Instituto Nacional de Astrofísica, Óptica y Electrónica.

LPS: un algoritmo de ensambles de secuencias cortas de ADN

Por
Francisco Vera Voronisky

Tesis sometida como requisito para obtener el grado de Maestría en Ciencias, en la especialidad de Ciencias Computacionales en el Instituto Nacional de Astrofísica, Óptica y Electrónica.

Supervisada por:

Dr. Jesús A. González Bernal
Coordinación de Ciencias Computacionales
INAOE

Dr. Ansel Yoan Rodríguez González
Coordinación de Ciencias Computacionales
INAOE

Tonantzintla, Pue. Julio, 2014.

©INAOE 2014
Derechos Reservados
El autor otorga al INAOE el permiso de reproducir y distribuir copias de esta tesis en su *totalidad o en partes.*

RESUMEN

La obtención de un genoma de un organismo es un proceso complejo el cual es posible realizarlo mezclando fragmentos de secuencias de ADN usando el solape entre pares. Actualmente, la tecnología existente permite obtener fragmentos con una longitud de a lo máximo de cientos de pares de bases. El proceso de dividir el genoma en fragmentos se le conoce como *secuenciación*. Existen diferentes tecnologías para secuenciar un genoma. Las tecnologías actuales de secuenciación se caracterizan por obtener secuencias de longitudes cortas, las cuales varían en longitud entre 25 y 500 pares de bases. Estos secuenciadores se caracterizan por poder obtener muchos fragmentos a un costo bajo. Sin embargo, las secuencias cortas tienen como deficiencia que un fragmento puede aparecer en varias posiciones en el genoma. Al tratar de realizar el ensamble del genoma, si no consideramos la posibilidad de que este fragmento pueda aparecer repetido en varias posiciones, el genoma reconstruido tendrá varias secuencias con saltos, los cuales son originados debido a que si este fragmento fue usado para unir un par de secuencias, ya no lo volveremos a considerar para unir otros pares de secuencias.

En este trabajo de tesis se presenta el algoritmo LPS para ensamble de secuencias cortas de ADN. LPS propone un mecanismo para construir fragmentos más largos a partir de fragmentos cortos. La obtención de fragmentos largos permite tratar el problema de la posible repetición de los fragmentos cortos que lo componen, ya que al crecer las secuencias, llega un momento en el que se obtendrán pares de bases diferentes, ya sea al principio o al final de la secuencia para las diferentes repeticiones.

LPS es una modificación del algoritmo PadeNA que permite tratar el problema de las repeticiones de secuencias cortas. En la parte experimental evaluaremos la calidad de la reconstrucción de cinco secuencias utilizando fragmentos obtenidos sintéticamente simulando la secuenciación en una computadora. Los ensambles se obtendrán usando PadeNA y LPS+PadeNA y se compararán usando las medidas de ‘número de fragmentos obtenidos o *contigs*’, ‘suma de las longitudes de los *contigs*’, ‘N50’, ‘longitud del *contig* más largo’, ‘promedio de las longitudes de los *contigs*’ y ‘error’. En estos experimentos se mostrará la ventaja de usar LPS con PadeNA sobre el uso de PadeNA por sí solo.

ABSTRACT

Obtaining the genome of an organism is a complex process which is possible to perform merging DNA sequence fragments using overlaps between pairs of sequences. Currently, existing technologies allow us to obtain DNA fragments with a length of hundreds of base pairs. The process of dividing the genome into fragments is called *DNA sequencing*. There are different technologies to sequence a genome. Current sequencing technologies are characterized by obtaining sequences of short lengths, which are sequences with a length between 25 and 500 base pairs. The advantage of this sequencing technology is that it makes possible to obtain many fragments at a low cost. However, short sequences have the shortcoming that they may appear at different positions in the genome. If we do not consider the possibility that this fragment may appear repeated times at various positions, the reconstructed genome will have several gaps. This is caused when a fragment that is a repeat was used once to bridge a pair of sequences and this fragment is not used again for joining other pairs of sequences.

This thesis presents the LPS algorithm for the assembly of short DNA sequences. LPS proposes a mechanism to construct larger fragments from the short fragments. Obtaining long fragments can treat the problem of the possible repetition of short fragments that are part of the longer one. As the sequence grows, it will reach a length in which it will differ in a base pair at the beginning or the end of the sequence. In this way we avoid the repeat.

LPS is a modification of the PadeNA algorithm that allows us to face the problem of short sequence *repeats*. In the experimental results we will evaluate the quality of the reconstruction of five DNA sequences using fragments synthetically obtained by simulation on a computer. DNA Assemblies are obtained using the LPS + PadeNA and PadeNA assemblers. Their quality are compared using measurements such as: number of fragments or *contigs*, sum of the lengths of the *contigs*, N50, length of the longest *contig*, average lengths of *contigs* and error. In these experiments, we will show the advantage of using LPS with PadeNA against PadeNA by itself.

AGRADECIMIENTOS

Para la realización de una tesis, además de llevar un esfuerzo personal, influye también en su éxito el apoyo que haya sido brindado por otras personas que se hayan involucrado, directa o indirectamente en su realización. Es por ello que me gustaría enviar mis agradecimientos a las siguientes personas.

Al doctor Jesús González Bernal que más que un asesor para la tesis ha sido mi consejero y amigo desde que llegué al INAOE.

Al doctor Ansel Yoan Rodríguez González, ex compañero de trabajo en Cuba, co-asesor de la tesis y convertido en familia en México.

A los doctores Iván Olmos Pineda, Patricia Sánchez Alonso y Candelario Vázquez Cruz de la BUAP con cuya ayuda como consultores han permitido la realización de este trabajo.

A la tropa de cubanos del INAOE los cuales han sido mi familia directa en este tiempo.

A la familia Ramos que me aceptó como un miembro más y que sin su apoyo no fuera posible la realización de este trabajo.

A Miguel Ángel Medina Pérez y María Esther Meléndez Abrego por darme la oportunidad de compartir con ellos todos los días parte de su tiempo y mostrar la compatibilidad entre cubanos y mexicanos.

A mis compañeros de maestría con los que he compartido por más de dos años.

A las familias poblanas que me han dado asilo y que se preocuparon que nunca me faltara un plato de comida y una cama para pasar alguna noche fuera de casa.

Al colectivo de doctores de la Coordinación de Ciencias Computacionales del INAOE que me formó profesionalmente y me aportaron parte de su experiencia para formarme como un mejor profesionista.

Al CONACyT que me ha permitido el apoyo económico gracias a sus planes de becas para realizar mi maestría.

Finalmente a las personas que no pude mencionar en esta lista pero que su grano de arena aportó a la terminación de este trabajo tesis. A todos ellos GRACIAS.

DEDICATORIAS

Quisiera dedicar esta tesis primeramente a mis padres, los cuales me han dado su apoyo y nunca dudaron de que podría llegar a donde estoy. Sin ellos no hubiera sido posible estudiar en México en un centro de excelencia como el INAOE. Esta tesis es el resultado del amor y la confianza depositada por ellos.

Además, quisiera dedicar esta tesis a dos personas que ya no están con nosotros y que impactaron en mi vida profesional, la primera es profesor de Matemáticas de preparatoria Oscar Miyar Echemendía quien me enseño el amor por esta ciencia exacta y la segunda mi tutor de tesis de grado de licenciado en Ciencias Computacionales, el Doctor José Luis Gil Rodríguez, quien me formó como profesionista y con cuya asesoría me enseño como introducirme en el ambiente laboral. A ambos, gracias.

CONTENIDO

1. INTRODUCCIÓN 1

1.1 Descripción de la problemática 4

1.2 Motivación 5

1.3 Objetivos 6

1.3.1 Objetivo General 6

1.3.2 Objetivos específicos 7

1.4 Solución Propuesta 7

1.5 Organización de la Tesis 8

2. MARCO TEÓRICO 10

2.1 Conceptos básicos de grafos 10

2.1.1 Definiciones con grafos 10

2.2 Algoritmos de ensamble de secuencias de ADN 12

2.2.1 Algoritmos voraces 14

2.2.2 Solapamiento – presentación – consenso 15

2.2.3 Métodos basados en grafos de de Bruijn 18

2.3 Agrupamiento 21

2.4 Conclusiones 24

3 TRABAJOS PREVIOS 26

3.1 Algoritmos voraces 26

3.2 Solapamiento – presentación – consenso 27

3.3 Métodos basados en grafos de de Bruijn 30

3.4 Conclusiones 34

4 ALGORITMO PROPUESTO 37

4.1 Algoritmo de ensamble LPS 38

4.1.1 Agrupamiento utilizando macro criterio 39

4.1.2 Agrupamiento utilizando micro criterio 41

4.2 Conclusiones 43

5 EXPERIMENTOS Y RESULTADOS OBTENIDOS 45

5.1 Evaluación de la calidad del ensamble 45

5.2 Experimento 1 Ensamble del plásmido Bacillus cereus ATCC 14579 plasmid pBClin15 47

5.2.1 Encontrando el tamaño de los *k*-mers para PadeNA 48

5.2.2 Ensamblado con LPS 51

5.2.3 Ensamblado con *n* caracteres para LPS 1 - *n* caracteres para LPS 2 53

5.2.4 Ensamblado con *n*% caracteres para LPS 1 – *n*% caracteres para LPS 2 58

5.2.5 Ensamblado con *n* caracteres para LPS 1 – 20% caracteres para LPS 2 62

5.2.6 Ensamblado con LPS + PadeNA utilizando las configuraciones anteriores 66

5.3 Experimento 2 Ensamble del plásmido Aeromonas hydrophila plasmid pRA3 71

5.3.1 Encontrando el tamaño de los *k*-mers para PadeNA 71

5.3.2 Ensamblado con LPS 75

5.3.3 Ensamblado con *n* caracteres para LPS 1 - *n* caracteres para LPS 2 75

5.3.4 Ensamblado con *n* caracteres para LPS 1 – 20% caracteres para LPS 2 78

5.3.5 Resumen de la calidad de los ensambles obtenidos del plásmido Aeromonas hydrophila plasmid pRA3 82

5.4 Experimento 3 Ensamble del plásmido Bacillus anthracis str. 'Ames Ancestor' plasmid pXO2 86

5.4.1 Encontrando el tamaño de los *k*-mers para PadeNA 87

5.4.2 Ensamblado con LPS 90

5.4.3 Ensamblado con *n* caracteres para LPS 1 - *n* caracteres para LPS 2 91

5.4.4 Ensamblado con *n* caracteres para LPS 1 – 20% caracteres para LPS 2 94

5.4.5 Resumen de la calidad de los ensambles obtenidos del plásmido Bacillus anthracis str. 'Ames Ancestor' plasmid pXO2 98

5.5 Experimento 4 Ensamble del plásmido Acaryochloris marina MBIC11017 plasmid pREB8 103

5.5.1 Encontrando el tamaño de los *k*-mers para PadeNA 103

5.5.2 Ensamblado con LPS 106

5.5.3 Ensamblado con *n* caracteres para LPS 1 - *n* caracteres para LPS 2 107

5.5.4 Ensamblado con *n* caracteres para LPS 1 – 20% caracteres para LPS 2 110

5.5.5 Resumen de la calidad de los ensambles obtenidos del plásmido Acaryochloris marina MBIC11017 plasmid pREB8 114

5.6 Experimento 5 Ensamble de la bacteria Mycoplasma hominis 119

5.6.1 Encontrando el tamaño de los *k*-mers para PadeNA 120

5.6.2 Ensamblado con LPS 123

5.6.3 Ensamblado con *n* caracteres para LPS 1 - *n* caracteres para LPS 2 123

5.6.4 Ensamblado con *n* caracteres para LPS 1 – 30% caracteres para LPS 2 127

5.6.5 Datos resumidos de la calidad de los ensambles obtenidos de la bacteria Mycoplasma hominis 130

5.7 Resumen y conclusiones de los resultados experimentales..... 133

6 CONCLUSIONES Y TRABAJO FUTURO 137

6.1 Conclusiones 137

6.2 Trabajo Futuro 138

Glosario de Términos 139

REFERENCIAS 141

ÍNDICE DE FIGURAS

Fig. 1 Estructura del ADN *2*
Fig. 2 Subsecuencias obtenidas usando el método de escopeta *3*
Fig. 3 Reads por pares *5*
Fig. 4 Grafos *11*
Fig. 5 Pasos realizados en el ensamble de secuencias de ADN *13*
Fig. 6 Fragmentos de secuencias representadas como grafos *15*
Fig. 7 Grafo de solapamiento formado a partir de 3-mers *17*
Fig. 8 Compresión de la representación en grafos de un conjunto de fragmentos en un solo nodo *18*
Fig. 9 Grafo de de Bruijn de un genoma *19*
Fig. 10 Super read formado por un read por pares *30*
Fig. 11 Unipath en un genoma *32*
Fig. 12 Construcción de un unipath que una un par de reads *32*
Fig. 13 Solapamiento entre la secuencia líder del grupo y otra secuencia analizada *39*
Fig. 14 Suma de las longitudes de los contigs con PadeNA *48*
Fig. 15 Contigs generandos con PadeNA *49*
Fig. 16 Medidas de N50, Longitud del contig más largo y longitud media de los contigs con PadeNA *50*
Fig. 17 Por ciento de error con PadeNA *51*
Fig. 18 Por ciento de error *54*
Fig. 19 Número de contigs *54*
Fig. 20 Suma de las longitudes de los contigs *55*
Fig. 21 N50 ..... *55*
Fig. 22 Longitud del contig más largo *56*
Fig. 23 Longitud media de los contigs *56*
Fig. 24 Por ciento de error *58*
Fig. 25 Número de contigs *59*
Fig. 26 Suma de las longitudes de los contigs *59*
Fig. 27 N50 ..... *60*
Fig. 28 Longitud del contig más largo *60*
Fig. 29 Longitud media de los contigs *61*
Fig. 30 Porciento de error *63*
Fig. 31 Número de contigs *63*
Fig. 32 Suma de las longitudes de los contigs *64*
Fig. 33 N50 ..... *64*
Fig. 34 Longitud del contig más largo *65*

Fig. 35 Longitud media de los contigs *65*
Fig. 36 Por ciento de error *67*
Fig. 37 Número de contigs *67*
Fig. 38 Suma de las longitudes de los contigs *68*
Fig. 39 N50 ..... *68*
Fig. 40 Longitud del contig más largo *69*
Fig. 41 Longitud media de los contigs *69*
Fig. 42 Suma de las longitudes de los contigs con PadeNA *72*
Fig. 43 Contigs generandos con PadeNA *72*
Fig. 44 Medidas de N50, Longitud del contig más largo y longitud media de los contigs con PadeNA *73*
Fig. 45 Por ciento de error con PadeNA *74*
Fig. 46 Número de contigs *75*
Fig. 47 Suma de las longitudes de los contigs *76*
Fig. 48 Longitudes de los contigs *77*
Fig. 49 Por ciento de error *78*
Fig. 50 Número de contigs *79*
Fig. 51 Suma de las longitudes de los contigs *80*
Fig. 52 Longitudes de los contigs *81*
Fig. 53 Por ciento de error *82*
Fig. 54 Por ciento de error *83*
Fig. 55 Número de contigs *83*
Fig. 56 Suma de las longitudes de los contigs *84*
Fig. 57 N50 ..... *84*
Fig. 58 Longitud del contig más largo *85*
Fig. 59 Longitud media de los contigs *85*
Fig. 60 Suma de las longitudes de los contigs con PadeNA *88*
Fig. 61 Contigs generandos con PadeNA *88*
Fig. 62 Medidas de N50, Longitud del contig más largo y longitud media de los contigs con PadeNA *89*
Fig. 63 Por ciento de error con PadeNA *90*
Fig. 64 Número de contigs *91*
Fig. 65 Suma de las longitudes de los contigs *92*
Fig. 66 Longitudes de los contigs *93*
Fig. 67 Por ciento de error *93*
Fig. 68 Número de contigs *94*
Fig. 69 Suma de las longitudes de los contigs *96*
Fig. 70 Longitudes de los contigs *97*
Fig. 71 Por ciento de error *98*
Fig. 72 Por ciento de error *99*

Fig. 73 Número de contigs *99*
Fig. 74 Suma de las longitudes de los contigs *100*
Fig. 75 N50 ..... *100*
Fig. 76 Longitud del contig más largo *101*
Fig. 77 Longitud media de los contigs *101*
Fig. 78 Suma de las longitudes de los contigs con PadeNA *104*
Fig. 79 Contigs generandos con PadeNA *104*
Fig. 80 Medidas de N50, Longitud del contig más largo y longitud media de los contigs con PadeNA *105*
Fig. 81 Por ciento de error con PadeNA *106*
Fig. 82 Número de contigs *107*
Fig. 83 Suma de las longitudes de los contigs *108*
Fig. 84 Longitudes de los contigs *109*
Fig. 85 Por ciento de error *109*
Fig. 86 Número de contigs *111*
Fig. 87 Suma de las longitudes de los contigs *112*
Fig. 88 Longitudes de los contigs *113*
Fig. 89 Por ciento de error *114*
Fig. 90 Por ciento de error *115*
Fig. 91 Número de contigs *116*
Fig. 92 Suma de las longitudes de los contigs *116*
Fig. 93 N50 ..... *116*
Fig. 94 Longitud del contig más largo *117*
Fig. 95 Longitud media de los contigs *117*
Fig. 96 Suma de las longitudes de los contigs con PadeNA *120*
Fig. 97 Contigs generandos con PadeNA *121*
Fig. 98 Medidas de N50, Longitud del contig más largo y longitud media de los contigs con PadeNA *122*
Fig. 99 Por ciento de error con PadeNA *122*
Fig. 100 Número de contigs *124*
Fig. 101 Suma de las longitudes de los contigs *125*
Fig. 102 Longitudes de los contigs *125*
Fig. 103 Por ciento de error *126*
Fig. 104 Número de contigs *127*
Fig. 105 Suma de las longitudes de los contigs *128*
Fig. 106 Longitudes de contigs *129*
Fig. 107 Por ciento de error *130*
Fig. 108 N50 para LPS(15) + LPS (15) + PadeNA y LPS(60%) + LPS (30%) + PadeNA *132*

ÍNDICE DE TABLAS

Tabla 1 Comparación entre plataformas de secuenciación. *3*
Tabla 2 Complejidad de los algoritmos de agrupamiento más conocidos *23*
Tabla 3 Comparación de tipos de algoritmos de ensamble de secuencias de ADN *35*
Tabla 4 Secuencias que conforman un grupo formado utilizando el criterio de macroagrupamiento para un umbral de $\boldsymbol{u = 2}$ *41*
Tabla 5 Evaluación de la calidad de LPS *52*
Tabla 6 Evaluación de la calidad del ensamblado con PadeNA para k = 17 ..... *52*
Tabla 7 Evaluación de la calidad del ensamblado con PadeNA para k = 15 ..... *74*
Tabla 8 Evaluación de la calidad del ensamblado con PadeNA para k = 17 ..... *90*
Tabla 9 Evaluación de la calidad del ensamblado con PadeNA para k = 17 ..... *106*
Tabla 10 Comparación de los valores de calidad del ensamble de la bacteria mycoplasma hominis *131*
Tabla 11 Comparación de la calidad de los ensambles *134*

ÍNDICE DE ALGORTIMOS

Algoritmo 1 Algoritmo básico de ensamble de secuencias de ADN *14*
Algoritmo 2 Leader clustering *24*
Algoritmo 3 Agrupamiento por macro criterio *40*
Algoritmo 4 Agrupamiento por micro criterio *43*

1. INTRODUCCIÓN

El ácido desoxirribonucleico, abreviado ADN, es una molécula que contiene instrucciones que hace a cada organismo único. La molécula de ADN está formada por bloques llamados *nucleótidos* conectados entre sí. Cada *nucleótido* contiene una base nitrogenada que puede llamarse de alguna de las siguientes cuatro formas: adenina y se denota por la letra A, timina y se denota por la letra T, citosina y se denota por la letra C, y guanina y se denota por la letra G. El orden o secuencia de estas bases determina la información genética de un organismo y es responsable para su desarrollo, sobrevivencia y reproducción. Un ejemplo de una secuencia de ADN puede ser CAAAAGGTTTCAAAATG. En los organismos vivos el ADN se presenta como una doble cadena de nucleótidos, donde a cada cadena se le llama hebra. Las dos hebras están unidas entre sí mediante un enlace llamado puente de hidrógeno. Cada tipo de base en una hebra forma un enlace únicamente con un tipo de base en la otra hebra, lo que se denomina *complementariedad de las bases*. De esta forma la base A se enlaza sólo con T, y C con sólo G. Al par de nucleótidos de dos hebras opuestas complementarios se les llama *pares de bases*. Debido a la propiedad de *complementariedad de las bases*, solo es necesario conocer la secuencia de una hebra, para poder deducir las bases de la hebra complementaria. De esta manera, una secuencia de ADN puede ser visto abstractamente como una cadena sobre un alfabeto formado por cuatro letras {A, T, C, G} donde cada letra se identifica con el nucleótido al que denota. Se conoce como *genoma* de un organismo a la secuencia de nucleótidos completa de su ADN. La Fig. 1 muestra la estructura de ADN anteriormente descrita.

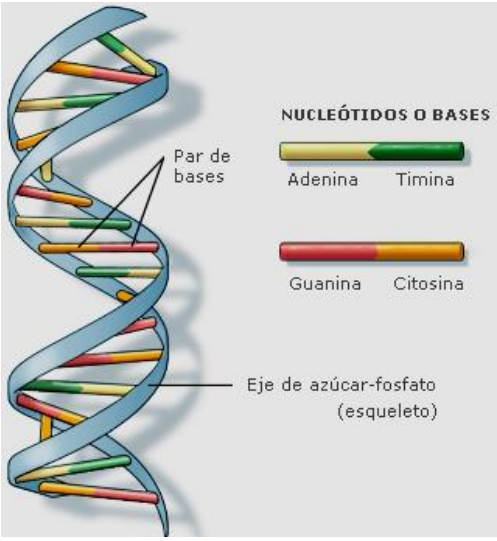

Fig. 1 Estructura del ADN

El problema de la *secuenciación* de un genoma consiste en realizar una copia digital del ADN de un organismo. En el proceso de secuenciar un genoma, el ADN de un organismo es aislado, leído por un secuenciador y convertido en información digital que puede ser procesada por una computadora. Debido a que la tecnología de secuenciación actual puede leer unos pocos pares de bases de ADN (en el rango de cientos de pares de bases) (Margulies et al., 2005), el genoma debe ser separado en pequeños fragmentos llamados '*lecturas'* o '*reads'*. Estos fragmentos deben ser utilizados para reconstruir el genoma mediante un proceso llamado ensamblado.

El ensamblado de secuencias es el proceso de alineamiento y mezcla de fragmentos de secuencias de ADN para reconstruir la secuencia original. En el proceso de obtención de las secuencias, la molécula de ADN es recortada en varios lugares aleatoriamente en fragmentos que puedan ser secuenciados. Los fragmentos que se solapan son luego utilizados para reconstruir la molécula original. Esta estrategia es conocida como *secuenciación de escopeta* o *secuenciación perdigonada* (*shotgun sequencing*). En la Fig. 2 se muestra en la parte superior la secuencia original de ADN mediante una línea alargada. Debajo de esta línea se muestran las subsecuencias de ADN obtenidas con el método de escopeta usando líneas pequeñas. Estas subsecuencias están ubicadas en la posición correspondiente donde irían en la secuencia original.

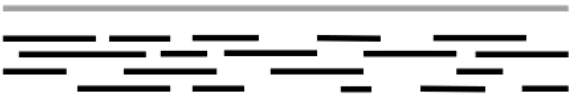

Fig. 2 Subsecuencias obtenidas usando el método de escopeta

El método de secuenciación de escopeta más utilizado hasta el año 2005 era la secuenciación de Sanger (Perkel, 2009). La secuenciación de Sanger es bastante precisa pero es costosa económicamente, requiere de supervisión humana constante y la obtención de genomas completos lleva mucho tiempo.

Actualmente, la secuenciación de escopeta se utiliza en una nueva generación de métodos llamados '*next –generation secuencing*' o NGS por sus siglas en inglés. NGS en contraste con la secuenciación Sanger obtiene muchas lecturas en poco tiempo y con un bajo costo. Gracias a esto es posible repetir la secuenciación varias veces con lo cual se puede aumentar la precisión del ensamble mediante la repetición de datos en las lecturas. Desafortunadamente, NGS produce lecturas con una longitud corta (entre 25 y 500 pares de bases) (Voelkerding, Dames, & Durtschi, 2009).

En el trabajo de (Liu et al., 2012) se realiza una comparación de diferentes tecnologías de NGS. Entre los criterios expuestos en la comparación tenemos la longitud de los *reads*, el porcentaje de aciertos (*accuracy*) y el costo por millones de bases. En la Tabla 1 mostramos varias NGS: 454, SOLiD, Illumina, Ion PGM los cuales son comparados con la tecnología Sanger. Podemos apreciar que la tecnología Sanger tiene un mayor porcentaje de aciertos respecto a los NGS y es posible obtener *reads* de mayor longitud; sin embargo, el muy bajo costo de obtención de los *reads* para los NGS con un alto por ciento de aciertos, siendo en la mayoría de los casos superior al 98%, ha resultado en que sean las plataformas más utilizadas en la actualidad.

Tabla 1 Comparación entre plataformas de secuenciación.

	Roche 454 System	AB SOLiD System	Illumina GA	Ion PGM	Sanger
Longitud de los reads	100-150 bp	35~85 bp	50 ~ 300 bp	200 bp	**400~900 bp**
Accuracy	99%	99.9%	98%	99%	**99.999%**
Costo por 1 millón de bases	$10	$0.13	$0.05~ $0.15	$1	**$2400**

Este trabajo se enfocará en la realización de ensamblados de secuencias de ADN a partir *reads* de longitud corta los cuales pueden ser obtenidos a partir de cualquiera de estas tecnologías NGS mostradas en la Tabla 1.

