TESIS DEFENDIDA POR

Giovanna Martínez Arellano

Y aprobada por el siguiente comité:

Dr. Carlos Alberto Brizuela Rodríguez

Director del Comité

Dr. Hugo Homero Hidalgo Silva

Miembro del Comité

Dr. Andrei Tchernykh

Miembro del Comité

Dr. Axayácatl Rocha Olivares

Miembro del Comité

Dr. Pedro Gilberto López Mariscal

Coordinador del programa en Ciencias de la Computación

Dr. Edgar Gerardo Pavía López

Director de Estudios de Posgrado

5 de Octubre del 2007.

CENTRO DE INVESTIGACIÓN CIENTÍFICA Y DE EDUCACIÓN
SUPERIOR DE ENSENADA

PROGRAMA DE POSGRADO EN CIENCIAS
EN CIENCIAS DE LA COMPUTACIÓN

Heurísticas para el problema de Búsqueda de Motivos en secuencias de ADN

TESIS

que para cubrir parcialmente los requisitos necesarios para obtener el grado de

MAESTRO EN CIENCIAS

Presenta:

Giovanna Martínez Arellano

Ensenada, Baja California, México. Octubre del 2007.

RESUMEN de la tesis que presenta **Giovanna Martínez Arellano**, como requisito parcial para obtener el grado de MAESTRO EN CIENCIAS en CIENCIAS DE LA COMPUTACIÓN. Ensenada, B. C. Octubre del 2007.

Heurísticas para el problema de Búsqueda de Motivos en secuencias de ADN

Resumen aprobado por:

Dr. Carlos Alberto Brizuela Rodríguez

Director de Tesis

En esta tesis se describe una propuesta de solución para el problema de la búsqueda de motivos en secuencias de ADN. Los motivos son pequeños fragmentos de ADN los cuales están ubicados en la región promotora del gen. Estos fragmentos son sitios de acoplamiento de proteínas llamadas factores de transcripción las cuales permiten que se comience o se inhiba la transcripción de un gen. Debido a la complejidad de este problema, se han propuesto diferentes modelos para poder abordarlo. Uno de los modelos más estudiados es el modelo FM (Fixed number of Mutations) propuesto por Pevzner y Sze (2000). Este modelo consiste en buscar un motivo de longitud l con exactamente d mutaciones el cual se encuentra implantado en T secuencias de ADN cada una con una longitud de N nucleótidos. Junto con este modelo se propone el modelo VM (Variable number of Mutations), en el cual, la única diferencia con respecto a FM es que las d bases son mutadas con cierta probabilidad, teniendo cada ocurrencia del motivo a lo más d mutaciones.

En este trabajo se proponen varios algoritmos genéticos para atacar este problema. La motivación de utilizar algoritmos genéticos la da el éxito que tienen éstos en problemas de optimización combinatoria mono- y multi-objetivo. A pesar de que ya hay trabajo en esta área para el problema de búsqueda de motivos, hace falta aún una investigación más profunda en cuanto a la comparación de distintos esquemas de representación de la solución en distintos casos del problema, y en cuanto a cómo mejorarlos.

Los algoritmos genéticos propuestos utilizan dos codificaciones: una basada en las posiciones de inicio y otra basada en un motivo consenso. Con el objetivo de mejorar el desempeño de los algoritmos, se propone incorporar en ellos estrategias de búsqueda local.

En los experimentos elaborados, se utilizaron casos artificiales del modelo VM (Pevzner y Sze, 2000) y un caso real del sitio de acoplamiento CRP (cyclic-AMP Receptor Protein)(Stormo y Hartzell III, 1989). Resultados experimentales muestran que el algoritmo genético que utiliza una codificación basada en motivo es mejor que el algoritmo basado en posiciones de inicio, además hay una mejora importante en cuanto a calidad de solución y tiempos de ejecución del algoritmo genético estándar utilizando las estrategias de búsqueda local. Por otro lado, los resultados también muestran que la combinación de estrategias de búsqueda local fuera del genético pueden encontrar la solución al problema en un menor

tiempo, sin embargo, los promedios de calidad de la solución nos muestran que el algoritmo genético que implementa las dos búsquedas combinadas obtiene más soluciones cercanas al óptimo que la búsqueda local por sí sola.

Palabras clave: Bioinformática, Algoritmos Genéticos, Búsqueda Local, Motivo.

ABSTRACT of the thesis presented by **Giovanna Martínez Arellano**, as a partial requirement to obtain the MASTER SCIENCE degree in COMPUTER SCIENCES. Ensenada, B. C. October 2007.

Heuristics for the Motif Finding Problem

Abstract approved by:

Dr. Carlos Alberto Brizuela Rodríguez

Thesis director

In this work we present several enhanced heuristics for the motif finding problem. Motifs are small fragments of DNA located in the gene promoter region. These fragments are binding sites for proteins known as transcription factors which are involved in the gene regulation process. Due to the complexity of this problem, many models have been proposed. Pevzner and Sze (2000) studied a precise combinatorial formulation of this problem, called the planted motif problem (FM Model), which is of particular interest because it is a challenging model for commonly used motif-finding algorithms. The FM model (Fixed number of Mutations) consists on finding a motif of length l with exactly d mutations which is implanted in T DNA sequences, each of length N. In addition to this model, Pevzner and Sze proposed the VM model (Variable number of Mutations) where the occurrences of the motif are mutated with certain probability.

In this work, several genetic algorithms are proposed to solve this problem. The motivation of using genetic algorithms is given by the success of this strategy with mono- and multi-objective combinatorial optimization problems. Although there are previous works done in this area, studies comparing common encoding schemes and how to improve them are scarce.

The genetic algorithms proposed in this work use two encoding schemes: one based on the initial positions of the occurrences in the input sequences and the other based on a consensus motif. In order to improve the average score and computation time of these algorithms, specific local search strategies are proposed.

For the experiments, artificial instances were used, based on the VM model (Pevzner y Sze, 2000), as well as a real instance, the CRP binding site (cyclic-AMP Receptor Protein) (Stormo y Hartzell III, 1989). Experimental results show that an encoding scheme based on a consensus motif is better than the one based on initial positions. Experimental results also show that the memetic algorithm (genetic algorithm with local search strategies) provides better results in terms of average score and computation time compared to the standard genetic algorithm. Combined local search strategies also gave good results comparing to the standard genetic algorithm, nevertheless, the average score shows that the memetic algorithm gives more solutions that are close to the optimal than those obtained from the combined local search strategies.

Keywords: Bioinformatics, Genetic Algorithms, Local Search, Motif.

A mis padres

Agradecimientos

A mis padres Alejandro y Blanca, porque sin ellos no estaría aquí y porque a ellos les debo todos mis éxitos.

A mi abuelita Juanita que siempre me ha apoyado en mis decisiones.

Al Dr. Carlos A. Brizuela Rodríguez por su gran apoyo y por haberme guiado a cada paso de este trabajo.

Y a la persona más especial que me ha ayudado incondicionalmente hasta el final y a la que agradezco infinitamente por estar a mi lado, Abraham.

Ensenada, México
5 de Octubre del 2007.

Giovanna Martínez Arellano

Tabla de Contenido

Capítulo Página

Tabla de Contenido (Continuación)

Lista de Figuras

Lista de Tablas

Lista de Tablas (Continuación)

Lista de Tablas (Continuación)

Capítulo I

Introducción

I.1 Antecedentes

El problema de búsqueda de motivos es de gran importancia para la biología molecular. Los motivos juegan un papel importante en todos los procesos biológicos ya que controlan la producción de proteínas prendiendo y apagando los genes que tienen la información necesaria para producirlas. Un paso hacia la comprensión del funcionamiento de los mecanismos de regulación de la expresión de los genes es la habilidad de identificar los elementos regulatorios, es decir los sitios de unión de los factores de transcripción. Los factores de transcripción son proteínas que se unen al ADN normalmente cerca del sitio donde se comienza la transcripción del gen, por lo que los sitios de unión o motivos se pueden encontrar en cualquier parte de la región promotora del gen. Estos motivos consisten de pequeñas cadenas de ADN de las cuales se desconoce su longitud, aunque se estima que es de alrededor de 10 pares de bases (Tompa *et al.*, 2005).

Ya que el descubrimiento de motivos es un problema complejo, se han hecho distintos planteamientos con el objetivo de obtener versiones más simplificadas del mismo y así resolverlo. Pevzner y Sze (2000) estudiaron una formulación combinatoria precisa de este problema, llamada *problema del motivo implantado*, la cual es de interés particular para este trabajo, ya que es un modelo comunmente utilizado por algoritmos de búsqueda de motivos (Price *et al.*, 2003).

Diversos algoritmos han sido desarrollados para la búsqueda de motivos utilizando distintas estrategias como son enumeración (Blanchette *et al.*, 2002), (Brazma *et al.*, 1998), (Sinha y Tompa, 2000), búsqueda local (Hertz y Stormo, 1999), (Lawrence *et al.*, 1993),

(Bailey y Elkan, 1995), árboles de sufijos (Sagot, 1998), proyecciones aleatorias (Buhler y Tompa, 2002), entre otras. Algunos de ellos, construidos para abordar el problema bajo el modelo del motivo implantado, han obtenido un alto desempeño en casos artificiales como occure con el algoritmo Pattern Branching (Price *et al.*, 2003), sin embargo, esto no asegura un alto desempeño en los casos reales. Por otra parte, algunos otros algoritmos que no están basados en este modelo, como lo son Gibbs Sampler (Lawrence *et al.*, 1993) y MEME (Bailey y Elkan, 1995), son muy utilizados en la práctica aunque para los casos implantados tienen un bajo desempeño.

I.2 Definición del Problema

El problema de búsqueda de motivos consiste entonces en identificar pequeños sitios conservados en el ADN sin conocer, *a priori*, la longitud ni la composición química de éstos. Para secuencias de ADN, la búsqueda de motivos es comúnmente aplicada a conjuntos de secuencias (de un mismo genoma o de genomas de distintas especies) que han sido identificadas por poseer un motivo en común, lo cual convierte el problema biológico original, en uno combinatorio donde herramientas computacionales pueden ser utilizadas para resolverlo.

La dificultad de este problema recae en que los motivos presentan mutaciones, inserciones o ausencia de algunos nucleótidos y usualmente no ocurren exactamente igual, por lo que, mientras que ocurrencias aproximadas de un solo patrón pueden ser encontradas eficientemente, buscar todos los posibles 4^l patrones se vuelve más costoso conforme crece la longitud l del motivo.

El problema de búsqueda de motivos se puede definir de la siguiente manera:

Dado un conjunto de N secuencias cada una de longitud T y dada la longitud del motivo l y el número máximo de mutaciones permitidas d, encontrar todas las ocurrencias del motivo-(l, d) que se encuentran implantadas en las N secuencias.

I.3 Motivación

Muchas de las enfermedades, no nada mas del ser humano, sino también de otras especies, están relacionadas con la falta de producción de cierta proteína en el organismo. Muchas veces la falta de ésta se debe a que el motivo al que se unirá el factor de transcripción que se encarga de encender la transcripción del gen está mutado por lo que dicho factor no se une y no se realiza el proceso de transcripción. La importancia del descubrimiento de motivos reside en poder identificar aquellos motivos que se encuentran mutados y son responsables de la falta de producción de alguna proteína que consecuentemente pueda causar una enfermedad, ya sea experimentalmente o con herramientas computacionales, y así ayudar a encontrar una solución.

En este trabajo se pretende atacar el problema de búsqueda de motivos con un enfoque evolutivo. Los algoritmos genéticos son algoritmos de búsqueda estocástica que se inspiran en la evolución natural. Éstos han sido utilizados con mucho éxito en problemas difíciles (computacionalmente hablando), como lo son los problemas combinatorios. El uso de algoritmos genéticos para la búsqueda de motivos fué primero estudiado por Stravrovskaya y Mironov (2003), sin embargo, no se ha hecho un estudio exhaustivo del funcionamiento de éstos en las distintas variantes del problema. Este trabajo es un paso hacia este punto, ya que, además de proponer un algoritmo genético para el problema de búsqueda de motivos, se pretende estudiar su comportamiento con una serie de experimentos utilizando distintos casos del problema.

Para comenzar esta investigación, se hizo el planteamiento del objetivo general y de una serie de puntos específicos que serán alcanzados al finalizar este trabajo. Éstos son presentados a continuación.

I.4 Objetivos

I.4.1 Objetivo General

Diseñar un algoritmo evolutivo híbrido para el problema de búsqueda de motivos y determinar su desempeño relativo con respecto al enfoque evolutivo puro.

I.4.2 Objetivos Específicos

Para lograr el objetivo general, se definieron una serie de puntos que nos ayudaron a guiar el trabajo. Estos fueron planteados como preguntas, de tal manera que se pretende sean contestadas al finalizar el trabajo de investigación:

- ¿Cómo funcionan los métodos actuales para la búsqueda de motivos?
- ¿Cómo se mide y cuál es el desempeño de los métodos actuales?
- ¿Cuál es el desempeño teórico posible del mejor algoritmo hipotético para el problema?
- ¿Qué tan alejados están los métodos actuales de este desempeño límite?
- ¿Cuál es el desempeño del algoritmo evolutivo?
- ¿Cómo es su desempeño con respecto a los métodos actuales?
- ¿Cuál es el costo computacional del algoritmo evolutivo?
- ¿Cómo es el costo con respecto a los métodos actuales?

I.5 Metodología de Investigación

La metodología seguida para alcanzar los objetivos establecidos es como sigue:

1. Observación: revisión bibliográfica que permitió de una manera más concreta definir objetivos y limitantes.

2. Hipótesis: se hicieron una serie de suposiciones sobre las cuales se diseñaron los algoritmos basados en cómputo evolutivo utilizando estrategias de búsqueda local.

3. Experimentación: se construyeron casos de prueba artificiales para ejecutar el algoritmo así como también se experimentó con un caso real. Las variables a medir fueron la calidad de solución y el costo computacional del algoritmo.

4. Análisis de Resultados: una vez medidas las dos variables, se llevó a cabo un análisis estadístico que nos permitió determinar si había una diferencia significativa entre un algortimo y otro y se hizo una comparación con algunos de los métodos actuales para determinar si había una mejora en los resultados obtenidos por los otros métodos a un costo similar.

5. Se iteró sobre los puntos 2 a 4 hasta que se obtuvo el nivel de calidad deseado en el algoritmo genético.

I.6 Organización de la Tesis

Este trabajo consta de cinco capítulos y está organizado de la siguiente manera:

En el Capítulo II " Planteamiento del problema y Antecedentes" se describe el problema de búsqueda de motivos, su definición formal y el trabajo previo, que abarca desde el descubrimento del primer motivo, una descripción de distintos algoritmos que utilizan estrategias como enumeración, búsqueda local, proyecciones aleatorias y se finaliza con el trabajo realizado en el área de algoritmos evolutivos para el problema de búsqueda de motivos.

En el Capítulo III "Algoritmos Propuestos" se comienza por definir las representaciones de la solución así como los operadores genéticos que se utilizarán en el algoritmo genético

estándar desarrollado. Posteriormente se describen las estrategias de búsqueda a distancia y búsqueda lateral las cuales son implementadas dentro del algoritmo genético estándar (algoritmo memético).

En el Capítulo IV "Experimentos y Resultados" se describen los experimentos realizados así como los resultados correspondientes. Para esto, se expone cómo fueron generados los casos. Posteriormente, se presenta un análisis estadístico de los resultados obtenidos.

En el Capítulo V "Conclusiones y trabajo futuro" se discuten los resultados obtenidos y se comentan los problemas y propuestas a tratar como trabajo futuro.

Capítulo II

Planteamiento del problema

II.1 Definición del Problema

El mecanismo de transcripción de los genes está regulado, en parte, por proteínas que se acoplan al ADN llamadas factores de transcripción. Estas proteínas reconocen y se acoplan a secuencias específicas de ADN (sitios de acoplamiento o motivos) los cuales muestran, a menudo, una variación considerable tanto de gen a gen como de especie a especie. Entender el mecanismo de interacción y predecir los sitios de acoplamiento sin ningún tipo de información previa es un área de investigación que abarca una gran cantidad de problemas por resolver. Una vez que los sitios de acoplamiento de un cierto factor de transcripción son identificados, ya sea por predicción *de novo* o por experimentación, se puede obtener información acerca de la proteína y las secuencias con las que ésta interactúa (Carlson, 2004).

Al igual que los genes relacionados se derivan de ancestros en común, también las regiones reguladoras de éstos se derivan de un mismo ancestro, por lo que las instancias de un motivo pueden variar a causa de diferentes procesos biológicos como la duplicación y la translocación (Strachan y Andrew, 1999). Por esta razón, algunos genes pueden tener más de una copia de la región reguladora o no tener copia alguna. Aquellos genes con más de una copia tienen una respuesta más fuerte al factor de transcripción que aquellos que tienen sólo una o ninguna (Wu *et al.*, 2004).

Para secuencias de ADN, el descubrimiento de motivos es aplicado usualmente a conjuntos de secuencias (conjunto de genes) de un mismo genoma que han sido identificadas porque la transcripción de éstas es regulada por un mismo factor de transcripción. Sin embargo,

existe también el enfoque ortogonal el cual pretende identificar motivos dado un conjunto de genes ortogonales de distintos genomas en los cuales su distancia filogenética varía (Zaslavsky y Singh, 2006).

El problema de búsqueda de motivos se puede definir de manera general como sigue:

Dado un conjunto de secuencias desalineadas de ADN, estimar la longitud del motivo y localizar todas las ocurrencias (motivo con algunas mutaciones) que contienen las secuencias de entrada con la longitud estimada. No es necesario que todas las secuencias tengan ocurrencias del motivo y algunas de ellas podrán tener más de una ocurrencia.

Algo que está implícito dentro de la búsqueda de motivos es la manera en que se representará éste, es decir el modelo del motivo. El modelo busca representar los sitios de acoplamiento de tal manera que éstos se puedan clasificar de la manera más precisa posible. Cada modelo lleva de manera implícita algunas suposiciones sobre la naturaleza de los sitios de acoplamiento que pueden impactar el desempeño del algoritmo que utilice dicho modelo. En muchos casos un modelo más apropiado del motivo llevará a mejores resultados. Entre los modelos más usados en computación se encuentran la matriz de pesos de posiciones (PWM, por sus siglas en inglés) y consensus (Carlson, 2004). La matriz de pesos de posiciones modela la variación específica de cada posición incluyendo las preferencias cuantitativas entre bases en cada posición, mientras que consensus modela sólo la variación específica a cada posición. Más específicamente, el modelo PWM está representado por una matriz W de pesos de $4 \times n$, con los renglones indexados por $k \in \{A, C, G, T\}$ y columnas indexadas por $j \in \{1, 2, ..., n\}$, siendo n la longitud de las secuencias. Cada elemento $W(k, j)$ representa la probabilidad de que el j-ésimo caracter, en cierto sitio de entrenamiento $b \in B'$(B'es un subconjunto de motivos sobre los que es entrenada la matriz), sea la base k. Si se tiene alguna información sobre la afinidad del factor de transcripción F sobre el motivo b, entonces la contribución de b es medida de acuerdo a dicha afinidad. El modelo consensus está representado por una cadena $c = c_1c_2...c_l$ donde l es la longitud estimada del motivo buscado y c_i la base que más veces aparece en la posición i de todas

las ocurrencias identificadas del motivo.

Podemos identificar tres casos del problema de descubrimiento de motivos. En el primer caso, el motivo aparece en cada una de las secuencias de entrada con algunas mutaciones. En el segundo caso, el motivo aparece sólo en algunas secuencias presentando algunas mutaciones, y, en el tercer caso, más de una ocurrencia puede aparecer en una sola secuencia, presentando diferentes mutaciones.

Debido a la dificultad del problema, se han diseñado algunos modelos de éste que permiten reducir su dificultad y así poder resolverlo. Entre estos modelos se encuentran FM (Pevzner y Sze, 2000) (que cae en el primer caso del problema) donde cada una de las n secuencias generadas al azar contiene exactamente una ocurrencia de un motivo (l, d), es decir, un patrón de longitud l generado al azar que contiene exactamente d posiciones mutadas al azar. Otro modelo es el VM (Pevzner y Sze, 2000) donde de nuevo cada una de las n secuencias contiene exactamente una ocurrencia, solo que ahora cada posición de la ocurrencia es mutada, independientemente de las otras posiciones, con una probabilidad p. El modelo FM ya estaba implícitamente definido por Sagot (1998). A pesar de que el problema del motivo implantado no estaba explícitamente definido, el autor introdujo dos variantes del problema de búsqueda de motivos y una de ellas incluye al modelo introducido por Pevzner y Sze (2000). Esta variante introducida en Sagot (1998) es conocida como el problema del motivo común "The Common Motifs Problem" y consiste en encontrar todos los motivos m en N secuencias de entrada, dado $e > 0$ que indica el máximo número de mutaciones presentes y dado $2 \leq q \leq N$ que indica el número mínimo de secuencias donde se encuentra una ocurrencia del motivo.

Casi cualquier variante del problema de búsqueda de motivos pertenece a la clase NP-difícil (Zaslavsky y Singh, 2006). Más específicamente, el problema del motivo implantado pertenece a la clase NP-Completo (Davila *et al.*, 2006).

II.1.1 Definición Formal del Problema

El problema de búsqueda de motivos es planteado en este trabajo de la siguiente manera en base al modelo VM propuesto en Pevzner y Sze (2000):

Entrada: Un conjunto de n secuencias conteniendo cada una de ellas una ocurrencia de un motivo (l, d) donde l indica la longitud de éste y d el número máximo de mutaciones que puede tener cada ocurrencia.

Salida: Un conjunto S de posiciones de inicio $\{s_1, s_2, ..., s_n\}$ las cuales indican la primera base de cada una de las n ocurrencias encontradas que maximizan cierta función de evaluación (*score*).

El valor d nos podrá ayudar a determinar el máximo *score* que se puede lograr.

II.2 Trabajo Previo

La búsqueda de señales (descubrimiento de patrones en secuencias desalineadas de ADN) es un problema fundamental de la biología molecular cuyos modelos computacionales son también problemas fundamentales en ciencias de la computación.

