoritmo Criptográfico RSA

A continuación se muestra un ejemplo donde el mensaje M consiste de la palabra "HOLA". De este modo si se desea intercambiar el mensaje "HOLA" de manera segura entre dos partes: A (emisor) y B (receptor), de modo tal, que A cifra el mensaje con su clave privada y B descifra el mismo con la clave pública de A, y de manera análoga si B envía un mensaje a A, éste será cifrado con la clave privada de B y descifrado por A con la clave pública del B.

Suponiendo que la clave privada de A es (53, 143) y su clave pública es (77,143), se procedería de la siguiente manera:

H $72^{53} \bmod 143 = 128$
O $79^{53} \bmod 143 = 118$
L $77^{53} \bmod 143 = 15$
A $69^{53} \bmod 143 = 49$

Siendo "HOLA" $\rightarrow H = 72, O = 79, L = 77, A = 69$

Nótese que el procedimiento detallado en el Algoritmo 2 debe aplicarse para cada caracterer del mensaje a cifrar.

En función de lo anterior, para fines prácticos, dicho proceso puede ser optimizado empleando la siguiente cadena de adición: $C = [1; 2; 4; 6; 7; 13; 20; 33; 53]$ cuya longitud $Lc = 8$, puesto que es fácil notar que sólo para cifrar la letra "H" deben llevarse a cabo 53 multiplicaciones del exponente e igual cantidad de operaciones módulo. A fin de demostrar de forma práctica la resolución mediante el uso de la cadena C, se muestra el esquema sólo para cifrar la letra "H" del mensaje, no obstante para cada letra a cifrar se debe realizar el mismo proceso de forma iterativa.

El mecanismo consiste en utilizar productos de potencias de igual base, donde la base será igual a cada ocurrencia de carácter del mensaje y las potencias que se utilizan en cada paso son los elementos de la cadena de adición, expresadas como producto de potencias de igual base. A continuación se muestra, a modo de ejemplo, los pasos necesarios para cifrar el carácter "H" de la palabra "HOLA", donde cada elemento c_i es un término de la cadena de adición empleada.

c_1=1: $a^1 \rightarrow a^1 = 72$
c_2=2: $a^2 \rightarrow a \times a = 72 \times 72 \mod 143 = 36$
c_3=4: $a^4 \rightarrow a^2 \times a^2 = 36 \times 36 \mod 143 = 9$
c_4=6: $a^6 \rightarrow a^4 \times a^2 = 9 \times 36 \mod 143 = 38$
c_5=7: $a^7 \rightarrow a^6 \times a = 38 \times 72 \mod 143 = 19$
c_6=13: $a^{13} \rightarrow a^7 \times a^6 = 19 \times 38 \mod 143 = 7$
c_7=20: $a^{20} \rightarrow a^7 \times a^{13} = 7 \times 19 \mod 143 = 133$
c_8=33: $a^{33} \rightarrow a^{13} \times a^{20} = 7 \times 133 \mod 143 = 73$
c_9=53: $a^{53} \rightarrow a^{20} \times a^{33} = 133 \times 73 \mod 143 = 128 \equiv$ H

De esta manera con el empleo de la cadena C, tras 8 operaciones de potencia y 8 de módulo reducimos la complejidad de haber efectuado 53 de cada tipo, por cada carácter a cifrar dentro del texto. Cada paso del ejemplo anterior constituye un elemento de la cadena de adición C=[1; 2; 4; 6; 7; 13; 20; 33; 53], empleada para llevar a cabo el cifrado.

En este Capítulo se ha presentado el marco teórico de referencia de la tesis, introduciendo al lector en conceptos básicos de Criptografía con algunas reseñas históricas sobre la misma. Por otra parte se explica el concepto de cadenas de adición y las operaciones de la aritmética modular, los tipos de cadenas y aplicaciones de las mismas en el ámbito de la Criptografía.

Capítulo 3

Algoritmos Bioinspirados

En virtud de la existencia de diversas formas de resolver un problema de optimización combinatoria y la dificultad para abordar ciertos problemas con los métodos tradicionales, las ciencias de la computación, a través de la inteligencia artificial, han impulsado el estudio de diversas técnicas, sustentadas en el hecho de que en la naturaleza existen procesos que por sus características pueden ayudarnos a cumplir con el propósito de resolver problemas que de otro modo serían difíciles de tratar. De esta manera es que han surgido métodos, que por su similitud con procesos de la naturaleza, se han denominado bioinspirados [33].

Estos métodos constituyen una opción alternativa cuando los problemas de optimización plantean dificultad para ser resueltos, en función de:

- El espacio de búsqueda es demasiado grande. Es decir, que el número de posibles soluciones el alto.
- La complejidad del problema. Esto puede llevar a plantear modelos simplificados del mismo y por tanto ser poco útil una solución obtenida.
- Es demasiado complejo arribar a, al menos, una solución factible (que cumpla con las restricciones del problema), debido a que las posibles soluciones están demasiado restringidas.
- La función de evaluación que determina la calidad de cada posible solución fluctúa en el tiempo o tiene ruido.

3.1 Componentes de los algoritmos bioinspirados

El funcionamiento de un algoritmo bioinspirado se puede esquematizar de manera general mediante la secuencia de pasos del Algoritmo 3.

Si bien existen diferencias específicas en cada propuesta de algoritmos bioinspirados, también tienen componentes comunes, como la representación de las soluciones, la generación de un conjunto de soluciones iniciales (paso 1), mecanismos de selección (paso 3) y el uso de operadores de variación (paso 4).

Algoritmo 3 Algoritmo bioinspirado

1: Generar un conjunto de soluciones para el problema en cuestión.
2: Evaluar cada solución en función del objetivo a optimizar.
3: Seleccionar las mejores soluciones del conjunto (aquellas que alcanzaron mejor valor para la función objetivo).
4: Generar nuevas soluciones (a partir de aplicar operadores de variación sobre las mejores soluciones obtenidas en 3).
5: Evaluar las nuevas soluciones.
6: Seleccionar las nuevas mejores soluciones para la siguiente generación.
7: Iterar a partir de 2 hasta alcanzar un criterio de parada.

De acuerdo con los fenómenos naturales en los que se encuentran inspirados estos algoritmos se clasifican en dos grandes grupos: algoritmos o procesos evolutivos [32] y algoritmos de inteligencia colectiva [14]. A continuación se describen estos dos grupos de algoritmos.

3.2 Algoritmos evolutivos

Los Algoritmos Evolutivos (EAs, por sus siglas en inglés: Evolutionary Algorithms), son un conjunto de metaheurísticas basadas en principios de la teoría de evolución de las especies, factibles de aplicar a diversos problemas de optimización numérica y combinatoria para poder hallar soluciones subóptimas, dentro de un espacio de soluciones, en tiempo polinomial.
En los EAs se parte de un población (espacio de soluciones) para el problema a resolver. El proceso de búsqueda sobre este espacio de soluciones, es conducido por los operadores de variación (mutación y cruce) y selección, aplicados en cada una de las iteraciones, hasta cumplido un criterio de finalización de las ejecuciones. El objetivo de estos operadores es crear diversidad en la población, explorando nuevas zonas del espacio de búsqueda.

3.3 Algoritmos de inteligencia colectiva

Los algoritmos de inteligencia colectiva (SIAs, por sus siglas en inglés: Swarm Intelligence Algorithms) son aquellos que imitan el comportamiento de seres vivos que interactúan entre sí con la finalidad de resolver un problema complejo de manera conjunta. Los SIAs basan su funcionamiento en el comportamiento social de algunas

especies de animales, insectos, bacterias, entre otros. A continuación se describen algunos algoritmos bioinspirados que forman parte de este grupo.

3.3.1 Algoritmos basados en cúmulo de partículas (PSO)

PSO [3] es un proceso inteligente, estocástico, inspirado en el comportamiento social manifestado por las aves durante el vuelo. Este método ha sido empleado ampliamente en áreas de ingeniería y procesos de optimización de cómputo debido a su simplicidad de implementación, ya que se vale de una única búsqueda. PSO (Particle Swarm Optimization) se basa en una población de partículas (individuos) que se desplazan por el espacio de búsqueda multidimensional, sujetos a velocidades y aceleraciones. Estos cambios de posición reflejan la tendencia psico-social de los individuos, a intentar imitar el éxito de otros individuos (soluciones).

Cada partícula en el enjambre o cúmulo representa una posible solución al problema; y su posición varía de acuerdo a su propia experiencia y a la de sus partículas vecinas dentro del espacio. Cada partícula representa, en sí misma, una solución en el espacio multidimensional mediante 4 vectores con la siguiente información:

- Su posición actual.
- Mejor posición encontrada al momento.
- Mejor posición encontrada por su vecindario al momento.
- Su velocidad y ajuste de posición en el espacio de búsqueda, está basada en la mejor posición alcanzada por dicha partícula y en la mejor posición alcanzada por su vecindario, durante el proceso de búsqueda.

Algoritmo 4 Algoritmo PSO

```
1: Generar el cúmulo e inicializarlo mediante la asignación de una posición aleatoria
   en el espacio del problema a cada partícula.
2: Evaluar cada partícula del cúmulo inicial mediante una función de aptitud.
3: mientras no se cumpla criterio de parada hacer
4:     Comparar aptitud de cada partícula del cúmulo y seleccionar líder.
5:     si la aptitud de la partícula actual es mejor que la aptitud del líder, entonces
6:         Establecer partícula actual como líder y actualizar su posición.
7:         Evaluar valor de aptitud de cada partícula.
8:     fin si
9: fin mientras
```

El Algoritmo 4 describe como en cada iteración, cada partícula actualiza su posición y la velocidad. El criterio de parada del método puede ser un número máximo de iteraciones o un valor suficientemente bueno de adaptación (fitness).

3.3.2 Algoritmos basados en colonias de hormigas (ACO)

Los algoritmos ACO (por sus siglas en inglés: Ant Colony Optimization) descriptos en [3], son metaheurísticas inspiradas en el comportamiento colectivo de las colonias de hormigas y en un fenómeno denominado *Estigmergia* [1] que básicamente se caracterizada por el mecanismo de depositar una señal o rastro de feromonas en el terreno, como un sistema de auto-organización para propiciar la comunicación entre seres vivos, modificando características del entorno local. El aspecto más interesante de este mecanismo colaborativo es la capacidad de encontrar caminos más cortos, entre la fuente de alimento y los hormigueros, y la posibilidad de comunicarlos a otros miembros de la colonia mediante senderos trazados con rastro de feromonas que van depositando las hormigas durante el trayecto entre estos puntos. Estos senderos en la medida que otras hormigas los transitan y también depositan su rastro de feromona se intensifican debido a la concentración de feromona, generando que otros individuos opten por la elección de dicha ruta para llegar desde la fuente de comida al hormiguero o viceversa.
Desde su formulación estos algoritmos fueron empleados para modelar y abordar con éxito problemas de optimización combinatoria complejos.

Un algoritmo ACO posee tres etapas o procedimientos fundamentes [3].

- Construcción de soluciones (hormigas).
  Función generadora de las soluciones (hormigas artificiales) que conformarán el espacio de búsqueda del problema. Dichas hormigas se movilizarán de acuerdo con una regla de transmisión, la cual se puede representar por ejemplo mediante un grafo dirigido. Cada hormiga se moviliza tomando decisiones basadas en criterios estocásticos, usando para tal fin la información de los rastros de feromona e información heurística. Este proceso se realiza de forma iterativa.

- Actualización de feromona.
  Los rastros de feromona para las soluciones construidas, son actualizados, posterior a cada iteración del método, mediante una función. En relación a este proceso pueden darse dos situaciones: que el rastro se incremente porque las hormigas depositan feromona en cada componente o conexión que usan para moverse dentro del espacio de búsqueda, reforzando dicha conexión; o bien que el mismo se evapore para una conexión por tener pocas "visitas" por dicho punto, lo que favorece que otras hormigas lo ignoren. De esta manera se van perdiendo las "malas soluciones".

- Ejecución de acciones.
  Este proceso es opcional y no posee un equivalente en la naturaleza porque no pueden ser realizados por cada individuo (hormiga). Esto puede incluir la aplicación de refuerzo del rastro de feromonas a la mejor solución generada. Pueden incluirse estrategias para mejorar el rendimiento del método como por ejemplo una actualización del rastro de feromona diferente, introduciendo cambios localizados en el espacio de búsqueda para favorecer la exploración.

La estructura general de un ACO se puede representar de manera general mediante el Algoritmo 5.

Algoritmo 5 Algoritmo ACO

1: Inicializar valores de feromona.
2: Activar hormigas que iniciarán proceso de búsqueda.
3: **mientras** no se cumpla criterio de parada **hacer**
4: Ejecutar proceso de búsqueda con hormigas activas.
5: Por cada hormiga que detenga su búsqueda, actualizar memoria compartida.
6: Actualizar rastro de feromona global (memoria compartida).
7: Reubicar hormigas activas.
8: **fin mientras**

3.3.3 Algoritmos basados en bacterias (BFOA)

En [20] fue propuesto un método denominado "Biomimicry of Social Foraging Bacteria for Distributed Optimization" (Biomimetismo de forrajeo bacteriano para la optimización distribuida, BFOA por sus siglas en inglés) este enfoque simula el proceso completo de forrajeo de las bacterias, dicho de otro modo, el comportamiento de una colonia de bacterias en busca de una concentración óptima de nutrientes.

El proceso de búsqueda de alimento que efectúan las bacterias en BFOA se puede representar mediante el Algoritmo 6. Dicho proceso inicia distribuyendo las bacterias aleatoriamente sobre el mapa de nutrientes. Luego en una primera etapa se mueven las bacterias hacia un punto de concentración de nutrientes. Dentro de esta etapa tendrán lugar diferentes eventos, dispersión de bacterias, eliminación de otras por ubicarse en regiones que poseen sustancias nocivas, reproducción y agrupamiento de bacterias que se ubicaron en puntos de mayor concentración de nutrientes [23].

Algoritmo 6 Algoritmo BFOA

1: Inicializar, distribuyendo las bacterias de forma aleatoria sobre un mapa de nutrientes.
2: Mover las bacterias en busca de puntos de concentraciones de nutrientes (en este punto se produce la reproducción y algunas bacterias se acercan a sustancias nocivas y son eliminadas).
3: Comunicación entre las bacterias que han encontrado concentración de nutrientes.
4: Agrupar las bacterias que han encontrado concentraciones de nutrientes.
5: Iterar para dar lugar a un nuevo ciclo. Las bacterias se dispersan nuevamente en busca de otra concentración de nutrientes, hasta alcanzar criterio de finalización.

Durante este proceso de búsqueda de alimento muchas de las bacterias afrontan situaciones problemáticas, como por ejemplo encontrarse con sustancias nocivas que pueden causar su eliminación o dispersión y que deberán sortear mediante algún mecanismo de defensa como la segregación de sustancias químicas empleada como mecanismo de comunicación.

Debido a que la implementación de BFOA, requiere de muchos parámetros que se deben ajustar y al alto costo computacional que plantea, se ideó otro método que consiste en una modificación del modelo, denominado MBFOA (BFOA modificado) [23].

En esta variante la representación de las soluciones es igual que en BFOA. El algoritmo inicia generando un cúmulo de bacterias de manera aleatoria (población inicial). En un segundo paso se evalúa la población generada inicialmente. En cada generación tendrá lugar un proceso llamado *Quimiotaxismo*, se trata de un fenómeno por el cual bacterias y otros organismos celulares dirigen sus movimientos mediante un mecanismo de comunicación arbitrado por la concentración de ciertas sustancias químicas, en su medio ambiente [5]. Además se ejecuta dentro de este ciclo el proceso de reproducción y eliminación o dispersión de bacterias [23]. El Algoritmo 7 esquematiza dicho mecanismo.

Algoritmo 7 Algoritmo MBFOA

```
1: Inicializar, generando cúmulo inicial (aleatorio) de bacterias.
2: Evaluar población de bacterias
3: mientras No se alcance condición de fin, hacer
4:     mientras No se alcance condición de fin, hacer
5:         Ejecutar proceso de quimiotaxismo.
6:     fin mientras
7:     Ejecutar proceso de reproducción.
8:     Eliminar las bacterias no aptas de la población.
9:     Generar nuevas bacterias de manera aleatoria.
10: fin mientras
```

3.3.4 Algoritmos basados en enjambres de abejas (ABC)

Los algoritmos basados en enjambres de abejas (Artificial Bee Colony o ABC, por sus siglas en inglés) [15], son métodos inspirados en el proceso de búsqueda de alimento por parte de las abejas.

El modelo conceptual del algoritmo ABC, plantea que las colonias de abejas contienen tres grupos de abejas artificiales: recolectoras productivas (en adelante, *obreras*), y dos tipos de recolectoras no productivas o desempleadas (*observadoras* y *exploradoras*). A continuación se describe cada uno de los tipos.

- Obreras: se trata de aquellas abejas que se encuentran explotando una fuente particular de alimento, es decir, están empleadas con la misma. Las abejas de este tipo comparten con otros tipos de abejas, como las *observadoras*, la información sobre su fuente de alimento, por ejemplo: la ubicación de la misma respecto de la colmena, la concentración de alimento y la facilidad para extraerlo.
- Observadoras: se denomina así al tipo de abejas que se encuentran esperando en la colmena, aguardando por una fuente de alimento que elegirá con base en la información compartida por parte de una *obrera*.
- Exploradoras: grupo de abejas que llevan a cabo una búsqueda aleatoria de nuevas fuentes de alimento en las cercarnías de la colmena.

En los ABCs, usualmente la mitad de la dotación de la colonia consiste de abejas artificiales *obreras* y la segunda mitad constituyen las *exploradoras* y *observadoras*. Para cada fuente de alimento, sólo hay una abeja *obrera*, por ende, el número de abejas de este tipo, es igual al número de fuentes de alimentos alrededor de la colmena. La abeja *obrera* cuya fuente de alimento se agota se convertirá en una abeja *exploradora* en busca de una nueva fuente u *observadora*, debiendo tomar la decisión respecto de que rol desempeñará.

Los pasos contenidos en cada ciclo del ABC (pasos 4 a 10 del Algoritmo 8) pueden resumirse en las siguientes acciones:

- Ubicar las abejas *obreras* en las fuentes de alimento.
- Ubicar las abejas *observadoras* en las fuentes de alimento.
- Enviar las abejas *exploradoras* a la zona de búsqueda, para descubrir nuevas fuentes de alimento.

La primer acción consiste en el envío de la abejas *obreras* a las fuentes de alimentos y posteriormente medir las cantidades de néctar recolectado. Luego son seleccionadas las fuentes de alimentos por parte de las abejas *observadoras*, después de compartir la información de las abejas *obreras* y a partir de ésta determinar la cantidad de néctar de las fuentes.

En la fase de inicialización del método, se selecciona al azar un conjunto de posiciones de fuentes de alimento por medio de las abejas. Entonces, estas abejas entran en la colmena y la información sobre el néctar de las fuentes y ubicación es transmitida a las abejas que esperaban en la zona de baile dentro de la colmena. Dicho baile radica en un movimiento que realizan las abejas ya sea cuando se reproducen o van en búsqueda de alimento. Debido a este tipo de movimientos, los individuos de la población (abejas) recorren el espacio de búsqueda de tal forma que encuentran soluciones subóptimas. En la segunda etapa, luego de compartir la información, cada

Algoritmo 8 Algoritmo ABC

1: Inicializar población de fuentes de alimento (soluciones).
2: Evaluar desempeño de la población.
3: **mientras** No se alcance condición de fin, **hacer**
4: Generar nuevas soluciones para las abejas empleadas.
5: Evaluar soluciones en base a la función objetivo a optimizar y conservar las mejores.
6: Seleccionar solución que será visitada por una abeja observadora en base a su aptitud.
7: Generar nuevas soluciones para las abejas observadoras.
8: Evaluar y conservar las mejores soluciones.
9: Determinar si existen fuentes abandonadas (agotadas) y reemplazarlas (por medio de una exploradora).
10: Memorizar la mejor solución encontrada hasta el momento.
11: **fin mientras**

abeja *obrera* se dirige a la zona (fuente de alimento) visitada por sí misma en el ciclo anterior, ya que existe una fuente de alimento almacenada en su memoria, y a continuación, elige una nueva fuente de alimento por medio de la información visual, en el "barrio" actual. En la tercera etapa, una *exploradora* buscará una fuente de alimento con base en la información comunicada por las abejas *obreras*, en la zona de baile.

3.3.5 Algoritmo basado en comportamiento de lobos grises (GWO)

GWO (por sus siglas en inglés: Grey Wolf Optimizer) [24] es una metaheurística poblacional, basada en inteligencia colectiva, inspirada en el comportamiento social de las manadas de lobos grises (Canis Lupus) y su organización para la actividad de caza de presas. GWO plantea que una manada de lobos grises se dividen en tres tipos de individuos: α (líder o líderes), β y δ (subordinados de los anteriores y que le asisten en la toma de decisiones) y resto de los individuos son ω.

El proceso de caza de los lobos grises y su organización social para esta actividad, ha sido modelado metamáticamente para la resolución de problemas complejos, dando lugar a una metaheurística. Dicho proceso de optimización consta de las siguientes acciones [24]:

- Seguir, perseguir, y acercarse a la presa.
- Rodear, y acosar a la presa hasta que se "paraliza".
- Atacar a la presa.

El algoritmo GWO será explicado en detalle en el Capítulo 5, pues constituye el principal objeto de estudio de la propuesta de esta tesis que es adaptar dicha metaheurística para tratamiento del problema de generación óptima de cadenas de adición.

Capítulo 4

Trabajos relacionados

En esta sección se exponen algunos de los métodos existentes, que son de interés en el desarrollo de esta tesis y representan el estado del arte para generar cadenas de adición de longitud mínima. Como se ha mencionado el sustento de las mismas es la posibilidad de disminuir el número de operaciones (multiplicaciones y divisiones) que plantea la *exponenciación modular* empleada en los algoritmos criptográficos asimétricos. Tales métodos pueden clasificarse de manera general en: determinísticos y estocásticos. Dentro del grupo de los determinísticos se describe la *estrategia binaria*, *estrategia de ventana* y *ventana adaptativa* o deslizante y *método factor*. Dentro del grupo de los estocásticos se enumeran y describen las metaheurísticas que resultaron de interés dentro de estado del arte como antecedentes de tratamiento del problema a tratar y cuyos resultados serán empleados para comparar los obtenidos por las propuestas de la tesis.

4.1 Métodos determinísticos

Los métodos determinísticos, son algoritmos que se caracterizan por implementar un procedimiento o conjunto de acciones y obtener la misma salida, a partir de un parámetro o conjunto de parámetros de entrada, independientemente de las veces que sea ejecutado.

4.1.1 Estrategia binaria

Dado un exponente e, este método consiste en considerar al mismo en una representación binaria, de la forma $e = (e_{m-1} \ ... \ e_1 \ e_0)$, siendo: $(e_{m-1} \ ... \ e_1 \ e_0)$, las componentes binarias (bits) de la cadena asociada a e. Una vez transformado e en una cadena binaria, la misma se escanea de izquierda a derecha o en sentido contrario, aplicando la *regla de Horner* de forma iterativa en base al siguiente criterio

[6]:

- Por cada bit de la secuencia escaneado, se multiplica por dos al último número de la cadena de adición.
- Si el bit escaneado resulta ser uno, se le suma uno al último valor de la cadena de adición.

Este método requiere un total de $m - 1$ iteraciones, considerando que la representación binaria de e posee una longitud de m-bits. Este mecanismo se puede representar mediante el Algoritmo 9.

Algoritmo 9 Estrategia binaria

Entrada: e=(e_{m-1} ... e_1 e_0)
Salida: C = [1;...;e]
1: $x = 1$ (Se inicializa secuencia en 1).
2: Colocar x en la primer posición de C.
3: **para** i: [(m-2) .. e_0] decreciendo, **hacer**
4: $x = 2 \times x$.
5: Agregar x al término de la cadena C.
6: **si** $i = 1$, **entonces**
7: $x = x + 1$.
8: Agregar x al término de la cadena C.
9: **fin si**
10: **fin para**
11: **devolver** C

Donde:

e: exponente dado. Número entero representado por una cadena binaria.
C: cadena de adición asociada a e.

Por ejemplo, dado $e = 177$, para aplicar este método tomaremos el exponente en binario, siendo entonces $(10110001)_2$. Una vez aplicado el Algoritmo 9 se obtiene la siguiente cadena de adición C = [1; 2; 4; 5; 10; 11; 22; 44; 88; 176; 177].

4.1.2 Estrategia de la ventana

Al igual que el método anterior; éste opera sobre una representación binaria del exponente e, como una secuencia de bits de la forma $e = (e_{m-1}$ ... e_1 $e_0)$. En este caso los bits de la representación binaria son particionados en fragmentos de k-longitud

denominados cadenas o ventanas, utilizaremos W para referirnos a dichas ventanas. En este método la ventana tendrá una longitud máxima la cual puede definirse de acuerdo con el contexto de aplicación y sobre esto existen estudios particulares como [4], donde se describe el método con un tamaño de ventana de 5 bits.

A continuación describimos de manera más formal el método planteado en [6]. Sea el valor k un pequeño divisor. Entonces, esta representación binaria del exponente e puede ser dividida en ventanas o cadenas de ϕ longitud, de tal manera que ($k \times \phi$ - m) si no divide de manera exacta, entonces el exponente e expresado de forma binaria, deberá ser completado con k-1 ceros.
El método o estrategia de la ventana pre-computa primero los valores de x^j para j = 1,2,3 .... 2^k - 1. Entonces, el exponente e se escanea k bits a la vez de la ventana más significativa ($W_{\phi-1}$) a la ventana o cadena menos significativa (W_0). En cada iteración, el resultado parcial actual es elevado a la potencia 2^k y multiplicado con X_{W_i}, donde W_i, es la ventana no nula actual que se está procesando en ese momento. Se puede observar que:

- El primer paso del algoritmo consiste en pre-computar las primeras 2^k potencias de X. Esta operación implica un costo computacional de procesamiento de multiplicaciones de 2^k-2.
- En cada iteración del bucle principal, Y^{2k} puede ser calculada mediante la ejecución de sucesivas operaciones de potencia, siendo $(\phi - 1)k - m - k$ el número total de operaciones de potencia.
- En cada iteración, se realiza una multiplicación cada vez que la i-ésima ventana W es diferente de cero. Todos menos uno de los 2^k diferentes valores de W_i es distinto de cero. El número promedio de multiplicaciones requeridas en este sentido está dado por: (ϕ -1) $(1 - 2^{-k}) - [(m/k)$ - 1)] $(1 - 2^{-k})$.

El Algoritmo 10 muestra de manera detallada los pasos previamente descriptos de la estrategia de ventana.

