TESIS DEFENDIDA POR

Milton Rodríguez Zambrano

Y aprobada por el siguiente comité:

Dr. Carlos Alberto Brizuela Rodríguez

Director del Comité

Dr. Andrei Tchernykh

Miembro del Comité

Dr. Axayacatl Rocha Olivares

Miembro del Comité

Dr. José Alberto Fernández Zepeda

Miembro del Comité

Dr. Pedro Gilberto López Mariscal

Coordinador del programa en Ciencias de la Computación

Dr. Raúl Ramón Castro Escamilla

Director de Estudios de Posgrado

19 de Octubre del 2005.

CENTRO DE INVESTIGACIÓN CIENTÍFICA Y DE EDUCACIÓN
SUPERIOR DE ENSENADA

PROGRAMA DE POSGRADO EN CIENCIAS
EN CIENCIAS DE LA COMPUTACIÓN

Un algoritmo genético para el ensamble de secuencias de ADN

TESIS

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

MAESTRO EN CIENCIAS

Presenta:

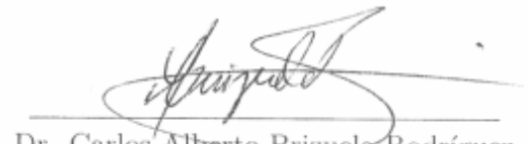
Milton Rodríguez Zambrano

Ensenada, Baja California, México. Octubre del 2005.

RESUMEN de la tesis que presenta **Milton Rodríguez Zambrano**, como requisito parcial para obtener el grado de MAESTRO EN CIENCIAS en CIENCIAS DE LA COMPUTACIÓN. Ensenada, Baja California. Octubre del 2005.

Un algoritmo genético para el ensamble de secuencias de ADN

Resumen aprobado por:



Dr. Carlos Alberto Brizuela Rodríguez

Director de Tesis

La secuenciación del genoma de un organismo permite el estudio y entendimiento de éste, para el caso de parásitos o virus, podría lograrse el desarrollo de nuevos medicamentos y vacunas. La secuenciación permite también el estudio de la evolución de las especies, mediante la identificación de variaciones genéticas.

Las técnicas de secuenciación que se aplican dependen del tamaño de la cadena de ADN que se desea secuenciar. Para el caso de genomas grandes, la técnica más utilizada se denomina Shotgun, la cual comprende varias etapas, la última de ellas se conoce como Ensamble de Secuencias. El problema de ensamble de secuencias de ADN, en su modelo más sencillo, pertenece a la clase NP-Difícil, lo que significa que no se conoce método alguno que proporcione una solución óptima en tiempo polinomial.

Entre las estrategias propuestas para este problema se encuentran principalmente las estrategias voraces y los algoritmos genéticos. En este trabajo se propone un algoritmo genético, para un modelo de este problema que consiste en encontrar una ruta con características específicas en un grafo dirigido completo. Este modelo considera errores de bases y la orientación desconocida de los segmentos. El algoritmo propuesto se basa en un algoritmo genético básico, al cual se le agrega una fase de eliminación de segmentos y ajuste de individuos. Se propone también una función de aptitud que intenta favorecer la generación de soluciones con la menor cantidad posible de contigs, y consecuentemente contigs con la mayor longitud posible.

Se comparan los resultados del algoritmo genético propuesto contra los de un algoritmo genético básico, notándose claramente el mejor desempeño del primero. Finalmente, se comparan los resultados contra los de un algoritmo voraz disponible públicamente, obteniendo resultados comparables e incluso a favor del algoritmo genético para casos con cierta tasa de error de base. Los resultados son independientes del tamaño de los fragmentos a ensamblar y de la función de aptitud utilizada.

Palabras clave: Algoritmos Genéticos, Secuenciación, Ensamble, ADN, Alineamiento.

ABSTRACT of the thesis presented by **Milton Rodríguez Zambrano**, as a partial requirement to obtain the MASTER IN SCIENCES degree in COMPUTER SCIENCE. Ensenada, Baja California. October 2005.

A Genetic Algorithm for DNA Sequence Assembly

Abstract approved by:

Dr. Carlos Alberto Brizuela Rodríguez

Thesis director

The sequencing of an organism genome allows us to study and understand it, knowing the genome of parasites and viruses help us to develop new medicines and vaccines. The sequencing also allows us to study the evolution of the species through the identification of genetic variations.

The selection of a sequencing technique depends on the size of the target DNA chain. For large genomes, the most used technique is known as Shotgun, which includes several stages, and the last one is denominated Assembling. The DNA assembling problem, under its simplest model, belongs to the NP-Hard class of problems, which implies that a method providing the optimal solution in polynomial time is unknown.

Among the proposed strategies to solve this problem the greedy strategies and the genetic algorithms are the most common. In this work we propose a genetic algorithm to solve a model where the objective is to find a path with specific characteristics in a complete digraph. This model considers the base call errors and the segments' unknown orientation. The proposed algorithm is based on a simple genetic algorithm, where we propose a segment elimination and individuals adjustment phase. We also propose a fitness function, which is intended to help on generating solutions with the minimum possible amount of contigs and, consequently, contigs with the maximum possible size.

The results of the proposed genetic algorithm outperform those obtained by a basic genetic algorithm. Finally, the experimental results show that the solution quality obtained by our proposed algorithm is comparable with the one produced by a greedy algorithm and, in some cases our algorithm performs better. These results are independent of the fragments to assemble size and of the fitness function.

Keywords: Genetic Algorithms, Sequencing, Assembly, DNA, Alignment.

A mis padres y hermanos

Agradecimientos

A Dios y a la vida por esta linda experiencia.

A mis padres por el cariño y apoyo incondicional, que junto a mis hermanos representan mi mayor motivación.

Al Dr. Carlos Brizuela por su invaluable atención, apoyo y paciencia en todo momento.

A los miembros de mi comité Dr. Alberto Fernández, Dr. Andrei Tchernykh y Dr. Axayacatl Rocha por sus acertadas observaciones durante el desarrollo de la tesis y en la escritura de la misma.

Al departamento de Ciencias de la Computación, por la oportunidad y por las facilidades que siempre tuve.

A mi compa Luis, Abelardo, J y Tentori por los buenos y también por los malos momentos, indispensables para una buena amistad.

A mis demás compañeros y amigos que hicieron de estos dos años algo más que un periodo de estudio, Albino, David, Leo, Pedro, Chesar, Bob, Liz, Isabel, Diana, Cintya, Abi, Doris y Marcel.

A los hermosillos Memo, Manuel, Ricardo y Luz por esa sincera y especial amistad.

A Caro y Lidia por el buen humor y la amabilidad.

A Los del sur por los gloriosos (3) momentos y a nuestros pacientes seguidores también muchas gracias.

Al Consejo Nacional de Ciencia y Tecnología.

Ensenada, México
19 de Octubre del 2005.

Milton Rodríguez Zambrano

Tabla de Contenido

Lista de Figuras

Lista de Tablas

Lista de Tablas (Continuación)

Capítulo I

Introducción

La información genética en nuestras células está contenida en cromosomas, los cromosomas contienen a los genes y cada uno de estos está compuesto de una secuencia de pares de bases que son parte del Ácido Desoxirribonucleico (ADN). El conjunto completo de estas secuencias, *el genoma*, representa un conjunto de instrucciones que controlan la replicación y función celular y por ende la función de cada organismo. El ADN es una doble hélice conformada por un esqueleto de grupos fosfato y azúcares alternados, estas dos hebras o cadenas se encuentran unidas, como se muestra en la Figura 1, mediante enlaces de hidrógeno entre bases nitrogenadas. Estas bases son: Adenina (A), Timina (T), Citosina (C) y Guanina (G), respectivamente (Watson y Crick, 1953).

Estas bases están enlazadas a lo largo de la cadena siguiendo la complementariedad siguiente: Adenina-Timina y Citosina-Guanina, como lo muestra la Figura 1. Tammi (2003) menciona que el ADN puede ser visto como el lenguaje de la vida; primero, esas cuatro letras agrupadas en palabras, donde cada palabra está formada por una sub-cadena de tres letras llamada codón, cada codón codificará a un amino-ácido, y una sub-cadena de amino-ácidos se plegará para formar una proteína. De esta manera, el lenguaje determina qué proteinas son hechas, cuándo y en qué cantidad. La secuenciación de ADN es el proceso por el cual se establece el orden preciso de bases a lo largo de una cadena de ADN.

Ahora bien, con la tecnología actual pueden secuenciarse (directamente) únicamente

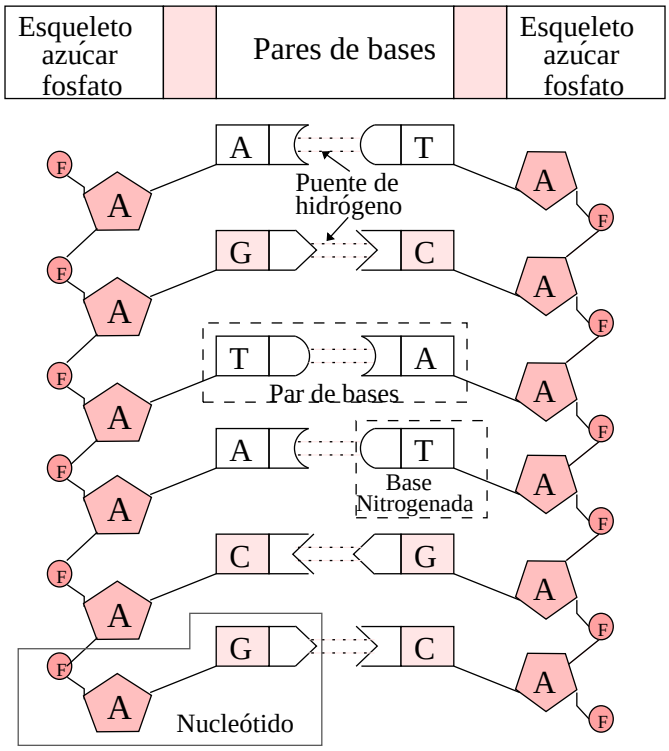


Figura 1: ADN (Ácido Desoxiribonucleico)

cadenas de ADN de 300 y hasta 1000 pares de bases (Tammi, 2003), (Pop *et al.*, 2002). Pero generalmente los genomas son mucho más grandes, como el del ser humano, que sin ser el más grande es del orden de 10^9 pares de bases; la limitante tecnológica ha llevado al desarrollo de técnicas para secuenciar genomas de esta magnitud, siendo la técnica *Shotgun* la más utilizada para este propósito.

I.1 Antecedentes y motivación

El 14 de abril del 2003 fue anunciada la culminación del Proyecto del Genoma Humano, conducido en Estados Unidos por el Instituto Nacional de Investigación del Genoma Humano (NHGRI por sus siglas en inglés) y el Departamento de Energía. Según datos

del NHGRI[1], la secuencia terminada consta de 3000 millones de bases, y cubre alrededor del 99% de las regiones del genoma humano que contienen genes, y este resultado tiene una exactitud del 99.99%. Venter *et al.* (2001) dan a conocer que la secuenciación de este genoma se llevó a cabo mediante el método "whole genome shotgun", complementado con una etapa de ensamble de segmentos secuenciados.

Un dato importante que Venter *et al.* (2001) mencionan es que los seres humanos compartimos el 99.9% de esta secuencia, el 0.1% restante varía entre cada individuo, siendo las variaciones más comunes aquellas en que cambia una sola letra, conocidas como SNPs[2] (por sus siglas en inglés). Según datos del Instituto Nacional de Medicina Genómica (INMEGEN)[3] se han identificado cerca de 3.2 millones de SNPs, y son precisamente estas variaciones las que dan la individualidad genómica y confieren susceptibilidad o resistencia a enfermedades como cáncer, enfermedades cardiacas, diabetes o Alzheimer, también confieren variabilidad en la respuesta a medicamentos de uso común.

A pesar de los más de 3 millones de SNPs, Venter *et al.* (2001) mencionan que menos del 1% de estos SNPs se encuentran en genes, y son éstos los que más interesan a los investigadores porque son los que tienen más probabilidad de afectar a las proteínas.

Collins *et al.* (2003) plantean la visión de la investigación genómica, que puede dirigirse a tres frentes: a la biología, a la salud y a la sociedad.

---

[1]http://www.genome.gov/11006929, agosto-2005

[2]Single Nucleotide Polymorphism

[3]http://www.inmegen.gob.mx, agosto-2005

En cuanto a biología los retos son:

- Identificar los componentes estructurales y funcionales que codifican el genoma humano, porque a pesar de que es fácil entender químicamente el ADN, la estructura es mucho más compleja.
- Desarrollar un entendimiento detallado de la variación hereditaria en el genoma humano.
- Entender la variación en la evolución dentro de las especies y el mecanismo detrás de ésta.
- Desarrollar políticas que faciliten el uso de la información genómica, tanto para la investigación como para cuestiones clínicas.

En el siguiente tema, que es de la salud, pueden listarse los siguientes retos:

- Traducir el conocimiento sobre los genomas en beneficios para la salud mediante el entendimiento del papel de los factores genéticos en una determinada enfermedad. Similarmente, definir aquellos factores no genéticos que también están involucrados en esa enfermedad con el fin de prevenir, diagnosticar y dar el tratamiento para la enfermedad.
- Desarrollar estrategias para identificar variaciones de genes que contribuyen a una buena salud y aumentan la resistencia a las enfermedades.
- Ligado al punto anterior, desarrollar maneras de predecir el efecto de los medicamentos para cierta enfermedad.

Los retos en cuanto a salud pueden englobarse en algo que el INMEGEN llama medicina genómica, definida como la identificación de las variaciones en el genoma humano, que confieren riesgo a padecer enfermedades comunes, con la finalidad de dar paso a una práctica médica más individualizada, más preventiva y más predictiva, como también lo apunta Celera Genomics[4].

Además del genoma humano existen otros organismos cuyos genomas han sido secuenciados. La Tabla I muestra datos obtenidos del EBI[5], con la cantidad de genomas secuenciados por cada clase de organismos.

Tabla I: Cantidad de genomas secueciados por clase de organismos.

Clasificación	Cantidad de genomas secuenciados
Archaeas	24
Bacterias	245
Eucariotas	47
Organelos	749
Virus	1042

En las tablas II y III se muestra información sobre algunos de los organismos más importantes, cuyos genomás han sido secuenciados.

Haemophilus Influenzae es la bacteria cuyo genoma fue el primero en ser secuenciado. El parásito *Plasmodium Falciparum* es la especie más mortal de las cuatro que causan la malaria, Yeh *et al.* (2004) hacen ver la importancia de la secuenciación del genoma de este parásito, lo cual permitiría estudiar a este organismo, y a través del entendimiento de los mecanismos celulares y las interacciones entre los componentes celulares, lograr el desarrollo de nuevos medicamentos y vacunas.

[4]http://www.celera.com/celera/targeted_medicine, agosto-2005

[5]European Bioinformatics Institute

Tabla II: Genomas secuenciados. Descripción y año de descubrimiento.

Nombre Científico	Descripción	Anunciado por	Año
Haemophilus Influenzae	Bacteria causante de una enfermedad potencialmente fatal	The Institute of Genomic Research	1995
Anopheles Gambiae	Mosquito responsable de la mayoría de los casos de malaria en África	Celera Genomics	2002
Plasmodium Falciparum	Parásito causante de la malaria	The Sanger Institute	2002
SARS coronavirus asociado	Coronavirus causante del Síndrome Agudo Respiratorio Severo	Genome Sciences Centre	2003

Tabla III: Genomas secuenciados. Tamaño y Método de Ensamblado.

Nombre Científico	Tamaño	Datos de la secuenciación	Fuente
Haemophilus Influenzae	1.83 Mb	Shotgun	Fleischmann *et al.* (1995) Pop *et al.* (2002)
Anopheles Gambiae	27.8 Mb	A partir de una cobertura shotgun de 10 de la variedad PEST de A. Gambiae	Holt *et al.* (2002)
Plasmodium Falciparum	22.8 Mb	Se utilizó la estrategia "Whole chromosome Shotgun"	Gardner *et al.* (2002)
SARS coronavirus asociado	29,751 b	Ensamblado mediante el programa PHRAP	Marra *et al.* (2003)

En cuanto a la motivación computacional, debe decirse que el problema de ensamble de secuencias se puede modelar como el problema de la Super Cadena más Corta (Tammi, 2003), (Waterman, 2000), (Gusfield, 1997), el cual pertenece a la clase NP-difícil (Gusfield, 1997), lo que significa que no se conoce método alguno que proporcione una solución óptima en tiempo polinomial.

Ante esta clase de problemas se han propuesto diversas estrategias, como las voraces y las evolutivas. Dentro de esta última encontramos las propuestas de Parsons *et al.* (1995) y Luque *et al.* (2005). Ellos plantean por primera vez un algoritmo evolutivo para el problema de ensamble de secuencias, pero los criterios utilizados para medir la

calidad de una solución no son completos, por ello no queda muy clara la efectividad de la propuesta para casos de ensamble de secuencias. En el presente trabajo se propone un algoritmo genético, y se utilizan varios criterios de calidad para medir el desempeño del algoritmo, los cuales también permiten comparar el rendimiento de este algoritmo contra el de una estrategia voraz.

I.2 Planteamiento del problema

Ante las ideas expuestas anteriormente, no cabe duda sobre la importancia de llevar a cabo la secuenciación del ADN, bien sea de genomas completos o de genes específicos. Un factor importante a mencionar es el tamaño de la cadena a secuenciar, de esto depende la técnica que puede aplicarse. En este trabajo la atención está puesta sobre el ensamblado de genomas grandes, donde la técnica más utilizada es la llamada *Shotgun*, que consiste a grandes rasgos en obtener aleatoriamente fragmentos de ADN de la cadena a secuenciar, posteriormente la secuenciación de estos fragmentos y finalmente una etapa de ensamble de los fragmentos secuenciados.

Como se ha mencionado, para la etapa de ensamble se cuenta con fragmentos secuenciados, que pueden tener errores derivados de la secuenciación y debido a la naturaleza del procedimiento shotgun; además la orientación (ver Capítulo II) de estos fragmentos es desconocida. Dado un conjunto de estos fragmentos secuenciados el objetivo es encontrar el orden correcto de tales fragmentos de tal manera que pueda reconstruirse la cadena de ADN de la cual fueron extraídos.

En este momento existen varios programas disponibles en Internet para el ensamble

de secuencias, algunos de los más importantes son: Phrap[6], TGI[7], CAP3 (Huang y Madan, 1999) y Euler[8] (Pevzner *et al.*, 2001), los tres primeros basados en algoritmos voraces, mientras que el último es la implementación de una propuesta apoyada en grafos *de Bruijn* en donde se busca una super-ruta Euleriana.

En este trabajo se hace frente a dicho problema con un algoritmo genético, motivado principalmente por los resultados satisfactorios de estos algoritmos en problemas complejos de optimización en general y en problemas de optimización combinatoria en particular. El problema de ensamblado pertenece justamente a esta última clase de problemas.

I.3 Objetivo de la tesis

I.3.1 Objetivo general

Evaluar el desempeño de los algoritmos genéticos para el ensamble de secuencias de ADN.

I.3.2 Objetivos específicos

- Diseñar un algoritmo genético para el ensamble de secuencias de ADN.
- Sintonización de los parámetros para el buen desempeño del algoritmo.
- Identificar características del problema que hacen difícil su resolución mediante el algoritmo genético diseñado.

[6]http://www.phrap.org, marzo-2005
[7]http://www.tigr.org/tdb/tgi/software, marzo-2005
[8]http://www-cse.ucsd.edu/groups/bioinformatics, abril-2005

- Comparar el algoritmo genético con TGI, programa (basado en una estrategia voraz) para el ensamble de secuencias disponible públicamente.

I.4 Organización de la tesis

En el Capítulo II, se explica la diferencia entre secuenciación y ensamble, este último, objeto de este trabajo. Asimismo se detalla el procedimiento *shotgun*, y con estos conceptos se define el problema del ensamble de secuencias de ADN. En este mismo capítulo se describe el enfoque utilizado para enfrentar el problema. También se propone un modelo para el problema y finalmente se menciona el trabajo previo.

En el Capítulo III se describe la metodología usada para enfrentar el problema de ensamble de secuencias de ADN. Primero se presenta una introducción hacia la computación evolutiva y en específico a los algoritmos genéticos. Posteriormente se describe el algoritmo genético propuesto, las variantes y la explicación de cada uno de los componentes de este algoritmo, incluyendo por supuesto los operadores genéticos.

En el Capítulo IV se describe el modelo de Engle y Burks (1993), usado para la generación de los casos de entrada para los experimentos. Se explican también los criterios de calidad utilizados para evaluar los resultados proporcionados por el algoritmo y se listan las secuencias utilizadas para los experimentos y los datos de cada uno de ellos. Con los resultados de los experimentos se realiza una comparación entre una función de aptitud propuesta por Parsons *et al.* (1995) y una propuesta en este trabajo. Se hace una comparación también entre los resultados del algoritmo genético y los resultados del programa TGI. Finalmente, se presenta una discusión sobre los resultados de los experimentos.

En el Capítulo V se presentan las conclusiones de este trabajo, junto con las aportaciones y algunas ideas para trabajo futuro.

En el Apéndice A se describe *seqaln*, programa disponible públicamente que ha sido integrado en este trabajo, para cálculo de dos tipos de alineamiento. El primero para la primera etapa del enfoque OLC y el segundo para medir la calidad de los resultados.

En el Apéndice B se muestran los resultados de algunos experimentos, para analizar el desempeño de las estrategias basadas en el enfoque OLC frente a casos cuyos segmentos son obtenidos de secuencias con repeticiones.

Capítulo II

Descripción del problema

En este capítulo se describe el problema del ensamble de secuencias de ADN. Primeramente se hace una comparación entre el proceso de secuenciación y el de ensamble de secuencias grandes; enseguida se describe el procedimiento de la técnica *shotgun* que es el utilizado para el ensamble de secuencias; posteriormente se presenta la definición del problema; también se describe el enfoque y el modelo utilizados para enfrentar este problema; y finalmente se describe el trabajo previo.

II.1 Secuenciación de ADN y ensamble de secuencias

Blazewicz *et al.* (2004) dividen el proceso de lectura de secuencias genómicas en tres fases, dependiendo de la longitud de la cadena analizada:

- **Secuenciación:** Consiste en determinar una secuencia de nucleótidos en un fragmento de ADN, es decir, encontrar el orden de las letras A, C, T y G dentro del fragmento. La longitud de estos fragmentos es usualmente de 500 pares de bases (letras), aunque esta longitud varía entre 350 y 1000 bases (Blazewicz *et al.*, 2004), (Staden *et al.*, 2001), (Gusfield, 1997).

- **Ensamble:** Consiste en combinar los fragmentos secuenciados para formar un conjunto de bloques traslapados que producen una cadena o secuencia más larga. Generalmente el ensamble se aplica a genomas completos de procariotas, o a

partes de cromosomas de genomas de eucariotas, como el genoma del ser humano que tiene más de 3000 millones de pares de bases (Tammi, 2003), (Venter *et al.*, 2001), (Parsons *et al.*, 1995).

- **Mapeo:** Consiste en encontrar la ubicación de esos bloques traslapados en el cromosoma.