1.1 Descripción de la problemática

El ensamblado de secuencias de ADN o la reconstrucción de secuencias de ADN (Peltola, Söderlund, & Ukkonen, 1984) es definido como encontrar una secuencia de longitud más corta $\mathcal{S}$, dada una colección $\mathcal{F}$ de fragmentos de secuencias y un valor de error $0 \leq \varepsilon \leq 1$, tal que para cada fragmento $A \in \mathcal{F}$, existe una subcadena B de $\mathcal{S}$ tal que

$$d(A, B) \leq \varepsilon |A|$$

donde $d(A, B)$ es la distancia de edición (Levenshtein, 1966) entre A y B. La distancia de edición entre dos cadenas A y B se define como la mínima cantidad de inserciones, eliminaciones y sustituciones necesarias para trasformar la cadena A en la cadena B.

Este problema es equivalente al problema de la super cadena común más corta, en el cual dada una colección de cadenas, se busca la cadena más corta $\mathcal{S}$, llamada super cadena, tal que cada miembro de la colección es subcadena de $\mathcal{S}$. Dado que este problema es NP-duro, implica que el ensamblaje de secuencias de ADN es NP-duro (Garey & Johnson, 1990).

Adicionalmente a la complejidad del costo computacional del ensamblado de ADN, es posible encontrar fragmentos de ADN que pueden aparecer replicados en la secuencia de referencia, o sea, este fragmento aparece en más de una posición en la secuencia de ADN que se desea reconstruir. A estos fragmentos se les conoce como *repeticiones* o '*repeats'* y no tratarlos en el proceso de ensamblado puede traer como consecuencias que el genoma reconstruido tenga *saltos* o '*gaps*'.

Con el objetivo de reducir los fragmentos que posiblemente sean *repeticiones* o '*repeats'* se pueden seguir dos estrategias:

- Obtener *reads* con una longitud larga.
- Obtener *reads* con información de contexto.

La primera opción es tratar de obtener *reads* con la mayor longitud posible que pueda el secuenciador obtener. Esta opción reduce la cantidad de *repeats* debido a que mientras más larga sea la longitud de los *repeats*, menor será la probabilidad de que el mismo *read* aparezca en otra posición en el genoma a reconstruir. Como se había mencionado anteriormente, los métodos de secuenciación NGS producen *reads* de longitudes cortas (Voelkerding et al., 2009) lo cual hace que esta estrategia no sea viable.

La segunda opción es obtener *reads* con alguna información de contexto o vecindad de fragmentos. Esta información se puede obtener si en lugar de obtener cada *read* independiente uno de otro, los *reads* son obtenidos mediante pares separados a una distancia de *d* caracteres aproximadamente (Bankevich et al., 2012; Pavel A. Pevzner & Tang, 2001; Pham et al., 2012), a estos *reads* se les conoce como *reads por pares* o '*mate pair reads*'. Obtener *reads* por pares reduce la posibilidad de que estos pares pertenezcan a un *repeat* a medida de que la distancia *d* entre los *reads* del par sea mayor. La Fig. 3 muestra un *read* por pares con una distancia *d* medida en la posición de inicio. Cada uno de los pares de los *reads* son representados como una saeta, naranja y azul.

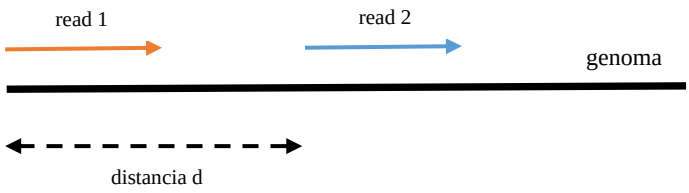

Fig. 3 *Reads* por pares

Como desventaja, estos métodos requieren de una heurística para poder reconstruir un camino entre el par de secuencias. Este camino puede contener errores, los cuales pueden crecer proporcionalmente a la distancia entre el par de *reads*. Adicionalmente, no siempre es posible realizar la secuenciación con información por pares o esta información es subjetiva.

1.2 Motivación

Como se ha mostrado en las secciones anteriores, el problema de ensamble de secuencias de ADN es un problema que presenta una complejidad exponencial. Es por ello que es necesario

el desarrollo de heurísticas que permitan obtener un ensamble que cumpla con las propiedades de

- Cobertura: Los elementos pertenecientes a las secuencias ensambladas cubran lo más posible los elementos de la secuencia de referencia. Es decir, que las secuencias ensambladas sean subcadenas de la secuencia original y que la unión de las subcadenas contengan la mayor cantidad de elementos de la cadena original
- Longitud mínima: La cantidad de elementos obtenidos en el ensamble debe de tener la menor cantidad de elementos que cubra la secuencia original. Es de esperarse que al aplicarse alguna heurística, la longitud de la secuencia ensamblada supere la longitud de la secuencia original, ya que en el proceso de cotejo entre pares de fragmentos se pueden introducir *saltos*.

Adicionalmente, se desea ensamblar *reads* de longitud corta obtenidos simulando las secuencias obtenidas utilizando un secuenciador NGS. Como fue mostrado anteriormente, la desventaja de tener *reads* de longitud corta es que aumenta la posibilidad de obtener un ensamble con *gaps*. En este trabajo de tesis se propone un método partiendo de *reads* de longitud corta para generar secuencias con una mayor longitud utilizando información de solape entre los fragmentos para generar nuevos fragmentos de longitud mayor en vista de detectar *repeticiones* utilizando técnicas de *agrupamientos* (*clustering*). De esta manera, con esta propuesta sería equivalente a obtener un ensamblador de secuencias de ADN el cual construya como resultado secuencias con una longitud larga. Este método lo llamaremos LPS (*Leader, Prefix, Suffix*).

1.3 Objetivos

1.3.1 Objetivo General

Con base en lo antes mencionado, el objetivo general de esta tesis es:

- Desarrollo de un algoritmo de ensamble de secuencias de ADN sin utilizar información de referencia o conocimiento previo.

1.3.2 Objetivos específicos

1. Implementar un método de secuenciación que fragmente una secuencia de ADN o genoma en fragmentos de longitud pequeña simulando a los fragmentos obtenidos utilizando un secuenciador NGS.
2. Proponer un algoritmo de ensamble de secuencias de ADN que utilice *reads* de longitud corta y genere un ensamble con secuencias de ADN con una longitud mayor.
3. Realizar la combinación en cascada de ensambladores para obtener un ensamble con una mejor calidad en comparación con los ensamblados obtenidos utilizando los algoritmos de ensamble por si solos.
4. Evaluar la calidad de los ensamblados obtenidos.

1.4 Solución Propuesta

Este trabajo de investigación se enfoca en el ensamble de secuencias de ADN de longitud corta. Estas secuencias como habíamos mencionado anteriormente son comúnmente obtenidas cuando se utiliza algún método de secuenciación NGS.

Como propuesta de solución se propone primeramente implementar un algoritmo de ensambles de secuencias cortas de ADN al cual llamaremos LPS (*Leader – Prefix – Suffix*). LPS está basado en la utilización de algoritmos de agrupamiento para agrupar las secuencias de ADN que tengan cierta cantidad de caracteres empalmados. Como fase final de LPS se construirán nuevas secuencias con una longitud mayor.

Posteriormente, se propone la utilización de LPS en conjunto con la combinación de otros ensambladores de secuencias de ADN con el objetivo de obtener un ensamblado de mejor calidad. El algoritmo de ensamble de secuencias de ADN para combinar con LPS escogido es PadeNA (Thareja, Kumar, Zyskowski, Mercer, & Davidson, 2011). La elección de este ensamblador se expone en el epígrafe ‘3.3 Métodos basados en grafos de de Bruijn’ en la descripción de PadeNA.

Finalmente se compararán los resultados obtenidos mediante la combinación de ensambladores de secuencias de ADN utilizando LPS + PadeNA contra el ensamble obtenido con PadeNA solamente.

1.5 Organización de la Tesis

A continuación se presenta un panorama general de la organización del presente documento. Este documento cuenta con seis capítulos organizados de la siguiente forma:

El Capítulo 1 “Introducción” es el capítulo actual y es donde describirnos el problema, los objetivos, la motivación y la solución propuesta al problema planteado.

En el Capítulo 2 “Marco Teórico” se tratarán los conceptos fundamentales que se utilizarán en este trabajo de tesis. Primeramente se hará una breve introducción a la teoría de grafos ya que los grafos son una abstracción matemática que se utiliza en el ensamble de secuencias de ADN en los métodos de ensamble mediante grafos de solapamiento – presentación – consenso y los métodos de grafos de de Bruijn. Seguidamente a la introducción de grafos presentamos los tres paradigmas fundamentales de algoritmos para el ensamble de secuencias de ADN: ensamble mediante algoritmos voraces, ensamble mediante grafos de solapamiento – presentación – consenso y los métodos de grafos de de Bruijn. Finalmente se realizará una introducción al agrupamiento o *clustering* debido a que el algoritmo LPS propuesto en este trabajo de tesis se basa en el uso de técnicas de agrupamiento.

En el Capítulo 3, “Trabajos Previos”, se mencionan los trabajos principales organizados por los paradigmas fundamentales de algoritmos de ensamble de ADN mostrados en el Capítulo 2.

El Capítulo 4 “Algoritmo Propuesto” se presenta el algoritmo LPS para el ensamble de secuencias cortas de ADN dividido en el macro criterio y el micro criterio de agrupamiento.

El Capítulo 5 “Experimentos y Resultados Obtenidos” presenta los resultados obtenidos usando nuestra propuesta de LPS con PadeNA y es comparado con PadeNA, un algoritmo de ensamble de secuencias de ADN al que queremos adicionar una mejora. En este capítulo se tomaron 5 secuencias de ADN para ser ensambladas usando LPS junto a PadeNA y comparar el resultado con los obtenidos con PadeNA. El orden de los ensambles de las secuencias es de menor a mayor de manera que se pueda evaluar mejor el rendimiento de LPS con PadeNA.

Finalmente, en el Capítulo 6 “Conclusiones y Trabajo Futuro” mostramos las Conclusiones arribadas de los ensambles obtenidos con la combinación de LPS + PadeNA y exponemos recomendaciones para trabajos futuros.

2. MARCO TEÓRICO

En este capítulo se expondrán los conceptos básicos utilizados en el ensamble de ADN.

Primeramente definiremos lo que es un grafo, ya que este concepto será utilizado posteriormente en los algoritmos de 'solapamiento – presentación – consenso' (epígrafe 2.2.2) y los algoritmos basados en 'grafos de de Bruijn' (epígrafe 2.2.3).

Posteriormente mostramos los conceptos de ensamble de ADN y los algoritmos para ensamblar ADN: los algoritmos voraces, los algoritmos basados en grafos de solapamiento – presentación – consenso y los grafos de de Bruijn.

Finalmente se hará una introducción al agrupamiento o *clustering* debido a que el algoritmo propuesto se basa en uso de algoritmos de agrupamiento para realizar el ensamble de secuencias.

2.1 Conceptos básicos de grafos

Un grafo es una representación abstracta de un conjunto de objetos llamados vértices o nodos. Los nodos pueden estar conectados entre ellos, las conexiones se representan por una arista entre los nodos. Las aristas pueden ser dirigidas o no dirigidas. Las aristas dirigidas pueden ser recorridas en una sola dirección; mientras que las no dirigidas se pueden recorrer en ambos sentidos. Las siguientes definiciones de grafos son tomadas del libro Grimaldi de matemáticas discretas (Grimaldi, 1998).

2.1.1 Definiciones con grafos

Grafo

Un *grafo* G se define como un par ordenado $G = (V, E)$, donde:

- V es un conjunto de vértices o nodos. $V \neq \emptyset$
- E un conjunto de aristas o arcos que relacionan estos nodos. E es un conjunto de pares no ordenados de elementos de V. Un par no ordenado es un conjunto de la forma $\{a, b\}$ de manera tal que $\{a, b\} = \{b, a\}$

Un *grafo dirigido* o *digrafo* es un grafo $G = (V, E)$ donde:

- V es un conjunto de vértices o nodos. $V \neq \emptyset$
- $E \subseteq \{(a, b) \in V \times V : a \neq b\}$ es un conjunto de pares ordenados de elementos de V.

La Fig. 4 muestra dos ejemplos de grafos, a la izquierda un grafo no dirigido con $V = \{A, B\}$ y $E = \{\{a, b\}\}$. Note que el orden de los nodos en la arista $\{a, b\}$ es equivalente a $\{b, a\}$. En la imagen de la derecha se muestra el grafo dirigido $V = \{A, B\}$ y $E = \{(a, b)\}$. En este caso importa el orden de las aristas ya que $(a, b) \neq (b, a)$.

Grafo no dirigido Grafo dirigido

Fig. 4 Grafos

Camino

Sea x, y vértices (no necesariamente distintos) de un grafo no dirigido $G = (V, E)$. Un *camino* $x - y$ en G es una sucesión

$$x = x_0, e_1, x_1, e_2, x_2, e_3, \dots, x_{n-1}, e_n, x_n = y$$

de vértices y aristas de G, que comienza en el vértice x y termina en el vértice y y que contiene las n aristas $e_i = \{x_{i-1}, x_i\}$ donde $1 \leq i \leq n$.

Cualquier camino $x - y$ donde $x = y$ y $n > 1$ es un *camino cerrado*. En caso contrario el *camino es abierto*.

Ciclo

Consideremos un camino $x - y$ en un grafo no dirigido $G = (V, E)$.

a) Si no se repite ninguna arista en el camino $x - y$, entonces el camino es un *recorrido* $x - y$. Un recorrido $x - x$ cerrado es un *circuito*.

b) Cuando ningún vértice del camino $x - y$ se presenta más de una vez, el camino es un *camino simple* $x - y$. El término *ciclo* se usa para describir un camino simple cerrado $x - x$.

Grados de un vértice

Sea $G = (V, E)$ un grafo no dirigido. Para cualquier vértice $v \in V$, el *grado* de v, se denota $\deg(v)$, es el número de aristas en E que son incidentes en v.

Sea $G = (V, E)$ un grafo dirigido. Para cualquier $v \in V$,

a) El grado de entrada de v es el número de aristas de G que llegan a v y se denota por $indeg(v)$.
b) El grado de salida de v es el número de aristas de G que parten de v y se denota por $outdeg(v)$.

Se ha dedicado un espacio para una breve introducción a los grafos, ya que son estructuras utilizadas por métodos de ensamblado de ADN como se verá más adelante en los epígrafes '2.2.2 Solapamiento – presentación – consenso' y '2.2.3 Métodos basados en grafos de de Bruijn'.

2.2 Algoritmos de ensamble de secuencias de ADN

El ensamble de ADN puede ser visto como tomar varias copias de un libro y pasar cada copia del libro por una trituradora y volver a re-ensamblar el libro a partir de los fragmentos obtenidos.

Los pasos realizados en el ensamble son mostrados en la Fig. 5 (Baker, 2012) y estos son:

1. Secuenciar el ADN: El proceso de ensamble consiste en reconstruir una secuencia de ADN a partir de fragmentos más pequeños utilizando información de solapamiento entre ellos. Actualmente solo es posible obtener unos pocos pares de bases de un genoma. Estos pequeños fragmentos, llamados *reads*, son obtenidos mediante un secuenciador (ver Tabla 1) y constituyen los elementos con los cuales se reconstruirá la secuencia de ADN.

2. Encontrar los solapes: Para reconstruir la secuencia de ADN se buscarán los solapes entre los pequeños fragmentos de ADN obtenidos. Con esta información se trazará una estrategia para formar secuencias más largas.
3. Ensamblar los solapes en *contigs*: Un *contig* es una secuencia continua de ADN que ha sido ensamblada a partir de fragmentos de ADN empalmados (Kendrew & Lawrence, 1995). En este paso se crean nuevas secuencias más grandes a partir de los solapes utilizando una estrategia de ensamblado.
4. Ensamblar los *contigs* en *scaffolds*: Un *scaffold* es una secuencia no redundante formada por la unión de uno o más *contigs*. La diferencia principal de los *scaffolds* con los *contigs* es que no se requiere que exista un solape entre secuencias para construir secuencias más largas. Generalmente son utilizados cuando se cuenta con alguna información adicional y se aplica alguna heurística. Este paso es opcional en un ensamble de ADN y puede ser visto como un post procesamiento para mejorar la calidad del ensamble. A diferencia de los *contigs* donde se obtiene un conjunto de secuencias continuas, las secuencias formadas por *scaffolds* típicamente introducen saltos o *gaps* entre ellas.

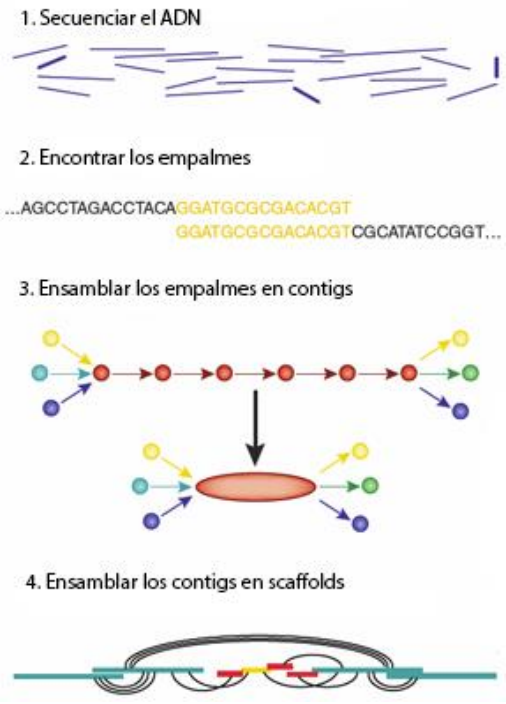

Fig. 5 Pasos realizados en el ensamble de secuencias de ADN

El ensamblado de secuencias de ADN se puede realizar de alguna de las siguientes dos formas: contar con alguna información de referencia con la cual nos podamos apoyar o; simplemente no contamos con conocimiento extra con el cual apoyarnos. Estas dos formas posibles de realizar el ensamble son conocidas como:

- *mapping*: las secuencias son comparadas con secuencias de referencia y se construye una secuencia que es similar, pero no idéntica a la secuencia de referencia.
- *de novo*: pequeñas secuencias son usadas para crear una secuencia grande sin información de referencia extra.

Adicionalmente a estas dos posibles formas de realizar el ensamble de secuencias de ADN, los algoritmos de ensamblaje pueden ser ubicados en algunas de las siguientes categorías: algoritmos voraces, algoritmos basados en solapamiento – presentación – consenso, algoritmos basados en grafos de de Bruijn.

El trabajo de tesis se enfocará en desarrollar un método de ensamble *de novo*, por lo que es de interés revisar los trabajos realizados usando las estrategias de ensamble voraces, solapamiento - presentación – consenso y ensamble basados en grafos de de Bruijn.

2.2.1 Algoritmos voraces

Las primeras técnicas desarrolladas para el ensamblado de secuencias de ADN se basan en técnicas voraces. Esta familia de algoritmos emplea una operación básica, la mezcla de secuencias que tienen el score más alto o porcentaje de solape mayor para formar la nueva cadena.

Un algoritmo básico de ensamble de secuencia de ADN se describe en el Algoritmo 1:

Algoritmo 1 Algoritmo básico de ensamble de secuencias de ADN

```
1. Calcular el alineamiento para todo par de secuencias
2. Seleccionar el par de fragmentos con el mayor solape
3. Mezclar el par de secuencias seleccionado
4. if queda solo un fragmento then
      return
5. else goto 1
```

Los algoritmos voraces sufren del problema de que no son capaces de identificar secuencias repetidas, al elegir siempre la opción con mayor solape, obteniéndose un ensamble fragmentado.

2.2.2 Solapamiento – presentación – consenso

Los ensambladores que se basan en *solapamiento – presentación – consenso,* o en inglés *ovelap – layout – consensus* (OLC), se basan en la construcción de un grafo de solapamiento. Este grafo representa explícitamente en sus nodos los *reads* y las aristas relacionan a los *reads* que se solapen. Usando esta representación, la solución del problema de ensamble de cadenas de ADN consiste en encontrar un camino que recorra todos los nodos. Este problema es conocido como encontrar un camino Hamiltoniano.

A continuación ilustraremos mediante un ejemplo la formación de un grafo para un algoritmo OLC.

Supongamos que tenemos un genoma ATGGCGTGCA, en el resultado de la secuenciación obtenemos los fragmentos F = {ATGGCGT, GGCGTGC, CGTGCAA, TGCAATG, CAATGGC}. El grafo formado se muestra en la Fig. 6 en la parte izquierda, en la parte derecha se muestran los fragmentos alineados y la secuencia reconstruida. Para este ejemplo el camino Hamiltoniano sería ATGGCGT → GGCGTGC → CGTGCAA → TGCAATG → CAATGGC → ATGGCGT, el cual es el camino que pasa por todos los nodos del grafo. En la fase de consenso, utilizando el solape entre los *reads*, podemos reconstruir la secuencia del genoma ATGGCGTGCA.

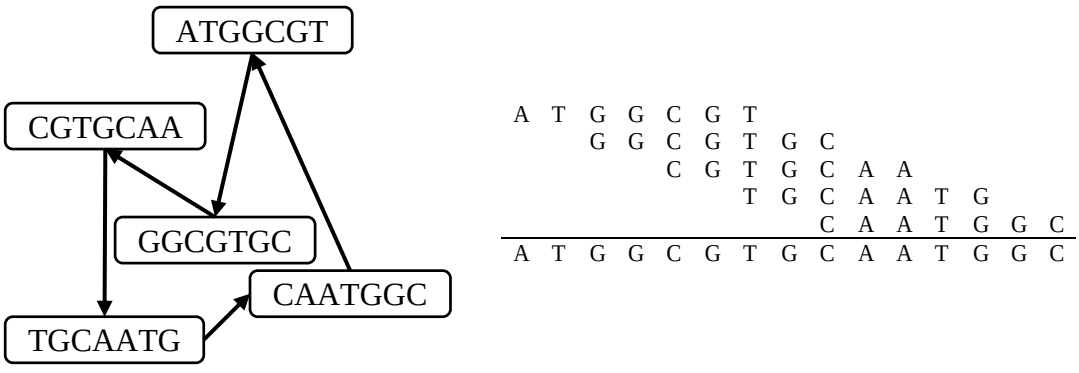

Fig. 6 Fragmentos de secuencias representadas como grafos

Estos ensambladores operan en tres fases. La operación de solapamiento construye un grafo de solapamiento. La operación de presentación comprime el grafo eliminando aristas redundantes. La operación de consenso determina la secuencia del genoma basado en el grafo construido en las fases anteriores.

La etapa de solapamiento es la que generalmente realiza muchos cálculos intensivos ya que deben ser comparados todos los pares de fragmentos para construir el grafo. Para dar un ejemplo del tamaño del volumen de datos, la secuenciación usando el método de escopeta del genoma de la mosca Drosophila tiene 3 millones (3×10^6) de subsecuencias de longitud 500 (5×10^2) pares, siendo 10^{13} el orden de las comparaciones (Deonier, Tavara, & Waterman, 2005). Dada la cantidad de comparaciones necesarias se requiere reducir la cantidad de comparaciones a realizar. Una de las heurísticas usadas para evitar todas las comparaciones y ver los posibles solapes entre todos los pares de *reads* es subdividir cada *read* en subcadenas de tamaño *k* a los cuales le llamaremos *k*-mer. Dividir un *read* en varios *k*-mers permite ya tener un conjunto de secuencias de tamaño *k* con un solape conocido, en lugar de tener que calcularlos. Por ejemplo, para el *read* ATGGCGT, si lo dividimos en 3-mers obtenemos el conjunto de subsecuencias de tamaño 3 {ATG, TGG, GGC, GCG, CGT}. Podemos observar que este conjunto está conformado por todas las subsecuencias de tamaño 3 obtenidas a partir del *read*. Además, dado un *k*-mer, su sufijo es la cadena formada por todos sus nucleótidos excepto el primero y su prefijo está formado por todos los nucleótidos excepto el último. Entonces podemos obtener un solape entre *k*-mers si el sufijo de uno es igual al prefijo de otro. De esta manera, subdividiendo los *reads*, es posible obtener un conjunto de cadenas donde se conoce su solape. La ventaja de los *k*-mers respecto a calcular todos los solapamientos consiste en que es más probable encontrar una subcadena repetida en varios *reads* y comprimir esa información en un nodo de un grafo que realizar el cálculo entre todos los pares de *reads* para determinar el solape.

Este mismo grafo lo podemos trasformar usando 3-mers. Para el genoma ATGGCGTGCA y el conjunto de *reads* F = {ATGGCGT, GGCGTGC, CGTGCAA, TGCAATG, CAATGGC} obtenemos el conjunto de 3-mer M = {ATG, TGG, GGC, GCG, CGT, GTG, TGC, GCA, CAA, AAT}. La construcción de este grafo se realiza de la siguiente manera, cada *k*-mers se representa como un nodo en el grafo y dos *k*-mers se conectan mediante una arista dirigida

si el sufijo del primer *k*-mer es igual que el prefijo del segundo *k*-mer. Este grafo construido según el método descrito se conoce como grafo de de Bruijn (véase epígrafe 2.2.3). En la Fig. 7 se muestra el grafo de de Bruijn para el conjunto M de 3-mers. Para el caso del 3-mer ATG, su sufijo es TG, entonces los nodos adyacentes son los 3-mers que tengan como sufijo a TG, para el conjunto de 3-mers encontramos que TGG y TGC cumplen con esto por lo que conectamos a ATG con TGG y con TGC. Similarmente esto pasa con los nodos GCC, GTG y TGC los cuales tienen más de dos aristas de origen. Para este grafo el camino Hamiltoniano que recorre todos los nodos del grafo es representado por aristas en color naranja. Realizando el recorrido de este camino es posible reconstruir el genoma empalmando los *k*-mers de los nodos.