Algoritmo 10 Estrategia de la ventana

Entrada: $e = (e_{m-1} \ ... \ e_1 \ e_0)$, $\phi = m/k$
Salida: C = [1;...;e]

1: Generar y almacenar (pre-computar) x^j = 1,2,3,4,...,2^k - 1.
2: Dividir e en ventanas de k bits $\rightarrow W_i$ con i = 1, 2, 3,... ϕ -1.
3: $Y = X^{W^{\phi-1}}$ // Agregar este resultado al final de la cadena C.
4: **para** i: [(ϕ-2) .. e_0] decreciendo, **hacer**
5: $Y = Y^{2k}$ // (es decir, $x = 2 \times x$).
6: Agregar x al término de la cadena C.
7: **si** $W_i \neq 0$, **entonces**
8: (es decir, que es el último bit de la ventana actual)
9: $Y = Y \times X^{Wi}$.
10: // ubicar como último valor de la cadena el valor en decimal de la ventana actual.
11: Agregar x al término de la cadena C.
12: **fin si**
13: **fin para**
14: Ordenar (x^j unión C).
15: **devolver** C

Donde:

e: exponente dado. Número entero representado como una cadena binaria.
C: cadena de adición asociada a e.

Por ejemplo, dado $e = 177$, utilizado para demostrar la estrategia binaria, tomaremos el exponente en binario siendo entonces $(10110001)_2$. Con $m = 8$ y $k = 3$, con lo cual para dividir la cadena en 3-bits será necesario completar con un cero la cadena, obteniéndose $(\underline{010}\ \underline{110}\ \underline{001})_2$.
Una vez aplicado el Algoritmo 10 se obtiene:

x^j = 1, 2, 3, 5, 7. La última posición será 7 puesto que resulta de 2^k – 1.

C = [2; 4; 8; 16; 22; 44; 88; 176; 177] (secuencia intermedia)

Una vez obtenido lo anterior, resta calcular la operación (x^j unión C) cuyo resultado es la siguiente secuencia final:

C=[1; 2; 3; 4; 5; 7; 8; 16; 22; 44; 88; 176; 177] (cadena de adición para e=177)

4.1.3 Estrategia de la ventana adaptativa

Este método es una variante del anterior y constituye una alternativa para abordar exponenciaciones con exponentes extremadamente grandes (aquellos cuya longitud

supera los 128-bits) debido a su capacidad para ajustar su método de cálculo de acuerdo con el exponente a trabajar. Dicho ajuste se realiza mediante la división del exponente e de entrada, en una serie de cadenas de longitud variable (no nulas) llamada ventanas, tal como se encuentra planteado en [6].

A diferencia de la estrategia de ventana tradicional, el algoritmo que implementa esta estrategia, suministra una solución compromiso de rendimiento, puesto que permite el procesamiento de cadenas (ventanas) de longitud variable, mediante una técnica que consiste en generar ventanas con la particularidad de que habrá algunas conformadas por ceros y otras por elementos distintos de cero (teniendo en cuenta la representación binaria del exponente e).

Este método consta de las siguientes fases:

- La primera corresponde a la generación de cadenas (ventanas) de elementos ceros y otras de elementos distintos de cero, a partir de la representación binaria del exponente, mediante una estrategia de partición.
- A las ventanas de elementos ceros (que en adelante denominamos VC), se les permite tener una longitud arbitraria. Sin embargo, para las ventanas de elementos distintos de cero (en adelante llamadas VNC), su longitud máxima no deberá exceder los k bits.
- Después de haber realizado la fase de particionamiento, la siguiente tarea consiste en calcular la secuencia de adición necesaria, para obtener todas las ventanas de elementos no nulos (VNC).
- El paso posterior consiste en inicializar la cadena a devolver por el método, mediante el resultado de $Y = X^{W\phi-1}$. Nótese que se supone siempre que $W^{\phi-1}$ es distinto de 0.
- En cada iteración del bucle principal, se puede computar Y^{2k} mediante la realización de 2^k multiplicaciones consecutivas.
- En cada iteración, se realiza una multiplicación cada vez que la i-ésima ventana de W^i es un elemento distinto de cero. Teniendo en cuenta que VNC representa el número de ventanas de elementos diferentes de cero, el número de multiplicaciones necesarias en este paso del algoritmo será de VNC – 1.
- El valor exacto de VNCs dependerá de la estrategia de partición instrumentada.

Estos pasos se pueden esquematizar mediante el pseudocódigo del Algoritmo 11 y en efecto es muy similar al Algoritmo 10. La diferencia radica en la generación de los dos tipos de ventanas, tal como se expone en [6].

Algoritmo 11 Estrategia de ventana adaptativa

Entrada: e=(e_{m-1} ... e_1 e_0), $\phi = m/k$
Salida: C = [1;...;e]

1: Generar y almacenar (pre-computar) x^j = 1,2,3,4,...,ϕ - 1. // Se generan las ventanas VC y VNC.
2: $Y = X^{W^{\phi-1}}$ // Agregar este resultado al final de la cadena C.
3: **para** i: [(ϕ-2) .. e_0] decreciendo, **hacer**
4: $Y = Y^{2k}$ // (es decir, $x = 2 \times x$).
5: Agregar x al término de la cadena C.
6: **si** $W_i \neq 0$, **entonces**
7: (es decir, que es el último bit de la ventana actual)
8: $Y = Y \times X^{W_i}$.
9: // ubicar como último valor de la cadena el valor en decimal de la ventana actual.
10: Agregar x al término de la cadena C.
11: **fin si**
12: **fin para**
13: **devolver** C

Donde:

e: exponente dado. Número entero representado como una cadena binaria.
C: cadena de adición asociada a e.

4.1.4 Método factor

Este método descripto en [16] consiste en factorizar un exponente e, mediante la operación $e = r \times s$, donde r es un factor primo más pequeño de e y s es un número menor que 1. Esta factorización consiste en generar un árbol cuyo nodo raíz es el exponente e y cuyos n nodos lo conforman r y s, tal que $e = r \times s$. El proceso se repite de forma iterativa para cada nodo hijo, hasta que cada nodo no puede continuar siendo factorizado, es decir, hasta que se convierte en 1.

Desafortunadamente este método puede llegar a ser dificultoso para grandes exponentes, debido a las operaciones de factorización necesarias para componer el árbol y luego tener que obtener la cadena de adición asociada al exponente. A continuación vemos en la Figura 4.1 un ejemplo de un árbol de factores generados para un exponente $e = 77$.

Donde:
n: número entero, equivalente al valor del exponente e en cuestión para el que se desea obtener una cadena de adición.
r y s: factores.

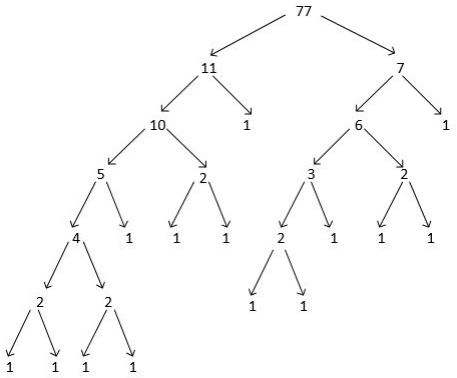

Figura 4.1: Árbol de factorización.

Algoritmo 12 Método factor

Entrada: n

1: Asignar n como nodo raíz del árbol.
2: **si** $n \neq 1$, **entonces**
3: Tomar r y s de modo que $n = r \times s$.
4: **fin si**
5: **si** $r = n$ y $s = 1$, **entonces**
6: $s = (n - 1)$; $r = 1$
7: Agregar r como nodo hijo de la derecha de n y s como nodo hijo a la izquierda de n.
8: **si** $r > 1$, **entonces**
9: Llamar a la construcción del árbol (con nodo raíz en r).
10: **fin si**
11: **si** $s > 1$, **entonces**
12: Llamar a construcción del árbol (con nodo raíz en s).
13: **fin si**
14: **fin si**

El Algoritmo 12, presenta el esquema del proceso de construcción del árbol de factorización como un método recursivo. Una vez que se ha generado el árbol para un exponente e, mediante el Algoritmo 12 deberá calcularse la cadena de adición correspondiente. Cabe mencionar que este método puede resultar ser poco eficiente, incluso para exponentes pequeños (por debajo de los 128-bits), en función del procesamiento que conlleva la factorización para la construcción del árbol.

4.2 Métodos estocásticos

Se sabe que la generación de cadenas de adición óptimas (de longitud mínima) constituye un problema NP-completo, teniendo en cuenta que para encontrar una cadena óptima de longitud r, se considera un espacio de búsqueda asociado a dicho

problema, comparable al resultado de la operación $r!$ (factorial de r). En virtud de lo anterior se formularon diversos métodos estocásticos para afrontar la complejidad del problema, a continuación se describen, en ésta sección, las que resultan de interés para la tesis.

4.2.1 Uso de un algoritmo genético para generar cadenas de adición

En [7] se propuso el uso de un algoritmo genético cuya esquematización se puede representar de la siguiente manera:

Algoritmo 13 Algoritmo genético

Entrada: e=(e_{m-1} ... e_1 e_0), N= tamaño de la población
Salida: C = [1;...;e]

1: Generar de manera aleatoria una población inicial de tamaño N.
2: **mientras** (no se alcance el número máximo de generaciones), **hacer**
3: Calcular aptitud (fitness) de cada individuo.
4: Seleccionar N padres para cruzarse // de los individuos del paso 3.
5: Aplicar cruce a los individuos del punto 4 con probabilidad p_c. // Para obtener hijos.
6: Mutar con probabilidad p_m los hijos obtenidos en paso 5.
7: Reemplazar los padres por los hijos del paso anterior para nueva generación.
8: **fin mientras**
9: Calcular aptitudes (fitness) de los individuos de cada iteración.
10: C = individuo de mejor aptitud.
11: **devolver** C.

Donde:

e: exponente entero.
N: tamaño máximo de población.
p_c: probabilidad de cruce.
p_m: probabilidad de mutación.
C: cadena de adición asociada a e.

El Algoritmo 13 muestra una adaptación del algoritmo genético o GA (por sus siglas en inglés), donde se emplean los operadores usuales de mutación y cruce y donde la aptitud está determinada por la longitud de la cadena de adición.
Este enfoque fue verificado en [7] para exponentes e del rango de longitudes de [1, 1000] y para exponentes especiales cuyas cadenas de adición óptima, son particularmente difíciles de hallar. Los resultados obtenidos por este método fueron comparados con varios algoritmos deterministas superando los resultados de los mismos.

En dicho trabajo, se adoptó una codificación de enteros para el GA, con cromosomas de longitud variable que son el equivalente a las cadenas de adición buscadas. Cada elemento de la cadena es asignado directamente en cada gen o elemento del cromosoma, en función de esto, se trata de un GA donde el genotipo y el fenotipo son iguales.
De este modo, si lo que se busca es minimizar la cadena de adición para un exponente $e = 5288$, una solución candidata podría ser: [1; 2; 4; 8; 16; 32; 64; 128; 256; 512; 1024; 2048; 4096; 5120; 5248; 5280; 5288], como cadena de adición óptima, la cual representa un cromosoma de longitud 16, teniendo en cuenta que este método permita el "doblado" de números, es decir, que algunos genes del cromosoma puedan ser generados por la suma de otro elemento consigo mismo, para dar origen a un nuevo elemento de la secuencia.

Las pruebas utilizando GA en la propuesta [7], se realizaron empleando la siguiente combinación de parámetros:

Parámetro	Valor
Tamaño de población	100
Número de generaciones	300
Probabilidad de cruce	0.6
Probabilidad de mutación	0.5 (con operador de mutación de un punto)
Operador de selección	torneo binario

Los resultados fueron obtenidos a partir de 30 ejecuciones independientes del algoritmo, utilizando diferentes valores de e independientes con semillas aleatorias.

4.2.2 Algoritmo de recocido genético

En [27] se plantea el problema de optimización de cadenas de adición con un algoritmo denominado "Genetic Annealing Optimization" ("recocido genético"), es decir, una hibridación que resulta de combinar el GA y el recocido simulado ("Simulated Annealing") [18]. El fundamento de esta estrategia combinada radica en el hecho de que el recocido simulado puede ayudar al GA a escapar de los óptimos locales y esta sería la principal ventaja que mostró la propuesta [27] respecto del antecedente de abordaje del problema con GA tradicional.

"Genetic Annealing Optimization" pretende brindar lo mejor de las propuestas que combina. Su funcionamiento es análogo al de GA puesto que crea nuevas soluciones a partir del intercambio de material genético entre los miembros de una población. A diferencia de GA que emplea selección por torneo para decidir qué individuos pasarán a la siguiente generación, aplica un criterio adaptativo propio del algoritmo

de recocido, que se basa en un esquema simple de retroalimentación. El Algoritmo 14 esquematiza la propuesta de [27].

Algoritmo 14 Recocido genético

```
1: Inicializar energyBank.
2: para cada individuo, hacer
3:    mutante = mutante (individuo).
4:    si fitness(mutante) < fitness(individuo) + umbral(individuo), entonces
5:       diff = fitness(individuo) + Umbral(individuo) - fitness(mutante)
6:       energyBank += diff
7:       Reemplazar individuo con mutante.
8:    fin si
9:    energyDiff = energyBank × C / N
10:   para cada individuo, hacer
11:      Umbral (individuo) += energyDiff
12:   fin para
13: fin para
```

A continuación se resume el funcionamiento del Algoritmo 14:

- Se asigna un umbral de energía a cada individuo.
- Inicialmente cada umbral será igual a la energía del individuo a la que es asignado. Este proceso puede traducirse como la mutación de los individuos.
- Si la energía de un mutante, supera al umbral que posee el individuo que generó dicha energía, el mutante es rechazado y el algoritmo se desplaza al siguiente individuo, por el contrario si resulta menor o igual, el algoritmo acepta el individuo como reemplazo de aquel que le dio origen ("individuo padre"). Como se puede observar el proceso de recocido de esta variante de algoritmo genético es guiado por la aptitud (fitness) de los individuos.
- Se emplea un banco de energía "energyBank", el cual es inicializado en el paso 1 del Algoritmo 14, y cuya finalidad es seguir los cambios de la energía producto de la liberación de la misma, cuando los individuos que resultan exitosos son aceptados por el proceso.
- Por cada individuo mutado que supera el umbral, se añade la diferencia ("Diff", ver Algortimo 14) entre el umbral y la energía del mutante, al banco de energía, para almacenamiento temporal.
- Luego de lo anterior, se restablece el umbral de manera que resulta igual a la energía del mutante aceptado, para dar lugar al algoritmo a que tome el siguiente individuo.

Una vez que la población de N individuos ha sido mutada de manera aleatoria, tiene lugar el "recalentamiento" como se expone en [27], elevando los umbrales y dicha variación será determinada en función de la energía acumulada en el banco de energía y de la velocidad a que se desea "enfriar" la población.

El recocido genético controla la velocidad de enfriamiento, mediante una configuración constante de C (número real entre 0 y 1). La importancia de controlar este evento radica en que es necesario enfriar los individuos lentamente para alcanzar el estado óptimo de dichas soluciones.

4.2.3 Otra variante de uso de GA con nuevos operadores

En el reciente trabajo [26], se plantea el uso de un AG que emplea una representación novedosa para las soluciones y nuevos operadores de cruce y mutación para lograr cadenas de adición de longitud mínima, para un exponente dado. Por otro lado, también implementa una estrategia de reparación de las soluciones a fin de mejorar el desempeño del enfoque propuesto por los autores.

Características del algoritmo propuesto en [26]:

- En esta propuesta se trabaja con una codificación de soluciones de manera tal que cada elemento de la cadena es un número entero pero a nivel interno, cada elemento está conformado por un par de dos posiciones (i_1, i_2) que retienen valores para n_1 y n_2, conformando un nuevo valor como la suma de las posiciones. Debido a que tenemos dos posiciones por cada elemento de la cadena, la longitud de la misma, codificada con posiciones será siempre dos veces más que la codificada con valores simples, es decir tomando en consideración una representación que no calcule cada componentes mediante una suma de otros dos números.

- A pesar de que la codificación propuesta genera cromosomas (individuos) de mayor longitud, los valores codificados internamente son mucho más pequeños y logran optimizarse los requisitos de memoria, para el almacenamiento de cada individuo.

- El uso de esta codificación particular implica que es necesario contar con un mecanismo alternativo (personalizado) tanto para la inicialización como para los operadores de cruce y mutación del algoritmo, solo escapando de esta necesidad el operador de selección con respecto a los GA tradicionales.

- No se brinda en [26] el nombre de los operadores propuestos pero se describen sus características. Se proponen 2 operadores de cruce sugiriendo el uso de uno u otro en función del exponente que se pretende tratar en cada caso (cruce de un punto y de dos puntos), ambos con sus propias reglas de construcción de soluciones. El operador de mutación es similar a los presentados en el literatura

relacionada, pero buscando mayor diversidad en el proceso de generación. Ambos tipos de operadores incluirán la repación dentro de su secuencia de pasos.

- En relación al operador de selección, en las pruebas del trabajo de [26] se utilizó la selección por torneos de k individuos con $k = 3$, de modo tal que en cada torneo, el peor de k tomados al azar se sustituye por la descendencia de los otros mejores dos del mismo torneo.
- Por otro lado el mecanismo de reparación de soluciones que plantea este método es necesario en función de que se espera que tanto los operadores de variación (cruce y mutación) como el de selección, produzcan muchas soluciones no válidas, tal como lo explicaron sus autores en dicha propuesta.

A continuación se describen los operadores utilizados por esta variante de GA:

Proceso de inicialización de individuos:

1. Establecer como primer y segundo elemento los número 1 y 2, respectivamente.
2. De manera al azar pero uniforme, elegir entre todas las sub cadenas mínimas formadas por tres elementos (es decir, la segunda, tercera y cuarta posición en la cadena) y una elección al azar del segundo elemento tomando valor 3 o 4.
3. Con probabilidad igual a 3/5, tomar el doble de los elementos hasta alcanzar la mitad del tamaño exponente (longitud de la cadena).
4. Comprobar en todo momento que el elemento actual y cualquier elemento anterior resumen el valor del exponente asociado a la cadena.
5. De manera al azar pero uniforme, generar el próximo valor de la cadena, aplicando uno de los siguientes mecanismos:
   - Suma de dos elementos anteriores.
   - Suma del elemento anterior y otro aleatorio de la cadena.
   - Suma de dos elementos aleatorios, considerando el siguiente criterio: uno de los elementos será elegido al azar entre la primera posición y el elemento medio de la cadena. El segundo será elegido entre el elemento medio y el valor final de la cadena (exponente).
   - Búsqueda, enlazando desde el elemento cuya posición es (i-1) hasta localizar el elemento más grande que se puede resumir con el último elemento hallado.

Operadores de variación utilizados

Tanto el operador de cruce como el de mutación son similares a los utilizados por otras propuestas. El operador de mutación empleado es análogo a la mutación de un solo punto y este enfoque implementa también dos operadores de cruce, en aras de favorecer la diversidad. Puesto que en [26] no se brindan los nombres dados a los operadores de variación propuestos, se esquematizan los mismos a continuación (ver Algoritmos 15 y 16.

Algoritmo 15 Operador de cruce

Entrada: $exp > 0$; $P1$ y $P2$
$rand$ = Random (3, $exp - 1$)
para todo i tal que: $0 \leq i \leq rand$, **hacer**
  $e_i = P1_i$
fin para
para todo i tal que: $rand \leq i + 1 \leq n$, **hacer**
  FindLowestPair ($P2$, i, pair1, pair2)
  $e_i = e_{pair1} + e_{pair2}$
fin para
ReparарCadena (e, exp)
devolver $e = e_0, e_1,...,e_n$

Dónde:

exp:	exponente a optimizar.
e:	cadena de adición asociada a exp.
$P1$, $P2$:	cadenas de adición ("padres").
ReparарCadena:	función de reparación de cadenas (ver Algoritmo 17).
FindLowestPair:	función que determina par de elementos, con índices (posiciones) más bajos.

Algoritmo 16 Operador de mutación

Entrada: $exp > 0$; $e = e_0, e_1,...,e_n$

$rand$ = Random (2, $exp - 1$)

$rand2$ = Random (0,1)

1: **si** $rand2$, **entonces**

2: $e_{rand} = e_{rand-1} + e_{rand-2}$

3: **si no**

4: $rand3$ = Random (2, $rand - 1$)

5: $e_{rand} = e_{rand-1} + e_{rand-3}$

6: **fin si**

7: ReparарCadena (e, exp)

8: **devolver** $e = e_0, e_1,...,e_n$

Función de reparación de cadenas

Ambos operadores de variación de esta propuesta utilizan una estrategia de reparación de soluciones (cadenas). El Algoritmo 17 esquematiza el comportamiento de dicha función.

Algoritmo 17 Función de reparación de cadenas

Entrada: exp; e

El proceso de reparación consiste en aplicar los siguientes pasos sobre la cadena a tratar:

1: Eliminar elementos duplicados.

2: Eliminar los elementos de valor superior al exponente (exp).

3: Comprobar y ordenar, si es necesario, para asegurar que todos los elementos queden en orden ascendente.

4: Asegurar finalización de la cadena con el valor de (exp), por aplicación de las operaciones, en el orden que se listan a continuación:

1. Hallar dos elementos de la cadena que se traduzcan como el valor de (exp).
2. Aplicar uniformemente al azar las siguientes operaciones:
   (a) "Doblar" (sumar un número consigo mismo) último elemento de la cadena, en tanto sea menor que (exp).
   (b) Añadir el último elemento y un elemento aleatorio.
   (c) Añadir dos elementos aleatorios.

Al igual que en las otras propuestas de abordaje del problema, en función de que el objetivo es la reducción al mínimo posible del número de elementos de la cadena asociada al exponente, la función de aptitud empleada es la longitud de la cadena.

En la ejecución de esta propuesta se utilizó una configuración de 50 ejecuciones por cada experimento y se utilizó el criterio de 100 ejecuciones sin mejora de los resultados, como criterio de finalización. Por otro lado se empleó un número total de las generaciones de 1500 y tamaño de población de 300 individuos, en todos los experimentos.

La observación de los resultados de [26] arrojó que los tamaños de población más grandes se destacaron, producto de una mayor diversidad inducida por el mecanismo de inicialización, pero sin embargo, la evolución de los individuos se mantuvo costosa en tiempo de ejecución para los valores grandes exponente, insumiendo para los exponente más grandes, menos de una hora.

Los experimentos realizados con esta propuesta contemplaron pruebas con exponentes hasta el valor $2^{127} - 3$, es decir el número (170 141 183 460 469 231 731 687 303 715 884 105 725).

4.2.4 Uso de un algoritmo inmune para generar cadenas de adición óptimas

Los algoritmos inmunes AIS (por sus siglas en inglés: Artificial Immune System), son aquellos inspirados en un principio denominado selección clonal, que es la manera en que los anticuerpos de un organismo eliminan los antígenos (cuerpos extraños). En este enfoque un antígeno será representado como el exponente e que se desea alcanzar, mientras que los anticuerpos, son representados por el par (U, L), donde U es la secuencia de la cadena de adición, construida previamente mediante alguno de los métodos ya mencionados y L es un número entero positivo que representa la longitud de U, que se traduce en el número de pasos necesarios para lograr el objetivo deseado. La población de anticuerpos representa las posibles soluciones para el problema. En [6] se propone el uso de la metaheurística AIS, haciendo uso de los siguientes elementos:

- Construcción de anticuerpos.
- Un operador denominado “Hipermutación”.
- Memoria inmune.
- Un mecanismo de selección, denominado “Selección Clonal”.

Construcción de anticuerpos.

Cada anticuerpo se representa por el par ordenado (U, L), donde U es una cadena de adición válida de longitud L. En función de esto, se debe definir un procedimiento para construir cadenas de adición válidas a fin de que la población de anticuerpos del sistema inmune pueda crearse y ser mutada.

Operador de hipermutación.

Este operador es inversamente proporcional a la afinidad de los clones, puesto que cuanto mayor es la afinidad de un clon, menor tasa de mutación tendrá y viceversa. La perturbación (mutación) puede ser introducida tanto al inicio (parte inferior) como al final de la cadena (parte superior). En el primer caso será más notable la perturbación y la misma será producida por el grado de afinidad de los clones.

Memoria inmune.

Se trata de cadenas de adición almacenadas en memoria para futuras consultas y podrían servir para aplicar en futuros exponentes.

Algoritmo de selección clonal.

El algoritmo opera ordenando los N anticuerpos de la población asociados al exponente a optimizar, de manera ascendente de acuerdo con su grado de afinidad (longitud de las cadenas). Luego selecciona los mejores anticuerpos sobre los que aplicará la "clonación".

En [6] se expone un ejemplo para clarificar el funcionamiento del algoritmo:

1. Se construye una población inicial de N anticuerpos cada uno de componentes (U, L). Suponiendo por ejemplo que el tercero de ellos es la cadena de adición [1; 2; 4; 8; 16; 24; 48; 49; 98; 196; 294; 588; 612; 1224; 1836; 1844; 1893; 1901; 1903] con una afinidad de 18 (longitud de la cadena).

2. Se clasifica la población de manera ascendente de acuerdo con los valores de afinidad.

3. Se seleccionan los mejores anticuerpos de la población (que poseen diferente longitud), es decir solo aquellos que serán clonados.

4. Se determina el número de clones a ser generados, para los anticuerpos seleccionados.

5. Se crean los clones de los anticuerpos seleccionados.

6. Se aplica a cada clon el operador de hipermutación, de la siguiente manera:

   (a) Por ejemplo, un clon generado desde el más alto grado de afinidad (i-ésimo anticuerpo) obtendrá un punto de mutación seleccionada a partir de la mitad superior de su cadena. Digamos por ejemplo que este punto es $i = 14$.

   (b) Se selecciona un número j aleatorio con $(0 \leq j < i < e)$ por ejemplo $j = 7$.

   (c) El nuevo valor de la cadena de adición clonada en el punto de mutación, será: $U_{i+1} = U_i + U_j$, entonces tenemos por ejemplo: $U_{15} = 1836 + 49 =$

1885. En este punto nuestra cadena es [1; 2; 4; 8; 16; 24; 48; 49; 98; 196; 294; 588; 612; 1224; 1836; 1885].

(d) Se repara la parte superior de la cadena $\{U_k > i + 1\}$ con la operación de llenado, donde k es la ubicación a partir de donde se aplicará dicha operación. Supongamos entonces que la cadena de adición es [1; 2; 4; 8; 16; 24; 48; 49; 98; 196; 294; 588; 612; 1224; 1836; 1885; 1901; 1903] con afinidad $L = 17$ (longitud de cadena).

7. Se calculan los valores de afinidad asociados a los clones mutados.

8. Se seleccionan los N mejores, del conjunto de anticuerpos originales y clones modificados. El resto son desechados.

9. Se reemplazan los anticuerpos que mostraron menos afinidad por otros nuevos. Por ejemplo, digamos que uno de los flamantes candidatos así producidos es [1; 2; 3; 6; 12; 15; 30; 33; 66; 99; 198; 213; 426; 852; 885; 1737; 1836; 1869; 1902; 1903].

10. Se calculan los valores de afinidad relacionados a los nuevos individuos.

11. Se repite desde el paso 3 un determinado número de iteraciones.

12. Se selecciona el mejor anticuerpo B.

13. Puesto que e aún no es = 1903, se continúa con el siguiente paso.

14. Se almacena B en memoria.

15. Se reporta B como la mejor solución encontrada.

En el estudio reportado en [6] se observa que AIS para grandes exponentes se emplea en combinación con el método determinista de ventana adaptativa (deslizamiento de ventana) para la generación de cadenas de adición subóptimas.