Puede decirse que existe una proporción entre la secuenciación de ADN y el ensamble de secuencias. Por ejemplo, en la secuenciación por hibridación se desea encontrar una cadena final de cierta longitud, supongamos 500 bases, y como entrada se tienen oligonucleótidos[9] de cierto tamaño, por ejemplo de 10 a 20 bases. Por otro lado, en el ensamble se busca una cadena final de longitud mayor que en la secuenciación y como entrada no se tienen oligonucleótidos sino segmentos o bloques ya secuenciados, de hecho, cada uno de los fragmentos de entrada para el ensamble es una salida de un proceso de secuenciación. La Figura 2 muestra dicha comparación, primero en la Figura 2a se aprecia un conjunto de pequeños fragmentos ($seg\ i$), después del proceso de secuenciación se obtiene una cadena de una longitud mayor (Figura 2b); esta cadena, como lo muestra la Figura 2c, pasa a ser un elemento del conjunto de entrada para otro problema, que consiste en determinar una cadena final ensamblada, como se muestra en la Figura 2d.

El presente trabajo de tesis es una extensión natural del trabajo desarrollado por González (2004) sobre secuenciación de ADN; aunque nosotros estudiamos el ensamble de secuencias, seguimos una heurística similar a la de él. Además de la proporción que se menciona, existen otras diferencias entre secuenciación y ensamble, como los tipos de errores que se pueden encontrar en la entrada del segundo.

[9]Fragmentos pequeños de ADN

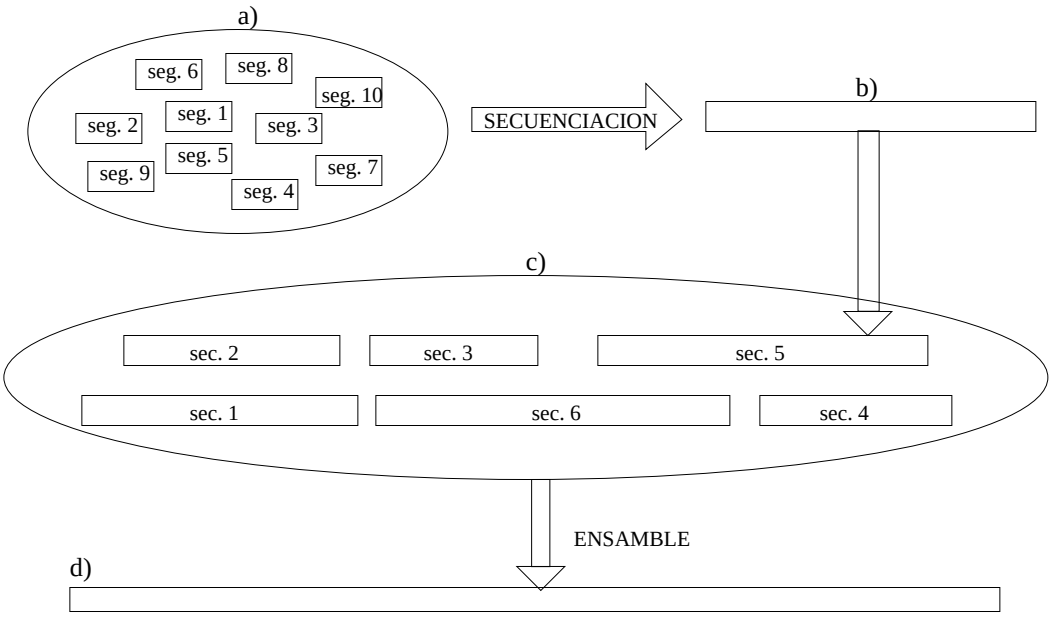


Figura 2: Secuenciación de ADN y ensamblado de secuencias. a) Entrada para el problema de secuenciación. b) Cadena producida en la secuenciación. c) Entrada para el problema de ensamble. d) Cadena producida en el ensamblado de secuencias

Para la secuenciación de genomas grandes, la técnica más utilizada (Tammi, 2003), (Staden *et al.*, 2001), (Parsons *et al.*, 1995) se conoce como Shotgun, la cual se describe en la siguiente sección.

II.2 Shotgun

Esta técnica fue desarrollada por Sanger *et al.* (1982), y fue probada al secuenciar el genoma del bacteriófago *Lambda*. En la Figura 3 se muestra una analogía del procedimiento shotgun mencionada por Zaritsky y Sipper (2004), los pasos que se siguen son los siguientes (Figura 3):

1. Se hacen varias copias de la cadena de ADN a secuenciar (Figura 3a y 3b).

2. Cada una de las copias se divide aleatoriamente (Figura 3c).

3. Se extraen los fragmentos (Figura 3d), y se secuencian. Tammi (2003) menciona que del conjunto de fragmentos, se eliminan a aquellos fragmentos muy pequeños o muy grandes, es decir, se toman solamente aquellos cuya longitud está dentro de un intervalo determinado. Una vez que se han secuenciado estos fragmentos, el problema consiste en ensamblarlos con el objetivo de armar la cadena de la cual fueron extraidos. Se espera que exista la suficiente cantidad de segmentos traslapantes para lograr este fin.

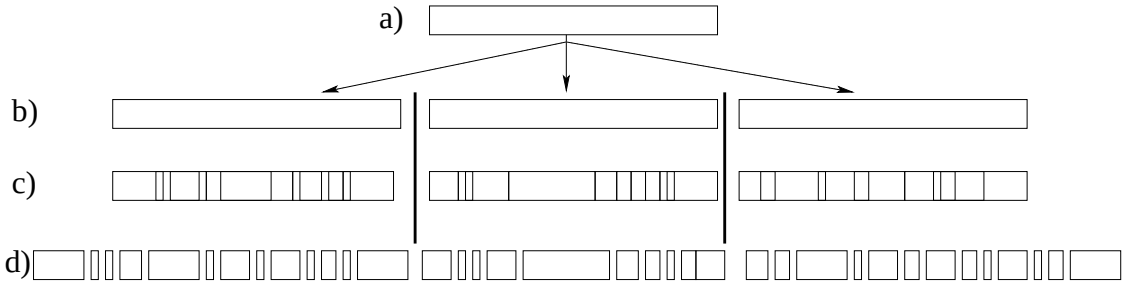


Figura 3: Analogía del procedimiento de la técnica Shotgun. a) Cadena a secuenciar. b) La cadena se copia. c) Las copias se dividen aleatoriamente. d) Los fragmentos obtenidos se secuencian y eso representa la entrada para el problema de ensamble.

Para completar la analogía del problema, se debe indicar que no se tiene información alguna sobre el orden de los segmentos.

Una analogía más extensa es mencionada por Pavel Pevzner[10]: imagine que se tienen varias copias de un libro; ahora por separado, cada una de las copias se divide aleatoriamente en miles de pedazos, enseguida se elimina el 10% de los pedazos, y al resto se le mancha con tinta, el objetivo es ahora armar una copia completa del libro. El conjunto de elementos eliminados corresponde a los fragmentos que no están dentro

[10]http://www.cs.ucsd.edu/users/ppevzner/research1.html, octubre-2004

de un intervalo de tamaño, y la mancha con tinta corresponde a los errores que pueden existir en los fragmentos, tales errores se explican posteriormente.

II.3 Definición del problema

Blazewicz *et al.* (2004) definen el ensamble de secuencias de ADN de la siguiente manera:

Definición 1. Como entrada se tiene un conjunto de secuencias sobre el alfabeto $\sum = \{A, T, C, G\}$, las secuencias pueden tener diferentes longitudes, desde cientos hasta miles de bases. La salida o solución es la secuencia más corta (SCS[11]) posible que contiene todas las secuencias de entrada como subcadenas.

El siguiente ejemplo describe brevemente la definición anterior. Supóngase un conjunto de segmentos P:

$\{p_1,...,p_8\}$ = {ATA,TGAT,GAC,ATAC,ACGA,CGATGA,ATGA,TACG}

La cadena más corta S^* es: CGATGATACGAC, donde cada uno de los segmentos $\{p_1,...,p_8\}$ es una subcadena en S^*.

Como puede observarse en la Figura 4, la solución propuesta es una cadena, pero a la vez es una secuencia de índices que representa a las subcadenas en el conjunto de entrada, en este ejemplo, esa cadena es la representación de la siguiente secuencia: p_6, p_7, p_2, p_1, p_4, p_8, p_5, p_3, que es el orden en el que se debe acomodar los segmentos para lograr el objetivo. Desde este momento puede verse que una solución para este

[11]Shortest Common Superstring

problema es una permutación de los índices que representan a cada segmento de la entrada.

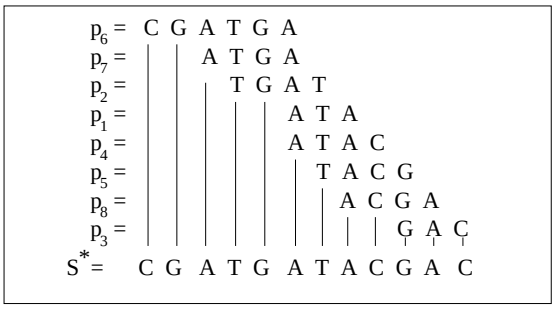


Figura 4: Supercadena más corta (SCS)

La definición 1 expone el problema de manera sencilla, sin embargo, se deben mencionar algunos factores que complican el ensamble de secuencias, como son los errores de bases y la orientación desconocida de los segmentos, tales factores se describen a continuación.

II.3.1 Errores en la entrada para el ensamble de secuencias

La entrada para el problema del ensamble consta de un conjunto de segmentos que pueden variar en tamaño, y como resultado de la etapa de secuenciación, estos segmentos pueden contener errores de bases. Los errores de bases se clasifican de la siguiente manera:

- Inserciones de nucleótidos (letras) en un segmento. En la Figura 5a se tiene un segmento p_1 al que se le denomina segmento correcto, y p_2 el segmento con error; el error en el segmento p_2 es la letra señalada que está sobrando, es decir, no aparece en el segmento p_1. En este caso la longitud del segmento p_2 aumenta en 1 debido a la letra sobrante.

- Eliminación de nucleótidos en un segmento. En la Figura 5b se señala con un recuadro una letra en el segmento p_1 que no aparece en el segmento p_2, entonces se dice que el segmento p_2 tiene un error de eliminación. La longitud del segmento p_2 se reduce en 1, debido a la letra faltante.

  Un error de inserción o eliminación es referido como indel, por los términos en inglés "**in**sertion/**del**etion".

- Sustitución de nucleótidos en un segmento. La Figura 5c señala un error de sustitución, en donde la letra C se cambia por la T, en este caso la longitud del segmento p_2 se mantiene igual, pero conteniendo un error.

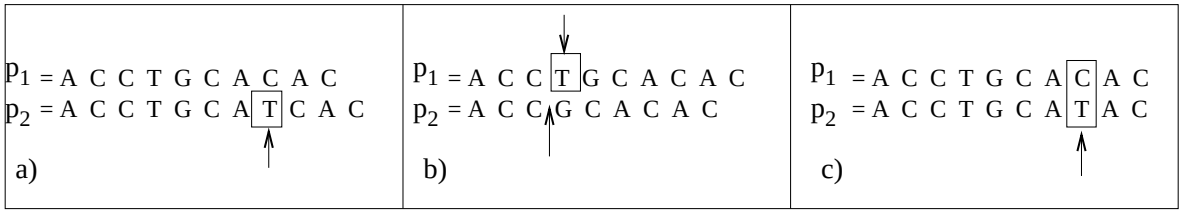


Figura 5: Errores de bases en los segmentos de entrada. a) Inserción de un nucleótido, b) Eliminación de un nucleótido, c) Sustitución de un nucleótido

Además de los errores descritos, existe otro factor que complica el problema del ensamble de secuencias, la orientación de los segmentos. Con la técnica de shotgun, cuando se extraen los segmentos de una copia de la cadena objetivo, estos se pueden obtener de cualquiera de las dos hebras de la molécula de ADN. La Figura 6a muestra una región ampliada de la doble hélice de una molécula de ADN, en ella se aprecian dos segmentos. En la Figura 6b se muestra el primer segmento (segmento a) y se realiza la lectura en el sentido de 5' a 3': ACTGTGACCAAC; en la Figura 6c se muestra el segundo segmento (segmento b), y la lectura se realiza en el mismo sentido, de 5' a 3', de esta manera el segmento que se extrae no es TGACACTGGTTG sino

GTTGGTCACAGT. Esta es una fuente de variación, y no hay forma de saber de cuál de las dos hebras se ha extraído un segmento.

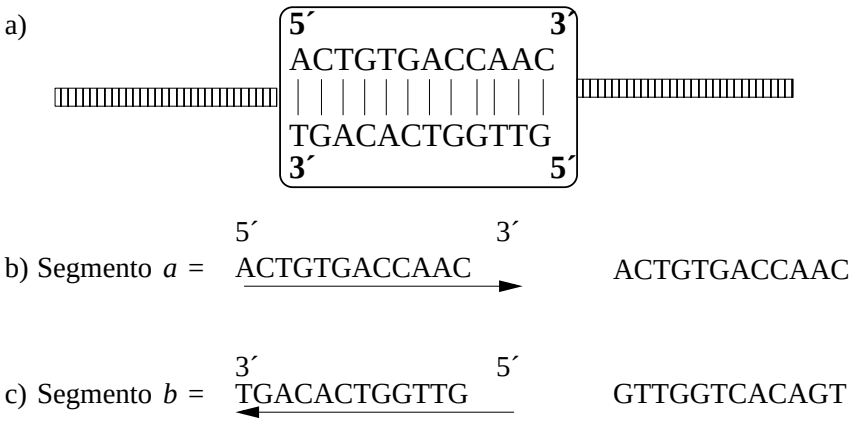


Figura 6: Orientación de los segmentos de ADN. a) Región ampliada de una molécula de ADN. b) Lectura de un segmento a. c) Lectura de un segmento b (complemento inverso de a).

Integrando toda la información sobre shotgun se procede a definir el problema del ensamble de secuencias de ADN de la siguiente manera:

Definición 2. La entrada es un conjunto P de segmentos que están sobre el alfabeto $\sum = \{A, T, C, G\}$, cada uno de los segmentos p_i puede tener diferente longitud, la orientación de cada segmento p_i es desconocida, y además cada segmento puede contener errores de bases. La salida o solución consiste en encontrar una permutación Π de los $|P|$ segmentos, de tal manera que el puntaje de alineamiento F(Π) entre ellos sea máximo:

$$F(\Pi) = \frac{\sum_{i=0}^{n-2} w(\Pi[i], \Pi[i+1])}{c(\Pi)} \tag{1}$$

donde,

- n es el total de segmentos.

- $\Pi[i]$ devuelve el índice del segmento ubicado en la posición i dentro de la permutación Π.
- $w(\Pi[i], \Pi[i+1])$ es el puntaje de alineamiento local condicionado entre el segmento $\Pi[i]$ y el segmento $\Pi[i+1]$.
- $c(\Pi)$ es la cantidad de contigs[12] en la solución dada por la permutación Π.

Cabe mencionar que el puntaje de alineamiento local condicionado se obtiene a partir del cálculo de un alineamiento local bajo ciertas condiciones, una vez hecho este cálculo han de considerarse otros parámetros, y es la interpretación de este conjunto de datos la que da como resultado un puntaje de *alineamiento local condicionado (alc)*. En el Capítulo III se dan más detalles al respecto.

II.4 Enfoque para resolver el problema

Antes de continuar con la descripción del enfoque utilizado, se lista una serie de conceptos que se usan con frecuencia en las secciones posteriores:

Gusfield (1997) define:

- Cadena: Una cadena S es una lista ordenada de caracteres escritos contiguamente de izquierda a derecha.
- Subcadena: Los caracteres en una subcadena de S deben estar contiguamente en S. Para cualquier cadena S, S[$i..j$] es la subcadena de S que inicia en la posición i y termina en la posición j de S.

[12] Disposición de fragmentos traslapantes contiguos

- Prefijo: S[1..i] indica los primeros i caracteres de S.

- Sufijo: S[i..|S|] es la subcadena de S que inicia en la posición i y finaliza en $|S|$, donde $|S|$ denota el número de caracteres de S. Un prefijo, sufijo o subcadena propios, son respectivamente un prefijo, sufijo o subcadena que no es la cadena S completa, ni la cadena vacía.

  La Figura 7a muestra una cadena S, y una subcadena S[4..10], donde cada uno de los caracteres de la subcadena están contiguos en la cadena S. La Figura 7b ilustra el ejemplo de un prefijo S[1..7], en este caso de los primeros siete caracteres de la cadena S. En la Figura 7c se muestra un sufijo S[7..13], que inicia en la posición 7 de la cadena S.

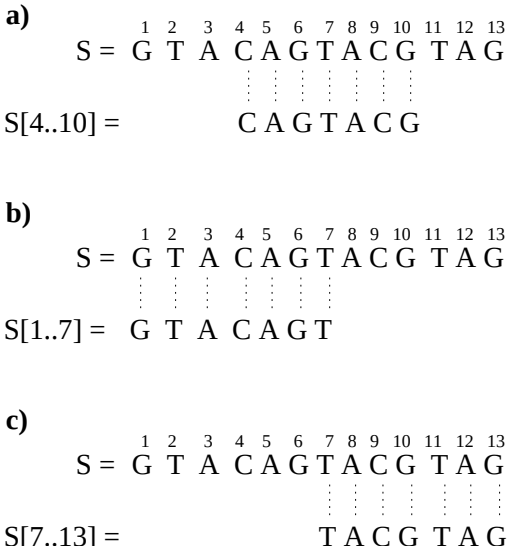


Figura 7: Subcadena, prefijo y sufijo. a) Subcadena [4..10] de la cadena S. b) Prefijo S[1..7]. c) Sufijo S[7..13].

- S[i..j] es la cadena vacía si $i > j$.

- Traslape: El traslape entre dos cadenas S_1 y S_2 es el sufijo propio más largo de S_1 que es prefijo de S_2.

- Contig (Staden, 1980): Es una disposición de fragmentos traslapantes contiguos. En la Figura 8a se observa una disposición de segmentos, por un lado traslapando los segmentos a y b, y por otro lado los segmentos, c, d, e y f, esto da lugar a los dos contigs mostrados en la Figura 8b.

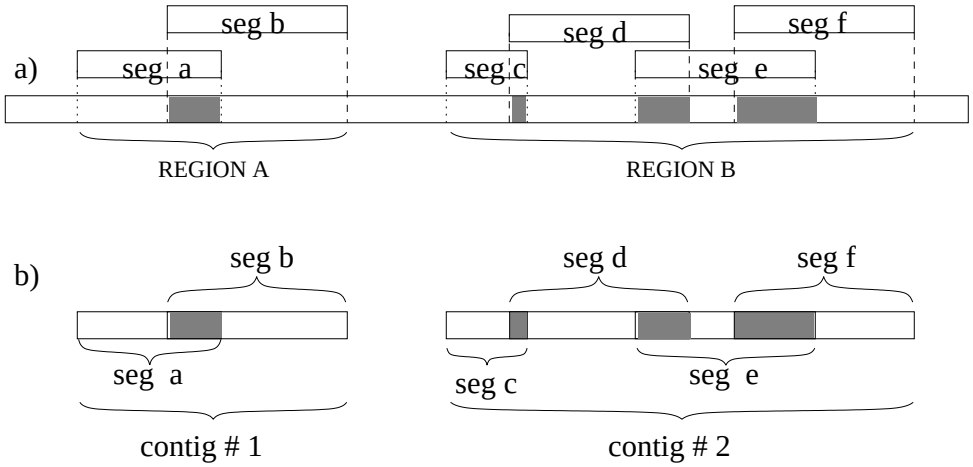


Figura 8: Traslapes y contigs. a) En la región A se observan dos fragmentos traslapantes, y cuatro en la región B. b) Contigs formados con los seis fragmentos traslapantes.

- Fusión: La fusión de dos fragmentos traslapantes a y b es el fragmento formado por los primeros i caracteres de a que no traslapen con b, seguidos de los j caracteres en los que traslapan a y b, más los k caracteres que no traslapan de b, este concepto se ilustra en la Figura 9.

El enfoque utilizado en este trabajo para resolver el problema de ensamble de secuencias se denomina "Overlap-Layout-Consensus" (OLC), fue propuesto por Peltola *et al.* (1984), Kececioglu y Myers (Pop *et al.*, 2002). Este enfoque se basa en la suposición

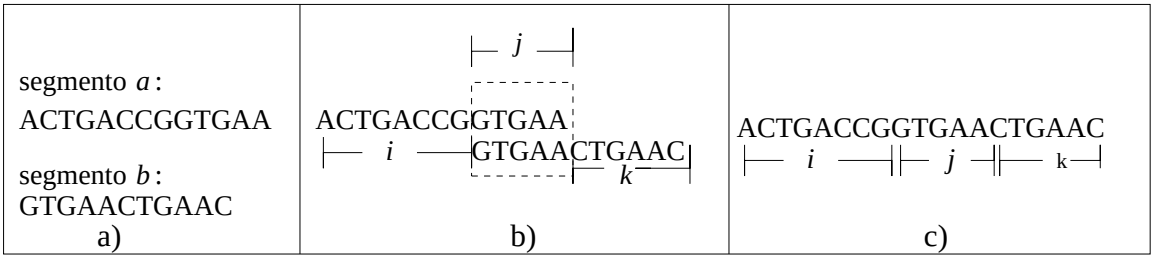


Figura 9: Fusión de segmentos. a) Dos segmentos a y b. b) Traslape entre los segmentos a y b. c) Fusión de los segmentos a y b.

de que una vez obtenidos los fragmentos mediante la técnica shotgun, existe suficiente cantidad de ellos y su distribución es lo más uniforme posible para lograr ensamblar por completo la secuencia original. La Figura 10 ilustra el ejemplo de un caso, en donde se cuenta con la información (segmentos) suficiente para ensamblar completamente la secuencia.

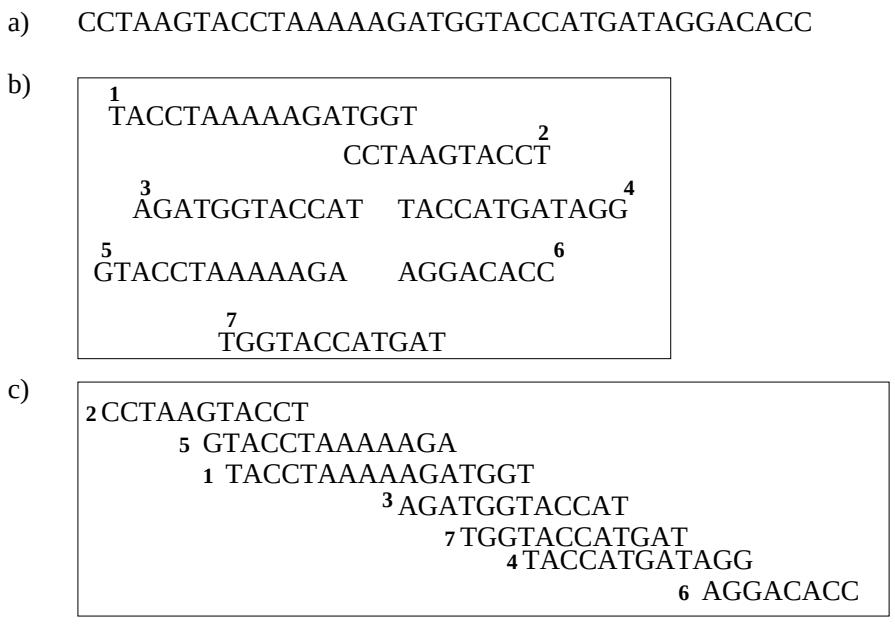


Figura 10: Caso con información suficiente para un ensamble completo. a) Cadena original. b) Conjunto de segmentos extraidos de la cadena original. c) Ensamble de los segmentos del conjunto de entrada

La descripción de las tres fases del enfoque se presenta a continuación:

- *Detección de traslape.* En esta fase se calculan los traslapes entre cada par de segmentos. La razón por la que se hace esto es porque la única información disponible para resolver el problema es el conjunto de segmentos del que se ha hablado. y se hace la suposición que si dos segmentos son vecinos, es decir, provienen de la misma zona de la secuencia original, entonces debe existir un traslape entre ellos, a menos que fueran segmentos contiguos.