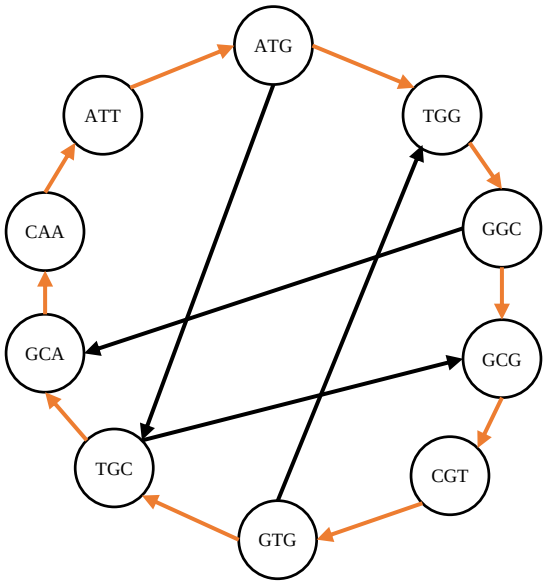

Fig. 7 Grafo de solapamiento formado a partir de 3-mers

La etapa de presentación trata de simplificar el grafo formado en la etapa de solapamiento. El problema de ensamblar las secuencias de ADN consiste en encontrar un camino Hamiltoniano en dicho grafo. Este problema es NP-Completo, siendo necesario simplificar el grafo. Una forma de reducir el tamaño del grafo es realizar compresiones de fragmentos que tengan un alto nivel de solapamiento. La Fig. 8 muestra en la parte superior 5 fragmentos con un alto nivel de solapamiento. En el centro se muestra el grafo correspondiente a las secuencias construido en la fase de solapamiento. En la parte inferior se muestra el nodo en que se comprime el grafo.

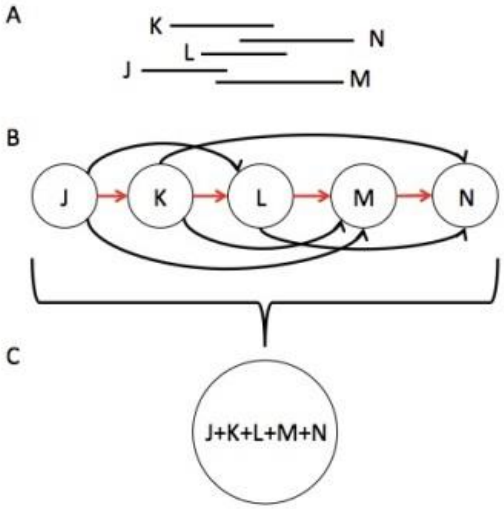

Fig. 8 Compresión de la representación en grafos de un conjunto de fragmentos en un solo nodo

En la fase de consenso se realiza un múltiple consenso de las secuencias pertenecientes a cada uno de los nodos que fueron comprimidos.

Una dificultad de estos métodos, es que se requiere realizar la comparación de todos los pares de secuencias para la construcción del grafo. Adicionalmente, encontrar un ensamble se resuelve mediante un problema de complejidad exponencial. Realizar el consenso entre las secuencias que se encuentren en el camino Hamiltoniano también es de complejidad exponencial (Wang & Jiang, 1994). Es por esto que es necesario el uso de diversas heurísticas que den una solución al problema de ensambles de secuencias.

2.2.3 Métodos basados en grafos de de Bruijn

Los grafos de de Bruijn han sido utilizados para resolver el problema de encontrar la súper cadena común más corta (De Bruijn, 1946). Se han demostrado las ventajas de aplicar estos grafos en el ensamble de ADN (Compeau, Pevzner, & Tesler, 2011).

En estos grafos, a diferencia de los grafos de solapamiento en donde los k-mers son representados en los nodos, serán representados en las aristas. La construcción de estos grafos se realiza de la siguiente manera. Primeramente se forman los nodos para cada secuencia prefijo o sufijo de un k-mer, la cual es una secuencia de longitud k-1. Dos nodos (x, y) se

conectan mediante una arista si existe algún k-mer que tenga como prefijo a x y como sufijo a y.

Por ejemplo, para el genoma ATGGCGTGCA, el conjunto de *reads* F = {ATGGCGT, GGCGTGC, CGTGCAA, TGCAATG, CAATGGC} y el conjunto de 3-mers M = {ATG, TGG, GGC, GCG, CGT, GTG, TGC, GCA, CAA, AAT}, se puede extraer el conjunto de prefijos/sufijos de los 3-mers PS = {AT, TG, GG, GC, CG, GT, CA, AA}, donde cada elemento de este conjunto es una secuencia de longitud 2. El nodo AT lo podemos conectar con el nodo TG mediante la arista representada por el 3-mer ATG debido a que el prefijo de ATG es AT (el primer nodo) y el sufijo es TG (el segundo nodo). De esta forma se construye el grafo de Bruijn mostrado en la Fig. 9.

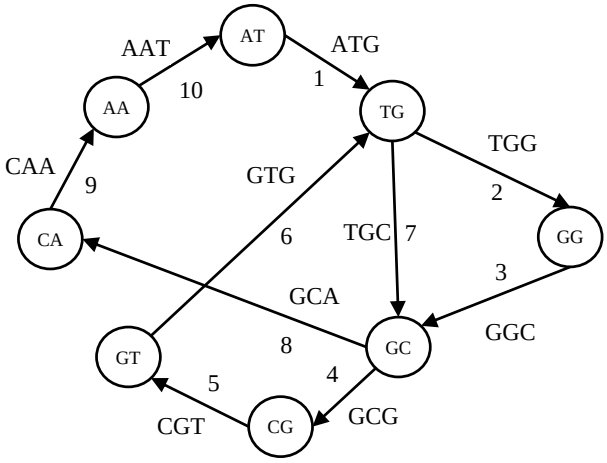

Fig. 9 Grafo de de Bruijn de un genoma

De esta manera, cambiando la representación de los *k*-mers mediante nodos de un grafo hacia las aristas en otro grafo, el problema de ensamblar una secuencia de ADN se transforma en el de realizar el recorrido por todos los nodos de un grafo, para obtener el recorrido por todas las aristas de un grafo, más conocido como recorrido Euleriano. Utilizando el teorema de Euler (Euler, 1736) este problema se puede resolver bajo una complejidad lineal respecto al número de vértices y aristas (Rosen, 2011), en contraste con el recorrido Hamiltoniano que es un problema NP.

Euler demostró que un grafo conexo no dirigido, el cual tiene exactamente dos vértices de grado impar, tiene un camino Euleriano. En caso de que todos los vértices tengan grado par entonces el grafo tiene un ciclo Euleriano.

En teoría de grafos, un camino Euleriano es un camino que pasa por cada arista una y sola una vez. Un ciclo o circuito Euleriano es un camino que recorre todas las aristas de un grafo tan solo una única vez siendo condición necesaria que regrese al vértice de salida. Se debe de tener en cuenta que no importa la repetición de vértices mientras que no se repitan aristas.

Para grafos dirigidos, definimos el grado de entrada de un nodo como el número de aristas que inciden sobre un nodo y el grado de salida de un nodo como el número de aristas que salen de este. Un grafo en el cual, para cada nodo *v*, el grado de entrada es igual al grado de salida se dice que está balanceado. El teorema de Euler entonces plantea que existe un ciclo Euleriano en un grafo dirigido si y solo si el grafo está balanceado.

En la Fig. 9 cada arista además de tener la etiqueta del 3-mer que la representa tiene un número que representa el orden en que se recorren éstas en el camino Euleriano.

Otra de las mejoras usadas en los métodos basados en los grafos de de Bruijn es que se utilizan los *k*-mers en lugar de calcular todos los solapes, como lo hacen los métodos solapamiento – presentación – consenso.

Como deficiencia de esta estrategia es que es sensible al ruido, por lo que es posible la generación de *k*-mers correspondientes a lecturas erróneas en el proceso de secuenciación. Otro de los problemas es que al buscar un camino Euleriano, eliminamos secuencias que aparecen repetidas, al quitar los ciclos del grafo. Esto conduce a resolver un problema de un super camino Euleriano el cual se describe como: "Dado un grafo Euleriano y una colección de caminos en el grafo, encontrar un camino Euleriano que contenga todos estos caminos como subcaminos" (Pavel A. Pevzner, Tang, & Waterman, 2001). A pesar que el problema del super camino Euleriano puede resolverse en tiempo polinomial, cuando existen errores en los *reads* este problema es de complejidad NP-Duro (Paul Medvedev, Georgiou, Myers, & Brudno, 2007).

2.3 Agrupamiento

El *agrupamiento* o *clustering* es una técnica que utilizaremos para desarrollar el algoritmo de ensamble de fragmentos de ADN, es por ello que dedicamos esta sección para mostrar los puntos importantes del *clustering*.

El *agrupamiento* o *clustering* es la tarea de agrupar un conjunto de objetos de tal manera que objetos que estén en un mismo *grupo* o *clúster* son más similares entre ellos (utilizando algún criterio) que elementos de otros grupos.

En el trabajo de (Jain, Murty, & Flynn, 1999) se define al agrupamiento como la organización de patrones en grupos basados en una similitud. Un patrón (o vector de rasgos, observación o dato) es un elemento simple usado por un algoritmo de agrupamiento.

En el ensamble de secuencias de ADN podemos utilizar un algoritmo de agrupamiento para agrupar secuencias que sean semejantes. Luego en cada grupo podemos realizar el ensamble usando solo las secuencias presentes en el grupo. De esta manera podemos evitar realizar la comparación de todos los pares de secuencias para determinar cuáles son las mejores secuencias a mezclar para conformar una nueva secuencia más larga producto de la mezcla.

Los elementos fundamentales de interés que utiliza un algoritmo de agrupamiento son:

1. La representación de un objeto o patrón como un vector.
2. Uso de algún criterio o medida de similitud entre objetos.

En el caso del algoritmo desarrollado en este trabajo se usa la propia representación en caracteres de la secuencia como el vector que describe al objeto.

El criterio de similitud entre objetos a utilizar es la cantidad de caracteres que se empalman entre un par de secuencias. Mientras más caracteres se solapen entre un par de secuencias, más similares serán estas secuencias entre sí.

Para realizar el agrupamiento, existen varios algoritmos los cuales utilizan la representación de los datos y la medida de similitud.

Las técnicas de agrupamiento se dividen en agrupamiento jerárquico y en agrupamiento particional. El agrupamiento jerárquico puede ser divisible o aglomerativo (Berkhin, 2006; Jain et al., 1999). Los métodos divisibles (*top-down*) inician con todo el conjunto de datos en un clúster y forma una jerarquía dividiendo los datos en pequeños subconjuntos sucesivamente dependiendo del criterio de similitud. Los métodos aglomerativos (*bottom-up*) comienzan con n clusters de un solo elemento y forman la jerarquía mezclando sucesivamente los clusters. Ejemplos de algoritmos de agrupamiento jerárquico son single link (Sneath, 1957) y complete link (Defays, 1977). Estos algoritmos requieren de la construcción de una matriz de similitud consumiendo espacio de memoria del orden $O(n^2)$, $O(n^2 d)$ para el cálculo de la distancia y $O(n^3 d)$ para el procedimiento de agrupamiento, donde n es la cantidad de vectores en el conjunto y d es la longitud de los vectores.

El agrupamiento particional identifica una partición de los datos que optimiza el criterio de semejanza. El algoritmo de agrupamiento particional más conocido es *k*-means (MacQueen, 1967). El costo en memoria de éste es $O(kd)$ y la complejidad computacional es $O(nkdt)$, donde n es la cantidad de objetos o vectores a agrupar, k es la cantidad de categorías o grupos en los cuales se van a agrupar los n vectores, d es la longitud de los vectores y t la cantidad de iteraciones que se correrá *k*-means.

El costo computacional y de memoria de estos algoritmos cuando se usan muchos datos, como es el caso del ensamble de secuencias de ADN, hace prohibitivo su uso, por lo que se requiere de otros algoritmos de agrupamiento.

Los métodos de agrupamiento incremental operan de la siguiente manera, el primer objeto es asignado a un grupo o cluster. Luego, los siguientes objetos son asignados a un cluster existente o a un nuevo cluster dependiendo de la similitud del objeto en cuestión y el objeto representativo del cluster. El proceso es repetido hasta que todos los objetos sean asignados a clusters. Solo requieren de una pasada en el conjunto de objetos por lo que su costo computacional es del orden de la cantidad de objetos. Solo es necesario almacenar los elementos representativos de los clusters por lo que la memoria necesaria es del orden de la cantidad de clusters. Como ejemplo de estos algoritmos tenemos a Leaders (Hartigan, 1975).

En la Tabla 2 se muestran la complejidad de los algoritmos de agrupamiento más conocidos según (Vijaya, Narasimha Murty, & Subramanian, 2004). Si comparamos los algoritmos vemos que el algoritmo Leader tiene menor complejidad computacional y requiere menor memoria respecto al resto de los algoritmos de agrupamiento. Ya que los datos de secuencias de ADN son datos grandes, se requiere el uso de algoritmos que permitan ahorrar memoria y tengan menor complejidad algorítmica.

Tabla 2 Complejidad de los algoritmos de agrupamiento más conocidos

Algoritmo	Esquema	Complejidad computacional	Complejidad de memoria
Single/Complete link (Defays, 1977; Sneath, 1957)	Jerárquico	$O(n^3 d)$	$O(n^2)$
k-means (MacQueen, 1967)	Particional	$O(nkdt)$	$O(kd)$
k-medians (Kohonen, 1985)	Particional	$O(n^2 dt)$	$O(nd)$
k-medoids (PAM) (Kaufman & Rousseeuw, 1990)	Particional	$O(k(n-k)^2 d)$	$O(kd)$
DBSCAN (Ester, Kriegel, Sander, & Xu, 1996)	Particional/Jerárquico	$O(n(\log n)d)$	$O(nd)$
CURE (Guha, Rastogi, & Shim, 1998)	Jerárquico	$O(N_{muestras}^2 d)$	$O(N_{muestras}^2)$
BIRCH (Zhang, Ramakrishnan, & Livny, 1996)	Jerárquico	$O(nd)$ (2 pasadas por el conjunto de objetos)	$O(Ndb)$
Leaders (Hartigan, 1975)	**Particional**	$\boldsymbol{O(nd)}$ **(1 pasada por el conjunto de objetos)**	$\boldsymbol{O(ld)}$

n: número de patrones, objetos, d longitud de los vectores que caracterizan a los patrones, k número de grupos en que se deben de agrupar los objetos, t número de iteraciones o corridas, N número de nodos, $N_{muestras}$ número de muestras, b factor de ramificación, l número de líderes.

Debido a su bajo costo computacional y de memoria se seleccionó el algoritmo de Leaders (Hartigan, 1975) para ser adaptado en la propuesta del macro agrupamiento de LPS. Para más detalle se describirá en el tópico 4.1.1. Para conjuntos de datos grandes se ha sugerido

el uso del algoritmo Leaders para realizar agrupamiento (Srinivasulu M, 2012; Vijaya et al., 2004; Vijaya, Narasimha Murty, & Subramanian, 2006).

Sea D el conjunto de objetos a ser agrupados, t un umbral para comparar la similitud entre pares de objetos, C el conjunto de clusters, c un cluster de C y l el líder del cluster c. El algoritmo de Leader es mostrado en Algoritmo 2:

Algoritmo 2 Leader clustering

```
Leader_Clustering(D, t)
input: D -> set of objects, t -> threshold value for
similarity
output: C -> set of clusters
C ← ∅
for each object d ∈ D
  if C = ∅ then
    create a new cluster c
    add d to c and select d as leader
    add c to C
  else
    c ← best_similar_cluster(d, C)
    l ← c.leader
    if sim(l, d) ≥ t then
      add d to c
    else
      create a new cluster c
      add d to c and select d as leader
      add c to C
  return C
```

El algoritmo Leader realiza las siguientes operaciones: dado un umbral t para comparar la similitud, un objeto d y un conjunto de clusters C, se compara el objeto d con el elemento líder de cada cluster en C y tomamos el cluster más similar. Si el valor de similitud supera al umbral t entonces agregamos el objeto d al cluster en cuestión; en otro caso, se crea un nuevo cluster y el líder de ese nuevo cluster es el objeto d.

En el Capítulo 4 se mostrará la modificación de este algoritmo de agrupamiento para adaptarlo al uso de secuencias de ADN.

2.4 Conclusiones

En este capítulo realizamos una breve introducción a los grafos y se definieron sus conceptos básicos. Los grafos son la base de dos grupos de algoritmos de ensamble: los basados en grafos de solapamiento – presentación – consenso y los basados en grafos de de Bruijn.

Posteriormente se describieron los principales esquemas de ensamble. Finalmente se dio una breve introducción a los algoritmos de agrupamiento, de los cuales se utilizará una modificación del algoritmo Leader para el desarrollo de la solución propuesta en este trabajo de tesis.

3 TRABAJOS PREVIOS

En este capítulo se expondrán los principales trabajos realizados en el área de ensambles de ADN. Los trabajos los dividimos según su clasificación en: algoritmos voraces, solapamiento – presentación – consenso y en grafos de de Bruijn.

3.1 Algoritmos voraces

El ensamblador une de manera voraz las secuencias con mayor solapamiento.

El ensamblador TIGR (acrónimo de The Institute for Genomic Research) (Sutton, White, Adams, & Kerlavage, 1995) compara todos los pares de fragmentos y genera una lista de posibles solapes. Usa la distribución de posibles solapes como fragmentos repetidos y no repetidos. Inicialmente, los fragmentos que se consideren como no repetidos son mezclados. Seguidamente se comienzan a mezclar los fragmentos que se consideren repetidos y si se encuentra un posible solape con algún fragmento no repetido que no haya sido mezclado, entonces se utiliza éste en la próxima mezcla.

SSAKE (acrónimo de Short Sequence Assembly by progressive Extension) (Warren, Sutton, Jones, & Holt, 2007) busca las secuencias que se solapan asumiendo que los fragmentos están libres de errores. Las secuencias candidatas a empalmar deben de empalmar su sufijo y prefijo idénticamente hasta cierta longitud establecida como un umbral. Se hace uso de la estructura de árboles de sufijo para el manejo eficiente de los datos.

SHARCGS (SHort-read Assembler based on Robust Contig extension for Genome Sequencing) (Dohm, Lottaz, Borodina, & Himmelbauer, 2007) agrega funcionalidades de preprocesamiento y postprocesamiento a SSAKE. El preprocesamiento realiza un filtrado de secuencias con errores de lectura exigiendo una cantidad mínima de cotejos exactos entre las demás cadenas. El proceso de filtrado es realizado tres veces con diferentes configuraciones, por lo que se generan tres conjuntos de secuencias a ensamblar. El ensamblaje se realiza usando un método basado en SSAKE para cada conjunto. Finalmente, un postprocesamiento es realizado para mezclar los tres ensambles obtenidos y llegar a un consenso final.

VCAKE (Verified Consensus Assembly by K-mer Extension) (Jeck et al., 2007) es una extensión de SSAKE la cual incorpora el cotejo inexacto de cadenas para considerar los errores en las cadenas. Para extender los fragmentos de secuencias, VCAKE utiliza un mecanismo de votación. Dada una secuencia que estamos creciendo y una cierta cantidad de caracteres *n* requeridos para el solape, se buscan entre los *reads* que superen el umbral *n* en el solape. El valor de los caracteres que tomarán los nuevos caracteres que serán agregados a la secuencia se determina por un mecanismo de votación preguntando a los *reads* que se solapen. El valor de la base en esa posición será el valor que más se repita en los *reads* empalmados. Utilizando el mecanismo de votación VCAKE trata de minimizar los errores de los *reads* considerando que la identidad de una base con pocos votos es más probable que sea un error en comparación con una identidad con mayor voto.

QSRA (Quality-value-guided Short Read Assembler) (Bryant, Wong, & Mockler, 2009) es una extensión de VCAKE para regiones en que los *reads* cubran pobremente una región del genoma. QSRA extiende el mecanismo de votación agregando valores de calidad. La extensión de la secuencia en VCAKE procede si se supera un número de votos lo cual se garantiza si se cubre con varios *reads* una misma región del genoma. En caso de que la cobertura sea baja, no es posible realizar la extensión del fragmento de ADN. En cambio QSRA continúa la extensión del fragmento si el solape supera una mínima cantidad de caracteres.

3.2 Solapamiento – presentación – consenso

Solapamiento – presentación – consenso: Las relaciones entre las secuencias son representadas como un grafo, donde los nodos representan a cada una de las secuencias y las aristas conectan a dos nodos si existe un solapamiento entre las secuencias. El problema de ensamblado de secuencias puede ser visto entonces como encontrar un camino que contenga a todos los nodos del grafo (encontrar un camino Hamiltoniano), el cual es un problema NP-Completo (Garey & Johnson, 1990). En la fase de solapamiento, el ensamblador encuentra todos los solapamientos entre las secuencias. En la fase de presentación, el grafo formado es simplificado eliminando información redundante. En la fase de consenso, el ensamblador construye un alineamiento de todas las secuencias.

El ensamblador Celera Assembler (E. Myers et al., 2000) fue inicialmente desarrollado para ensamblar *reads* obtenidos mediante un secuenciador Sanger (ver Tabla 1). Posteriormente fue adaptado para poder operar con *reads* obtenidos mediante un secuenciador con tecnología 454 (Miller et al., 2008) y fue renombrado a CABOG. Celera/CABOG introduce el concepto de solapamiento único (*unitigs*) y de solapamiento *repeat*. Los *unitigs* son subsecuencias obtenidas a partir de la mezclas de otras, que tienen un alto nivel de confiabilidad y que su aparición dentro del genoma completo es de solo una vez. Los *unitigs* son utilizados para construir secuencias más largas y se construyen a partir de *reads* que no sean subcadenas de una secuencia de longitud mayor. Esto se realiza para evitar empalmar fragmentos que pueden ser *repeats* introduciendo *saltos* o *gaps* en el ensamble. Para construir el grafo de solapamiento, se analizan los reads y sus solapes. Los *unitigs* son construidos a partir de caminos maximales simples libres de bifurcaciones, esto es, que todo nodo que se encuentre en un camino simple tiene como grado máximo de entrada 1 y grado máximo de salida 1. Una vez que ya no pueden ser utilizados más *unitigs*, se trata de unir las secuencias formadas usando las secuencias que se consideran como solapamiento *repeat*.

Newbler (Margulies et al., 2005) es un ensamblador de tecnología cerrada distribuido para secuencias obtenidas mediante un secuenciador con tecnología 454 (ver Tabla 1). Inicialmente era un ensamblador para *reads* individuales de aproximadamente 100 bases pero actualmente trabaja con *reads* pareados. Realiza en dos fases la construcción de grafos de solapamiento – presentación - consenso. En la primera fase se construye un grafo OLC para generar pequeños fragmentos de alta confiabilidad libre de errores similares a los *unitigs* de Celera a partir de los fragmentos leídos. En la segunda fase se construye otro grafo OLC para generar secuencias más grandes a partir de los *unitigs*. En esta fase se unen los *unitigs* basado en sus solapes. Los *unitigs* pueden ser divididos en casos donde el prefijo y sufijo de uno de ellos se alinee con varios *contigs*. En este caso, se divide el *unitig* en los *reads* que lo componen siendo estos últimos los que se mezclen con los diferentes *contigs*.

Edena (Hernandez, François, Farinelli, Østerås, & Schrenzel, 2008), a diferencia de los ensambladores OLC presentados anteriormente que trabajan con *reads* obtenidos a partir de secuenciadores con tecnología 454 o Sanger, es un ensamblador para *read* de longitudes cortas obtenidos de los secuenciadores Illumina y SOLiD (ver Tabla 1). Edena, inicialmente,

descarta los *reads* duplicados, seguidamente, se buscan solapamientos entre *reads* de al menos una cierta cantidad de caracteres. Pares de secuencias que tengan solapes redundantes con otros pares son eliminados. Finalmente, los *contigs* son generados a partir de caminos sin ambigüedades obtenidos del grafo de solapamiento.

El ensamblador MaSuRCA, el cual es la abreviación de Maryland Super-Read Celera Assembler (Zimin et al., 2013), es un ensamblador basado en grafos OLC. Este ensamblador construye unos *super-reads*. El objetivo es crear un conjunto de *reads* de longitudes más largas a partir de los *reads* originales reduciendo los datos que alimentarán al ensamblador. Los *super-reads* se forman extendiendo los *reads* hacia adelante y hacia atrás, base a base, hasta que la extensión sea única. Para cada *k*-mer realizamos una tabla con un conteo de la cantidad de apariciones que tiene el conjunto de *reads*. Si un *k*-mer aparece al final de algún *read*, existen cuatro posibles *k*-mers que pudieran seguir a continuación en el genoma: los resultantes de concatenar a las últimas $k - 1$ bases del *read* con alguna de las bases A, C, G o T. MaSuRCA revisa cuántas veces ocurren algunos de esos *k*-mers en la tabla. Si ocurre alguno entonces se dice que tiene un único *k*-mer siguiente y añadimos dicha base al *read*. Este proceso se continúa hasta que el *read* no pueda seguir siendo extendido unívocamente, es decir, el conteo de apariciones de los nuevos *k*-mers es 0 o mayor que 1. Al nuevo *read* creado es a lo que se le conoce como *super-reads*. Esta idea llevada a *reads* por pares es mostrada en la Fig. 10. Extendiendo cada uno de los pares de *reads*, obtenemos dos *super-reads*, en caso de que estos dos *super-reads* se solapen, entonces obtenemos un solo *super-read* resultante de la mezcla de estos dos. Para finalizar el proceso de ensamblado y la creación de los *contigs*, los *super-reads* formados son pasados a una versión modificada del ensamblador basado en solapamiento-presentación-consenso CABOG (Miller et al., 2008). Recordemos que la unidad fundamental de ensamble de CABOG consistía en el uso de *unitigs*. MaSuRCA sustituye estos *unitigs* por los *super-reads*.