4.2.5 Uso de un algoritmo basado en cúmulo de partículas para generar cadenas de adición óptimas

En [19] se describe una propuesta de abordaje del problema de optimización de cadenas de adición mediante un algoritmo de optimización basado en cúmulo de partículas, PSO (por sus siglas en inglés: Particle Swarm Optimization).

El Algoritmo 18 muestra la adaptación de PSO para la problemática de referencia y al igual que otras propuestas las cadenas de adición es una partícula generada de manera estocástica.

Algoritmo 18 Algoritmo PSO para generar cadenas de adición.

Entrada: e
Salida: C = [1;...;e]

```
1: Generar población inicial de partículas (cadenas).
2: mientras El número máximo de generaciones no sea alcanzado, hacer
3:    para cada partícula, hacer
4:       Calcular aptitud (fitness) de cada individuo.
5:       si aptitud de la partícula es mejor que la mejor patícula, entonces
6:          mejor partícula = aptitud de la partícula actual
7:       fin si
8:       Buscar al mejor vecino y asignarle la posición de la mejor partícula.
9:       Actualizar la posición y velocidad de la partícula actual.
10:   fin para
11: fin mientras
12: devolver C
```

Donde:

e: número entero.
C: cadena de adición. Partícula de menor longitud.

Los resultados publicados para PSO en [19], se centraron en la búsqueda de cadenas de longitud mínima para exponentes pequeños y no reportan casos de prueba sobre conjuntos de exponentes difíciles.

4.2.6 Uso de un algoritmo de programación evolutiva para generar cadenas de adición óptimas

En [8] se propone un algoritmo de programación evolutiva EP (por sus siglas en inglés) para la generación de cadenas de adición de longitud mínima. El fundamento del autor para optar por este método radica en la sencillez de implementar un algoritmo EP en comparación con otras metaheurísticas del grupo de las estrategias evolutivas [10] y la falta de pericia de trabajo relacionado con EP, en el estado del arte, para la problemática de referencia.

La sencillez de implementación que encuentra el autor radica en el uso de un único operador (de mutación) a diferencia de los GA que implementan operadores de cruce, mutación y un mecanismo de selección.

Un algoritmo EP posee los siguientes pasos:

- Generar población inicial aleatoriamente de n individuos.
- Posteriormente, mutar cada individuo para generar un solo hijo.
- Aplicar torneo estocástico sobre los individuos generados en paso anterior.
- Los individuos con mayor número de victorias en torneos estocásticos pasarán a la próxima generación, a diferencia de otros algoritmos que seleccionan los individuos con mayor grado de adaptación. Este proceso genera diversidad y pretende evitar la convergencia prematura.

Cabe destacar que EP no requiere efectuar selección de padres ya que todos los individuos de la población generan un hijo mediante el operador de mutación.

- En este enfoque cada individuo de la población estará representado por una cadena: $U = [u_1; u_2; \ldots ; u_i; \ldots ; u_{L_u} = e]$, donde L_u es la longitud de cadena; secuencias numéricas de enteros que cumplan con las propiedades de las cadenas aditivas, ya mencionadas.
- El valor de aptitud de cada individuo estará determinado por su longitud.
- En función de que cada individuo será una secuencia de números enteros positivos (cadena de adición), al igual que la propuesta [6], las soluciones (individuos) estarán representadas a nivel de fenotipo, por lo que no es necesario un mecanismo de decodificación de las soluciones.

A continuación se describen los elementos fundamentales que plantea este enfoque.

Población inicial

Tomando como base las características de las cadenas de adición, cada individuo será generado en base a estas reglas:

1. Se permite el doblado de un número como elemento de la cadena. De este modo en una cadena U, un componente u_i podrá ser igual a: $2 \times u_i \equiv (u_{i+1} = u_i + u_i)$, por ejemplo para cadena [1; 2; 3; 6; 9; 12; 15; 21] el cuarto elemento se compone de sumar el tercer elemento consigo mismo.
2. Suma de dos números previos, de modo que: $u_{i+1} = u_i + u_{i-1}$. Tomando como ejemplo la misma cadena que en punto 1, el quinto elemento (9) es igual al tercero sumado al cuarto.
3. Suma de un número previo más un número aleatorio de la cadena, es decir: $u_{i+1} = u_i + u_{rand}$. Siguiendo el mismo ejemplo anterior, el elemento (15) se conforma de la suma del sexto (12) y tercer elemento (3).

Esta reglas se aplican para generar los términos teniendo en cuenta la premisa de que los dos primeros elementos de las cadenas serán el 1 y 2 de forma consecutiva, seguidos de 3 o 4 (tomado de forma aleatoria) de acuerdo con las 3 reglas anteriores, posteriormente se puede aplicar para obtener nuevos términos una función como la que se presenta en [6].

Operador de variación (Mutación)

Este estudio propuso utilizar como operador de variación una búsqueda local donde se generan t mutantes para cada individuo de los cuales se toma como individuo hijo aquel de mejor valor de adaptación.

En este mecanismo la población inicial y sus hijos (individuos mutados) se agrupan formando un único conjunto. Cada individuo compite con otra q cantidad de individuos escogidos aleatoriamente, mediante enfrentamientos (torneos), donde el criterio para definir el vencedor será la longitud de los individuos. Cada individuo tendrá asociado un número de victoria (contador). Se ejecutará un incremento del contador de número de victorias por cada individuo que resulte mejor respecto de otro de los q individuos elegidos al azar. Por último se ordenan los resultados según el número de victorias, y de este conjunto se tomará la primer mitad de elementos para la siguiente generación y la otra mitad será descartada.

En [8] se efectuaron pruebas con el modelo basado en EP, sobre exponentes pequeños y grandes. Cabe destacar que se considera pequeño a todo exponentes de longitud menor a los 128-bits. Estos experimentos condujeron a tener que proponer modificaciones del EP para el tratamiento de exponentes pequeños y por otro lado se observó también que EP se quedaba "atrapado" en óptimos locales para algunos problemas con exponentes grandes, lo que se evidenció frente a la dificultad de hallar cadenas más cortas que los métodos deterministas para los casos de exponentes de longitud mayor a los 128-bits.

4.2.7 Uso de un algoritmo de optimización basado en colonia de hormigas (ACO)

En [25] fue propuesto un algoritmo ACO (por sus siglas en inglés: Ant Colony optimization) para la optimización de cadenas de adición. La novedad de este aporte radica en el uso de un algoritmo basado en inteligencia colectiva (esquema multiagente).

El enfoque de ACO propone ver el método como un sistema multiagente donde se emplea una memoria compartida a través de la cual se comunican las hormigas (agentes) y una memoria local que almacena información sobre las soluciones alcanzadas.

Aquí se representa la estructura general de un sistema de hormigas, en el que A_i representa cada agente de orden i (i-ésima hormiga) y LM_i la memoria local de la misma, respectivamente. Por otro lado, la memoria compartida SM, contiene la información del rastro de feromona, mientras que la memoria LM_i local (de cada hormiga) mantiene la solución (posiblemente parcial), es decir, que tan lejos llegó el agente A_i. El Algoritmo 19 esquematiza el funcionamiento de ACO, cuyos parámetros son (N, C), detallado en [25].

Algoritmo 19 Artificial Ant Colony.

Entrada: (N, C)
Salida: S

1: Inicializar SM con valores iniciales de feromona.
2: **para** $i : [1..N]$, **hacer**
3: Comienzo búsqueda con (A_i, LM_i)
4: Activar = Activar A_i // activar hormigas.
5: hacer,
6: Actualizar SM.
7: Cuando una hormiga (A_i) comunica que se detiene, hacer:
8: Activo: Activo = $\{A_i\}$
9: Φ: = Feromonas (LM_i)
10: Actualizar SM con valor (Ψ)
11: S = ExtractSolution (LM_i) // se extrae solución del agente A_i y se asigna a S.
12: hasta características (S) = CorActive = ϕ
13: **mientras** Activa = ϕ, **hacer**
14: Detener la hormiga $A_i \mid A_i$ activo.
15: Activo: Activo = $\{A_i\}$
16: hasta características (S) = C
17: **fin mientras**
18: **fin para**
19: **devolver** S

Donde:

A_i: cada uno de los agentes (hormigas).
LM_i: memoria local de la i-ésima hormiga.
N: número de hormigas artificiales o agentes que conforman la colonia.
C: características de la solución esperada.
SM: memoria compartida, para almacenar información de feromonas.
Ψ: valor global de feromona.
Φ: cantidad de feromona relacionada con la calidad de una solución determinada.
S: solución.

El comportamiento de una colonia de hormigas se resume en el Algoritmo 19, donde el primer paso consiste en la activación de N hormigas que deberán funcionar en forma simultánea (paso 4, "Activar"). Cada vez que una hormiga finaliza su búsqueda se actualiza la memoria compartida con un valor de feromona (cantidad de feromona) que será proporcional a la calidad de la solución alcanzada. Entonces luego se ejecuta la actualización de la memoria compartida con el valor de feromona global (paso 10). Cuando el rendimiento de una solución obtenida por el trabajo de una hormiga es adecuado (es decir, que se ajusta con C), se detienen todas las hormigas activas (paso 14). En caso contrario, se itera el proceso hasta que se encuentra una solución adecuada (paso 15).

Sistema de memoria compartida.

En este método la memoria compartida es la encargada de propiciar la comunicación entre las hormigas y es en función de la misma que las hormigas colaboran en la construcción de las cadena de adición. Como se mencionó, las hormigas utilizan un sistema de memoria compartida para la comunicación que se representa mediante una matriz de 2 dimensiones, las cuales son: el valor del exponente e que define las filas y el número de columnas será proporcional al de filas. Otro punto a tener en cuenta es que la memoria compartida se divide en dos partes: una de ellas almacena la cadena de adición obtenida por una hormiga (solución) y la otra almacena las características de dicha solución.

Este trabajo es de fecha anterior a otros aportes ya descritos para otras metaheurísticas (AG, EP). En el mismo no se muestra evidencias de pruebas con exponentes grandes y si para pequeños del rango de los 32-bits a los 128-bits.

Capítulo 5

Algoritmo Propuesto

5.1 Algoritmo basado en comportamiento de lobos grises (Grey Wolf Optimizer)

En [24] se propone una metaheurística denominada Grey Wolf Optimizer, GWO (por su siglas en inglés), inspirada en el comportamiento de las manadas de lobos grises (Canis Lupus) y su organización social para la caza de presas.

Los lobos grises por naturaleza prefieren vivir en grupo de entre 5 y 12 miembros en promedio. La manada se conforma de dos líderes llamados alfa (un macho y una hembra) y el resto de la manada. En esta estructura social, los miembros alfa son los responsables de la toma de decisiones sobre la caza, lugar donde dormir, hora de despertar, y el resto de los miembros debe seguir sus órdenes. El miembro alfa de una manada no necesariamente debe ser el más fuerte pero si el mejor en términos de conducción del grupo y capacidad para tomar decisiones. El segundo nivel en la jerarquía del grupo son los miembros beta (subordinados de los alfa), que deben colaborar con la toma de decisiones y el cumplimiento de las órdenes de los alfa. Estos miembros son los candidatos a convertirse en alfa cuando alguno de estos fallece o envejece. Existe otra clasificación que se encuentra más abajo en la jerarquía de las manadas de lobos grises se denomina omega, estos deben someterse a los alfa y betas obedeciéndoles, en muchos casos suele verse en este tipo a miembros que ofician de niñeros de las crías, dentro de la manada [22].

Las principales fases de la caza en manadas de lobo gris son las siguientes:

- Seguir, perseguir, y acercarse a la presa.
- Rodear, y acosar a la presa hasta que se “paraliza”.
- Atacar a la presa.

Esta técnica de caza naturalmente e intuitivamente realizada por los lobos grises, fue modelada matemáticamente en [24] con la finalidad de proponer un método de optimización y emplearlo en la resolución de problemas complejos.

5.1.1 Modelo matemático y el algoritmo GWO

A continuación se describe el modelo matemático de la jerarquía social de las manadas de lobos grises en base a las tres acciones principales relacionadas con la actividad de la caza: acorralar, seguir y atacar.

Jerarquía social de las manadas:

Para modelar matemáticamente la jerarquía social de los lobos en el diseño de GWO, se considera que la solución más apta para el problema que se intenta resolver es la alfa (α). En consecuencia, la segunda y tercera mejores soluciones se denominan beta (β) y delta (δ), respectivamente. El resto de las soluciones candidatas, se supone que son omega (ω), esta jerarquía social se ilustra en la Figura 5.1. En el algoritmo GWO la actividad de caza se traduce como proceso de optimización, y el mismo es guiado por α, β, y δ, mientras que los lobos ω siguen a los 3 tipos previamente descriptos.

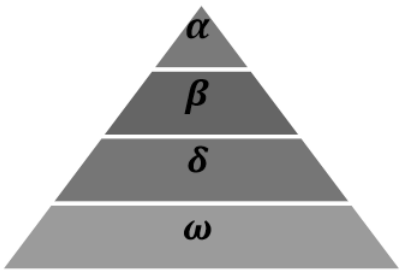

Figura 5.1: Jerarquía de miembros de una manada de lobos grises.

Acorralamiento de la presa:

Los lobos grises rodean la presa durante la caza. Matemáticamente este comportamiento se modela de la siguiente manera:

$$D = |C \cdot X_p(t) - X(t)| \tag{5.1a}$$

$$X(t+1) = X_p(t) - A \cdot D \tag{5.1b}$$

Donde t indica la iteración actual, A y C son vectores de coeficientes, X_p es el vector de posición de la presa, y X indica la posición vector de un determinado lobo gris.

Por otro lado, los vectores A y C se calculan como sigue:

$$A = 2a \cdot r_1 - a \tag{5.2a}$$
$$C = 2 \cdot r_2 \tag{5.2b}$$

Según la formulación original GWO en [24] los componentes del vector A disminuyen linealmente desde 2 hasta 0, sobre el curso de las iteraciones y r_1, r_2 son vectores aleatorios en el rango $[0, 1]$.

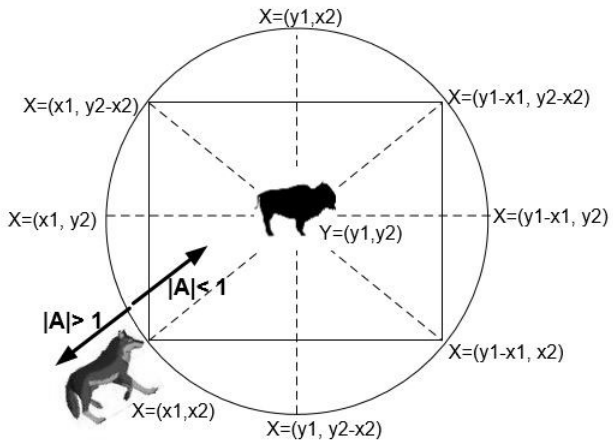

Figura 5.2: Acorrralamiento de la presa.

La Figura 5.2 esquematiza los efectos de las ecuaciones (5.1a) y (5.1b). Se muestra en la figura un lobo gris en la posición (x_1,x_2), éste sería el equivalente a X de la ecuación (5.1b) y puede actualizar su posición según la posición de la presa Y=(y_1,y_2), es decir el equivalente a Xp de la ecuación (5.1a). Como se observa en la figura, pueden alcanzarse diferentes lugares alrededor de la presa, con respecto a la posición actual del lobo, ajustando el valor de las coordenadas de los los vectores de posición. Por ejemplo, X=(x_1, y_2-x_2).

El modelo matemático de GWO describe la operación de acercamiento en función de que un lobo gris en una determinada posición (en el espacio n-dimensional), podrá actualizar su posición respecto de la presa, trasladando la ubicación del lobo (o agente) mediante el ajuste del valor de los vectores A y C.

Ataque a la presa:

Los lobos grises tienen la capacidad de reconocer la ubicación de la presa y rodearla. En la naturaleza la caza suele ser guiada por el miembro alfa, aunque los miembros beta y delta que le siguen en jerarquía, pueden participar eventualmente de la caza. Con el fin de simular matemáticamente el comportamiento de caza de los lobos grises,

suponemos que el miembro alfa (mejor solución candidata), beta y delta tienen un mejor conocimiento sobre la posible ubicación de la presa. Por lo tanto, guardamos las primeras tres mejores soluciones obtenidas hasta el momento y obligamos a los otros agentes de búsqueda (incluidos los omegas) a actualizar sus posiciones de acuerdo a la posición de los mejores agentes de búsqueda. En este sentido se proponen las siguientes fórmulas:

$$D_\alpha = |C_1 \cdot X_\alpha - X(t)|, D_\beta = |C_2 \cdot X_\beta - X(t)|, D_\delta = |C_3 \cdot X_\delta - X(t)| \tag{5.3a}$$

$$X_1 = X_\alpha - A_1 \cdot (D_\alpha), X_2 = X_\beta - A_2 \cdot (D_\beta), X_3 = X_\delta - A_3 \cdot (D_\delta) \tag{5.3b}$$

$$X(t+1) = (X_1 + X_2 + X_3)/3 \tag{5.3c}$$

X_1, X_2 y X_3 son las mejores 3 soluciones en la población para la iteración t, dichos valores son evaluados en las ecuaciones (5.3b). Los valores D_α, D_β y D_δ son obtenidos con las ecuaciones (5.3a). Los valores de C_1, C_2 y C_3 se evalúan mediante la ecuación (5.2b). Los valores de A_1, A_2 y A_3 son evaluados con la ecuación (5.2a).

En función de que cada agente actualiza su posición según: alfa, beta y delta en un espacio de búsqueda, se puede decir que el rol de alfa, beta y delta es estimar la posición de la presa y los otros lobos actualizan sus posiciones al azar alrededor de la misma.

Exploración (seguimiento) versus Explotación (ataque):

Los lobos se acercan a la presa y la atacan cuando la misma deja de moverse. El modelo matemático de tal comportamiento de acercarse a la presa, se traduce como la disminución del valor del vector A (coeficiente). Cabe destacar que el rango de fluctuación del vector A, también se reduce por el valor de "a" (5.2a). Cuando $|A| < 1$, GWO, obliga a los lobos a atacar a la presa. Esta característica está relacionada con la explotación del algoritmo.

GWO es propenso al estancamiento en las soluciones locales con estos operadores. Y si bien es cierto que el mecanismo de "rodeo" que posee genera exploración hasta cierto punto, necesita más operadores para enfatizar dicha exploración.

Los lobos tienen posiciones diferentes respectos de la presa pero convergen a la misma para atacar. El modelo matemático de esta divergencia se refleja en el uso de valores aleatorios para el vector A (mayores que 1 y menores que -1), a través de r_1 (ver 5.2a), para asegurar divergencia y que la búsqueda abarque de mejor manera el espacio de búsqueda, característica relacionada con la exploración, es decir que cuando $|A| > 1$ se está obligando a los lobos a divergir respecto de la presa.

En resumen, el proceso de búsqueda en GWO se inicia con la creación de población de lobos al azar (soluciones candidatas). En el transcurso de las iteraciones, los lobos alfa, beta y delta, estiman la posición probable de la presa. Cada solución candidata

actualiza su distancia respecto de la presa. El parámetro "a" es disminuido desde 2 hasta 0 con el fin de enfatizar la exploración y explotación, respectivamente. Las soluciones candidatas divergen respecto de la presa cuando $|A| > 1$ y convergen hacia la presa cuando $|A| < 1$ (ver figura 5.2). Por último, el algoritmo GWO termina por la satisfacción de un criterio de fin. Tal como describe en [24] GWO ha sido aplicado a una serie de problemas de optimización y dentro de ellos se aplicó a un problema real del ámbito de la ingeniería óptica [24]. En el Algoritmo 20 se presenta el esquema general del método.

Algoritmo 20 GWO.

Inicializar población de lobos X_i (con $i = 1, 2, ..., n$)
Inicializar α, A y C
Calcular fitness de cada agente de búsqueda
$X\alpha =$ el mejor agente.
$X\beta =$ el segundo mejor agente.
$X\delta =$ el tercer mejor agente.
mientras ($t <$ número máximo de iteraciones) **hacer**
  para $i = [1..n]$ **hacer**
    Actualizar la posición de cada agente mediante la ecuación (5.3c)
  fin para
  Actualizar α, A y C
  Calcular el fitness de todos los agentes de búsqueda
  Actualizar X_α, X_β y X_δ
  $t = t + 1$
fin mientras
Retornar $X\alpha$

5.2 Propuesta inicial - GWOc

El objetivo del presente trabajo es abordar el problema de optimización de generación de cadenas de adición cuya longitud sea mínima y abordarlo con una metaheurística no explorada o escasamente explorada para la problemática como lo es GWO y la cual es de más reciente surgimiento que las ya utilizadas para el problema de referencia. Se empleó una función programada sobre lenguaje MATLAB©, capaz de generar cadenas de adición asociada a exponentes que recibe como parámetros de entrada [8] (ver Algortimo 21) y la misma fue optimizada mediante GWO.

En el Algoritmo 21, e es el exponente para el cual se genera la cadena de adición, $U = [u_0; u_1; u_2; \ldots; e]$ es la cadena de adición completa de longitud L, Rand(3,4) representa el método que toma de manera aleatoria el valor 3 o 4 para generar la tercera componente de la cadena y FILL es la función llenar, que se utiliza para completar la cadena y se describe en el Algoritmo 22.

Algoritmo 21 Producir cadena de adición válida.

Entrada: e
Salida: (U, L)
Asignar a $u_0 = 1$ y $u_1 = 2$
Asignar a $u_2 = \text{Rand}(3,4)$
Completar la secuencia invocando la función FILL(U, k, e) // parámetros que emplea la función para "llenar" la cadena.
Retornar (U, L)

La función FILL opera tomando como parámetro de entrada una cadena de adición incompleta U de longitud L, asociada al exponente e. El valor de k representa la siguiente posición de la cadena a completar.

El resultado de la ejecución de FILL será una cadena de adición completa y factible para el exponente e en cuestión.

Algoritmo 22 FILL(U, k, e) // completar cadena.

```
Establecer i = k – 1
mientras u_i ≠ e hacer
  si FLIP(f) entonces
    u_{i+1} = 2u_i // aplicar regla 1
  si no
    si FLIP(g) entonces
      u_{i+1} = u_i + u_{i-1} // aplicar regla 2
    si no
      u_{i+1} = u_i + u_rand // aplicar regla 3
    fin si
  fin si
  mientras (u_{i+1}) > e // Para que la siguiente posición sea mayor que e hacer
    j = i – 1
    u_{i+1}=u_i+ u_{i-j}
    j = j – 1 // decrementar para no superar el valor de e.
  fin mientras
fin mientras
Retornar (U)
```

Donde:

U: cadena a completar (no vacía), parámetro de la función.
f: probabilidad de aplicar regla 1.
g: probabilidad de aplicar regla 2.

Al igual que en [8] los individuos de la población serán generados aplicando las 3 reglas mencionadas a continuación, siempre con la premisa de que las dos primeras componentes de la cadena son 1 y 2, en este mismo orden:

1. Se permite que un elemento de la cadena resulte de sumar al elemento previo por sí mismo. De este modo, en una cadena U, un componente u_i+1 podrá ser igual a $2u_i \equiv (u_{i+1} = u_i + u_i)$. Por ejemplo, para la cadena [1; 2; 3; 6; 9; 12; 15; 21], el cuarto elemento se compone de sumar el elemento (3) consigo mismo.

2. Suma de dos números previos, de modo que: $u_{i+1} = u_i + u_{i-1}$. Tomando como ejemplo la misma cadena del punto 1, el quinto elemento (9) es igual al tercero sumado al cuarto.

3. Suma de un número previo más un número aleatorio de la cadena, es decir $u_{i+1} = u_i + u_{rand}$. Siguiendo el mismo ejemplo anterior, el elemento (15) se conforma de la suma del sexto (12) y el tercer elemento (3).

Para adaptar el comportamiento de GWO para resolver la problemática objeto de estudio, vamos a establecer que nuestra población de lobos (soluciones) es una población de cadenas de adición válidas para un exponente e determinado, es decir secuencias numéricas de enteros y que responden a las reglas mencionadas anteriormente. Por otro lado, teniendo en cuenta que el propósito de optimizar cadenas para un exponente determinado consiste en obtener cadenas de longitud mínima, la función de adaptación (fitness) a utilizar, será la longitud de las cadenas.

De esta manera, la función FILL aplica las reglas anteriores con determinados valores de probabilidad. Para los experimentos se ha tomado como referencia los valores reportados por el enfoque [8] que ha demostrado resultados competitivos para varios rangos de exponentes, a fin de poder comparar nuestra propuesta bajo condiciones similares en términos de generación de soluciones y desempeño de la metaheurística propuesta. Dichos valores son:

- Tasa de determinación de un elemento de la cadena a partir de la suma del número anterior por sí mismo (regla 1) = 0.7
- Tasa de suma de elementos anteriores (regla 2) = 0.2

En la siguiente sección, se presentan los resultados vertidos de una serie de experimentos y pruebas estadísticas realizadas sobre la metaheurística GWO aplicada al problema objeto de estudio, cuya implementación hemos llamado en el contexto de esta tesis GWOc ("Grey Wolf Optimizer of Chains": "Lobo Gris Optimizador de Cadenas"). Todos los resultados obtenidos con el enfoque GWOc, han sido generados aplicando la siguiente configuración de parámetros:

- Agentes_de_búsqueda= 10 (número de lobos).
- Máximo_iterationes= 30 (cantidad de ejecuciones).

A continuación se describe de qué manera se probó el desempeño de GWOc para la optimización de cadenas de adición. Asimismo se describen los experimentos ejecutados y los resultados obtenidos. Para todos los experimentos se utilizó la metaheurística GWOc sobre MATLAB© v.7.8.0 (R2009a) bajo Sistema Operativo Windows 8.1 Enterprise 64 bits, en un computador con Intel Core i-3., 2.10 GHz. y 4 Gb. RAM.

5.3 Descripción de los experimentos

En esta sección se describe de qué manera se estudió el desempeño del algoritmo propuesto (GWOc) para la búsqueda de cadenas de adición de longitud mínima y la calidad de soluciones obtenidas, mediante comparación con trabajos precedentes y pruebas estadísticas.

Se diseñaron tres experimentos, en función de los objetivos de la tesis. En cada experimento se comparó los resultados obtenidos con GWOc, con los reportados por otras propuestas del estado del arte.

5.3.1 Longitudes acumuladas de GWOc para exponentes en [1, 512]

El primer experimento planteado radica en calcular el total de longitudes acumuladas de cadenas aditivas para conjuntos de exponentes de pequeña longitud, a fin de comparar el desempeño de algoritmo GWOc con otros resultados obtenidos por métodos deterministas de la literatura y por otras metaheurísticas del estado del arte.

Tabla 5.1: Promedio de resultados obtenidos por GWOc sobre los siguientes rangos de exponentes con 30 ejecuciones independientes.

Exponente	Longitud acumulada	Promedio	Mediana	Desviación
[1, 128]	960	7,5	4	2,159
[1, 256]	2459	9,6	9	2,986
[1, 512]	4994	9,8	10	1,9501

La Tabla 5.1 resume los mejores resultados obtenidos por enumeración, para los exponentes de cada rango expresado ([1,128], [1,256] y [1,512]), es decir, para el rango [1,128] se obtuvo las cadenas de adición para los números del 1 al 128 y

así sucesivamente para los otros rangos numéricos. La operación antes mencionada se ejecutó 30 veces por cada rango, tomando el mejor resultado en términos de longitud de cadenas acumuladas dentro de los resultados de cada rango, a fin de poder comparar tales resultados con otras propuestas para las que se han ejecutado experimentos empleando los mismos rangos de exponentes.