  En la Figura 11a vemos que los segmentos a y b se extraen de la región A, razón por la cual tienen en común la zona sombreada, es decir, traslapan en esa zona; lo mismo se observa con los segmentos c y d que provienen de la región B. En la Figura 11b se muestra un caso en el que los segmentos a y b provienen de una región A, pero no existe un traslape entre ellos, y no hay forma de saber que el segmento b está a continuación del a, a menos que hubiera un segmento c que enlazara ambos segmentos, como se muestra en la Figura 11c.

  Un factor que complica a este enfoque es la existencia de regiones repetidas dentro de la secuencia original, tal y como mencionan Pevzner *et al.* (2001) y Parsons *et al.* (1995). Los programas ensambladores de secuencias que se basan en este enfoque suelen tener dificultades para ensamblar este tipo de secuencias. En la Figura 12a se muestra el caso de una secuencia con una región repetida, llamada región R, supóngase que se han extraído los segmentos a, b, c, d y e, donde los segmentos a y b traslapan y cuya fusión contiene a la región R, y por otro lado, la fusión de los segmentos d y e también contiene a la región R. En la Figura 12b aparece el ensamble correcto de estos segmentos. La confusión aparece si el

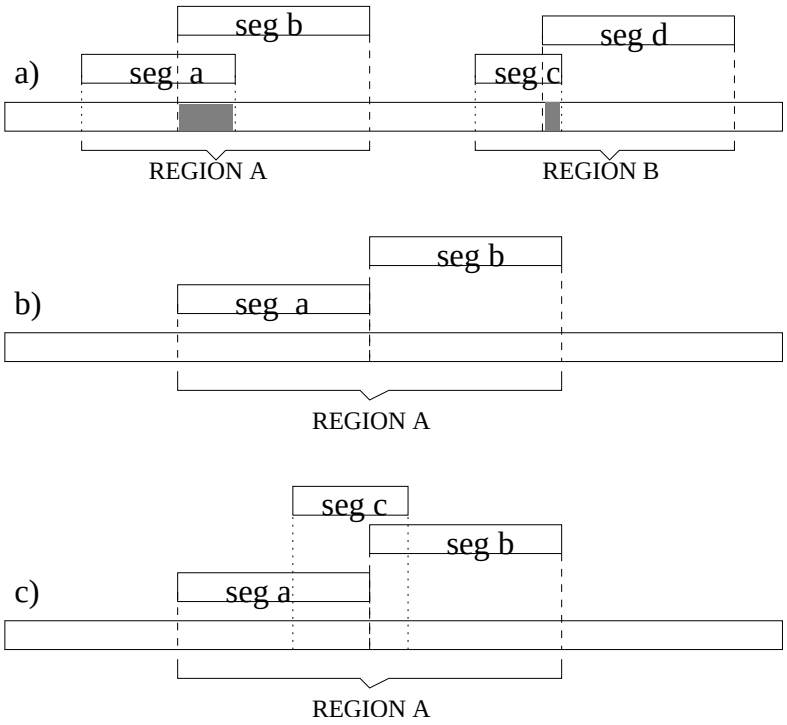


Figura 11: Traslapes entre segmentos. a) Los segmentos traslapantes a y b extraidos de una region A y los segmentos c y d de una región B. b) Segmentos extraidos de la misma región pero sin traslape. c) Segmentos extraidos de la misma región

fragmento *a* traslapa con el fragmento *e* y produce también la región R, entonces la incógnita es saber cuál de las dos fusiones es la correcta, porque si se elige la incorrecta se estarían fusionando regiones que realmente están separadas. En la Figura 12c se observa un ensamblado incorrecto ocasionado por una región repetida.

- *Ordenamiento de fragmentos.* Esta es la etapa en donde se infiere el orden de los fragmentos que permita la reconstrucción de la cadena original. Para lo cual únicamente se tiene la información de los traslapes.

- *Consenso.* En esta etapa se trata de hacer las fusiones adecuadamente para formar los "contigs". Puede parecer una tarea sencilla, sin embargo, hay que recordar

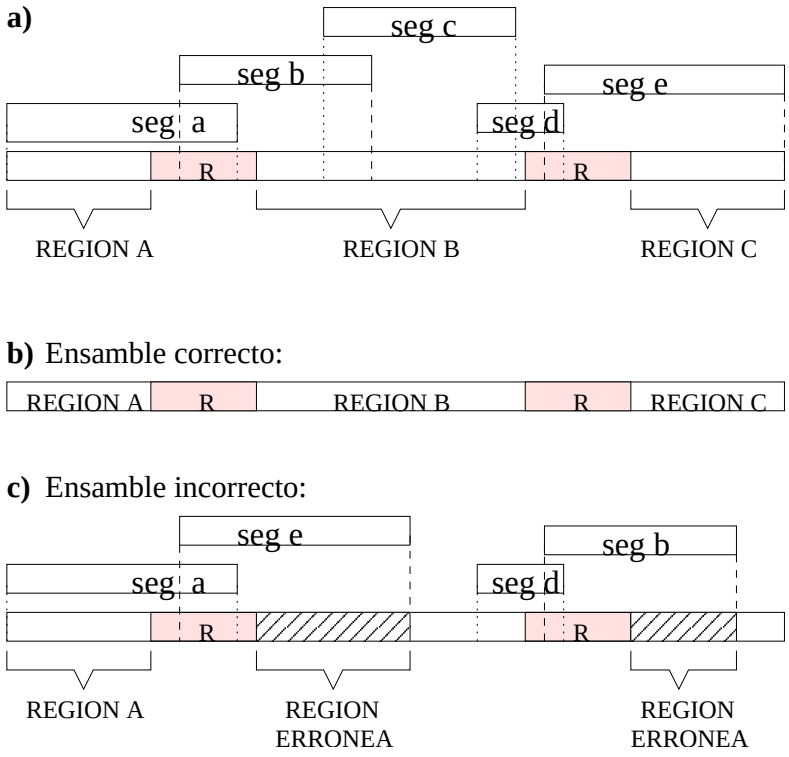


Figura 12: Regiones repetidas. a) Conjunto de segmentos traslapantes. b) Ensamble correcto de los segmentos de entrada. c) Ensamble incorrecto

los errores posibles y por ende las complicaciones que pueden presentarse.

El éxito de cualquier enfoque depende en gran medida de la calidad de los segmentos y de la cantidad de ellos, es decir, de la cobertura, concepto sumamente importante, descrito en el Capítulo IV.

II.5 Modelos para el problema de ensamble

Existen varios modelos para el problema de ensamble de secuencias, Blazewicz *et al.* (2004) mencionan que se puede modelar como el problema de la supercadena común más corta (SCS) que a la vez se puede reducir al problema del agente viajero (TSP) (Tammi, 2003), (Parsons *et al.*, 1995). Tammi (2003) menciona que SCS es un modelo

sub-óptimo para el problema del ensamble de secuencias, porque este modelo tiende a colapsar o contraer las regiones con repeticiones, y porque también el modelo no considera ninguno de los tipos de error ni el factor de la orientación de los segmentos; así mismo, Tammi (2003) y Waterman (2000) mencionan el modelo llamado Problema de Reconstrucción de Secuencia (SRP), el cual sí considera los errores y la orientación desconocida, este modelo contempla una fase de corrección de errores, donde se busca el mínimo de correcciones en cada uno de los segmentos de entrada para producir la cadena más corta.

II.5.1 Modelo propuesto

El modelo propuesto en este trabajo para el ensamble de secuencias se representa mediante un grafo. Partiendo de la entrada del problema, en la que se tiene un conjunto P formado de k segmentos de ADN de longitud variable, cuya orientación es desconocida y que pueden contener errores de bases, se propone formar un grafo completo dirigido G=(V,E) de la siguiente manera:

- **Los vértices:** son cada uno de los k segmentos iniciales a los cuales se les denomina $p_1, p_2, ..., p_k$, además se suman a ellos k segmentos más, que son los complementos inversos de cada uno de los k segmentos iniciales, a los cuales se les denomina $p_{1'}, p_{2'}, ..., p_{k'}$, cada uno de estos $2k$ vértices $\in$ V tiene un peso igual a 1.

- **Los arcos:** En el grafo G, cada arco $(p_i, p_j) \in$ E tiene una ganancia $g_{i,j}$ que representa el alineamiento local entre los segmentos i y j, siempre y cuando éste alineamiento se de entre un sufijo propio de i y un prefijo de j.

- **Objetivo:** Encontrar una ruta r en el grafo, de tal manera que la suma de las ganancias de los arcos recorridos sea máxima, y la suma de los pesos de los vértices

no exceda k, esto para evitar utilizar más de los k segmentos originalmente en la entrada; y una condición más importante es la siguiente: para cualquier ruta r, un segmento p_i es considerado en ella siempre y cuando el segmento $p_{i'}$ no esté incluido previamente en esa ruta r, y de igual forma, un segmento $p_{i'}$ puede ser incluido en una ruta r, siempre y cuando el segmento p_i no se haya considerado.

En la Figura 13 se muestra un grafo formado a partir de una entrada de tres segmentos, los vértices p_1, p_2 y p_3 representan a estos segmentos, el grafo se completa con los vértices: $p_{1'}$, $p_{2'}$ y $p_{3'}$ que representan a los complementos inversos de los tres segmentos originales.

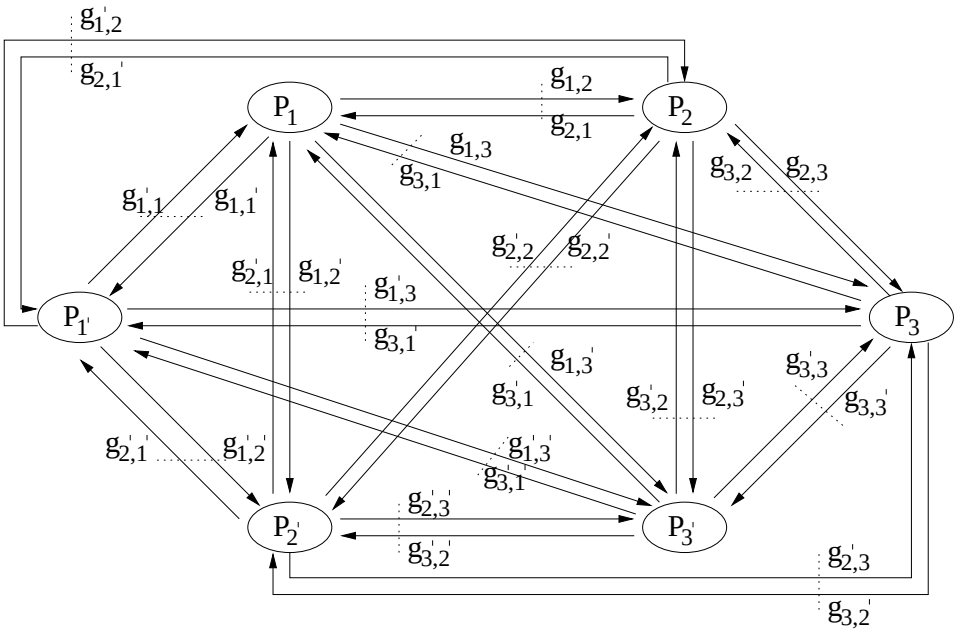


Figura 13: Grafo formado a partir de una entrada original de tres segmentos

II.6 Trabajo previo

II.6.1 Parsons *et al.* (1995)

En su trabajo Parsons *et al.* (1995) utilizan un algoritmo genético con el enfoque OLC. Para la representación utilizan una variante a la llamada "sorted-order" ó "random-key". Proponen dos funciones de aptitud, una tiene como objetivo maximizar la suma de los traslapes de los fragmentos adyacentes en una solución dada, y la otra tiene como objetivo minimizar la ocurrencia de traslapes fuertes entre fragmentos no adyacentes. El operador de cruzamiento utilizado se llama "cruza de dos puntos". Mencionan dos operadores de mutación: inversión y transposición, que afectan a bloques completos, específicamente a contigs, es decir, la inversión y transposición no es a nivel de fragmentos sino a nivel de contigs.

Para algunos experimentos, introducen un porcentaje de error de 10% para sustituciones y 5% para indels, y en cuanto al factor de la orientación desconocida, únicamente se menciona que en la etapa de detección de traslapes, para cada par de segmentos se busca el traslape considerando ambas orientaciones de ambos segmentos. Aunque se mencionan la similitud de los contigs con la cadena original y la cantidad de contigs como criterios de calidad, solamente la cantidad de contigs es la que aparece en los resultados que muestran.

II.6.2 Luque *et al.* (2005)

Luque *et al.* (2005) atacan el problema de ensamble de secuencias también con el enfoque OLC, utilizan la representación por permutación, las funciones de aptitud son las propuestas por Parsons *et al.* (1995). Con respecto a los operadores de cruzamiento, utilizan el "order-based crossover (OX)" y el "edge-recombination crossover (ER)", el

operador de mutación utilizado se llama de intercambio, y consiste en elegir aleatoriamente dos posiciones del individuo o solución y se intercambian los fragmentos de esas posiciones.

A pesar que dentro del planteamiento del problema mencionan los errores de bases y la orientación desconocida de los segmentos, ninguno de estos dos factores se menciona dentro de los experimentos que realizaron, el criterio que usan para calificar los resultados de los experimentos es la cantidad de contigs generados por el algoritmo y el tiempo de cómputo.

II.6.3 Pevzner *et al.* (2001)

En este trabajo se presenta una propuesta que abandona el enfoque OLC, según Pevzner *et al.* (2001) el enfoque OLC no tiene un buen desempeño cuando se trata de ensamblar genomas grandes. En ese trabajo se hace referencia a varios modelos que siguen el enfoque OLC y que presentan dificultades, también se mencionan las dificultades de este enfoque frente a las repeticiones.

La propuesta de Pevzner *et al.* (2001) parte de una vieja idea denominada SBH (Sequencing by Hybridization), se menciona que conceptualmente son similares SBH y el problema del ensamble de secuencias, la diferencia radica en que los segmentos en SBH son mucho más pequeños.

En este trabajo se explica la idea de la super ruta Euleriana aplicada a SBH, para lo cual se construye un grafo cuyas aristas corresponden a l-tuplas y el objetivo es encontrar una ruta que visite cada arista en el grafo sólo una vez. Ahora bien, en SBH se

tienen tuplas o segmentos, todas del mismo tamaño, como se ha visto, mediante shotgun se producen segmentos que pueden variar en tamaño, he aqui uno de los puntos interesantes de esta propuesta, la reducción del problema del ensamble al de SBH.

Lo que proponen es dividir un segmento de tamaño n en $n - l + 1$ l-tuplas, con esta reducción y algunos ajustes puede aplicarse la idea que menciona Pevzner *et al.* (2001) para SBH.

II.7 Implementaciones disponibles en Internet

A continuación se listan algunas implementaciones disponibles en internet de ensambladores de secuencias y las direcciones electrónicas en donde se pueden encontrar.

- Phrap en http://www.phrap.org/
- CAP3 en http://seq.cs.iastate.edu/
- Euler en http://www-cse.ucsd.edu/groups/bioinformatics
- TGI en http://www.tigr.org/tdb/tgi/software

El programa Phrap usa una combinación de información sobre la calidad de datos, una parte proporcionada por el usuario y otra calculada internamente, con el fin de mejorar el ensamble en la presencia de repeticiones.

Huang y Madan (1999) recomiendan usar CAP3 para casos en los que no se tiene la información sobre la calidad de los datos. Ellos mencionan que para esos casos, CAP3 produce mejores resultados que PHRAP en cuanto a errores, aunque PHRAP produce

contigs más largos.

TGI es un programa que utiliza una versión modificada del programa NCBI's[13] megablast para agrupar segmentos de acuerdo a la similitud entre ellos; y en una etapa de ensamble utiliza CAP3. Este programa es bastante sencillo de utilizar, funciona con o sin los archivos de calidad mencionados anteriormente; además puede paralelizarse.

Los programas anteriores están basados en el enfoque OLC, mientras que el programa Euler es la implementación de la propuesta de Pevzner *et al.* (2001). Aunque Euler es un programa de fácil utilización, los resultados que produce con los datos de entrada (con errores) utilizados en este trabajo son de baja calidad.

[13]http://www.ncbi.nlm.nih.gov/BLAST, marzo-2005

Capítulo III

Metodología

En este capítulo se describe la metodología utilizada para enfrentar el problema de ensamble de secuencias, metodología que involucra el uso de técnicas evolutivas. En este capítulo se proporciona una introducción hacia la computación evolutiva y específicamente a los algoritmos genéticos; seguido a ello se menciona el trabajo previo en cuanto a algoritmos genéticos para el problema de ensamble de secuencias. Posteriormente se muestra el algoritmo genético propuesto, detallando cada una de las variantes propuestas, y por supuesto los operadores genéticos utilizados.

III.1 Computación Evolutiva

La metodología de este trabajo se basa en la computación evolutiva, ésta es básicamente una técnica inspirada en los principios de la teoría Neo-Darwiniana de la evolución natural. Dentro de la computación evolutiva o algoritmos evolutivos, como también se le conoce, existen tres paradigmas, Eiben y Smith (2003) se refieren a ellos de la siguiente manera:

- **Programación evolutiva:** Fue propuesta en los años 60's, ésta consiste en hacer evolucionar autómatas de estados finitos; los autómatas son expuestos a una serie de símbolos de entrada, y se espera que éstos evolucionen de tal manera que puedan predecir futuras secuencias de símbolos. Para tal objetivo se utiliza un operador de mutación que afecta a los cambios de transición y a los estados del autómata; desde luego, existe un indicador para hacer los cambios, esto es, una

función de aptitud, concepto que se describe más adelante.

- **Estrategias evolutivas:** Fueron desarrolladas también en los años 60's para resolver problemas hidrodinámicos. En este caso, se parte de una solución determinada para un problema dado, se realizan ajustes discretos aleatorios a esta solución y el resultado es considerado como una segunda solución, es decir, un descendiente, éste descendiente se evalua y en caso de ser mejor que su predecesor, la primera solución se sustituye por la segunda y se repite el procedimiento.

- **Algoritmos genéticos:** Fueron desarrollados por John H. Holland a inicio de los 60's, motivado para resolver problemas de aprendizaje de máquina. El algoritmo genético enfatiza la importancia del cruzamiento sobre la mutación.

En general, para simular el proceso evolutivo a través de una computadora, se necesita lo siguiente:

- Codificación de las estructuras, es decir, la traduccción de una solución en el contexto real del problema ($fenotipo$) al contexto de la computación evolutiva ($genotipo$), cada solución codificada o traducida se llama *individuo*.

- Operaciones que afecten a los individuos, por ejemplo, el cruzamiento y la mutación.

- Una función de aptitud, que evalua a los individuos.

- Un mecanismo de selección de los mejores individuos que han de reproducirse o cruzarse.

La siguiente subsección está dedicada a los algoritmos genéticos, renglón de la computación evolutiva que hemos tomado para el presente trabajo.

III.1.1 Algoritmos Genéticos

Un algoritmo genético contempla los siguientes pasos, en los cuales está contenida la esencia de la computación evolutiva:

- Generación aleatoria de una población[14] inicial.
- Cálculo de la función de aptitud para cada individuo.
- Selección probabilística de los individuos en base a la aptitud.
- Aplicación de operadores genéticos, como el de cruzamiento y mutación, para generar la siguiente población.
- Repetición del proceso hasta que cierta condición se satisfaga.

Enseguida se definene algunos conceptos importantes dentro de la computación evolutiva, éstos se ilustran en la Figura 14:

- Cromosoma: es la estructura que representa al individuo, en este caso es un arreglo en el cual habrá una permutación con los n índices de los segmentos. Se dice que un cromosoma está compuesto por genes.
- Individuo: en términos de un cromosoma, se dice que un individuo es un cromosoma con valores en los genes, es decir, es un cromosoma con valores particulares.
- Locus: son cada una de las posiciones del cromosoma, es decir, los índices del arreglo.
- Alelo: es el valor de un gen, es decir, es el valor que éste toma en un determinado locus.

[14] Conjunto de individuos

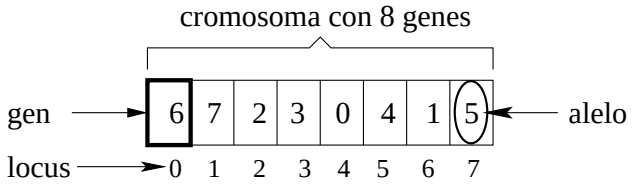


Figura 14: Conceptos: cromosoma, gen, locus y alelo

En la siguiente subsección se menciona el trabajo previo en cuanto a algoritmos genéticos para el problema de ensamble de secuencias.

III.1.2 Trabajo previo

Se han propuesto algunos algoritmos genéticos para el problema de ensamble de secuencias. A continuación se presentan algunas propuestas y una breve descripción de los puntos más importantes, como lo es la función de aptitud y los operadores genéticos.

Parsons *et al.* (1995)

En este trabajo utilizan un algoritmo genético con el enfoque OLC. La codificación o representación utilizada por Parsons *et al.* (1995) es una variante a la llamada "sorted-order" ó "random-key" Bean (1992). En ese trabajo se proponen dos funciones de aptitud, la primera:

$$F1(\Pi) = \sum_{i=0}^{n-2} w(\Pi[i], \Pi[i+1]) \tag{2}$$

donde,

- Π es una permutación de los índices de los segmentos.
- n es la cantidad de fragmentos en la entrada.

- $\Pi[i]$ devuelve el índice del segmento ubicado en la posición i dentro de la permutación Π.
- $w(\Pi[i], \Pi[i+1])$ es la longitud del traslape entre los segmentos $\Pi[i]$ y $\Pi[i+1]$.

Esta función devuelve la suma de los traslapes de los fragmentos adyacentes en una solución dada, el objetivo es maximizar el valor de F1. La Figura 15 ilustra el cálculo de la aptitud utilizando la Ecuación 2.