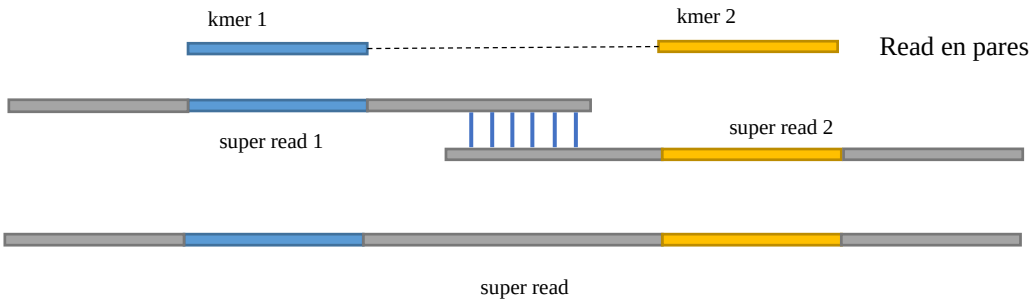

Fig. 10 Super read formado por un read por pares

3.3 Métodos basados en grafos de de Bruijn

A diferencia de representar en los nodos un fragmento de la secuencia, ponemos en las aristas los *k*-mers. De esta manera, en vez de realizar un recorrido Hamiltoniano en el cual se deben de recorrer todos los nodos del grafo, el problema se transforma en recorrer las aristas de un grafo, el cual es el problema del recorrido Euleriano, que puede resolverse eficientemente (Skiena, 1998). Estos grafos son conocidos como grafos de de Bruijn (De Bruijn, 1946).

El ensamblador Euler utiliza una estrategia basada en grafos de de Bruijn (Pavel A. Pevzner et al., 2001). Inicialmente fue desarrollado para *reads* obtenidos mediante un secuenciador Sanger. Posteriormente ha sido modificado para operar con *reads* obtenidos mediante la tecnología 454 de secuenciación (M. Chaisson, Pevzner, & Tang, 2004), *reads* simples obtenidos mediante secuenciador Illumina (M. J. Chaisson & Pevzner, 2008) y *reads* por pares obtenidos mediante Illumina (M. J. Chaisson, Brinza, & Pevzner, 2009). Para eliminar errores, primeramente se aplica un filtro a los fragmentos iniciales. La idea en la que se basa es que los fragmentos que se repitan pocas veces son errores, por lo que son eliminados de la colección de *reads*. A partir de los *reads* restantes se procede a dividirlos en *k*-mers y con estos se construye un grafo de de Bruijn. A este proceso de filtrado se le conoce como *alineamiento espectral* o *spectral alignment* (Pavel A. Pevzner et al., 2001). Los *reads* por pares son tratados en Euler como un solo *read* de longitud larga con desconocimiento de cuáles bases hay entre los dos pares. Para manejar los errores posibles en las secuencias, algunas de las modificaciones de Euler fue la introducción de la estructura de A-grafos, la cual es una combinación de un grafo de de Bruijn y una matriz de adyacencia o A-matriz. El A-grafo es utilizado para identificar los *repeats* (Paul A. Pevzner, Tang, & Tesler, 2004).

El ensamblador Velvet (Zerbino & Birney, 2008) es un ensamblador basado en grafos de de Bruijn para *reads* obtenidos por pares. Este ensamblador realiza compresiones en el grafo, en donde se simplifica a un nodo los caminos que sean identificados como únicos. Este proceso de simplificación resulta en un grafo más simple sin pérdida de información. Esta técnica de compresión de nodos es similar a la utilizada con los *unitigs* en los grafos de solapamiento (E. W. Myers, 1995) por los ensambladores solapamiento – presentación – consenso (E. Myers et al., 2000).

ABySS (Assembly By Short Sequencing) fue diseñado para tratar los problemas de memoria al trabajar con genomas grandes (Simpson et al., 2009). Este ensamblador utiliza *reads* obtenidos por pares. Es un ensamblador programado para operar en sistemas distribuidos utilizando la tecnología de intercambio de mensajes MPI. Este distribuye los grafos entre varias computadoras para utilizar la memoria combinada de todas ellas. Para particionar el ensamblado de secuencias de ADN en el conjunto de computadoras, ABySS distribuye cada nodo individualmente a una computadora. Utilizando esta representación, secuencias adyacentes puede que no estén presentes en la misma computadora. Para poder acceder entonces a un *k*-mer específico implementan una función de hash y para acceder a los *k*-mer adyacentes es necesario calcular una matriz de adyacencia para cada *k*-mer. ABySS trabaja en dos fases. La primera fase no utiliza los *reads* por pares. Los fragmentos que no son ambiguos (fragmentos que no son *repeats*) se mezclan. Inicialmente se leen los datos y se fragmentan en *k*-mers, se construye el grafo de de Bruijn distribuido y se distribuyen los nodos del grafo por las diferentes computadoras que componen el sistema, se calculan las funciones de hash para ubicar los nodos así como la matriz de adyacencia de los nodos. Posteriormente se hace un tratamiento de eliminación de errores los cuales son propios de los *reads* obtenidos por el secuenciador. Finalmente se mezclan los vértices que estén enlazados que no sean ambiguos. La segunda fase usa la información de los *reads* por pares para resolver las ambigüedades y terminar el proceso de mezcla de fragmentos.

El ensamblador AllPaths (Butler et al., 2008) también se basa en un recorrido Euleriano. Al igual que el alineamiento espectral que realiza el ensamblador Euler, AllPaths confía en los *k*-mers que ocurren con alta frecuencia. Solo se retienen los *k*-mers que sobrepasen un umbral de frecuencia. Seguidamente crea los *unipath*. Un *unipath* es una secuencia formada por un

camino maximal sin ramificar. En la Fig. 11 se muestran 5 *unipath* para un genoma. Cada *unipath* se muestra de un color diferente. Los nodos etiquetados como L y R son nodos en los que hay ramificación, por lo que marcan el fin de algún *unipath* y el comienzo de otro.

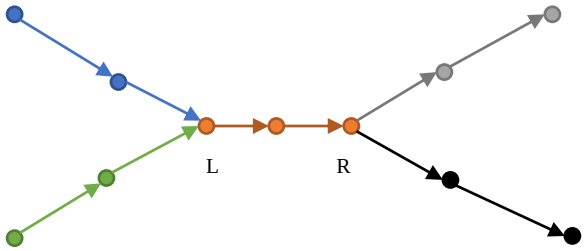

Fig. 11 Unipath en un genoma

Luego de tener los *unipath*, se selecciona un subconjunto de *unipath* y los cataloga como *semillas*. Una semilla ideal sería un *unipath* largo que solo tuviera una copia (la reconstrucción del genoma completo), por lo que es de interés conservar como semillas los *unipath* que sean de longitud larga y que tengan pocos *unipath* que se solapen con este. Seguidamente, utilizando los *reads* por pares, se trata que los *unipath* conecten estos dos *reads* incorporando iterativamente *unipath* cortos a un *unipath* de longitud larga que es generado a partir de los *unipath semillas*. La Fig. 12 muestra un *read* por pares como un par de líneas gruesas negras. A partir de un *unipath semilla* representado por una línea azul, se le empiezan a unir *unipath* pequeños, representados en color naranja, hasta reconstruir un camino que una a los pares de *reads*.

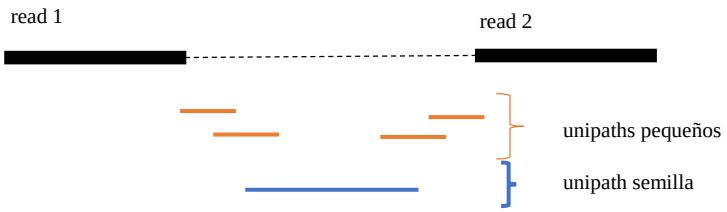

Fig. 12 Construcción de un *unipath* que una un par de *reads*

PadeNA (Parallel de Novo Assembler) (Thareja et al., 2011) es un ensamblador de secuencias de ADN basado en grafos de de Bruijn cuya implementación utiliza técnicas de paralelismo. En la fase de ensamblado PadeNA ensambla las secuencias lo más que pueda mientras que no detecte una ambigüedad. Luego en una segunda fase realiza el proceso de *scaffold* en el

cual utiliza información de los *reads* por pares. En este trabajo nos centramos más en el proceso de ensamble y no de *scaffold* por lo que no es de interés desarrollar mejoras a la segunda fase de PadeNA. En la fase de ensamble, PadeNA construye el grafo de de Bruijn a partir de *reads* simples. La implementación del grafo de de Bruijn, a diferencia de ABySS, está basado en una arquitectura de memoria compartida. Inicialmente los *reads* son particionados en todos los núcleos del procesador que tiene el sistema. Cada núcleo construye un diccionario con los *k*-mers como llave y como valor del diccionario asociado a la llave es el nodo del grafo. La información de adyacencia de los *k*-mers también es guardada en los nodos del grafo. Finalmente, todos los diccionarios son mezclados en un solo diccionario. PadeNA también tiene un algoritmo para eliminar errores propios de los *reads* generados en el proceso de secuenciación.

SPAdes (St. Petersburg genome assembler) (Bankevich et al., 2012; Nurk et al., 2013) es un ensamblador diseñado para ensamblar *reads* por pares, llamados *bi-reads*, obtenidos por los secuenciadores Illumina e Ion Torrent. SPAdes se basa en el uso de grafos de de Bruijn. Los ensambladores como Euler (Pavel A. Pevzner et al., 2001), Velvet (Zerbino & Birney, 2008), AllPath (Butler et al., 2008) a pesar de estar basados en el uso de *bi-reads*, esta información no es utilizada en los grafos de de Bruijn. SPAdes propone el uso de los Grafos por Pares de de Bruijn, o en inglés *Paired de Bruijn graphs* (PDBG) (P. Medvedev, Pham, Chaisson, Tesler, & Pevzner, 2011), una abstracción de grafos de de Bruijn que contiene la información de pares de *k*-mers obtenidos de los *bireads*, llamados *k*-bimers.

Cerulean (Deshpande, Fung, Pham, & Bafna, 2013) es un ensamblador que utiliza secuencias de longitudes cortas y largas. Este ensamblador utiliza *reads* obtenidos por pares. Los *reads* de longitud corta son obtenidos mediante el secuenciador Illumina y los *reads* de longitud larga mediante un secuenciador de Pacific Bioscience, el cual produce *reads* de una longitud promedio entre 4,200 a 8,500 pares de bases. Sin embargo, estos *reads* pueden tener una tasa de error media de un 16%, siendo difícil poder obtener un ensamble utilizando solamente estos *reads* requiriendo una alta cobertura del genoma (Chin et al., 2013). Combinando los *reads* de longitudes cortas, los cuales cuentan con una mayor precisión, con los *reads* de longitudes más largas, es posible producir *contigs* largos de buena calidad. Celurean trabaja en dos fases, la primera utiliza los *reads* de longitudes cortas y construye un grafo de Bruijn

utilizando el algoritmo ABySS, aunque se puede aplicar otro ensamblador basado en grafos de de Bruijn. En la segunda fase, se realiza un mapeo de los *reads* de longitudes largas con los *contigs* obtenidos en el grafo utilizando el algoritmo de alineamiento BLASR (M. Chaisson & Tesler, 2012).

En el trabajo de tesis estamos interesados en desarrollar un algoritmo de ensamble de fragmentos de secuencias de ADN, para los cuales no contamos con información de referencia (ensamble *de novo*).

3.4 Conclusiones

En este capítulo se han expuesto los trabajos principales realizados en el ensamble de secuencias de ADN. Los trabajos los hemos dividido en las tres familias de métodos principales: algoritmos voraces, algoritmos basados en grafos de solapamiento – presentación – consenso y algoritmos basados en grafos de de Bruijn.

Los algoritmos voraces fueron los primeros métodos de ensambles realizados y tienen como dificultad la de no identificar secuencias repetidas.

Los algoritmos basados en grafos de solapamiento – presentación – consenso se basan en encontrar un camino Hamiltoniano el cual tiene una complejidad exponencial actualmente. Además estos métodos consumen mucha memoria ya que necesitan calcular todos los solapes entre pares de secuencias.

Los métodos basados en grafos de de Bruijn transforman el problema de realizar un recorrido Hamiltoniano a un recorrido Euleriano de manera tal que sea computable. Para agilizar los cálculos estos métodos no utilizan el solape entre pares de secuencias, sino que se basan en la simplificación de las secuencias en subcadenas de tamaño k llamadas k-mers y la información de los solapes para la construcción del grafo es obtenida de la descomposición en k-mers de las secuencias. Como desventaja, secuencias más cortas que la longitud de un *repeat* puede introducir gaps. Normalmente estos problemas se resuelven usando secuencias por pares. Otra forma de resolver los *repeat* es obtener *reads* más largos y es la solución propuesta en este trabajo.

A modo resumen en la Tabla 3 mostramos lo explicado en los párrafos anteriores sobre los algoritmos de ensamble de secuencias de ADN.

Tabla 3 Comparación de tipos de algoritmos de ensamble de secuencias de ADN

Tipo de algoritmos	Estrategia	Ventajas	Desventajas	Ejemplo de algoritmo
Voraces	Se unen las secuencias con mayor solape.	Consumen menos memoria. Procesamiento rápido.	No identifican secuencias repetidas	TIGR SSAKE SHARCGS VCAKE QSRA
Grafos de solapamiento – presentación – consenso (OLC)	Representan las relaciones entre las secuencias como un grafo, donde los nodos representan a cada una de las secuencias y las aristas conectan a dos nodos si existe un solapamiento entre las secuencias	Puede representar los *repeats* como aristas múltiples entre secuencias.	Encontrar un camino que recorra todas las secuencias es equivalente a encontrar un camino Hamiltoniano. Consumo alto de memoria al necesitar guardar el cálculo de todos los solapes entre secuencias.	Celera Assembler Newbler Edena MaSuRCA
Grafos de de Bruijn	Representan las secuencias y sus ensambles como grafos de de Bruijn. A diferencias de los grafos OLC, donde los nodos son un fragmento de la secuencia, aquí los fragmentos se ponen en las aristas. En lugar de usar las secuencias completas, se dividen en substring de tamaño *k*, llamados k-mer, para reducir cálculos	Al cambiar de representación repecto a OLC, el problema de encontrar un camino Hamiltoniano es transformado a encontrar un camino Euleriano (complejidad polinomial).	Si el *k*-mer es más pequeño que la longitud de un repeat, se introduce un *salto* o *gap* en el ensamble	Euler Velvet ABySS AllPaths PadeNA SPAdes Cerulean

La propuesta de tesis se basará en modificar un algoritmo basado en grafos de de Bruijn que permita que trabaje con secuencias simples no pareadas para mejorar el tratamiento de las posibles apariciones de los *repeats* y mejorar la calidad del ensamble. El algoritmo seleccionado es PadeNA, el cual es un algoritmo open source desarrollado por Microsoft recientemente, el cual tiene una implementación de procesamiento paralelo con memoria compartida, la cual podemos utilizar en una computadora multi núcleo sin requerir un *cluster*. Adicionalmente este algoritmo permite la corrección de errores de los *reads* y maneja la doble posible dirección de una secuencia.

La modificación propuesta será desarrollar un algoritmo de ensamble basado en *reads* simples que utilice los prefijos y sufijos de manera similar a SSAKE. Este utiliza el criterio

de empalmar el sufijo y prefijo idénticamente con cierta longitud establecida como un umbral entre un par de fragmentos. Haciendo esta operación pretendemos generar *reads* de longitudes más largas o similar a los *unipath* de Celera o a los *super-reads* de MaSuRCA. Finalmente se utilizará el ensamblador PadeNA el cual acepta *reads* simples para mejorar la calidad del ensamble.

4 ALGORITMO PROPUESTO

En este capítulo se describe el algoritmo LPS propuesto empleado en el proceso de ensamblado de secuencias de ADN de longitudes cortas con el objetivo de generar secuencias de longitud más larga.

Para realizar el ensamble es necesario encontrar los solapes entre pares de *reads*. Una estrategia sería encontrar los solapes entre todos los pares de *reads* posibles, lo cual tiene un costo computacional cuadrático utilizando una representación matricial donde se almacenen las posiciones de los solapes entre los dos pares de *reads* y la longitud del solape. Para el caso de bio-informática que se trabaja con grandes volúmenes de datos, realizar esta comparación entre todos los pares es prácticamente imposible. Por ejemplo, para un conjunto con 6,000 *reads*, realizar el cálculo de todos los solapes por pares nos dará un total de $6{,}000 \times 6{,}000 = 12{,}000{,}000$ de operaciones en cada paso en el ensamblado.

Por esta razón se propone utilizar un algoritmo de agrupamiento en donde se agrupen secuencias que sobre pasen cierto umbral de caracteres en el solape. La ventaja de tener grupos de secuencias de ADN es que para realizar la comparación, solo se realizará con el elemento representativo del grupo, y no con todas las secuencias, para determinar la pertenencia o no de la secuencia en cuestión.

El algoritmo de agrupamiento debe ser eficiente. Hemos escogido el algoritmo de agrupamiento Leader (Hartigan, 1975) ya que este realiza una sola pasada por todos los elementos de la base de datos de secuencias de entradas. En la Tabla 2 se muestra que Leader es el algoritmo de agrupamiento que requiere menos memoria y de menor complejidad computacional en comparación con otros algoritmos de agrupamiento clásicos.

El agrupamiento por Leader se realizará en una primera etapa de LPS mostrado en el epígrafe 4.1.1. Posteriormente se realizará una segunda etapa donde se realizará el consenso de las secuencias que pertenecen a cada grupo mostrado en el epígrafe 4.1.2.

LPS es un algoritmo de ensamble a ser aplicado a *reads* simples de longitud corta obtenidos independientemente uno de otro, o sea, no utiliza los *reads* secuenciados por pares. Como se

había planteado, estos *reads* tienen el problema de que al ser cortos, éstos pueden aparecer repetidos en diferentes posiciones del genoma que se desea reconstruir. Con LPS es posible realizar en ensamble utilizando una heurística sencilla para mezclar pares de secuencias, con el objetivo de

- Reducir comparaciones entre pares de secuencias
- Generar secuencias de longitudes largas sin descartar posibles mezclas.
- Reducir la posibilidad de que la secuencia generada sea un *repeat*.

Las secuencias resultantes obtenidas por LPS pueden ser vistas como *reads* de longitud más larga o *super-reads* y estos pueden ser pasados como entrada a otro algoritmo de ensamblado que reciba secuencias simples y mejorar la calidad del ensamble.

El ensamblador escogido en este trabajo de tesis para mejorar los resultados fue PadeNA. Este es un ensamblador desarrollado recientemente, basado en grafos de de Bruijn, paradigma en el cual se basan la mayoría de los ensambladores más recientes según lo mostrado en el capítulo de trabajos previos. Este ensamblador también admite la corrección de errores para los cuales no está preparado LPS, maneja la posibilidad de doble dirección de una secuencia de ADN, además su implementación está basada en técnicas de paralelización utilizando memoria compartida.

En los próximos epígrafes explicaremos el algoritmo LPS propuesto en este trabajo de tesis.

4.1 Algoritmo de ensamble LPS

La propuesta presentada es un algoritmo para generar secuencias a partir de *reads* de longitud corta, secuencias con una longitud mayor mezclando secuencias con un alto nivel de solapamiento entre los prefijos y sufijos.

Para realizar la mezcla de secuencias, se realizará un proceso de agrupamiento en dos pasos. Un primer paso usando un criterio de macro agrupamiento en donde se agrupan fragmentos con un prefijo o sufijo en común y un segundo paso utilizando un criterio de micro agrupamiento aplicado sobre las secuencias que pertenecen a los grupos obtenidos en el primer paso, para separar las secuencias que tengan el mismo prefijo (sufijo), y su sufijo

(prefijo) que no coincidan. Este criterio se utiliza para la detección de posibles repeticiones y generar secuencias independientes que representen estos *repeats*.

El algoritmo propuesto lo llamaremos LPS, que son las iniciales de *Leader – Prefix – Suffix*, los cuales son los tres elementos principales en que se basa: una secuencia líder del *cluster*, el conjunto de secuencias con prefijo que coincidente con un fragmento de la secuencia líder y el conjunto de secuencias con sufijos coincidente con un fragmento de la secuencia líder.

4.1.1 Agrupamiento utilizando macro criterio

En la fase de macro agrupamiento utilizaremos el algoritmo de agrupamiento de Leader (Hartigan, 1975).

Para realizar el agrupamiento seleccionamos un fragmento como el elemento líder del grupo y para los restantes fragmentos se cuenta la cantidad de caracteres que se solapan del prefijo del líder del grupo con el sufijo del fragmento analizado. Similarmente se cuentan los caracteres que se solapan entre el sufijo del líder y el prefijo de la secuencia analizada. Si cualquiera de estos solapamientos supera un umbral establecido de caracteres, entonces consideramos que el fragmento analizado pertenece a ese grupo, en otro caso, se creará un nuevo grupo con ese fragmento como líder. La Fig. 13 muestra en color gris el solapamiento del sufijo de la cadena Líder con el prefijo del fragmento analizado.

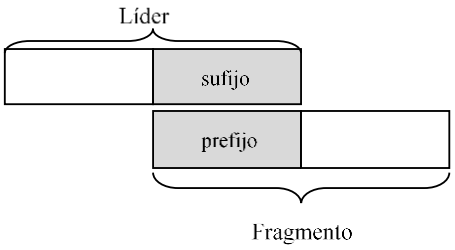

Fig. 13 Solapamiento entre la secuencia líder del grupo y otra secuencia analizada

Denotemos como F el conjunto de fragmentos de secuencia f obtenido del secuenciador, C un conjunto de grupos de fragmentos, u un valor de umbral y $|s|$ la longitud de la secuencia s.

El algoritmo para realizar el macro agrupamiento es mostrado en el Algoritmo 3:

Algoritmo 3 Agrupamiento por macro criterio

```
macro criterion cluster(F, u)
input: F -> set of sequences fragments, u -> threshold value
output: C -> set of clusters fragments
for each fragment f ∈ F
  if C = ∅ then
    create a new cluster c
    add f to c and select f as leader
    add c to C
  else
    for each cluster c ∈ C
      l ← c.leader
      s' ← longest prefix of f that is suffix of l
      s'' ← longest suffix of f that is prefix of l
      k ← max(|s'|, |s''|)
      if k > u then
        add f to c
      else
        create a new cluster c'
        add f to c' and select f as leader
        add c' to C
```

El Algoritmo 3 es una modificación del algoritmo de agrupamiento Leader mostrado en Algoritmo 2. Recordemos que el algoritmo de Leader realiza el agrupamiento de un objeto d en un conjunto de clusters C, usando un umbral u para comparar la similitud. Si el valor de similitud supera al umbral u entonces agregamos el objeto d al cluster en cuestión; en otro caso, se crea un nuevo cluster y el líder de ese nuevo cluster es el objeto d. La modificación propuesta es para especializar el objeto d como una secuencia de ADN. Las operaciones realizadas en la modificación propuesta en el Algoritmo 3 son: dado un umbral u para comparar la similitud, una secuencia d y un conjunto de clusters C, se compara la secuencia d con la secuencia líder de cada cluster en C y tomamos el cluster más similar. Para comparar la similitud entre pares de secuencias usamos como criterio la cantidad de caracteres que se solapan entre estas. Si el valor de similitud supera al umbral u entonces agregamos la secuencia d al cluster en cuestión; en otro caso, se crea un nuevo cluster y el líder de ese nuevo cluster es la secuencia d.

Como resultado son obtenidos grupos que tienen la forma como se muestra en la Tabla 4. En la primera fila se muestra la secuencia AAAA, la cual es líder del grupo. Las secuencias AAAG, AAGG, AATG, CAAA son los fragmentos que pertenecen al grupo. Los fragmentos del grupo en las filas 2, 3 y 4 su prefijo coincide con un sufijo del líder, las subcadenas de estos fragmentos que no se solapan con el líder son G, GG y TG respectivamente y están

mostradas en gris. En la última fila se muestra un fragmento donde su sufijo se solapa con un prefijo del líder. La subcadena del fragmento formada por la letra C no se solapa con el líder y es mostrado en gris.

Tabla 4 Secuencias que conforman un grupo formado utilizando el criterio de macroagrupamiento para un umbral de $u = 2$

-	A	A	A	A	-	-
-	-	A	A	A	G	-
-	-	-	A	A	G	G
-	-	-	A	A	T	G
C	A	A	A	-	-	-

4.1.2 Agrupamiento utilizando micro criterio

A partir de las cadenas obtenidas usando el macro agrupamiento, vemos que podemos fusionar las secuencias con la secuencia líder para crear las nuevas secuencias AAAAG, AAAAGG, AAAATG, CAAAA. Se puede observar que el fragmento AAAAG está contenido dentro del fragmento AAAAGG, por lo que en vez de generar un nuevo fragmento obtenido de la mezcla del líder con cada una de las secuencias, podemos analizar las subcadenas que no se solapen con el líder. Si alguna de estas subcadenas está contenida dentro de otra, la podemos descartar ya que el fragmento de la subcadena más corta estará contenido dentro del fragmento generado de la subcadena más larga. En caso de que no se pueda fusionar entonces se creará un nuevo fragmento, este sería el caso del fragmento AATG. Nótese que las subcadenas que no empalman con el líder, GG y TG, no se contienen entre ellas. Este proceso se realizará en la fase de micro agrupamiento.

En esta fase de agrupamiento, se unirán las secuencias que su sufijo sea subcadena de otro sufijo, por ejemplo, en la Tabla 4, el sufijo G se unirá con el sufijo GG. Similarmente esta operación se realizará para los prefijos. De esta forma se pretende reducir la cantidad de fragmentos en los clústeres formados en la fase de macro criterio y se formarán cadenas más largas, lo cual es lo que buscamos realizar con el algoritmo LPS.