La primera señal en ADN fué encontrada en 1970 por Hamilton Smith después del descubrimiento de la enzima de restricción conocida como Hind II. Encontrar el sitio palíndrome de esta señal no fué un problema sencillo en 1970 y casi cuarenta años después, a pesar de muchos estudios, el problema de búsqueda de señales está lejos de ser resuelto. La mayoría de las señales en el ADN (sitios promotores, sitios de empalme, etc) son tan complicadas que todavía no se tienen buenos modelos o algoritmos confiables para su reconocimiento.

II.2.1 Enfoques No Evolutivos

En Sagot (1998) se define implicitamente el problema del motivo implantado describiendo dos variantes del problema de búsqueda de motivos. Para atacar estas variantes, Sagot

utiliza árboles de sufijos construídos a partir de todos los sufijos que existen en las secuencias de entrada. Cada sufijo finaliza con una hoja que tiene un marcador especial de acuerdo a la secuencia a la que pertenece. Para buscar los patrones de longitud l con a lo más d mutaciones comienza con un patrón vacío a partir de la raíz del árbol y lo expande recursivamente. Esto es, que compara el primer símbolo de las aristas que salen de la raíz con el caracter A. Si A ocurre en al menos q secuencias (el nodo que lleva la arista A tiene hojas que pertenecen a q distintas secuencias), entonces expande el patrón a AA. Si AA es también válido, se mueve a AAA y así sucesivamente. Si no es válido, entonces se procede con C, buscando ocurrencias de AC. Sagot presenta un análisis de la complejidad de este algoritmo así como de otros algoritmos propuestos para distintos casos del problema. Sin embargo, no se presenta ningún experimento que nos demuestre el desempeño del algoritmo para distintas longitudes de motivos ni sus tiempos de ejecución.

Dado que los avances en el descubrimiento de motivos eran muy lentos, en Pevzner y Sze (2000) se plantea que no se ha identificado un algoritmo que sea capaz de resolver el siguiente problema "sencillo": Encontrar una señal en una muestra de secuencias de 600 nucleótidos cada una conteniendo una señal desconocida (patrón) de longitud 15 con 4 mutaciones.

Esta señal $(15, 4)$ es algo fuerte (es decir que es fácilmente identificable del resto de la secuencia dada su longitud y porcentaje conservado), más fuerte que la señal $(6, 0)$ del sitio de restricción Hind II. Sin embargo, algoritmos como CONSENSUS (Hertz y Stormo, 1999), Gibbs Sampler (Lawrence *et al.*, 1993), y MEME (Bailey y Elkan, 1995) fracasan en la búsqueda de la señal aún cuando las probabilidades de cada nucleótido en las secuencias de entrada son de $\frac{1}{4}$. Si un patrón de l letras aparece exactamente en cada secuencia, uno puede encontrar la señal haciendo una enumeración directa de todas las subcadenas de longitud l que aparecen en la muestra. Sin embargo, las señales biológicas están sujetas a mutaciones. Un modelo natural es permitir que el patrón desconocido aparezca con algún número de letras cambiadas, insertadas o borradas dentro de las secuencias de fondo (ADN).

Por qué entonces si $(15, 4)$ es una señal fuerte, es tan difícil identificarla del resto de la secuencia de ADN? El problema es que cualesquier dos instancias de la señal (l, d) mutada pueden diferir en incluso $2d$ posiciones, por lo que dos instancias en el problema planteado anteriormente pueden diferir en hasta ocho posiciones. Un gran número de similitudes falsas con ocho mutaciones en 15 posiciones hacen menos perceptible la señal real, por lo que la búsqueda se vuelve más complicada.

En Pevzner y Sze (2000) se propone el algoritmo Winnower para resolver el problema del motivo implantado. Éste construye un grafo a partir de una muestra S de secuencias y un parámetro l (longitud del motivo). Para cada posición j en la secuencia s_i construye un vértice que representa la subcadena s_{ij} de longitud l empezando en la posición j de s_i, donde $1 \leq j \leq T - l + 1$. Cada vértice s_{ij} se conecta con un vértice s_{pq} mediante una arista si i es diferente a p y la distancia entre s_{ij} y s_{pq} no es mayor a d.

Una vez construído el grafo multipartito, WINNOWER busca un clique expandible (aquél que cuenta con al menos un vecino u en cada parte del grafo, es decir nodos que corresponden a cada una de las secuencias de entrada, u es un vecino del clique $C = \{v_1, v_{2,}..., v_k\}$ si $C = \{v_1, ..., v_k, u\}$ es también un clique) utilizando algunas técnicas de podado. La estrategia más simple para WINNOWER es asegurar que en el grafo final, cada vértice tenga al menos un vecino en cada parte de G (es decir que cada nodo del grafo final que corresponde a la ocurrencia de una secuencia esté conectado con los nodos que corresponden a las ocurrencias que se encuentran presentes en el resto de las secuencias) y aquellos que no satisfacen esta condición sean borrados.

La desventaja de WINNOWER es que requiere de grandes recursos computacionales (tiempo y memoria) y se vuelve lento con señales sutiles que se encuentran en muestras muy grandes de ADN. Debido a esto, en Pevzner y Sze (2000) se propone otro algoritmo, SP-STAR. Éste primero encuentra un patrón W que corresponde a aquél que tiene la menor distancia $d(P, s_i)$, la distancia entre el patrón P y cualquiera de los $T-l+1$ posibles patrones de la secuencia s_i que sea menor, con todas las secuencias de entrada. Posteriormente se

emplea una estrategia de búsqueda local para mejorar la señal inicialmente encontrada, ya que esta pudiera ser una instancia de la señal y no la señal misma. Este algoritmo trabaja bajo la suposición de que se conoce la longitud del motivo buscado, por lo que, para los casos reales, se necesita correr el algoritmo considerando un rango de longitudes probables.

En Hertz y Stormo (1999) se propone el algoritmo CONSENSUS, el cual está basado en búsqueda local al igual que Gibbs (Lawrence *et al.*, 1993) y MEME (Bailey y Elkan, 1995). Hay algunos otros que trabajan con estrategias de enumeración como lo son el propuesto en (Blanchette *et al.*, 2002), (Brazma *et al.*, 1998), (Sinha y Tompa, 2000), entre otros.

A pesar de que los buscadores de motivos mencionados anteriormente han tenido mucho éxito en la práctica, en (Pevzner y Sze, 2000) se ha demostrado que CONSENSUS, Gibbs y MEME tienen un desempeño pobre en el problema del motivo $(15, 4)$. Los métodos basados en búsqueda local terminan usualmente en máximos locales, esto debido a que están fuertemente influenciados por la distribución precisa de las mutaciones del motivo. Típicamente los métodos basados en búsqueda local comienzan su búsqueda adivinando una ocurrencia inicial del motivo y posteriormente tratan de encontrar otras ocurrencias seleccionando aquellas que son similares a la inicial. Entre más distribuídas estén las mutaciones en el motivo, aumenta la probabilidad de que las ocurrencias que se encuentran similares a la inicial no sean las verdaderas ocurrencias, sino cadenas aleatorias de fondo.

En Buhler y Tompa (2002) se diseña el algoritmo PROJECTION basado en proyecciones aleatorias. La meta inicial es adivinar al menos s ocurrencias de un motivo implantado. Para este fin, el algoritmo lleva a cabo m intentos independientes, en cada uno de los cuales genera múltiples ocurrencias. En cada intento, elige una proyección aleatoria f y mapea cada posible ocurrencia x en una cubeta con etiqueta $f(x)$. Cualquier cubeta que reciba un número suficiente de entradas, será explorada como motivo potencial, utilizando un procedimiento de búsqueda local (maximización de la esperanza).

Una proyección f es construída eligiendo k posiciones al azar de manera uniforme del conjunto $\{1, ..., l\}$ para cierto valor de k. Si x es la cadena de longitud l, entonces $f(x)$ es

el resultado de concatenar las bases de las k posiciones de x. La intuición fundamental de las proyecciones aleatorias es que hay una gran posibilidad de que al menos s ocurrencias de cierto motivo sean mapeadas en la cubeta "implantada". El valor de k debe ser tal que impida que ocurrencias aleatorias de fondo se mapeen a la cubeta implantada. Suponiendo que requerimos a lo más E ocurrencias de fondo por cubeta en promedio, PROJECTION mapea $t(n-l+1)$ ocurrencias en 4^k cubetas, por lo que se fija

$$k \geq log_4(\frac{t(n-l+1)}{E}).$$

PROJECTION refina cada cubeta que sea lo suficientemente grande con la meta de recuperar el motivo implantado. Para refinar los motivos candidatos se utiliza el algoritmo conocido como maximización de la esperanza (EM). Esta formulación se deriva del siguiente modelo probabilístico simplificado. Una ocurrencia de un motivo aparece exactamente una vez en cada secuencia de entrada. Las ocurrencias del motivo son generadas de manera aleatoria de un modelo W de matriz de pesos de $4 \times l$ en la cual su entrada (i, j) da la probabilidad de que la base i aparezca en la posición j de una ocurrencia, independientemente de las otras posiciones. Las $n-l$ bases restantes de cada secuencia son elegidas de manera aleatoria e independiente de acuerdo a la distribución de fondo.

Sea T un conjunto de t secuencias de entrada y sea P la distribución de fondo. El refinamiento busca una matriz de pesos W^* que maximice la siguiente relación de verosimilitud:

$$LR(W^*) = \frac{Pr(T|W^*,P)}{Pr(T|P)},$$

esto es, un modelo de motivo que explique las secuencias de entrada de mejor manera que el modelo de fondo por sí solo.

PROJECTION lleva a cabo el refinamiento en cada cubeta que contiene al menos s ocurrencias. Se construye para cada una de las cubetas candidatas una matriz inicial W_b de la misma manera como se ha explicado anteriormente. Sea W_b^* el modelo resultante del refinamiento de W_b, se selecciona ahora una ocurrencia de cada secuencia de entrada con

base en éste, de tal manera que maximicen $Pr(x \mid W_b^*)/Pr(x \mid P)$. Este subconjunto S_b representa el motivo que es más consistente con el modelo W_b^*. Una vez que se tienen todos los subconjuntos que representan los posibles motivos, se selecciona aquel que maximice su puntaje sobre todas las cubetas y los m intentos.

Se construyeron varios experimentos, algunos con datos ficticios y otros con datos reales. Fué probado también con ADN ribosomal, el cual hace el problema más dificil, ya que no todas las secuencias de entrada contienen ocurrencias del motivo. Además de estos casos de prueba, se corrió el algoritmo haciendo cambios en la construcción de las secuencias de entrada para el caso ficticio. Se aumentó y disminuyó la frecuencia de bases G y C de las secuencias y se analizaron los resultados, esto con la intención de hacer las secuencias de entrada más parecidas a las reales, ya que se sabe de antemano que en las secuencias reales las bases no tienen la misma frecuencia de aparición; varían según el organismo. También se modificó la longitud de las secuencias, realizando experimentos con longitudes desde 600 hasta 2000 nucleótidos. En estos últimos experimentos, PROJECTION tuvo resultados con un desempeño por encima de los algoritmos basados en búsqueda local como Gibbs y MEME.

En (Zaslavsky y Singh, 2006) se propone un enfoque con herramientas de optimización combinatoria. En éste se define el problema de búsqueda de motivos como el de encontrar el alineamiento local de múltiples secuencias sin "gaps" utilizando el esquema de suma de pares y tomando en cuenta la distancia filogenética. El problema se modela con grafos y se plantea como uno de encontrar el clique de mayor costo dentro del grafo completo donde cada vértice representa una subsecuencia dentro de cada secuencia de los datos de entrada. Posteriormente se formula como un caso de un problema de programación lineal entera, el cual es NP-difícil, por lo que éste se relaja. Típicamente, los programas lineales son muy extensos ya que contienen millones de variables, por lo que son algo pesados para los resolvedores. Es por esto que para reducir la complejidad del programa lineal, se utilizan técnicas de podado donde posiciones de la secuencia que no sean compatibles con la solución

óptima serán descartadas. Con esta estrategia, se han llegado a resultados muy buenos, incluso para motivos muy degenerados, sin embargo, se continúa trabajando en el tema buscando mejoras para este enfoque.

En Price *et al.* (2003) se diseña el algoritmo Pattern Branching el cual parte de la idea de que si tenemos una ocurrencia de un motivo, es posible obtener el motivo del cual éste proviene, mediante una serie de iteraciones a las cuales se les llama ramificación ("branching"). Sea M un motivo y A_0 una ocurrencia de M con exactamente k mutaciones, ya que la distancia de Hamming $d(M, A_0) = k$, tenemos $M \in D_{=k}(A_0)$, definido como el conjunto de patrones de distancia k de A_0. Se tendrían que evaluar los $\binom{l}{k} \times 3^k$ patrones del conjunto anterior para encontrar el motivo. Ahora, como hay que aplicar este procedimiento para cada una de las cadenas A_0 de longitud l que se encuentran en la muestra, el algoritmo sería bastante lento. Es por esto que se propone que se construya un camino de patrones:

$$A_0 \rightarrow A_1 \rightarrow ... \rightarrow A_k,$$

aplicando iterativamente una función que mapea el patrón A a su mejor vecino en $D_{=1}(A)$, cambiando sólo un nucleótido de A. Esta caminata nos llevará, a un menor costo computacional, al óptimo global del conjunto de $\binom{l}{k} \times 3^k$ posibles patrones. Al final, el patrón A_k será considerado como un candidato a motivo. En otras palabras, si A es realmente una ocurrencia de M con a lo más k mutaciones, entonces A_j será calificada como un candidato a ser el motivo M en la iteración k. Este procedimiento (*branching*) permite reducir el número de vecinos a analizar.

Ahora, para calificar el patrón y elegir el mejor vecino, se utiliza su distancia total con respecto a la muestra. Es decir, que para cada secuencia S_i en la muestra $S = \{S_1, S_2, ..., S_n\}$, $d(A, S_i) = min\{d(A, P) \mid P \in S_i\}$, donde P denota una ocurrencia de longitud l. Entonces la distancia total de A con respecto a la muestra es $d(A, S) = \sum_{S_i \in S} d(A, S_i)$. El mejor vecino de A es el patrón $B \in D_{=1}(A)$ con la distancia total $d(B, S)$ más pequeña.

Pattern Branching es probado con casos de motivos implantados y obtiene rangos de éxito muy altos comparando con PROJECTION (Buhler y Tompa, 2002), MITRA (Eskin y Pevzner, 2002) y MULTIPROFILER (Keich y Pevzner, 2002). En cuanto a tiempos de ejecución, éstos están muy por debajo de aquellos obtenidos por los algoritmos anteriormente mencionados. En cuanto a las pruebas que se realizaron con muestras reales, el algoritmo logró identificar motivos que obtuvieron una distancia total mucho menor a aquellos previamente identificados por los métodos más comunes. Sin embargo, no se ha identificado con este algoritmo motivo real alguno que no haya sido antes identificado por métodos más comunes como MEME (Bailey y Elkan, 1995).

II.2.2 Enfoques Evolutivos

En Stavrovskaya y Mironov (2003) se introduce por primera vez la aplicación de los algoritmos genéticos al problema de búsqueda de motivos. Se desarrollaron dos algoritmos denominados GA1 y GA2. El primero está basado en la matriz de pesos de posiciones. Cada cromosoma representa una alineación de todas las secuencias y lo que se pretende es encontrar todas las alineaciones que maximicen la suma de las frecuencias máximas de los nucleótidos en cada posición de la alineación. En el segundo, el cromosoma representa un consenso candidato de longitud l y el algoritmo pretende maximizar la aptitud de la cadena consenso con respecto a todas las secuencias de entrada. Para un consenso de longitud l, el algoritmo tendrá que buscar el mejor de todos los 4^l posibles motivos.

En base a los experimentos realizados, se concluye que debido a que en el algoritmo GA1 el campo de búsqueda es más grande, se necesitan poblaciones más grandes y más iteraciones para obtener resultados razonables, mientras que en GA2, el campo de búsqueda es más pequeño y se reduce significativamente el número de iteraciones. Por último se concluye que GA2 es más aplicable en casos reales ya que las probabilidades de identificar motivos correctos son mayores en los casos en que no todas las secuencias de entrada cuentan con una ocurrencia de un motivo.

En (Che *et al.*, 2005) se propone un algoritmo genético, MDGA, el cual pretende predecir de manera eficiente los sitios de acoplamiento en genes homólogos. En MDGA, un individuo está conformado por un conjunto de posiciones de inicio de los sitios de acoplamiento en cada una de las secuencias de entrada. La aptitud del individuo es evaluada alineando todos los sitios de acoplamiento y observando la similitud entre ellos. Aquellos individuos que tengan baja similitud entre sus sitios de acoplamiento, serán penalizados, permitiendo así que sobrevivan aquellos con sitios mejor conservados.

Para probar el desempeño de MDGA, se usaron éste y otras dos herramientas computacionales, Gibbs Sampler y BioProspector, para predecir las posiciones de los sitios de acoplamiento en un conjunto de 18 secuencias de la proteína receptora CRP (Stormo y Hartzell III, 1989). Los resultados de MDGA son superiores a aquellos obtenidos por Gibbs Sampler y BioProspector. MDGA predice casi todos los sitios con un corrimiento de sólamente una base a la derecha, mientras que los otros métodos tienen un margen de error más grande.

En Karaoglu *et al.* (2006) se diseña el algoritmo GAMOT, el cual utiliza algoritmos genéticos de estado estacionario para resolver el problema propuesto en Pevzner y Sze (2000). En este algoritmo, cada cromosoma representa un motivo candidato de longitud l del cual se busca maximizar su aptitud utilizando algunos operadores genéticos.

GAMOT hace una búsqueda rápida de motivos para obtener una población inicial. Para hacer esta búsqueda, considera todas las posibles subcadenas de longitud l que se encuentran en las secuencias de entrada, en vez de buscar en todos los 4^l posibles motivos y selecciona aquellas que tienen una aptitud mayor a cierto umbral. El algoritmo básico recorre todas las secuencias de entrada, removiendo una a la vez del conjunto, toma x ocurrencias elegidas al azar de la secuencia removida, calcula sus aptitudes y toma aquellas que superan el umbral (x es parámetro de entrada del algoritmo, pudiendo tomar un valor máximo de $T - l + 1$). La aptitud de la ocurrencia elegida es calculada en base a las $N - 1$ secuencias de entrada restantes. Después de haber recorrido todas las secuencias de entrada,

aquellas ocurrencias que hayan superado el umbral serán tomadas como la población inicial del algoritmo genético. GAMOT sólo selecciona x motivos al azar del total de ocurrencias posibles para reducir el tiempo y complejidad de la búsqueda, sin embargo el valor de N puede ser igual a $(n - l)$, lo que equivale a si se tomaran todas las subcadenas posibles de longitud l de las secuencias de entrada. La función que calcula la aptitud se basa en la distancia total (*TotalDistance)* del motivo candidato con respecto a las secuencias restantes de entrada, la cual está definida como la distancia de Hamming más pequeña entre la cadena m y todas las posibles cadenas de las secuencias de ADN. En el mejor caso, una de las variantes del motivo real tendrá la distancia total más pequeña y será regresada por el algoritmo de búsqueda como motivo candidato. Algunos de estos motivos candidatos serán la población inicial del algoritmo genético de acuerdo a su aptitud y al parámetro de entrada del algoritmo que indica la cantidad de individuos que conformarán la población.

Una vez encontrada la población inicial, el algoritmo selecciona dos individuos de la población utilizando "linear ranking" (Eiben y Smith, 2003). Estos dos individuos se cruzan, utilizando cruzamiento de dos puntos, para crear un nuevo individuo el cual remplazará el peor de los dos anteriores. El algoritmo aplica dos operadores de exploración. El primero de ellos es la mutación, el cual es aplicado al azar, cambiando algunas bases elegidas también al azar. El segundo operador es el de recorrimiento, el cual recorre los nucleótidos a la izquierda o derecha una posición para lograr un mejor alineamiento. La posición que queda abierta por el recorrimiento, se llena con un nucleótido elegido al azar. La aplicación de estos operadores queda ambigua en el artículo (Karaoglu *et al.*, 2006), así como también el cálculo de la complejidad de la función de búsqueda rápida de motivos.

GAMOT lleva a cabo los pasos anteriormente explicados hasta que la aptitud del mejor individuo alcance cierto umbral o hasta que un cierto número de iteraciones hayan sido completadas. El motivo que es arrojado como salida del algoritmo, se utiliza para buscar las ocurrencias de éste en las secuencias de entrada. Esto se hace buscando en cada secuencia la subcadena que tiene la menor distancia de Hamming con respecto al motivo.

Para probar el desempeño de GAMOT, se probaron casos de motivos muy cortos, casos típicos como $(15, 4)$, y motivos muy grandes, comparando los resultados obtenidos con métodos de búsqueda exhaustiva, Random Projections, Algoritmos Genéticos Estándar, entre otros. GAMOT obtiene una calidad de solución que supera a todos los demás algoritmos y sus tiempos de ejecución son muy pequeños.

Capítulo III

Algoritmos propuestos

III.1 Algoritmo Genético

Para atacar el problema de búsqueda de motivos, se plantea en este trabajo utilizar un enfoque evolutivo, más específicamente, algoritmos genéticos. Para diseñar el algoritmo genético, se deben definir en primera instancia la manera en que se representará la solución (individuo) y los operadores genéticos (cruzamiento y mutación) a utilizar. Entre los esquemas de representación de los motivos en algoritmos genéticos se encuentra el basado en el consenso y el basado en posiciones de inicio (Stavrovskaya y Mironov, 2003). Dado que los dos esquemas son sencillos de implementar, en este trabajo se utilizarán ambos con el objetivo de determinar con cuál se obtienen mejores resultados. A continuación se describe cada una de estas codificaciones y los algoritmos estándar correspondientes a éstas.

III.2 Algoritmo Basado en Posiciones de Inicio (PbGA)

El esquema de codificación de la solución utilizado en este algoritmo es el siguiente:

$$I = p_1, p_2, ..., p_N,$$

donde p_i indica la posición de inicio de la ocurrencia encontrada en la secuencia i. La Figura 1 muestra un ejemplo de la codificación basada en posiciones de inicio.