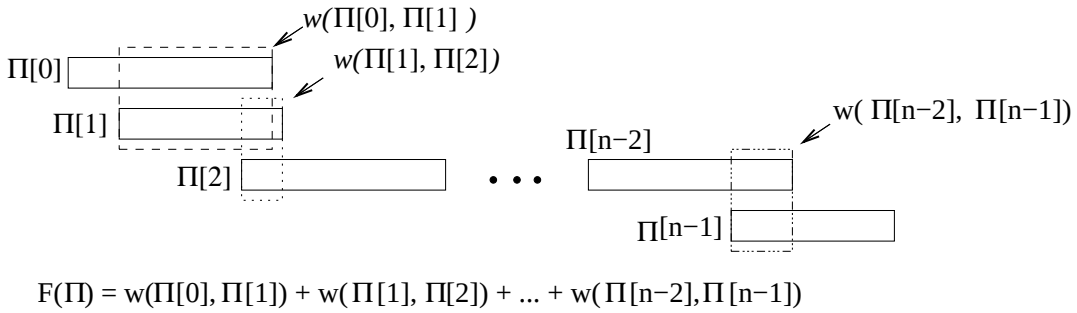


Figura 15: Función de aptitud definida en la Ecuación 2.

La segunda función de aptitud es:

$$F2(\Pi) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} |i-j| * w(\Pi[i], \Pi[j]) \tag{3}$$

Esta función considera además del traslape entre fragmentos adyacentes, al traslape entre todos los segmentos de una solución, el objetivo es minimizar el valor de F2. El punto clave de esta función es que penaliza soluciones en las que existen traslapes fuertes entre segmentos no adyacentes, dentro de una permutación Π. Supóngase el ejemplo mostrado en la Figura 16, donde se muestra una solución Π en la que existen dos segmentos $\Pi[1]$ y $\Pi[i]$ que tienen un buen traslape pero no son adyacentes (en la permutación Π), este caso se penaliza, dependiendo de la longitud del traslape y de la

cantidad de segmentos que separa al segmento $\Pi[1]$ del $\Pi[i]$ dentro de esa permutación Π. Se espera que al utilizar esta función de aptitud, los segmentos que traslapan se vayan agrupando.

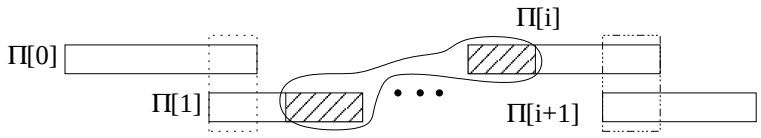


Figura 16: Función de aptitud definida en la Ecuación 3

El operador de cruzamiento utilizado en Parsons *et al.* (1995) se llama "cruzamiento de dos puntos", en el cual se seleccionan dos puntos aleatoriamente y los bits en esos dos grupos se intercambian, debido a que cada segmento de un individuo se representa mediante un conjunto de bits, el intercambio es de conjuntos de bits. Se mencionan dos operadores de mutación: inversión y transposición, que afectan a bloques completos, específicamente a contigs, es decir, la inversión y transposición no es a nivel de fragmentos sino a nivel de contigs.

La mayoría de los conjuntos de entrada para los experimentos realizados por Parsons *et al.* (1995) se generan artificialmente mediante el generador Genfrag de Engle y Burks (1993). Las secuencias de ADN que utilizan como base son: HUMATPK01 de 2026 pb[15] de longitud, con número de acceso[16] M55090; HUMMHCFIB de 3835 bp de longitud y número de acceso X60189; y HUMAPOBF de 10089 pb de longitud con número de acceso M15421. En tales experimentos utilizan rango de cobertura[17] entre 3 y 7, y longitud media de los fragmentos entre 300 y 500 bases. En cuanto a los errores

[15]pares de bases

[16]número de acceso en el Genbank

[17]medida de redundancia de los datos de entrada (detalles en el sig. capítulo)

introducidos en los conjuntos de entrada, manejan un 10% para errores de sustitución y 5% para errores de inserción y eliminación. También experimentan con una secuencia más larga, una secuencia de 20 Kpb del genoma del bacteriófago lamba (LAMCG), con número de acceso J02456. En el primer paso del enfoque OLC, Parsons *et al.* (1995) mencionan que para cada par de segmentos se busca un traslape considerando ambas direcciones de ambos segmentos, para la etapa de ordenamiento de fragmentos la única información es la de la etapa anterior, y mediante la función de aptitud se trata de inferir el orden de los segmentos, una vez establecido un orden se lleva a cabo un alineamiento múltiple, ésto para detectar posibles errores de bases y finalmente el consenso se realiza analizando columna por columna el alineamiento múltiple.

En cuanto a los resultados, a pesar que mencionan que el mejor criterio de calidad de una solución es la similitud de ésta con la cadena original, los resultados que muestran en su trabajo no incluyen tal criterio. Parsons *et al.* (1995) ponen énfasis en comparar el desempeño de diferentes operadores pero utilizando la cantidad de contigs como el criterio principal de calidad, y el puntaje de la función de aptitud como otro criterio más.

Luque *et al.* (2005)

Luque *et al.* (2005) mencionan que un algoritmo genético secuencial no es capaz de dar buenos resultados cuando se trata de secuencias largas de ADN (arriba de 15 kpb), primeramente en cuanto al tiempo de cómputo, y además mencionan que obtienen mejores resultados con la propuesta llamada *algoritmo genético distribuido*. En términos generales, la propuesta de ellos es una implementación en paralelo en forma de anillo, el cual se forma por un conjunto de k subpoblaciones (subconjunto de individuos) que

evolucionan independientemente. Con cierta frecuencia estas subpoblaciones intercambian determinado número de individuos. El algoritmo está basado en el enfoque OLC, para la fase de detección de traslapes utilizan un algoritmo de programación dinámica aplicado a un alineamiento semiglobal, y es en la segunda etapa en donde aplican un algoritmo genético.

Utilizan la representación por permutación, en donde se asigna un identificador entero a cada segmento de la entrada; para asegurar la legalidad de las soluciones, se establecen las condiciones siguientes:

- Todos los fragmentos de la entrada deben tener un número que los identifique.
- No deben existir segmentos con el mismo número de identificación.

En este trabajo utilizan las funciones de aptitud propuestas por Parsons *et al.* (1995). Con respecto a los operadores de cruzamiento, Luque *et al.* (2005) utilizan el "order-based crossover (OX)" (Eiben y Smith, 2003) y el "edge-recombination crossover (ER)" (Eiben y Smith, 2003). El operador de mutación utilizado se llama de intercambio, y consiste en elegir aleatoriamente dos posiciones del individuo o solución y se intercambian los fragmentos de esas posiciones.

Las conclusiones de Luque *et al.* (2005) son que la primera función de aptitud de Parsons *et al.* (1995) es mejor que la segunda. Concluyen que el operador "order-based crossover (OX)" tiene mejor desempeño que el "edge-recombination crossover (ER)", y que éste se debe aplicar a todos los individuos de la población. La probabilidad de mutación para cada individuo la establecen en 0.3, en base a los resultados de algunos experimentos. Ellos mencionan que encontraron que en algunos casos, es posible que

una solución con mejor aptitud que otra, genere más contigs, y lo que se busca es una solución con la mejor aptitud y la menor cantidad de contigs, y es aquí donde hace falta saber cómo mapear la aptitud al número de contigs. La secuencia de ADN con la que experimentan pertenece al cromosoma artifical bacteriano *Neurospora crassa*, con número de acceso BX842596, y 77292 pb de longitud; utilizan dos casos obtenidos de esta secuencia, el primero con segmentos de longitud promedio de 708 pb y una cobertura de 4 y el segundo con segmentos de longitud promedio de 703 y una cobertura de 7.

III.2 Nuestro algoritmo genético

En los algoritmos evolutivos debe elegirse una forma adecuada de representar las soluciones del problema, esto es la codificación; otro punto importante es la elección de los operadores que han de aplicarse a los individuos, tanto el de cruzamiento, que es el más importante y el de mutación. Importante también es la función de aptitud que permite calificar a los individuos. De la elección adecuada de estos elementos u operadores depende el desempeño de un algoritmo genético. Antes de pasar al algoritmo propuesto, el Algoritmo 1 muestra un algoritmo genético básico.

```
   Algoritmo 1.  Algoritmo genético básico
-------------------------------------------
  1. Generación de la población inicial
  2. Cálculo de aptitudes
  3. Inicio de ciclo
  4.      Selección
  5.      Cruzamiento
  6.      Mutación
  7.      Cálculo de aptitudes
  8. Termina ciclo
```

Enseguida se muestra el pseudocódigo de un algoritmo genético (Algoritmo 2), suponiendo que ya se han elegido los elementos u operadores antes mencionados; también se señalan en el mismo pseudócodigo los cambios (a un algoritmo básico) propuestos para enfrentar el problema del ensamble.

Algoritmo 2. Algoritmo genético propuesto

```
Entrada:  Conjunto de segmentos de ADN
Salida:  Contig(s) resultante(s) del ensamble
```

1 **Preprocesamiento** { Generación de segmentos complementos inversos; Detección de traslapes; Eliminación de segmentos }

"Inicia etapa genética"

2 Generación de la población inicial

3 Cálculo de aptitudes

4 **Inicia ciclo**

5 Selección

6 Cruzamiento

7 Mutación

8 Cálculo de aptitudes

9 **Después de l iteraciones** { Fase de eliminación de segmentos y ajuste de individuos }

10 **Termina ciclo**

11 Consenso

A continuación se describen en forma detallada cada uno de los componentes del Algoritmo 2.

III.2.1 Etapa de Preprocesamiento

El algoritmo inicia con esta etapa en la cual se hace un preprocesamiento a los segmentos de entrada, dicha etapa está compuesta de tres fases:

Generación de segmentos complementos inversos

Debido a que la orientación de los segmentos se desconoce, éstos no se pueden ensamblar correctamente a menos que todos los segmentos provengan de la misma hebra de ADN, lo cual es poco probable.

El modelo utilizado en este trabajo propone la generación del complemento inverso de cada uno de los segmentos de la entrada, de esta manera se tiene la certeza que existirá la información suficiente para ensamblar los contigs. El inconveniente es que el tamaño del caso se duplica. Posteriormente se explica una propuesta para compensar este incremento.

Detección de traslapes

Esta es la segunda fase de la etapa de preprocesamiento y representa la primera fase del enfoque OLC. Para cada par de segmentos no se busca un simple traslape, la razón son los posibles errores de bases en los segmentos que impedirían lograrlo, por esa razón se busca para cada par de segmentos un alineamiento local, y como se menciona en el capítulo anterior, éste debe ser bajo ciertas condiciones.

Una condición para el alineamiento es que los espacios al inicio o al final de los segmentos no sean penalizados, esto para favorecer el alineamiento del sufijo propio de un segmento a y el prefijo de un segmento b. En la Figura 17a se ve un traslape entre los segmentos a y b, en este caso de cuatro bases. Suponiendo que el segmento

b contiene un error de inserción, el traslape entre los segmentos a y b es menor, tal y como se aprecia en la Figura 17b. Por otro lado, si se busca un alineamiento local entre a y b, en donde los espacios al inicio o al final de los segmentos no se penalicen, se tiene un caso como el que se ilustra en la Figura 17c.

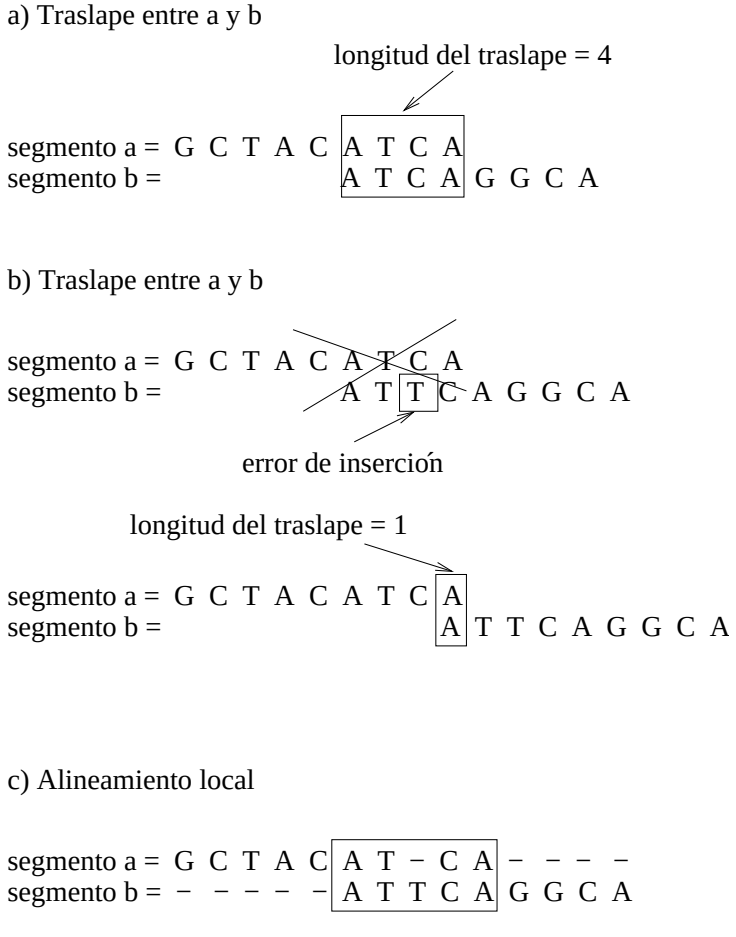


Figura 17: Traslapes y alineamiento local. a) Traslape entre a y b. b) Traslape 1 entre a y b. c) Alineamiento local sin penalización de espacios iniciales y finales

Esta etapa de detección de traslapes tiene como objetivo calificar el "traslape" entre cada par de segmentos, es decir, asignarle un valor numérico; en el Apéndice A se detalla la manera de obtener este valor y las condiciones para ello.

Eliminación de segmentos: sub-segmentos, prefijos o sufijos

El propósito de esta eliminación es reducir la complejidad al algoritmo genético. Como se ha mencionado, utilizamos el enfoque OLC, entonces si en la etapa de detección de traslapes se nota que un segmento $\Pi[i]$ es un sub-segmento, prefijo o sufijo de un segmento $\Pi[j]$, se espera que en la etapa de ordenamiento el segmento $\Pi[j]$ esté contiguo al $\Pi[i]$.

La Figura 18 ilustra un caso para el que se aplica esta eliminación, supóngase que en la etapa de ordenamiento se encontró el orden de segmentos mostrado en la Figura 18a, en donde se aprecia que el segmento $\Pi[1]$ es un sub-segmento del segmento $\Pi[0]$ y también se observa un traslape entre el segmento $\Pi[0]$ y el $\Pi[2]$; en la siguiente etapa del enfoque, en la del consenso, el segmento $\Pi[1]$ prácticamente desaparece, como se observa en la Figura 18b, es decir, el contig mostrado en la Figura 18c puede formarse correctamente con sólo dos de los tres segmentos, lo cual hace al segmento $\Pi[1]$ prescindible. Se propone eliminar dicho segmento en una etapa inicial, en este caso en la de preprocesamiento. La finalidad es reducir el tamaño del problema.

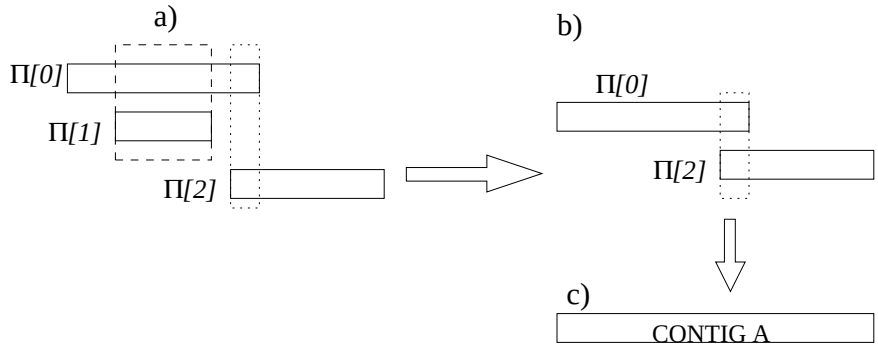


Figura 18: Aplicación de la eliminación de segmentos. a) Etapa de ordenamiento de segmentos. b) Etapa de consenso. c) Contig formado después del consenso.

Cabe mencionar que la detección de sub-segmentos, sufijos o prefijos considera la

existencia de errores de base; en el Apéndice A se detalla este proceso de detección.

III.2.2 Codificación de las estructuras

Para poder generar la población inicial es necesario conocer el tipo de codificación a utilizar, porque esta población debe estar formada por individuos o soluciones válidas para el problema.

Como se menciona en el capítulo anterior, una solución para el problema de ensamble de secuencias es una permutación de los índices de los segmentos, la forma de asignar los índices a los segmentos de la entrada es la siguiente: antes de la generación de segmentos complementos inversos se tienen k segmentos, los cuales se enumeran con los índices $1, 2, .., k$, enseguida se generan para cada segmento i su complemento inverso, al cual se identifica con el índice $i + k$, finalmente se tienen $2k$ segmentos, para fines prácticos se asigna $n = 2k$.

La codificación utilizada es la basada en adyacencia (Grefenstette *et al.*, 1985). Antes de mostrar la codificación por adyacencia se muestra un ejemplo sencillo de una codificación por permutación, la Figura 19a muestra la codificación de una solución, es decir, el genotipo; en la Figura 19b apreciamos la decodificación del cromosoma, es decir, el fenotipo.

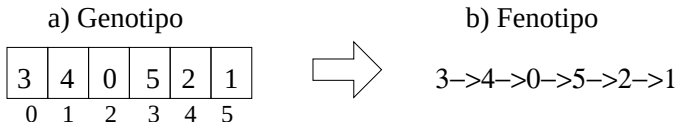


Figura 19: Ejemplo de la codificación por permutación. a) Genotipo. b) Fenotipo

Lo que hace interesante a la codificación por adyacencia es lo siguiente: cuando en el genotipo se tiene un valor i en una posición j significa que el segmento i es sucesor del j en el fenotipo. En la Figura 20 se muestra la codificación por adyacencia de un individuo; en la Figura 20a se muestra el genotipo de un individuo, cuya decodificación se lleva a cabo de la siguiente manera. Primero se elige una posición del genotipo, suponga la posición 3, ésta se convierte en el *primer elemento* del fenotipo (Figura 20b). Para obtener el segundo elemento del fenotipo se debe posicionar en el genotipo, en la posición *"primer elemento"* (en este caso 3) y el valor en esa posición es el segundo elemento del fenotipo, en este caso es el elemento o segmento 5. Ahora debe posicionarse en la posición 5 del genotipo y el valor en esa posición es el siguiente elemento del fenotipo, como lo muestra la Figura 20d, es el segmento 1. Se repite el procedimiento hasta obtener el fenotipo completo, mostrado en la Figura 20e.

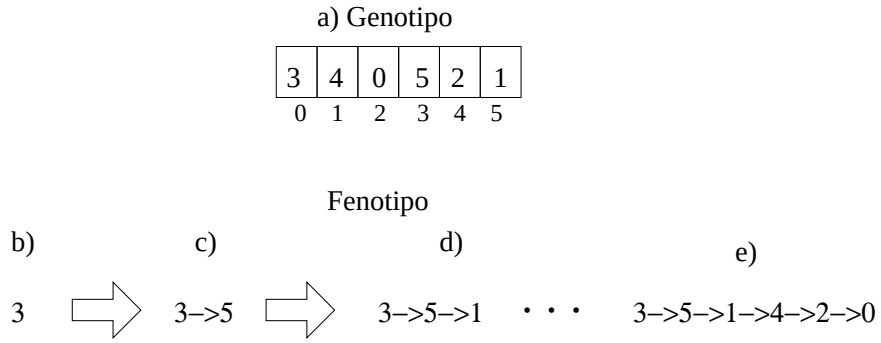


Figura 20: Ejemplo de la codificación por adyacencia

A primera vista no se aprecia la ventaja de la codificación por adyacencia, sin embargo, dicha ventaja se hace notar posteriormente, cuando se explica el operador de cruzamiento. Una vez establecido el tipo de codificación a utilizar, se prosigue a generar la población aleatoriamente, dichos individuos deben representar soluciones válidas.

III.2.3 Función de Aptitud

Se proponen dos funciones de aptitud, las expresiones son:

$$F_1(\Pi) = \sum_{i=0}^{n-2} w(\Pi[i], \Pi[i+1]) + \sum_{i=sm}^{em-2} w(\Pi[i], \Pi[i+1]) \tag{4}$$

y

$$F_2(\Pi) = \frac{\sum_{i=0}^{n-2} w(\Pi[i], \Pi[i+1])}{c[\Pi]} \tag{5}$$

donde,

- Π es una permutación de los índices de los segmentos.
- n es el total de segmentos.
- $\Pi[i]$ devuelve el índice del segmento ubicado en la posición i dentro de la permutación Π.
- $w(\Pi[i], \Pi[i+1])$ es el alineamiento local condicionado entre el segmento $\Pi[i]$ y el $\Pi[i+1]$.
- sm es la posición del primer segmento del contig más largo en la solución dada por la permutación Π.
- em es la posición del último segmento del contig más largo en la solución dada por la permutación Π.
- $c[\Pi]$ es la cantidad de contigs en la solución dada por la permutación Π.

Con la primera función de aptitud se pretende premiar a las soluciones que contienen contigs largos, y al igual que la segunda función de aptitud, se espera que la cantidad

de contigs en las soluciones se reduzca al paso de las generaciones.

Las funciones aquí mostradas se basan en el concepto de Parsons *et al.* (1995), en donde utilizan la suma de los traslapes de segmentos adyacentes; sin embargo, en este caso se propone utilizar el alineamiento local condicionado.

Es importante aclarar la formación de los contigs, en el capítulo anterior se menciona que es una disposición de segmentos contiguos traslapantes; ahora debe decirse que los contigs se forman por segmentos contiguos cuyo puntaje de *alc* es aceptable, es decir, que cumple las condiciones mencionadas en el Apéndice A.

De la definición anterior se puede notar que en una solución dada por una permutación Π, pueden no utilizarse los n segmentos del conjunto P, por lo que:

$$\sum_{j=0}^{c[\Pi]-1} k_j \leq n$$

donde,

- $c[\Pi]$ es la cantidad de contigs en la solución dada por la permutación Π.
- k_j es el número de segmentos en el contig j.

Mecanismo de Selección

Se utiliza la técnica Universal Estocástica (Eiben y Smith, 2003), cuyo objetivo minimizar la mala distribución de los individuos en la población en función de sus valores esperados. Esta técnica es fácil de implementar, además la complejidad de la técnica es $O(n)$. El Algoritmo 3 presenta el pseudocódigo para esta técnica (Coello, 2001).

Algoritmo 3. Selección universal estocástica

```
1  ptr = random(0,1);
2  for (sum=0, i=1; i<=n; i++)
3       for (sum+=Ve(i); sum>ptr; ptr++)
4            seleccionar(i);
```

donde,

- ptr es un número aleatorio entre 0 y 1
- n es el número de individuos
- $Ve(i)$ es el valor esperado del individuo i

La idea general de este mecanismo es la siguiente: en una iteración i se compara la suma acumulada de los valores esperados de los primeros i individuos contra el valor de ptr. Si la suma acumulada de valores es mayor que ptr, entonces el inviduo i es elegido y el valor de ptr se incrementa en 1, el ciclo se repite hasta que no se cumple la condición anterior. La intención es que un individuo i se seleccione tantas veces como su valor esperado.

El valor esperado se calcula de la siguiente manera:

$$Ve(i) = \frac{(aptitud(i) * n)}{F_{\sum}} \tag{6}$$

donde,

$F_{\sum}$ es la sumatoria de las aptitudes de todos los individuos de la población.