En el proceso de macro agrupamiento, las cadenas que pertenecen al grupo las podemos dividir en dos conjuntos: el conjunto P que contiene a los fragmentos cuyo sufijo coincide con el prefijo del fragmento líder, y el conjunto S que contiene los fragmentos cuyo prefijo

coincide con algún sufijo del fragmento líder. Por ejemplo, en la Tabla 4, $P = \{CAAAA\}$ y $S = \{AAAAG, AAAAGG, AAAATG\}$.

Sea L el conjunto formado por el fragmento líder del grupo, podemos denotar por $\bar{P}$ al conjunto de prefijos de los fragmentos que pertenecen a P que no tienen solape con el fragmento líder, o sea, $\bar{P} = \{p \in P - L\}$ y $\bar{S}$ el conjunto de sufijos de los fragmentos de S que no tienen solape con el fragmento líder, o sea, $\bar{S} = \{s \in S - L\}$. Para el grupo mostrado en la Tabla 4, obtenemos que $L = \{AAAA\}$, $\bar{P} = \{C\}$ y $\bar{S} = \{G, GG, TG\}$.

El proceso de micro agrupamiento se centrará en las cadenas que pertenecen a los conjuntos $\bar{P}$ y $\bar{S}$. Para cada uno de estos conjuntos se identifican las cadenas que son subcadenas de otro elemento de ese mismo conjunto, las cadenas que cumplan esto se pueden agrupar en la cadena de longitud más larga que es la que contendrá a las restantes. A estos nuevos conjuntos obtenidos del agrupamiento los denotamos por $\bar{P}'$ y $\bar{S}'$, y llamémosle conjunto prefijo reducido y conjunto sufijo respectivamente.

Para el conjunto $\bar{S} = \{G, GG, TG\}$ obtenido de los fragmentos del grupo mostrado en la Tabla 4, podemos agrupar a las cadenas G con GG, entonces el nuevo conjunto creado sería $\bar{S}' = \{GG, TG\}$. En el caso del conjunto $\bar{P} = \{C\}$, que contiene un solo elemento, no es posible reducirlo por lo que $\bar{P}' = \bar{P} = \{C\}$

Una vez obtenidos el conjunto prefijo reducido y el conjunto sufijo reducido, podemos construir las secuencias obtenidas de la fusión de los fragmentos que forman el grupo. Los nuevos fragmentos de las secuencias mezcladas pertenecerán al conjunto $\bar{P}' \times L = \{pl | p \in \bar{P}' \wedge l \in L\}$ y $L \times \bar{S}' = \{ls | l \in L \wedge s \in \bar{S}'\}$, donde el operador $\times$ denota el producto cartesiano entre dos conjuntos. Como las secuencias que pertenecen a $\bar{P}' \times L$ tienen como sufijo a L, que a su vez es prefijo de las secuencias en $L \times \bar{S}'$, entonces los fragmentos obtenidos como resultado de la fase de micro agrupamiento pertenecen al conjunto $\bar{P}' \times L \times \bar{S}' = \{pls | p \in \bar{P}' \wedge l \in L \wedge s \in \bar{S}'\}$.

Para el grupo mostrado en la Tabla 4, al aplicar la estrategia de micro agrupamiento las secuencias obtenidas son $\bar{P}' \times L \times \bar{S}' = \{CAAAAGG, CAAAATG\}$.

Sea C el conjunto de grupos obtenido por el macro agrupamiento y C' el nuevo conjunto de grupos que se construirá usando micro criterio. El algoritmo de micro agrupamiento es mostrado en el Algoritmo 4:

Algoritmo 4 Agrupamiento por micro criterio

```
micro criterion cluster(C)
input: C -> set of clusters fragments
output: C' -> set of sequences fragments
C' = ∅
for each cluster c ∈ C:
  L ← c.leader
  Compute P̄
  Compute S̄
  P̄' ← P̄
  S̄' ← S̄
  for each p ∈ P̄:
    if p is substring of t such us t ∈ P̄ − {p} then:
      P̄' ← P̄' − {p}
  for each s ∈ S̄:
    if s is substring of t such us t ∈ S̄ − {s} then:
      S̄' ← S̄' − {s}
  c' ← P̄' × L × S̄' = {pls|p ∈ P̄' ∧ l ∈ L ∧ s ∈ S̄'}
  C' ← c' ∪ C'
```

En esta fase se utiliza información de los prefijos y sufijos de las posibles mezclas de secuencias, por lo que si hay alguna repetición de una misma secuencia, la podemos separar en varias secuencias al reconocer elementos que no coincidan a principio o final de estos fragmentos. Por ejemplo, dados los *reads* obtenidos de la secuencia $CAAAAGGTTTCAAAATG$, el fragmento $AAAA$ está presente en dos posiciones diferentes en la secuencia, siendo entonces un *repeat*. Para los *reads* de la Tabla 4, usando los macro criterios y micro criterio de agrupamiento, es posible tratar el *read* $AAAA$ como un *repeat* y reconocer las dos posiciones donde aparece generándose las secuencias $CAAAAGG$ y $CAAAATG$ mostradas en el ejemplo anterior. Nótese que estas secuencias ya no son *repeats*.

4.2 Conclusiones

En este capítulo presentamos el algoritmo LPS de ensamble de secuencias de ADN. Este algoritmo está compuesto por dos fases de agrupamiento, un macro criterio donde se agrupan las secuencias que se empalman, y un micro criterio donde se generan las nuevas secuencias producto de las mezclas de las secuencias agrupadas en la fase de macro criterio. El algoritmo

LPS será combinado con el ensamblador PadeNA en vista de obtener un mejor ensamble. Los resultados obtenidos serán mostrados en el próximo capítulo.

5 EXPERIMENTOS Y RESULTADOS OBTENIDOS

En este capítulo mostraremos la calidad de dos ensambles obtenidos mediante la combinación de LPS + PadeNA y lo compararemos con el ensamble obtenido sólo con PadeNA.

Inicialmente definiremos los criterios de calidad utilizados para cuantificar la calidad de los ensambles. Seguidamente realizaremos los ensambles de diversas secuencias de ADN. Los ensambles se realizarán usando diferentes secuencias ordenadas de menor longitud a mayor longitud. Al terminar cada experimento se realizarán comentarios interpretando los resultados mostrados.

5.1 Evaluación de la calidad del ensamble

Para evaluar la calidad de un ensamble, lamentablemente no existen medidas estandarizadas (Narzisi & Mishra, 2011). La validación de un ensamblado es un proceso realizado manualmente y generalmente los genomas ensamblados son aceptados como una versión borrador. Por ejemplo, para el genoma humano, la primera versión del borrador (Lander et al., 2001) ha sido revisada varias veces y aún sigue considerada como una versión inexacta y con un número desconocido de errores.

No obstante existen algunas métricas populares para evaluar la calidad de un ensamblado sugeridas en (Miller, Koren, & Sutton, 2010) las cuales utilizaremos para reportar los resultados en la experimentación:

- Cantidad de *contigs* ensamblados: Un buen ensamble debe de minimizar el número de fragmentos obtenidos del proceso de ensamblado a partir de un conjunto inicial de fragmentos. En caso de obtener un solo fragmento en este proceso, es de esperarse que este fragmento sea la secuencia que se desea ensamblar.
- Suma de las longitudes de los *contigs*: Un buen ensamble debe de aproximar la suma de las longitudes de los *contigs* obtenidos a la longitud de la secuencia que se desea reconstruir. Mientras más cercano sea el valor de la suma de las longitudes de los *contigs* a la longitud de la secuencia, se considerará mejor el ensamble.

- Por ciento de error: Los fragmentos con que se realizarán los experimentos son obtenidos libres de errores. Dada esta observación consideraremos que un *contig* obtenido es correcto si este es subcadena de la secuencia original. En caso de que algún carácter del *contig* no forme parte de alguna subcadena de la secuencia original, se considerará como un *contig* erróneo. La medida de error que se utilizará será por ciento de error. El por ciento de error se mide como el porciento de la cantidad de *contigs* erróneos sobre el total de *contigs* obtenidos. Mientras más pequeño sea este valor, mejor será el ensamble.
- Longitud del c*ontig* más largo: Es la longitud de la secuencia ensamblada más larga. Idealmente se desea obtener una sola secuencia con la longitud de la secuencia de referencia.
- Promedio de la longitud de los *contigs*: Es el promedio de las longitudes de los fragmentos ensamblados. Esta medida en conjunto con la longitud del *contig* más largo nos da idea de cuán grandes son los fragmentos encontrados.
- N50: Es la longitud del *contig* más pequeño del conjunto de *contigs* más largos cuya suma de longitudes es al menos el 50% de la suma de las longitudes de todos los *contigs* (Miller et al., 2010). El tamaño del N50 se calcula ordenando todos los *contigs* de mayor a menor y determinando el conjunto mínimo de todos los *contigs* cuyo tamaño sea el 50% de la suma de las longitudes de todos los *contigs*. El N50 es una medida de la media de las longitudes de un conjunto de secuencias, con mayor peso dado a las secuencias más largas. Un ensamblado se considerará más útil para ser analizado por humanos si este tiene *contigs* con longitudes más grandes que la media de las longitudes (Bradnam et al., 2013).

En los experimentos mostrados a en la presente tesis tomamos secuencias de la base de datos del National Center for Biotechnology Information (NCBI). NCBI contiene una base de datos de secuencias de ADN de libre acceso llamada GenBank. En GenBank se guardan las secuencias producidas en laboratorios de todo el mundo y se ha convertido en la base de datos más importante para la investigación en el campo biológico. En los experimentos se mostrarán cinco casos de estudio.

El orden de realización de estos experimentos está dado por la longitud de la secuencia que se desea ensamblar, siendo el primer experimento la secuencia de menor longitud y el último

experimento la secuencia de mayo longitud. Se mostrará el comportamiento de LPS combinado con PadeNA considerando la longitud de la secuencia que se desea ensamblar.

En cada experimento los *reads* fueron obtenidos mediante una simulación en la computadora. A este proceso de simulación con una computadora se le llama *in silico*.

5.2 Experimento 1 Ensamble del plásmido Bacillus cereus ATCC 14579 plasmid pBClin15

En este experimento usaremos a LPS con una secuencia del plásmido Bacillus cereus ATCC 14579 plasmid pBClin15, disponible en Accession NC_004721.2 (Ivanova et al., 2003). El tamaño de esta secuencia es de 15,274 pares de bases, siendo una secuencia pequeña. Esta fue secuenciada sintéticamente *in silico* en diez ocasiones con cobertura de diez veces la molécula completa. Para cada secuenciación, se realizó el proceso de ensamblado y se reportan los resultados utilizando las medidas de calidad expuestas previamente.

En la simulación de la secuenciación se crearon fragmentos de secuencias de longitud uniforme de 50 caracteres, considerando una sola dirección de izquierda a derecha (de 3’ a 5’), libre de errores, y todos los caracteres de las secuencias del plásmido están contenidas en al menos un fragmento (cobertura total). El número de fragmentos obtenidos en cada secuenciación oscila en el rango de 6,000 a 6,500 *reads*.

Primeramente, realizaremos un ensamble usando PadeNA solamente. Los resultados obtenidos por PadeNA serán utilizados como criterio base de comparación para evaluar la calidad de la propuesta de la tesis que es inicialmente aplicar LPS previo a PadeNA. PadeNA es un algoritmo de ensamble basado en el uso de grafos de de Bruijn. El parámetro fundamental de estos algoritmos es utilizar un valor de *k* para especificar el tamaño de los *k*-mers. En el próximo epígrafe explicaremos cómo buscamos un valor de *k* que brinde un buen ensamble contra el cual nos compararemos utilizando como criterio las medidas de evaluación explicadas previamente.

5.2.1 Encontrando el tamaño de los *k*-mers para PadeNA

En este epígrafe se buscará un valor de *k* para determinar cuál longitud de los *k*-mers arroja un buen resultado. Los resultados obtenidos por el valor de *k* seleccionado serán utilizados como base de la comparación del algoritmo propuesto de LPS combinado con PadeNA.

Para buscar el valor de *k*, tomamos las 10 secuenciaciones obtenidas del plásmido Bacillus cereus. Para cada una de estas secuenciaciones se realizan ensamblados usando diferentes valores de *k* admisibles, los cuales están en el rango de 2 a 31. Para cada ensamblado calculamos los valores de calidad descritos previamente. Finalmente, para reportar los resultados, promediamos los valores de calidad de cada uno de los diez ensamblados.

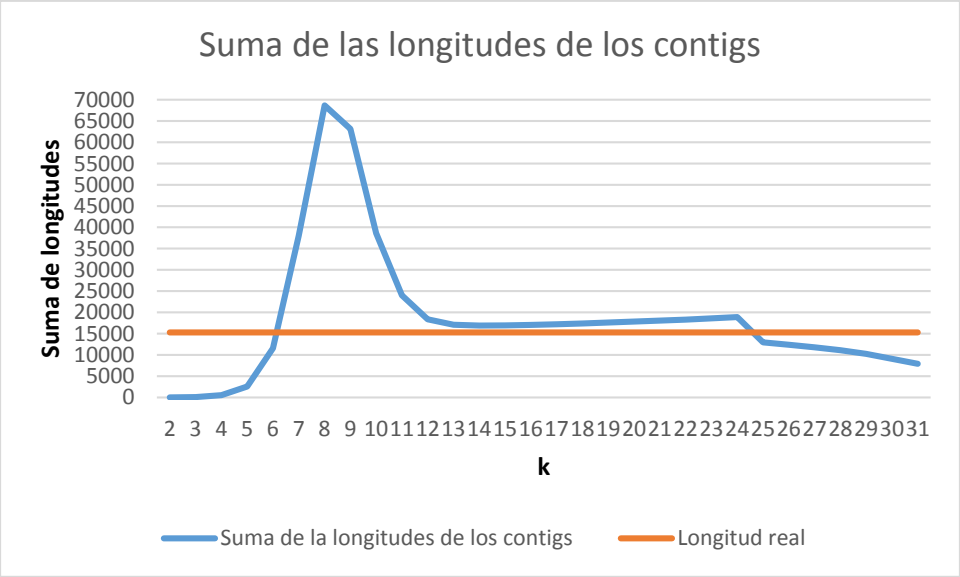

Fig. 14 Suma de las longitudes de los contigs con PadeNA

La Fig. 14 muestra en azul la suma total de los *contigs* puestos uno detrás del otro. El valor ideal a alcanzar es el que se aproxime al valor real de la longitud de la secuencia original mostrado en rojo el cual es 15,274. Los valores de *k* que dan un ensamble con una medida que se aproxima a este valor están en el rango de 13 a 24, siendo este el intervalo de valores de *k* el que nos interesa y se encuentra entre 13 y 24.

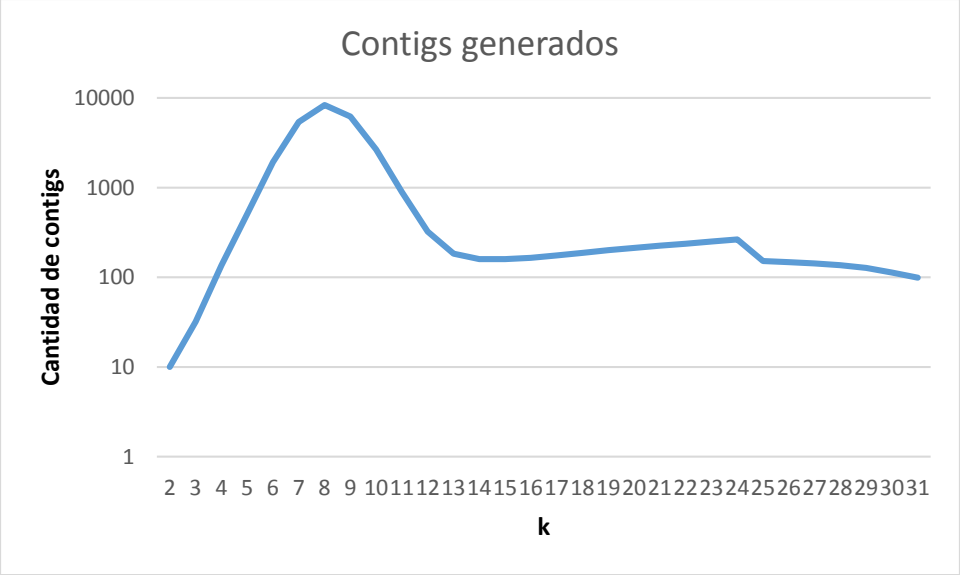

Fig. 15 Contigs generandos con PadeNA

La Fig. 15 contiene la cantidad de *contigs* generados con PadeNA utilizando los diferentes valores de *k*. La tabla muestra los valores en el eje de las abscisas utilizando una escala logarítmica. El resultado ideal es obtener un solo *contig* con una longitud igual a la secuencia original. Observando la Fig. 14 y la Fig. 15 vemos que para valores de *k* entre 2 y 5 se obtienen pocos fragmentos pero la suma de las longitudes está muy por debajo de la longitud de la secuencia original, por lo que estos valores de *k* tan pequeños son descartados. Para valores de *k* en el intervalo de interés comprendido entre 13 a 24 se observa que se obtiene una cantidad de *contigs* no tan elevada como los encontrados en el intervalo de *k* entre 6 y 12. Dado este comportamiento este intervalo sigue siendo candidato para obtener un valor de *k* con el que se obtenga una referencia para comparar con LPS combinado con PadeNA. Para valores de *k* de 25 en adelante, la cantidad de *contigs* disminuye pero podemos observar que la suma de las longitudes de los *contigs* disminuye por debajo de la longitud de la secuencia original siendo esto un síntoma que se están quedando regiones sin poder reconstruir de la secuencia original.

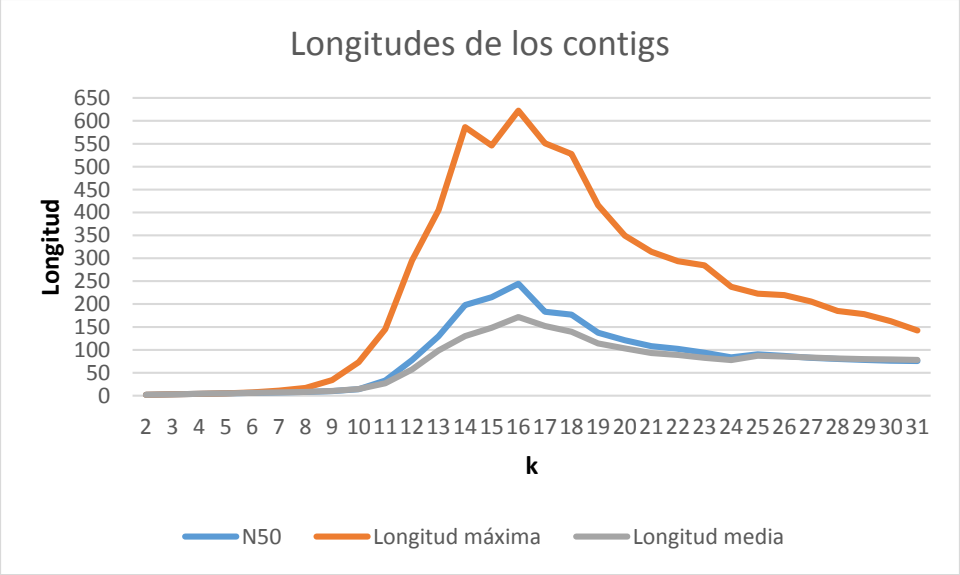

Fig. 16 Medidas de N50, Longitud del contig más largo y longitud media de los contigs con PadeNA

La Fig. 16 muestra los valores de las longitudes de los *contigs* usando las medidas N50, longitud del *contig* más largo y la longitud media de todos los *contigs*. Un ensamble que pueda ser interpretado por humanos necesita que los *contigs* encontrados tengan una longitud larga por lo que los valores de *k* serán los que muestren mayor valor de estas medidas. El intervalo que cumple con esto está comprendido en el rango de valores de *k* entre 13 y 23, reduciendo un poco más el rango de valores de *k* candidatos para el ensamble contra el cual se realizará la comparación.

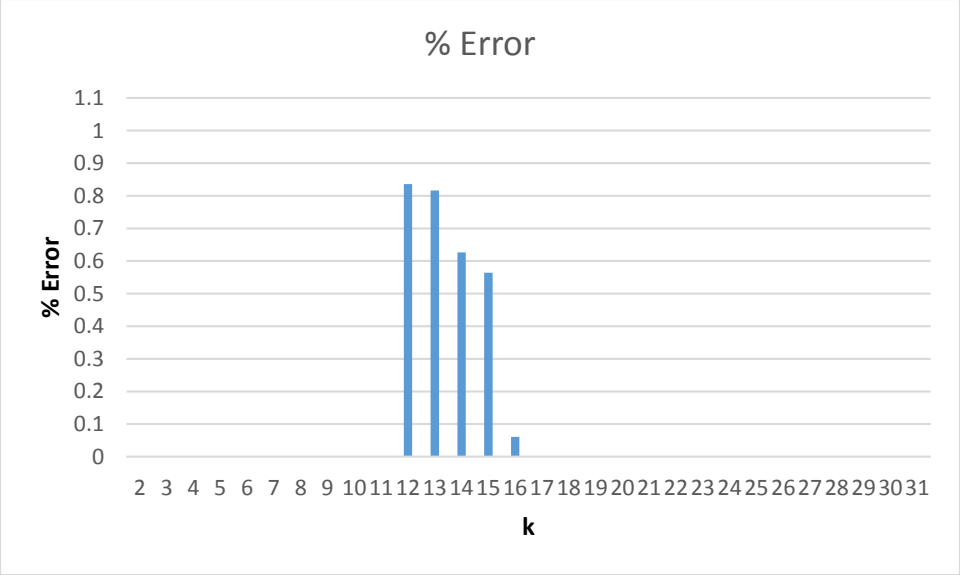

Fig. 17 Por ciento de error con PadeNA

La Fig. 17 muestra los errores en los ensambles. En el intervalo candidato de valores de *k* comprendido entre 13 y 20 vemos que el primer valor de *k* donde el error es 0% es para *k* = 17 por lo que tomamos este valor de *k* para obtener el ensamble de referencia y comparar la calidad de los ensambles obtenidos con LPS combinado con PadeNA.

5.2.2 Ensamblado con LPS

Primeramente, antes de entrar a la combinación de LPS con PadeNA mostraremos el comportamiento de LPS por sí solo. En este epígrafe mostraremos la evaluación de los ensambles utilizando las medidas de calidad descritas en el epígrafe 5.1 Evaluación de la calidad del ensamble.

Para estos experimentos tomamos como valor n de solapamiento entre caracteres a $n \in \{5,10,15,20,25,30,35,40,45\}$.

Tabla 5 Evaluación de la calidad de LPS

n	5	10	15	20	25	30	35	40	45
Total de contigs	281.3	**261.4**	313.5	346.1	383.2	425.6	478.8	538.1	609.5
Contigs correctos	188.1	261.2	313.5	346.1	383.2	425.6	478.8	538.1	609.5
% error	33.13	0.20	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
Suma long de contigs	30174.2	23418.4	**22379.7**	22551.2	23074.2	23983.4	25578.6	27731.8	30722
N50	**111.7**	91	76.1	68.9	61.4	52.6	50.8	50.3	50
Long máxima	**139**	126.5	113.8	102.9	92.5	83	71.8	60.9	55.4
Long media	**107.18**	89.61	71.37	65.15	60.19	56.31	53.35	51.47	50.38

La Tabla 5 muestra los valores de la calidad de los ensambles obtenidos con LPS para diferentes valores de *n*. Los valores en letra oscura muestran los mejores resultados así como los valores subrayados muestran los peores resultados. Para valores de $n = 5$ vemos que el por ciento de error es muy alto comparado con el del resto de los ensamblados obtenidos por lo que podemos descartar este ensamble. Para el valor de $n = 45$ obtenemos el ensamble con mayor suma de longitudes por lo que este ensamble también lo descartamos. Para $n = 15$ se obtuvo un ensamblado sin errores y con menor suma de longitudes de los *contigs*, adicionalmente tiene el segundo mejor para el total de *contigs*, el N50, la longitud del *contig* más largo y la longitud media de los *contigs*, es por ellos que tomamos los valores de calidad de los ensambles obtenidos con LPS usando $n = 15$ y los compararemos con la calidad de los ensambles obtenidos con PadeNA con valor de parámetro $k = 17$. En la Tabla 6 se muestran los valores de calidad de PadeNA para $k = 17$ los cuales superan a los de LPS con $k = 17$.

Tabla 6 Evaluación de la calidad del ensamblado con PadeNA para k = 17

	Total de contigs	Contigs correctos	% error	Suma long de contigs	N50	Long máxima	Long media
PadeNA K = 17	175.7	175.7	0	17199.4	183.2	551	152.02

Valores bajos en las longitudes máximas, N50 y media de LPS comparado con PadeNA son de esperarse debido a que LPS mezclará secuencias donde la longitud máxima esperada de la nueva secuencia es $3 * long\ de\ read - 2 * n$, donde $long\ de\ read$ es la longitud de los *reads*, que para este caso es 50. Esta cota se calcula usando la longitud de 3 *reads*, el prefijo, el líder y el sufijo. A este valor se le restarán los caracteres de solape, entre el prefijo con el líder, y entre el líder con el sufijo. Es de esperarse que con *contigs* cortos se obtengan una gran cantidad de estos por lo que la suma de todos ellos también sea alta.

Es por ello que se requerirá mejorar la calidad del ensamblado obtenido usando LPS. Una variante es volver a aplicar LPS al ensamble obtenido previamente. En lo adelante usemos como notación LPS 1 para la primera corrida de LPS y LPS 2 a la segunda corrida de LPS aplicada al resultado obtenido por LPS 1. La combinación del resultado obtenido mediante las dos corridas de LPS con PadeNA las denotaremos como LPS + PadeNA.