La aptitud del individuo será calculada alineando las N ocurrencias a las que éste hace referencia y sumando por cada columna la letra que más se repite. En el ejemplo mostrado a continuación

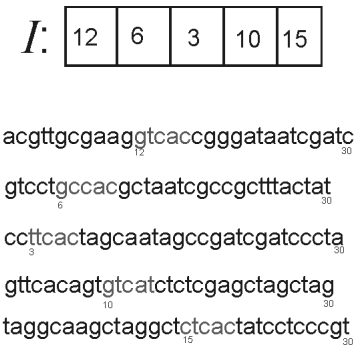

Figura 1. Ejemplo de una codificación basada en posiciones de inicio. La longitud del motivo es $l = 4$ y la longitud del individuo es cinco debido a que son cinco secuencias de entrada.

a	g	g	c	t
a	g	t	c	t
c	g	g	c	t
a	g	c	c	g
a	g	t	a	t
Apt: 4	5	2	4	4	= 19

se puede observar cómo se lleva a cabo el cálculo de la aptitud del individuo. La primera columna muestra que la letra que más se repite es la a; cuatro veces. En la segunda columna, la letra que más se repite es g; cinco veces. Así con cada columna, se determina cuál es la letra que más se repite y se va acumulando el número de repeticiones. En el ejemplo anterior, la suma de las repeticiones más frecuentes en todas las columnas fué igual a 19.

En este caso se busca maximizar la aptitud a través de las generaciones. No es difícil ver que el costo de calcular la aptitud es de $O(l * N)$, con l la longitud del motivo y N el número de secuencias.

Una vez definida la representación del individuo, definimos los operadores genéticos. Para el algoritmo basado en posiciones de inicio, el cruzamiento entre dos individuos es de la siguiente manera:

Dados

$$P_1 = \boxed{p_{11}}\boxed{p_{21}}\cdots\boxed{p_{N1}}$$

$$P_2 = \boxed{p_{12}}\boxed{p_{22}}\cdots\boxed{p_{N2}},$$

se elige al azar un punto de cruza, de tal manera que en el hijo toda la parte izquierda antes del punto de cruza será heredada del padre 1 (P_1) y toda la parte derecha a partir del punto de cruza será heredada del padre 2 (P_2), obteniendo asi un nuevo individuo de longitud l. En el ejemplo que se muestra a continuación

$$P_1 = \boxed{\mathbf{23}}\boxed{\mathbf{309}}\boxed{276}\boxed{12}\boxed{513}$$

$$P_2 = \boxed{506}\boxed{281}\boxed{\mathbf{105}}\boxed{\mathbf{33}}\boxed{\mathbf{447}}$$

$$I = \boxed{23}\boxed{309}\boxed{105}\boxed{33}\boxed{447}$$

el punto de cruza cae entre la segunda y tercera posición, por lo que los dos primeros elementos son copiados del padre 1 (P_1) y el resto del padre 2 (P_2).

Para mutar un individuo, simplemente se escoge una de las N posiciones de inicio al azar y se reemplaza con otra posición de inicio generada de manera aleatoria, pudiendo tomar cualquier valor entre 1 y $T - l + 1$.

A continuación se presenta el pseudocódigo del algoritmo.

III.2.0.1 Algoritmo 1. PBGA

Entrada: Número de cadenas N, longitud de las cadenas T, longitud del motivo l, número de bases mutadas d, probabilidad de mutación P_m, probabilidad de cruzamiento P_c, tamaño de la población $PopSize$, número de jugadores en el torneo J, máximo número de iteraciones sin cambio en la aptitud del mejor individuo $Max\_Iter\_No\_Change$.

Salida: Un conjunto de posiciones $\{p_1, p_2, ..., p_N\}$ y su calidad correspondiente.

1 Inicializar $PopSize$ individuos con posiciones de inicio aleatorias, cada posición de inicio se elige uniformemente de $\{1, ..., T - l + 1\}$.

2 Evaluar la aptitud de los $PopSize$ individuos y el mejor individuo es tomado como super individuo.

3 Mientras el número de generaciones sin cambio en la aptitud sea menor a $Max\_Iter\_No\_Change$, hacer:

4 De $i = 1$ hasta $PopSize$

5 Escoger J individuos para participar en un torneo y el ganador será P_1

6 Escoger J individuos para participar en un torneo y el ganador será P_2

7 Cruzar P_1 y P_2 con probabilidad P_c para obtener un nuevo individuo

8 Aplicar mutación a cada individuo con probabilidad P_m

9 Reemplazar la población actual con los nuevos $PopSize$ individuos

10 Evaluar la aptitud de la nueva población para escoger el mejor individuo y compararlo con el super individuo actual y guardar el mejor (nuevo super individuo)

11 Termina Mientras

El algoritmo inicia creando $PopSize$ soluciones de manera aleatoria (línea 1), es decir que cada solución contendrá N posiciones de inicio con valores aleatorios entre 1 y $T-l+1$. Una vez creadas, se evalúa la aptitud de cada una de éstas (línea 2), ya que es necesaria para la selección de individuos en el cruzamiento (líneas 5 y 6). El mejor de todos se convierte

en el super individuo. Debido a que en este algoritmo todos los hijos substituyen a todos los padres, se deben crear, a cada iteración del algoritmo, $PopSize$ individuos nuevos (línea 4). Para esto, se llevan a cabo dos torneos por cada individuo nuevo. En el primer torneo (línea 5) juegan J individuos elegidos al azar y el de mejor aptitud se convierte en P_1. En el segundo torneo (línea 6), de la misma manera se eligen J individuos al azar y el de mejor aptitud se convierte en P_2. El cruzamiento de P_1 y P_2 genera un nuevo individuo (línea 7). Una vez creados los $PopSize$ individuos, se les aplica a éstos mutación (línea 8) con cierta probabilidad P_m. Posteriormente, se hace el reemplazo generacional (línea 9) y el mejor individuo de la nueva generación se compara contra el super individuo. Si el mejor individuo de la nueva generación tiene mejor aptitud, entonces éste se convierte en el nuevo super individuo (línea 10). Todo el proceso, desde la creación de los nuevos individuos hasta el remplazo generacional se repite mientras que el número de generaciones sin cambio en la aptitud del mejor individuo sea menor a $Max\_Iter\_No\_Change$.

Analizando las líneas del pseudocódigo, podemos ver que los valores de N y l, que son los que varían según el caso del problema, afectan en los procedimientos de inicialización y cálculo de la aptitud de la población (líneas 1, 2 y 10). Para la línea 1, tenemos $PopSize*N$ asignaciones y en la línea 2 tenemos $PopSize * N * l$ operaciones donde $N * l$ es el costo de calcular la función objetivo. El costo de la línea 3 (ciclo mientras) lo expresaremos con la variable X, la cual puede tomar valores desde $M < ax\_Iter\_No\_Change$ hasta un número infinito, por lo que sólo la dejaremos expresada como X. Ya que el algoritmo genético itera dentro de este ciclo, podemos ver que las líneas interiores al ciclo estarán multiplicadas por este factor. De las líneas interiores (4 a la 10) tenemos un ciclo que implica $PopSize + 1$ comparaciones. Dentro de este ciclo se llevan a cabo 2 torneos, cada uno de los cuales implica $J - 1$ comparaciones y un cruzamiento que implica N asignaciones. Podemos ver entonces que en este ciclo tenemos $2(J-1)Popsize+NPopSize+PopSize+1$ operaciones que dependen del tamaño de la población inicial y del número de secuencias de entrada. La línea 8 implica una operación por cada individuo, es decir, $PopSize$ operaciones y la línea

9 implica N asignaciones por cada individuo, es decir, $NPopSize$. Como se puede ver, las líneas del pseudocódigo nos indican una alta dependencia del tiempo de ejecución con los valores de $PopSize$, N y l, todas ellas multiplicadas por X que nos indica el número de iteraciones del algoritmo. Después de determinar el costo de cada línea, podemos ver que el algoritmo tiene un costo de $O(NlPopSizeX)$.

III.3 Algoritmo Basado en Consenso (MbGA)

El esquema de codificación de la solución utilizado en este algoritmo se ilustra con el siguiente ejemplo:

$I =$ | a | a | g | t | t | c | a | c | c | g |,

donde el individuo I representa un motivo candidato. Para calcular la aptitud del individuo, se utiliza el concepto de *distancia total* [Price et al., 2003]. Para calcular esta distancia primero recordaremos algunos conceptos básicos. La distancia de Hamming $d(s_1, s_2)$ entre dos subcadenas s_1 y s_2 de longitud l es el número de caracteres en los cuales éstas difieren. Para cada cadena S_j, sea $d(m, S_j) = min\{d(m, p) \mid p \in S_j\}$, donde p denota una subcadena de longitud l y m el motivo candidato. Entonces la distancia total de m con respecto a las N cadenas de entrada está dada por $d(m, S) = \sum_{j=1}^{N} d(m, S_j)$. En este caso se busca minimizar esta distancia, de tal manera que al finalizar el algoritmo, sobrevivan aquellos individuos que representen un motivo candidato que sea el implantado. Otras propuestas de evaluación de la aptitud son presentadas por Stavrovskaya y Mironov (2003).

Una clara limitación de la representación basada en posiciones de inicio es que no es trivial lidiar con repeticiones de ocurrencias en una o más secuencias o con la ausencia de éstas. Para mejorar este esquema, se tomaron ideas del algoritmo Pattern Branching (PBA) (Price *et al.*, 2003). La idea principal de PBA es trabajar con el motivo mismo, no con las posiciones, y medir su aptitud calculando la distancia de Hamming entre el motivo

candidato y las secuencias de entrada. Si bien la motivación de este trabajo para usar MbGA fué el PBA (Price *et al.*, 2003), ya hay trabajos en GA que usan esta representación. Si se utiliza la representación basada en posiciones de inicio, el espacio de búsqueda es de tamaño $(T - l + 1)^N$. Éste crece exponencialmente con el número de secuencias N. Utilizando el consenso el espacio de búsqueda sigue siendo exponencial, 4^l, sin embargo, es independiente del número de secuencias de entrada y de la longitud de éstas.

Para generar cada individuo de la población inicial, ya que ahora se manejarán motivos, se eligen l bases al azar, cada una con la misma posibilidad de ser escogida.

Los operadores genéticos para MbGA son similares a los utilizados en PbGA. Para el operador de recombinación, se utiliza cruzamiento de un punto, el cual es elegido al azar. Para la mutación, se eligen uno de los l caracteres al azar y se remplaza con otro, también generado al azar del alfabeto $\{a, c, g, t\}$, donde cada caracter tiene la misma probabilidad de ser seleccionado.

El algoritmo MbGA resultante es similar a PbGA, cambiando sólamente la representación y la función objetivo, la cual se busca minimizar.

Ya que el algoritmo MbGA sólo varía de PbGA en la codificación del individuo y la manera de calcular la función objetivo (con costo de Nl), podemos ver que la complejidad de MbGA es la misma que la de PbGA.

III.4 Algoritmos de Búsqueda Local

Como un intento de mejorar el desempeño del algoritmo genético basado en el consenso (MbGA), se implementaron dos estrategias de búsqueda local. Lo que se pretende, es que estas estrategias ayuden al algoritmo genético a converger más rápidamente a soluciones de mejor calidad.

III.4.1 Búsqueda de d pasos bajo el vecindario $D_{=1}$

La idea principal de esta búsqueda es que si tenemos una ocurrencia m' de un motivo m, es posible obtener m en d pasos cambiando d bases en m' (Price *et al.*, 2003). Para esto, comenzamos en una solución inicial o motivo candidato (I_0), luego pasamos al vecino I_1, luego al I_2, así hasta llegar a un vecino ubicado a d pasos de I_0:

$$I_0 \rightarrow I_1 \rightarrow ... \rightarrow I_d.$$

Para encontrar el mejor vecino I_1 de I_0, se debe hacer una búsqueda en el vecindario $D_{=1}(I_0)$, es decir, todas las cadenas de longitud l que difieren de la solución actual I_0 en un y sólo un caracter, y tomar la cadena de mejor función objetivo (menor distancia de Hamming con respecto a las secuencias de entrada). Si ésta es al menos igual en función objetivo que I_0, entonces será considerada nuestra nueva solución I_1. Si tal vecino no existe, entonces I_0 será considerado óptimo local. Este proceso se repite d veces, de tal manera que al final, I_d deberá representar el motivo del cual la ocurrencia I_0 deriva (Price *et al.*, 2003). Esta búsqueda bajo el vecindario $D_{=1}$ se implementa en el procedimiento i' =BúsquedaADistancia(i) utilizado a continuación, donde i es la solución a la que se le aplica la búsqueda bajo el vecindario $D_{=1}$ e i' la solución resultante de la búsqueda aplicada a i.

Es importante notar que esta estrategia no es propiamente un algoritmo de búsqueda local ya que sólo lleva a cabo d pasos, sin importar si llega o no a un óptimo local.

Algoritmo de Búsqueda local bajo el vecindario $D_{=1}$

El algoritmo de búsqueda a distancia $D_{=1}$ consiste en aplicar dicha búsqueda a un conjunto de soluciones iniciales, sólo que no se restringirá el número de pasos a d, sino que ésta terminará una vez que haya convergido a un óptimo local.

Algoritmo 2. Búsqueda a Distancia $D_{=1}$

Entrada: Un conjunto de S soluciones iniciales de acuerdo al $caso(l, d)$ del motivo implantado, la longitud del motivo l y el número máximo d de mutaciones presentes en el motivo.

Salida: el mejor óptimo local.

1 Para cada solución i de las S soluciones iniciales

2 i'=BúsquedaADistancia(i)

3 Fin Para

4 Regresar la solución de mejor función objetivo (Menor distancia de Hamming con respecto a las secuencias de entrada).

Ya que buscar el mejor vecino de m en el vecindario $D_{=1}$ implica calcular las funciones objetivo de todas las cadenas que difieren en un caracter del motivo m ($3l$ vecinos), podemos ver que el costo de una búsqueda en el vecindario es de $Nl3l = 3Nl^2$. El algoritmo entonces tiene un costo de $O(Nl^2 x_i S)$, donde x_i indica el número de iteraciones de la búsqueda aplicada a la i-ésima solución la cual finaliza cuando la función objetivo ya no mejora y S es el número de soluciones iniciales.

III.4.2 Búsqueda Lateral

El objetivo de esta búsqueda es explorar el vecindario que existe recorriendo la solución actual un caracter a la izquierda o uno a la derecha. Si la solución se recorre un caracter a la derecha, un nuevo caracter será elegido al azar del alfabeto $\{a, c, g, t\}$, y será insertado al inicio de la cadena. Si la cadena se recorre un caracter a la izquierda, un nuevo caracter será insertado al final de la cadena. De los ocho vecinos que existen, el mejor de ellos será tomado y reemplazará a la cadena actual sólo si éste tiene una función objetivo al menos igual a la de la cadena actual. Suponiendo que para un conjunto dado de secuencias, se evaluan los siguientes motivos de la siguiente manera:

$$I_0 : \text{TGTCACGTAT} \rightarrow Score : 38$$

$$I_0^{'} : \text{GTCACGTATC} \rightarrow Score : 34$$

En este caso, la solución I_0 se recorrió a la derecha, ya que, de sus ocho vecinos, GTCACGTATC fué el vecino que tenía la mejor función objetivo y además mejoraba la solución actual.

Ya que la búsqueda local es aplicada sobre soluciones basadas en el esquema de consenso, la función objetivo de éstas se mide con la distancia de Hamming con respecto a las secuencias de entrada.

Esta búsqueda se implementó en el procedimiento i' =BúsquedaLateral(i), donde i es la solución a la que se le aplica la búsqueda lateral e i' la solución resultante después de aplicar la búsqueda a i.

Algoritmo de Búsqueda Lateral

Este algoritmo, al igual que la búsqueda local anteriormente expuesta, consiste en aplicar la búsqueda lateral a un conjunto de soluciones iniciales. Esta búsqueda se detiene cuando ya no se encuentra una solución vecina que mejore la actual. El algoritmo es similar al Algoritmo 2, con la diferencia que en la línea 2 se emplea la búsqueda lateral expuesta en la sección anterior.

Algoritmo 3. Búsqueda Lateral

Entrada: Un conjunto de S soluciones iniciales de acuerdo al $caso(l, d)$ del motivo implantado, la longitud del motivo l y el número máximo d de mutaciones presentes en el motivo.

Salida: el mejor óptimo local.

```
1  Para cada solución i de las S soluciones iniciales
2      i'=BúsquedaLateral(i)
```

3 Fin Para

4 Regresar la solución de mejor función objetivo

Ya que la búsqueda lateral debe calcular la función objetivo de los 8 posibles vecinos de m, el costo de aplicar una vez la búsqueda es de $8Nl$ por lo que el costo del algoritmo de Búsqueda Lateral es igual a $O(Nlx_iS)$ donde x_i es el número de veces que se aplica la búsqueda lateral a la i-ésima solución la cual finaliza cuando ya no hay una mejora en la función objetivo de m y S el número de soluciones iniciales.

III.5 Combinación de la búsqueda local con vecindario $D_{=1}$ y la búsqueda lateral

Una vez diseñados los algoritmos de búsqueda local, se decidió diseñar un algoritmo que combinara las dos estrategias de búsqueda de la siguiente manera: dado un conjunto inicial de soluciones, se aplica de manera iterativa a cada solución la búsqueda local con vecindario $D_{=1}$ y posteriormente la búsqueda lateral, mientras que la calidad de ésta mejore. De esta forma se espera que el número de éxitos obtenidos por la búsqueda a distancia incremente, ya que al aplicar la búsqueda lateral después de la anterior de manera iterativa, ayudará a que se escape de óptimos locales y así encontrar una mejor solución. Esta estrategia es un caso especial de la búsqueda conocida como búsqueda de vecindario de tamaño variable (Hansen *et al.*, 2006).

III.5.0.1 Algoritmo 4. DosBúsquedas

Entrada: Un conjunto de S soluciones iniciales de acuerdo al $caso(l, d)$ del motivo implantado, la longitud del motivo l y el número máximo d de mutaciones presentes en el motivo.

Salida: el mejor óptimo local

1 Para cada solución i de las S soluciones iniciales

2 Hacer

3 i'= BúsquedaADistancia(i)

4 i = BúsquedaLateral(i')

5 Mientras la calidad de i sea mejor a la de i'

6 Fin Para

7 Regresar la solución de mejor función objetivo

A diferencia del Algoritmo 2 y el Algoritmo 3 que hacen una búsqueda local en sólo un vecindario, DosBúsquedas (Algoritmo 4) busca de manera iterativa en dos vecindarios de tamaño diferente. Esto nos indica que el tiempo que tardará el algoritmo será, bajo ciertas suposiciones, lo que tarde el Algoritmo 2 más lo que tarde el Algoritmo 3 multiplicado por una variable x_i que será el número de veces que se aplicaron iterativamente las dos búsquedas en la solución i (variable que cambia de valor según la solución inicial).

La complejidad de la combinación de las dos búsquedas es la suma del costo de la búsqueda en un vecindario $D_{=1}$ y el costo de la búsqueda lateral multiplicada por el costo de las líneas 2 y 5 (ciclo mientras). El costo total del algoritmo es de $Sk(8Nlx_i 3Nl^2 y_i) = O(Skx_i y_i N^2 l^3)$ donde k es el número de iteraciones del ciclo mientras, x_i el número de veces que se aplica la búsqueda bajo vecindario $D_{=1}$ a la i-ésima solución, y_i el número de veces que se aplica la búsqueda lateral a la i-ésima solución y S el número de soluciones iniciales.

III.6 Algoritmo MbGA con Búsqueda Local (MbGALS)

Una vez implementadas las estrategias de búsqueda local, se decidió probarlas dentro del algoritmo genético de la siguiente manera: a cada iteración del algoritmo genético, la

búsqueda a distancia $D_{=1}$ es aplicada a los k mejores individuos de la nueva población antes del remplazo generacional. Posteriormente, al finalizar el algoritmo, se aplica la búsqueda a los lados al motivo arrojado por el algoritmo. Este nuevo algoritmo puede ser considerado como un caso especial de lo que se conoce como "Algoritmo Memético" (Krasnogor y Smith, 2005), el cual es una extensión de los algoritmos evolutivos que aplican procesos separados para refinar los individuos.

III.6.0.2 Algoritmo 5. MbGALS

Entrada: Número de cadenas N, longitud de las cadenas T, longitud del motivo l, número de bases mutadas d, número de individuos a los que se aplicará búsqueda local k, probabilidad de mutación P_m, probabilidad de cruzamiento P_c, tamaño de la población $PopSize$, número de jugadores en el torneo J, máximo número de iteraciones sin cambio en la aptitud del mejor individuo $Max\_Iter\_No\_Change$.

Salida: un conjunto de posiciones $\{p_1, p_2, ..., p_N\}$ y su calidad correspondiente.

1 Inicializar $PopSize$ individuos con bases aleatorias

2 Evaluar la aptitud de los $PopSize$ individuos y el mejor individuo es tomado como super individuo

3 Mientras el número de generaciones sin cambio en la aptitud sea menor a $Max\_Iter\_No\_Change$, hacer:

4 De i=1 hasta $PopSize$

5 Escoger J individuos para participar en un torneo y el ganador será P_1

6 Escoger J individuos para participar en un torneo y el ganador será P_2

7 Cruzar P_1 y P_2 con probabilidad P_c para obtener un nuevo individuo

8 Aplicar mutación a cada nuevo individuo con probabilidad P_m

9 Para cada uno de los k mejores individuos s_i de la nueva población hacer:

10 Aplicar a s_i la búsqueda a d pasos bajo el vecindario $D_{=1}$

11 Termina Para

12 Reemplazar la población actual con los nuevos $PopSize$ individuos

13 Evaluar la aptitud de la nueva población para escoger el mejor individuo y compararlo con el super individuo actual y guardar el mejor (nuevo super individuo)

14 Termina Mientras

15 Aplicar la Búsqueda Lateral a la mejor solución de la última generación

El Algoritmo 5, al igual que la versión estándar (MbGA), inicia creando una población de tamaño $PopSize$ de manera aleatoria (línea 1). Se evalúan las funciones objetivo de estas soluciones (línea 2) y se determina cuál es la mejor de ellas, es decir, el super individuo. En este algoritmo (Algoritmo 5), al igual que en el Algoritmo MbGA, se reemplazan todos los individuos de la generación actual por nuevos creados a partir de los actuales, por lo que, para cada iteración del algoritmo, se deberán crear $PopSize$ individuos nuevos (línea 4). La creación de cada nuevo individuo se hace de la misma forma que en MbGA (líneas 5,6 y 7), y también de la misma manera, se les aplica mutación a cada nueva solución con cierta probabilidad P_m (línea 8). Básicamente en lo que difiere este algoritmo con respecto a MbGA es en la línea 10, donde después de haber aplicado la mutación, se le aplica a los tres mejores individuos de la nueva generación una búsqueda de d pasos antes de hacer el reemplazo generacional (línea 12). Una vez hecho el reemplazo generacional, al igual que

en MbGA, se busca en la nueva generación el individuo de mejor aptitud (línea 13) y si éste es mejor que el super individuo, entonces el primero se convierte en el super individuo. Por último (línea 15), a la mejor solución de la última generación se le aplica la búsqueda lateral.