III.2.4 Operadores

Cruzamiento

Se utiliza el mismo operador que en González (2004), el cual es una mejora al propuesto por Blazewicz *et al.* (2002), con este operador se produce un solo hijo por cada cruzamiento. Se utiliza este operador voraz porque ha tenido buenos resultados para el problema de SBH (González, 2004), en donde se utiliza la codificación por adyacencia para aprovechar de la mejor manera este operador de cruzamiento. El procedimiento consta de los siguientes pasos (ver Figura 21):

1 Se eligen dos padres aleatoriamente $Padre1$ y $Padre2$ (Figura 21a).

2 Ahora, se escoge aleatoriamente un *locus* y un *alelo* iniciales (Figura 21b). La única condición es que el *alelo* y el *locus* no sean iguales. En el ejemplo de la Figura 21 el *locus* es el 3 y el *alelo* es 1.

3 Se extrae tanto del $Padre1$ como del $Padre2$ el *alelo* ubicado en el *locus* = *alelo* inicial, en este ejemplo, se extraen los alelos 0 y 4 del $Padre1$ y $Padre2$ respectivamente, puesto que son los alelos en el *locus* 1, que es el *alelo* inicial.

4 Enseguida se verifica el *alc* entre el segmento 1 (alelo inicial en el nuevo hijo) y los segmentos 0 y 4 (Figura 21c). Se escoge el segmento que presente el mayor *alc*, siempre y cuando no se genere un ciclo. En este caso es el segmento 4, de esta manera se coloca el alelo 4 en el *locus* 1 (Figura 21d).

5 Se repite el procedimiento, ahora se elige el mejor *alc* entre el segmento 4 y los segmentos a los que corresponden los índices del *locus* 4 del $Padre1$ y $Padre2$. De esta manera se coloca en el *locus* 4 el índice 2, como lo ilustra la Figura 21e.

6 Continuando con la secuencia se busca el mejor sucesor para el segmento 2, en

este caso los segmentos candidatos son 1 y 3, pero ninguno de los dos se puede seleccionar puesto que se genera un ciclo (Figura 21f); el segmento 1 genera un ciclo porque el índice 1 ya ha sido colocado en el cromosoma, y el segmento 3 debe ser el último índice en colocarse en el cromosoma. En estos casos se busca entre los segmentos aún no visitados el segmento que tenga el mejor *alc* con el segmento en cuestión, en este caso con el 2, en el ejemplo sólo puede elegirse el segmento 0 (Figura 21g).

7 Finalmente se completa el cromosoma con el índice = *locus* inicial, en este ejemplo, el 3 (Figura 21h).

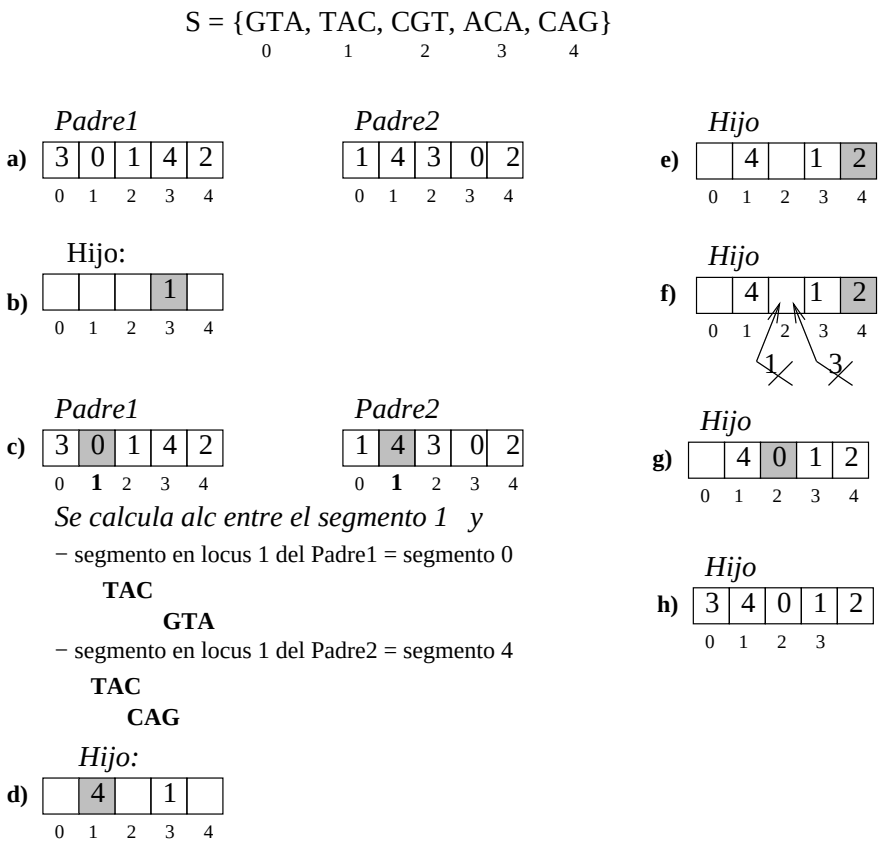


Figura 21: Ejemplo del operador de cruzamiento voraz

Mutación

Se utiliza un operador que Eiben y Smith (2003) llaman de intercambio, el cual es de fácil implementación. Se deja abierta la opción de experimentar con otros operadores, sin embargo por el momento el objetivo no es afinar la elección de un operador de mutación, sino estudiar la inclusión de las etapas propuestas en nuestro algoritmo genético.

Con este operador de mutación se seleccionan dos *loci* aleatorios diferentes de un individuo y se intercambian los alelos de dichos *loci*. Para llevar a cabo correctamente la mutación de un individuo, ésta no la hacemos a nivel genotipo sino a nivel fenotipo, con esto se evita la generación de ciclos en el individuo.

La Figura 22 muestra un ejemplo de la mutación, en la Figura 22a se observa el genotipo de un individuo el cual ha de ser mutado, para ello éste se decodifica, es decir, se obtiene el fenotipo (Figura 22b). Supóngase que se han elegido los *loci* 1 y 3, entonces los alelos 3 y 1 han de intercambiarse, lo que da como resultado el fenotipo de la Figura 22c, el que al ser codificado da lugar al genotipo de la Figura 22d.

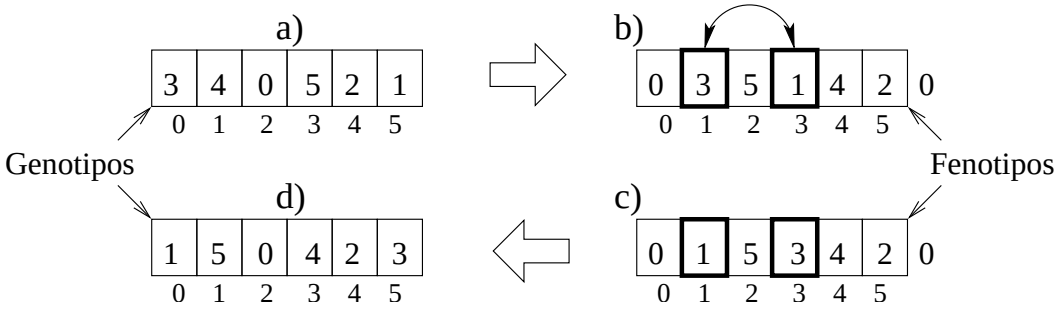


Figura 22: Ejemplo de mutación. a) Genotipo antes de la mutación. b)Fenotipo antes de la mutación. c) Fenotipo después de la mutación. d) Genotipo después de la mutación.

III.2.5 Fase de eliminación de segmentos y ajuste de individuos

Se propone esta fase de eliminación con la intención de contrarrestar la duplicación de la cantidad de segmentos mencionada en el paso 1 del algoritmo genético; el procedimiento propuesto es el siguiente:

Después de l iteraciones, se identifican los contigs del mejor individuo, posteriormente se ordenan de acuerdo a la longitud de éstos y se toma el contig más largo. La intención es eliminar de la entrada a aquellos segmentos cuyos complementos inversos están contenidos en un contig. Para ello se elimina el complemento inverso de cada uno de los segmentos contenidos en ese primer contig, se repite el procedimiento con el siguiente contig más largo, siempre y cuando la cantidad de segmentos considerada hasta ese punto no sobrepase $n/2$, es decir k, que es la cantidad original de segmentos en la entrada. Como se menciona en la sección de codificación, originalmente se tienen k segmentos en la entrada, posteriormente se duplica la cantidad, cuando se generan los segmentos complementos inversos, resultando $2k$, a lo que se denominó n.

La razón por la que no se debe utilizar más de k segmentos es la siguiente, suponiendo que los k segmentos de la entrada se obtienen de la misma hebra de ADN, no hay necesidad de incluir un segmento de la otra hebra para completar el ensamble; ahora bien, suponiendo que de los k segmentos x tienen una orientación invertida, lo que se necesita para tener el ensamble correcto son los complementos inversos de los x segmentos, de esta manera se necesitan únicamente k segmentos.

El objetivo de la etapa de ajuste de individuos es ajustar los cromosomas de todos los

individuos, de tal manera que la cantidad de segmentos considerados por los individuos se reduzca. En la Figura 23 se ilustra el resultado de esta etapa. En la Figura 23a aparece el genotipo de un individuo y suponga que los segmentos 3 y 1 se han eliminado (Figura 23b), el objetivo es ajustar al individuo para que el fenotipo sea como el de la Figura 23c. Tal ajuste debe producir un genotipo libre de ciclos, como el mostrado en la Figura 23d. En ese genotipo se aprecian sombreados los genes que se ignorarán en cualquier operación posterior, para asegurar esto únicamente se necesita conocer el índice de uno de los segmentos que no se ha eliminado e iniciar a partir de ese índice la decodificación de cualquier individuo.

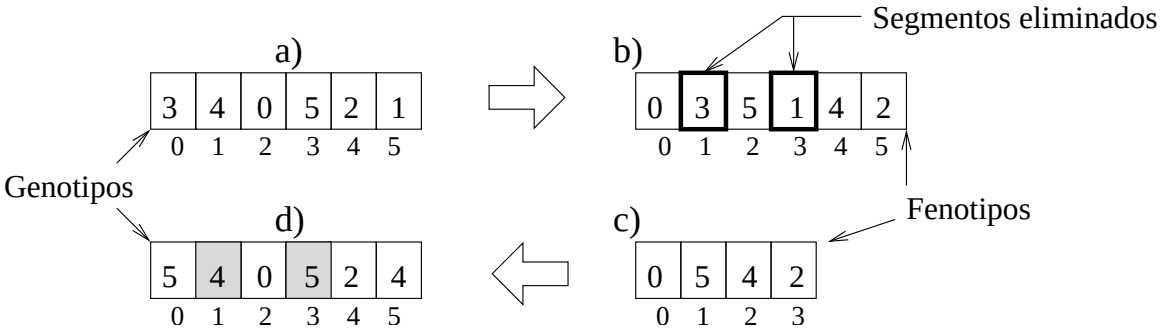


Figura 23: Ajuste de individuos. a) Genotipo antes de eliminación. b) Fenotipo antes de eliminación. c) Fenotipo ajustado después de la eliminación. d) Genotipo ajustado después de la eliminación.

Se propone hacer la fase de eliminación después de l iteraciones porque se espera que a partir de esta iteración, el algoritmo haya generado un contig de tamaño considerable, y entonces no tiene sentido armar el contig con la orientación opuesta a éste. Al eliminar los segmentos complementos inversos el tamaño de la entrada se reduce. El valor de l se elige a prueba y error, este valor no debe ser muy grande para poder aplicar la fase de eliminación cuando aún hay diversidad genética. Tampoco debe ser tan pequeño, porque el contig más largo podría ser un contig sin mucha calidad y podría eliminarse un segmento importante para la formación de un buen contig.

Todas las operaciones contenidas en el ciclo del Algoritmo 2 representan la segunda fase del enfoque OLC, al terminar el ciclo, se espera que el algoritmo haya inferido el orden correcto de los segmentos para reconstruir la cadena original. El siguiente paso del algoritmo y del enfoque es el consenso, descrito a continuación.

III.2.6 Consenso

La finalidad de esta fase es llevar a cabo correctamente la fusión de los segmentos de una solución dada por una permutación Π para formar el(los) contig(s).

Dada una permutación $\Pi = \Pi[1], \Pi[2], ..., \Pi[i], \Pi[i+1], \Pi[i+2], ..., \Pi[n]$, se verifica si existe un *alc* entre el segmento $\Pi[1]$ y el $\Pi[2]$, entre el $\Pi[2]$ y $\Pi[3]$ y asi sucesivamente hasta verificar el segmento $\Pi[n-1]$ con el $\Pi[n]$. Supóngase que existe un *alc* aceptable entre los segmentos $\Pi[i]$ y $\Pi[i+1]$, $\Pi[i+1]$ y $\Pi[i+2]$, y entre el segmento $\Pi[i+2]$ y el $\Pi[i+3]$ no existe un *alc* aceptable, lo cual indica que los segmentos $\Pi[i]$, $\Pi[i+1]$ y $\Pi[i+2]$ (Figura 24a) deben fusionarse para formar un contig, el procedimiento se lleva a cabo de la siguiente manera.

El segmento $\Pi[i]$ se convierte en la parte inicial del contig, como se aprecia en la Figura 24b; enseguida se concatena al contig la subsecuencia del segmento $\Pi[i+1]$ que no es incluida en el alineamiento entre los segmentos $\Pi[i]$ y $\Pi[i+1]$, suponiendo que el alineamiento es el mostrado en la Figura 24c, entonces la subsecuencia que se concatena es C C G A T T, como se muestra en la Figura 24d. Finalmente, se concatena al contig en formación la subsecuencia que no fue incluida en el alineamiento entre los segmentos $\Pi[i+1]$ y $\Pi[i+2]$, como se aprecia en la Figura 24e, es la subsecuencia C G A T. El resultado es el contig mostrado en la Figura 24f.

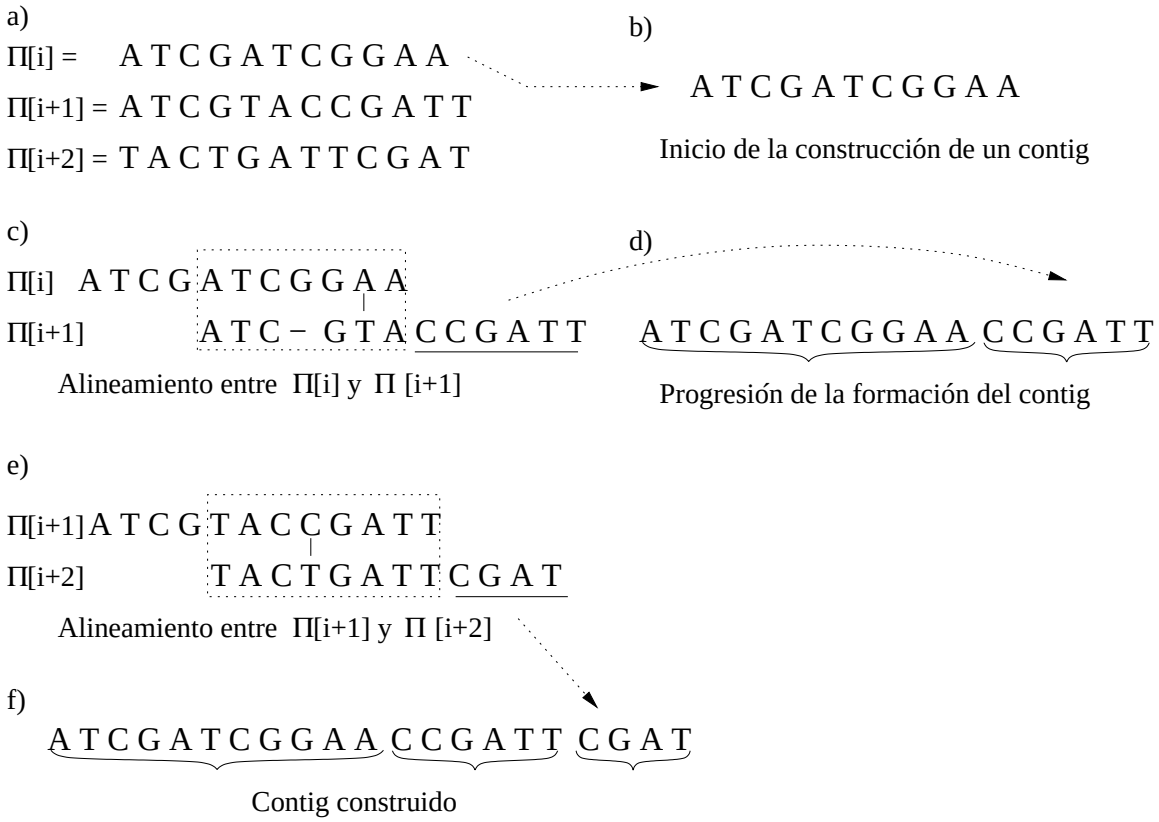


Figura 24: Consenso. a) Segmentos traslapantes. b) Inicio de la contrucción de un contig. c) Alineamiento de los semgentos $\Pi[i]$ y $\Pi[i+1]$. d) Progresión de la construcción de un contig. e) Alineamiento de los segmentos $\Pi[i+1]$ y $\Pi[i+2]$. f) Contig construido.

Capítulo IV

Experimentos computacionales y resultados

En este capítulo se presenta el modelo usado para la generación de casos de entrada, se explican los criterios de calidad de una solución, también se muestran las secuencias utilizadas para generar los casos de entrada, y se listan los algoritmos utilizados para los diferentes experimentos. Posteriormente se compara una función de aptitud propuesta por Parsons *et al.* (1995) con una propuesta en este trabajo y se comparan los resultados generados por el algoritmo genético con los generados por TGI, programa disponible públicamente.

IV.1 Generación de casos

En el Capítulo II se mencionan dos analogías del experimento *shotgun*, mediante el cual se producen los segmentos de ADN que sirven de entrada al problema del ensamble de secuencias. Sin embargo, el modelo descrito en el trabajo de Engle y Burks (1993) es el más aceptado para la generación de casos (Parsons *et al.*, 1995), (Luque *et al.*, 2005), (Pop *et al.*, 2002).

IV.1.1 Engle y Burks (1993)

Engle y Burks (1993) desarrollaron un conjunto de herramientas llamado *genfrag* para generar casos de entrada que sirven para probar algoritmos para el ensamble de secuencias de ADN. Aunque en este trabajo no se usa *genfrag*, si se usa el modelo de

Engle y Burks (1993). Este conjunto de herramientas consta de ciertos parámetros que lo hacen flexible y permiten estudiar el desempeño de los algoritmos frente a diversas características de los datos de entrada. Los parámetros de entrada que se manejan son:

- Longitud media de los segmentos.
- Porcentaje de intervalo de variación de longitud, con respecto a la longitud media.
- Cobertura: Puede decirse que es una medida de la redudancia de datos, o el número de veces que puede cubrirse la longitud de la secuencia original, dado los segmentos de la entrada. La expresión para la cobertura es:

$$C = \frac{n * l}{L} \tag{7}$$

  donde,
  C = Cobertura
  n = número de segmentos.
  l = longitud media de los segmentos.
  L = longitud de la cadena original.

- Errores. En cuanto a los errores, este modelo comprende un parámetro para indicar el porcentaje de error, y también puede elegirse el modo de error, bien sea sustituciones y/o inserciones/eliminaciones (*indels*).

Para la introducción de errores, cada posición de cada uno de los segmentos se considera independientemente, dada una probabilidad de error, si una posición se elige para introducir un error, se elige el modo y se lleva a cabo la operación dada en esa posición.

En cuanto a la orientación de los segmentos, Engle y Burks (1993) mencionan que con el 0.5 de probabilidad se selecciona el segmento con la orientación normal o el complemento inverso de éste.

IV.1.2 Generador de casos

Para este trabajo se ha implementado un generador de casos que recibe los mismos parámetros que el modelo descrito anteriormente, tales casos se forman como se explica a continuación.

Considérese la Ecuación 7 y los parámetros: longitud media y cobertura, y dado que para fines experimentales se conoce la longitud de la cadena original, se prosigue a calcular la cantidad de segmentos (n) a extraer para un caso determinado. En tal caso de la Ecuación 7 se tiene:

$$n = \frac{C * L}{l} \tag{8}$$

Estos n segmentos se extraen de la siguiente manera:

1 Se elige una posición aleatoria[18] inicial dentro de la cadena original, tal y como se muestra en la Figura 25a.

2 Se elige aleatoriamente el tamaño del segmento. Este segmento debe estar dentro del intervalo de variación de longitud. En la Figura 25b se observa la zona de donde se extrae el segmento.

Supóngase ahora la extracción de otro segmento. Primero se coloca el punto inicial

[18]En este trabajo todas las variables aleatorias tienen una distribución uniforme

y enseguida se determina el tamaño del segmento. La Figura 25c muestra un caso en el que el punto inicial está cerca del extremo de la cadena original, y por esa razón no puede extraerse un segmento del tamaño que se ha elegido, en estos casos, se repite el procedimiento de extracción.

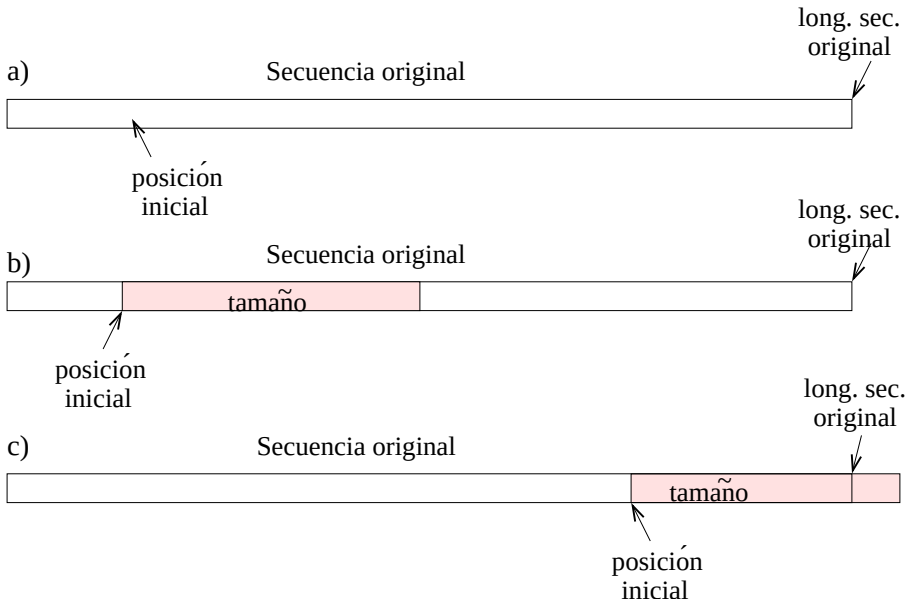


Figura 25: Extracción de segmentos. a) Posición inicial de un segmento. b) Tamaño de un segmento. c) Extracción fallida de un segmento

Al igual que en el modelo de Engle y Burks (1993), una vez extraído un segmento, con 0.5 de probabilidad se toma el segmento con la orientación normal o el complemento inverso del mismo. Los errores se introducen tal y como se describió en la sección anterior.