Tabla 5.2: Cadenas de adición acumuladas para todas las longitudes de exponentes $e \in [1, 512]$

e	$[1, 512]$
Optimal [6]	4924
Binary [6]	5388
Çuaternary [6]	5226

Resultados GWOc

Longitudes acumuladas	4994
Promedio	9,77
Mediana	10
Desvío	1,95

Tabla 5.3: Comparación de los mejores resultados acumulados para GWOc con otros enfoques, para el mismo rango de exponentes.

$e \in [1, 512]$	AIS	GA	PSO	EP	GWOc
	4924	4924	—	4924	4994

La Tabla 5.2 muestra una comparación de los resultados para GWOc respecto de los métodos deterministas del estado del arte tomados de [6] cuyos resultados fueron utilizados para comparar con el enfoque AIS [6]. Por otro lado, el Cuadro 5.3 brinda una comparación de los mejores resultados de cadenas de adición acumuladas para el mismo rango de exponentes $[1, 512]$ obtenidos por métodos estocásticos y comparados en [8], salvo PSO que no reportó resultados para este rango de exponentes, respecto de la propuesta GWOc. En el Cuadro 5.2 se puede observar además que GWOc se aproxima al resultado del método "Optimal"[6], siendo éste quien mejor desempeño logró para dicho rango de exponentes según [6]. Finalmente, en el Cuadro 5.3 se observa que GWOc, en su mejor ejecución, se acerca a los mejores resultados que arrojaron AIS, GA, PSO, EP para el rango [1, 512]. En comparación con los resultados de los métodos deterministas se observa que en su mejor ejecución se aproxima al mejor resultado arrojado por "Optimal", mientras que en los resultados comparativos con los demás métodos estocásticos se acerca a los mejores resultados reportados de las otras propuestas, no superando las mismas por un porcentaje de $\sim 0,9$ %.

Puesto que GWOc ha obtenido valores muy cercanos a otros enfoques del estado del arte como AIS y EP para el mismo rango de exponentes [1, 512] se aplicará la prueba "T" de Student para demostrar que tales resultados son competitivos y que

las diferencias no son significativas.
Se toma GA[7], AIS[6] y EP[8] como puntos de comparación pues se tiene para sendos estudios los valores de promedios de longitudes acumuladas y desvío estándar, obtenidos por las metaheurísticas para las muestras. PSO [19] no será tomado en cuenta puesto que no reporta resultados para dicho rango de exponentes.

GWOc respecto de GA[7]

- Promedio de las muestras:

  Para GWOc: $X1 = 4994$ promedio = 9,77.
  Para GA: $X2 = 4927,7$ promedio = 9,62.
  $X1, X2$: longitudes acumuladas para GWOc y GA respectivamente.

- Desvíos de las muestras 1 y 2: s_1= 1,95 - s_2= 4,74

- Tamaño de las muestras: n_1=n_2=512

- Planteamiento de la hipótesis de términos estadísticos:
  H_0 (hipótesis nula) aunque las muestran registren promedios y desvíos diferentes provienen del mismo universo y por tanto la diferencia observada se debe al azar.
  N_a (hipótesis alternativa) las muestran provienen de universos con promedios distintos.

- Nivel de significación:
  α=0,05

- Estadístico de prueba:
  $S^2 = [(n_1 - 1){S_1}^2 + (n_2 - 1){S_2}^2]/(n_1 + n_2 - 2) = [(512 - 1)3,8025 + (512 - 1)22,4676]/1022 = (1943,0775 + 11480,9436)/1022 = 13,13$

- $$T = \frac{(X1 - X2)}{\sqrt{S^2}(\frac{1}{n^1} + \frac{1}{n^2})} = 10,6$$

- Los grados de libertad son 1022, (n_1+n_2-2)=(512+512-2), entonces verificamos en la tabla de "T de Student" el valor 10,6 de T obtenido, para 1022 grados de libertad.

En la tabla de "T de Student" el valor 10,6 para 1022 grados de libertad se encuentra entre 1,645 y 1,960.
El hecho de que el valor del estadístico de prueba sea mayor que la mitad de α (nivel de significación) implica la aceptación de la hipótesis nula (pues dicho resultado se encuentra en la "zona de confianza"), por tanto no existe diferencia significativa entre los resultados de GWOc y GA para el experimento de referencia.

GWOc respecto de AIS[6]

- Promedios de las muestras:

  Para GWOc: $X1$=4994 promedio = 9,77
  Para AIS: $X2$=4925,03 promedio = 9,62
  $X1, X2$: longitudes acumuladas para GWOc y AIS respectivamente.

- Desvíos de las muestras 1 y 2: s_1=1,95 – s_2=0,89

- Tamaño de las muestras: n_1=n_2=512

- Planteamiento de la hipótesis de términos estadísticos:
  H_0 (hipótesis nula) aunque las muestran registren promedios y desvíos diferentes provienen del mismo universo y por tanto la diferencia observada se debe al azar.
  N_a (hipótesis alternativa) las muestran provienen de universos con promedios distintos.

- Nivel de significación:
  α=0,05

- Estadístico de prueba:

  S^2= $[(n_1 - 1){S_1}^2 + (n_2 - 1){S_2}^2]/(n_1 + n_2 - 2) = [(512 - 1)3,8025 + (512 - 1))0,7921]/1022 = (1943,0775 + 404,7631)/1022 = 2,3$

- $$T = \frac{(X1 - X2)}{\sqrt{S^2}(\frac{1}{n^1} + \frac{1}{n^2})} = 25,26$$

  Para 25,26 con 1022 grados de libertad, los valores de probabilidad están entre 1,708 y 2,060. Esto implica que se debe aceptar la hipótesis nula y por tanto al no existir diferencias significativas entre ambas metaheurísticas se concluyen en que los resultados de GWOc fueron satisfactorios.

GWOc respecto de EP[8]

- Promedios de las muestras:

  Para GWOc: X1= 4994 promedio: 9,77
  Para EP: X2= 4924,10 promedio:9,62
  $X1, X2$: longitudes acumuladas para GWOc y EP respectivamente.

- Desvíos de las muestras 1 y 2: s_1=1,95 – s_2= 0,305

- Tamaño de las muestras: $n_1 = n_2$=512

- Planteamiento de la hipótesis de términos estadísticos:
  H_0 (hipótesis nula) aunque las muestran registren promedios y desvíos diferentes provienen del mismo universo y por tanto la diferencia observada se debe al azar.
  N_a (hipótesis alternativa) las muestran provienen de universos con promedios distintos.

- Nivel de significación:
  α=0,05

- Estadístico de prueba:
  S_2= $[(n_1 - 1){S_1}^2 + (n2 - 1){S_2}^2]/(n_1 + n_2 - 2) = [(512 - 1)3,8025 + (512 - 1)0,093025]/1022 = (1943,0775 + 47,535775)/1022 = 1,95$

- $$T = \frac{(X1 - X2)}{\sqrt{S^2}(\frac{1}{n^1} + \frac{1}{n^2})} = 27,51$$

Para 27,51 con 1022 grados de libertad los valores de probabilidad están entre 1,703 y 2,052, por lo tanto se debe aceptar la hipótesis nula considerando que no existen diferencias significativas entre los resultados de ambas metaheurísticas.

Tanto para los resultados de la prueba de comparación de GWOc con AIS como GWOc respecto de EP, el valor del estadístico obtenido conduce a la aceptación de la hipótesis nula y por tanto se concluye que las diferencias en los resultados de los experimentos entre las metaheurísticas comparadas no son significativos dejando en evidencia que GWOc obtuvo resultados competitivos.

5.3.2 GWOc vs. GA Annealing para exponentes específicos

El experimento plantea la comparación del desempeño de GWOc respecto de una propuesta reciente [27] del estado del arte.

Tabla 5.4: Comparación de GA Annealing [27] y GWOc para el mismo conjunto de exponentes.

Exp.	Cadena	GA Annealing	GWOc
e		Longitud	
23	1, 2, 4, 5, 10, 20, 21, 23	7	7
55	1, 2, 3, 6, 12, 24, 27, 54, 55	9	**8**
130	1, 2, 4, 8, 16, 32, 64, 128, 129, 130	11	**9**
250	1, 2, 3, 5, 10, 20, 30, 50, 100, 150, 250	13	**10**
768	1, 2, 3, 6, 12, 24, 48, 96, 192, 384, 768	23	**10**

La Tabla 5.4 muestra una comparación de los resultados de GWOc respecto de GA Annealing [27] para una serie de exponentes diversos y especialmente difíciles de

optimizar. Se considera dentro de esta categoría a aquellos exponentes para los cuales no es posible hallar cadenas de longitud mínima mediante los métodos deterministas. Se puede observar que en los 5 casos, GWOc ha obtenido cadenas de longitud igual o menor al enfoque propuesto en [27]. Si bien los métodos aplicados en [6], [19] y [8] también han sido probados con esta clase de exponentes diversos, no son los mismos que los empleados en [27]. Por esta razón, se efectúa la comparación en Tabla 5.4 y en el próximo experimento se hace respecto a los otros métodos del estado del arte.

Se aplicó la prueba de los rangos de signos de Wilcoxon a fin de verificar los resultados obtenidos por GWOc respecto de [27] con α= 0,05 a fin de validar si existen diferencias significativas. La comparación fue realizada en función de las longitudes de cadenas obtenidas con cada metaheurística. A continuación se describe el procedimiento.

- Planteamiento de la hipótesis en términos estadísticos:
  Se plantean las hipótesis para efectuar la prueba de las dos colas.
  H_0 (hipótesis nula) la media de las longitudes de cadenas de [27] es igual a la media de la longitud de cadenas de GWOc. En este caso los resultados de GWOc son iguales a los de [27].
  H_a (hipótesis alternativa) la media de las longitudes de cadenas de [27] es mayor que la media de los resultados de GWOc, por tanto los resultados de GWOc son superiores a los de [27] pues si la media es más pequeña significa que los valores de longitud son menores que los obtenidos por la otra propuesta. Nivel de significancia: α=0,05.

- Obtención de los signos de rangos de Wilcoxon.

  Primer paso

  Este paso consiste de calcular la diferencia y expresar los mismos en valor absoluto, entre los mejores resultados de cada propuesta (GWOc y GA Annealing), obtenidos para cada exponente (ver Tabla 5.5).

Tabla 5.5: Cálculo de las diferencias entre los resultados de las metaheurísticas.

Exp.	GA annealing	GWOc	Diferencia	Valor Absoluto
e	Longitud			
23	7	7	0	0
55	9	8	1	1
130	11	9	2	2
250	13	10	3	3
768	23	10	13	13

Segundo paso

Puesto que lo importante son las filas de la tabla que contienen los resultados de las diferencias en valor absoluto para cada exponente, nos quedaremos solo con las columnas exponente (“Exp.”) y “Valor Absoluto”. Las filas de la tabla

deben quedar ordenadas de manera decreciente, en función del valor absoluto, en este caso ya se encontraban ordenadas desde el paso anterior (ver Tabla 5.6).

Tabla 5.6: Diferencias de cada metaheurística en valor absoluto para cada exponente.

Exp.	Valor Absoluto
e	
23	0
55	1
130	2
250	3
768	13

Tercer paso

A cada valor de diferencia de la Tabla 5.6 se le asignará un número de rango, de manera creciente.

Tabla 5.7: valores absolutos y número de rango para cada diferencia entre los resultados de GWOc y GA Annealing.

Exp.	Valor Absoluto	Rango
e		
23	0	1
55	1	2
130	2	3
250	3	4
768	13	5

Cuarto paso

En este paso se obtienen los signos asociados a cada rango del siguiente modo, los rangos asociados a valores de la columna "Diferencia" con signo positivo se colocarán en la columna "(+)" y aquellos con signo negativo en la columna "(−)", adicionalmente se descartan las diferencias nulas (ver Tabla 5.8).

Tabla 5.8: Signos asociados a los rangos de cada diferencia.

Posición	Diferencia	Rango	(+)	(−)
1	1	1	1	-
2	2	2	2	-
3	3	3	3	-
4	13	4	4	-
Σ			10	-

- Cálculo del estadístico W de Wilcoxon.
  El valor del estadístico W será la suma de los rangos correspondientes a las diferencias positivas, en este caso todas las diferencias resultaron ser positivas, por lo que $W = 10$.

- Criterio de decisión.
  El p-valor de la prueba se calcula en 0,00195, y se obtiene de la siguiente manera.

p-valor unilateral	$p = k \times (1/2)^n \quad p = 4 \times (1/2)^{10}$
p-valor bilateral (de dos colas)	$p = 2 \times p$
n:	rangos de magnitudes observadas.
k:	cantidad de rangos positivos.

  Teniendo en cuenta que se aplica prueba de dos colas sería 0,0039 ($2\times p$-valor) y este resultado menor que α=0,05, con lo cual se rechaza la hipótesis nula en favor de la hipótesis alternativa.

- Conclusión. Se rechaza H_0 y por tanto las dos poblaciones no son idénticas y existe una diferencia significativa entre los resultados de GWOc y [27], siendo superiores los de GWOc.

A continuación se muestran las curvas de convergencia del algoritmo GWOc para los casos ejecutados en el experimento, conforme el avance de las iteraciones, correspondiente a la ejecución con la cual se obtuvo el valor mínimo de longitud de cadena de adición para cada exponente.
Vale aclarar que en las siguientes gráficas los valores correspondientes al eje de abscisas no parten de cero sino que lo hacen desde el menor valor encontrado de longitud de cadena de adición para cada exponente.

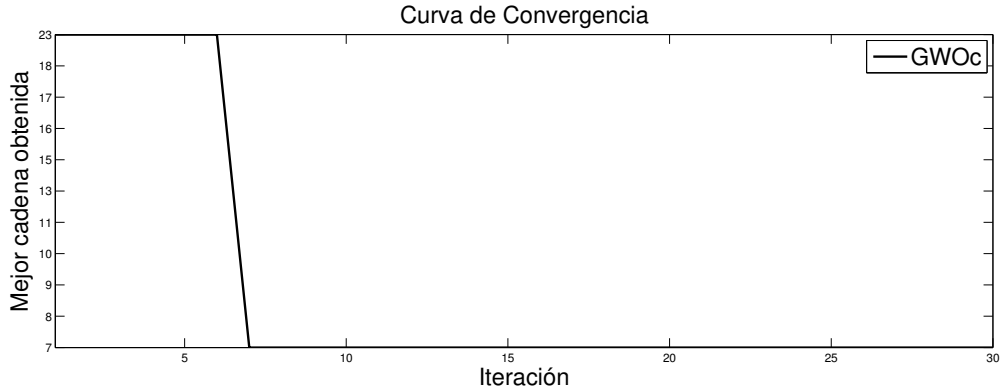

Figura 5.3: Cadena para e=23.

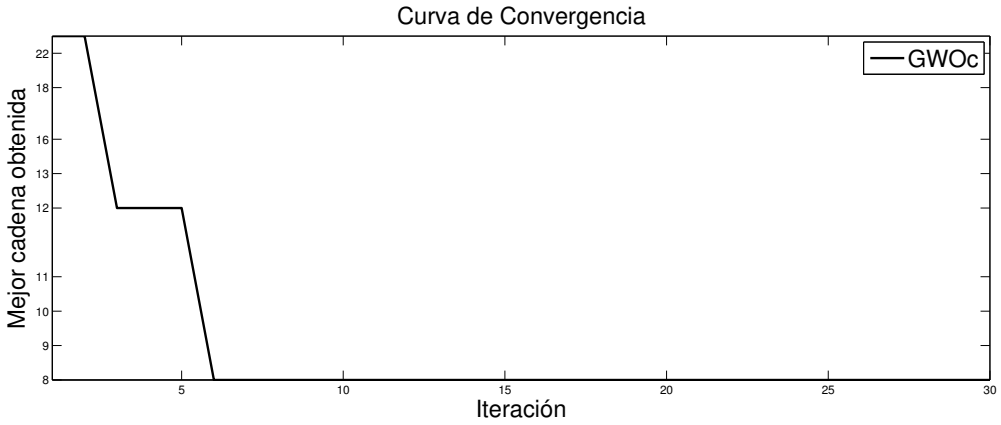

Figura 5.4: Cadena para e=55.

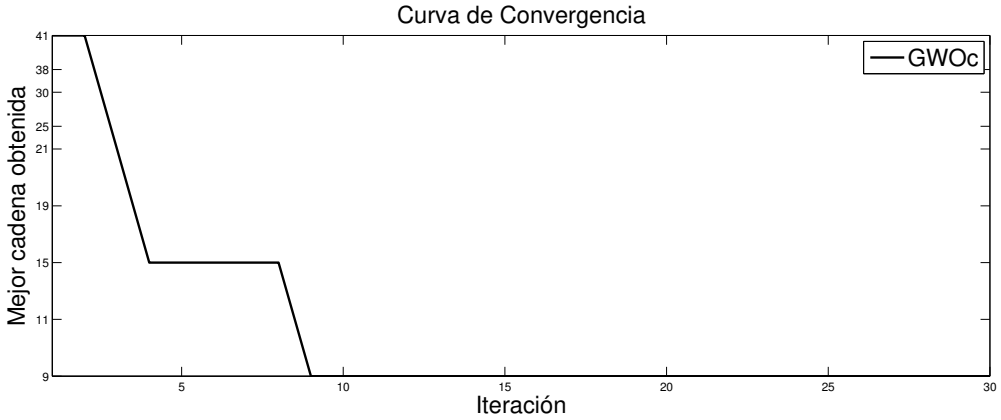

Figura 5.5: Cadena para e=130.

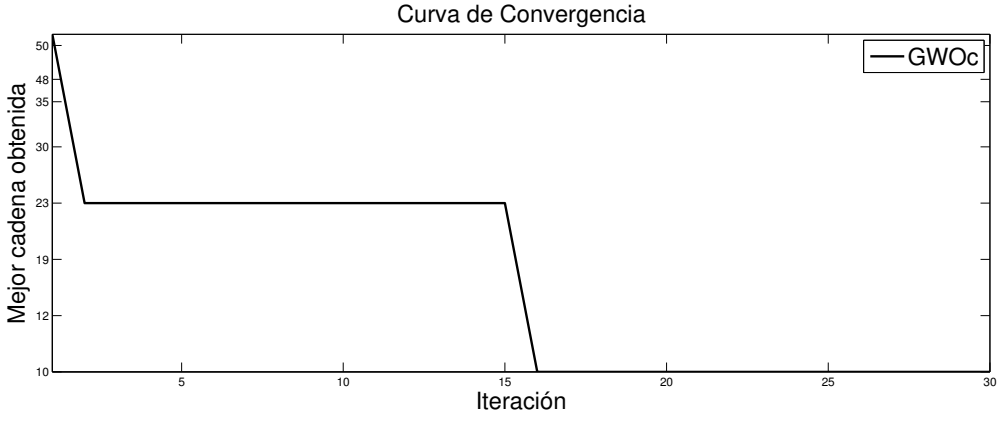

Figura 5.6: Cadena para e=250.

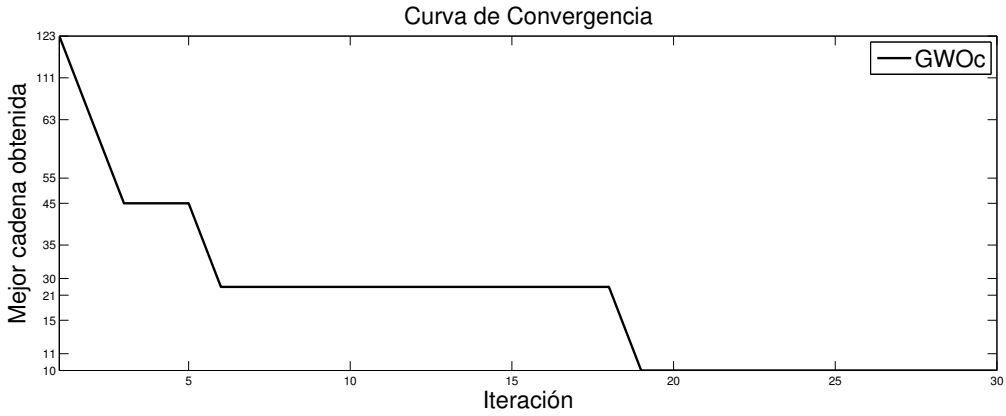

Figura 5.7: Cadena para e=768.

5.3.3 Desempeño de GWOc respecto de otras propuestas del estado el arte para exponentes "difíciles"

El experimento plantea la comparación del desempeño de GWOc respecto de otros abordajes del estado del arte.

La Tabla 5.9 resume los resultados del tercer experimento, correspondiente a los mejores resultados obtenidos por GWOc para un conjunto de exponentes diversos y difíciles de optimizar al igual que los resultados del experimento mostrados en la Tabla 5.4 pero comparando dichos resultados con los obtenidos por AIS [6], PSO [19] y EP [8] para los mismos casos. GWOc ha sido comparado tomando algunos exponentes de las metaheurísticas mencionadas en el párrafo anterior a excepción de [7] pues tal como se menciona en [8], todas las demás ([6], [19]) superan sus resultados. Si bien GWOc no supera los resultados obtenidos para los exponentes listados mediante el uso de los otros enfoques (ver Tabla 5.9) tampoco ha arrojado resultados de menor calidad en términos de longitud de cadena, salvo para el caso del exponente $e = 1087$ empleado en este lote de casos de prueba. Cabe destacar que es correcto que GWOc no supere los valores de longitud de cadena reportados por las otras metaheurísticas con las cuales se comparó en este experimento pues dichos resultados corresponden a la mínima longitud conocida para cada caso, tal como se muestra en la Tabla 5.10 de exponentes "diversos" presentada en [6].

Tabla 5.9: Mejores resultados obtenidos por AIS [6], PSO [19], EP [8] y GWOc para un conjunto de exponentes diversos y difíciles de optimizar.

Exponente		Longitud			
e	Cadena	AIS [6]	PSO [19]	EP [8]	GWOc
5	1, 2, 3, 5	3	3	3	3
7	1, 2, 4, 5, 7	4	4	4	4
11	1, 2, 4, 8, 10, 11	5	5	5	5
19	1, 2 ,4, 8, 16, 18, 19	6	6	6	6
29	1, 2, 3, 6, 7, 14, 28, 29	7	7	7	7
47	1, 2, 4, 8, 9, 18, 36, 45, 47	8	8	8	8
71	1, 2, 3, 6, 7, 14, 21, 35, 70, 71	9	9	9	9
127	1,2, 3, 6, 12, 18, 30, 60, 63, 126, 127	10	10	10	10
191	1, 2, 3, 6, 12, 18, 19, 38, 57, 95, 190, 191	11	11	11	11
379	1, 2, 3, 6, 9, 18, 27, 45, 72, 117, 189, 378, 379	12	12	12	12
607	1, 2, 3, 6, 9, 15, 30, 60, 61, 121, 182, 303, 606, 607	13	13	13	13
1087	1, 2, 3, 6, 9, 18, 36, 54, 108, 216, 324, 540, 541, 1082, 1085, 1087	14	14	14	15

Tabla 5.10: Conjunto de exponentes que tienen una cadena de adición óptima asociada de longitud determinada.

Longitud de cadena	Exponentes (e)
1	2
2	3, 4
3	5, 6, 8
4	7, 9, 10, 12, 16
5	11, 13, 14, 15, 17, 18, 20, 24, 32
6	19, 21, 22, 23, 25, 26, 27, 28, 30, 33, 34, 36, 40, 48, 64
7	29, 31, 35, 37, 38, 39, 41, 42, 43, 44, 45, 46, 49, 50, 51, 52, 54, 56, 60, 65, 66, 68, 72, 80, 96, 128
8	47, 53, 55, 57, 58, 59, 61, 62, 63, 67, 69, 70, 73, 74, 75, 76, 77, 78, 81, 82, 83, 84, 85, 86, 88, 90, 92, 97, 98, 99, 100, 102, 104, 108, 112, 120, 129, 130, 132, 136, 144, 160, 192, 256
9	71, 79, 87, 89, 91, 93, 94, 95, 101, 103, 105, 106, 107, 109, 110, 111, 113, 114, 115, 116, 117, 118, 119, 121, 122, 123, 124, 125, 126, 131, 133, 134, 135, 137, 138, 140, 145, 146, 147, 148, 149, 150, 152, 153, 154, 156, 161, 162, 163, 164, 165, 166, 168, 170, 172, 176, 180, 184, 193, 194, 195, 196, 198, 200, 204, 208, 216, 224, 240, 257, 258, 260, 264, 272, 288, 320, 384, 512

Los siguientes gráficos muestran las curvas de convergencia del algoritmo GWOc para los casos ejecutados en el experimento, conforme el avance de las iteraciones, correspondiente a la ejecución con la cual se obtuvo el valor mínimo de longitud de cadena de adición para cada exponente.
Vale aclarar que en las siguientes gráficas los valores correspondientes al eje de abscisas no parten de cero sino que lo hacen desde el menor valor encontrado de longitud de cadena de adición para cada exponente.

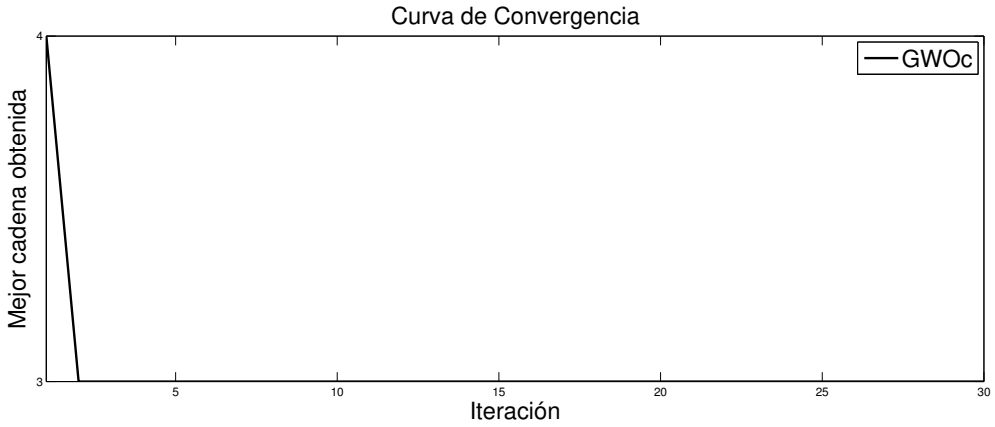

Figura 5.8: Cadena para e=5.