Ya que el Algoritmo 5 incluye una búsqueda local, comparando con MbGA, se podría pensar que tardaría más tiempo ya que se hacen más cálculos y asignaciones. Sin embargo, esta búsqueda local le permite al algoritmo genético converger más rápido por lo que necesita menos iteraciones para llegar al óptimo, lo que puede compensar lo primero. Es por esto que se espera que el algoritmo tenga tanto mejor calidad de solución como una disminución en los tiempos de ejecución.

El objetivo de aplicar la búsqueda lateral al final del algoritmo es evitar que éste arroje una solución que se encuentre ligeramente recorrida uno o más caracteres a la derecha o a la izquierda de la solución óptima.

Para calcular la complejidad de MbGALS hay que tomar en cuenta que se añadieron dos estrategias de búsqueda. La primera es la búsqueda bajo el vecindario $D_{=1}$ que añade un costo de $O(NL^2kdX)$ ya que se aplica a k soluciones d veces en un ciclo mientras que itera X veces. La búsqueda lateral se aplica a una sola solución, por lo que añade un costo de $O(Nlx)$. Estos dos costos se añaden al costo de MbGA.

III.7 Algoritmo MbGA con Búsqueda Local Combinada

Una vez combinadas las dos estrategias de búsqueda local, surge la idea de probar esta combinación dentro del algoritmo genético estándar basado en el esquema de consenso. A diferencia del genético con Búsqueda Local (MbGALS), que sólo aplica la búsqueda a d pasos a los k mejores individuos de cada nueva generación, en este nuevo algoritmo se pretende aplicar iterativamente las dos búsquedas, primero a distancia y luego lateral, a

los k mejores individuos de cada nueva generación hasta que la aptitud de éstos ya no mejore. Después de aplicar las búsquedas, se hace el reemplazo generacional y se prosigue de la misma manera que lo hace el algoritmo estándar. A continuación se muestra el pseudocódigo del algoritmo resultante.

III.7.0.3 Algoritmo 6. AG2Búsquedas

Entrada: Número de cadenas N, longitud de las cadenas T, longitud del motivo l, número de bases mutadas d, número de individuos a los que se le aplica la búsqueda local k, probabilidad de mutación P_m, probabilidad de cruzamiento P_c, tamaño de la población $PopSize$, número de jugadores en el torneo J, máximo número de iteraciones sin cambio en la aptitud del mejor individuo $Max\_Iter\_No\_Change$.
Salida: un conjunto de posiciones $\{p_1, p_2, ..., p_N\}$ y su calidad correspondiente.

```
1 Inicializar PopSize individuos con bases
  aleatorias
2 Evaluar la aptitud de los PopSize individuos y el mejor
  individuo es tomado como super individuo
3 Mientras el número de generaciones sin cambio en la aptitud
  sea menor a Max_Iter_No_Change, hacer:
4     De i=1 hasta PopSize
5         Escoger J individuos para participar en un torneo y
          el ganador será P1
6         Escoger J individuos para participar en un torneo y
          el ganador será P2
7         Cruzar P1 y P2 con probabilidad Pc para obtener un
```

```
        nuevo individuo
8     Aplicar mutación a cada nuevo individuo con probabilidad
      P_m
9     Para cada uno de los k mejores individuos s_i de la nueva
      población hacer:
10        Mientras la aptitud de s_i mejore hacer:
11            Aplicar Búsqueda a distancia a s_i para obtener s'_i
12            Aplicar Búsqueda Lateral a s'_i para obtener s_i
13        Termina Mientras
14    Termina Para
15    Reemplazar la población actual con los nuevos PopSize
      individuos
16    Evaluar la aptitud de la nueva población para escoger
      el mejor individuo y compararlo con el super individuo
      actual y guardar el mejor (nuevo super individuo)
17 Termina Mientras
```

El Algoritmo 6 presenta la combinación de las dos estrategias de búsqueda local dentro del algoritmo genético estándar. Este algoritmo, en vez de sólo aplicar una búsqueda a d pasos a los k mejores individuos de cada nueva generación como en el Algoritmo 5, les aplica a éstos una combinación iterativa de la búsqueda local bajo vecindario $D_{=1}$ y la búsqueda lateral (líneas 9, 10, 11, 12, 13 y 14). Mientras la búsqueda lateral permita mejorar la función objetivo de la solución, el algoritmo volverá a aplicar la búsqueda a distancia y posteriormente la búsqueda lateral a la misma solución. Cuando la búsqueda lateral no mejore la aptitud de la solución obtenida después de haber aplicado la búsqueda a distancia, se terminará la búsqueda para dicha solución. Después de haber aplicado la búsqueda a las k mejores soluciones, se hace el reemplazo generacional (línea 15) y se busca el mejor individuo de la nueva generación. Si éste mejora al super individuo, entonces el

primero se convierte en el nuevo super individuo (línea 16).

Para calcular la complejidad de AG2Búsquedas, tenemos que añadir al costo de MbGA lo que toma aplicar a k soluciones las dos búsquedas combinadas. Sabemos que las dos búsquedas combinadas toman $kO(x_k y_k N^2 l^3)$, donde x_k es el número de veces que se aplica la búsqueda local bajo el vecindario $D_{=1}$ a la k-ésima solución, y_k es el número de veces que se aplica la búsqueda lateral a la k-ésima solución y k el número de soluciones a las que se le aplican las dos busquedas combinadas. El costo del algoritmo es entonces el costo de MbGA + $O(kx_k y_k N^2 l^3 X)$, donde X es el número de veces que itera el algoritmo genético.

Capítulo IV

Experimentos y Resultados

En el capítulo anterior se describieron varios algoritmos propuestos para atacar el problema de búsqueda de motivos. Empezando por la versión más estándar, se probaron dos codificaciones: una basada en posiciones de inicio y otra basada en motivo candidato. Posteriormente, se introdujeron dos estrategias de búsqueda local. Primero se desarrollaron los algoritmos correspondientes a éstas y después se implementaron dentro del algoritmo genético. Por último, surge la idea de combinar las dos estrategias de búsqueda local, por lo que se desarrolla un algoritmo sólo de búsqueda local que aplica iterativamente las dos estrategias a cada una de las soluciones iniciales y finalmente, esta combinación se prueba dentro del algoritmo genético estándar MbGA.

IV.1 Experimentos realizados con los algoritmos PbGA y MbGA

Se llevó a cabo la implementación de dos algoritmos genéticos estándar bajo el modelo de motivo implantado, el primero utilizando la codificación basada en posiciones de inicio (PbGA) y el segundo utilizando la codificación basada en motivo (MbGA). El algoritmo se programó en lenguaje C++ y se llevaron a cabo diferentes corridas para evaluar su desempeño en los diferentes casos del problema implantado.

Para esto, se generaron los casos de motivos-(l, d) implantados $(10, 2)$, $(10, 3)$, $(11, 2)$, $(11, 3)$, $(12, 3)$, $(12, 4)$, $(15, 4)$, y $(16, 5)$ de la siguiente manera: primero, un motivo M de longitud l es generado tomando l bases al azar del alfabeto $\{a, c, g, t\}$, teniendo cada base una probabilidad de $1/4$ de ser elegida. Después, $N = 20$ ocurrencias del motivo son creadas

escogiendo de manera aleatoria a lo más d posiciones por ocurrencia y reemplazando éstas por otra elegida también al azar (pudiendo ser elegida la misma base que ya se encontraba en esa posición). Posteriormente, se construyeron N secuencias de fondo, cada una de longitud $T = 600$, seleccionando las bases al azar (cada base con la misma probabilidad). Finalmente, se insertó cada una de las ocurrencias en una posición aleatoria de las secuencias de fondo (una ocurrencia por secuencia de fondo).

Después de un proceso de sintonización a prueba y error, se seleccionaron los parámetros del algoritmo de la siguiente manera. Para el algoritmo basado en posiciones de inicio, la población inicial se fijó en $1,000$ individuos ($PopSize = 1000$), seleccionando tres de ellos para cada torneo ($J = 3$), y utilizando una probabilidad de mutación $P_m = 0.4$ y una probabilidad de cruzamiento $P_c = 1.0$. Para el algoritmo basado en motivo, se utilizó una población de $1,000$ individuos, utilizando torneo binario ($J = 2$) y probabilidad de mutación de 0.4. Ambos algoritmos se corrieron 30 veces para cada caso del problema. En el PbGA, se utilizó como condición de paro 100 generaciones sin cambio en la función objetivo, mientras que para MbGA se utilizaron 15 generaciones sin cambio.

Puesto que el MbGA genera un motivo candidato y el PbGA da como salida posiciones de inicio de las ocurrencias del motivo en las secuencias de entrada, necesitamos generar un motivo candidato para PbGA o un conjunto de posiciones de inicio para MbGA de tal manera que se pueda hacer una comparación entre los dos algoritmos. Se decidió tomar la segunda opción. La manera en que se generan las posiciones de inicio es tomando el motivo generado por MbGA y seleccionando la subcadena de longitud l más parecida al motivo (menor distancia de Hamming) en cada secuencia de entrada. Una vez obtenidas las posiciones de inicio, se calcula la aptitud correspondiente.

La Tabla 1 compara la calidad de solución promedio para ambos algoritmos en todos los casos. La primera columna indica el identificador de cada caso, la segunda columna da los valores promedio de la función objetivo para el algoritmo PbGA, la tercera columna da los valores promedio de la función objetivo obtenidos por MbGA y las columnas 4 y 5

Tabla I. Comparación de la función objetivo promedio $\overline{F} \pm$ desviación estándar. PbGA corresponde a Algoritmo Genético Estándar basado en posiciones de inicio y MbGA corresponde a Algoritmo Genético basado en motivo. La aptitud entre paréntesis es la mejor aptitud encontrada por el algoritmo en 30 corridas, y aquellas en negritas corresponden a aquellas que coinciden con el óptimo.

$caso(l,d)$	$\overline{F}$PbGA	$\overline{F}$MbGA
(10,2)	144.07(151) ±3.37	157.8(**166**) ±3.6
(11,2)	153.37(160) ±3.47	178.47(**193**) ±8.41
(12,3)	167.07(201) ±7.97	196.33(**210**) ±6.16
(15,4)	195.97(232) ±7.69	239.33(**250**) ±8.55
(16,5)	204.18(211) ±4.68	240.57(**254**) ±10.6

proporcionan las desviaciones estándar correspondientes.

Los resultados en la Tabla I muestran que el algoritmo basado en motivo obtiene, para el caso $(10, 2)$, soluciones más cercanas al óptimo que aquellas obtenidas por PbGA. Para este mismo caso, el algoritmo PbGA no pudo encontrar el óptimo en ninguna de las 30 ejecuciones, mientras que MbGA lo obtuvo almenos una vez. En el segundo caso $(11, 2)$ podemos ver resultados similares al primer caso. En el caso $(12, 3)$ de nuevo MbGA fué capaz de encontrar el óptimo. En el caso $(15, 4)$ MbGA obtuvo un buen desempeño ya que, como en los casos $(10, 2)$ y $(12, 3)$, las soluciones estuvieron menos alejadas del óptimo que aquellas producidas por PbGA. En contraste con lo que sucede con MbGA, el algoritmo PbGA tiene un desempeño más bajo, ya que los promedios de la función objetivo están más lejos del óptimo. En el último caso $(16, 5)$, se puede ver que los resultados obtenidos por MbGA sobrepasan aquellos obtenidos por PbGA. En todos los casos podemos ver que hay una clara mejora de MbGA sobre PbGA en términos de la calidad de la solución. Si consideramos los valores promedio, podemos ver que tenemos una mejora del 9.5% para el caso $(10, 2)$, 16.36% para el caso $(11, 2)$, 17.51% para el caso $(12, 3)$, 22.13% para el caso $(15, 4)$ y una mejora del 17.82% para el caso $(16, 5)$. Es claro que, conforme incrementa la longitud del motivo, la diferencia entre MbGA y PbGA también se incrementa.

Otro aspecto importante es el tiempo de cómputo. En la Tabla II podemos ver que para

Tabla II. Tiempos de Ejecución Promedio ($\overline{T}$) ± desviación estándar para los algoritmos PbGA y MbGA

$caso(l,d)$	$\overline{T}$PbGA	$\overline{T}$MbGA
(10,2)	11.1 ±2.58	81.67 ±17.95
(11,2)	11.77 ±3.05	91.8 ±21.57
(12,3)	12.4 ±2.34	131.5 ±28.92
(15,4)	31.1 ±10.66	155.17 ±55.55
(16,5)	12.39 ±2.35	335.1 ±95.52

ambos algoritmos, el tiempo se incrementa conforme la longitud del motivo aumenta. Sin embargo, PbGA es más rápido que MbGA en todos los casos. La razón de estos resultados es que, a pesar de que MbGA tiene una condición de paro de 15 generaciones a diferencia de PbGA que tiene 100 generaciones, el MbGA tiene una mayor sensibilidad a la longitud del motivo l y el cálculo de la función objetivo se vuelve más costoso.

Se realizó otro conjunto de experimentos, permitiéndole a PbGA correr más tiempo (aumentando el número de individuos de la población según el caso (l,d)) de tal manera que empatara en tiempo a MbGA y ver si así mejoraba su desempeño. Los resultados se muestran en la Tabla III. Podemos ver que no hay una mejora significativa del desempeño de PbGA comparando con la Tabla II. Por otra parte, podemos ver también en la Tabla III los resultados de Gibbs Sampler (Lawrence *et al.*, 1993), los cuales son equivalentes al óptimo en la mayoría de los casos, salvo para aquellos de longitud $l = 10$.

En la Tabla III podemos ver que MbGA obtuvo el óptimo al menos una vez en todos los casos mientras que PbGA sólo obtuvo el óptimo en tres de ellos. De los casos en los que el motivo empieza a aumentar considerablemente su longitud, como el caso $(18, 6)$ se puede ver que PbGA obtiene un promedio muy por debajo del óptimo. Este caso en particular se considera complicado ya que el porcentaje de bases que pudieron haber mutado abarca exactamente la tercera parte del motivo. En la Tabla III también se puede ver que Gibbs Sampler no logra obtener el óptimo en todos los casos, sólo en siete de ellos y en casos como $(18, 6)$ y $(30, 11)$ tuvo resultados muy por debajo del óptimo. Para verificar

Tabla III. Comparación de la aptitud total promedio ($\overline{F}$) obtenido por PbGA permitiéndole correr más tiempo, el MbGA, y Gibbs Sampler. La aptitud entre paréntesis es la mejor aptitud encontrada por el algoritmo en 30 corridas, y aquellas en negritas corresponden a la mejor solución encontrada que corresponde con el óptimo.

$caso(l,d)$	Num. Indiv. PbGA	$\overline{F}$PbGA	$\overline{F}$MbGA	Gibbs Sampler
(10,2)	9,000	145.97(153)	157.8(**166**)	160
(10,3)	8,000	148.13(155)	155.57(**158**)	144
(11,2)	10,000	159.17(**193**)	178.47(**193**)	**193**
(11,3)	9,000	156.7(168)	173.67(**176**)	**176**
(12,3)	13,000	176.57(**210**)	196.33(**210**)	**210**
(12,4)	16,000	167.3(173)	178.2(**184**)	**184**
(15,4)	16,000	199.8(240)	239.33(**250**)	**250**
(16,5)	30,000	209.4(246)	240.57(**254**)	**254**
(18,6)	16,000	229.07(272)	238.53(**279**)	209
(20,7)	18,000	245.3(255)	265.5(**300**)	296
(30,11)	17,000	342.03(443)	375.93(**454**)	314
(40,15)	40,000	543.8(**613**)	584.13(**613**)	**613**

si hay o no una diferencia significativa entre los promedios de las aptitudes obtenidas por los dos algoritmos, se realizó un análisis estadístico. En el caso de estos dos algoritmos, los resultados del análisis estadístico (ver Apéndice A) nos dicen que hay una diferencia significativa entre los promedios al usar una codificación basada en posiciones de inicio y una codificación basada en el motivo, obteniéndose resultados de mejor calidad con la segunda.

En la Tabla IV se puede observar que al utilizar una codificación que es más sensible a los cambios tanto en la longitud como a aquellos producidos por la mutación, el cálculo de la función objetivo se vuelve más costoso y se incrementa el número de iteraciones del algoritmo. También se puede observar en la Tabla IV que para la representación basada en posiciones de inicio, los tiempos de ejecución de una corrida a otra son menos variables (desviación estándar más pequeña) que la basada en motivo. Los tiempos de Gibbs Sampler no se muestran puesto que el algoritmo se ejecutó en forma remota[1], por lo que no se obtuvo

[1]http://bayesweb.wadsworth.org/gibbs/gibbs.html

Tabla IV. Tiempos de Ejecución Promedio y desviación estándar de PbGA y MbGA.

$caso(l,d)$	$\overline{T}$PbGA	$\overline{T}$MbGA
(10,2)	85.1±7.06	81.67±17.95
(10,3)	100.07±54.71	69.3±16.14
(11,2)	98.63±10.78	91.8±21.57
(11,3)	85±7.98	87.97±13.99
(12,3)	138.5±25.56	131.5±28.92
(12,4)	190.07±69.53	157.63±94.13
(15,4)	191.8±44.9	155.17±55.55
(16,5)	385.3±135.81	335.1±95.52
(18,6)	241.37±82.62	159.87±45.57
(20,7)	242.97±62.64	194.6±52.97
(30,11)	283.17±75.4	337.13±96.87
(40,15)	587.17±105.12	518.27±43.53

registro de los tiempos.

Después de experimentar con casos artificiales, se decidió probar los dos algoritmos en un caso real. Éste consta de 18 secuencias de 105 bases cada una. Algunas secuencias tienen más de una ocurrencia (secuencias 5 y 18). Este caso es conocido como el sitio de acoplamiento CRP y se tomó de Stormo y Hartzell III (1989). La Tabla V muestra los resultados de este caso. En el primer renglón, se muestran las posiciones reales de las ocurrencias en las secuencias de entrada y su aptitud correspondiente. En el segundo renglón, se pueden ver las posiciones correspondientes a la aptitud más alta obtenida por el algoritmo. Por último, en el tercer renglón podemos ver la solución más cercana a la real. Esta solución coincide con más posiciones reales que la solución de mejor aptitud encontrada por MbGA, sin embargo, podemos ver que la aptitud de ésta es menor. Esto indica que no siempre las aptitudes más altas nos llevan a la solución del problema.

En la Tabla V se puede ver que PbGA y MbGA produjeron resultados similares. PbGA predice una posición más que MbGA. Sin embargo, MbGA arroja la mejor solución en términos de su aptitud. Es interesante notar que los algoritmos arrojan soluciones de mejor aptitud que la producida por la solución correspondiente a las ocurrencias del motivo

Tabla V. Comparación del desempeño obtenido por PbGA y MbGA en un caso real. El sitio de acoplamiento CRP (Stormo y Hartzell III, 1989)

Sec.	Pos. Real	PbGA Mejor Sol.	Desv.	MbGA Mejor Sol.	Desv.
1	61,17	61	0	45	-16
2	55,17	55	0	55	0
3	76	31	-45	76	0
4	63	63	0	63	0
5	50	81	31	50	0
6	7,60	7	0	7	0
7	42	24	-18	24	-18
8	39	66	27	66	27
9	9,80	9	0	9	0
10	14	14	0	14	0
11	61	61	0	29	-32
12	41	41	0	41	0
13	48	48	0	48	0
14	71	71	0	71	0
15	17	17	0	6	-11
16	53	53	0	53	0
17	1,84	5	4	75	-9
18	78	79	1	8	-70
Puntaje	240	241		246	

verdadero.

IV.2 Experimentos realizados con la Búsqueda a Distancia y la Búsqueda Lateral implementadas en MbGA

Al ver los resultados obtenidos por los algoritmos estándar PbGA y MbGA, surge la idea de implementar en MbGA (que obtuvo mejores resultados) dos estrategias de búsqueda local (algoritmo memético (Krasnogor y Smith, 2005)). La primera de éstas es la búsqueda a distancia d bajo el vecindario $D_{=1}$ (ver Algoritmo 2). Ésta es implementada dentro del genético justo antes de hacer el reemplazo generacional, aplicando una búsqueda de d pasos a los tres mejores individuos de la nueva población ($k = 3$). La segunda estrategia, búsqueda lateral (Algoritmo 3), es implementada justo al final del algoritmo, esto con el propósito de evitar que se tenga una solución que esté ligeramente recorrida un caracter a la izquierda o la derecha (característica que se observó en los resultados de los primeros experimentos con MbGA).

Primero se hizo una prueba utilizando sólo la búsqueda a distancia en el MbGA utilizando tres de los casos usados anteriormente $(10, 2)$, $(12, 3)$ y $(15, 4)$. La población inicial se fijó a $1,000$ individuos con probabilidad de mutación de $P_m = 0.5$ y se utilizó un torneo de tres jugadores. Los resultados se muestran en la Tabla VI. Nótese que ahora las funciones objetivo que se muestran en esta tabla y las siguientes son con base en la distancia de Hamming, ya que no se están manejando posiciones de inicio si no motivos, buscando por lo tanto minimizar dicha función objetivo.