Los esquemas considerados para aplicar LPS en una segunda corrida usando como parámetro *n* para la cantidad de caracteres mínimos para solape son:

- *n* caracteres para LPS 1 - *n* caracteres para LPS 2: Para este experimento fijamos un valor de *n* para la primera corrida de LPS como para una segunda corrida. Los valores de *n* a probar son $n \in \{5,10,15,20,25,30,35,40,45\}$.
- *n%* para LPS 1 - *n%* para LPS 2: En este caso en lugar de usar una cantidad fija *n* de caracteres para el solape, se usará un porcentaje de la longitud del *read* más pequeño. De esta manera se tomarán en cuenta las nuevas longitudes de las secuencias obtenidas al aplicar una vez LPS para su segunda corrida. Los valores de *n%* a probar son $n \in \{10\%, 20\%, 30\%, 40\%, 50\%, 60\%, 70\%, 80\%, 90\%\}$.
- *n* caracteres para LPS 1 – 20% para LPS 2: En este se usará una cantidad *n* de caracteres para el solape en la primera corrida de LPS y luego se utilizará el 20% de la longitud del *read* más pequeño para la segunda corrida de LPS. El valor del 20% fue fijado debido a que con este parámetro de LPS se obtuvieron los ensambles con mejor calidad. Esta se sustenta con los resultados mostrados con los epígrafes 5.2.3 y 5.2.4.

Como hemos visto, si usamos una segunda pasada de LPS, la longitud de los *contigs* generados será menor que los obtenidos por PadeNA por si solos, es por ello que proponemos realizar la combinación de dos corridas de LPS con PadeNA y comparar los resultados obtenidos con PadeNA por sí solo.

5.2.3 Ensamblado con *n* caracteres para LPS 1 - *n* caracteres para LPS 2

En este esquema se utilizará la misma cantidad de caracteres de solape para ambas corridas de LPS. La calidad de estos ensambles está reflejada en las Fig. 18 - Fig. 23. Como convenio, tomamos que los resultados mostrados en color amarillo corresponden a la primera corrida

de LPS, los de color rojo a PadeNA, los de color verde a la segunda corrida de LPS, en color azul los obtenidos con las dos corridas de LPS + PadeNA y la línea gris representa la longitud de la secuencia original. La Fig. 18 correspondiente al porciento de error no muestra el cometido por PadeNA ya que el ensamble seleccionado tiene 0% de error.

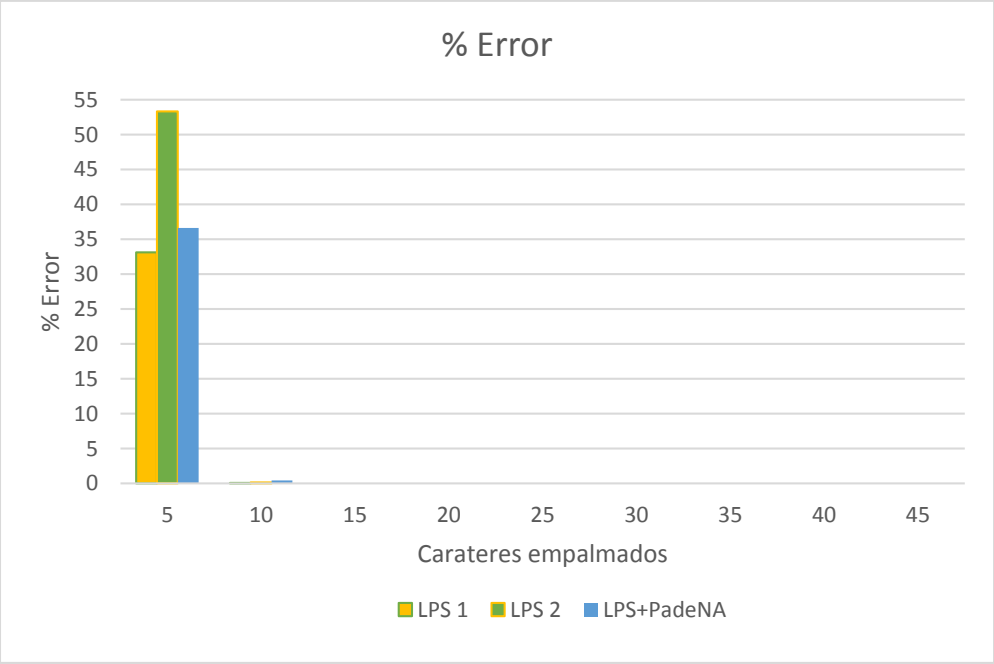

Fig. 18 Por ciento de error

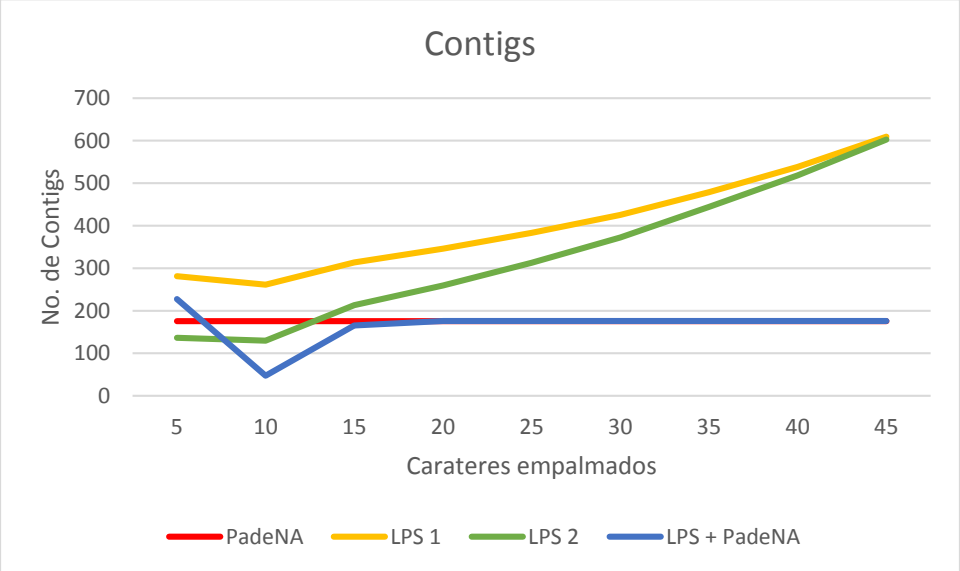

Fig. 19 Número de contigs

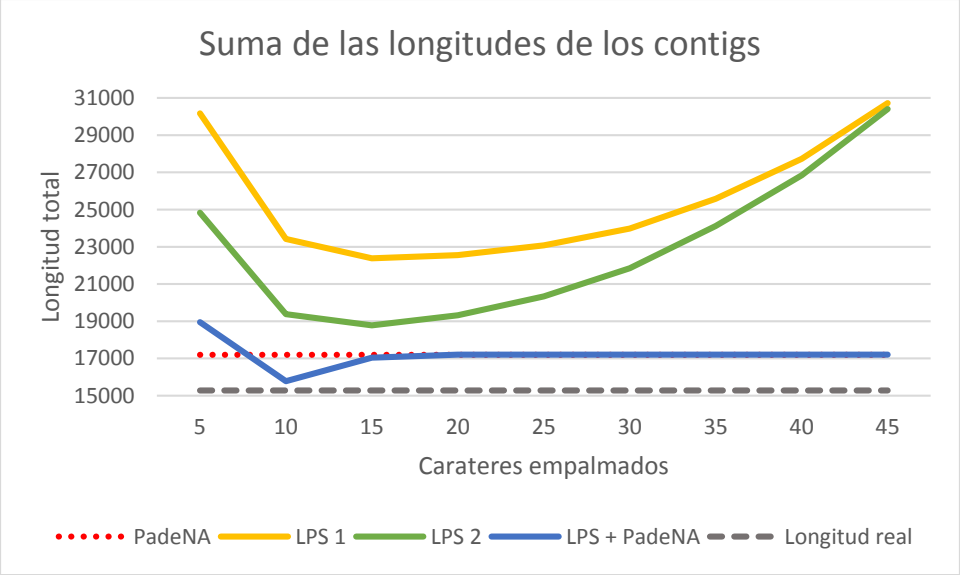

Fig. 20 Suma de las longitudes de los contigs

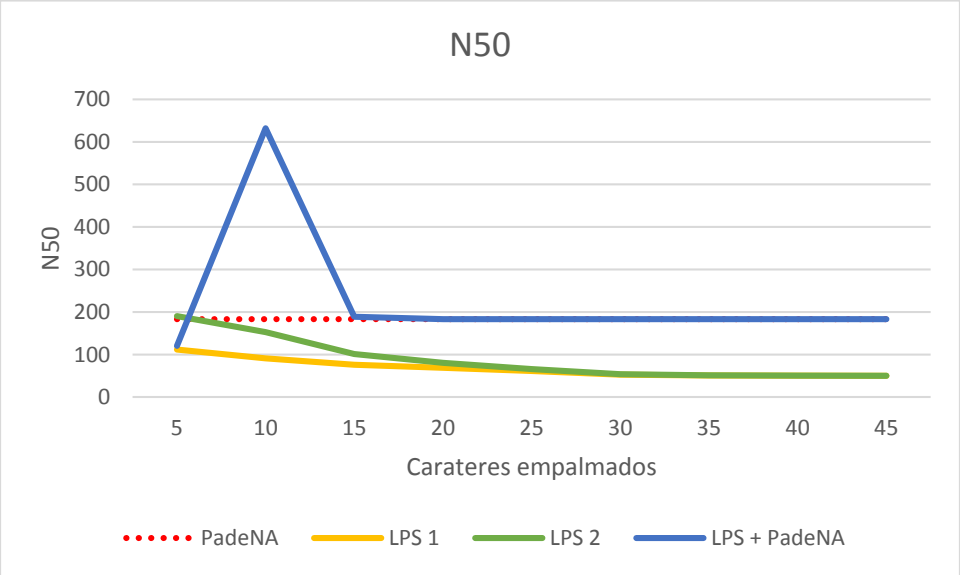

Fig. 21 N50

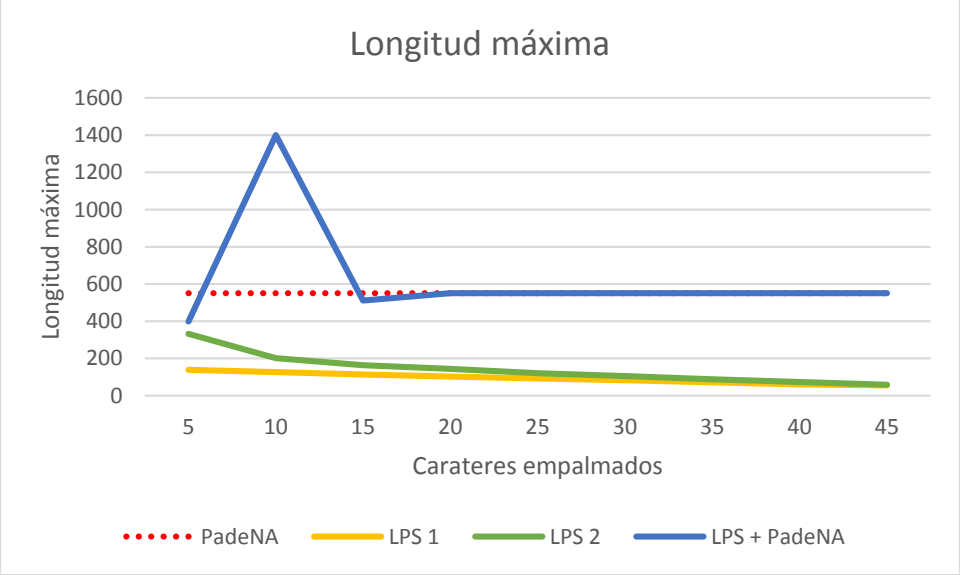

Fig. 22 Longitud del contig más largo

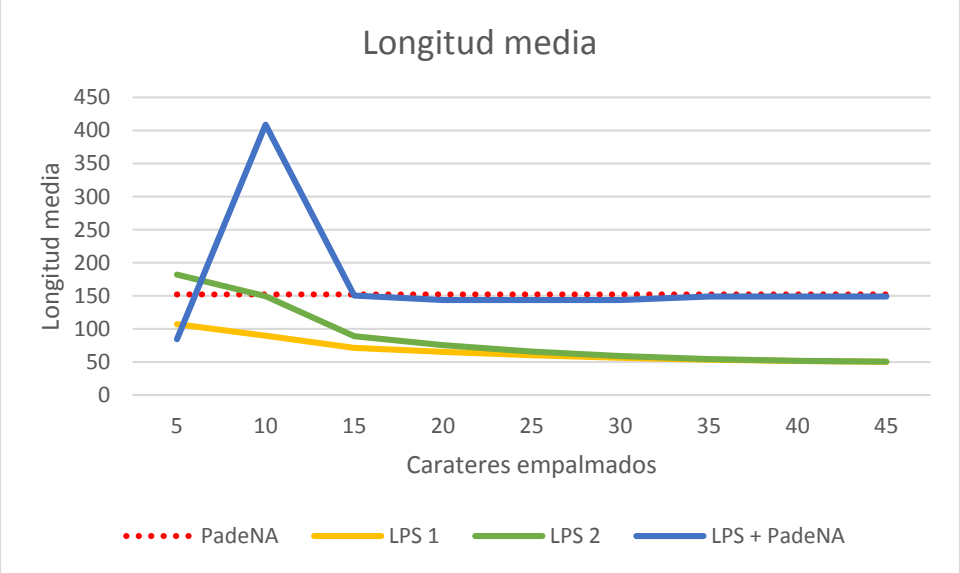

Fig. 23 Longitud media de los contigs

La Fig. 18 muestra los errores cometidos por las dos corridas de LPS y por la combinación de LPS con PadeNA para diferentes valores de *n*. El valor del error de PadeNA contra el cual compararemos es 0% según habíamos seleccionado anteriormente. Es notable que para *n* = 5 se obtiene un por ciento de error muy elevado. Esto es debido a que como se exigen muy pocos caracteres para el solape, es muy probable que al mezclar secuencias que se solapen se produzcan nuevos fragmentos que no aparezcan en el genoma original. Es por ello que

descartaremos este valor para comparar con PadeNA. Para el valor del solape *n* = 10, el porcentaje de error es menor de 1% considerándose un error bajo. Para valores mayores de *n* el porcentaje de error es 0% siendo también estos ensambles aceptables.

La Fig. 19 muestra la cantidad de *contigs* encontrados. La línea horizontal roja muestra los *contigs* encontrados con PadeNA. Para el valor de *n* = 10, la cantidad de *contigs* obtenidos mediante la segunda corrida de LPS es inferior que los obtenidos por PadeNA, si observamos los valores de la Fig. 21, la Fig. 22 y la Fig. 23 vemos que las longitudes de los *contigs* son menores que los obtenidos con PadeNA, sin embargo, la suma de las longitudes en la segunda corrida de LPS es mayor que con PadeNA (Fig. 20), mostrando que todavía existe algún solape entre fragmentos que puede ser resuelto. Es por ello que finalmente aplicamos PadeNA a estos fragmentos. Para este valor podemos observar que la cantidad de *contigs* obtenidos se reduce en más de la mitad comparado con PadeNA. La suma de las longitudes de los *contigs* también se aproxima a la longitud de la secuencia de ADN original mejorando a PadeNA. También las medidas de la longitud como el N50, la media de las longitudes y longitud del *contig* más largo superan por más los resultados de PadeNA. Para valores de *n* a partir de 15 en adelante, la calidad de los ensambles obtenidos con LPS comienza a empeorar. Esto se debe a que mientras más caracteres se exijan para empalmarse, más corta será la secuencia resultante de la mezcla, además existirán solapes correctos pero con un valor menor al establecido a *n* que serán desechados. Sin embargo, si aplicamos PadeNA, podemos superar esta dificultad presentada por LPS y obtener ensambles con una calidad mejor o similar a los obtenidos por PadeNA solamente debido a que con PadeNA podemos obtener secuencias con una mayor longitud que la de los *reads* iniciales y por tanto es posible desambiguar la ocurrencia de *repeats*.

Para esta configuración de valores de solape, el mejor ensamble obtenido fue utilizando 10 caracteres. El ensamble obtenido con LPS + PadeNA mejora la calidad de PadeNA en casi dos veces para los valores de número de *contigs* (Fig. 19), N50 (Fig. 21), longitud máxima del *contigs* más largo (Fig. 22) y longitud media de los *contigs* (Fig. 23). Para valores de *n* a partir de 15 caracteres en adelante los ensambles obtenidos con LPS + PadeNA son similares a PadeNA, por lo que no se pierde en calidad. La ventaja para estos casos de aplicar LPS es que reducimos la cantidad de fragmentos de secuencias con los que trabajará PadeNA.

5.2.4 Ensamblado con *n*% caracteres para LPS 1 – *n*% caracteres para LPS 2

A diferencia del esquema anterior en el que se utiliza un número fijo de caracteres mínimos para el solape, en este esquema se utilizará un determinado por ciento de la longitud del *contig*. Este esquema se propone para cuando se realice la segunda corrida de LPS, el solape se considera respecto a la longitud de los fragmentos recién creados y que la elección del solape no sea un número fijo. Los valores de porcentaje n que se analizarán son $n \in \{10,20,30,40,50,60,70,80,90\}$. Obsérvese que para la primera corrida de LPS estos valores de porcentaje coinciden con los valores de solape de $\{5,10,15,20,25,30,35,40,45\}$ caracteres para *reads* de longitud 50.

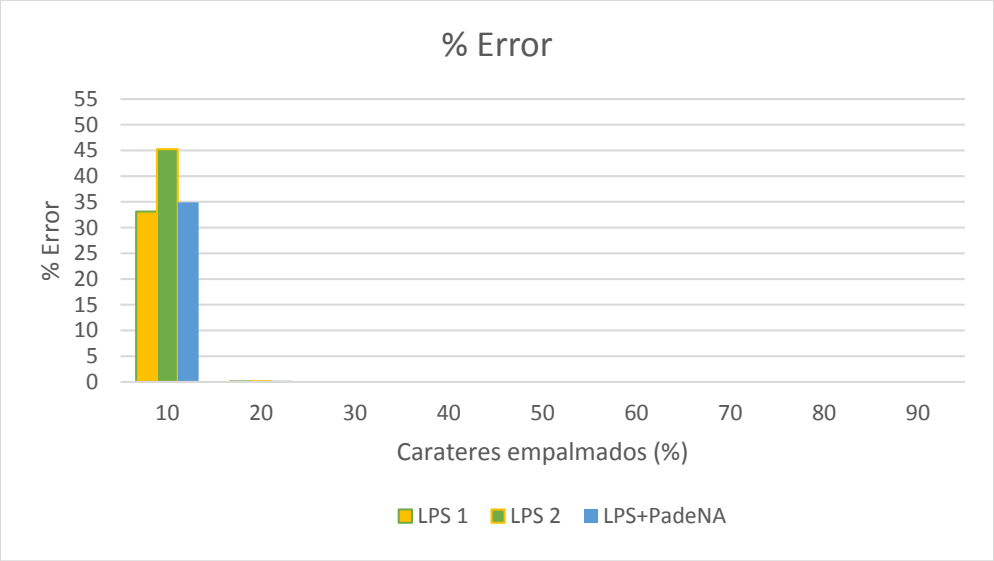

Fig. 24 Por ciento de error

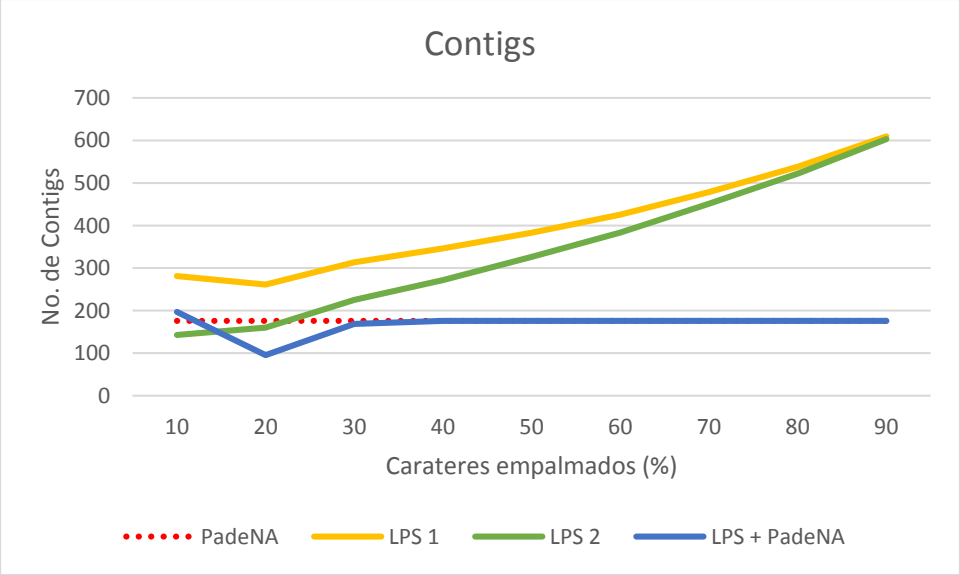

Fig. 25 Número de contigs

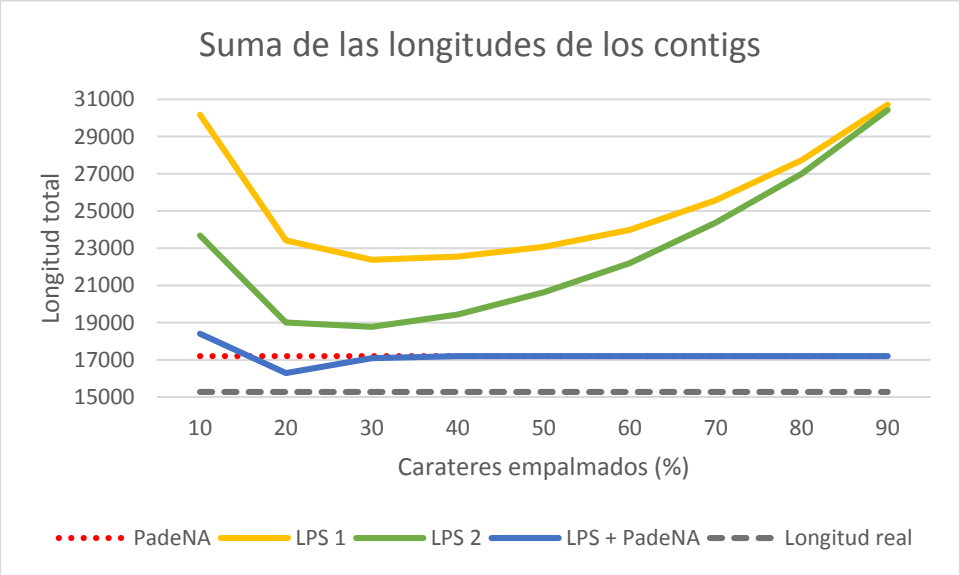

Fig. 26 Suma de las longitudes de los contigs

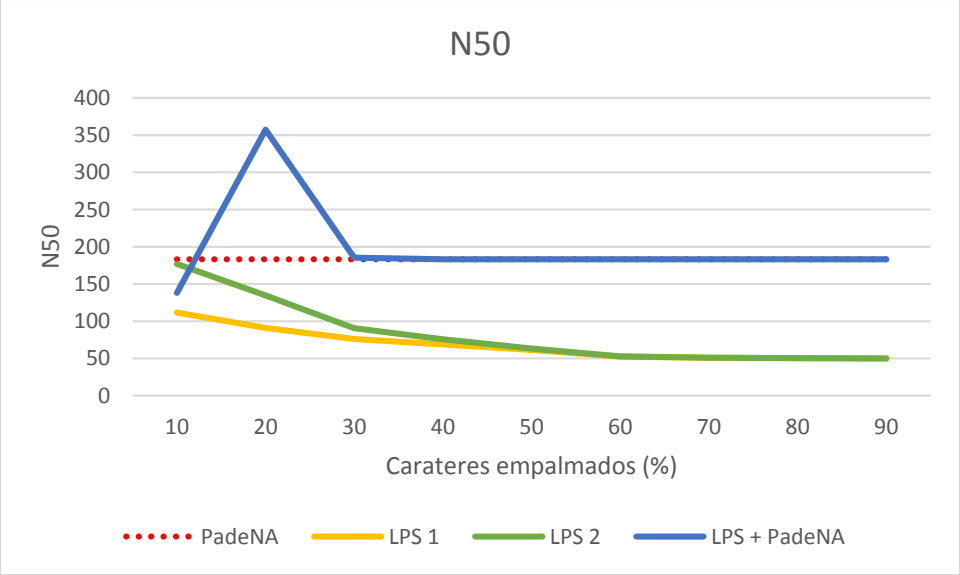

Fig. 27 N50

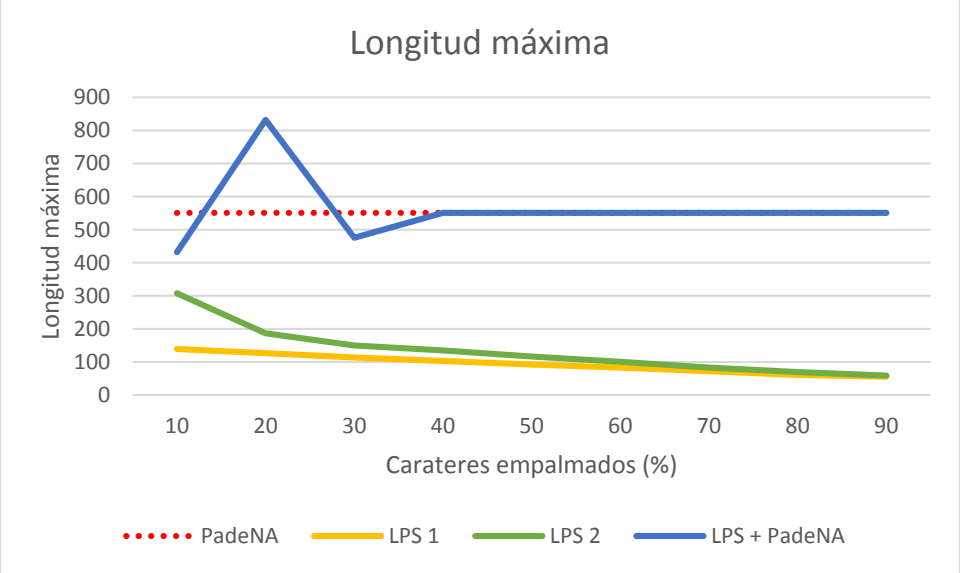

Fig. 28 Longitud del contig más largo

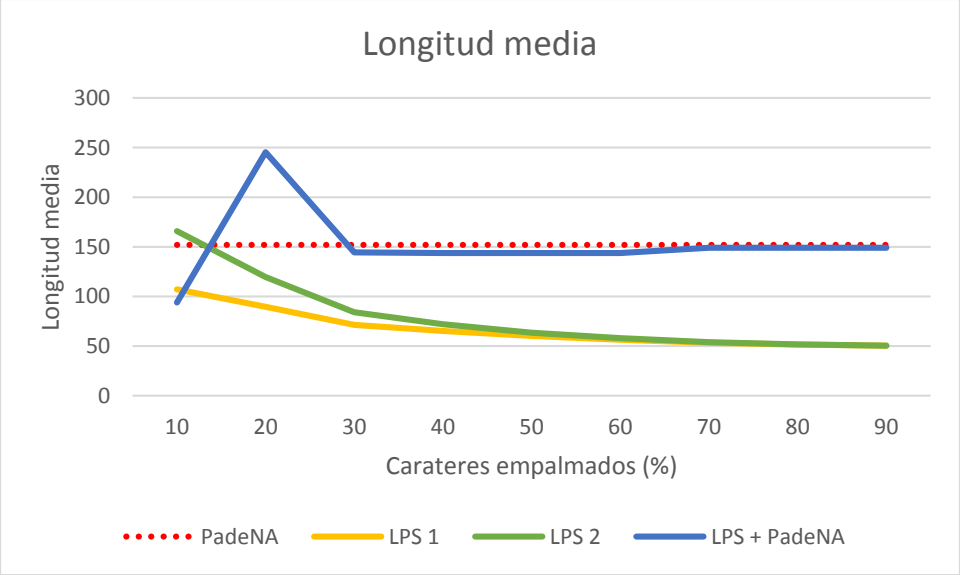

Fig. 29 Longitud media de los contigs

La Fig. 24 muestra los errores cometidos por las dos corridas de LPS y LPS + PadeNA. El valor de error cometido por PadeNA es 0% por lo que no se muestra. Para *n* = 10%, vemos que los ensambles obtenidos tienen un alto por ciento de error por lo que esta solución la podemos desechar. Para *n* = 20% el error cometido es menor al 1% siendo muy bajo lo cual pudiera considerarse como una solución candidata para ser aceptada. Para *n* = 30% en adelante el error cometido es de 0%. Siendo también estos ensambles candidatos a ser una solución.