IV.2 Criterios para medir la calidad de una solución

Dada la naturaleza del problema del ensamble, puede pensarse que el resultado generado por un algoritmo no es forzosamente una sola secuencia, sino un conjunto de

subsecuencias o contigs (ver Capítulo II), pero independientemente de la cantidad de contigs devueltos por un algoritmo, debe establecerse un criterio adecuado para medir la calidad de una solución.

En este trabajo se utilizan dos criterios de calidad: la similitud global y la utilización, sin menospreciar otros datos que también son objeto de atención, como son la cantidad, longitud y calidad de los contigs, tiempo de ejecución y convergencia, pero en primer lugar se describen los dos criterios mencionados al inicio.

IV.2.1 Similitud global

La similitud global se mide buscando coincidencia exacta entre la cadena original y cada uno de los contigs de una solución dada. La Figura 26a muestra una solución que consta de tres contigs, entonces se busca la coincidencia exacta de cada uno de los contigs con la cadena original, señalando para cada contig la zona de la cadena original cubierta. Como se aprecia en la Figura 26b, los tres contigs del ejemplo cubren dos zonas de la cadena original; finalmente se obtiene un valor numérico para la similitud global mediante la siguiente expresión:

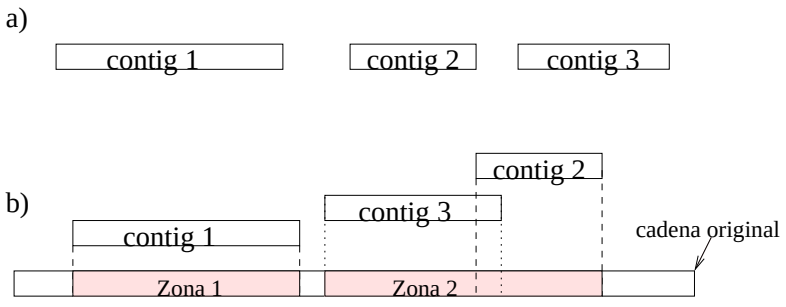


Figura 26: Criterio denominado similitud global

$$\%similitud = \frac{zona\ cubierta}{n} \tag{9}$$

donde,

zona cubierta = zona cubierta por los contigs, en este caso la zona sombreada.

n = longitud de la cadena original.

IV.2.2 Utilización

El criterio de utilización está asociado a la similitud global, porque indica la cantidad de información que fue requerida para lograr un porcentaje de similitud determinado. Este criterio está dado por:

$$\%utilizacion = \frac{zona\ cubierta}{\sum_{i=1}^{k} l_i} = \frac{\%similitud * n}{\sum_{i=1}^{k} l_i} \tag{10}$$

donde,

zona cubierta = zona sombreada (ver Figura 26b).

k = cantidad de contigs.

l_i = longitud del contig i.

n = longitud de la cadena original.

En cierta medida una utilización baja nos puede indicar que existe traslape entre los contigs de la solución, lo que significa que ésta se pudo haber mejorado, traslapando contigs y reduciendo así la cantidad de ellos; también puede indicar que los contigs fueron mal construidos, lo cual se podría verificar con la calidad de los contigs, criterio que se explica en la siguiente sección.

Tanto en la *similitud* como en la *utilización*, se ha mencionado que se busca coincidencia exacta entre la cadena original y los contigs de una solución para determinar la calidad de ésta; sin embargo, con esto solamente se están considerando los casos en los que no existen errores de bases (ver Capítulo II), en esos casos la coincidencia debería ser exacta, siempre y cuando los contigs estén armados correctamente. Ahora bien, para los casos en los que existen errores de bases, no se busca una coincidencia exacta, puesto que la presencia de errores de bases impediría lograrlo, lo que se busca es un alineamiento local entre la secuencia original y cada uno de los contigs.

Entonces tanto la Ecuación 9 como la 10 cambian el valor *zona cubierta* por *número de coincidencias en la zona cubierta*.

Para evaluar la calidad de las soluciones bajo estos dos criterios nos apoyamos en el paquete *seqaln* descrito en el Apéndice A.

IV.2.3 Contigs

Aunque los dos criterios de calidad anteriores son los principales, se ha utilizado un criterio referente a los contigs, dentro del cual se encuentran tres medidas:

1 Cantidad. Se refiere a la cantidad de contigs generados en una solución, esperando el menor número de ellos.

2 Longitud. Se refiere a la longitud del contig más largo, se espera la mayor longitud posible.

3 Calidad. En este caso, se calcula la calidad promedio de los contigs generados en una solución. Para calcular la calidad de un contig se busca un alineamiento

local entre éste y la cadena original, y con los datos del alineamiento se calcula la calidad mediante la siguiente ecuación:

$$calidad = \frac{\text{núm. de coincidencias}}{\text{longitud del contig}} \tag{11}$$

Finalmente, el último criterio que se considera en este trabajo es el tiempo de cómputo requerido para encontrar la mejor solución.

IV.2.4 Convergencia

Se señala la generación o iteración en que fue encontrada la mejor solución. Hay que aclarar, sin embargo, que para algoritmos estocásticos basados en población (como este caso), la convergencia se refiere a cuando el multiconjunto de soluciones llega a su mínima variabilidad y ésta se mantiene con las iteraciones.

IV.3 Secuencias, casos y parámetros de prueba

Las secuencias utilizadas para generar los casos de entrada se muestran en la Tabla IV. Se han seleccionado estas secuencias porque se han probado con otros algoritmos, lo que da un punto de referencia.

Tabla IV: Secuencias utilizadas para los experimentos.

Nombre	Número de acceso	Longitud (pb)
HUMAPOBF	M15421	10089
AMCG	J02459	48502

El número de acceso al que se refiere la Tabla IV es el número con el que se identifica a la secuencia dentro del Genbank[19], sitio público de donde puede obtenerse esa secuencia. De las secuencias mostradas en la Tabla IV se derivaron las subsecuencias mostradas en la Tabla V.

Tabla V: Subsecuencias utilizadas para los experimentos.

Subsecuencia	Obtenida de la secuencia	Longitud de la secuencia original (pb)	Longitud de la subsecuencia (pb)
HUMA_10	HUMAPOBF	10089	10089
AMCG_10	AMCG	48502	10089
AMCG_20	AMCG	48502	20000

La Tabla VI muestra los casos generados a partir de las subsecuencias listadas en la Tabla V, también se muestran los parámetros para cada uno de estos casos.

Tabla VI: Casos generados para los experimentos.

Nombre del caso	Obtenido de la subsecuencia	Probabilidad de errores de base	Tipo de error
HUMA_10_SER	HUMA_10	0	Ninguno
HUMA_10_2ER	HUMA_10	0.02	Sustituciones e indels
HUMA_10_5ER	HUMA_10	0.05	Sustituciones e indels
AMCG_10_SER	AMCG_10	0	Ninguno
AMCG_10_2ER	AMCG_10	0.02	Sustituciones e indels
AMCG_10_5ER	AMCG_10	0.05	Sustituciones e indels
AMCG_20_2ER	AMCG_20	0.02	Sustituciones e indels
AMCG_20_5ER	AMCG_20	0.05	Sustituciones e indels

Todos los casos mostrados en la Tabla VI tienen en común la información mostrada en la Tabla VII, además cada uno de estos casos contiene segmentos cuya orientación es desconocida.

[19]http://www.ncbi.nlm.nih.gov, noviembre-2004

Tabla VII: Casos generados (parámetros en común).

Longitud media de los segmentos	% de intervalo de variación de longitud	Cobertura
300	25	5

Dentro de la Tabla VI, el parámetro *probabilidad de errores de base* indica la probabilidad de errores de bases introducidos en cada uno de los segmentos del caso, y en la siguiente columna se muestran los posibles tipos de error dentro de los segmentos.

Los errores de base se han clasificado en dos tipos, errores de sustitución e *indels*, y durante el proceso de introducción de errores, cada posición de un segmento se considera independientemente para introducir un error; supóngase el segmento de la Figura 27a en donde la posición 2 se ha elegido para introducir un error, ahora la pregunta es qué tipo de error ha de introducirse. Enseguida se muestran tres posibles tipos de error:

- Sólo sustituciones: La letra de la posición elegida se sustituye por una letra (A,C,T o G) elegida aleatoriamente. En la Figura 27b se representa con el caracter X la letra aleatoria que ha de ir en esa posición.

- Sólo *indels*: En este caso, con 0.5 de probabilidad se elige entre una inserción y una eliminación. Supóngase que se ha elegido una inserción, primeramente se elige una letra aleatoriamente y ésta se inserta una posición después de la posición elegida, como lo ilustra la Figura 27c. En el caso de una eliminación, simplemente se remueve la letra de la posición elegida, como se aprecia en la Figura 27d.

- Sustituciones e *indels*: Cuando se han elegido ambos tipos de error, con 0.5 de probabilidad se introduce un error de sustitución o un *indel*, y como se menciona en el punto anterior, en el caso de un error *indel*, con 0.5 de probabilidad se escoge

una inserción o una eliminación.

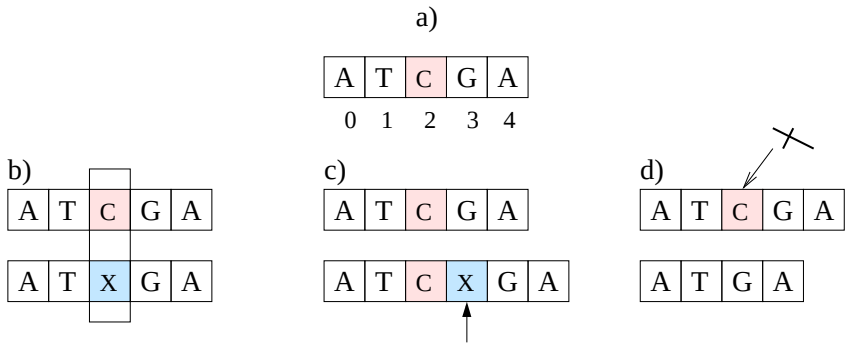


Figura 27: Introducción de error a un segmento. a) Segmento al que se le introducirá un error. b) Sustitución. c) Inserción. d) Eliminación

IV.4 Algoritmos

Antes de describir los resultados de los experimentos, se muestra en la Figura 28 la nomenclatura de los algoritmos utilizados:

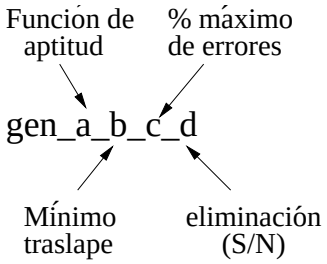


Figura 28: Nomenclatura empleada para describir los algoritmos

El primer parámetro (a) indica la función de aptitud usada, en este trabajo se ha experimentado con dos funciones de aptitud, la primera de ellas es la propuesta por Parsons *et al.* (1995) y que también se usó en Luque *et al.* (2005), a la cual se identifica

con el número 4 y con el número 5 se identifica a la función de aptitud propuesta en este trabajo.

El segundo parámetro (b) indica el traslape mínimo que deben tener dos segmentos para que sea considerado un traslape aceptable. El tercer parámetro (c) indica la cantidad de errores permitidos por cada 100 bases, este margen de error se utiliza para determinar si un traslape es aceptable o no; dado el parámetro c, se establece el número de errores permitidos dentro de un traslape mediante la expresión siguiente:

$$errores\ permitidos = \frac{longitud\ del\ traslape * \%\ máximo\ de\ errorres}{100} \qquad (12)$$

El cuarto parámetro (d) indica si el algoritmo contiene o no la fase de eliminación de segmentos y ajuste de individuos. Para los algoritmos que utilizan la fase de eliminación antes mencionada, ésta se lleva a cabo en las iteraciones 15, 30 y 45.

Todos los algoritmos utilizados en los experimentos tienen en común la información mostrada en la Tabla VIII.

Tabla VIII: Algoritmos utilizados (parámetros en común).

Población	Generaciones	Probabilidad de cruzamiento	Probabilidad de mutación
100	100	1.0	0.01

IV.5 Resultados (subsecuencias de 10,000 pb)

En esta sección se analizan los resultados de experimentos realizados sobre casos obtenidos de subsecuencias de 10,000 pares de bases; primero casos sin errores de bases, enseguida casos con 2% y 5% de errores de bases.

Cabe mencionar que los datos mostrados en todas las tablas corresponden al promedio de 30 corridas de 30 casos diferentes de una secuencia dada; por ejemplo, el promedio de similitud se obtiene promediando los promedios de los 30 casos, donde el promedio de un caso se obtiene sobre 30 corridas. La desviación estándar de similitud es sobre los promedios de similitud de los 30 casos. La cantidad, longitud (del contig más largo) y calidad de contigs, utilización, tiempo, convergencia promedio y aptitud promedio, se calculan de la misma manera que el promedio de similitud; y la desviación estándar de convergencia y la de aptitud se calculan de la misma forma que la desviación estándar de similitud.

En cuanto al tiempo se debe resaltar que para el caso de los algoritmos genéticos, éstos leen los traslapes desde una matriz previamente calculada; el tiempo *ct* (variable para cada caso) no se ha sumado al que se muestra en las tablas. Esto obedece a que se utiliza el módulo de *seqaln*, el cual no está integrado como parte del código del algoritmo genético, entonces el cálculo de traslapes para cada par de segmentos se realiza desde la línea de comandos del sistema operativo empleado. Lo anterior significa que no se pueden comparar los tiempos entre los algoritmos genéticos y TGI, pero si entre los algoritmos genéticos.

IV.5.1 Error de la orientación desconocida

En esta sección se analiza el desempeño de los algoritmos con casos que únicamente cuentan con el error de la orientación desconocida. El primer propósito es ver la diferencia entre usar o no la fase de eliminación y ajuste que se está proponiendo, el segundo propósito es ver si existe diferencia significativa entre el desempeño de las dos funciones de aptitud. Los datos mostrados en la tablas IX y X son de experimentos realizados sobre el caso HUMA_10_SER.

Tabla IX: Comparación de los algoritmos genéticos y TGI sobre el caso HUMA_10_SER. Similitud, contigs, utilización y tiempo. Error de orientación únicamente.

	Similitud		Contigs				
Algoritmos	Promedio	Desv. est.	Cantidad de contigs	Long. del contig más largo	% de Calidad	% de util.[20]	Tiempo seg/caso
gen_4_15_0_N	91.69	7.58	2.43	7435.20	100	86.19	$3.19 + ct$
gen_5_15_0_N	90.19	7.78	2.27	7555.87	100	87.22	$3.11 + ct$
gen_4_15_0_S	97.63	1.73	2.20	7432.60	100	98.88	$1.41 + ct$
gen_5_15_0_S	96.58	2.13	1.90	7516.07	100	99.40	$2.73 + ct$
TGI	98.11	1.33	2.43	6932.77	100	99.85	5.0

Tabla X: Comparación de los algoritmos genéticos sobre el caso HUMA_10_SER. Convergencia y aptitud. Error de orientación únicamente.

	Convergencia		Apt. del mejor ind.		Apt. Final del mejor ind.	
Algoritmo	Promedio	Desv. Est.	Promedio	Desv. Est.	Promedio	Desv. Est.
gen_4_15_0_N	35.77	3.86	52763.40	3511.55	25414.20	2321.96
gen_5_15_0_N	35.10	5.34	11520.00	6613.98	5205.57	2983.87
gen_4_15_0_S	20.20	5.24	26701.13	1734.95	24659.17	2178.23
gen_5_15_0_S	47.77	4.92	15016.13	9122.04	3882.23	1800.72

Los primeros dos renglones de las tablas IX y X muestran dos algoritmos genéticos que no incluyen la fase de eliminación de segmentos y de ajuste de individuos, los siguientes dos renglones muestran algoritmos que si utilizan esta fase.

[20] utilización

En la Tabla IX se aprecia una diferencia significativa a favor de los algoritmos que utilizan la fase de elimininación de segmentos y de ajuste de individuos, tal diferencia es clara en cuanto a la similitud y la utilización, que son los criterios principales, también en cuanto al tiempo se nota una pequeña ventaja a favor de los algoritmos que incluyen la fase mencionada.

En cuanto al desempeño de las funciones de aptitud, la única diferencia significativa es que la convergencia es más rápida cuando se utiliza la primera función de aptitud; lo anterior únicamente entre los algoritmos que utilizan la fase de eliminación propuesta. Aunque en los criterios principales no existe diferencia significativa entre el uso de una u otra función de aptitud, se utilizan ambas en los demás experimentos para ver el desempeño de éstas en diversos casos, con diferentes secuencias y porcentajes de error.

Finalmente, se analiza el desempeño de los algoritmos genéticos contra TGI. En la Tabla IX se observa un mejor resultado del programa TGI en cuanto a la similitud y al porcentaje de utilización, sin embargo, en cuanto a la cantidad y longitud de contigs es el algoritmo genético el que obtiene una ligera ventaja. La convergencia no se muestra para TGI, puesto que éste utiliza un método determinístico.

Para el caso AMCG_10_SER, los resultados son los que se muestran en las tablas XI y XII. Los resultados mostrados en la Tabla XI muestran claramente la ventaja de usar la fase de eliminación de segmentos y ajuste de individuos, tanto en similitud como en utilización, sin menospreciar que el tiempo de cómputo se reduce considerablemente. Comparando los resultados de los dos últimos algoritmos contra los que genera el programa TGI, no se aprecia una diferencia significativa.

Tabla XI: Comparación de los algoritmos genéticos y TGI sobre el caso AMCG_10_SER. Similitud, contigs, utilización y tiempo. Error de orientación únicamente.

	Similitud		Contigs				
Algoritmos	Promedio	Desv. est.	Cantidad de contigs	Long. del contig más largo	% de Calidad	% de util.	Tiempo seg/caso
gen_4_15_0_N	89.52	9.53	2.63	6958.40	100	83.43	4.70 + ct
gen_5_15_0_N	88.68	10.88	2.43	7064.00	100	83.27	4.05 + ct
gen_4_15_0_S	97.44	1.61	2.47	6964.27	100	98.96	1.58 + ct
gen_5_15_0_S	96.19	1.89	2.20	7034.53	100	99.52	2.51 + ct
TGI	97.88	1.31	2.80	6596.10	100	99.84	4.20

Tabla XII: Comparación de los algoritmos genéticos sobre el caso AMCG_10_SER. Convergencia y aptitud. Error de orientación únicamente.

	Convergencia		Apt. del mejor ind.		Apt. Final del mejor ind.	
Algoritmo	Promedio	Desv. Est.	Promedio	Desv. Est.	Promedio	Desv. Est.
gen_4_15_0_N	36.37	3.98	51275.20	2991.23	24938.37	1189.14
gen_5_15_0_N	35.33	5.77	9976.03	6092.41	5236.80	3554.13
gen_4_15_0_S	19.17	5.53	25955.50	1446.25	24205.33	1685.30
gen_5_15_0_S	43.70	10.96	13024.40	8504.84	3319.67	1118.03

IV.5.2 Casos con 2% de errores de base

En esta sección se analiza el desempeño de los algoritmos en casos con 2% de errores de bases, tanto de sustituciones como *indels*.

En la tablas XIII y XIV se muestran los resultados de los algoritmos genéticos sobre el caso HUMA_10_2ER, con la finalidad de verificar la conveniencia de la mencionada fase de eliminación y ajuste, y también para comparar los resultados de los algoritmos genéticos contra los resultados de TGI.

Los resultados siguen favoreciendo a los algoritmos que incluyen la fase de eliminación de segmentos y ajuste de individuos, tanto en similitud como en utilización y tiempo, como se aprecia en la Tabla XIII. Comparando las dos funciones de aptitud, no

Tabla XIII: Comparación de los algoritmos genéticos y TGI sobre el caso HUMA_10_2ER. Similitud, contigs, utilización y tiempo. Error de orientación y 2% de errores de bases.

	Similitud		Contigs				
Algoritmos	Promedio	Desv. est.	Cantidad de contigs	Long. del contig más largo	% de Calidad	% de util.	Tiempo seg/caso
gen_4_15_8_N	90.78	5.83	3.70	6685.87	98.80	74.88	$4.04 + ct$
gen_5_15_8_N	91.16	5.85	3.20	6733.93	98.79	76.62	$4.85 + ct$
gen_4_15_8_S	96.30	1.85	2.87	6637.87	98.79	97.59	$2.04 + ct$
gen_5_15_8_S	95.03	2.21	2.50	6671.10	98.79	98.17	$3.02 + ct$
TGI	97.11	1.84	4.87	4730.57	99.50	96.76	4.20

Tabla XIV: Comparación de los algoritmos genéticos sobre el caso HUMA_10_2ER. Convergencia y aptitud. Error de orientación y 2% de errores de bases.

	Convergencia		Apt. del mejor ind.		Apt. Final del mejor ind.	
Algoritmo	Promedio	Desv. Est.	Promedio	Desv. Est.	Promedio	Desv. Est.
gen_4_15_8_N	36.90	4.45	52339.07	4083.10	25699.63	2495.57
gen_5_15_8_N	34.63	4.31	9083.23	5788.10	4727.93	3091.58
gen_4_15_8_S	19.90	6.12	26535.80	1986.46	24777.40	1991.05
gen_5_15_8_S	43.53	8.14	10908.37	7196.55	3440.97	1711.42

se encuentra diferencia significativa en ninguno de los criterios; excepto si se comparan los dos algoritmos que incluyen la fase propuesta, en cuanto al tiempo la función de aptitud número 4 presenta el mejor, siendo la convergencia el factor que produce tal resultado; como se ve en la Tabla XIV, la función de aptitud de Parsons *et al.* (1995) es la que converge más rápido.

Los resultados de la Tabla XIII permiten comparar los resultados de los algoritmos genéticos con la fase propuesta y TGI, en esa tabla se aprecia una ligera ventaja en la columna de similitud a favor de TGI; sin embargo, en la columna de contigs, tanto en cantidad como en longitud, los algoritmos genéticos muestran una clara superioridad sobre TGI. En cuanto a la utilización, no se observan diferencias significativas.

Las tablas XV y XVI muestran una comparación de los algoritmos genéticos y TGI para el caso AMCG_10_2ER. En dichas tablas se aprecian resultados similares a los de las tablas XIII y XIV, es decir, una diferencia a favor de los algoritmos que incluyen la fase propuesta. También se aprecia una pequeña ventaja de TGI (sobre los algoritmos genéticos) en cuanto a similitud, pero una ventaja considerable a favor de los algoritmos genéticos (con la fase propuesta) en cuanto a contigs.

Tabla XV: Comparación de los algoritmos genéticos y TGI sobre el caso AMCG_10_2ER. Similitud, contigs, utilización y tiempo. Error de orientación y 2% de errores de bases.

	Similitud		Contigs				
Algoritmos	Promedio	Desv. est.	Cantidad de contigs	Long. del contig más largo	% de Calidad	% de util.	Tiempo seg/caso
gen_4_15_8_N	90.93	6.99	3.17	7123.07	98.87	77.06	$4.20 + ct$
gen_5_15_8_N	90.85	7.90	2.97	7189.90	98.87	77.13	$4.59 + ct$
gen_4_15_8_S	96.25	2.03	2.57	7041.67	98.84	97.69	$1.84 + ct$
gen_5_15_8_S	95.07	2.19	2.23	7149.90	98.85	98.36	$3.25 + ct$
TGI	97.77	1.49	4.67	5257.77	99.47	97.24	7.96

Tabla XVI: Comparación de los algoritmos genéticos sobre el caso AMCG_10_2ER. Convergencia y aptitud. Error de orientación y 2% de errores de bases.