Se puede ver en las Tablas VI y VII que tanto funciones objetivo promedio como tiempos de ejecución disminuyeron. Ya que la búsqueda local le permite al algoritmo explorar todo

Tabla VI. Comparación de la función objetivo promedio ± desviación estándar obtenida por el AG estándar y con Búsqueda de d pasos bajo el vecindario $D_{=1}$.

caso(l,d)	MbGA	AG con Búsqueda $D_{=1}$
(10-2)	42.23 ±3.6	36.67 ±2.37
(12-3)	45.33 ±6.16	33.17 ±5.9
(15-4)	63.3 ±8.55	51.83 ±3.73

Tabla VII. Comparación de los tiempos de ejecución (promedio y desviación estándar) del AG estándar y con Búsqueda de d pasos bajo el vecindario $D_{=1}$.

caso(l,d)	MbGA	AG con Búsqueda $D_{=1}$
(10-2)	81.67 ±17.95	67.23 ±13.37
(12-3)	131.5 ±28.92	72.37 ±9.85
(15-4)	155.17 ±55.55	109.63 ±18.43

el campo de soluciones cercanas (a distancia $D_{=1}$) a las actuales, éste puede moverse hacia aquellas soluciones vecinas que son mucho mejores en cuanto a función objetivo variando sólo algunas bases; soluciones a las cuales el algoritmo estándar llegaría más lentamente o no llegaría. En cuanto al tiempo de ejecución (Tabla VII), aunque la búsqueda implica un mayor número de comparaciones y asignaciones, ésta le permite al algoritmo converger más rápido hacia algún óptimo local, que, en un gran porcentaje de las ejecuciones, es el óptimo global. En aquellas que no logra llegar al óptimo global, éste arroja una solución muy cercana, que se encuentra ligeramente recorrida hacia la derecha o a la izquierda un par de caracteres, coincidiendo en un gran porcentaje de bases con el motivo buscado.

Se puede ver en la Tabla VII que las desviaciones estándar para los casos $(10, 2)$ y $(12, 3)$ son más pequeñas en el algoritmo MbGA con búsqueda a d pasos bajo el vecindario $D_{=1}$ que el algoritmo MbGA solo, esto indica que los tiempos en esos casos no varían tanto en MbGA con búsqueda a d pasos como con MbGA. Para el caso $(15, 4)$ los tiempos varían considerablemente, incluso en algunas corridas se obtienen tiempos más grandes que aquellos obtenidos por MbGA.

El análisis estadístico que se hizo comparando el algoritmo genético estándar MbGA y MbGA con la búsqueda a d pasos (ver detalles en el Apéndice A) muestra que hay

Tabla VIII. Comparación de la función objetivo promedio obtenida por el AG estándar, con Búsqueda a d pasos bajo el vecindario $D_{=1}$ y con Búsqueda Lateral.

caso(l,d)	MbGA	AG con Búsqueda $D_{=1}$	AG con dos Búsquedas
(10-2)	42.23 ±3.6	36.67 ±2.37	35.57 ±3.18
(12-3)	45.33 ±6.16	33.17 ±5.9	30 ±0
(15-4)	63.3 ±8.55	51.83 ±3.73	50.27 ±1.46

una diferencia significativa entre los resultados obtenidos por el algoritmo estándar y el algoritmo que aplica la estrategia de búsqueda local, lo que confirma que el uso de otras estrategias dentro del genético pueden ayudar en gran medida a mejorar la calidad de solución y tiempos de una implementación estándar.

Al ver que la mayoría de las soluciones no óptimas obtenidas por la implementación de la búsqueda a d pasos bajo el vecindario $D_{=1}$ están tan sólo recorridas uno o dos caracteres a la derecha o a la izquierda, se decidió implementar la búsqueda lateral al motivo producto del algoritmo probado anteriormente (MbGA con búsqueda a d pasos bajo el vecindario $D_{=1}$, ver Algoritmo 5). Se realizaron de nuevo varios experimentos para medir el desempeño de éste utilizando los mismos parámetros que en la implementación anterior. Se probó también disminuyendo la población inicial a 400 individuos y ya que la calidad de las soluciones no cambió considerablemente, se presentan sólo los resultados obtenidos con 400 individuos.

En la Tabla VIII se presenta la comparación entre las funciones objetivo promedio obtenidas por el algoritmo MbGA, MbGA con búsqueda a d pasos bajo el vecindario $D_{=1}$ y MbGA con las dos búsquedas, a distancia y lateral, en los casos $(10, 2)$, $(12, 3)$ y $(15, 4)$. Se puede observar, una vez más, cómo mejora el desempeño de MbGA implementando las dos búsquedas. Incluso se puede ver que para el caso $(12, 3)$ se obtiene el óptimo en las 30 corridas. Esto quiere decir que la búsqueda lateral está permitiendo llegar al óptimo en aquellos casos que el motivo encontrado por el genético no es el óptimo debido a que se encuentra recorrido.

Tabla IX. Comparación de los tiempos de ejecución (promedio y desviación estándar) del AG estándar, con Búsqueda a d pasos bajo el vecindario $D_{=1}$ y con dos búsquedas.

caso(l,d)	MbGA	AG con Búsqueda $D_{=1}$	AG con dos Búsquedas
(10-2)	81.67 ±17.95	67.23 ±13.37	78.27 ±27.42
(12-3)	131.5 ±28.92	72.37 ±9.85	88.17 ±12.35
(15-4)	155.17 ±55.55	109.63 ±18.43	128.73 ±18.73

La Tabla IX muestra la comparación de los tiempos de ejecución promedio que obtuvieron MbGA, MbGA con búsqueda a d pasos bajo el vecindario $D_{=1}$ y MbGA con las dos búsquedas en los casos $(10, 2)$, $(12, 3)$ y $(15, 4)$. Se puede observar que el AG con las dos búsquedas implementadas aumentó los tiempos de ejecución del MbGA con búsqueda a distancia, sin embargo, al ver que el algoritmo que implementa las dos búsquedas obtiene un buen desempeño, podemos pensar en correr este algoritmo con una población inicial menor, de tal manera que se pueda conservar el nivel de desempeño del algoritmo, disminuyendo los tiempos de ejecución. Podemos observar que las diferencias en tiempo de ejecución para los tres casos permance igual, de hecho, disminuye un poco para el caso $(15, 4)$. En cuanto a la desviación estándar, se puede observar que ésta disminuye y la diferencia entre MbGA con y sin búsqueda a distancia es más grande en el caso más grande, lo cual nos indica que MbGA con las dos búsquedas presenta menos variaciones en sus tiempos para todos los casos.

El análisis estadístico que se llevó a cabo entre MbGA con una búsqueda y MbGA con las dos búsquedas (ver detalles en el Apéndice A) muestran que hay una diferencia significativa entre los promedios obtenidos por una y otra implementación, obteniendo un mejor desempeño con el algoritmo que implementa las dos búsquedas. Inicialmente, la búsqueda lateral fué implementada para mejorar aquellas soluciones que la búsqueda a d pasos por sí sola no podía. Tantos los resultados de los experimentos como el análisis estadístico nos indican que este objetivo fué logrado.

Como se puede ver en la Figura 2, al implementar las dos búsquedas en el algoritmo

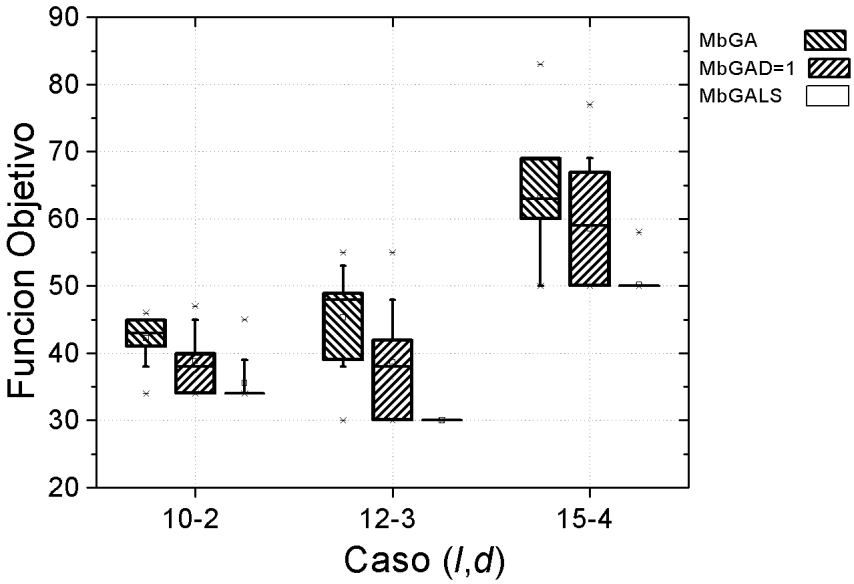

Figura 2. Funciones objetivo obtenidas por los tres algoritmos genéticos implementados

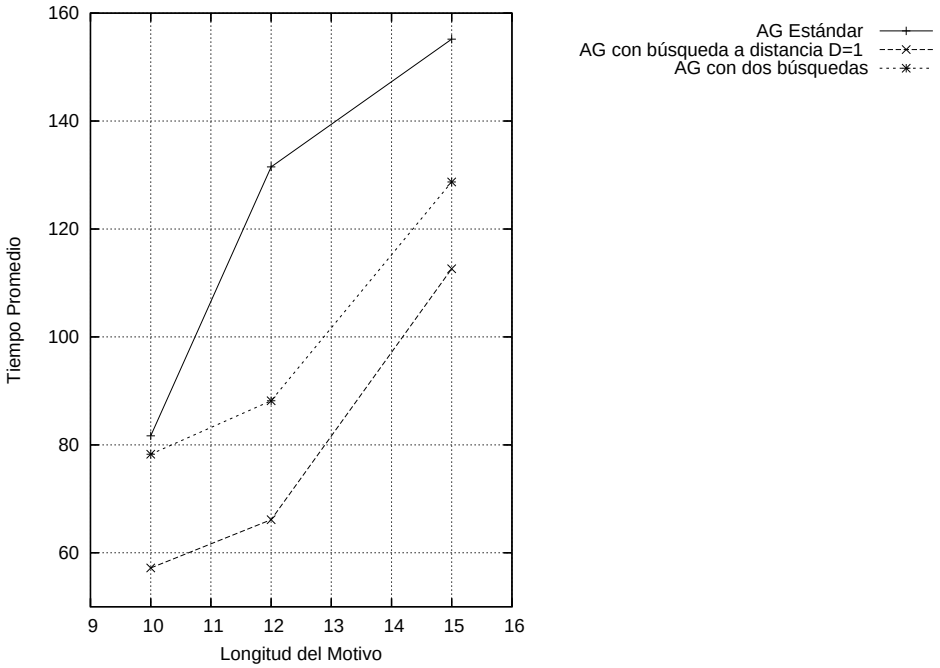

Figura 3. Tiempos promedio obtenidos por los tres algoritmos genéticos implementados

genético, se disminuyeron los promedios de las funciones objetivo obtenidas por el algoritmo estándar, llegando, para estos tres casos mostrados, al óptimo global en casi las 30 ejecuciones del algoritmo. Los tiempos de ejecución, mostrados en la Figura 3, disminuyen a más de la mitad del tiempo obtenido por el algoritmo estándar. A pesar de que las estrategias de búsqueda implican que el algoritmo haga más comparaciones y asignaciones, aplicar estas estrategias le permite a éste converger más pronto a la solución, por lo que se requieren menos generaciones para llegar a la solución óptima.

IV.3 Experimentos con la Búsqueda Local bajo el vecindario $D_{=1}$ y la Búsqueda Lateral

Los resultados obtenidos implementando las búsquedas locales en el algoritmo genético fueron muy buenos ya que en un gran porcentaje de los experimentos realizados se obtuvo el óptimo global. Con esto, surge la pregunta si es la búsqueda local la que hace todo el trabajo o si en realidad el algoritmo genético juega un papel importante en el éxito del algoritmo. Para aclarar este punto se prueban las búsquedas locales solas sobre un conjunto de soluciones iniciales.

IV.3.1 Búsqueda bajo el vecindario $D_{=1}$

A diferencia de la búsqueda implementada en el algoritmo genético, para probar sólamente la búsqueda local bajo el vecindario $D_{=1}$, en lugar de que se cambien sólo d bases, el algoritmo cambiará cuantas bases sean necesarias hasta que se encuentre un óptimo local. Se elaboraron dos tipos de experimentos; en el primero, siempre se busca moverse hacia una solución estrictamente mejor a la que tenemos. En el segundo, se busca moverse a un vecino mejor o igual a la solución actual, guardando un registro de la historia del motivo, para no caer de nuevo en el mismo motivo y así evitar que se cicle el algoritmo. Este

registro se inicia cuando se aplica por primera vez la búsqueda local bajo el vecindario $D_{=1}$ en la solución, guardando todos los cambios por los que pasa la solución hasta que la búsqueda termina. Al cambiar la solución que tenemos por una vecina de igual calidad, se guarda en memoria la solución anterior para que al volver a aplicar la búsqueda local bajo el vecindario $D_{=1}$ en la nueva solución, no se vuelva a caer en la solución anterior por tener la misma calidad, evitando así que el algoritmo quede en un ciclo infinito.

IV.3.2 Búsqueda Lateral

La búsqueda lateral es implementada de la misma manera que en el algoritmo genético. El recorrimiento a la izquierda o derecha se repetirá una y otra vez hasta que ya no se mejore la calidad de la solución. En este caso, también se diseñaron dos experimentos. En el primero se busca aquella solución que sea estrictamente mejor a la actual y en el segundo se busca alguna que sea mejor o igual, sin caer, de nuevo, en motivos repetidos.

IV.3.3 Resultados con las Búquedas Locales

Cada uno de los cuatro experimentos explicados anteriormente parte de las mismas 1000 soluciones iniciales y arroja como salida 1000 soluciones obtenidas después de aplicar la búsqueda local. Los cuatro algoritmos corrieron en los ocho casos manejados anteriormente: $(10, 2)$, $(12, 3)$, $(15, 4)$, $(16, 5)$, $(18, 6)$, $(20, 7)$, $(30, 11)$ y $(40, 15)$. Los resultados obtenidos por ambas búsquedas se muestran en las tablas X, XI, XII y XIII.

La Tabla X muestra las calidades promedio obtenidas en los cuatro experimentos de búsqueda local. La primera columna especifica la longitud del motivo y el número de posiciones mutadas. La segunda columna muestra los resultados obtenidos por el algoritmo de búsqueda local bajo el vecindario $D_{=1}$ que se mueve a soluciones estrictamente mejores a la actual ($<$). La tercera columna muestra los resultados obtenidos por el algoritmo de búsqueda lateral que se mueve a soluciones estrictamente mejores a la actual. La cuarta

Tabla X. Valores de función objetivo promedio obtenidos en 1000 corridas de la Búsqueda a Distancia y la Búsqueda Lateral.

$caso(l,d)$	B. Dist=1 (<)	B. Lat. (<)	B. Dist=1 (<=)	B. Lat. (<=)
(10,2)	48.71(**34**)	52.49(38)	47.83(**34**)	51.53(**34**)
(12,3)	66.02(**30**)	73.03(**30**)	64.38(**30**)	71.76(**30**)
(15,4)	97.17(**50**)	106.35(75)	94.75(**50**)	105.08(75)
(16,5)	109.37(**66**)	117.89(105)	107.35(**66**)	116.55(**66**)
(18,6)	130.68(**81**)	140.98(115)	128.40(**81**)	139.59(116)
(20,7)	152.41(**100**)	164.27(121)	149.85(**100**)	162.75(121)
(30,11)	257.37(**146**)	286.33(252)	250.96(**146**)	284.38(169)
(40,15)	358.13(**187**)	408.66(337)	347.75(**187**)	407.25(322)

columna muestra los resultados obtenidos por la búsqueda a distancia que se mueve a soluciones iguales o mejores a la actual, y por último, la quinta columna muestra los resultados obtenidos por el algoritmo de búsqueda lateral que se mueve a soluciones iguales o mejores a la actual (<=). Las soluciones entre paréntesis muestran la mejor solución obtenida por el algoritmo y aquellas en negritas coiciden con la solución óptima. Podemos ver en la Tabla X que la búsqueda a distancia en sus dos implementaciones obtiene en todos los casos, al menos una vez, el óptimo. También se puede observar que los resultados obtenidos por la búsqueda lateral que se mueve a soluciones iguales o mejores a la actual (<=) fueron mejores a la búsqueda lateral que sólo se mueve a soluciones estrictamente mejores (<). Esto se debe a que la primera le permite moverse con más facilidad a otras soluciones que lo pueden llevar a un óptimo local mejor.

La Tabla XI muestra el número de éxitos que obtuvo cada algoritmo de búsqueda local en las $1,000$ corridas en cada uno de los casos. Se puede ver que la búsqueda a distancia que se mueve a soluciones iguales o mejores a la actual (<=) tuvo el mayor número de éxitos, aunque tardó más que cualquier otro (Tabla XIII). En la Tabla XII se muestran las desviaciones estándar de las funciones objetivo promedio obtenidas por los cuatro algoritmos en los ocho casos. Se puede observar que la búsqueda lateral tiene desviaciones más pequeñas pero sus resultados no son mejores que la búsqueda a distancia.

Tabla XI. Número de éxitos obtenidos en 1,000 corridas de la Búsqueda a Distancia y la Búsqueda Lateral en sus dos implementaciones

$caso(l,d)$	B. Dist=1 (<)	B. Lat. (<)	B. Dist=1 (<=)	B. Lat. (<=)
(10,2)	12	0	14	8
(12,3)	23	1	25	2
(15,4)	7	0	17	0
(16,5)	4	0	7	1
(18,6)	2	0	7	0
(20,7)	5	0	6	0
(30,11)	6	0	13	0
(40,15)	4	0	7	0

Tabla XII. Desviaciones estándar de la función objetivo promedio obtenidas en 1000 corridas de la búsqueda local

$caso(l,d)$	B. Dist=1 (<)	B. Lat. (<)	B. Dist=1 (<=)	B. Lat. (<=)
(10,2)	3.11	2.64	3.15	2.93
(12,3)	8.59	3.98	8.94	4.75
(15,4)	9.10	3.93	10.71	4.32
(16,5)	5.93	3.51	6.59	3.79
(18,6)	7.04	3.95	7.86	3.69
(20,7)	7.16	4.06	8.15	4.19
(30,11)	27.00	5.18	30.41	6.30
(40,15)	50.44	6.41	54.82	7.03

La Tabla XIII muestra el tiempo acumulado que tardó cada algoritmo en correr para 1000 soluciones iniciales en los ocho casos. Los tiempos más pequeños son obtenidos por la búsqueda lateral, debido a que ésta explora en un vecindario más pequeño (ver sección III.4); es por esta misma razón que tiene menos probabilidades de llegar a soluciones óptimas partiendo de soluciones aleatorias.

En general, se puede ver que la búsqueda local bajo el vecindario $D_{=1}$ obtiene mejores resultados que la búsqueda lateral en sus dos implementaciones. Esto es razonable dado que el espacio de búsqueda que explora la búsqueda local bajo el vecindario $D_{=1}$ es mucho mayor ($3l$) que la búsqueda lateral (8). A pesar de que de las 1000 corridas sólo un

Tabla XIII. Tiempos de ejecución en segundos de 1000 corridas de búsqueda local.

$caso(l,d)$	B. Dist=1 (<)	B. Lat. (<)	B. Dist=1 (<=)	B. Lat. (<=)
(10,2)	228	50	318	75
(12,3)	377	62	515	96
(15,4)	638	78	928	115
(16,5)	727	82	1071	123
(18,6)	992	92	1470	141
(20,7)	1345	104	1962	158
(30,11)	4604	159	6698	242
(40,15)	11083	184	16024	269

pequeño porcentaje logró llegar al óptimo global, en los cuatro algoritmos, éstos son mucho más rápidos que el algoritmo genético, por lo que podemos pensar que si le permitimos a la búsqueda local correr el tiempo que el algoritmo genético tarda en realizar una corrida, podría posiblemente encontrar varias veces el óptimo cuando el anterior sólo lo encuentra una vez.

IV.4 Experimentos con la Búqueda Local bajo el vecindario $D_{=1}$ y Búsqueda Lateral Combinadas

Ya que los resultados de las estrategias de búsqueda local en los experimentos anteriores fueron prometedores, se decidió implementar un algoritmo que combinara las dos estrategias de búsqueda de la siguiente manera.

Dado un conjunto inicial de soluciones, se aplicará de manera iterativa a cada solución la búsqueda local bajo el vecindario $D_{=1}$ y posteriormente la búsqueda a los lados mientras que la calidad de ésta mejore después de aplicar la búsqueda lateral. De esta manera, se espera que el número de éxitos obtenidos por la búsqueda local se incremente, ya que al aplicar la búsqueda lateral después de la búsqueda local nos puede permitir escapar de los óptimos locales de los vecindarios individuales correspondientes.

Para ver el desempeño de este nuevo algoritmo, se hicieron experimentos con ocho casos de motivos implantados, los mismos utilizados para la búsqueda bajo el vecindario $D_{=1}$, búsqueda lateral y algoritmo genético MbGA con dos búsquedas. En cada experimento se crearon 1000 soluciones iniciales y a cada una de ellas se les aplicó iterativamente las dos búsquedas combinadas moviéndose siempre a soluciones iguales o mejores a la actual hasta llegar a un óptimo local. Para estos experimentos, el algoritmo genético se corrió utilizando 400 individuos como población inicial, torneo binario ($J = 2$) y $P_m = 0.3$. Los resultados se presentan en las siguientes tablas.

En la Tabla XIV se muestran las funciones objetivo promedio obtenidas por las estrategias de búsqueda local, el algoritmo genético con las dos búsquedas y las dos búsquedas combinadas. Los resultados entre paréntesis son la mejor solución obtenida por los algoritmos y aquellas en negritas son la solución óptima al caso correspondiente. En esta tabla se puede observar que de los algoritmos de búsqueda local, la búsqueda combinada obtiene los mejores promedios, confirmando los resultados obtenidos en los experimentos anteriores. En cuanto al genético, podemos ver que disminuyendo la población, la calidad de las soluciones no varía mucho, lo cual es conveniente, puesto que nos permite disminuir los tiempos de manera considerable.