La Fig. 25 muestra la cantidad de *contigs* encontrados. Para el valor *n* = 20%, la cantidad de *contigs* obtenidos en la segunda corrida de LPS es menor que con LPS, sin embargo, si observamos las Fig. 26 - Fig. 29 los resultados obtenidos por PadeNA mejoran a la segunda corrida de LPS. Para LPS la suma de la longitud de los *contigs* es superior y las longitudes de los *contigs* es menor que PadeNA. Similarmente al comportamiento de cuando solo se usaba una cantidad fija de *n* elementos para el solape, estos valores muestran que existen para los *contigs* obtenidos con *n* = 20% solapes entre ellos. Es por ello que aplicamos PadeNA después de LPS para mejorar la calidad del ensamble.

El ensamble por LPS + PadeNA muestra que tiene menos *contigs* que los encontrados por PadeNA reduciéndolos a más de la mitad. Los resultados de las Fig. 26 - Fig. 29 confirman

que la calidad de este ensamble supera a PadeNA. La suma de las longitudes de los *contigs* muestra que este valor se acerca más a la longitud de la secuencia original. Se puede observar un pico en los valores la medida N50, longitud del *contig* más largo y media de las longitudes de los *contigs*.

Para n = 30% los valores de calidad de los ensambles obtenidos por ambas corridas de LPS comienzan a empeorar. Este comportamiento se mantiene para los valores restantes de n. Para el caso de LPS + PadeNA, el promedio de las longitudes de los *contigs* es menor que con PadeNA, sin embargo si observamos los demás valores de calidad vemos que el ensamble es semejante al obtenido por PadeNA. Similarmente para valores de n superiores al 30% obtenemos ensambles similares a PadeNA combinando LPS + PadeNA.

Para esta configuración de valores de solape, el mejor ensamble obtenido fue utilizando el 20% de la longitud de los *reads*. El ensamble obtenido con LPS + PadeNA mejora la calidad de PadeNA en casi dos veces para los valores de número de *contigs* (Fig. 25), N50 (Fig. 27), longitud máxima del *contigs* más largo (Fig. 28) y longitud media de los *contigs* (Fig. 29). Para valores de n a partir del 30% en adelante los ensambles obtenidos con LPS + PadeNA son similares a PadeNA, por lo que no se pierde en calidad. La ventaja para estos casos de aplicar LPS es que reducimos la cantidad de fragmentos de secuencias con los que trabajará PadeNA.

5.2.5 Ensamblado con *n* caracteres para LPS 1 – 20% caracteres para LPS 2

En este esquema de experimentación se utilizarán en la primera etapa de LPS diferentes valores de n caracteres para el solape entre pares de secuencias donde $n \in \{5,10,15,20,25,30,35,40,45\}$ o lo que es equivalente en por ciento respecto a la longitud de los *reads* $n \in \{10,20,30,40,50,60,70,80,90\}$. Para la segunda fase tomaremos el porcentaje que mejor calidad de ensamble obtuvimos en los experimentos realizados en el epígrafe anterior. Este valor se alcanzó con el 20% de caracteres para el solape respecto a la longitud de la secuencia.

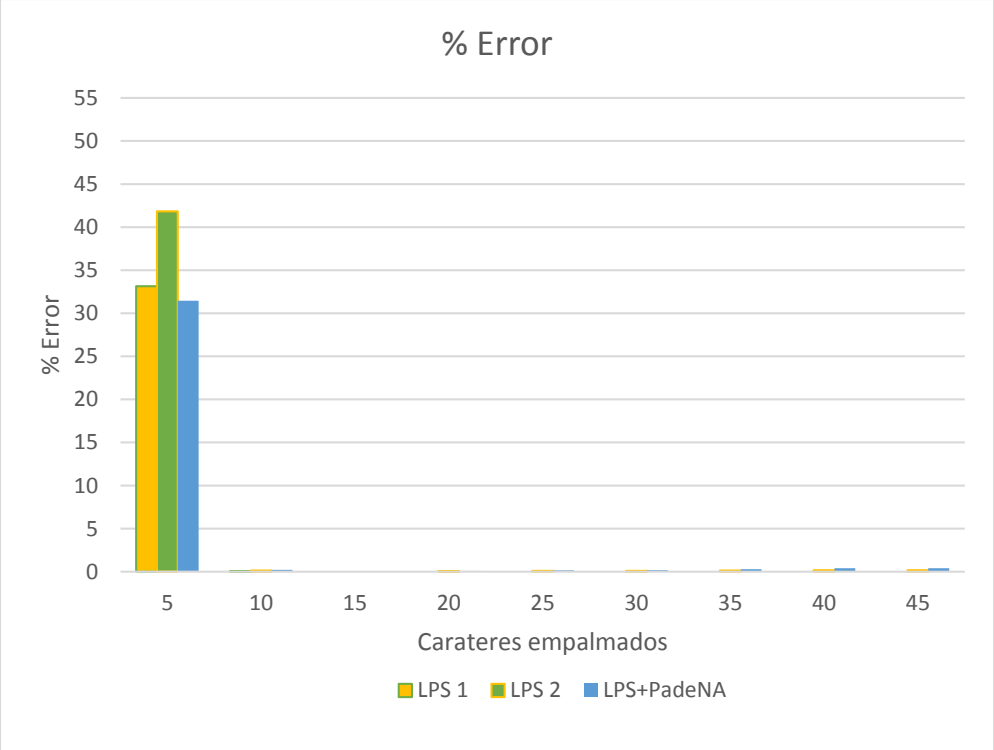

Fig. 30 Porciento de error

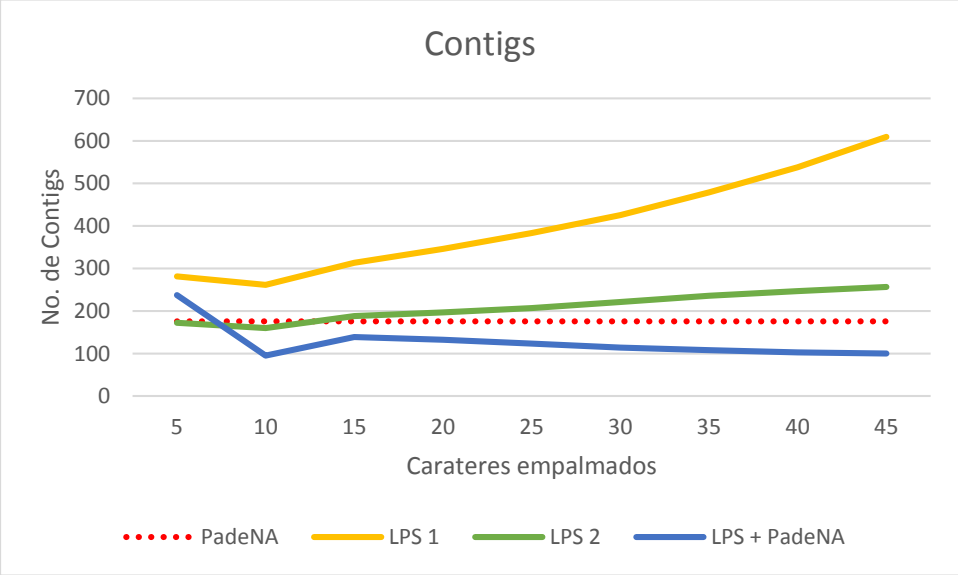

Fig. 31 Número de contigs

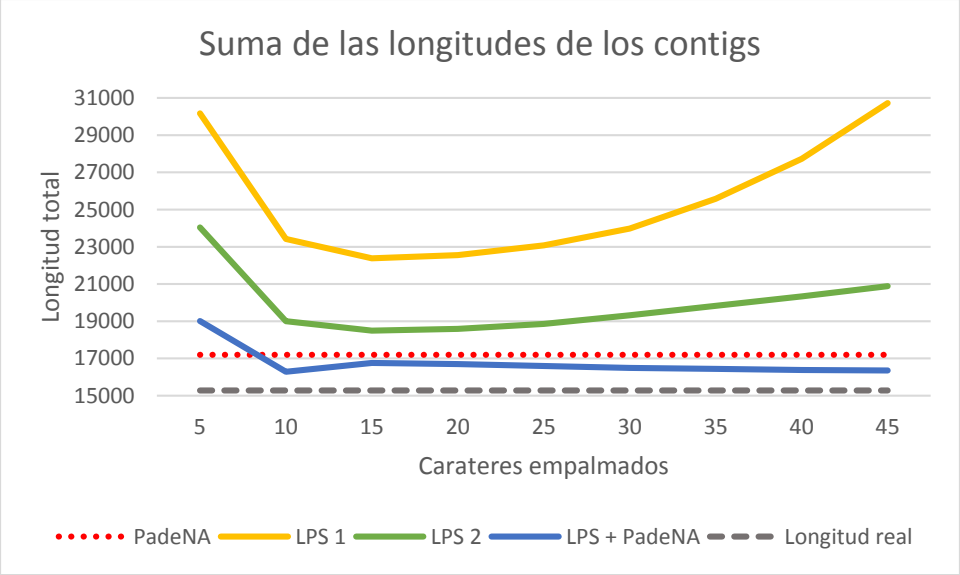

Fig. 32 Suma de las longitudes de los contigs

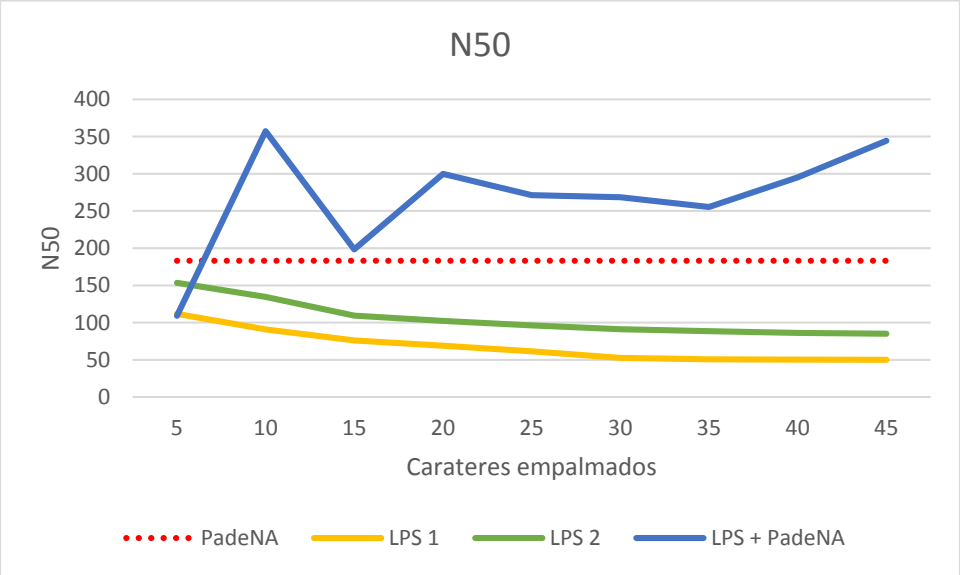

Fig. 33 N50

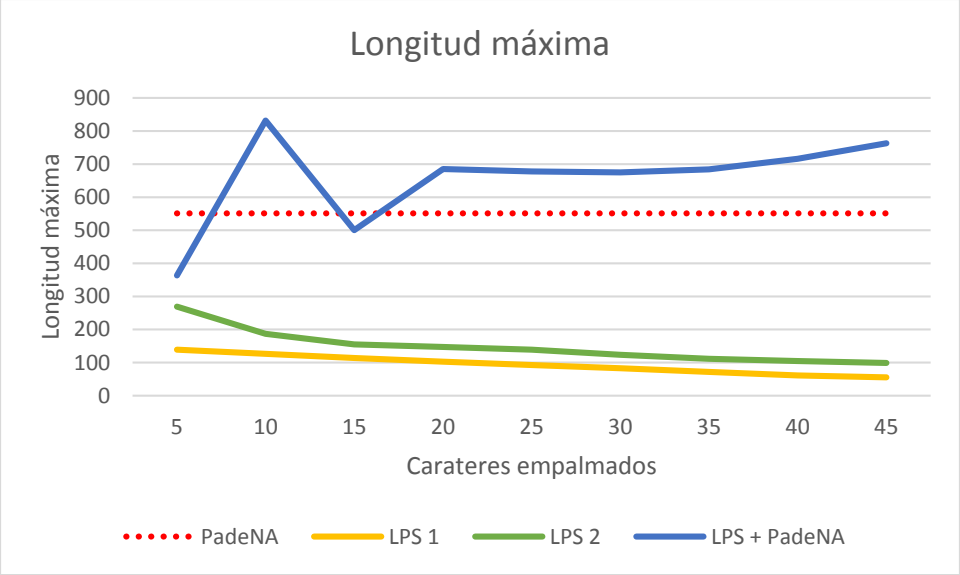

Fig. 34 Longitud del contig más largo

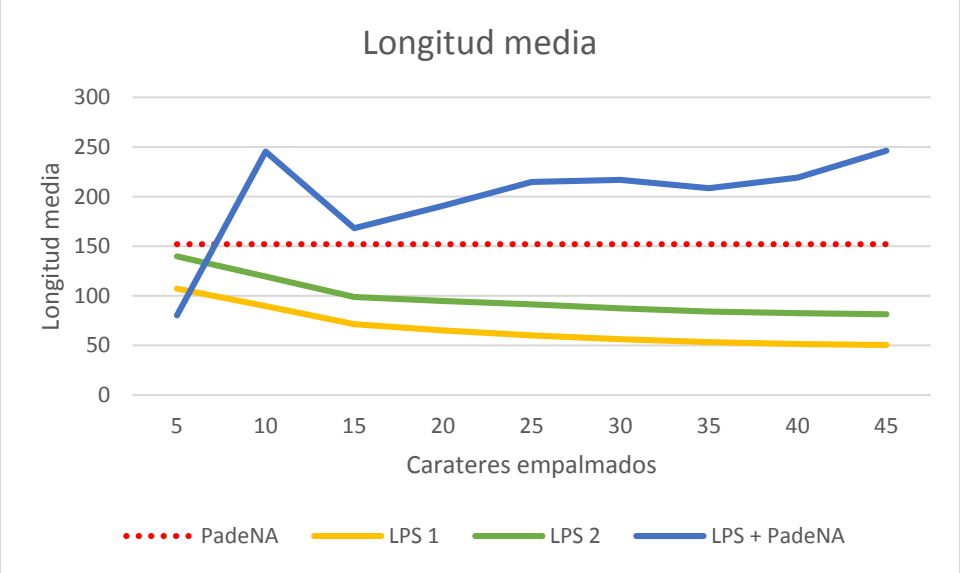

Fig. 35 Longitud media de los contigs

La Fig. 30 muestra el porciento de fragmentos ensamblados que son erróneos. El error cometido por PadeNA es 0% por lo que no es mostrado. Para la n = 5 se aprecia que los errores son muy elevados. Debido a ello descartamos el ensamble obtenido con este valor de n. Observemos que para valores de n = 20% en adelante, el error cometido es menor al 1% siendo posibles ensambles aceptables.

La Fig. 31 muestra la cantidad de *contigs* creados. Para el valor de n = 10 para la primera corrida de LPS + PadeNA, se obtiene que se reducen considerablemente los *contigs* comparados con PadeNA. La suma de las longitudes (Fig. 32) de los *contigs* también tiende a aproximarse a la longitud de la secuencia original mejorando el resultado de PadeNA. También las longitudes de los *contigs* de LPS + PadeNA es mayor que PadeNA (Fig. 33, Fig. 34, Fig. 35) mostrando que este ensamble es de mejor calidad.

Para el valor de n = 15, la calidad del ensamble obtenido por LPS + PadeNA decrece respecto a si usáramos el valor de n = 10, sin embargo este ensamble sigue siendo mejor que el obtenido por PadeNA. Solo el valor de la longitud media de los *contigs* baja respecto a PadeNA.

Como pudo observarse en las Fig. 31 - Fig. 35, para valores de n = 20 en adelante, la calidad de los ensambles obtenidos por LPS + PadeNA comienza a mejorar de nuevo sucesivamente, siendo siempre mejor que el ensamble obtenido por PadeNA. Si observamos la calidad del ensamble obtenido por la segunda corrida de LPS, la calidad de este es muy cercana al de PadeNA, pero no lo mejora. Aplicando a este ensamble PadeNA es que obtenemos la mejora mostrada.

5.2.6 Ensamblado con LPS + PadeNA utilizando las configuraciones anteriores

En este epígrafe solamente mostraremos a modo resumen los resultados obtenidos para las diferentes configuraciones de los caracteres de solapes entre pares de secuencias en los ensambles de LPS + PadeNA mostrados anteriormente. Este epígrafe solo constituye un resumen de las medidas de por ciento de error, cantidad de *contigs*, suma de las longitudes de los *contigs*, N50, longitud del *contig* más largo, longitud media de los *contigs*, tomadas en los epígrafes anteriores para los ensambles de LPS + PadeNA.

Las líneas horizontal color rojo muestran el valor obtenido por PadeNA. Las líneas color azul oscuro muestran los resultados obtenidos por el ensamble usando n caracteres en la primera corrida de LPS y n caracteres en la segunda corrida de LPS. Las líneas color azul claro muestran los resultados obtenidos por el ensamble usando $n\%$ caracteres en la primera corrida de LPS y $n\%$ caracteres en la segunda corrida de LPS. Las líneas color naranja muestran los

resultados obtenidos por el ensamble usando *n* caracteres en la primera corrida de LPS y 20% caracteres en la segunda corrida de LPS.