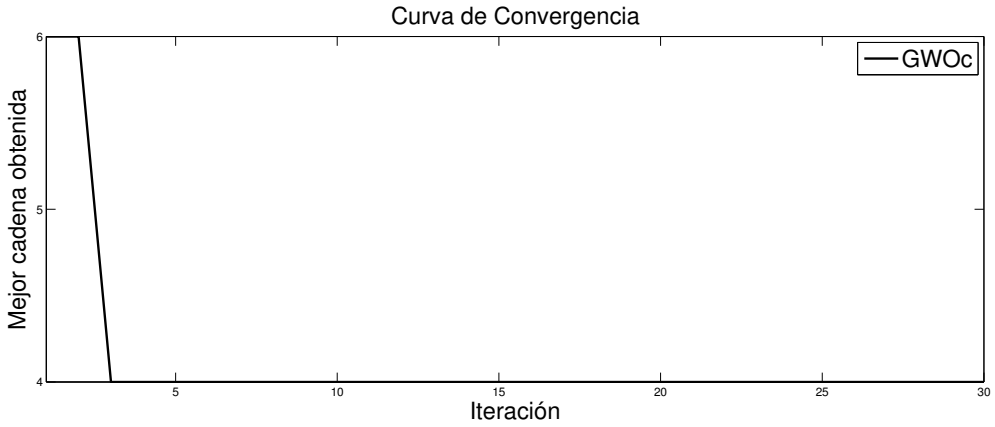

Figura 5.9: Cadena para e=7.

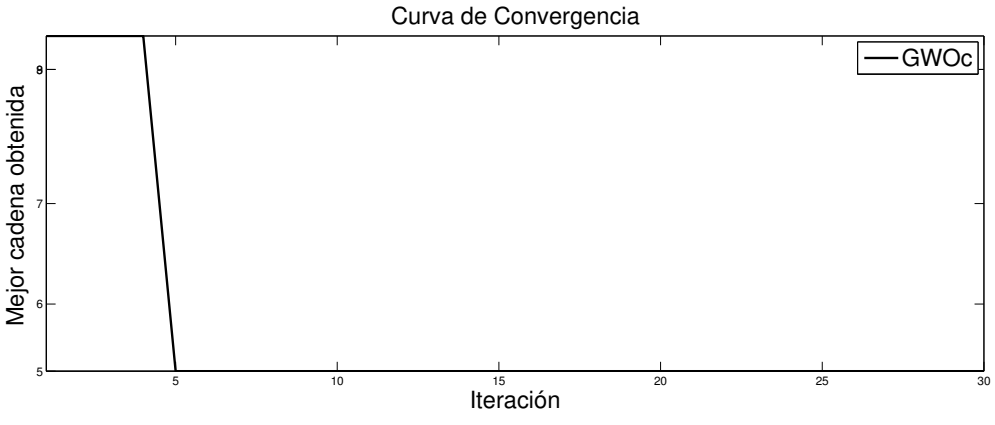

Figura 5.10: Cadena para e=11.

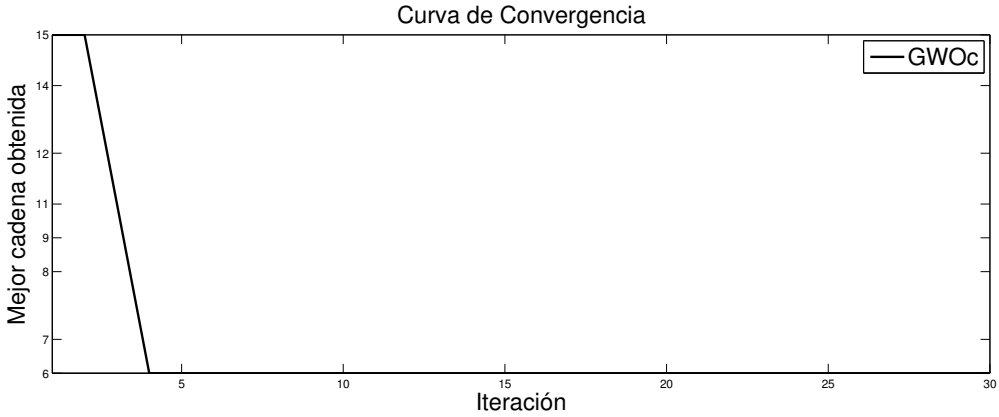

Figura 5.11: Cadena para e=19.

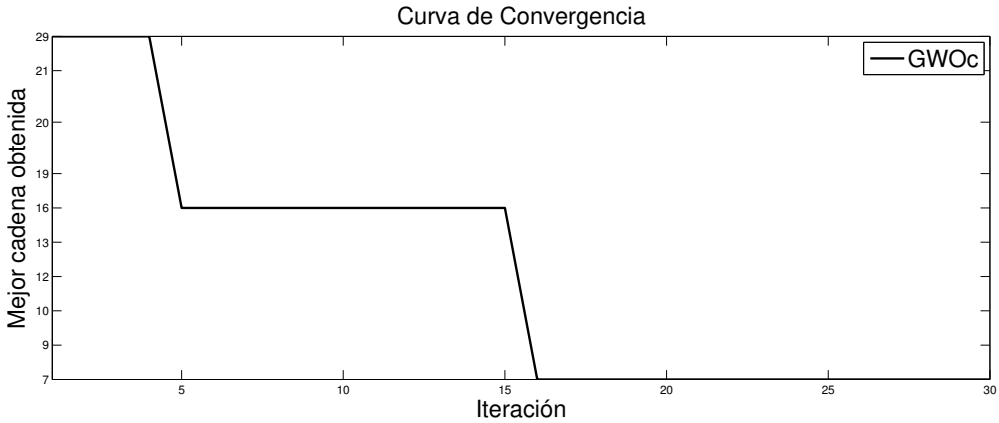

Figura 5.12: Cadena para e=29.

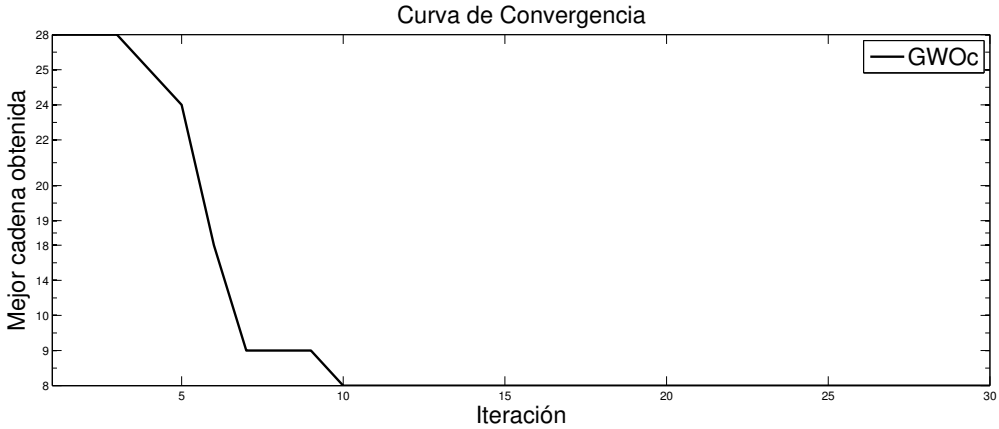

Figura 5.13: Cadena para e=47.

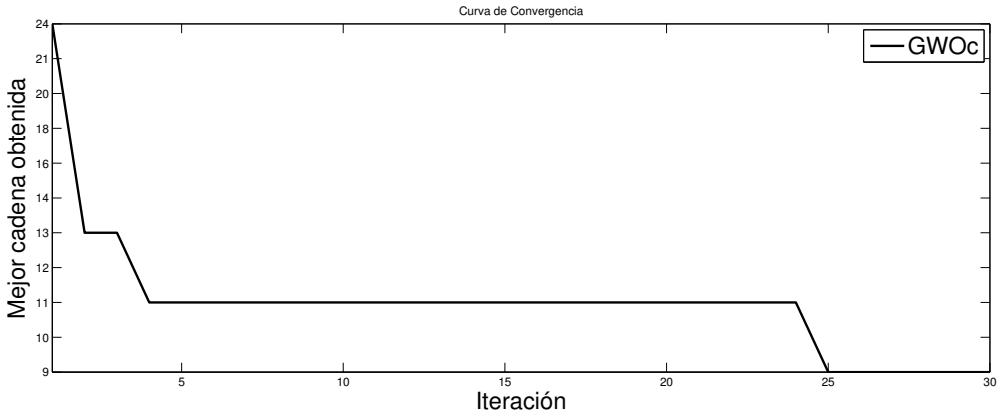

Figura 5.14: Cadena para e=71.

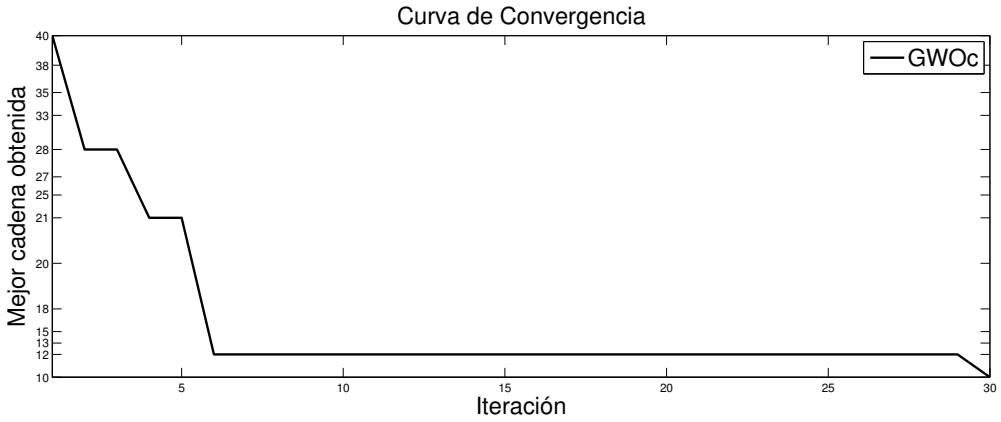

Figura 5.15: Cadena para e=127.

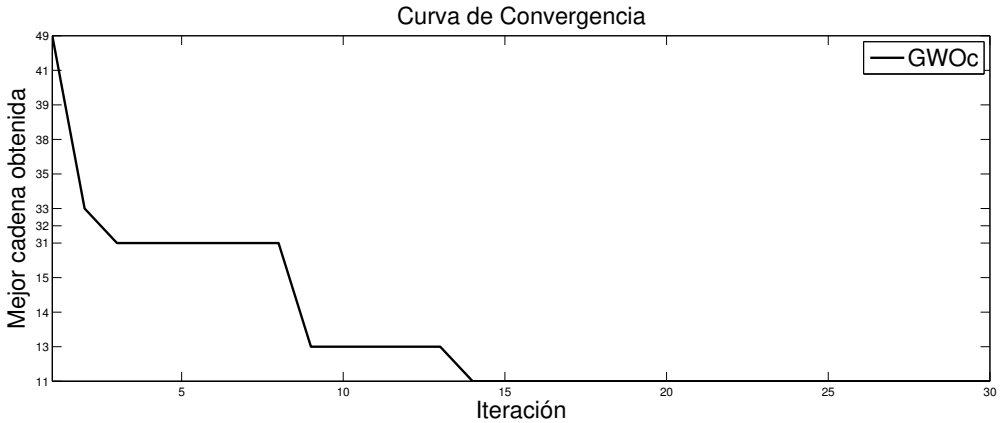

Figura 5.16: Cadena para e=191.

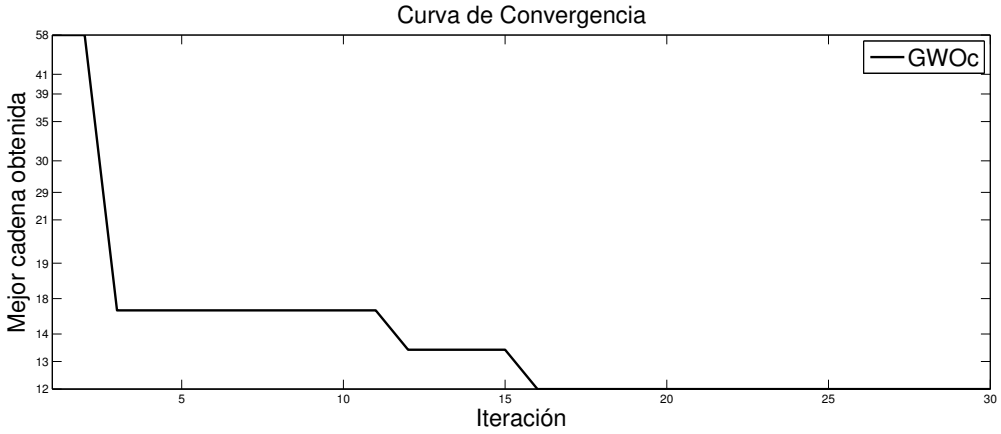

Figura 5.17: Cadena para e=379.

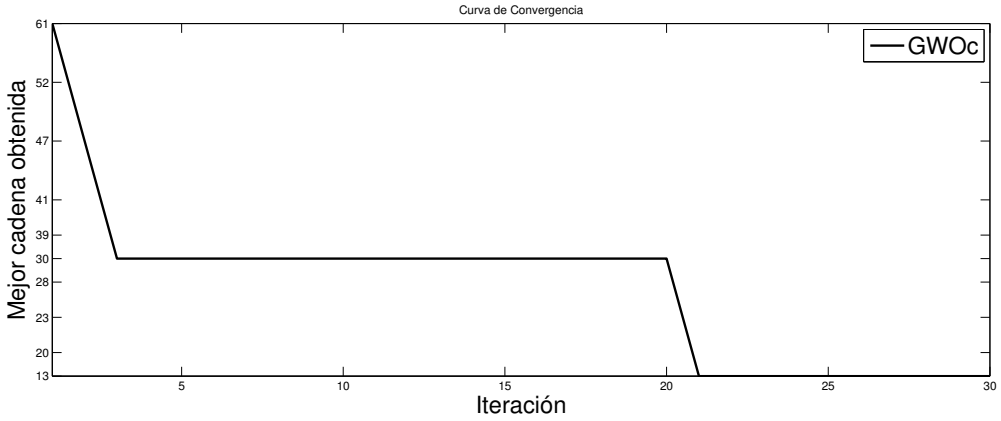

Figura 5.18: Cadena para e=607.

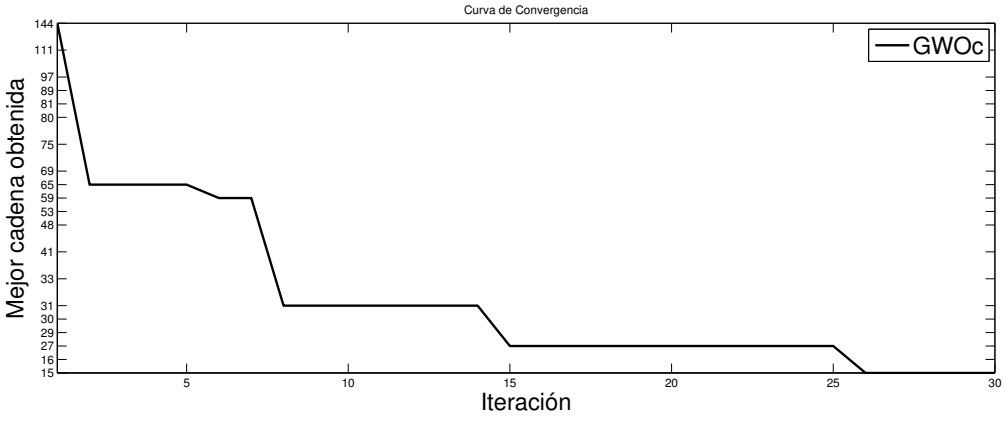

Figura 5.19: Cadena para e=1087.

5.4 Discusión de los resultados

En el experimento de Sección 5.3.1 se comparó GWOc con algunos métodos deterministas del estado del arte, donde el criterio de comparación fueron los resultados de las longitudes acumuladas de cadenas para el mismo rango de exponentes. Se observó que GWOc superó a dos de los métodos citados y con el tercero obtuvo una diferencia despreciable. Por otro lado se comparó también el desempeño de GWOc con métodos estocásticos del estado del arte para las longitudes de cadenas acumuladas para el mismo rango de exponentes, obteniendose resultados similares, respaldando la afirmación con los resultados de pruebas estadísticas. En todos los casos se tomó el mejor resultado de un lote de 30 ejecuciones consecutivas, al igual que en los estudios de la literatura citados.

En el experimento de Sección 5.3.2 se optó por una comparación individual con un trabajo precedente, por dos razones: los casos expuestos en [27] corresponden a exponentes para los que no se posee otro antecedente ya que otros estudios para analizar el desempeño de un método acuden a los exponentes conocidos como dificiles de optimizar, otro motivo es porque la propuesta [27] es una hibridación. Se observa en este experimento superioridad de GWOc sobre [27], desde la simple observación de los resultados y apoyado en las pruebas estadísticas.

En el experimento de Sección 5.3.3 se comparó GWOc con otras 3 metaheurísticas del estado del arte para el problema de referencia, para exponentes difíciles de optimizar. El desempeño de GWOc fue equivalente a las propuestas precedentes salvo por el último caso correspondiente al exponente más grande del escenario planteado. Por otro lado, observando las gráficas de convergencia del experimento para la mejor ejecución de cada exponente (tomada de 30 ejecuciones individuales consecutivas), se puede inferir que a medida que el exponente se incrementa se vuelve complejo hallar la cadena mínima, puesto que en algunos casos se obtuvo cercano a las últimas iteraciones de la ejecución.

Un dato relevante de los experimentos de este capítulo es que en todos los casos los resultados de GWOc se obtuvieron con 300 evaluaciones por ejecución, aunque también vale mencionar que GWOc fue sometido sólo a exponentes pequeños (menores a 128-bits) de acuerdo con el objetivo de la tesis, en tanto que otras propuestas contemplaron escenarios hasta el rango [1,4096-bits]. Sin embargo, se considera que es válido tener en cuenta que GWOc insumió menor cantidad de evaluaciones para alcanzar los mismos resultados en términos de longitud de cadena para varios exponentes "difíciles" también reportados por PSO[19], AIS [6] y EP [8], este último con 92000 evaluaciones. A continuación en la Tabla 5.11 se expone un resumen del número de evaluaciones insumidos por GWOc y otros abordajes.

Tabla 5.11: Desempeño de cada enfoque comparado en términos de cantidad de evaluaciones realizadas

Propuesta	# Evaluaciones
GWOc	300
PSO [19]	300000
AIS [6]	-
EP [8]	92000
GA [7]	30000
GA Annealing [27]	-

Como se observa en la Tabla 5.11, se evidencia una ventaja de GWOc respecto de los otros enfoques con los que se ha comparado la propuesta, en términos de cantidad de evaluaciones, teniendo en cuenta que en todos los experimentos con GWOc se utilizó una configuración de 30 iteraciones y 10 agentes de búsqueda (lobos), dando como resultado 300 evaluaciones por ejecución. Para el caso de la propuesta con PSO [19] según lo reportado se trabajó con una configuración de 30 iteraciones con 1000 partículas, dando un total de 300000 evaluaciones por ejecución de experimento. Por otro lado, EP [8] reportó 92000 evaluaciones, determinadas por la configuración de parámetros del algoritmo y la propuesta con GA [7] empleó 30000 evaluaciones por ejecución para sus experimentos. Las propuestas con AIS [6] y de GA Annealing [27] no reportaron cantidad de evaluaciones.

Capítulo 6

Propuesta para abordar exponentes grandes

6.1 GWOc ventana deslizante - GWOc_SWM

Producto de los escenarios de estudio descriptos en Capítulo 5, se observó que el algoritmo GWOc tal como fue aplicado a rangos reducidos de exponentes no logra obtener cadenas de adición de longitud mínima a medida que los exponentes se incrementan de manera considerable. Esta observación surge producto de pruebas preliminares donde se determinó que GWOc no logra acercarse a los resultados obtenidos por otros métodos para tales rangos de exponentes (del orden de los 128-bits e incluso menores) con la configuración de parámetros mencionada en sección 5.2 u otras probadas.
Luego de un estudio para calibrar los valores de los parámetros A y C, de la metaheurística de referencia, no se lograron mejoras significativas. En función de esto se propone, al igual que en otras propuestas del estado del arte (AIS [6], EP [8]), el método de *ventana deslizante* combinado con GWOc para obtener cadenas de longitud mínima. Ésta nueva propuesta se ha denominado GWOc_SWM ("Grey Wolf Optimizer of chain with Sliding Window Method": "Lobo Gris Optimizador de cadenas con Método de Ventana Deslizante"). El Algoritmo 23 describe dicha estrategia y sus principales caracaterísticas y en la sección 6.3 se exponen los resultados obtenidos con dicho ajuste.

Método de ventana deslizante

Éste método, también llamado de ventana adaptativa [4], ha sido descripto en la sección 4.1.3 y constituye una estrategia determinística para generar cadenas de adición asociadas a exponentes grandes y opera sobre una representación binaria del exponente el cual subdivide en fragmentos (ventanas) de longitud variable, conformadas por elementos nulos (ceros) o no nulos (unos o combinación de unos

y ceros) y en función de algunos parámetros que emplea para definir la longitud de las ventanas no nulas y de los elementos "separadores" o ventanas nulas.

Algoritmo 23 Producir cadena de adición con GWOc_SWM

1: Entrada: e=(e_{m-1}...e_1, e_0), MAX_VDt, q
2: Salida: C = [1; 2;...; m]
3: Pre-computar y dividir e en h ventanas W_i.
4: Obtener ventanas VNC (en decimal) y establecer MVD.
5: Ordenar en forma ascendente y listar las ventanas VNC (decimales) ≠ MVD:
6: List = [W_0; W_1; W_2; ... ; W_{VNC-1}]
7: Obtener U = GWOc (MVD)
8: Seleccionar $x \in U$ de modo que $x > W_{VNC-2}$.
9: Establecer List = [W_0; W_1; W_2; ... ; W_{VNC-1}; x].
10: List = Generador de cadenas de adición (List)
11: Retornar C = List unión U

Donde:

e:	representación binaria de un número entero.
MAX_VDt:	tamaño máximo de ventana.
q:	número máximo de ceros consecutivos en ventanas VNC.
C:	cadena de adición.
W_i:	ventanas del tipo VC y VNC, de longitud L(W_i)
VC:	ventanas binarias de elementos nulos.
VNC:	ventanas binarias de elementos no nulos.
MVD:	valor decimal de la mayor ventana VNC.

El Algorimto 23 subdivide el exponente e (expresado en binario) en h ventanas W_i de las cuales algunas serán tipo VC y otras VNC. Se establecerá MAX_VDt como tamaño máximo de ventana a considerar, de la representación binaria del exponente e. Por otro lado q determina el número máximo de ceros consecutivos permitidos dentro de una ventana de elementos no nulos (VNC) y por tanto también determina separación entre ventanas. Cabe destacar que las ventanas VNCs también admiten q cantidad de ceros dentro de la secuencia pero inician y finalizan con 1. El resultado de las operaciones antes mencionadas da lugar un conjunto de ventanas VCs y VNCs. Sobre las ventanas VNC, se obtendrá el valor decimal asociado a cada una y el mayor será MVD, el resto de las ventanas VNCs expresadas en decimal serán listadas y ordenadas dentro de una cadena intermedia (List). A continuación se obtendrá una cadena de adición (también intermedia) U mediante GWOc con MDV como valor de exponente (parámetro de entrada). De la cadena U se tomará un elemento x de modo tal que el mismo resulte mayor que el valor de ventana W_{VNC-2} de List y será incorporado como última componente de dicha lista. Con la secuencia List como parámetro de entrada se obtiene una cadena de adición aplicando el Algoritmo 24. Finalmente se obtendrá una cadena de adición C, concatenando List con la cadena

intermedia U, vale decir que se aplica una unión, y C será completada aplicando ventana deslizante tal como se describe en [8, 16].

Algoritmo 24 Generador de cadenas de adición

Entrada: una secuencia ordenada de manera ascendente de enteros:
U =(u_0, u_1, ... , u_{n-1}, u_n).
Salida: secuencia de adición W = {e_0, e_1, ... , e_{n-1}, e_n} de longitud L
Establecer $k = n - 2$
Establecer U = [$u_1 = e_1$;$u_2 = e_2$;...;$u_k = e_{n-2}$] ; V= {v_1=e_{n-1},v_2=e_n} ; W = {}
mientras $U \neq \phi$ **hacer**
  $\Delta = (h_2 - h_1)$
  W = W $\cup$ h_2
  Asignar a h_1,h_2 respectivamente los dos valores mayores del conjunto: (u_k, Δ, h_1)
  si $\Delta < u_k$ y $\Delta \notin U$ **entonces**
    Asignar a U= SortSet (U unión Δ) // secuencia ordenada producto de la unión entre U, Δ.
  fin si
  si $\Delta \in U$ **entonces**
    Establecer k= $k - 1$
  fin si
fin mientras
devolver (W)

El Algoritmo 24 (Generador de cadenas de adición) es el mismo empleado en trabajos del estado del arte tales como [6] y [8] y es empleado como método auxiliar por el Algoritmo 23 en el paso 10 para obtener la secuencia intermedia List.

A continuación un ejemplo de aplicación de esta propuesta con un número cercano al orden de los 128-bits, para un valor de $MAX\_VDt = 8$ y $q = 3$. Dado un exponente e=265252859812191058636308480000000 cuya representación binaria es el número que se muestra en la Figura 6.1.

Ejemplo de aplicación de GWOc_SWM

1. Convertir e a representación binaria y establecer máxima ventana VNC con tamaño 8 y $q = 3$. Entonces establecer a partir de lo anterior las ventanas VC y VNC, ver Figura 6.1.

2. Obtener cadena de adición mínima con GWOc, asociada al exponente e, con el valor de la mayor ventana VNC en su expresión decimal.

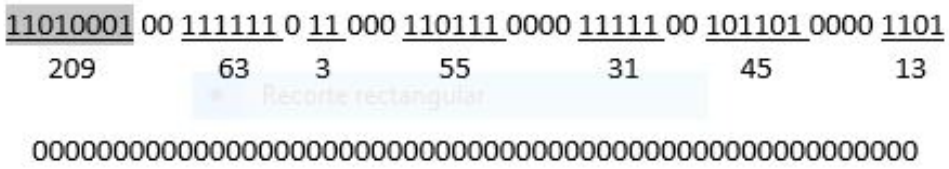

Figura 6.1: División en ventanas de la representación binaria del exponente.

3. MAX_VDt es la mayor ventana VNC (11010001), cuya representación decimal es MDV = 209. Entonces aplicar GWOc para el valor 209 como exponente para hallar la cadena de adición intermedia U.
   U = [1; 2; 3; 5; 8; 16; 24; 40; 64; 104; 208; 209]

4. Obtener lista ordenada de las ventanas VNC menores a MAX_VDt en su representación decimal:
   List = [W_0; W_1; W_2;...; W_{VNC-1}] = [3; 13; 45; 55; 63]

5. Establecer un valor para $x \in U$ tal que $x > W_{VNC-2}$.
   x = 64

6. Actualizar List incorporando el valor x.
   List = [3; 13; 45; 55; 63; 64]

7. Aplicar el Algoritmo 24 a List, obteniéndose:
   List = [5; 7; 8; 13; 15; 23; 31; 39; 47; 55; 63; 64]

8. Establecer C = List Unión U
   C =[1; 2; 3; 5; 7; 8; 13; 15; 16; 23; 24; 31; 39; 40; 47; 55; 63; 64; 104; 208; 209]

Finalmente el procedimiento para completar la cadena C termina aplicando el método de ventana deslizante [8, 16], es decir, recorriendo el exponente e expresado como cadena binaria (ver Figura 6.1), componente a componente, y por cada bit leído se multiplica por 2 al último valor de la cadena C del paso 8, salvo que el elemento de e que se lea en un determinado momento corresponda al bit final de una ventana VNC, entonces al último valor de C se le suma el valor decimal de la ventana a la que pertenece para obtener la nueva componente de C. El resultado de tal procedimiento, será la cadena de adición final, asociada al exponente e.