La Tabla XV muestra el número de éxitos obtenidos en 1000 corridas de los algoritmos de búsqueda local y 30 del genético con dos búsquedas. En la Tabla XVI se muestran las desviaciones estándar de las funciones objetivo promedio obtenidas por los cuatro algoritmos de búsqueda local, el MbGA con dos búsquedas y las dos búsquedas combinadas. En ésta se puede ver que el algoritmo genético es el que tiene las desviaciones estándar más pequeñas y los mejores resultados, por lo que podemos decir que en las 30 corridas, MbGA obtuvo resultados iguales al óptimo o muy cercanos a él. En la Tabla XVII se muestran los tiempos de ejecución. Para los cuatro algoritmos de búsqueda local se muestra el tiempo acumulado de 1000 corridas. Para el algoritmo genético con las dos búsquedas se muestra el tiempo acumulado de 30 corridas y para las dos búsquedas combinadas se muestra el resultado

Tabla XIV. Funciones objetivo promedio obtenidas en 1000 corridas de la Búsqueda a Distancia (<=), la Búsqueda Combinada y 30 corridas del algoritmo genético (MbGA con Búsqueda a Distancia y Búsqueda Lateral).

$caso(l,d)$	Búsq. $D_{=1}(<=)$	AG(1000 ind.)	AG(400 ind.)	DosBúsq.
(10,2)	47.83(**34**)	34.84(**34**)	35.57(**34**)	46.63(**34**)
(12,3)	64.38(**30**)	30(**30**)	30(**30**)	61.16(**30**)
(15,4)	94.75(**50**)	50(**50**)	50.27(**50**)	91.33(**50**)
(16,5)	107.35(**66**)	67.13(**66**)	67.2(**66**)	105.84(**66**)
(18,6)	128.40(**81**)	84.9(**81**)	88.17(**81**)	126.88(**81**)
(20,7)	149.85(**100**)	108.47(**100**)	111.07(**100**)	148.19(**100**)
(30,11)	250.96(**146**)	149.5(**146**)	160.7(**146**)	244.63(**146**)
(40,15)	347.75(**187**)	189.3(**187**)	193.17(**187**)	334.41(**187**)

Tabla XV. Número de éxitos obtenidos en 30 corridas del algoritmo genético con búsqueda a distancia y búsqueda lateral y 1,000 de la Búsqueda a Distancia y la Búsqueda Combinada

$caso(l,d)$	Búsq. $D_{=1}(<=)$	AG(1000 ind.)	AG(400 ind.)	DosBúsq.
(10,2)	14/1000	26/30	24/30	102/1000
(12,3)	25/1000	30/30	30/30	160/1000
(15,4)	17/1000	30/30	29/30	134/1000
(16,5)	7/1000	29/30	29/30	53/1000
(18,6)	7/1000	25/30	22/30	52/1000
(20,7)	6/1000	21/30	21/30	52/1000
(30,11)	13/1000	24/30	15/30	147/1000
(40,15)	7/1000	23/30	19/30	207/1000

acumulado de las 1000 corridas.

Los resultados de las tablas mostradas anteriormente (XIV, XV, XVI y XVII) se pueden ver resumidos en las siguientes figuras. En las figuras 4, 5 y 6 se puede ver que aún con la combinación de las dos búsquedas, los promedios de las calidades obtenidas siguen siendo altos en comparación con el algoritmo genético. Esto significa que la gran mayoría de las soluciones iniciales que no llegaron a ser óptimos globales quedaron con una calidad baja (recordando que se busca minimizar). Sin embargo, como se puede ver en la Figura 7, la cantidad de éxitos aumenta y los tiempos de ejecución mostrados en la Figura 8 siguen

Tabla XVI. Desviaciones estándar de las funciones objetivo promedio obtenidas en 1000 corridas de la búsqueda local y 30 del algoritmo genético (MbGA con Búsqueda a d pasos bajo el vecindario $D_{=1}$ y Búsqueda Lateral)

$caso(l,d)$	Búsq. $D_{=1}(<=)$	AG(1000 ind.)	AG(400 ind.)	DosBúsq.
(10,2)	3.15	2.31	3.18	4.59
(12,3)	8.94	0	0	13.78
(15,4)	10.71	0	1.46	16.45
(16,5)	6.59	6.21	6.57	9.81
(18,6)	7.86	10.52	14.54	11.17
(20,7)	8.15	15.95	18.48	11.76
(30,11)	30.41	10.38	26.78	41.19
(40,15)	54.82	5.63	10.28	75.54

Tabla XVII. Tiempos de ejecución en segundos de 1000 corridas de búsqueda local y tiempo de ejecución para 30 corridas del algoritmo genético (MbGA con Búsqueda a Distancia y Búsqueda Lateral).

$caso(l,d)$	Búsq. $D_{=1}(<=)$	AG(1000 ind.)	AG(400 ind.)	Dos Búsq.
(10,2)	318	2348	1008	385
(12,3)	515	2645	1262	596
(15,4)	928	3862	2091	1043
(16,5)	1071	4362	2786	1186
(18,6)	1470	6758	3711	1605
(20,7)	1962	7887	4788	2128
(30,11)	6698	16121	13456	7177
(40,15)	16024	35937	32046	16864

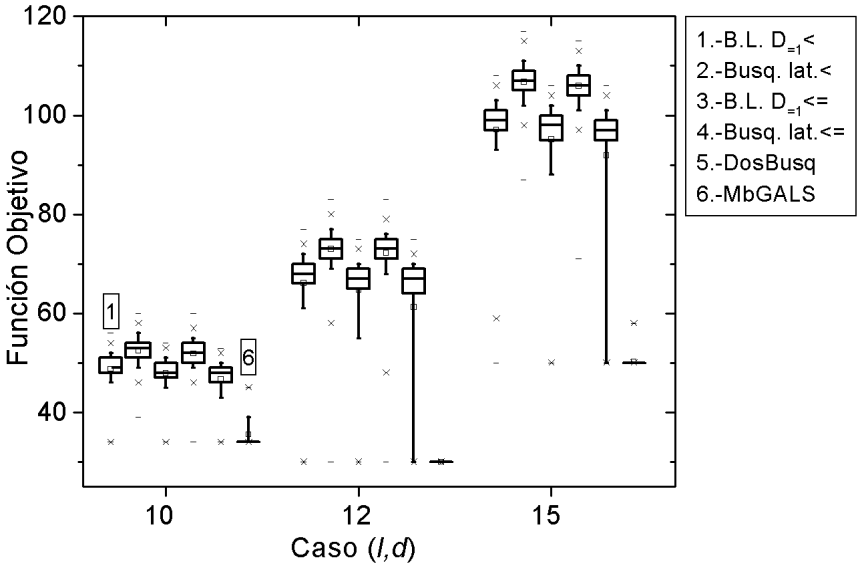

Figura 4. Calidades promedio obtenidas por la búsqueda local y el algoritmo genético (MbGA con búsqueda local bajo el vecindario $D_{=1}$ y búsqueda lateral) casos $(10, 2)$ $(12, 3)$ y $(15, 4)$

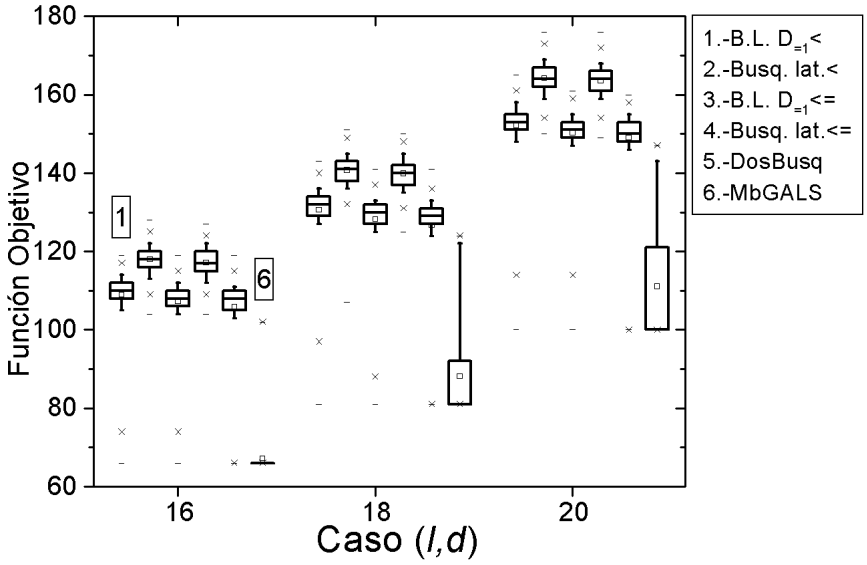

Figura 5. Calidades promedio obtenidas por la búsqueda local y el algoritmo genético (MbGA con búsqueda local bajo el vecindario $D_{=1}$ y búsqueda lateral) casos $(16, 5)$ $(18, 6)$ y $(20, 7)$

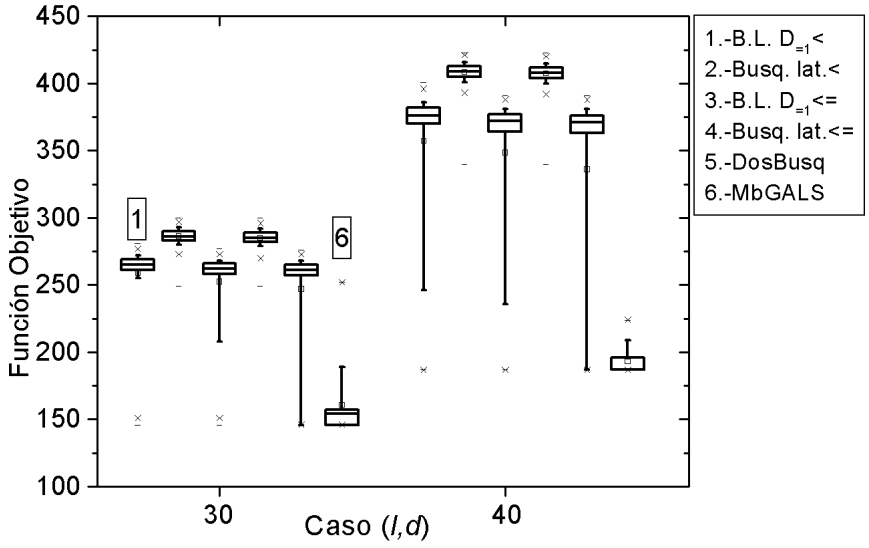

Figura 6. Calidades promedio obtenidas por la búsqueda local y el algoritmo genético (MbGA con búsqueda local bajo el vecindario $D_{=1}$ y búsqueda lateral) casos $(30, 11)$ y $(40, 15)$

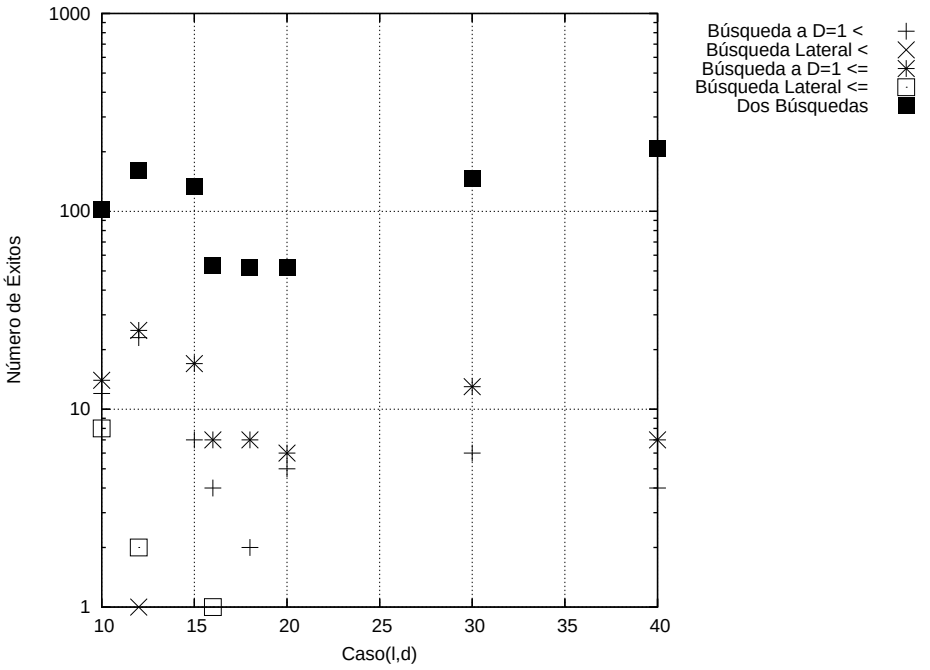

Figura 7. Número de éxitos obtenidos por la búsqueda local

siendo mucho menores que los del algoritmo genético, lo que nos hace pensar que si a la combinación de dos búsquedas le damos el tiempo de ejecución de una corrida del genético, se obtendrían más de un par de éxitos mientras que el genético no nos puede asegurar ni siquiera uno en la mayoría de los casos. En la Figura 9 se puede ver que tanto las dos búsquedas combinadas como el algoritmo genético obtienen para todos los casos el óptimo (la solución al problema).

Para probar que la combinación de dos búsquedas es mucho mejor que cualquiera de las dos búsquedas solas, se hizo una estimación del número de éxitos que estas últimas tendrían si se les diera el mismo tiempo de ejecución que a la combinación de dos búsquedas. Como se puede ver en la Figura 10, aún así no es suficiente para que las búsquedas locales obtengan tantos éxitos como combinando las dos.

Para asegurar que la combinación de dos búsquedas tiene mejor desempeño que el algoritmo genético, se calculó el valor esperado de éxitos que el primero tendría si se le diera el tiempo de una corrida del segundo. Se puede ver claramente en la Figura 11 que

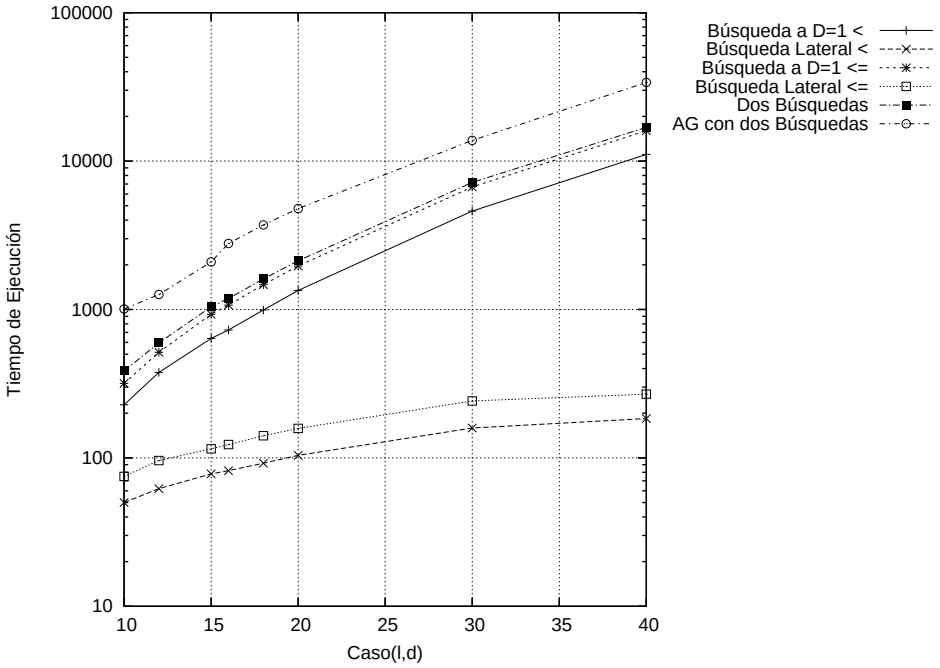

Figura 8. Tiempos obtenidos por la búsqueda local (1000 corridas) y el algoritmo genético (MbGA con búsquedas local bajo el vecindario $D_{=1}$ y lateral 30 corridas)

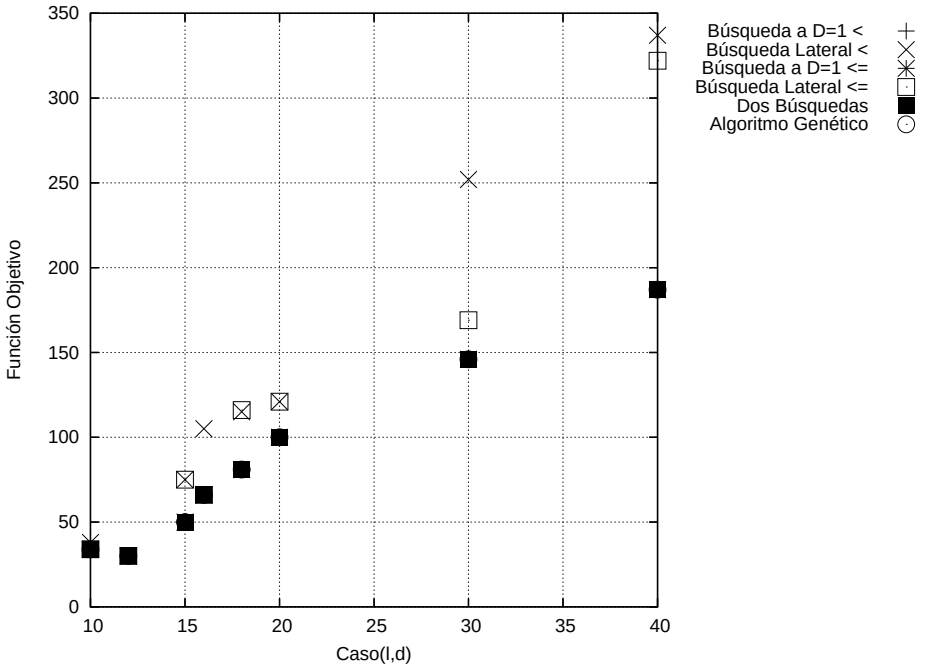

Figura 9. Mejores funciones objetivo obtenidas por la búsqueda local y el algoritmo genético (MbGA con búsqueda local bajo el vecindario $D_{=1}$ y búsqueda lateral)

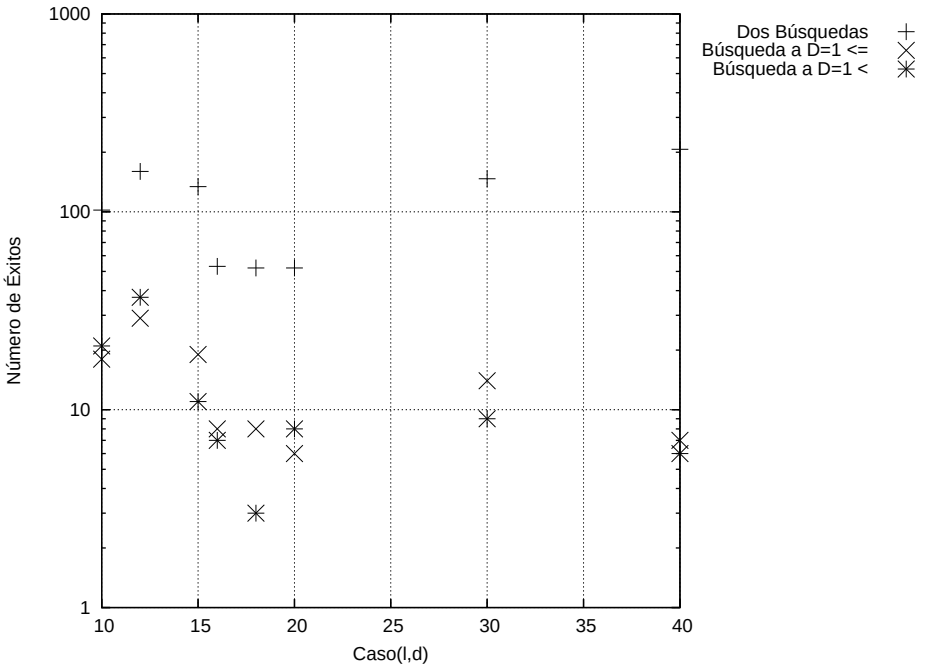

Figura 10. Número de éxitos esperados de la búsqueda local bajo el vecindario $D_{=1}$ empatando en tiempo a la combinación de dos búsquedas

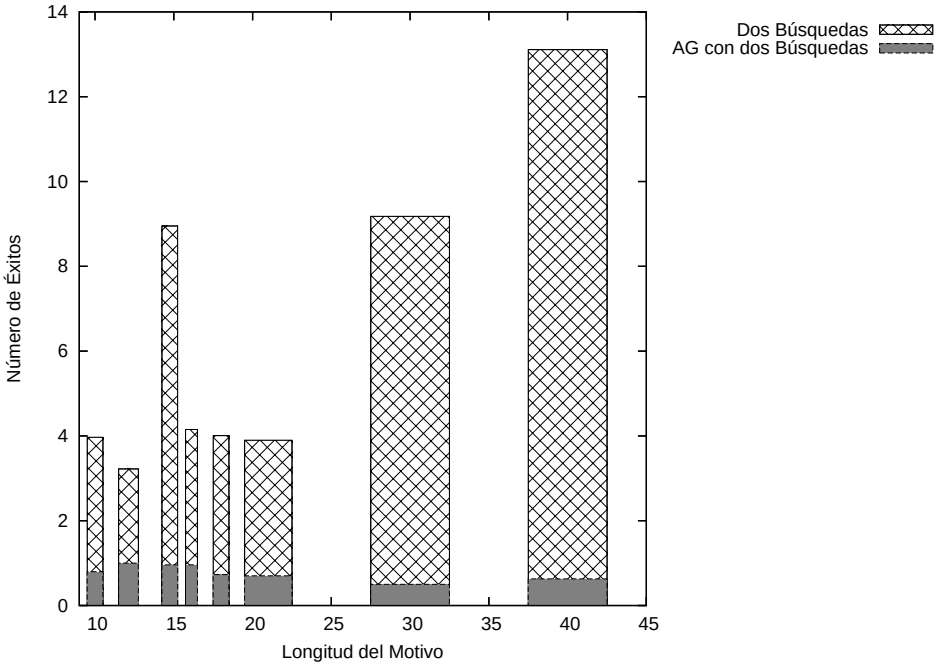

Figura 11. Valor esperado de éxitos de dos búsquedas combinadas empatando en tiempo con una corrida del MbGA con búsqueda local bajo el vecindario $D_{=1}$ y búsqueda lateral

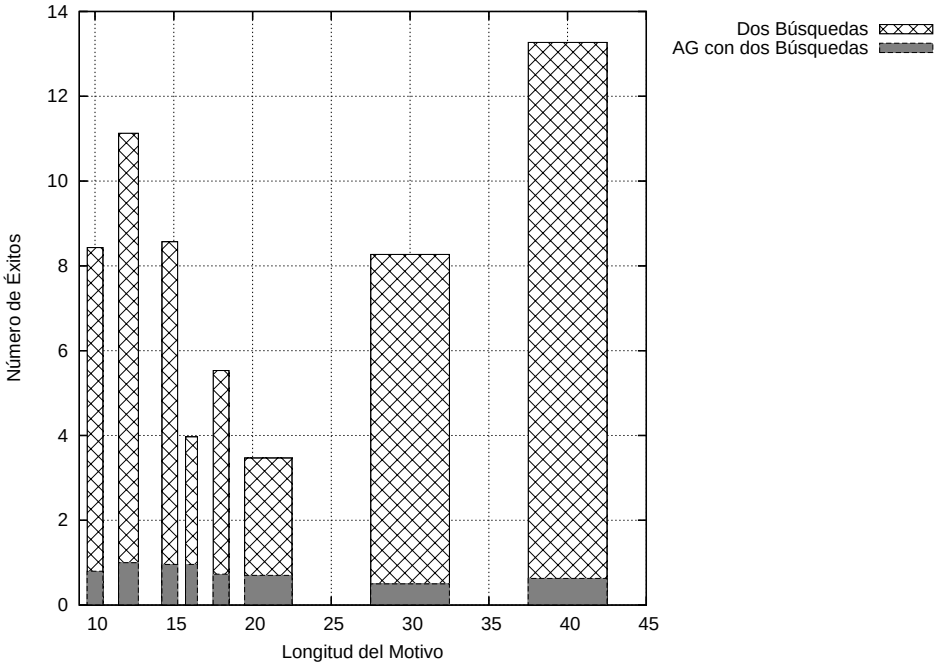

Figura 12. Número de éxitos promedio de 30 corridas de dos búsquedas combinadas empatando en tiempo con una corrida del MbGA con búsqueda local bajo el vecindario $D_{=1}$ y búsqueda lateral

en el tiempo que se ejecuta una vez el genético, podemos correr varias veces la combinación de búsquedas y tener más de un éxito, mientras que el genético no nos asegura, mas que en un caso, tener al menos un éxito. Esto tambíen se llevó a cabo experimentalmente (Figura 12), y los resultados confirman los valores estimados en la Figura 11. Por otro lado, el promedio de las soluciones obtenidas por las dos búsquedas combinadas en cada caso están muy alejadas del óptimo, mientras que los promedios de la población del algoritmo genético son muy cercanas al óptimo.