	Convergencia		Apt. del mejor ind.		Apt. Final del mejor ind.	
Algoritmo	Promedio	Desv. Est.	Promedio	Desv. Est.	Promedio	Desv. Est.
gen_4_15_8_N	36.73	3.73	52797.43	3036.23	25337.83	1863.91
gen_5_15_8_N	33.63	3.80	9456.17	5296.07	5298.93	3600.63
gen_4_15_8_S	21.53	5.41	26726.13	1540.33	24981.47	1558.46
gen_5_15_8_S	46.33	7.25	12597.83	8219.61	3478.77	1216.78

IV.5.3 Casos con 5% de errores de base

En esta sección se analiza el resultado de los algoritmos genéticos con casos que contienen 5% de errores de bases. En la tablas XVII y XVIII se muestran los resultados sobre el caso HUMA_10_5ER, primero para comparar los algoritmos genéticos que incluyen la fase ya mencionada, y también para comparar los resultados contra los de TGI.

En la Tabla XVII se nota el mejor desempeño de los algoritmos con la fase de eliminación y ajuste, tanto en similitud como en utilización. Comparando los resultados contra los que genera TGI, se nota una amplia diferencia a favor de los algoritmos genéticos, esta ventaja se aprecia en cada uno de los criterios: similitud, contigs y utilización. Entre las funciones de aptitud 4 y 5 no se aprecia una diferencia significativa, más que en el tiempo (Tabla XVII) y convergencia (Tabla XVIII).

Tabla XVII: Comparación de los algoritmos genéticos y TGI sobre el caso HUMA_10_5ER. Similitud, contigs, utilización y tiempo. Error de orientación y 5% de errores de bases.

	Similitud		Contigs				
Algoritmos	Promedio	Desv. est.	Cantidad de contigs	Long. del contig más largo	% de Calidad	% de util.	Tiempo seg/caso
gen_4_30_15_N	89.32	7.43	3.13	6811.03	96.96	73.21	4.00
gen_5_30_15_N	89.79	7.15	2.87	6931.33	96.97	78.16	3.87
gen_4_30_15_S	94.52	1.90	2.57	6793.73	96.97	95.71	1.78
gen_5_30_15_S	93.28	2.06	2.20	6862.50	96.97	96.35	3.00
TGI	82.05	5.69	26.57	1030.53	97.08	57.54	11.26

Tabla XVIII: Comparación de los algoritmos genéticos sobre el caso HUMA_10_5ER. Convergencia y aptitud. Error de orientación y 5% de errores de bases.

	Convergencia		Apt. del mejor ind.		Apt. Final del mejor ind.	
Algoritmo	Promedio	Desv. Est.	Promedio	Desv. Est.	Promedio	Desv. Est.
gen_4_15_8_N	36.43	3.78	52601.33	3037.81	25892.47	1851.13
gen_5_15_8_N	35.40	4.60	10044.27	5891.03	4988.17	2810.98
gen_4_15_8_S	22.47	6.09	26640.80	1473.74	24598.13	1880.73
gen_5_15_8_S	44.17	11.95	12751.07	8204.43	4011.93	4498.29

Las tablas XIX y XX muestran resultados de los algoritmos genéticos y TGI para el caso AMCG_10_5ER. De la misma manera que en las tablas XVII y XVIII, se nota una diferencia considerable a favor de los algoritmos genéticos que incluyen la fase de eliminación y ajuste propuesta. En cuanto a las funciones de aptitud, no se aprecia diferencia significativa entre el uso de una u otra. Finalmente, los resultados de la Tabla

XIX muestran que los algoritmos genéticos (con la fase propuesta) generan mejores resultados que TGI, en similitud, contigs y utilización.

Tabla XIX: Comparación de los algoritmos genéticos y TGI sobre el caso AMCG_10_5ER. Similitud, contigs, utilización y tiempo. Error de orientación y 5% de errores de bases.

	Similitud		Contigs				
Algoritmos	Promedio	Desv. est.	Cantidad de contigs	Long. del contig más largo	% de Calidad	% de util.	Tiempo seg/caso
gen_4_30_15_N	89.92	4.68	3.63	6170.47	97.23	73.00	4.00
gen_5_30_15_N	89.08	6.08	3.13	6331.07	97.20	74.44	3.84
gen_4_30_15_S	94.61	1.91	2.73	6163.87	97.22	95.78	1.72
gen_5_30_15_S	93.21	2.40	2.40	6335.50	97.22	96.63	2.86
TGI	82.05	5.69	26.57	1030.53	97.08	57.54	11.26

Tabla XX: Comparación de los algoritmos genéticos sobre el caso AMCG_10_5ER. Convergencia y aptitud. Error de orientación y 5% de errores de bases.

	Convergencia		Apt. del mejor ind.		Apt. Final del mejor ind.	
Algoritmo	Promedio	Desv. Est.	Promedio	Desv. Est.	Promedio	Desv. Est.
gen_4_15_8_N	36.50	3.98	52423.83	2820.64	25603.67	3259.71
gen_5_15_8_N	35.33	4.35	9009.27	5284.41	4296.03	2209.82
gen_4_15_8_S	21.10	7.19	26579.50	1391.78	24759.57	1711.50
gen_5_15_8_S	42.37	11.54	11597.60	7950.81	3334.77	1290.47

IV.6 Resultados (subsecuencias de 20,000 pb)

En esta sección se analizan resultados de experimentos realizados sobre casos obtenidos de subsecuencias de 20,000 pares de bases y con 2 y 5% de errores de bases.

IV.6.1 Casos con 2% de errores de base

En la Tabla XXI se muestran los resultados sobre el caso AMCG_20_2ER, tanto para los algoritmos genéticos como para TGI.

Tabla XXI: Comparación de los algoritmos genéticos y TGI sobre el caso AMCG_20_2ER. Error de orientación y 2% de errores de bases.

	Similitud		Contigs		% de	Tiempo	Convergencia	
Algoritmos	Prom.	Desv. est.	Cantidad de contigs	Long. del contig más largo	util.	seg/caso	Prom.	Desv. est.
gen_4_15_8_S	97.47	0.75	5.17	9750.50	96.61	5.32 $+ct$	11.20	0.92
gen_5_15_8_S	96.90	0.92	4.57	9867.70	96.96	8.26 $+ct$	22.23	9.28
TGI	98.24	0.0	8.10	6062.47	97.06	18		

De acuerdo a los datos mostrados en la tabla anterior, se aprecia una ligera ventaja del TGI con respecto a los algoritmos genéticos en cuanto a similitud y utilización, sin embargo, en cuanto a cantidad y longitud de contigs, los algoritmos genéticos presentan mejores resultados. De nueva cuenta, no se aprecia diferencia significativa entre los resultados de las funciones de aptitud 4 y 5.

IV.6.2 Casos con 5% de errores de base

En esta sección se muestran los resultados obtenidos sobre el caso AMCG_20_5ER con el algoritmo gen_4_35_15_2, los cuales se comparan en la Tabla XXII junto con TGI:

Tabla XXII: Comparación de los algoritmos genéticos y TGI sobre el caso AMCG_20_5ER. Error de orientación y 5% de errores de bases.

	Similitud		Contigs		% de	Tiempo	Convergencia	
Algoritmos	Prom.	Desv. est.	Cantidad de contigs	Long. del contig más largo	util.	seg/caso	Prom.	Desv. est.
gen_4_35_15_S	94.39	1.01	8.77	5305.83	90.47	5.91 $+ct$	15.27	2.75
TGI	85.73	0.0	54.77	1200.33	57.59	22.40		

En la tabla anterior se aprecia una diferencia significativa a favor del algoritmo genético en cuanto a similitud, contigs y utlización.

IV.7 Repeticiones

Pevzner *et al.* (2001) mencionan que los algoritmos basados en el enfoque OLC presentan dificultades para ensamblar secuencias con repeticiones. En el Apéndice B se muestran los resultados de algunos experimentos sobre repeticiones, en los cuales se aprecia la disminución del desempeño de estas estrategias para estos casos.

Debe mencionarse que hace falta hacer un análisis estadístico con el objetivo de dar mayor rigurosidad al estudio aquí presentado.

Capítulo V

Conclusiones, aportaciones y trabajo futuro

V.1 Sumario

En este trabajo se presenta el problema de Ensamble de Secuencias de ADN, y se propone un modelo para resolverlo. Este modelo considera todos los factores que envuelven dicho problema, como los errores de bases en los segmentos de entrada, así como la orientación de los mismos. Asimismo, se propone un algoritmo genético para resolver el modelo, este algoritmo utiliza una representación, selección y operadores de cruzamiento y mutación, contempla además una fase de eliminación de segmentos y ajuste de individuos. Se propone también una función de aptitud que pretende generar soluciones con pocos contigs con la mayor longitud posible, mediante la integración del número de contigs en dicha función de aptitud. Se propone estudiar casos derivados de secuencias reales, con longitud de 10 Kpb y 20 Kpb.

V.2 Conclusiones

En base a los resultados experimentales obtenidos, se puede llegar a las siguientes conclusiones:

- Los algoritmos genéticos que incluyen la fase de eliminación de segmentos y ajuste de individuos, producen mejores resultados que aquellos que no la incluyen, ésto tanto en casos sin errores, como en aquellos con 2 y 5% de errores de bases. Los

resultados son bajo los criterios: similitud, utilización y tiempo de cómputo.

- El resultado anterior es independiente de la función de aptitud utilizada y del tamaño de la entrada.

- El algoritmo genético, basado en el modelo propuesto en este trabajo, produce resultados comparables a los generados por TGI en casos sin errores y casos con 2% de errores de bases, y para casos con 5% de errores de bases, el algoritmo genético muestra mejores resultados para cada uno de los criterios de calidad.

- Entre el desempeño de la función de aptitud de Parsons *et al.* (1995) y la propuesta en este trabajo no se aprecia diferencia significativa en similitud y utilización. En el tiempo de cómputo, la primera de estas dos funciones produce mejores resultados, lo que se explica por la convergencia que también es más rápida.

- Las repeticiones afectan el desempeño de los algoritmos genéticos y voraces basados en el enfoque OLC.

V.3 Aportaciones

Entre las aportaciones más importantes de este trabajo, se pueden mencionar las siguientes:

- Se ha propuesto un algoritmo genético para el problema de ensamble de secuencias, el algoritmo resuelve el problema modelado como el de encontrar una ruta en un grafo completo conectado, propuesto en este mismo trabajo. El algoritmo está basado en un algoritmo genético básico, al que se le ha integrado una etapa de eliminación de segmentos y una de ajuste de individuos.

  - Se ha propuesto también una función de aptitud que intenta favorecer la generación de soluciones con la menor cantidad posible de contigs, y consecuentemente contigs con la mayor longitud posible.

- Se ha propuesto un modelo para el problema del ensamble de secuencias de ADN, mediante un grafo completo conectado, este modelo considera los errores de bases y la orientación desconocida de los segmentos.

- Se desarrolló un generador de casos basado en el modelo de Engle y Burks (1993), con el cual fueron creados los casos usados en los experimentos.

V.4 Trabajo futuro

A continuación se listan las ideas principales para trabajo futuro derivadas de esta tesis:

- El enfoque del algoritmo genético es el tradicional OLC, estudiar como incluir la metodología propuesta por Pevzner *et al.* (2001) dentro de un algoritmo genético para estudiar el desempeño.

- Para la etapa de consenso:

  - Integrar un módulo de fusión de segmentos sobrantes, que intente fusionar aquellos segmentos que no fueron considerados en ningún contig, con el objetivo de fusionar posibles contigs traslapantes.

  - Integrar un módulo de alineamiento múltiple para la detección de posibles errores de bases.

- Experimentar con cadenas de longitud mucho mayores y comparar los resultados contra los de una estrategia voraz.

- Realizar un estudio estadístico que respalde las diferencias y similitudes encontradas en los resultados experimentales.

- Integrar totalmente el módulo *seqaln*, es decir, integrarlo como una función contenida dentro del programa, o mejor aún, desarrollar un módulo de alineamiento para la etapa de detección de traslapes del enfoque OLC, e incluir en éste la posibilidad de permitir determinado margen de errores dentro del alineamiento.

Bibliografía

Bean, J. 1992. "Genetics and random keys for sequencing and optimization". Reporte técnico , The University of Michigan, Ann Arbor, MI. 21 pp.

Blazewicz, J., M. Figlerowicz, P. Jackwioak, D. Janny, D. Jarczyński, M. Kasprzak, M. Nalewaj, B. Nowirski, R. Styszynski, L. Szajokski, y P. Widera 2004. "Parallel DNA sequence assembly". En: "Memorias del Encuentro Internacional de Ciencias de la Computación (ENC2004)", Colima, México. Sociedad Mexicana de Ciencias de la Computación. 378–382 p.

Blazewicz, J., M. Kasprzak, y W. Kuroczycki 2002. "Hybrid genetic algorithm for DNA sequencing with errors". Journal of Heuristics, 8(5):495–502 p.

Coello, C. A. 2001. "Introducción a la computación evolutiva". Reporte técnico , CINVESTAV-IPN, México, D.F. 285 pp.

Collins, F. S., E. D. Green, A. E. Guttmacher, y M. S. Guyer 2003. "A vision for the future of genomics research". Nature, 422:835–847 p.

Eiben, A. y J. Smith 2003. "Introduction to evolutionary computing". Springer, Alemania. 299 pp.

Engle, M. L. y C. Burks 1993. "Artificially generated data sets for testing DNA sequence assembly algorithms". Genomics, 16:286–288 p.

Fleischmann, R. D., M. Adams, O. White, R. Clayton, E. Kirkness, A. Kerlavage, C. Bult, J.-F. Tomb, B. Dougherty, J. Merrick, G. Sutton, W. FitzHugh, C. Fields, J. Gocayne, J. Scott, R. Shirley, L.-I. Liu, A. Glodek, J. Kelley, J. Weidman, C. A. P. andT. Spriggs, E. Hedblom, M. Cotton, T. Utterback, M. Hanna, D. Nguyen, D. Saudek, R. Brandon, L. Fine, J. L. Fritchman, J. Fuhrmann, N. Geoghagen, L. Gnehm, L. McDonald, K. Small, C. Fraser, H. Smith, , y J. Venter 1995. "Whole genome shotgun sequencing and assembly of the Haemophilus Influenzae Rd genome". Science, 269:496–512 p.

Gardner, M. J., N. Hall, E. Fung, O. White, M. Berriman, R. W. Hyman, J. M. Carlton, A. Pain, K. E. Nelson, S. Bowman, I. T. Paulsen, K. James, J. A. Eisen, K. Rutherford, S. L. Salzberg, A. Craig, S. Kyes, M.-S. Chan, V. Nene, S. J. Shallom, B. Suh, J. Peterson, S. Angiuoli, M. Pertea, J. Allen, J. Selengut, D. Haft, M. W.

Mather, A. B. Vaidya, D. M. A. Martin, A. H. Fairlamb, M. J. Fraunholz, D. S. Roos, S. A. Ralph, G. I. McFadden, L. M. Cummings, G. M. Subramanian, C. Mungall, J. C. Venter, D. J. Carucci, S. L. Hoffman, C. Newbold, R. W. Davis, C. M. Fraser, y B. Barrell 2002. "Genome sequence of the human malaria parasite Plasmodium Falciparum". Nature, 419:498–511 p.

González, L. C. 2004. "Un algoritmo genético para el problema de secuenciación de ADN por hibridación con errores positivos y negativos". Tesis de Maestría, Centro de Investigación Científica y de Educación Superior de Ensenada, Ensenada, Baja California. 121 pp.

Grefenstette, J. J., R. Gopal, B. J. Rosmaita, y D. V. Gucht 1985. "Genetic algorithms for the traveling salesman problem". En: Grefenstette, J. J., editor, "Proceedings of the 1st International Conference on Genetic Algorithms". Lawrence Erlbaum Associates. 1:160–168 p.

Gusfield, D. 1997. "Algorithms on strings, trees, and sequences: computer science and computational biology". Cambridge University Press, Nueva York, NY. 534 pp.

Holt, R. A., G. M. Subramanian, A. Halpern, G. G. Sutton, R. Charlab, D. Nusskern, P. Wincker, A. G. Clark, J. M. C. Ribeiro, R. Wides, S. L. Salzberg, B. Loftus, M. Yandell, W. H. Majoros, D. B. Rusch, Z. W. Lai, C. L. Kraft, J. F. Abril, V. Anthouard, P. Arensburger, P. W. Atkinson, H. Baden, V. de Berardinis, D. Baldwin, V. Benes, J. Biedler, C. Blass, R. Bolanos, D. Boscus, M. Barnstead, S. Cai, A. Center, K. Chatuverdi, G. K. Christophides, M. A. Chrystal, M. Clamp, A. Cravchik, V. Curwen, A. Dana, A. Delcher, I. Dew, C. A. Evans, M. Flanigan, A. Grundschober-Freimoser, L. Friedli, Z. P. Gu, P. Guan, R. Guigo, M. E. Hillenmeyer, S. L. Hladun, J. R. Hogan, Y. S. Hong, J. Hoover, O. Jaillon, Z. X. Ke, C. Kodira, E. Kokoza, A. Koutsos, I. Letunic, A. Levitsky, Y. Liang, J. J. Lin, N. F. Lobo, J. R. Lopez, J. A. Malek, T. C. McIntosh, S. Meister, J. Miller, C. Mobarry, E. Mongin, S. D. Murphy, D. A. O'Brochta, C. Pfannkoch, R. Qi, M. A. Regier, K. Remington, H. G. Shao, M. V. Sharakhova, C. D. Sitter, J. Shetty, T. J. Smith, R. Strong, J. T. Sun, D. Thomasova, L. Q. Ton, P. Topalis, Z. J. Tu, M. F. Unger, B. Walenz, A. H. Wang, J. Wang, M. Wang, X. L. Wang, K. J. Woodford, J. R. Wortman, M. Wu, A. Yao, E. M. Zdobnov, H. Y. Zhang, Q. Zhao, S. Y. Zhao, S. P. C. Zhu, I. Zhimulev, M. Coluzzi, A. della Torre, C. W. Roth, C. Louis, F. Kalush, R. J. Mural, E. W. Myers, M. D. Adams, H. O. Smith, S. Broder, M. J. Gardner, C. M. Fraser, E. Birney, P. Bork, P. T. Brey, J. C. Venter, J. Weissenbach, F. C. Kafatos, F. H. Collins, y S. L.

Hoffman 2002. “The genome sequence of the mosquito malaria Anopheles Gambiae”. Science, 298:129–149 p.

Huang, X. y A. Madan 1999. “CAP3: a DNA sequence assembly program”. Genome Research, 9:868–877 p.

Luque, G., E. Alba, y S. Khuri 2005. “Parallel algorithms for solving the fragment assembly problem in DNA strands”. En: A. Y. Zomaya (ed.), Parallel computing for bioinformatics and computational biology. John Wiley & Sons. New York. 287–304 p.

Marra, M. A., S. J. M. Jones, C. R. Astell, R. A. Holt, A. Brooks-Wilson, Y. S. N. Butterfield, J. Khattra, J. K. Asano, S. A. Barber, S. Y. Chan, A. Cloutier, S. M. Coughlin, D. Freeman, N. Girn, O. L. Griffith, S. R. Leach, M. Mayo, H. McDonald, S. B. Montgomery, P. K. Pandoh, A. S. Petrescu, A. G. Robertson, J. E. Schein, A. Siddiqui, D. E. Smailus, J. M. Stott, G. S. Yang, F. Plummer, A. Andonov, H. Artsob, N. Bastien, K. Bernard, T. F. Booth, D. Bowness, M. Drebot, L. Fernando, R. Flick, M. Garbutt, M. Gray, A. Grolla, S. Jones, H. Feldmann, A. Meyers, A. Kabani, Y. Li, S. Normand, U. Stroher, G. A. Tipples, S. Tyler, R. Vogrig, D. Ward, B. Watson, R. C. Brunham, M. Krajden, M. Petric, D. M. Skowronski, C. Upton, y R. L. Roper 2003. “The genome sequence of the SARS-associated coronavirus”. Science, 300:1399–1404 p.

Parsons, R. J., S. Forrest, y C. Burks 1995. “Genetic algorithms, operators, and DNA fragment assembly”. Machine learning, 21:11–33 p.

Peltola, H., H. Soderlund, y E. Ukkonen 1984. “SEQAID: a DNA sequence assembling program based on a mathematical model”. Nucleics Acids Research, 12(1):307–321 p.

Pevzner, P. A., H. Tang, y M. S. Waterman 2001. “An eulerian path approach to DNA fragment assembly”. PNAS, 98(17):9748–9753 p.

Pop, M., L. Salzberg, y M. Shumway 2002. “Genome sequence assembly: algorithms and issues”. IEEE computer, 35(7):47–54 p.

Sanger, F., A. R. Coulson, G. F. Hong, D. F. Hill, y G. B. Petersen 1982. “Nucleotide sequence of bacteriophage λ DNA”. Molecular Biology, 162(4):729–773 p.

Staden, R. 1980. “A new computer method for the storage and manipulation of DNA gel reading data”. Nucleic Acids Research, 8(16):729–773 p.

Staden, R., D. P. Judge, y J. K. Bonfield 2001. "Sequence assembly and finishing methods". En: A. Baxevanis y B. F. Francis (eds.), Bioinformatics: a practical guide to the analysis of genes and proteins. John Wiley & Sons. New York. 303–322 p.

Tammi, M. T. 2003. "The principles of Shotgun Sequencing and Automated Fragment Assembly". Reporte técnico , Center for Genomics and Bioinformatics. Karolinska Institute, Stockholm, Sweden. 43 pp.

Venter, J. C., M. D. Adams, y E. W. Myers 2001. "The Sequence of the Human Genome". Science, 291:302–311 p.

Waterman, M. S. 2000. "Introduction to Computational Biology: Maps, Sequence and Genomes". Chapman and Hall/CRC, Boca Raton, Florida. 431 pp.

Waterman, M. S. y M. Eggert 1987. "A new algorithm for best subsequence alignments with application to tRNA-rRNA comparisons". Mol. Biol., 197:723–728 p.

Watson, J. D. y F. H. Crick 1953. "Molecular structure of nucleic acids. A structure for deoxyribose nucleic acid". Nature, 171(4356):1304–1352 p.

Yeh, I., T. Hanekamp, S. Tsoka, P. D. Karp, y R. Altman 2004. "Computational Analysis of Plasmodium falciparum Metabolism: Organizing Genomic Information to Facilitate Drug Discovery". Genome Research, 14:917–924 p.

Zaritsky, A. y M. Sipper 2004. "Coevolving solutions to the shortest common superstring problem". Byosistems, 76(1-3):209–216 p.

Apéndice A

Alineamiento local

En este apéndice se describe brevemente *seqaln* que es una aplicación para alineamiento de secuencias, la cual sirve para calcular el alineamiento local condicionado, para la fase de *detección de traslapes*; con más detalle se explica la forma en que se utilizan el módulo $overS$ y brevemente $localS$, que son parte de *seqaln*.