De las dos variantes de métodos de ventana deslizante expuestos en [17], ambos son métodos m-arios que trabajan sobre la representación binaria del exponente e, el tipo de método de ventana aplicado es aquel que permite ventanas de tamaño variable (tanto si las mismas son de elementos nulos como elementos no nulos). Ésta variante presenta dos parámetros a considerar, el tamaño máximo que tendrá una ventana del

tipo VNC, y el otro parámetro es la cantidad de elementos ceros consecutivos que se admitirán dentro de una ventana VNC, teniendo en cuenta que dichas ventanas pueden tener elementos cero pero no todos ellos lo son.

6.2 Descripción de los experimentos

En esta sección se describe de qué manera se probó el desempeño de la propuesta GWOc_SWM para la búsqueda de cadenas de adición de longitud mínima. Además, se verificó la calidad de las soluciones obtenidas, mediante comparación con trabajos precedentes y pruebas estadísticas.

Se diseñaron dos experimentos, tomando como base para esta actividad los métodos de prueba expuestos en la literatura especializada del tema y adecuando los mismos a rangos de exponentes específicos, en función de los objetivos planteados para esta tesis.

6.2.1 Comparación de GWOc_SWM con propuestas similares para cadenas acumuladas

En este experimento se compara el desempeño de la estrategia descripta en la sección anterior (GWOc_SWM), contra SWM [16], AIS_SWM [6] y EP_SWM [8] donde se empleó también dicha estrategia. Para que dicho experimento sea representativo al igual que en [8] se tomaron 10 exponentes grandes aleatorios de una longitud de hasta 128-bits, correspondiente a la representación binaria de dichos exponentes. Se efectuaron 30 ejecuciones independientes tomando el mejor resultado de las mismas.

Tabla 6.1: Resultado de la mejor ejecución de 10 exponentes del rango de los 128-bits en representación binaria, tomados aleatoriamente:

Casos	Longitud	Promedio
1	139	
2	108	
3	106	
4	203	
5	155	
6	189	
7	173	
8	165	
9	186	
10	141	
Σ	**1565**	**156,5**

La Tabla 6.1 resume los casos correspondientes a la mejor ejecución del experimento de Sección 6.2.1 que para fines prácticos solo fueron enumerados (caso 1 al 10) en función de que se trata de 10 números aleatorios dentro del rango de los 128-bits y considerando su longitud. La columna "longitud" de la tabla corresponde a la longitud de la cadena de adición obtenida para cada caso (exponente) sometido al experimento, considerando que la tabla reune el resultado de la mejor de 30 ejecuciones, es decir, aquella donde se obtuvo menor longitud de cadena para cada uno de los 10 casos. La última columna muestra el promedio de longitud de cadena acumulada para todos los casos de la mejor ejecución.

Tabla 6.2: Comparación de los resultados obtenidos por GWOc_SWM para un rango de 10 exponentes aleatorios, contra SWM, AIS_SWM y EP_SWM:

Long.	SWM [16]		AIS_SWM [6]			EP_SWM [8]			GWOc_SWM		
128 bits	Longitud	k	Longitud	T	q	Longitud	T	q	Longitud	T	q
	156	4	153	17	2	154	8	2	156,5	8	2

En este experimento se comparan los resultados obtenidos por GWOc_SWM, contra sus antecedentes de empleo de la misma estrategia: SWM[16] y las adaptaciones de AIS y EP publicadas en [6] y [8], para un conjunto de 10 exponentes de 128-bits tomados de manera aleatoria, teniendo en cuenta que dicha longitud corresponde a la representación binaria de dichos exponentes a fin de adoptar la misma estrategia que los mencionados antecedentes y que resulte válida la comparación.

De los resultados de la Tabla 6.2 se considera que GWOc_SWM arroja resultados competitivos para exponentes de la magnitud de los 128-bits en representación binaria, comparando el desempeño obtenido con las anteriores propuestas. En la Tabla 6.2, k es el tamaño de la ventana usado por SWM [16]. Los valores T y q para AIS_SWM,

EP_SWM y GWOc_SWM representan tamaño de ventana y cantidad de ceros permitidos en ventanas VNCs respectivamente.

Los valores de T y q utilizados por GWOc_SWM se escogieron contemplando generar un escenario acorde a las propuestas anteriores como el caso de [8] y teniendo en cuenta un estudio previo [17] donde se justifica cuáles son los valores adecuados para estos parámetros de acuerdo con la magnitud de los exponentes a manipular, es decir: $q \in \{1, 2, 3\}$ y T $\in \{4, 5, 6, 7, 8\}$ para exponentes en el rango [128-bits, 2048-bits].

A fin de verificar si existen diferencias significativas entre los resultados de las pruebas de GWOc_SWM con respecto a las otras propuestas, se aplicó la prueba de Kruskal-Wallis, como opción no paramétrica para poder evaluar varias muestras en simultáneo. Teniendo en cuenta que no se posee el detalle de las propuestas anteriores a la presente, salvo los resultados reportados y asumiendo que en los 4 casos ([16],[6],[8] y GWOc_SWM) se trata de muestras independientes pero que procederian de la misma población porque se tomaron los casos del mismo rango numérico.

- Kruskal-Wallis se aplicó con las siguientes hipótesis:

  H_0 (hipótesis nula) no existen diferencias estadísticamente significativas entre las poblaciones de donde se tomaron las muestras para los métodos.
  H_a (hipótesis alternativa) existen diferencias significativas entre las poblaciones de donde se tomaron las muestras para la ejecución de cada método.

- Cálculo del estadístico de prueba.
  Llamaremos H al estadístico de prueba para el método de Kruskall-Wallis que para este caso da como resultado 2,4 y se ha obtenido mediante la siguiente fórmula:

  $$\text{H}=[\tfrac{12}{(n\times(n+1)} \times \textstyle\sum_{i=1}^{k} \tfrac{{R_i}^2}{n_i}] - 3 \times (n+1)$$

  n : cantidad de casos observados.
  n_i : cantidad de casos dentro de cada población.
  R_i: rango asociado a cada caso.
  i: cada población involucrada.
  k: cantidad de poblaciones involucradas.

- Criterio de decisión.
  Teniendo en cuenta que H=2,4, los grados de libertad del escenario ($k-1$=3) y el nivel de significancia empleado (α=0,05), según la tabla de χ^2 (Chi-cuadrado), obtenemos un valor crítico de 7,815.

- Conclusión.
  Puesto que el valor crítico obtenido es mayor que el valor H debemos aceptar

la hipótesis nula y por tanto concluimos en que las poblaciones de donde se tomaron las muestras para cada método utilizado son de idénticas características o todas las muestras proceden de la misma población.

En base a la conclusión y la escasa diferencia entre los resultados de GWOc_SWM y sus antecedentes, se asume que la metaheurística GWOc_SWM mantiene resultados cercanos a otros enfoques, teniendo en cuenta que si bien todas las propuestas han trabajado los experimentos con la misma cantidad de individuos, las muestras han sido tomadas en cada caso de manera aleatoria del conjunto de exponentes del rango [1, 128-bits.]

6.2.2 Desempeño de GWOc_SWM respecto de otras propuestas del estado el arte para exponentes "difíciles"

Al igual que en experimentos anteriores se tomaron una serie de casos especiales, siendo éstos exponentes difíciles de optimizar según la literatura del tema y donde GWOc con los parámetros por defecto no logró superar o alcanzar en todos los casos de pruebas a los obtenidos por otras metaheurísticas del estado de arte. Al igual que en experimento en Sección 6.2.1 se aplicará a dichos exponentes el método GWOc_SWM.

La Tabla 6.3 muestra los resultados de cadenas obtenidas con GWOc_SWM para el conjunto de exponentes difíciles escogidos y compara el desempeño del método con AIS [6], PSO [19], EP [8] y GA [26]. De la lectura de la tabla se observa que en la mejor de 30 ejecuciones para el experimento, GWOc_SWM iguala los resultados de las otras propuestas en términos de longitud de cadena hallada para cada valor de exponente y para el caso del exponente "3585" obtiene una menor longitud de cadena de adición, al igual que EP [8].

Tabla 6.3: Resultados obtenidos con GWOc_SWM para un conjunto de exponentes difíciles de optimizar, respecto de: AIS, PSO, EP y GA.

Exp.		Longitud				
e	Cadena	AIS [6]	PSO [19]	EP [8]	GA[26]	GWOc_SWM
1087	1, 2, 3, 6, 7, 14, 21, 42, 63, 64, 128, 256, 512, 1024, 1087	14	14	14	14	14
1903	1, 2, 3, 5, 10, 15, 30, 60, 90, 180, 360, 720, 1480, 1840, 1900, 1903	15	15	15	15	15
3585	1, 2, 3, 6, 7, 14, 28, 56, 112, 224, 448, 896, 1792, 3584, 3585	16	16	14	-	**14**
6271	1, 2, 3, 5, 10, 20, 30, 31, 60, 120, 240, 480, 960, 1920, 3840, 5760, 6240, 6271	17	17	17	17	17
11231	1, 2, 3, 5, 10, 20, 30, 35, 70, 140, 280, 560, 1120, 2240, 4480, 8960, 11200, 11230, 11231	18	18	18	18	18
34303	1, 2, 3, 5, 10, 15, 30, 60, 63, 126, 252, 504, 1008, 2016, 4032, 8064, 16128, 32256, 34272, 34302, 34303	20	20	20	20	20
65131	1, 2, 3, 5, 10, 15, 30, 60, 120, 240, 255, 480, 960, 1920, 3840, 7680, 14360, 28720, 57440, 65120, 65130, 65131	21	21	21	21	21
110591	1, 2, 3, 6, 12, 24, 30, 60, 120, 240, 360, 720, 1440, 2880, 5760, 11520, 23040, 46080, 92160, 184320, 195840, 196560, 196590, 196591	23	23	23	23	23

6.3 Discusión de los resultados

Puesto que en los experimentos del Capítulo 5 se observó que a medida que los exponentes se incrementan el desempeño de la propuesta GWOc declina, se efectuó una serie de pruebas, incrementando la cantidad de evaluaciones por ejecución hasta el máximo de 300 y el número de agentes de búsqueda (lobos) a 50 y 100 respectivamente, sin observar mejoras significativas. En función de lo anterior se diseño la propuesta GWOc_SWM, inspirada en trabajos precedentes [6, 8, 16].

En experimento de Sección 6.2.1 se comparan los resultados de la propuesta GWOc_SWM con otras estrategias similares del estado del arte (una determinista: SWM y dos estocásticas: AIS_SWM y EP_SWM. La estretegia del experimiento fue tomar 10 exponentes tomados de manera aleatoria dentro de un rango particular y comparar los promedios de cadenas acumuladas para el conjunto, tomando la mejor de 30 ejecuciones.
Se obtuvieron resultados equivalentes de la propuesta GWOc_SWM, con una diferencia no significativa según las pruebas estadísticas. Se asume que los resultados son competitivos tomando en cuenta la aleatoriedad de la elección de los exponentes en un conjunto del rango de los 128-bits y que tanto en GWOc_SWM como en la propuesta del mismo estilo más reciente (EP_SWM), se empleó la misma configuración de ventanas y elementos separadores de las mismas.
Si bien la propuesta EP_SWM [8] trabajó exponentes de hasta los 1024-bits en representación binaria, el trabajo expresa que los mejores resultados se obtuvieron para aquellos de 128-bits, es decir el mismo rango considerado en este experimento de GWOc_SWM y por esto resulta válida la comparación. En adición a lo anterior vale destacar que los resultados de EP_SWM [8] fueron obtenidos con 92000 evaluaciones por ejecución y AIS_SWM [6] no reportó la cantidad de evaluaciones, mientras que GWOc_SWM utilizó 300 evaluaciones por ejecución.

El experimento de la Sección 6.2.1 de la propuesta, se diseñó de la misma manera con el experimento de la Sección 5.3.3 (Capítulo 5), comparando GWOc_SWM con cuatro metaheurísticas de estudios precedentes, para casos conocidos de dificil optimización. Al igual que el experimento de Sección 5.3.3 lo que cada metaheurística revela es que ha obtenido una cadena de la menor longitud posible para cada exponente al que se sometió. Teniendo en cuenta lo anterior se asumen competitivos los resultados también de GWOc_SWM ya que en todos los casos se los ha equiparado con los resultados de las otras propuestas y en uno de ellos junto con EP[8] se ha obtenido el mejor resultado (exponente: 3585). En cada caso se tomó el mejor resultado de un total de 30 ejecuciones indiduales consecutivas de GWOc_SWM.

En relación a la cantidad de evaluaciones por ejecución, como se mencionó anteriormente AIS no reporta, PSO utilizó 300000 evaluaciones, EP 92000 y la propuesta GA [26] utilizó una configuración de 50 ejecuciones independientes por cada experimento y se utilizó el criterio de 100 ejecuciones sin mejora de los resultados, como criterio de finalización. Por otro lado se empleó un número total

de las generaciones a 1500 y tamaño de población de 300 individuos, en todos los experimentos con GA [26]. GWOc_SWM insumió menor cantidad de evaluaciones que los otros enfoques utilizados para la comparación, para la optimización de los exponentes empleados en el experimento descripto en Sección 6.2.2. La Tabla 6.4 refleja dicha situación.

Tabla 6.4: Desempeño de cada enfoque comparado en términos de cantidad de evaluaciones realizadas

Propuesta	# Evaluaciones
GWOc_SWM	300
PSO [19]	300000
AIS [6]	-
EP [8]	92000
GA [26]	450000

Como se observa en la Tabla 6.4, se evidencia una importante ventaja de GWOc_SWM respecto de los otros enfoques, en relación a la cantidad de evaluaciones, teniendo en cuenta que en todos los experimentos con GWOc_SWM se utilizó una configuración de 30 iteraciones y 10 agentes de búsqueda (lobos), dando como resultado 300 evaluaciones por ejecución, mucho menos que las otras propuestas.

Capítulo 7

Propuesta de combinación de GWOc con Búsqueda Local

7.1 GWOc búsqueda local - LGWOc y LGWOc_SWM

En este capítulo se propone una mejora para GWOc que se denominó LGWOc ("Local-Grey Wolf Optimizer of Chains"), basada en el uso de *Búsqueda Local* como el caso del algoritmo Hill-Climbing [21] y la estrategia de introducir información al proceso de búsqueda para mejorar el desempeño del algoritmo GWO aplicado a optimización de cadenas de adición (GWOc). También se combinó LGWOc con ventana deslizante, algoritmo denominado LGWOc_SWM "Local-Grey Wolf Optimizer of Chains with Sliding Window Method".

El Algoritmo 25 describe el esquema básico del método de Búsqueda Local:

Algoritmo 25 Búsqueda Local

1: Generar una solución inicial X.
2: **mientras** X no es un óptimo local **hacer**
3: **si** X' $\in N(X)$ con f(X) < f(X') **entonces**
4: $X = X$'
5: **fin si**
6: **fin mientras**
7: **devolver** X

Donde:

X: solución incial.
X': mejor solución dentro del vecindario de X.
$N(X)$: vencidario de X.

LGWOc emplea al Algoritmo 25 como un operador de búsqueda local para intentar mejorar la convergencia y evitar la tendencia de GWO al estancamiento en óptimos locales. Tal operador será un método de búsqueda local que tomará como parámetro de entrada (solución inicial) la mejor solución actual Xα de GWOc y buscará iterativamente hasta no satisfacer un criterio de finalización, por una solución mejor en el vecindario de Xα, utilizando como criterio de comparación para esto la longitud de cadena (función de fitness o adaptación) siendo la mejor solución aquella cadena con menor longitud. En caso de no poder mejorar el valor de solución, el proceso de búsqueda culminará retornando el mismo valor Xα que intentaba mejorar. Se emplea un parámetro que establece la cantidad de ejecuciones del operador de búsqueda local por iteración.
Hill Climbing [17] (Algoritmo 25) utilizado como operador de búsqueda local por LGWOc, es un algoritmo que generalmente opera comenzando con una solución elegida al azar pero en esta implementación el valor inicial será Xα de GWOc utilizado como parámetro de entrada. Dicha solución, denominada "padre" es mutada, generando otra denominada "hijo". De estas dos soluciones se escoge aquella que tenga mejor aptitud para permanecer en la población, para el presente caso considerando que cada posible solución es un cadena de adicion asociada a un exponente en particular, será la cadena de menor longitud.

En cada iteración, un nuevo punto es seleccionado de la vecindad del punto actual. Hill climbing es una técnica que permite ajustar un único elemento en cada iteración y determina si el cambio mejora el valor de la función objetivo o no (ver Algoritmo 25). De esta manera, cualquier cambio que mejore f(X) es aceptado por el algoritmo y el proceso continúa hasta que no se pueda encontrar un cambio que mejore el valor de dicha función objetivo [21]. En caso de no encontrar mejoras factibles el Algoritmo 25 devolverá el mismo valor que aceptó como parémtro de entrada para iniciar el proceso de optimización. En esencia LGWOc opera de manera análoga a GWOc con la diferencia que intentará mejorar Xα mediante el empleo del operador de búsqueda local (paso 12, Algoritmo 24) y en caso de no poder hacerlo mantendrá el Xα ya conocido para ese ciclo de ejecución, ésto a fin de no provocar que el método ingrese en un ciclo infinito y no pueda finalizar la ejecución del Algoritmo 24.

Puesto que esta hibridación está planteada para mejorar el desempeño general de GWO adaptado para obtener cadenas de adición de longitud mínima (GWOc), en los experimentos ejecutados con la nueva propuesta LGWOc, se utilizará la misma configuración empleada en GWOc:

- Agentes_de_búsqueda= 10 (número de lobos).
- Máximo_iterationes= 30 (cantidad de ejecuciones).

Algoritmo 26 LGWOc.

Inicializar población de lobos X_i (con $i = 1, 2, ..., n$)
Inicializar α, A y C
Calcular aptitud (fitness) de cada agente de búsqueda
Determinar los 3 mejores agentes búsqueda:
$X\alpha =$ el mejor agente.
$X\beta =$ el segundo mejor agente.
$X\delta =$ el tercer mejor agente.
mientras ($t <$ número máximo de iteraciones) **hacer**
  para $i = [1..n]$ **hacer**
    Actualizar la posición de cada agente mediante la ecuación (7.3c)
  fin para
  Elegir la mejor solución actual $X\alpha$ como punto de partida y generar una nueva solución $X'\alpha$ mediante el Algoritmo 25.
  si $X'\alpha < X\alpha$, **entonces**
    reemplazar $X\alpha$ con $X'\alpha$ // $X\alpha = X'\alpha$
  si no
    iterar desde el paso 12.
  fin si
  Actualizar α, A y C
  Calcular el fitness de todos los agentes de búsqueda
  Actualizar X_α, X_β y X_δ
  $t = t + 1$
fin mientras
Retornar $X\alpha$

Donde:

$$D = |C \cdot X_p(t) - X(t)| \tag{7.1a}$$
$$X(t+1) = X_p(t) - A \cdot D \tag{7.1b}$$

$$A = 2a \cdot r_1 - a \tag{7.2a}$$
$$C = 2 \cdot r_2 \tag{7.2b}$$

$$D_\alpha = |C_1 \cdot X_\alpha - X|, D_\beta = |C_2 \cdot X_\beta - X|, D_\delta = |C_3 \cdot X_\delta - X| \tag{7.3a}$$
$$X_1 = X_\alpha - A_1 \cdot (D_\alpha), X_2 = X_\beta - A_2 \cdot (D_\beta), X_3 = X_\delta - A_3 \cdot (D_\delta) \tag{7.3b}$$
$$X(t+1) = (X_1 + X_2 + X_3)/3 \tag{7.3c}$$

7.2 Descripción de los experimentos

En esta sección se describe de qué manera se probó el desempeño de las propuestas LGWOc y LGWOc_SWM.
Puesto que el objetivo es verificar si la estrategia de incoporar un mecanismo de búsqueda local favorece el desempeño de GWOc (algoritmos LGWOc y LGWOc_SWM) no se diseñaron nuevos experimentos, sinó que se ejecutaron los mismos ya propuestos para GWOc y GWOc_SWM en los Capítulos 5 y 6 respectivamente, a fin de poder evaluar los puntos de mejora.

7.2.1 Longitudes acumuladas de LGWOc para exponentes en [1, 512]

El primer experimento planteado es análogo al experimento de la Sección 5.3.1 y radica en calcular el total de longitudes acumuladas de cadenas aditivas para el conjunto de exponentes [1, 512], a fin de comparar el desempeño de LGWOc con los resultados obtenidos por otros métodos deterministas de la literatura y con el propio GWOc, para el mismo rango de exponentes (ver Tabla 7.1).

Tabla 7.1: Cadenas de adición acumuladas para todas las longitudes de exponentes $e \in [1, 512]$

e	$[1, 512]$
Optimal [6]	4924
Binary [6]	5388
Çuaternary [6]	5226

Resultados GWOc

Longitudes acumuladas	4994
Promedio	9,77
Mediana	10
Desvío	1,95

Resultados LGWOc

Longitudes acumuladas	4934
Promedio	9,65
Mediana	10
Desvío	1,89

La Tabla 7.1 resume los resultados del experimento donde se observa una leve mejora de LGWOc respecto de GWOc para el valor de cadena acumulada (4934) para todos los exponentes del rango [1, 512], acercándose más a lo reportado por la estrategia

determinista “Optimal” [6].

Por otro lado también fue comparado el valor mínimo de cadena acumulada obtenido con LGWOc respecto de las propuestas estocásticas del estado del arte y con el propio GWOc (ver Tabla 7.2), donde la mejor ejecución de LGWOc reportó resultados mejores a GWOc y más cercanos a las otras propuestas antecesoras.

Tabla 7.2: Comparación de los mejores resultados acumulados para LGWOc respecto de otros enfoques, para el mismo rango de exponentes.

$e \in [1, 512]$	AIS	GA	PSO	EP	GWOc	LGWOc
	4924	4924	—	4924	4994	4934

Además del experimento para cadenas acumuladas se planteó otro, con la estrategia de combinación de ventana deslizante con LGWOc, de manera análoga a como se describió en el Capítulo 6 pero empleando LGWOc en reemplazo de GWOc, cuyos resultados se describen en la Sección 7.2.2.

7.2.2 Comparación de LGWOc_SWM con propuestas similares para cadenas acumuladas

En este experimento se compara el desempeño de LGWOc_SWM, contra SWM [16], AIS_SWM [6], EP_SWM [8] y GWOc_SWM, donde se empleó también dicha estrategia. Para que dicho experimento sea representativo al igual que en [8] se tomaron 10 exponentes grandes aleatorios de una longitud de hasta 128-bits, correspondiente a la representación binaria de dichos exponentes. Se efectuaron 30 ejecuciones independientes tomando el mejor resultado de las mismas.

Tabla 7.3: Comparación de los resultados obtenidos por LGWOc_SWM para un rango de 10 exponentes aleatorios, contra: SWM, AIS_SWM, EP_SWM y GWOc_SWM:

Exp.	SWM		AIS_SWM			EP_SWM			GWOc_SWM			LGWOc_SWM		
128-bits	Long.	k	Long.	T	q	Long	T	q	Long.	T	q	Long.	T	q
	156	4	153	17	2	154	8	2	156,5	8	2	154,6	8	2

Los resultados de la Tabla 7.3 verifican que en la mejor de 30 ejecuciones, LGWOc_SWM supera tanto a GWOc_SWM como a [16], siendo ésta última la que reporta mejor desempeño dentro de los antecedentes citados. Cabe destacar que LGWOc_SWM supera a GWOc_SWM empleando la misma configuración de T (tamaño de mayor ventana) y q (cantidad de ceros consecutivos admitidos dentro

de ventana VNC), justificando el empleo de tales parámetros en función del rango numérico al que pertenecen los exponentes aleatorios y a lo expuesto en Seccion 6.2.1.

7.2.3 LGWOc vs. GWOc y GA Annealing para exponentes específicos

El experimento plantea la comparación del desempeño de LGWOc respecto de una propuesta reciente [27] del estado del arte y del propio GWOc para un conjunto de exponentes específicos.

La Tabla 7.4 muestra una comparación de los resultados de LGWOc respecto de GA Annealing [27] y GWOc, para una serie de exponentes diversos y especialmente difíciles de optimizar. Se considera dentro de esta categoría a aquellos exponentes para los cuales no es posible hallar cadenas de longitud mínima mediante los métodos deterministas. Se puede observar que en los 5 casos, LGWOc obtuvo cadenas de longitud igual o menor al enfoque propuesto en [27] e igualó los resultados de GWOc, incluso superó al mismo en el caso de exponente 130. Si bien los métodos aplicados en [6], [19] y [8] también han sido probados con esta clase de exponentes diversos, no son los mismos que los empleados en [27]. Por esta razón, se efectúa la comparación en Tabla 7.4 y en el experimento de la Sección 7.2.4 se lo hace respecto de los otros métodos del estado del arte.
Las figuras 7.1, 7.2, 7.3, 7.4 y 7.5, muestran las curvas de convergencia para cada uno de los casos sometidos a la propuesta LGWOc.

Tabla 7.4: Comparación de GA Annealing [27], GWOc y LGWOc para el mismo conjunto de exponentes.

Exp.	Cadena	GA Annealing	GWOc	LGWOc
e		Longitud		
23	1, 2, 4, 5, 10, 20, 21, 23	7	7	7
55	1, 2, 3, 6, 12, 24, 27, 54, 55	9	8	8
130	1, 2, 4, 8, 16, 32, 64, 128, 129, 130	11	9	**8**
250	1, 2, 3, 5, 10, 20, 30, 50, 100, 150, 250	13	10	10
768	1, 2, 3, 6, 12, 24, 48, 96, 192, 384, 768	23	10	10

Los siguientes gráficos muestran las curvas de convergencia del algoritmo LGWOc para los casos ejecutados en el experimento, conforme el avance de las iteraciones, correspondiente a la ejecución con la cual se obtuvo el valor mínimo de cadena para cada exponente. A fin de visualizar la mejora de LGWOc sobre GWOc respecto de la cantidad de evaluaciones se presenta las curvas de convergencia de ambas propuestas para cada exponente tratado en el experimento. Vale aclarar que en las siguientes gráficas los valores correspondientes al eje de abscisas no parten de cero sino que lo hacen desde el menor valor encontrado de longitud de cadena de adición para cada exponente.