Algo que se puede conlcuir de la mayoría de las gráficas, es que para todos los algoritmos, el caso $(12, 3)$ es el más sencillo pues todos aumentan su desempeño en ese caso en especial. Un comportamiento muy peculiar de la Figura 7 es que la mayoría de los algoritmos tienden a disminuir el número de éxitos conforme crece la longitud del motivo a buscar, mientras que la combinación de dos búsquedas logra una mayor cantidad de éxitos en los casos más

grandes.

IV.5 Experimentos realizados con Pattern Braching, Random Projections, Gibbs Sampler y MEME

Después de experimentar con el MbGA en sus distintas versiones, pudimos ver que la implementación de las estrategias de búsqueda local dentro de éste habían tenido bastante éxito, mejorando considerablemente el desempeño del MbGA. Una vez que se logró tener tal nivel de desempeño en nuestro algoritmo, se decidió probar los algoritmos más utilizados (Gibbs Sampler y MEME) y más robustos (Pattern Branching, Random Projections) que existen actualmente[2].

Para esto, se corrieron los cuatro algoritmos sobre los ocho casos utilizados anteriormente. A excepción de Pattern Branching que no nos permitió correr en los casos de longitud $l = 40$, todos los demás resultados se muestran a continuación en la Tabla XVIII. Los parámetros que se les dieron a los algoritmos fueron la longitud del motivo, y para los casos de Gibbs y MEME, la cantidad de ocurrencias implantadas. A Random Projections se le dió también de parámetro el número de mutaciones máximas que puede tener cada ocurrencia con respecto al motivo.

La Tabla XVIII muestra los resultados obtenidos por Pattern Branching, Gibbs Sampler, MEME y Random Projections, corriendo los tres primeros una sóla vez y el último 30 veces. Tanto Pattern Braching como Random Projections encuentran el óptimo en todos los casos. MEME sólo falla en un caso. Gibbs es el único de los cuatro que sólo para los casos $(15, 4)$, $(12, 3)$, $(16, 5)$ y $(40, 15)$ obtiene la solución óptima.

En la Tabla XIX se muestran los tiempos de ejecución de Pattern Branching y Random

[2]En la parte final de este trabajo se puede constatar la existencia de otros métodos, que no fueron estudiados aqui ver (Tompa *et al.*, 2005)

Tabla XVIII. Funciones objetivo obtenidas en una corrida de Gibbs Sampler, Pattern Branching y MEME y funciones objetivo promedio obtenidas en 30 corridas de Random Projections

$caso(l,d)$	Pattern Braching	Random Projections	Gibbs Sampler	MEME
(10,2)	34	34	48	34
(12,3)	30	30	30	30
(15,4)	50	50	50	50
(16,5)	66	66	66	66
(18,6)	81	81	151	81
(20,7)	100	100	167	106
(30,11)	146	146	286	146
(40,15)	187	187	187	187

Tabla XIX. Tiempos en segundos obtenidos en una corrida de Pattern Branching y 30 corridas de Random Projections. Entre paréntesis se muestra la desviación estándar del tiempo de ejecución en las 30 corridas de Random Projections

$caso(l,d)$	Pattern Braching	Random Projections
(10,2)	15	48.67(1.61)
(12,3)	14	269.75(7.31)
(15,4)	16	56.19(1.22)
(16,5)	12	684.75(17.21)
(18,6)	18	936.89(11.4)
(20,7)	16	1529.97(13.74)
(30,11)	27	698.68(11.4)
(40,15)	-	1036.64(10.92)

Projections. Ya que Gibbs Sampler y MEME se ejecutaron en forma remota[3,4] y no se obtuvieron los tiempos de ejecución. Como se puede observar en la tabla anterior, los tiempos de Pattern Branching se mantienen constantes salvo por el caso $(30, 11)$ donde se oberva que el tiempo no crece ni siquiera al doble de los casos más pequeños. El caso $(40, 15)$ no es mostrado porque el programa disponible no corre para casos más grandes a 30. En el caso de Random Projections, podemos ver que hay algunos comportamientos anormales en el crecimiento del tiempo de ejecución conforme crece la longitud del motivo. La razón por la que Random Projections tarda más en casos como (12,3) ó (20,7) es porque para esos casos en específico, el algoritmo hace más iteraciones (más proyecciones) para poder asegurar que obtendrá al menos una vez el óptimo. Aún así, para los otros casos, los tiempos son tres veces o más que aquellos obtenidos por Pattern Branching.

IV.6 Experimentos con AG2Búsquedas

Por último, después de ver el desempeño de las dos búsquedas combinadas, se decidió implementar éstas dentro del algoritmo genético MbGA. Para poder hacer una comparación con los algoritmos de las secciones anteriores, se probó este nuevo algoritmo con los casos usados anteriormente utilizando longitudes de motivo desde $l = 10$ hasta $l = 40$. A continuación se presentan los resultados.

Como se puede observar en la Tabla XX, para longitudes pequeñas, las funciones objetivo promedio de los dos algoritmos son similares. A partir del caso $(20, 7)$ podemos ver una pequeña diferencia en los promedios, mostrándose MbGA con Búsqueda a d pasos bajo el vecindario $D_{=1}$ y Búsqueda Lateral ligeramente superior. Sin embargo, el caso $(40, 15)$ es la excepción. AG2Búsquedas obtiene un mayor número de éxitos por lo que su promedio es menor.

En la Tabla XXI se muestran los tiempos promedio que tardan los dos algoritmos.

[3]http://bayesweb.wadsworth.org/gibbs/gibbs.html
[4]http://meme.sdsc.edu/meme/intro.html

Tabla XX. Comparación de la función objetivo promedio ($\overline{F}$) ± desviación estándar correspondiente al algoritmo genético con búsqueda a d pasos bajo el vecindario $D_{=1}$ y búsqueda lateral y el algoritmo AG2Búquedas que combina las dos búsquedas locales. La función objetivo entre paréntesis es la mejor aptitud encontrada por el algoritmo en 30 corridas, y aquellas en negritas corresponden a la mejor solución encontrada que equivale al óptimo.

$caso(l,d)$	$\overline{F}$MbGALS	$\overline{F}$AG2Búsquedas
(10,2)	35.57(**34**) ±3.18	34.73(**34**) ±2.79
(12,3)	30(**30**) ±0	30(**30**) ±0
(15,4)	50.27(**50**) ±1.46	50(**50**) ±0
(16,5)	67.2(**66**) ±6.57	68.4(**66**) ±9.13
(18,6)	88.17(**81**) ±14.54	88.17(**81**) ±16.33
(20,7)	111.07(**100**) ±18.48	116.17(**100**) ±21.67
(30,11)	160.7(**146**) ±26.78	174.27(**146**) ±47.7
(40,15)	193.17(**187**) ±10.28	192.9(**187**) ±32.32

Tabla XXI. Comparación del Tiempo promedio en segundos ($\overline{T}$) ± desviación estándar correspondiente al algoritmo genético con búsqueda a d pasos bajo el vecindario $D_{=1}$ y búsqueda lateral y el algoritmo AG2Búquedas que combina las dos búsquedas locales.

$caso(l,d)$	$\overline{T}$MbGALS	$\overline{T}$AG2Búsquedas
(10,2)	33.6 ±7.71	34.13 ±4.58
(12,3)	42.07 ±6.36	39.8 ±2.47
(15,4)	69.7 ±12.36	58.27 ±2.48
(16,5)	92.87 ±17	82.33 ±15.69
(18,6)	123.7 ±25.91	105.77 ±16.82
(20,7)	159.6 ±33.45	148.07 ±25.91
(30,11)	448.53 ±98.28	373.2 ±70.3
(40,15)	1068.2 ±259.26	741.73 ±36.89

Para los casos más pequeños como $(10, 2)$, $(12, 3)$ y $(15, 4)$, la diferencia del tiempo no es significativa, sin embargo, conforme aumenta la longitud del motivo, se ve un decremento en el tiempo en el segundo algoritmo (AG2Búsquedas).

El análisis estadístico que se llevó a cabo entre estos dos algoritmos (ver Apéndice A) mostró que no hay una diferencia significativa entre ambos algoritmos salvo por el caso más grande, sin embargo, podemos ver que AG2Búsquedas es más rápido que MbGALS en casi todos los casos.

Después de ver los resultados de estos experimentos se decidió probar el algoritmo AG2Búsquedas con tres tipos de casos. El primer tipo (como en los experimentos anteriores) consiste en 20 secuencias de 600 nucleótidos, conteniendo cada secuencia una ocurrencia de un motivo-(l, d). En el segundo tipo, una de las 20 secuencias de entrada no tiene insertada ocurrencia alguna, y en el tercer tipo de caso, una de las secuencias de entrada tiene implantada dos ocurrencias del mismo motivo. Para cada longitud del motivo, se hicieron varias pruebas con distintos motivos.

La Tabla XXII muestra los resultados del algoritmo genético que utiliza las dos estrategias de búsqueda combinadas en casos donde todas las secuencias tienen implantada una ocurrencia del mismo motivo. Se puede observar en la tabla que para los casos de longitud 16 y 40, el algoritmo fué menos exitoso, ya que los promedios están más alejados de la solución óptima. Uno de los casos que se puede considerar difícil es el de longitud 18 pues la cantidad de bases que pueden estar mutadas equivale al treinta por ciento de todo el motivo; los resultados obtenidos para este caso fueron equivalentes a los obtenidos en casos más sencillos como el de longitud 12 ó 15.

Los resultados mostrados en la Tabla XXIII corresponden a casos en los que una de las 20 secuencias de entrada no contiene ocurrencia del motivo. Los tiempos de ejecución del algoritmo genético se mantienen equivalentes a aquellos de la Tabla XXII. En cuanto a las funciones objetivo promedio, podemos ver un comportamiento diferente del algoritmo genético, comparando con la Tabla XXII. Los promedios aumentan para los casos pequeños,

Tabla XXII. Función objetivo promedio y Tiempo promedio de 30 corridas de AG2Búsquedas en casos donde cada secuencia de entrada tiene una ocurrencia de un motivo(l, d). El algoritmo se corrió con 300 individuos, $P_m = 0.2$ y torneo binario. La solución entre paréntesis es la mejor aptitud obtenida y el óptimo global del problema.

Caso(l,d)	F.O. Prom.	T. Prom.	**Caso(l,d)**	F.O. Prom.	T. Prom.
10-2-1	30.87(**30**)	29.27	18-6-1	82.43(**81**)	105.37
10-2-2	29(**29**)	26	18-6-2	70(**70**)	95.23
10-2-3	26(**26**)	24.7	18-6-3	82(**82**)	91.97
10-2-4	32.7(**32**)	27.2	18-6-4	83(**83**)	96.37
10-2-5	26(**26**)	25.5	18-6-5	84.33(**83**)	105.27
12-3-1	41(**41**)	34.9	20-7-1	94(**94**)	121.9
12-3-2	41(**41**)	34.57	20-7-2	87(**87**)	130.4
12-3-3	41(**41**)	33.9	20-7-3	92(**92**)	126.13
12-3-4	47.5(**47**)	39.57	20-7-4	86(**86**)	126.13
12-3-5	51.97(**44**)	44.8	20-7-5	91.27(**86**)	128
15-4-1	48(**48**)	53.43	30-11-1	131(**131**)	336.13
15-4-2	55(**55**)	54.2	30-11-2	139(**139**)	350.47
15-4-3	53(**53**)	54.4	30-11-3	137(**137**)	338.77
15-4-4	58.43(**56**)	60.1	30-11-4	139.1(**135**)	353.5
15-4-5	55.2(**54**)	59.6	30-11-5	140(**140**)	344.57
16-5-1	62.37(**61**)	76.77	40-15-1	201.57(**195**)	780.4
16-5-2	65.5(**63**)	70.3	40-15-2	200.7(**189**)	787.27
16-5-3	76.93(**74**)	87	40-15-3	191.73(**181**)	746.73
16-5-4	66.57(**64**)	71.37	40-15-4	202.67(**190**)	803.93
16-5-5	59(**59**)	73.1	40-15-5	191(**191**)	735.7

Tabla XXIII. Función objetivo promedio y Tiempo promedio de AG2Búsquedas en casos donde una de las secuencias de entrada no tiene implantada ocurrencia alguna. Los parámetros de entrada fueron 300 individuos, $P_m = 0.2$ y torneo binario.

Caso(l,d)	F.O. AG2Búsq.	Tmpo. AG2Búsq.
10-2-1	35(**31**)	33.1
10-2-2	27(**27**)	23.93
10-2-3	32.3(**31**)	27.13
12-3-1	47.33(**43**)	42.3
12-3-2	41(**41**)	33.67
12-3-3	52.73(**49**)	38.97
15-4-1	58(**58**)	56.57
15-4-2	54(**54**)	52.27
15-4-3	63.2(**59**)	60.13
16-5-1	80.53(**74**)	80.67
16-5-2	74.03(**73**)	81.6
16-5-3	69.07(**68**)	75.33
18-6-1	83.7(**81**)	98.27
18-6-2	82(**82**)	100.73
18-6-3	95.83(**85**)	112.33
20-7-1	91.7(**90**)	129.7
20-7-2	81(**81**)	122.17
20-7-3	97.6(**96**)	136.17
30-11-1	141(**141**)	335.67
30-11-2	159.4(**156**)	377.9
30-11-3	155(**155**)	365.93
40-15-1	185(**185**)	737.63
40-15-2	199(**199**)	779.57
40-15-3	210(**210**)	781.77

y para los casos más grandes, el algoritmo se va haciendo más exacto en las treinta corridas. Podemos ver que los promedios de los casos de longitud 30 y 40 son casi iguales al óptimo. Una razón posible para este comportamiento del algoritmo es que para casos más pequeños, hay mayor probabilidad de que el algoritmo encuentre otras subcadenas diferentes al motivo buscado que sean más parecidas entre sí y obtengan un mejor puntaje considerando las 20 secuencias. Hay que recordar que si hay una ausencia del motivo en una de las secuencias, esto afectará en la aptitud de la solución puesto que la ocurrencia identificada en la secuencia donde no hay motivo implantado (para calcular la aptitud de la solución) puede ser muy diferente al resto de las demás, haciendo que disminuya la calidad de la solución. Esto es menos probable en los casos más grandes, por lo que, aunque la aptitud de la solución se vea afectada por la ausencia de una ocurrencia, esta diferencia en la aptitud no será suficiente para que se identifiquen otras soluciones que sean mejores.

En la Tabla XXIV se muestran los resultados obtenidos por el algoritmo genético en casos donde una de las secuencias de entrada tiene dos ocurrencias del mismo motivo. Podemos ver que, en cuanto a tiempos de ejecución, la doble ocurrencia no afecta al algoritmo genético, esto debido a que el número de operaciones sigue siendo el mismo. Para calcular la aptitud de la solución, el algoritmo tiene que revisar todas las subcadenas de longitud l de todo el ADN. Sin importar que haya una o más ocurrencias en la secuencia, éste se va a quedar con aquella subcadena que esté más cercana al motivo candidato. En cuanto a las funciones objetivo promedio, el algoritmo genético tiene mejores resultados para los casos de motivos más grandes (longitud 20, 30 y 40) que para los pequeños. Para éstos, los promedios están más alejados del óptimo, comparando con los resultados en la Tabla XXII. El algoritmo Pattern Branching se corrió en los tres tipos de casos obteniendo siempre el óptimo para cada caso a un tiempo considerablemente pequeño.

En vista de que AG2Búsquedas pudo mejorar el desempeño del algoritmo genético estándar (MbGA), se decidió probar el primero en el caso real CRP (Stormo y Hartzell III, 1989). Los resultados se muestran en la Tabla XXV.

Tabla XXIV. Función objetivo promedio y Tiempo promedio de AG2Búsquedas con casos donde una de las secuencias de entrada tiene dos ocurrencias del motivo.

Caso(l,d)	F.O. AG2Búsq.	Tmpo. AG2Búsq.
10-2-1	27(**27**)	23.2
10-2-2	30.93(**27**)	25.3
10-2-3	30.5(**29**)	26
12-3-1	44.6(**44**)	34.8
12-3-2	47.1(**41**)	42.2
12-3-3	47.53(**47**)	42.83
15-4-1	51(**51**)	52.97
15-4-2	65.77(**63**)	62.9
15-4-3	55(**55**)	56.27
16-5-1	66(**66**)	70.27
16-5-2	67.2(**66**)	75
16-5-3	65.23(**59**)	74.17
18-6-1	92.1(**87**)	113.73
18-6-2	78(**78**)	97.2
18-6-3	85.23(**84**)	112.23
20-7-1	85(**85**)	134.67
20-7-2	97.2(**94**)	131.47
20-7-3	84(**84**)	126.9
30-11-1	142(**142**)	344.17
30-11-2	142(**142**)	345.87
30-11-3	140(**140**)	346
40-15-1	196(**196**)	778.3
40-15-2	185.93(**180**)	768.77
40-15-3	197(**197**)	783.67

Tabla XXV. Comparación del desempeño obtenido por AG2Búsquedas y DosBúsquedas (búsquedas locales combinadas) en un caso real. El sitio de acoplamiento CRP (Stormo y Hartzell III, 1989)

Sec.	Pos. Real	AG2Búsq. Mejor Sol.	Desv.	DosBúsq. Mejor Sol.	Desv.
1	61,17	61	0	61	0
2	55,17	55	0	55	0
3	76	76	0	76	0
4	63	63	0	63	0
5	50	50	0	50	0
6	7,60	7	0	7	0
7	42	24	-18	24	-18
8	39	66	27	66	27
9	9,80	9	0	9	0
10	14	14	0	14	0
11	61	29	-32	29	-32
12	41	41	0	41	0
13	48	48	0	48	0
14	71	71	0	71	0
15	17	17	0	17	0
16	53	53	0	53	0
17	1,84	75	-9	75	-9
18	78	8	-70	8	-50
Puntaje	240	246		246	

Comparando con los resultados mostrados en la Tabla V, podemos ver que la Tabla XXV muestra una mejora en la calidad de la solución ya que el número de posiciones asertadas aumenta a 13. Tanto AG2Búsquedas como las dos búsquedas combinadas obtienen un motivo de mejor aptitud (246) que el motivo solución al problema (240). El motivo encontrado por los dos algoritmos es muy cercano al consenso del problema, sin embargo, ya que siempre se buscan las subcadenas más cercanas (una por secuencia), en algunas secuencias de entrada se identifica otra ocurrencia que no es la real, es decir, que el algoritmo encuentra subcadenas de fondo mejores que las reales, por lo tanto algunas posiciones de inicio no coinciden con las reales.

IV.7 Discusión

De las dos codificaciones más obvias para representar las soluciones en el algoritmo genético diseñado para atacar el problema de la búsqueda de motivos, se encontró que la basada en motivo daba mejores resultados. Esto lo atribuímos a que el espacio de búsqueda es más pequeño a aquél de la codificación basada en posiciones de inicio, por lo que una mutación o cruzamiento en un individuo basado en motivo tiene mayores posibilidades de mejorar su aptitud que una mutación o cruzamiento en un individuo basado en posiciones de inicio.

La implementación de estrategias de búsqueda local dentro del algoritmo genético estándar ayudaron a mejorar la calidad de solución del algoritmo estándar y a disminuir sus tiempos de ejecución. Una de las razones por las cuales creemos que las estrategias mejoraron el desempeño es que le permiten al algoritmo explorar un mayor número de soluciones, a las cuales por sí solo no llegaría aunque se incrementara el número de iteraciones del algoritmo. La búsqueda a d pasos le permitió al algoritmo llegar a soluciones más rápido que como lo haría aplicando sólo mutación, ya que ésta no se aplica estrictamente a cada iteración a todos los individuos. La búsqueda lateral fué otro punto importante en la mejora del desempeño ya que muchas de las soluciones arrojadas por el genético contenían

sólo parte del motivo a buscar. La búsqueda lateral ayudó a llevar a este tipo de soluciones al óptimo global.