A.1 seqaln

seqaln 2.0 es una aplicación basada en programación dinámica y que fue diseñada y desarrollada por Paul Hardy y Michael Waterman, basándose en el trabajo de Waterman y Eggert (1987); ésta es una herramienta más que el grupo de Biología Computacional de la Universidad del Sur de California ha desarrollado y que se encuentra disponible públicamente[21]. *seqaln* es un paquete que permite hacer alineamiento de secuencias en las cuatro diferentes modalidades que se observan en la Figura 29 y que se describen a continuación:

1 Alineamiento global: alinea todas las letras de una secuencia contra todas las letras de otra secuencia.

2 Alineamiento ajustado: alinea una secuencia corta contra una secuencia larga, usada por ejemplo para encontrar un patrón dentro de una secuencia.

[21] http://www.cmb.usc.edu/software/, febrero-2005

3 Alineamiento de traslape: busca traslapes posibles entre el final de una secuencia y el inicio de otra.

4 Alineamiento local: busca regiones locales de dos secuencias que tienen alta similitud.

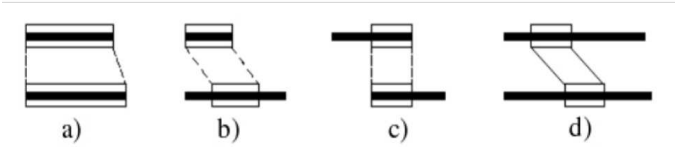


Figura 29: Modalidades de alineamiento: a) Global. b) Ajustado. c) De traslape. d) Local

La programación dinámica es una técnica que obtiene resultados óptimos considerando el problema como un todo (en este caso el problema de alineamiento de secuencias) y posteriormente se divide el problema en subproblemas (alineamiento de subsecuencias), y la solución se construirá a partir de los resultados óptimos de estos subproblemas.

El método general para el alineamiento de secuencias empieza al inicio de las dos secuencias y calcula una matriz de puntajes, llenándola de izquierda a derecha y de arriba a abajo. Suponiendo que las secuencias que se desean alinear son: GCTGATATAGCT y GGGTGATTAGCT, cada secuencia se coloca, como se aprecia en la Tabla XXIII, como columna y renglón respectivamente; cada posición (i, j) de la matriz se llena con el mayor de los tres puntajes mostrados en la siguiente expresión:

$$\begin{aligned} score_{i,j} = best\{ \quad & \{score_{i-1,j-1} + \delta(a_i, b_j)\}^1; \\ & \underset{1\leq k\leq i}{\text{best}}\{score_{i-k,j} + gap(k)\}^2; \\ & \underset{1\leq k\leq j}{\text{best}}\{score_{i,j-k} + gap(k)\}^3\}; \end{aligned} \tag{13}$$

Tabla XXIII: Matriz de puntajes para alineamiento de secuencias.

	_	G	G	G	T	G	A	T	T	A	G	C	T
_	0	-2	-4	-6	-8	-10	-12	-14	-16	-18	-20	-22	-24
G	-2	1	-1	-3	-5	-7	-9	-11	-13	-15	-17	-19	-21
C	-4	-1	0	-2	-4	-6	-8	-10	-12	-14	-16	-16	-18
T	-6	-3	-2	-1	-1	-3	.	.	.				
G													
A													
T													
A													
T													
A													
G													
C													
T													

donde,

- $score_{i,j}$ es el valor de la matriz de puntajes en la posición (i, j).
- $\delta(a_i, b_j)$ es la función que devuelve un puntaje por la coincidencia o no coincidencia de la letra a_i de la secuencia uno y b_j de la secuencia dos.
- $gap(k)$ es la función que representa el costo de una secuencia de *indels* de k letras.

$score_{i,j}$ es el puntaje óptimo para el alineamiento de las primeras i letras de la primera secuencia con las primeras j letras de la segunda secuencia

Una vez terminado el cálculo de la matriz de puntajes se necesita hacer dentro de ella un rastreo hacia atrás de los mejores puntajes, Waterman (2000) muestra los detalles de cada una de las modalidades de alineamiento y del rastreo mencionado. La función con la que se calculan los valores para la matriz de puntajes difiere levemente

para las diferentes modalidades de alineamiento, tales diferencias se muestran en Waterman (2000).

En este trabajo únicamente se utilizan las dos últimas modalidades o módulos de *seqaln*, los cuales se describen a continuación.

A.1.1 overS

Este módulo se utiliza para la detección de traslapes, que ha pasado a llamarse alineamiento local condicionado. Para cada par de segmentos se desea calcular un puntaje de *alc*; a pesar de que este módulo devuelve un puntaje, éste no representa necesariamente lo que se está buscando, el módulo $overS$ calcula dicho puntaje en función de los siguientes parámetros:

- $match$: puntuación a sumar cuando existe un alineamiento de letras idénticas.
- $mismatch$: puntuación a restar cuando existe un alineamiento de letras diferentes.
- $alpha$: puntuación a restar por la primera letra de una secuencia de $indels$.
- $beta$: puntuación a restar por cada una de las letras subsecuentes de una secuencia de $indels$.

La forma en la que se calcula un puntaje (p) es la siguiente:

$$p = c * match - nc * mismtach - \sum_{i=0}^{u-1} [alpha + beta * (k_i - 1)] \qquad (14)$$

donde,

c = cantidad de coincidencias.

nc = cantidad de no coincidencias.

u = cantidad de secuencias de *indels*.

k_i = longitud de la secuencia de *indels* número i.

El siguiente ejemplo muestra cómo se calcula p, supóngase dos segmentos a y b y supóngase el alineamiento (mediante $overS$) de ellos con los parámetros 1, 2, 2 y 2 para *match*, *mismatch*, *alpha* y *beta*, respectivamente, como se aprecia en la Figura 30, dentro del alineamiento existe una secuencia de 2 *indels* (representados con dos guiones) y una no coincidencia.

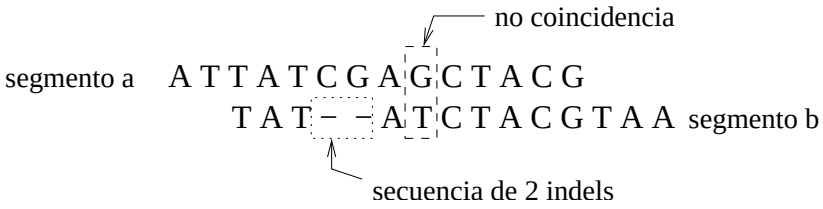


Figura 30: Alineamiento mediante overS.

Sustituyendo en 14 tenemos:

$$p = 9 * 1 - 1 * 2 - [2 + 2 * (2 - 1)] = 3$$

Cabe mencionar que el valor de *beta* debe ser menor que *alpha*, en caso contrario, *beta* toma el valor de *alpha*, convirtiéndose la Ecuación 14 en:

$$p = c * match - nc * mismtach - \sum_{i=0}^{u-1} [alpha * (k_i)] \tag{15}$$

Es importante mencionar también el aspecto de los parámetros para el alineamiento, los cuales han sido establecidos como en el ejemplo anterior: 1, 2, 2 y 2, obtenidos a prueba y error.

El puntaje de alineamiento calculado no representa el *alc* entre dos segmentos, a menos que el alineameinto fuera entre dos segmentos traslapantes y libres de errores de base. Pero estamos suponiendo que los errores de base pueden existir en los segmentos.

La Figura 31 retoma el ejemplo de la Figura 30, en ese alineamiento se observan 9 coincidencias y 3 errores de base, produciendo un puntaje de 3, lo que realmente no da una idea clara del alineamiento entre los dos segmentos, en todo caso, sería la longitud de las subsecuencias alineadas, en este caso de 12, la que podría ser más representativa del traslape.

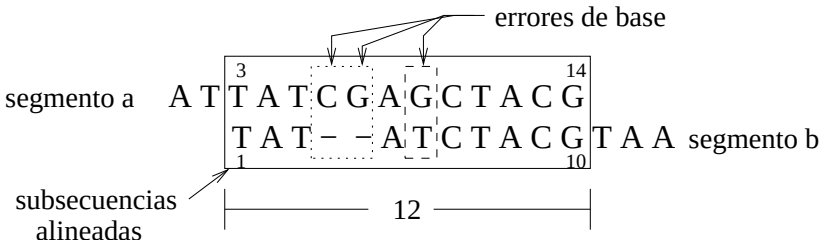


Figura 31: Longitud de un alineamiento mediante overS.

Como se menciona en el Capítulo III, el alineamiento debe ser condicionado a que sea entre el sufijo propio de un segmento y el prefijo de otro, este punto se resuelve parcialmente por el mismo módulo *overS* puesto que siempre busca un alineamiento entre el sufijo de un segmento a y el prefijo de un segmento b.

El puntaje de *alc* entre dos segmentos es la longitud de las subsecuencias alineadas siempre y cuando se cumplan la siguientes condiciones:

1 Mínimo traslape: Significa que el alineamiento debe tener al menos una *mínima* longitud de traslape.

2 Máximo de errores: Se refiere a la cantidad máxima de errores que se permiten dentro del alineamiento para considerarlo como aceptable.

El mínimo traslape y el máximo de errores son parámetros cuya elección es fundamental para la asignación del puntaje de *alc*.

3 Posición: Como se ha dicho, el módulo *overS* por si solo busca alineamiento entre el sufijo de un segmento a y el prefijo de un alineamiento b, pero en el caso de que el segmento b sea un prefijo o sufijo del segmento a, el alinemiento sería como el de la Figura 32, en donde el puntaje es de 12 y no contiene errores de base, pero dado que la longitud del segmento b es de 12, se deduce que el segmento b es un sufijo o prefijo del segmento a, en esos casos se asigna un *alc* igual a -1, otra forma de deducir que el segmento b es sufijo o prefijo del segmento a es mediante información adicional que puede obtenerse del alineamiento, *overS* muestra las posiciones iniciales y finales de las subsecuencias de cada segmento que participan en el traslape, por ejemplo en la Figura 32 se observan las posiciones 3 y 14 para el segmento a y 1 y 12 para el segmento b, entonces si la posición final del segmento b es igual a su longitud, b es un segmento sufijo o prefijo del segmento a (sufijo en este caso particular).

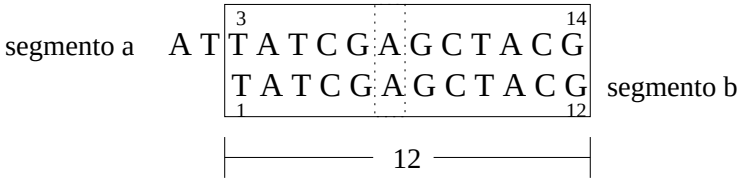


Figura 32: Alineamiento de un segmento sufijo mediante overS.

Supóngase ahora que el segmento b contiene un error de base, en este caso una

sustitución, como lo muestra la Figura 33, el puntaje es de 9, y con un solo error de base; si el máximo de errores permitido fuera de 2, éste alineamiento tendría un *alc* de 12, sin embargo, dado que la posición final del segmento *b* es igual a su longitud, deducimos que el segmento *b* es un sufijo/prefijo del *a*, al igual que el caso de la Figura 32, este alineamiento se califica con un *alc* igual a -1.

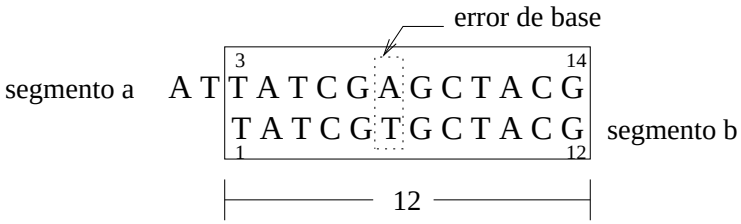


Figura 33: Alineamiento de un segmento sufijo con error mediante overS.

Cualquier alineamiento mediante *overS* que no cumple alguna de las dos primeras condiciones debe calificarse con un puntaje de *alc* igual a 0. Y cuando la condición 3 no se satisface, entonces el *alc* es de -1, el objetivo de asignar este valor negativo es identificar a los segmentos que son prefijos ó sufijos de otro.

A.1.2 localS

Este otro módulo de *seqaln* se utiliza para calcular la similitud de una solución, que estará compuesta por uno o más contigs, este módulo utiliza los mismos cuatro parámetros que *overS*, en este caso se han fijado en 1, 1, 1 y 1 para *match*, *mismatch*, *alpha* y *beta* respectivamente.

Apéndice B

Repeticiones

En este apéndice se muestra el resultado de algunos experimentos para evaluar el desempeño de estrategias basadas en el enfoque OLC, para ensamblar fragmentos obtenidos de secuencias con repeteciones.

B.1 Casos de entrada

En este trabajo se tomó la subsecuencia HUMA_10 (ver página 65), cuya longitud es de 10,089 pb, y se modificó introduciéndole tres repeticiones de 600 bases. Esta modificación consiste en sustituir la subsecuencia [1001..1600] por las subsecuencias [5001..5600] y [7001..7600], la secuencia modificada se llama HUMA_REP.

Dotter[22] es un programa disponible públicamente en Internet, utilizado para la detección de similitud entre secuencias. En este caso se utiliza Dotter para verificar la introducción de las repeticiones, para ello se compara la secuencia HUMA_REP contra sí misma; el resultado que se obtiene se muestra en la Figura 34.

En la Figura 34 se observa una línea diagonal que indica que ambas secuencias son idénticas, asimismo se aprecian seis segmentos, la interpretación de cada uno de ellos es la siguiente. Se toma como ejemplo el segmento 1, en el cual se indican la coordenada inicial y la final, en este caso (5001, 1001) y (5600, 1600) respectivamente;

[22] http://www.cgb.ki.se/cgb/groups/sonnhammer/Dotter.html, agosto-2005

esas coordenadas significan que la subsecuencia [5001..5600] es idéntica a la subsecuencia [1001..1600]. Analizando los cinco segmentos restantes se deduce que las subsecuencias [1001..1600], [5001..5600] y [7001..7600] son idénticas.

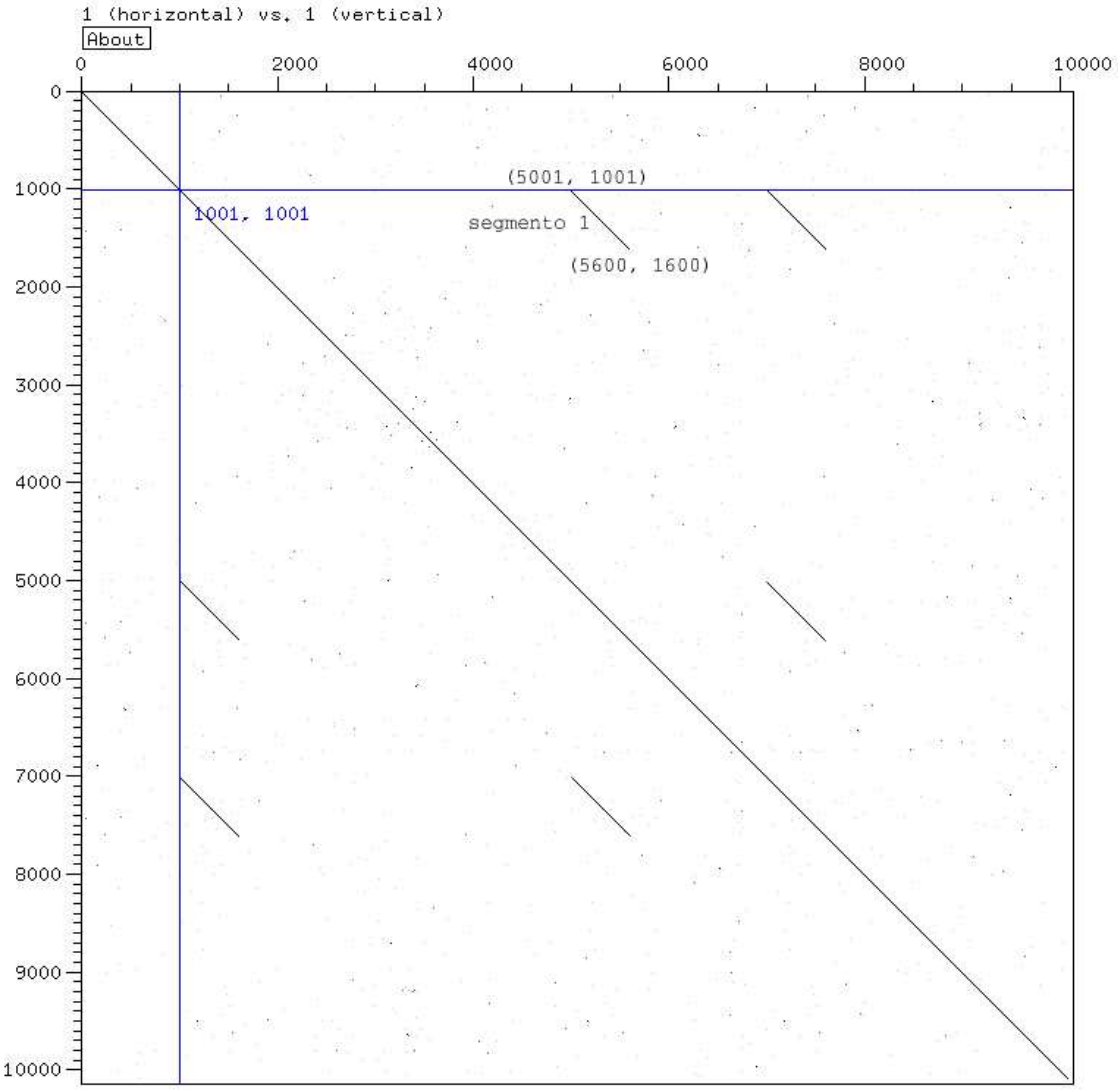


Figura 34: Secuencia con tres repeticiones de 600 pb.

A partir de la secuencia HUMA_REP se forman los casos mostrados en la Tabla XXIV, los cuales se generan como se describe en el Capítulo IV. Cada uno de estos casos contiene segmentos cuya orientación es desconocida.

Tabla XXIV: Casos generados para los experimentos sobre repeticiones.

Nombre del caso	Obtenido de la subsecuencia	Probabilidad de errores de base	Tipo de error
HUMA_REP_SER	HUMA_REP	0	Ninguno
HUMA_REP_2ER	HUMA_REP	0.02	Sustituciones e indels
HUMA_REP_5ER	HUMA_REP	0.05	Sustituciones e indels

B.2 Experimentos y resultados sobre repeticiones.

Enseguida se muestran los resultados de algunos experimentos con la secuencia HUMA_REP, la cual cuenta con tres repeticiones de 600 pb. Los experimentos se realizaron con los casos mostrados en la Tabla XXIV, primero considerando únicamente los errores de bases y posteriormente con 2 y 5% de errores de bases.

B.2.1 Error de la orientación desconocida

Enseguida se muestran los resultados de los experimentos realizados sobre el caso HUMA_REP_SER, obtenido de la secuencia con repeteciones mencionada, en este caso sólo se considera el factor de la orientación desconocida. Los resultados se muestran en la Tabla XXV.

Tabla XXV: Comparación de los algoritmos genéticos y TGI sobre el caso HUMA_REP_SER. TRES REPETICIONES DE LONGITUD 600. Similitud, contigs, utilización y tiempo. Error de orientación únicamente.

	Similitud		Contigs				
Algoritmos	Promedio	Desv. est.	Cantidad de contigs	Long. del contig más largo	% de Calidad	% de util.	Tiempo seg/caso
gen_4_15_0_S	86.91	4.00	4.20	4742.50	98.81	91.34	$1.35 + ct$
gen_5_15_0_S	85.00	4.15	3.40	5016.87	98.31	91.61	$2.12 + ct$
TGI	87.35	4.99	4.50	3942.20	98.87	92.97	7.13

La Tabla XXV muestra una pequeña ventaja a favor de TGI en cuanto a similitud,

mientras que en contigs y utilización no se aprecia una diferencia significativa. Un dato interesante que muestra esta tabla es el porcentaje de calidad, para este caso, en donde no se han introducido errores de bases, se espera que la calidad de los contigs ensamblados sea del 100%. Tanto para los algoritmos genéticos como para TGI la calidad no es la que se espera, lo que hace suponer la existencia de repeticiones, las cuales provocan la confusión mencionada en el Capítulo II.

Por otro lado, si se comparan estos resultados con los de la Tabla IX, en donde se muestran resultados con la secuencia original (antes de la inserción de repeticiones) de estos experimentos, se aprecia un disminución considerable en el desempeño de los algoritmos genéticos y también de TGI, para estos casos con repeticiones. En pocas palabras, la presencia de repeticiones si afecta el desempeño de los algoritmos genéticos aquí propuestos, y como se ve en la tabla anterior, también al desempeño del programa TGI.

B.2.2 Casos con 2% de errores de bases

La Tabla XXVI muestra los resultados sobre el caso HUMA_REP_2ER, en el cual se consideran el factor de la orientación desconocida de los segmentos y además 2% de errores de bases.

Tabla XXVI: Comparación de los algoritmos genéticos y TGI sobre el caso HUMA_REP_2ER. TRES REPETICIONES DE LONGITUD 600. Similitud, contigs, utilización y tiempo. Error de orientación y 2% de errores de bases.

	Similitud		Contigs				
Algoritmos	Promedio	Desv. est.	Cantidad de contigs	Long. del contig más largo	% de Calidad	% de util.	Tiempo seg/caso
gen_4_15_0_S	85.62	4.02	4.33	4619.20	97.67	90.46	$1.65 + ct$
gen_5_15_0_S	83.85	4.00	3.57	4944.47	97.43	91.04	$2.54 + ct$
TGI	90.81	3.52	6.10	3399.70	98.98	93.75	5.43

En la Tabla XXVI se observa una ventaja notable a favor del programa TGI, en cuanto a similitud, mientras que en utilización y contigs la diferencia es pequeña. Lo más importante de esta tabla es que muestra nuevamente que el desempeño de los algoritmos genéticos y de TGI decrece considerablemente para estos casos en donde se trata de ensamblar secuencias con repeticiones. Lo anterior se verifica al comparar la Tabla XIII con la XXVI.

B.2.3 Casos con 5% de errores de bases

Finalmente, la Tabla XXVII muestra los resultados de los experimentos sobre el caso HUMA_REP_5ER, en donde se considera la orientación desconocida de los segmentos y además 5% de errores de bases.

Tabla XXVII: Comparación de los algoritmos genéticos y TGI sobre el caso HUMA_REP_5ER. TRES REPETICIONES DE LONGITUD 600. Similitud, contigs, utilización y tiempo. Error de orientación y 5% de errores de bases.

	Similitud		Contigs				
Algoritmos	Promedio	Desv. est.	Cantidad de contigs	Long. del contig más largo	% de Calidad	% de util.	Tiempo seg/caso
gen_4_15_0_S	82.35	4.14	4.30	4369.43	95.58	87.70	$1.69 + ct$
gen_5_15_0_S	79.18	5.36	3.60	4789.63	94.97	87.13	$4.41 + ct$
TGI	80.29	6.83	27.10	1042.40	97.09	55.54	5.97

Los resultados anteriores muestran la misma tendencia decreciente para los algoritmos genéticos y también para TGI, para estos casos en los que se trata de ensamblar secuencias con repeticiones. Comparando esta tabla contra la Tabla XVII, se observa que el desempeño de TGI decrece pero ligeramente; por otro lado, el desempeño de los algoritmos genéticos si decrece en forma notable, tanto que incluso los resultados están por debajo de los que produce TGI.