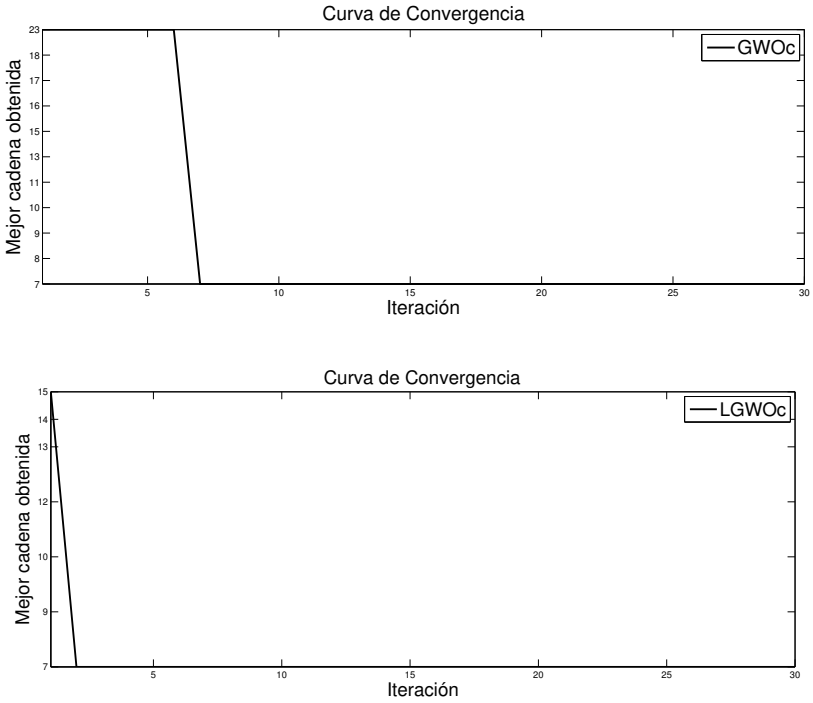

Figura 7.1: Cadena para e=23 para GWOc y LGWOc.

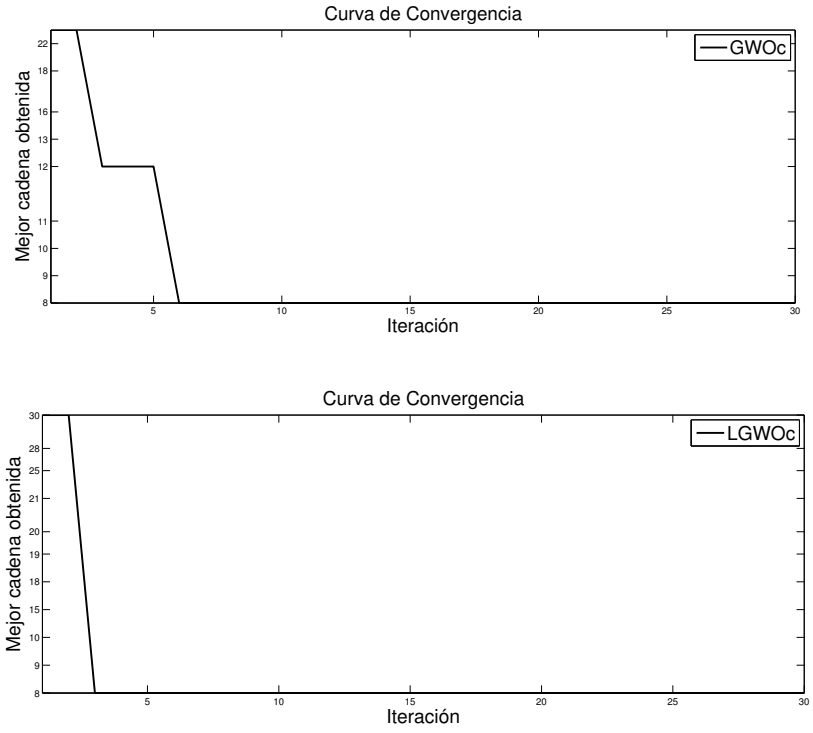

Figura 7.2: Cadena para e=55 para GWOc y LGWOc.

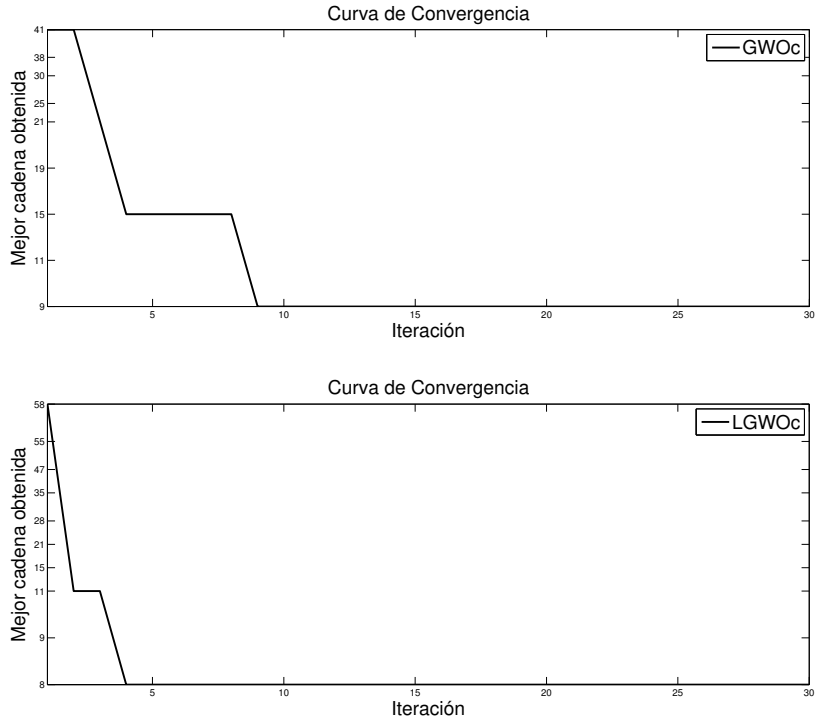

Figura 7.3: Cadena para e=130 para GWOc y LGWOc.

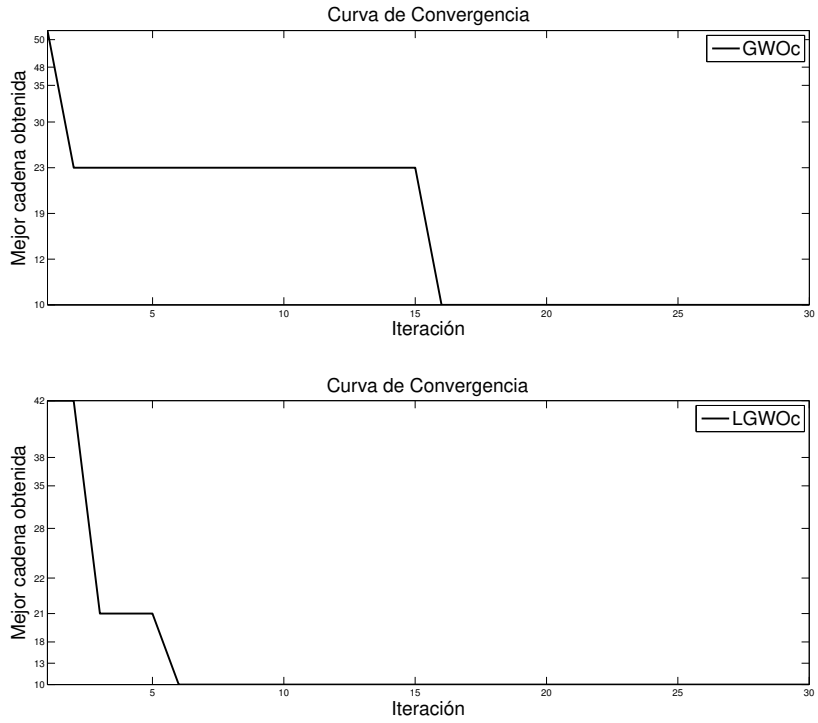

Figura 7.4: Cadena para e=250 para GWOc y LGWOc.

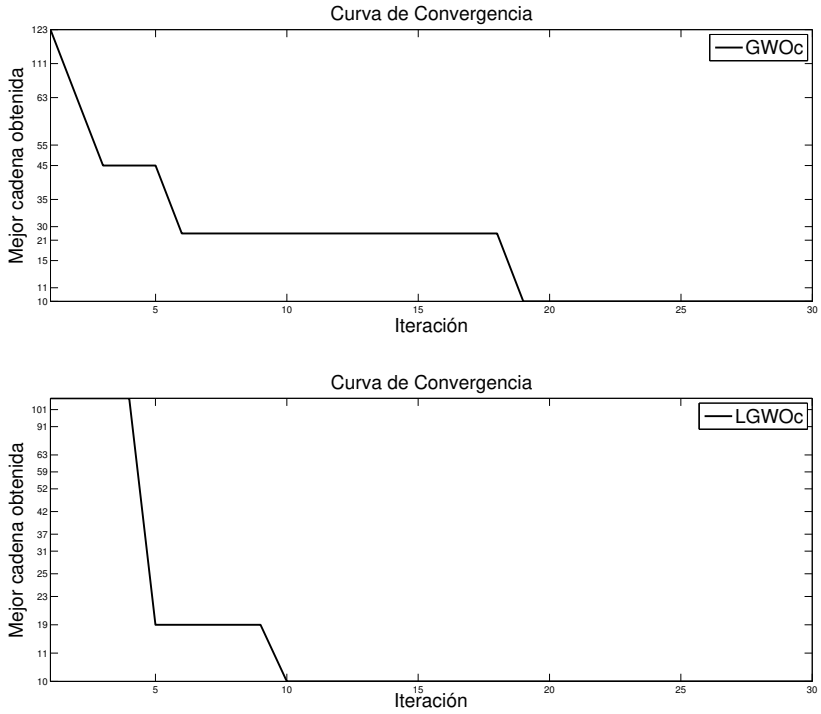

Figura 7.5: Cadena para e=768 para GWOc y LGWOc.

7.2.4 Desempeño de LGWOc respecto de GWOc y otras propuestas del estado el arte para exponentes "difíciles"

El experimento plantea la comparación del desempeño de LGWOc respecto de GWOc y otros abordajes del estado del arte para el mismo conjunto de exponentes empleado en el experimento de la Sección 5.3.3.

La Tabla 7.5 resume los resultados del experimento, donde se observa que LGWOc obtuvo resultados idénticos a GWOc y otras metaheurísticas del estado del arte, para el conjunto de exponentes evaluados, incluso superando el desempeño de GWOc para el exponente 1087 donde dicha propuesta no logró obtener la longitud óptima de cadena.

Tabla 7.5: Mejores resultados obtenidos por AIS [6], PSO [19], EP [8], GWOc y LGWOc para un conjunto de exponentes "difíciles".

Exp.		Longitud				
e	Cadena	AIS [6]	PSO [19]	EP [8]	GWOc	LGWOc
5	1, 2, 3, 5	3	3	3	3	3
7	1, 2, 4, 5, 7	4	4	4	4	4
11	1, 2, 4, 8, 10, 11	5	5	5	5	5
19	1, 2 ,4, 8, 16, 18, 19	6	6	6	6	6
29	1, 2, 3, 6, 7, 14, 28, 29	7	7	7	7	7
47	1, 2, 4, 8, 9, 18, 36, 45, 47	8	8	8	8	8
71	1, 2, 3, 6, 7, 14, 21, 35, 70, 71	9	9	9	9	9
127	1,2, 3, 6, 12, 18, 30, 60, 63, 126, 127	10	10	10	10	10
191	1, 2, 3, 6, 12, 18, 19, 38, 57, 95, 190, 191	11	11	11	11	11
379	1, 2, 3, 6, 9, 18, 27, 45, 72, 117, 189, 378, 379	12	12	12	12	12
607	1, 2, 3, 6, 9, 15, 30, 60, 61, 121, 182, 303, 606, 607	13	13	13	13	13
1087	1, 2, 3, 6, 9, 18, 36, 54, 108, 216, 324, 540, 541, 1082, 1085, 1087	14	14	14	15	**14**

A continuación se pueden observar las curvas de convergencia para cada uno de los casos tratados en el experimento bajo la propuesta LGWOc. A fin de visualizar la mejora de LGWOc sobre GWOc respecto de la cantidad de evaluaciones se presenta las curvas de convergencia de ambas propuestas para cada exponente tratado en el experimento.
Vale aclarar que en las siguientes gráficas los valores correspondientes al eje de abscisas no parten de cero sino que lo hacen desde el menor valor encontrado de longitud de cadena de adición para cada exponente.

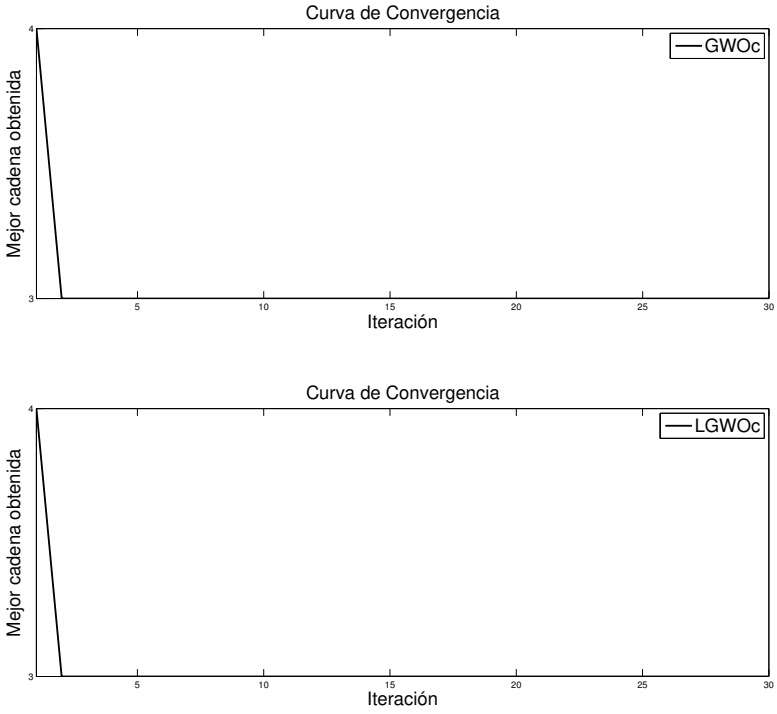

Figura 7.6: Cadena para e=5 para GWOc y LGWOc.

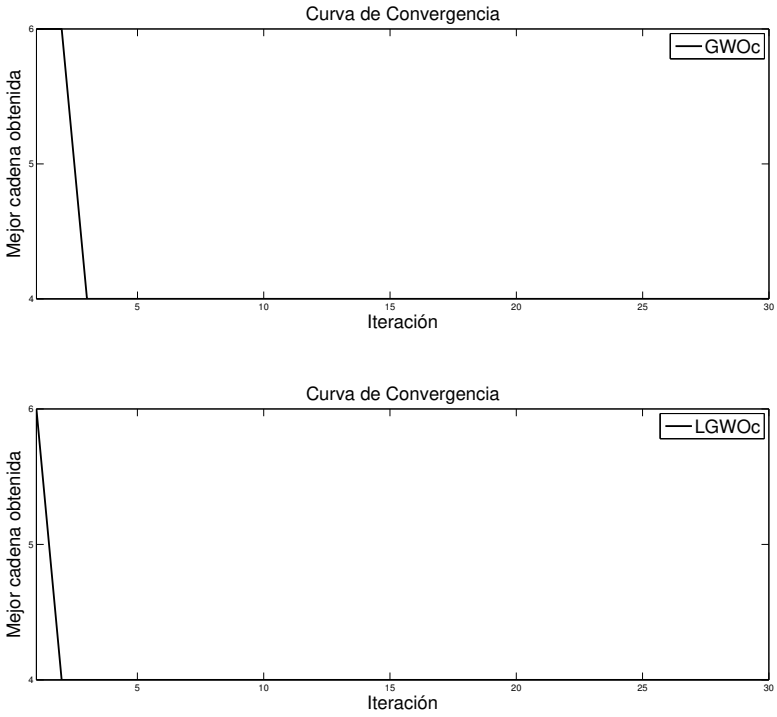

Figura 7.7: Cadena para e=7 para GWOc y LGWOc.

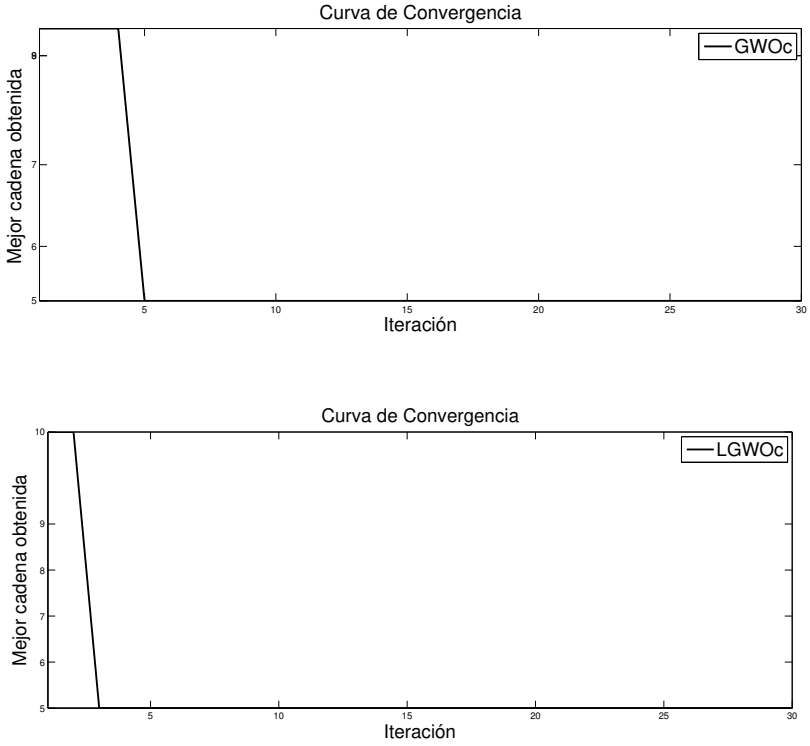

Figura 7.8: Cadena para e=11 para GWOc y LGWOc.

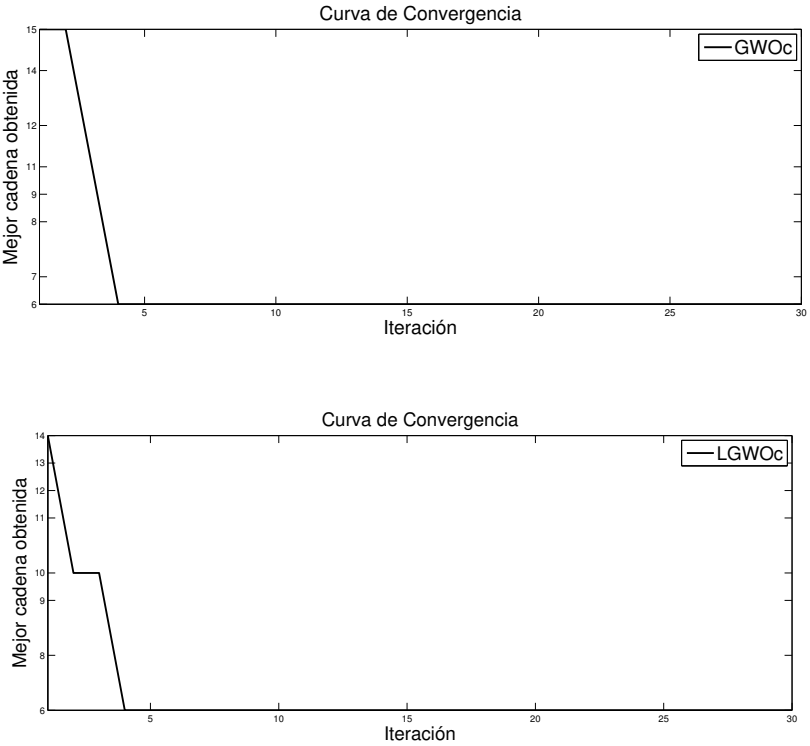

Figura 7.9: Cadena para e=19 para GWOc y LGWOc.

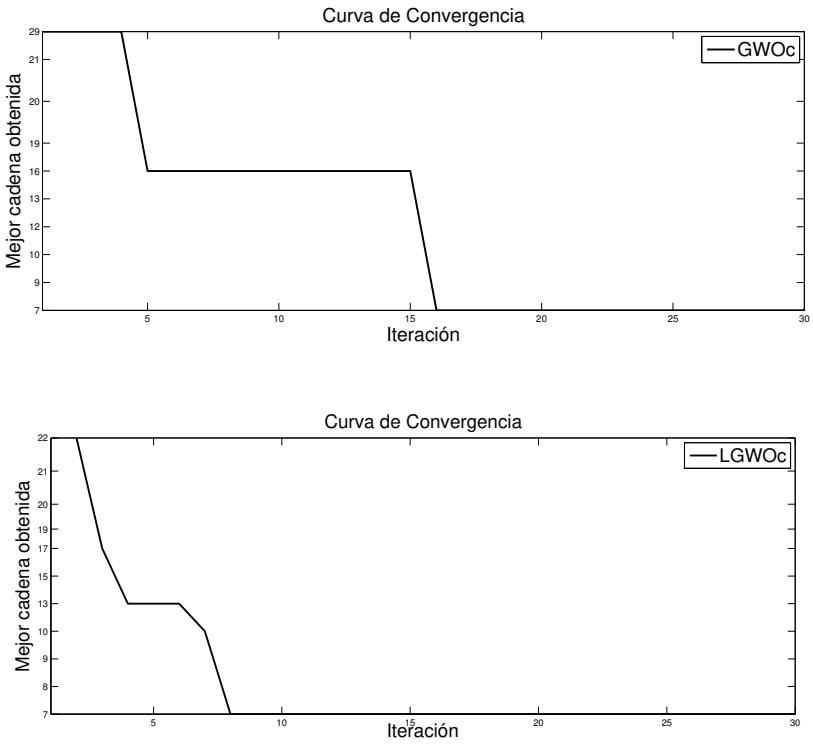

Figura 7.10: Cadena para e=29 para GWOc y LGWOc.

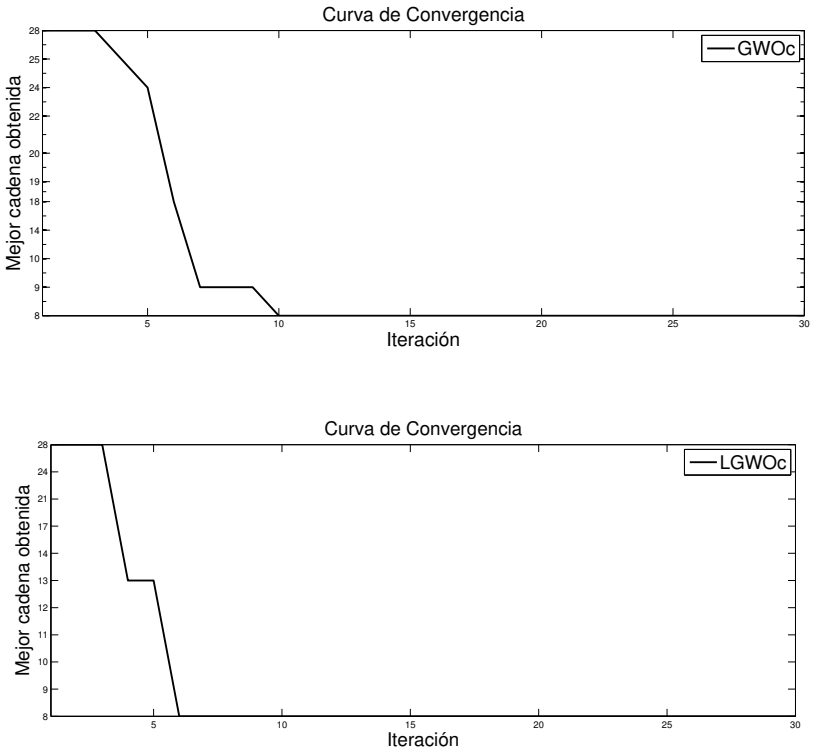

Figura 7.11: Cadena para e=47 para GWOc y LGWOc.

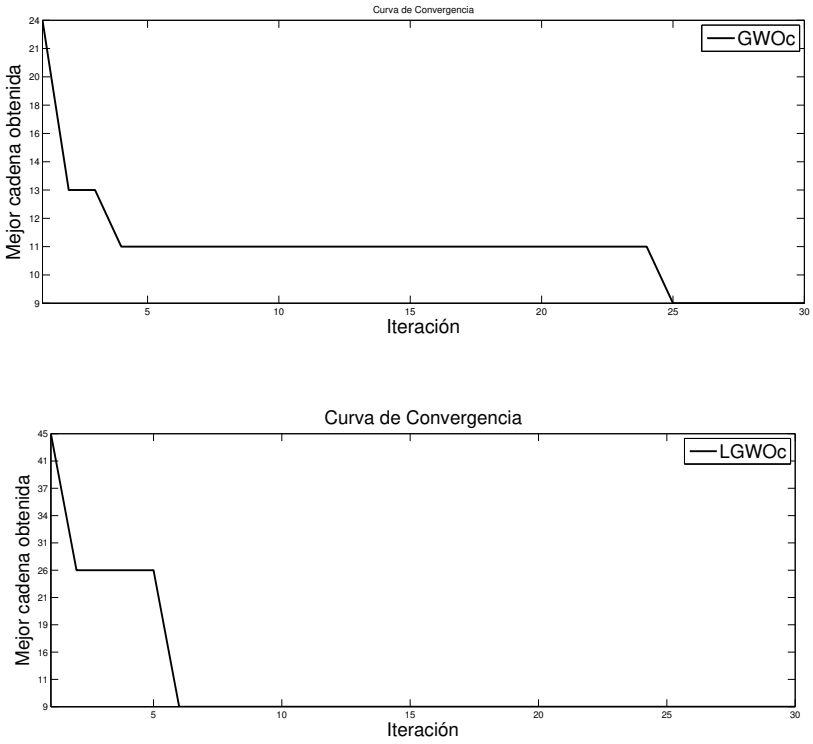

Figura 7.12: Cadena para e=71 para GWOc y LGWOc.

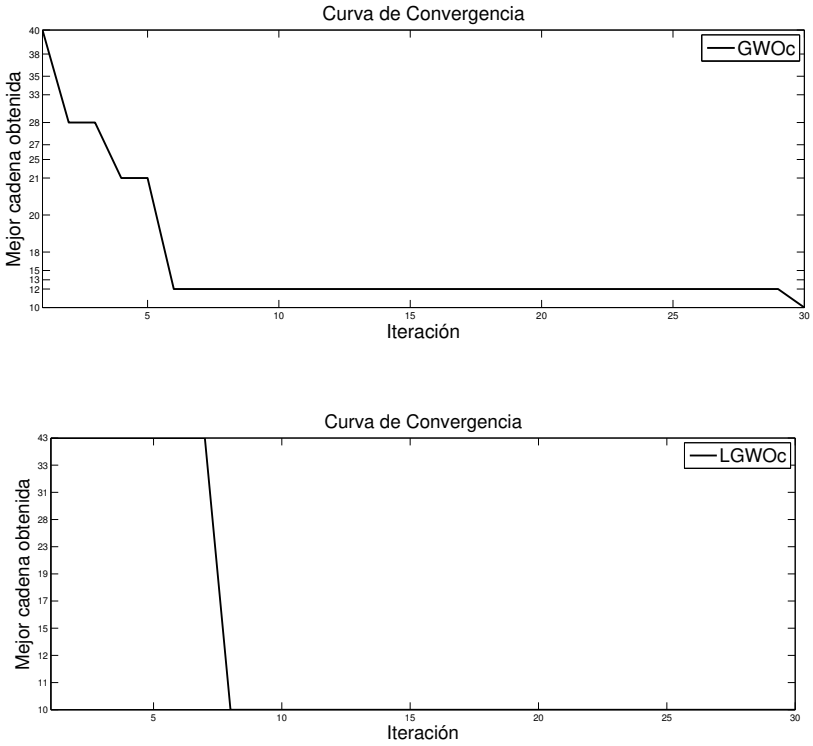

Figura 7.13: Cadena para e=127 para GWOc y LGWOc.