Como se pudo observar en los resultados, aplicar la búsqueda a d pasos disminuyó los tiempos de MbGA, sin embargo, al implementar junto con la búsqueda a d pasos la búsqueda lateral, hubo un incremento en el tiempo. Esto era predecible ya que se le estaba añadiendo al algoritmo un postprocesamiento. Sin embargo, a pesar de este pequeño incremento en el tiempo, debido a que la calidad de la solución del algoritmo había mejorado, pudimos disminuir el tamaño de la población inicial, lo que provocó que los tiempos de ejecución disminuyeran en gran medida comparando con MbGA y MbGA con sólo la búsqueda a d pasos, logrando un equilibrio en cuanto al nivel de calidad de solución y los tiempos de ejecución.

De acuerdo a los resultados experimentales obtenidos, podemos decir que, en cuanto a desempeño, no importa la manera en que implementemos las dos estrategias de búsqueda dentro del genético (primero una y al finalizar la segunda o las dos combinadas), la calidad de los resultados permanecerá la misma. Esto nos lleva a pensar a que ambos algoritmos se comportan igual a pesar de la manera en que las dos búsquedas se utilizan. En el primer algoritmo (MbGALS) no se aplican iterativamente las dos estrategias de búsqueda, si no que a cada iteración, se le está aplicando la búsqueda al mejor individuo, sin embargo, aunque le toma más tiempo (más iteraciones), como lo reflejan los tiempos de ejecución, se llega finalmente a soluciones de la misma calidad que aquellas de AG2Búsquedas. De una manera u otra, la búsqueda local bajo el vecindario $D_{=1}$ nos está asegurando llegar a soluciones de calidad óptima aunque en el primer algoritmo (MbGALS) requiera de un mayor número de iteraciones que el segundo (AG2Búsquedas).

En la parte final de este trabajo se pudo constatar la existencia de otros métodos, que no fueron estudiados aqui (Tompa *et al.*, 2005).

Capítulo V

Conclusiones y Trabajo a Futuro

V.1 Sumario

En este trabajo se abordó el problema de búsqueda de motivos utilizando algoritmos genéticos, así como también, algunas estrategias de búsqueda local. Como un primer intento de abordar el problema, se diseñaron dos algoritmos genéticos estándar utilizando dos codificaciones distintas de la solución, una basada en las posiciones de inicio y otra en un motivo consenso. Basándonos en el modelo de motivo implantando, se realizaron una serie de experimentos con datos ficticios los cuales reflejaron que la codificación de motivo consenso (MbGA) lograba soluciones de mejor calidad que aquellas de la codificación basada en posiciones de inicio (PbGA). Con el objetivo de ver la aplicabilidad de estos algoritmos en casos reales, se realizó un experimento con el sitio de pegado CRP (Stormo y Hartzell III, 1989) para el cual se obtuvieron resultados similares con los dos algoritmos.

Se implementaron dos estrategias de búsqueda local con el fin de mejorar tanto desempeño como tiempos de ejecución del algoritmo genético estándar basado en motivo. La primera estrategia implementada fué la búsqueda a d pasos bajo el vecindario $D_{=1}$, basándonos en las ideas de Pattern Branching (Price *et al.*, 2003). Posteriormente se implementó la búsqueda lateral. La implementación de esta estrategia mejoró considerablemente el desempeño del algoritmo, pues se obtuvieron un gran número de éxitos en los experimentos. Estos resultados nos llevaron a la idea de que había la posibilidad de que la búsqueda local, sola, estuviera haciendo todo el trabajo, es decir, que fuera la clave del éxito del genético. Es por esta razón que se decidió experimentar con las estrategias de búsqueda local fuera del algoritmo genético. Éstas, por sí solas, no lograron obtener un

mejor desempeño que aquél del genético, sin embargo, la combinación iterativa de ellas obtuvo resultados tan buenos que se pudo observar que en una sola ejecución del genético, la búsqueda combinada podía correr varias veces y lograr más de un éxito, mientras que el algoritmo genético no nos podía asegurar en algunos casos ni siquiera uno.

Finalmente, se implementaron las dos búsquedas combinadas dentro del algoritmo genético (AG2Búsquedas).

V.2 Conclusiones

- En cuanto a la representación de la solución en el algoritmo genético estándar, una codificación basada en motivo obtiene mejores resultados que una codificación basada en posiciones de inicio en términos de la calidad de solución a igual tiempo de cómputo.

- Las estrategias de búsqueda local implementadas dentro del algoritmo genético estándar MbGA permitieron mejorar la calidad de solución y disminuir los tiempos de ejecución.

- Para los casos generados con el modelo del motivo implantado con secuencias de fondo uniformemente distribuídas, una búsqueda local simple puede superar en número de éxitos al algoritmo genético estándar usando los mismos recursos computacionales.

- A pesar de que una sola corrida de la búsqueda local tiene muy poca probabilidad de encontrar el óptimo del problema, ésta corre en un tiempo muy pequeño, lo cual nos permite correrla varias veces en el tiempo de una corrida del algoritmo genético. Con esto podemos decir que si el tiempo es crucial para la búsqueda de un motivo, podríamos optar por aplicar un algoritmo de búsqueda local que nos puede asegurar varios éxitos en poco tiempo. Por otra parte, si lo que se busca es precisión, podríamos optar por correr varias veces el algoritmo genético que demuestra tener un buen desempeño en todos los casos (distinta longitud del motivo).

- El algoritmo Pattern Branching tiene un buen desempeño en casos basados en el modelo de motivo implantando, sin embargo, la implementación que se tiene de éste no funciona para casos de longitud mayor o igual a 40 ni para el caso real CRP.

- Los resultados experimentales y el análisis estadístico nos indican que no hay una diferencia significativa entre utilizar las dos estrategias de búsqueda local combinadas dentro del algoritmo genético (AG2Búsquedas) y utilizar las dos por separado dentro del genético (MbGALS).

V.3 Perspectivas de Investigación

Como trabajo a futuro se propone:

- Probar los algoritmos con un mayor número de casos reales para poder determinar si en efecto la búsqueda local o el genético con búsqueda local pueden ser utilizados con confianza en la práctica.

- Adaptar los algoritmos para que puedan trabajar con casos en los que se busquen ocurrencias de distinto tamaño.

- Probar los algoritmos (tanto de búsqueda local como genéticos) en casos donde la distribución de las secuencias de fondo no sea uniforme y en casos construídos con el modelo FM, donde los motivos implantados contengan exactamente d mutaciones (caso difícil para la búsqueda local planteada en (Pevzner y Sze, 2000)).

- Buscar casos patológicos para la búsqueda local y Pattern Branching ya que la búsqueda de motivos es un problema NP-difícil.

- Obtener una implementación de Pattern Branching que pueda funcionar en casos de longitudes mayores o iguales a 40 y que pueda ser probada en casos reales.

- Probar la búsqueda local en el vecindario $D_{=1}$ en el algoritmo genético estándar MbGA.

- Probar la búsqueda local en el vecindario $D_{=1}$ con sólo d pasos y probar la combinación de la búsqueda en vecindario $D_{=1}$ con d pasos y la búsqueda lateral.

- Buscar una función objetivo que modele mejor al problema del motivo real.

V.4 Productos de Investigación

- Herramienta para el estudio de descubrimiento de motivos en secuencias de ADN en casos ficticios y reales.

- Generador de casos basado en el modelo del motivo implantado.

- Artículo publicado en Proceedings del Simposio Brasileño en Bioinformática BSB 2007.

  Martínez-Arellano, G. and Brizuela, Carlos A. Comparison of Simple Encoding Schemes in GA's for the Motif Finding Problem: Preliminary Results. M.-F. Sagot and M.E.M.T. Walter, editores, BSB 2007, LNBI 4643, pp. 22-33, 2007.

- Artículo aceptado en el Congreso Mexicano de Cómputo Evolutivo COMCEV 2007.

  Martínez-Arellano, G. and Brizuela, Carlos A. An Enhanced Genetic Algorithm for the Motif Finding Problem. COMCEV 2007, pp.1-6, 2007.

Bibliografía

Bailey, T. y C. Elkan 1995. "Unsupervised learning of multiple motifs in biopolymers using expectation maximization.". Machine Learning, 21:51-80 p.

Blanchette, M., B. Schwikowski, y M. Tompa 2002. "Algorithms for philogenetic footprinting". J. Comp. Biol., 9:211-223 p.

Brazma, A., I. Jonassen, J. Vilo, y E. Ukkonen 1998. "Predicting gene regulatory elements *in silico* on a genomic scale". Genome Res., 15:1202-1215 p.

Buhler, J. y M. Tompa 2002. "Finding motifs using random projections". Journal of Computational Biology, 9(2):225-242 p.

Carlson, J. 2004. "Critical comparison of motif models".

Che, D., Y. Song, y K. Rasheed 2005. "Mdga: motif discovery using a genetic algorithm". En: Beyer, H.-G., U.-M. O'Reilly, D. V. Arnold, W. Banzhaf, C. Blum, E. W. Bonabeau, E. Cantu-Paz, D. Dasgupta, K. Deb, J. A. Foster, E. D. de Jong, H. Lipson, X. Llora, S. Mancoridis, M. Pelikan, G. R. Raidl, T. Soule, A. M. Tyrrell, J.-P. Watson, y E. Zitzler, editores, "GECCO 2005: Proceedings of the 2005 conference on Genetic and evolutionary computation", Washington DC, USA, volume 1. ACM Press. 447–452 p.

Davila, J., S. Balla, y S. Rajasekaran 2006. "Space and time efficient algorithms for planted motif search". En: Alexandrov, V. N., G. D. van Albada, P. M. A. Sloot, y J. Dongarra, editores, "International Conference on Computational Science (2)", volume 3992 of Lecture Notes in Computer Science. Springer. 822-829 p.

Eiben, A. E. y J. Smith 2003. "Introduction to evolutionary computing". Springer.

Eskin, E. y P. A. Pevzner 2002. "Finding composite regulatory patterns in dna sequences". En: "ISMB". 354-363 p.

Hansen, P., N. Mladenovic, y D. Urosevic 2006. "Variable neighborhood search and local branching". Computers and Operations Research, 33:3034-3045 p.

Hertz, G. y G. Stormo 1999. "Identifying dna and protein patterns with statistically significant alignments of multiple sequences". Bioinformatics, 15:563-677 p.

Karaoglu, N., S. Maurer-Stroh, y B. Manderick 2006. "Gamot: An efficient genetic algorithm for finding challenging motifs in dna sequences". En: "Proceedings of 3rd Annual Satellite Workshop on Regulatory Genomics RECOMB-RG06 (Singapore, Singapore, Julio 2006)". 4-12 p.

Keich, U. y P. Pevzner "2002". "Finding motifs in the twilight zone". Bioinformatics, 18:1374-1381 p.

Krasnogor, N. y J. Smith 2005. "A tutorial for competent memetic algorithms: model, taxonomy and design issues". IEEE Transactions on Evolutionary Computation, 9(5):474-488 p.

Lawrence, C., S. Altschul, M. Bogusky, J. Liu, A. Neuwald, y J. Wootton 1993. "Detecting subtle sequence signals: a gibbs sampling strategy for multiple alignment". Science.

Pevzner, P. A. y S.-H. Sze 2000. "Combinatorial approaches to finding subtle signals in dna sequences". En: Bourne, P. E., M. Gribskov, R. B. Altman, N. Jensen, D. A. Hope, T. Lengauer, J. C. Mitchell, E. D. Scheeff, C. Smith, S. Strande, y H. Weissig, editores, "ISMB". AAAI. 269-278 p.

Price, A., S. Ramabhadram, y P. Pevzner 2003. "Finding subtle motifs by branching from sample strings". Bioinformatics, 1(1):1-7 p.

Sagot, M.-F. 1998. "Spelling approximate repeated or common motifs using a suffix tree". En: Lucchesi, C. L. y A. V. Moura, editores, "LATIN", volume 1380 of Lecture Notes in Computer Science. Springer. 374-390 p.

Sinha, S. y M. Tompa 2000. "A statistical method for finding transcription factor binding sites". En: "Proc. 8th Int. Conf. Intelligent Systems for Molecular Biology". 344-354 p.

Stavrovskaya, E. D. y A. A. Mironov 2003. "Two genetic algorithms for identification of regulatory signals". In Silico Biology, 3:5 pp.

Stormo, G. y G. Hartzell III 1989. "Identifying protein-binding sites from unaligned dna fragments". PNAS, 86:1183-1187 p.

Strachan, T. y P. R. Andrew 1999. "Human molecular genetics 2". BIOS Scientific Publishers Ltd, Oxford, UK, 600 páginas.

Tompa, M., N. Li, T. L. Bailey, G. M. Church, B. De Moor, E. Eskin, A. V. Favorov, M. C. Frith, Y. Fu, W. J. Kent, V. J. Makeev, A. A. Mironov, W. Stafford Noble, G. Pavesi, G. Pesole, M. Régnier, N. Simonis, S. Sinha, G. Thijs, J. van Helden, M. Vandenbogaert, Z. Weng, C. Workman, C. Ye, y Z. Zhu 2005. "Assessing computational tools for the discovery of transcription factor binding sites". Nature Biotechnology, 23:137-144 p.

Wu, X., B. Wang, C. Song, y J. Cheng 2004. "A combined model and a varied gibbs sampling algorithm used for motif discovery". En: Chen, Y.-P. P., editor, "APBC", volume 29 of CRPIT. Australian Computer Society. 99-104 p.

Zaslavsky, E. y M. Singh 2006. "A combinatorial optimization approach for diverse motif finding applications". Algorithms for Molecular Biology, 1:13+ p.

Apéndice A

Tests de Normalidad y Mann Whitney

Una vez que se obtuvieron todos los resultados de los algoritmos descritos en este capítulo, se presenta ahora un análisis estadístico que nos ayudará a determinar si hay una diferencia significativa entre los resultados. Lo primero que se hizo fué ver si los resultados de los experimentos seguían una distribución normal, para esto se aplicó el test de normalidad a los resultados obtenidos de cada algoritmo. Partiendo de la hipótesis nula H_0 de que los resultados de un algoritmo tienen un comportamiento normal, un resultado de 0 confirmará la veracidad de la hipótesis mientras que un resultado de 1 nos indicará que se rechaza la hipótesis nula. El símbolo "-" en la Tabla XXVI es para aquellos algoritmos que no fueron probados con ciertos casos, por lo que no se llevó a cabo la prueba.

Los resultados de la Tabla XXVI muestran que ninguno de los algoritmos produce resultados que sigan un comportamiento normal. Estos resultados son razonables dado que los algoritmos no pueden arrojar valores solución menores (si se está minimizando) o mayores (si se está maximizando) al óptimo del problema.

En vista de los resultados obtenidos anteriormente, se procedió a aplicar la prueba Mann Whitney para determinar si los promedios entre uno y otro algoritmo eran significativamente diferentes. Para esto, se utilizaron los resultados de las 30 corridas de cada algoritmo para cada uno de los casos en los que fueron probados. Valores (obtenidos del test) menores a $p = 0.05$ ó $p = 0.01$ (nivel de confianza) nos indicarán que los resultados entre los dos algoritmos a los que se les aplicó el test son significativamente distintos, probando que un algoritmo es mejor que el otro. A continuación se muestran los resultados después de haber aplicado Mann Whitney a los algoritmos PbGA y MbGA.

Tabla XXVI. Tests de Normalidad para los algoritmos PbGA, MbGA, MbGA con búsqueda a distancia, MbGA con dos búsquedas y MbGA con dos búsquedas combinadas. El valor 1 nos indica que los resultados de los algoritmos no siguen un comportamiento normal y el símbolo "-" es para aquellos algoritmos que no fueron probados con ciertos casos por lo que no se llevó a cabo la prueba

$Caso(l,d)$	PbGA	MbGA	MbGA con $D_{=1}$	MbGA con dos Búsq.	AG2Búsq
$(10,2)$	1	1	1	1	1
$(10,3)$	1	1	-	-	-
$(11,2)$	1	1	-	-	-
$(11,3)$	1	1	-	-	-
$(12,3)$	1	1	1	1	1
$(12,4)$	1	1	-	-	-
$(15,4)$	1	1	1	1	1
$(16,5)$	1	1	-	1	1
$(18,6)$	1	1	-	1	1
$(20,7)$	1	1	-	1	1
$(30,11)$	1	1	-	1	1
$(40,15)$	1	1	-	1	1

Tabla XXVII. Prueba Mann Whitney para comparar los resultados de PbGA y MbGA

$caso(l,d)$	Prueba Mann Whitney
(10,2)	$2.3507e^{-11}$
(10,3)	$8.7872e^{-11}$
(11,2)	$3.5024e^{-9}$
(11,3)	$1.6843e^{-11}$
(12,3)	$5.4123e^{-6}$
(12,4)	$2.3342e^{-11}$
(15,4)	$1.2161e^{-10}$
(16,5)	$7.0952e^{-10}$
(18,6)	$2.9221e^{-8}$
(20,7)	$1.4950e^{-8}$
(30,11)	$2.1902e^{-6}$
(40,15)	0.2519

Tabla XXVIII. Prueba Mann Whitney para comparar los resultados de MbGA y MbGA con Búsqueda a Distancia $D_{=1}$

$caso(l, d)$	Prueba Mann Whitney
(10,2)	0.0011
(12,3)	$4.2123e^{-4}$
(15,4)	0.0114

La Tabla XXVII nos muestra que hay una diferencia significativa entre el promedio de la calidad de los resultados obtenidos por PbGA y aquellos obtenidos por MbGA, ya que todos los valores están por debajo de 0.01 salvo por el caso $(40, 15)$. Por lo que se puede concluir que MbGA es mejor que PbGA en cuanto a calidad de solución. Esto confirma la conclusión inicial que se había hecho respecto a esos resultados, lo que nos lleva ahora a probar si hay una mejora significativa entre MbGA y MbGA con la búsqueda a distancia (algoritmo que se creó con la intención de mejorar al primero tanto en tiempo como en calidad). A continuación se muestran los resultados de esta prueba.

La Tabla XXVIII muestra que MbGA y MbGA con búsqueda a distancia son significativamente distintos para los casos $(10, 2)$ y $(12, 3)$. Para el caso $(15, 4)$ la diferencia es menor aunque todavía se puede considerar que hay una diferencia entre MbGA y MbGA con búsqueda a distancia, siendo éste último mejor en cuanto a la calidad de sus soluciones.

Después de probar la búsqueda a distancia en el MbGA, se decidió probar también la búsqueda lateral, dado que las soluciones obtenidas si no eran las óptimas, se encontraban recorridas a la derecha o a la izquierda un par de caracteres. A continuación se muestran los resultados de la prueba Mann Whitney comparando MbGA con búsqueda a d pasos bajo el vecindario $D_{=1}$ y MbGALS.

Como se puede ver en la Tabla XXIX, los resultados obtenidos por MbGA con dos búsquedas son significativamente diferentes a aquellos obtenidos por MbGA con la búsqueda a distancia, dado que los resultados de la prueba Mann Whitney fueron valores menores a 0.05. Esto significa que MbGA con dos búsquedas logró tener resultados de mejor calidad

Tabla XXIX. Prueba Mann Whitney para comparar los resultados de MbGA con Búsqueda a Distancia $D_{=1}$ y MbGA con dos búsquedas

$caso(l, d)$	Prueba Mann Whitney
(10,2)	$8.2836e^{-4}$
(12,3)	$4.2339e^{-8}$
(15,4)	$2.1932e^{-6}$

Tabla XXX. Prueba Mann Whitney para comparar los resultados de MbGA con dos búsquedas y AG2Búsquedas (MbGA con dos búsquedas combinadas)

$caso(l, d)$	Prueba Mann Whitney
(10,2)	0.0954
(12,3)	1
(15,4)	0.3173
(16,5)	0.5311
(18,6)	0.5247
(20,7)	0.3781
(30,11)	0.4221
(40,15)	0.0022

que aquellos de MbGA con la búsqueda a distancia.

Por último se decidió comparar los resultados obtenidos por MbGA con dos búsquedas con los resultados obtenidos por MbGA con dos búsquedas combinadas (AG2Búsquedas). Sabemos que las dos búsquedas combinadas fuera del algoritmo genético funcionan mejor que cualquier otro algoritmo de búsqueda local probado anteriormente en este trabajo, por lo que esperaríamos que la prueba Mann Whitney nos ayudará a determinar si esto también aplica dentro del algoritmo genético. A continuación se muestran los resultados de la prueba.

En la Tabla XXX se puede ver que no hay una diferencia significativa entre los resultados obtenidos por MbGA con dos búsquedas y MbGA con búsquedas combinadas, salvo por el caso más grande $(40, 15)$. Podemos ver que para el caso $(12, 3)$ el valor de p es 1 lo que indica que para dicho caso, los resultados de los dos algoritmos fueron siempre iguales (iguales

al óptimo en las 30 ejecuciones). En el caso $(40, 15)$ MbGA con búsquedas combinadas encuentra el óptimo más veces, razón por la cual, el resultado de la Tabla XXX muestra una diferencia significativa entre los dos.

En conlusión podemos decir que el algoritmo memético (algoritmo genético con búsqueda local) en cualquiera de sus implementaciones (con búsqueda a distancia, dos búsquedas o búsqueda combinada) mejoró la calidad de solución del algoritmo genético estándar MbGA. De las búsquedas locales, se comprobó experimentalmente que las dos búsquedas combinadas obtenían un mayor número de éxitos en las $1,000$ corridas, por lo que se pensó que si se implementaba ésta en el genético, se obtendrían mejores resultados. Las pruebas Mann Whitney aplicadas en los resultados experimentales mostraron que no hay una diferencia significativa en utilizar, en el genético, una búsqueda a distancia y al finalizar una lateral a utilizar las dos búsquedas combinadas, salvo para casos grandes como $(40, 15)$ donde se logra un mejor desempeño al utilizar las dos búsquedas combinadas en el genético. Sin embargo, en cuanto a tiempos de ejecución, podemos ver que las dos búsquedas combinadas en el genético permiten que el algoritmo converja más rápidamente. Se puede observar también que cualquier uso de la búsqueda local dentro del genético mejora significativamente el desempeño del algoritmo genético estándar MbGA.