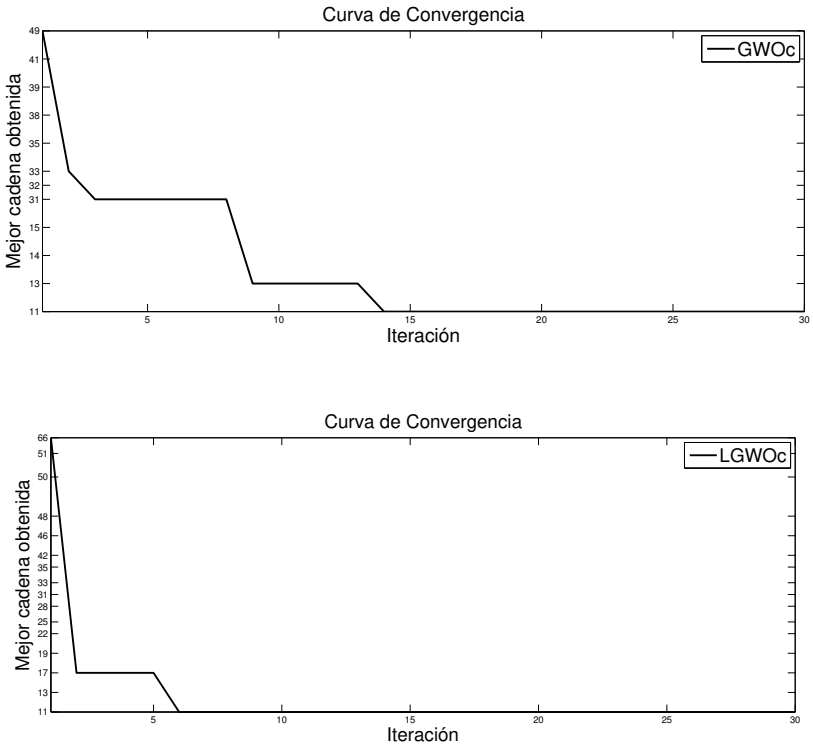

Figura 7.14: Cadena para e=191 para GWOc y LGWOc.

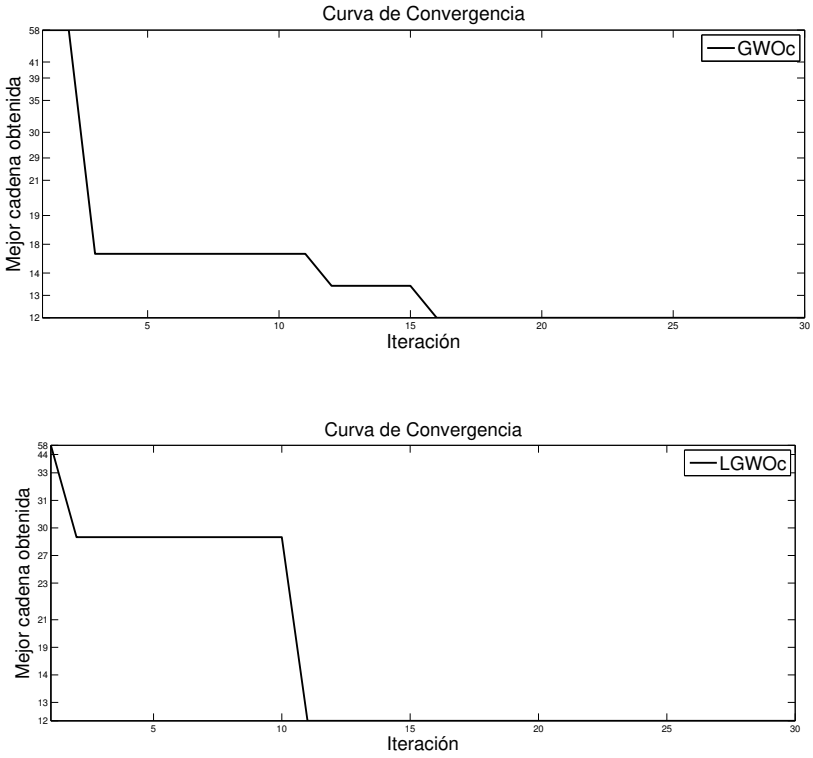

Figura 7.15: Cadena para e=379 para GWOc y LGWOc.

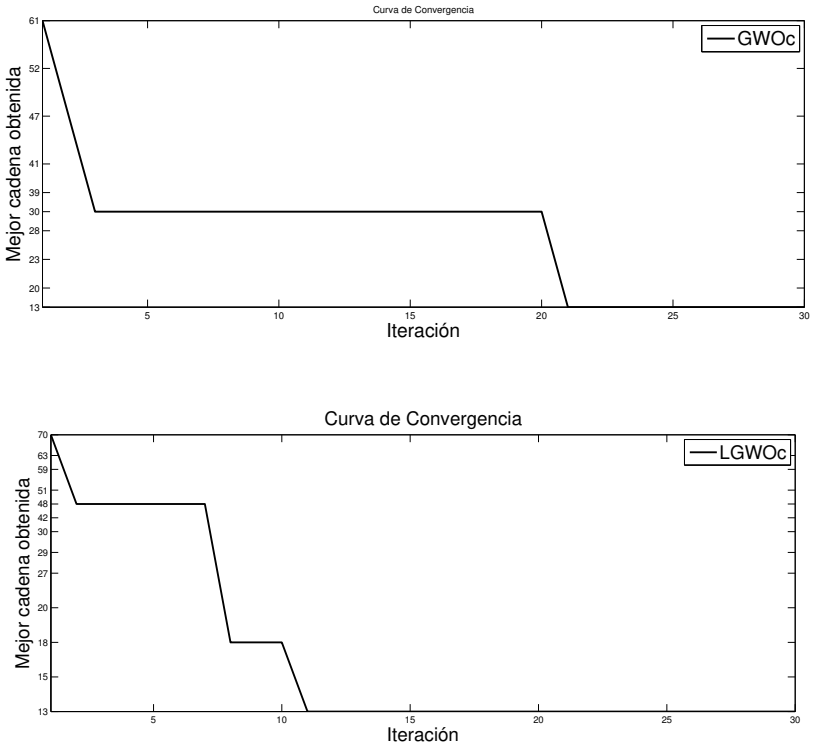

Figura 7.16: Cadena para e=607 para GWOc y LGWOc.

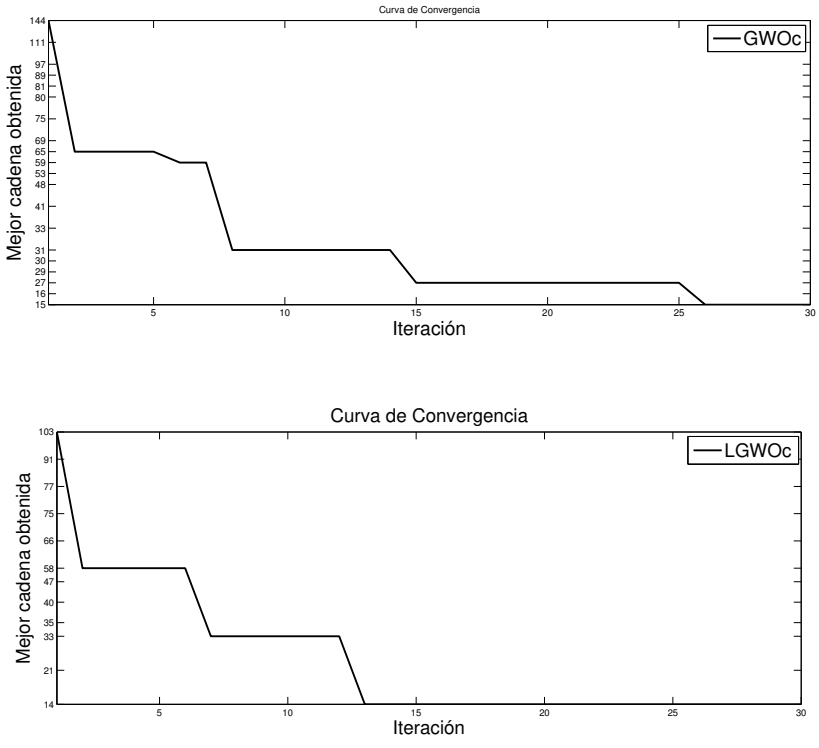

Figura 7.17: Cadena para e=1087 para GWOc y LGWOc.

7.3 Discusión de los resultados

Los experimentos planteados en este capítulo son análogos a los descriptos en el los Capítulos 5 y 6. Los experimentos de Secciones 7.2.1 y 7.2.2, pretenden evidenciar mejoras en el desempeño de LGWOc y LGWOc_WSM, respecto de GWOc y GWOc_SWM, comparando estas propuestas teniendo en cuentan que comparten el objetivo de comparar longitutes de cadenas acumuladas para ciertos rangos de exponentes.
En el experimento de Sección 7.2.1 se compara el rendimiento de LGWOc para hallar longitudes mínimas de cadenas acumuladas en el rango [1,512], donde se evidenció una mejora de LGWOc respecto de GWOc, ya que la longitud total acumulada para cadenas del rango de referencia estuvo más cercana al total acumulado del método determinista "Optimal" que GWOc, y por encima del resto de los otros métodos comparados.
El experimento de Sección 7.2.2 plantea demostrar la mejora en longitudes totales de cadenas acumuladas para exponentes en representación binaria de hasta una longitud de 128-bits, respecto de GWOc, para el mismo rango y tomando 10 exponentes aleatorios para replicar las condiciones del experimento de Sección 6.2.1. En función de tratarse de exponentes grandes para este experimento se utilizó una estrategia de combinación con mecanismo de ventana deslizante equivalente a la propuesta GWOc_SWM, llamada LGWOc_SWM, obteniendo ésta última mejores resultados que su antecesora.

Finalmente en los experimentos de las Secciones 7.2.3 y 7.2.4 se compara LGWOc respecto de la misma propuesta híbrida del experimento de Sección 5.3.2 (GA Annealing [27]) y con el propio GWOc, y por otro lado con las propuestas del estado del arte: AIS, PSO, EP y también el propio GWOc.
Respecto del experimento de Sección 7.2.3, se obtuvieron los mismos resultados, e incluso mejores, para el caso de uno de los exponentes evaluados y por otro lado en experimento de Sección 7.2.4 contra AIS, PSO, EP y GWOc para un conjunto de exponentes diversos para los cuales se conoce su longitud óptima de cadena, se observa que también ha alcanzado resultados competitivos, puesto que iguala los de otras metaheurísticas del estado del arte y para el caso de GWOc logra superarla en el exponente 1087 donde dicha propuesta no logró alcanzar longitud óptima de cadena. Cabe destacar que aquí no se puede esperar que LGWOc supere a los otros enfoques puesto que las longitudes alcanzadas corresponden a las mínimas conocidas para estos casos de estudio (algunos de estos casos se pueden encontrar en la Tabla 5.6 con sus longitudes mínimas asociadas).

En todos los casos se emplearon lotes de pruebas de 30 ejecuciones independentes, tomando los resultados de la mejor ejecución.
Vale destacar que si bien la estrategia LGWOc utiliza la misma configuración de parámetros que GWOc, en términos de agentes de búsqueda e iteraciones, ha demostrado mejoras en algunos casos respecto de dicho enfoque y por otro lado observando las gráficas de convergencia de los experimentos de las Secciones 7.2.3

y 7.2.4, que corresponden a la mejor ejeucución de cada caso, revelan que dichos resultados fueron obtenidos en menor cantidad de iteraciones en LGWOc respecto de GWOc, puesto que la curvas convergen a la longitud mínima de cadena asociada al exponente en menos iteraciones (en algunos casos en las primeras).

En relación a la cantidad de evaluaciones por ejecución, como se mencionó anteriormente AIS no reporta, PSO utilizó 300000 evaluaciones, EP 92000 y la propuesta GA [26] utilizó una configuración de 50 ejecuciones independientes por cada experimento y se utilizó el criterio de 100 ejecuciones sin mejora de los resultados, como criterio de finalización. Por otro lado se empleó un número total de las generaciones a 1500 y tamaño de población de 300 individuos, en todos los experimentos con GA [26]. GWOc y GWOc_SWM insumieron 300 evaluaciones por ejecución y las propuestas LGWOc y LGWOc_SWM un total de 3000 evaluaciones, producto de 30 iteraciones y 10 agentes de búsqueda con 10 ejecuciones del operador de búsqueda local para cada iteración, para la optimización de los exponentes empleados en los experimentos descriptos en Secciones 7.2.1, 7.2.2, 7.2.3 y 7.2.4. La Tabla 7.6 refleja dicha situación.

Tabla 7.6: Desempeño de cada enfoque comparado en términos de cantidad de evaluaciones realizadas.

Propuesta	# Evaluaciones
GWOc	300
GWOc_SWM	300
LGWOc	3000
LGWOc_SWM	3000
PSO [19]	300000
AIS [6]	-
EP [8]	92000
GA [26]	450000

Como se observa en la Tabla 7.6, se evidencia una importante ventaja de las propuestas GWOc, GWOc_SWM, LGWOc y LGWO_SWM respecto de los otros enfoques, en relación a la cantidad de evaluaciones, mucho menos que las otras propuestas.

Capítulo 8

Conclusiones y trabajo futuro

8.1 Discusión general de las propuestas

En ésta tesis se presentan diferentes algoritmos para abordar un problema de actualidad que es la generación óptima de cadenas de adición. Cada propuesta posee un propósito específico. El algoritmo propuesto en Capítulo 5 (GWOc) con la finalidad de cubrir el objetivo principal de la tesis (adaptar la metaheurística GWO al problema objeto de estudio) y para el cual se plantearon 3 experimentos. Producto de los resultados obtenidos en los experimentos con GWOc y otras pruebas empíricas realizadas, se evidencia que el desempeño del algoritmo se ve afectado en la medida que los exponentes a optimizar se incrementan. En función de lo anterior, se propone una estrategia de abordaje para exponentes grandes para alcanzar el objetivo de tratar exponentes de 128-bits de longitud o cercanos, el resultado es GWOc_SWM (Capítulo 6). Se comparó GWOc_SWM con otras propuestas similares para conjunto de exponentes aleatorios del rango 128-bits que no logró superar pero dónde mostró resultados competitivos y en otro experimento con exponentes de menor tamaño contra el propio GWOc y otras metaheurísticas donde sí se evidenciaron mejoras para el conjunto de exponentes tratados. Por último se plantean LGWOc y LGWOc_SWM, como una mejora de los dos primeros algoritmos (Capítulo 7), ambos revelaron resultados equivalentes y superaron en algún caso a otros del estado del arte con menor número de evaluaciones, evidenciado en los gráficos de convergencia que representan el progreso del método durante la evolución de las generaciones para cada caso. En cuanto a los tiempos, las propuestas reportaron tiempos para todos los casos por debajo del minuto (escasos segundos o incluso milisegundos) de acuerdo con el tamaño del exponente a optimizar. Sin embargo, no es posible establecer una comparación en base a este criterio pues la mayoría de los trabajos de la literatura del tema con los cuales se ha comparado las propuesta de la tesis no reportan los tiempos de ejecución, salvo la propuesta AIS [6], por tanto los criterios de comparación resultaron ser: alcanzar la longitud mínima de cadena para cada caso planteado y el menor número de evaluaciones por ejecución posible, para cada propuesta. Respecto

de la cantidad de ejecuciones se ha utilizado la misma configuración en todas las propuestas (30 ejecuciones) para que resulte válida la comparación pero se infiere que con la mejora de hibridación de LGWOc y LGWOc_SWM, se podría mejorar, observando las curvas de convergencia, puesto que en muchos casos el valor mínimo de longitud de cadena es alcanzado rápidamente en las primeras evaluaciones.

8.2 Conclusiones finales

La presente tesis plantea el uso de la metaheurística GWO[24] para abordar un problema de optimización ampliamente estudiado en el ámbito de la Criptografía, obtener cadenas de adición de longitud mínima y los resultados del desempeño de dicha propuesta en comparación con abordajes similares de la literatura especializada del tema. La opción de esta elección radica en que no se poseen antecedentes del uso de dicha metaheurística para tratar el problema de referencia, la facilidad de implementación del algoritmo GWO frente a otros y el hecho de que no trabaja con operadores específicos de cruce, mutación, elitismo, etc. sino que el proceso de búsqueda es conducido por los tres mejores agentes de búsqueda de la población (α, β y δ) y por los parámetros A y C.
Esta adaptación del algoritmo GWO está diseñada para tratar sólo con soluciones factibles a partir de una población inicial de agentes de búsqueda. En general, se ha podido observar en los resultados de todos los experimentos diseñados para probar el desempeño de la propuesta que GWO requirió menor cantidad evaluaciones con respecto a otras metaheurísticas del estado del arte, para alcanzar los mismos resultados.

Para estudiar el desempeño de GWO se diseñaron cuatro algoritmos: GWOc, GWOc_SWM, LGWOc y LGWOc_SWM, con sus experimentos asociados, cada uno de ellos adaptado a un propósito en particular alineado con los objetivos de la tesis.
Los experimentos para GWOc se centraron en longitudes de cadenas acumuladas y comparación con las metaheurísticas GA [7, 26], GA Annealing [27], PSO [19], AIS [6] y EP [8]. Se logró igualar los resultados de las propuestas anteriores con el uso de GWOc salvo algún caso puntual detallado en los experimentos y en el caso de la propuesta de hibridación GA Annealing se evidenció superioridad sobre la misma, también se evidenció que GWOc requirió menor cantidad de evaluaciones que sus antecedentes.
Además, se formuló una adaptación de GWOc para obtener cadenas acumuladas de longitud mínima para exponentes del rango [1,128-bits] (GWOc_SWM) que de acuerdo con los resultados, demostró ser competitiva respecto de otros abordajes similares y en los cuales se empleó la misma estrategia de combinar la metaheurística con método de ventana deslizante y representación binaria de los exponentes. Para este experimento no se logró mejorar los resultados de las propuestas antecesoras pero se logró equiparar los mismos con menor cantidad de evaluaciones por ejecución.
Por último, se planteó una propuesta híbrida de GWOc con un operador de búsqueda

local (LGWOc) a fin de validar si aportaba mejoras y ayudaba a la metaheurística a resolver un conocido problema de estancamiento en soluciones óptimas locales y su equivalente para manejo de exponentes grandes (LGWO_SWM). Ambas propuestas con la misma configuración de parámetros de GWOc y GWOc_SWM lograron igualar e incluso mejorar los resultados en menor cantidad de ejecuciones según los gráficos de convergencia de las pruebas experimentales.
Las propuestas se consideran muy promisorias respecto de los tiempos de ejecución, aunque no es posible establecer comparaciones mediante este criterio con las propuestas del estado del arte. Sólo AIS [6] reportó un tiempo máximo de 170 milisegundos para exponentes de hasta 20-bits y tiempos menores para exponentes de menor tamaño, pero no informa tiempos para rangos superiores. Los experimentos planteados en este trabajo con exponentes menores o iguales a los 32-bits insumieron máximo 180 milisegundos y para los casos de exponentes dentro del rango 128-bits (experimentos de Secciones 6.2.1 y 7.2.2.) insumieron tiempos por debajo del minuto.

8.3 Trabajos futuros

Como trabajo futuro se plantea la posibilidad de extender la propuesta GWOc, considerando posibles mejoras, hibridaciones u otras opciones, para la obtención de cadenas de adición de longitud óptima para exponentes de rangos superiores (1024-bits o mayores), a fin de contribuir con un aporte novedoso a la Seguridad Informática y teniendo en cuenta que actualmente en los algoritmos de cifrado asimétricos (tales cómo RSA) se recomiendan claves de al menos 1024-bits de longitud. Asimismo, considerando que en los experimentos de los algoritmos propuestos se obtuvo resultados competitivos con mucho menor cantidad de evaluaciones que otras propuestas motiva a futuro investigar posibles mejoras sobre GWOc para exponentes de mayor tamaño bajo una configuración de cantidad de evaluaciones igual a la empleada en esta tesis o incluso menores con el sustento de que el algoritmo LGWOc alcanzó valores mínimos de cadenas de adición en las primeras iteraciones de la mejor ejecución.

Bibliografía

[1] Baro, M. (2013). Swarming: La comunicación en múltiples direcciones y múltiples etapas. razón y palabra. *Primera Revista Electrónica en Iberoamérica Especializada en Comunicación. Centro Avanzado de Comunicación - 25° Aniversario Eulalio Ferrer. Número 83, Junio – Agosto.*

[2] Bertolín, J. A. (2000). Criptoanálisis: factor clave para la ingeniería de seguridad de redes de ordenadores. *Revista Española de Informática*, 544:85–92.

[3] Binitha, S. y Sathya, S. S. (2012). A survey of bio inspired optimization algorithms. *International Journal of Soft Computing and Engineering (IJSCE) ISSN: 2231-2307, Volume-2, Issue-2. India.*

[4] Bos, J. y Coster, M. (1990). Addition chain heuristics. *Centrum voor Wiskunde en Infomatica. Kruislaan 413. 1098 SJ Amsterdam. The Netherlands.*

[5] Bremermanns, H. (1974). Chemotaxis and optimization. *Journal of the Franklin Institute.*, 297:397–404.

[6] Cruz-Cortés, N., Rodríguez-Henríquez, F., y Coello Coello, C. (2008). An artificial immune system heuristic for generating short addition chains. *México.*

[7] Cruz-Cortés, N., Rodríguez-Henríquez, F., Juárez Morales, R., y Coello Coello, C. (2005). Finding optimal addition chains using a genetic algorithm approach. *México.*

[8] Domínguez, I. (2011). Optimización de cadenas de adición en criptografía utilizando programación evolutiva. *Tesis de Maestría. Laboratorio Nacional de Informática Avanzada Centro de Enseñanza LANIA, Xalapa, Veracruz, México.*

[9] Fernández, S. (2004). La criptografía clásica. *Sigma: Revista de Matemáticas*, pages 119–142.

[10] Gonzalez, T. F. (2007). Handbook of approximation algorithms and metaheuristics. *2nd ed., Chapman and Hall/CRC - Taylor and Francis Group. 6000 Broken Sound Parkway NW, Suite 300 Boca Raton, FL 33487-2742.*

[11] Goundar, R., Shiota, K., y Toyonaga, M. (2008). New strategy for doubling-free short addition-subtraction chain. *International Journal of Applied Mathematics 2.3. U.S.A.*, pages 438–445.

[12] Gómez Bello, M. (2011). La aritmética modular y algunas de sus aplicaciones. *Universidad Nacional de Colombia, Facultad de Ciencias - Maestría en Enseñanza de las Ciencias Exactas y Naturales. Bogotá.*

[13] Hopcroft, J., Rajeev, M., y Ullman, J. (2008). Introducción a la teoría de autómatas lenguajes y computación. *Tercera edición. Pearson Educación S.A., Madrid, España.*

[14] Karaboga, D. (2005). An idea based on honey bee swarm for numerical optimization. *Erciyes University, Engineering Faculty, Computer Engineering Department. Kayseri, Türkiye.*

[15] Karaboga, D. y Basturk, B. (2007). A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. *Department of Computer Engineering, Erciyes University. Kayseri, Turkey.*

[16] Kaya Koc, C. (1994). High-speed rsa implementation. *Revista Española de Informática. Technical report, RSA. Laboratories, Redwood City, CA.*

[17] Kaya Koc, C. (1995). Analysis of sliding window techniques for exponentiation. *Computers Math. Applic.*, 30, N10:17–24.

[18] Kirkpatrick, S., Gelatt, C. D., y Vecchi, M. P. (1983). Optimization by simulated annealing. *Science, New Series*, 220:671–680.

[19] León, A., Cruz-Cortés, N., Moreno-Armendáriz, M., y Orantes-Jiménez, S. (2009). Finding minimal addition chains with a particle swarm optimization algorithm. *Center for Computing Research, National Polytechnic Institute. México.*

[20] Liu, Y. y Passino, K. (2002). Biomimicry of social foraging bacteria for distributed optimization: Models, principles, and emergent behaviors. *Journal of Optimization Theory and Applications. Ohio State University. Columbus, Ohio.*, 115:603–628.

[21] Luke, S. (2011). Essentials of metaheuristics. *First Edition*, pages 15–20.

[22] Mech., D. (1999). Alpha status, dominance, and division of labor in wolf packs. *Canadian Journal of Zoology. Jamestown, Dakota del Norte.*

[23] Mezura-Montes, E. y Hernandez-Ocaña, B. (2008). Bacterial foraging for engineering design problems: Preliminary results. *Laboratorio Nacional de Informática Avanzada (LANIA A.C.) - Universidad Juárez Autónoma de Tabasco. México.*

[24] Mirjalili, S., Mirjalili, S., y Lewis, A. (2014). Grey wolf optimizer. School of Information and Communication Technology, Griffith University, Nathan Campus, Brisbane QLD 4111, Australia. Department of Electrical Engineering, Faculty of Electrical and Computer Engineering, ShahidBeheshti University, G.C. 1983963113. Tehran, Iran.

[25] Nedjah, N. y De Macedo-Mourelle, L. (2004). Finding minimal addition chains using ant colony. *Department of Systems Engineering and Computation Faculty of Engineering, State University of Rio de Janeiro. Rio de Janeiro, Brasil.*

[26] Picek, S., Coello Coello, A., Jakobovic, D., y Metens, N. (2016). Evolutionary algorithms for finding short addition chains: Going the distance. *Volume 9595 of the series Lecture Notes in Computer Science*, pages 121–137.

[27] Pogančić, M. (2014). Evolving minimal addition chain exponentiation. *University of Zagreb - Faculty of Electrical Engineering and Computing. Zagreb, Croacia.*

[28] Rivest, R., Shamir, A., y Adlman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Cambridge.*

[29] Rotger, L., Coma, J., y Tena-Ayuso, J. (2012). Criptografía con curvas elípticas. *Universitat Oberta de Catalunya. España.*

[30] Stallings, W. (2004). Cryptography and network security: principles and practice. *2nd ed. Editorial Prentice Hall.*

[31] Tall, A. y Sanghare, A. (2013). Efficient computation of addition-subtraction chains using generalized continued fractions. *African Institute for Mathematical Sciences. Senegal.*, 2. N1:76–83.

[32] Wong, K. C. (2015). Evolutionary algorithms: Concepts, designs, and applications in bioinformatics. Evolutionary algorithms for bioinformatics. *Department of Computer Science, University of Toronto. Ontario, Canada.*

[33] Yang, X. (2010). Nature – inspired metaheuristic algorithms, second edition. *University of Cambridge. Luniver Press, United Kingdom.*