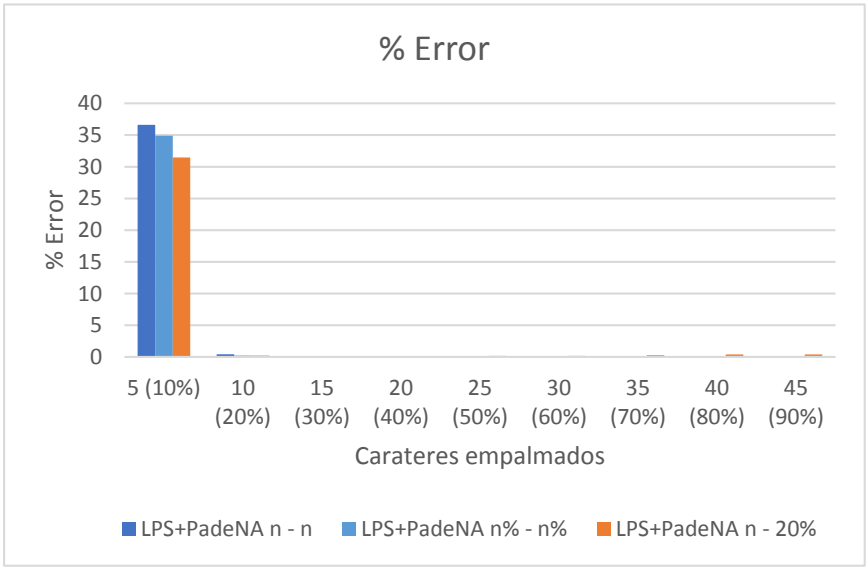

Fig. 36 Por ciento de error

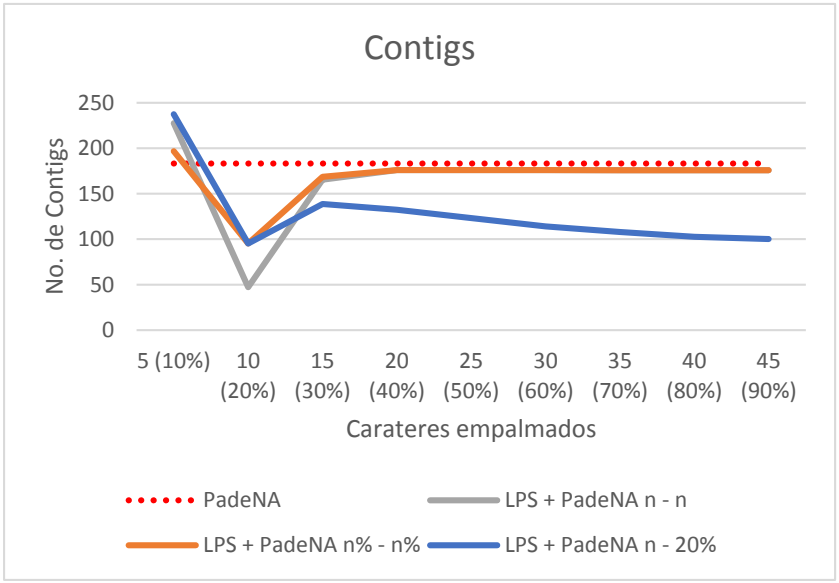

Fig. 37 Número de contigs

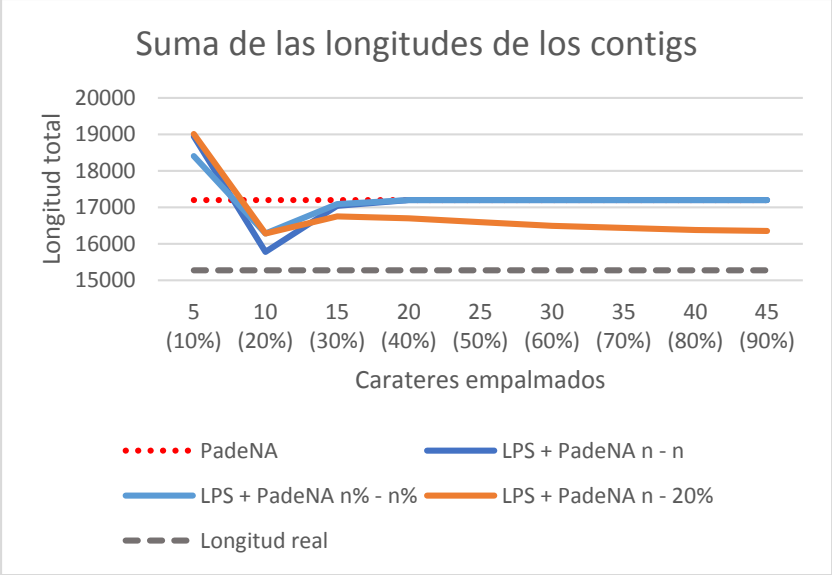

Fig. 38 Suma de las longitudes de los contigs

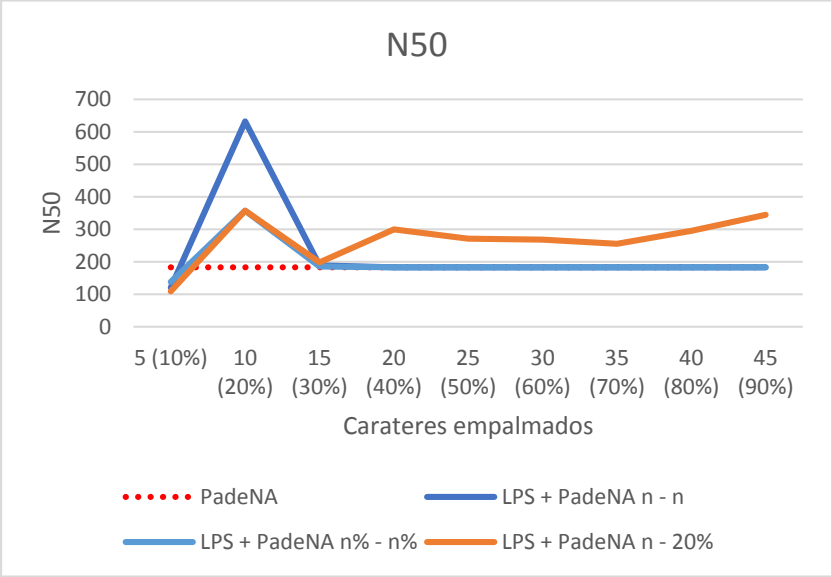

Fig. 39 N50

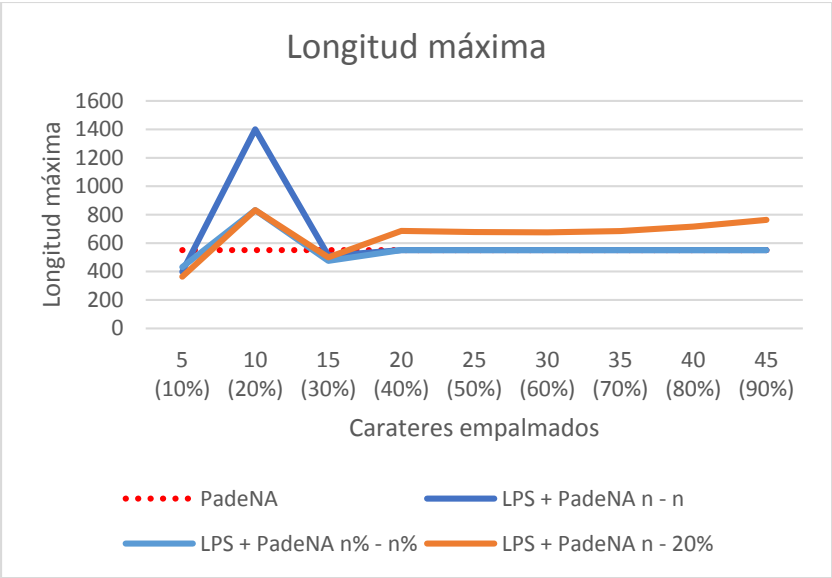

Fig. 40 Longitud del contig más largo

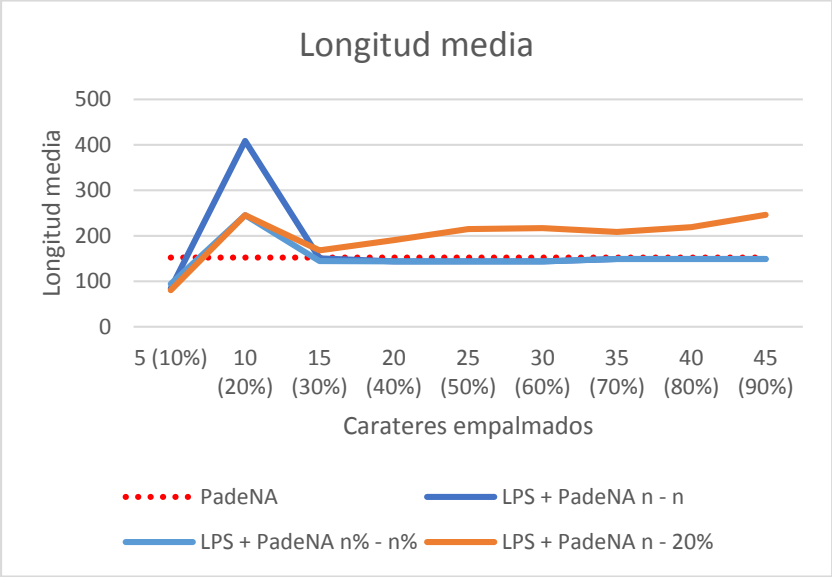

Fig. 41 Longitud media de los contigs

La Fig. 36 muestra los errores cometidos en los ensambles. El resultado de PadeNA no es mostrado ya que el error es de 0%. Para valores de *n* bajos, de 5 caracteres o 10% el por ciento del error cometido en el ensamble es muy elevado por lo que descartamos esos ensambles.

Para valores de *n* = 10 o 20%, para las tres combinaciones de ensamble, se obtienen los mejores valores de calidad. Esto se ve reflejado en las Fig. 37 - Fig. 41. La suma de las longitudes de los *contigs* se acerca a la longitud real de la secuencia original. La mejor combinación de valores obtenida para este caso es de 10 caracteres de solape para la primera corrida de LPS – 10 caracteres de solape para la segunda corrida de LPS. Estos resultados son alcanzados con un error menor de 1%. Para este caso se obtiene una buena mejoría debido a que la cantidad de caracteres necesarios para el solape entre secuencias es lo suficiente para crear nuevas secuencias a partir de las mezclas de este par empalmado tal que no sean erróneos y lo suficientemente para generar nuevas secuencias con longitudes más largas. Recordemos que el objetivo de aplicar LPS es generar secuencias de longitudes más largas para tratar los posibles *repeats*. A medida de que exijamos más caracteres para el solape, las nuevas secuencias creadas serán de longitud menor, pudiendo reducir en la calidad de ensamble. Esto puede ser visto para *n* = 15 o *n* = 30%. Para estos valores la calidad del ensamble decae de manera tal que es similar al obtenido a PadeNA solamente (Fig. 37 - Fig. 41). A partir de este valor en adelante para los esquemas de la misma cantidad de caracteres del solape para las dos corridas de LPS, así como el esquema con el mismo porcentaje, vemos que el ensamble tiene una calidad similar a PadeNA. Sin embargo, para el esquema de *n* caracteres en la primera corrida de LPS y 20% en la segunda corrida, a partir de *n* = 20 vemos que el ensamble mejora sucesivamente para los siguientes valores de *n* (Fig. 37 - Fig. 41). Esto implica que a pesar de exigir muchos caracteres para el solape en la primera corrida, si relajamos esta restricción para la segunda etapa, podemos construir secuencias de longitudes más largas que hagan que mejore la calidad del ensamble. A medida que incrementemos el valor de *n* la diferencia entre éste y la cantidad de 20% aumenta, resultando en secuencias de longitudes más largas y por tanto un mejor ensamble superando a PadeNA. A pesar de no poder superar a la calidad de los ensambles obtenidos usando el esquema de *n* = 10 o 20%, la opción de utilizar muchos caracteres en la primera corrida de LPS hace que se obtengan secuencias no tan extensas pero con pocos errores. Luego utilizando un valor bajo como el 20% para generar secuencias más largas con pocos errores, hace que este ensamble sea una solución que mejora el resultado obtenido por LPS. En el peor caso, exceptuando el obtenido con solo 10 caracteres o el 5% por ciento de solape para *reads* de longitud 50, el ensamble

obtenido tiene una calidad similar a LPS, aun así, aplicando PadeNA logramos reducir la cantidad de secuencias que darán como entrada a PadeNA, ayudándolo a realizar el ensamble.

5.3 Experimento 2 Ensamble del plásmido Aeromonas hydrophila plasmid pRA3

En este experimento usaremos a LPS con una secuencia del plásmido Aeromonas hydrophila plasmid pRA3, disponible en Accession NC_010919 (Kulinska, Czeredys, Hayes, & Jagura-Burdzy, 2008). El tamaño de esta secuencia es de 45,909 pares de bases. La secuenciación fue realizada sintéticamente *in silico* con cobertura de diez veces la molécula completa. A esta secuenciación, se realizó el proceso de ensamblado y se reportan los resultados utilizando las medidas de calidad expuestas previamente.

En la simulación de la secuenciación se crearon fragmentos de secuencias de longitud uniforme de 50 caracteres, considerando una sola dirección de izquierda a derecha (de 3' a 5'), libre de errores, y todos los caracteres de las secuencias del plásmido están contenidas en al menos un fragmento (cobertura total). El número de fragmentos obtenidos en la secuenciación son 18,580 *reads*.

Primeramente, realizaremos un ensamble usando PadeNA solamente. Los resultados obtenidos por PadeNA serán utilizados como criterio base de comparación para evaluar la calidad de la propuesta de la tesis que es inicialmente aplicar LPS previo a PadeNA. El parámetro fundamental de estos algoritmos es utilizar un valor de *k* para especificar el tamaño de los *k*-mers. En el próximo epígrafe explicaremos cómo buscamos un valor de *k* que brinde un buen ensamble contra el cual nos compararemos utilizando como criterio las medidas de evaluación explicadas previamente.

5.3.1 Encontrando el tamaño de los *k*-mers para PadeNA

En este epígrafe se buscará un valor de *k* para determinar cuál longitud de los *k*-mers arroja un buen resultado. Los resultados obtenidos por el valor de *k* seleccionado serán utilizados como base de la comparación del algoritmo propuesto de LPS combinado con PadeNA.

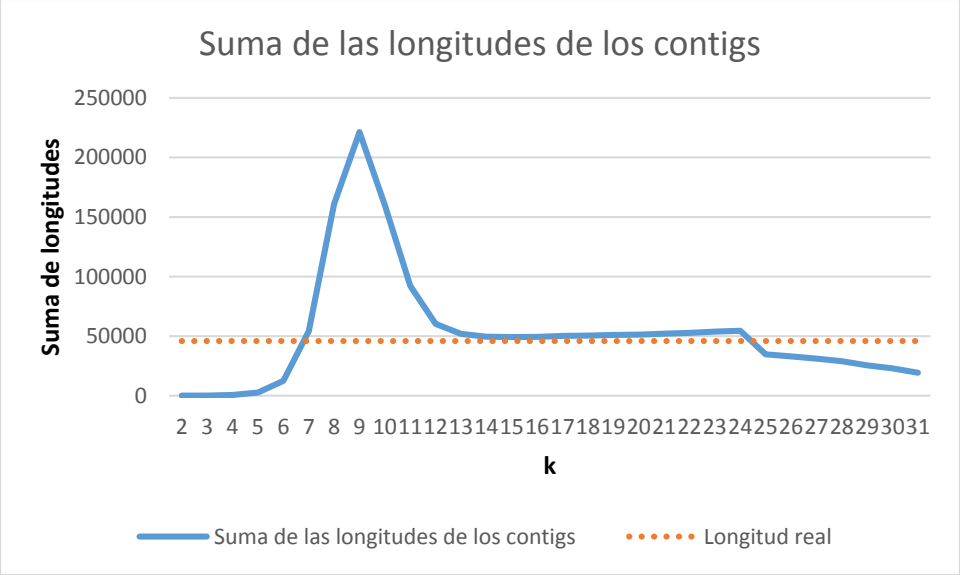

Fig. 42 Suma de las longitudes de los contigs con PadeNA

La Fig. 42 muestra en azul la suma total de los *contigs* puestos uno detrás del otro. El valor ideal a alcanzar es el que se aproxime al valor real de la longitud de la secuencia original mostrado en rojo el cual es 45,909. Los valores de *k* que dan un ensamble con una medida que se aproxima a este valor están en el rango de 13 a 24 siendo este el intervalo de valores de *k* que nos interesa.

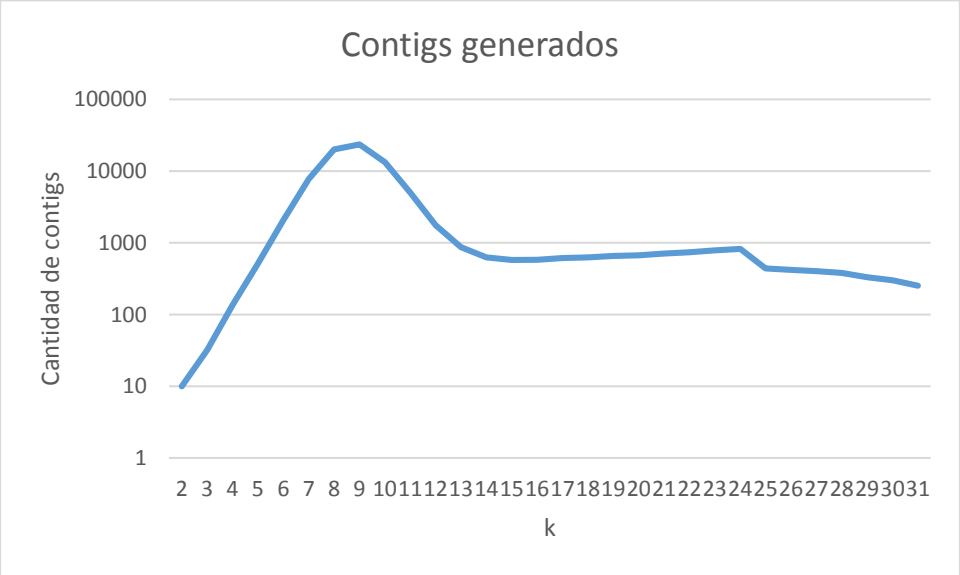

Fig. 43 Contigs generandos con PadeNA

La Fig. 43 contiene la cantidad de *contigs* generados con PadeNA utilizando los diferentes valores de *k*. La tabla muestra los valores en el eje de las abscisas utilizando una escala logarítmica. El resultado ideal es obtener un solo *contig* con una longitud igual a la secuencia original. Observando la Fig. 42 y la Fig. 43 vemos que para valores de *k* entre 2 y 5 se obtienen pocos fragmentos pero la suma de las longitudes está muy por debajo de la longitud de la secuencia original, por lo que estos valores de *k* tan pequeños son descartados. Para valores de *k* en el intervalo de interés comprendido entre 13 a 24 se observa que se obtiene una cantidad de *contigs* no tan elevada como los encontrados en el intervalo de *k* entre 6 y 12. Dado este comportamiento este intervalo sigue siendo candidato para obtener un valor de *k* con el que se obtenga una referencia para comparar con LPS combinado con PadeNA. Para valores de *k* de 25 en adelante, la cantidad de *contigs* disminuye pero podemos observar que la suma de las longitudes de los *contigs* disminuye por debajo de la longitud de la secuencia original siendo esto un síntoma que se están quedando regiones sin poder reconstruir de la secuencia original.

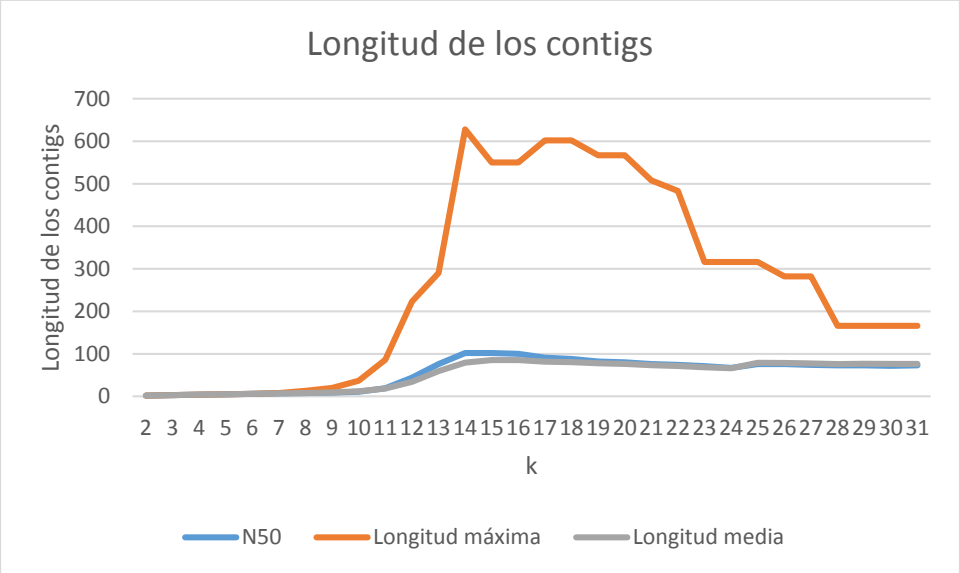

Fig. 44 Medidas de N50, Longitud del contig más largo y longitud media de los contigs con PadeNA

La Fig. 44 muestra las medidas usando las longitudes de los *contigs* N50, longitud del *contig* más largo y la longitud media de todos los *contigs*. Un ensamble que pueda ser interpretado por humanos necesita que los *contigs* encontrados tengan una longitud larga por lo que los valores de *k* serán los que muestren mayor valor de estas medidas. El intervalo que cumple

con esto está comprendido en el rango de valores de *k* entre 14 y 23, reduciendo un poco más el rango de valores de *k* candidatos para el ensamble contra el cual se realizará la comparación.

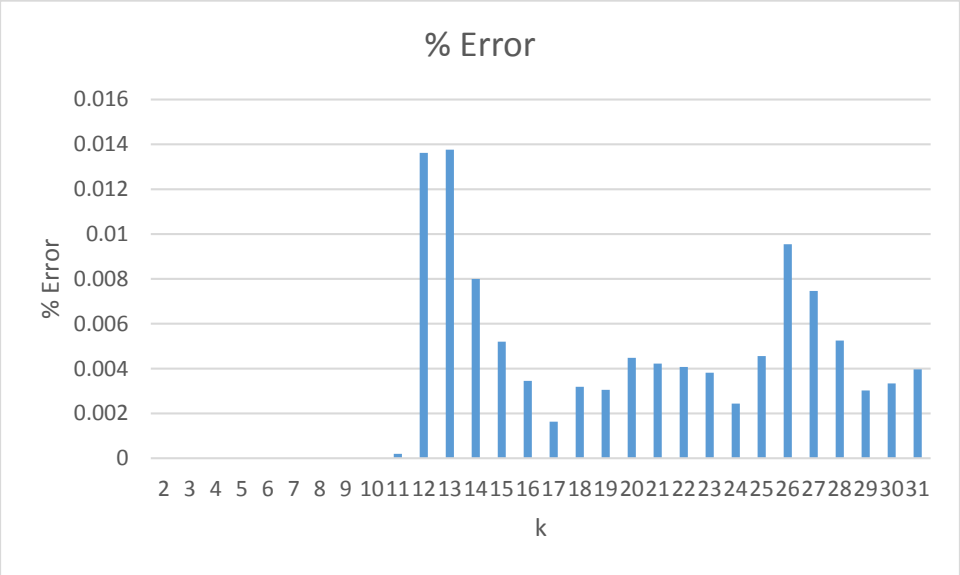

Fig. 45 Por ciento de error con PadeNA

La Fig. 45 muestra los errores en los ensambles. En el intervalo candidato de valores de *k* comprendido entre 14 y 23 vemos que para *k* = 15 donde se alcanza el menor error (0.001%).

Si tomamos como valor de *k* para el cual se obtiene que la suma de las longitudes de los *contigs* se acerca más a la longitud de la secuencia original, este valor se obtiene para *k*= 15 (Fig. 42). Además para este valor de *k* obtenemos la menor cantidad de *contigs* (Fig. 43) con un error bajo del 0.005% (Fig. 45). Este valor de *k* es el utilizado para obtener el ensamble de referencia (Tabla 7) y comparar la calidad de los ensambles obtenidos con LPS combinado con PadeNA.

Tabla 7 Evaluación de la calidad del ensamblado con PadeNA para k = 15

	Total de contigs	Contigs correctos	% error	Suma long de contigs	N50	Long máxima	Long media
PadeNA K = 15	577	574	0.005	49206	102	550	85.27

5.3.2 Ensamblado con LPS

El ensamble con LPS lo realizaremos similarmente a la forma en que hicimos el ensamble con LPS + PadeNA para el Bacillus cereus, donde probamos con los esquemas:

- *n* caracteres para LPS 1 - *n* caracteres para LPS 2: Para este experimento fijamos un valor de *n* para la primera corrida de LPS como para una segunda corrida. Los valores de *n* a probar son $n \in \{5,10,15,20,25,30,35,40,45\}$.
- *n* caracteres para LPS 1 – 20% para LPS 2: En este caso se usará una cantidad de *n* de caracteres para el solape en la primera corrida de LPS y luego se usará el 20% de la longitud del *read* más pequeño para la segunda corrida de LPS. El valor del 20% fue fijado debido a que con este parámetro de LPS se obtuvieron los ensambles con una buena calidad. La selección del valor del 20% se sustenta con los resultados mostrados en la próxima sección 5.3.4.

5.3.3 Ensamblado con *n* caracteres para LPS 1 - *n* caracteres para LPS 2

En este esquema se utilizará la misma cantidad de caracteres de solape para ambas corridas de LPS. Posteriormente, se aplicará PadeNA para mejorar la calidad del ensamble.

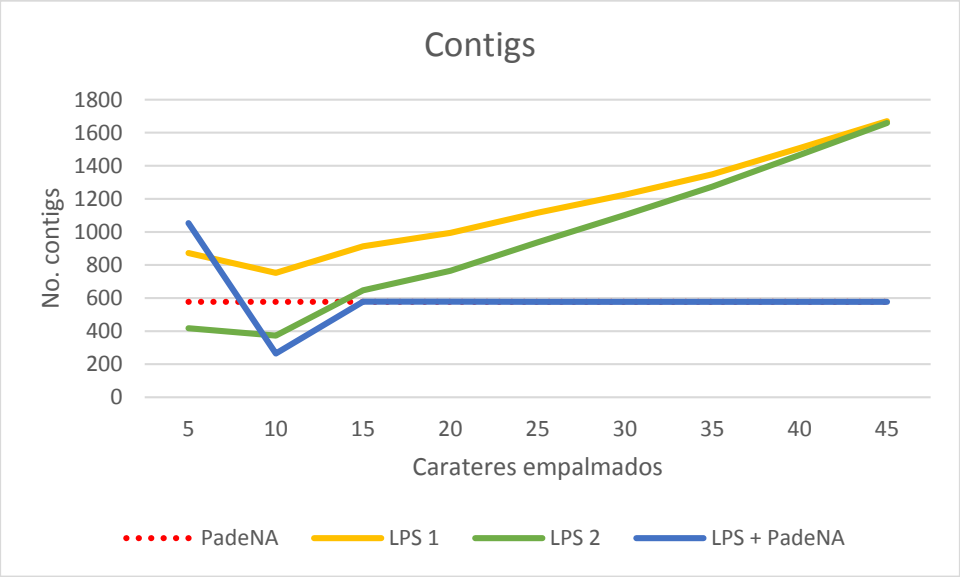

Fig. 46 Número de contigs

La Fig. 46 muestra la cantidad de *contigs* encontrados por ambas corridas de LPS y luego como mejora el resultado cuando usamos PadeNA en conjunto con LPS. Para n = 5 la cantidad de *contigs* obtenidos por LPS + PadeNA es superior que los obtenidos por PadeNA. La menor cantidad de *contigs* se alcanza para el valor de n = 10 en ambas corridas de LPS en conjunto con PadeNA siendo menor que PadeNA. Para PadeNA la cantidad de *contigs* encontrados son 577. Para la combinación de LPS + PadeNA este resultado mejoró reduciéndose en más de dos veces obteniéndose 265 *contigs*. Para valores de n igual o mayores de 15 la cantidad de *contigs* obtenidos mediante la combinación de LPS + PadeNA es similar a los obtenidos por PadeNA.

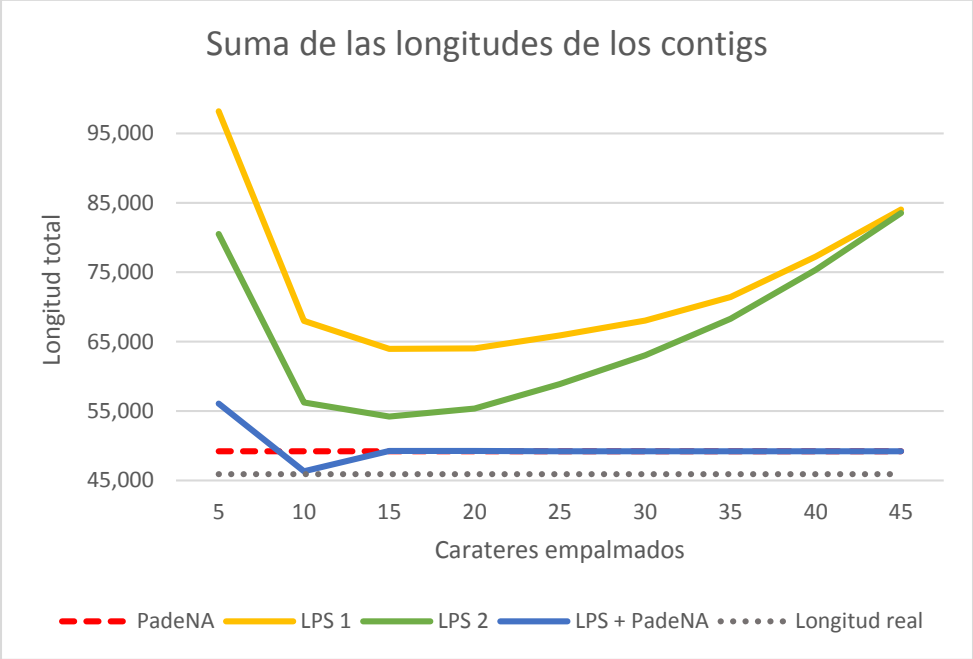

Fig. 47 Suma de las longitudes de los contigs

La Fig. 47 muestra la suma de las longitudes de los *contigs* para las dos corridas de LPS así como para LPS en conjunto con PadeNA. Para n = 5 la suma de los *contigs* obtenida mediante LPS + PadeNA es superior a la obtenida con PadeNA. Si observamos la Fig. 46 para este mismo valor de n, la cantidad de *contigs* resultantes de LPS + PadeNA es también mayor que los de PadeNA contribuyendo LPS + PadeNA más *contigs* a la suma de sus longitudes que PadeNA. El valor que se acerca más a la longitud del genoma real (45,909 pares de bases) se alcanza para n = 10 con LPS + PadeNA siendo 46,315 pares de bases, mejorando a la longitud

obtenida por PadeNA con 49,206 pares de bases. Para valores de *n* iguales o mayores de 15 la suma de las longitudes de los *contigs* resultantes de LPS + PadeNA son similares a las de PadeNA.

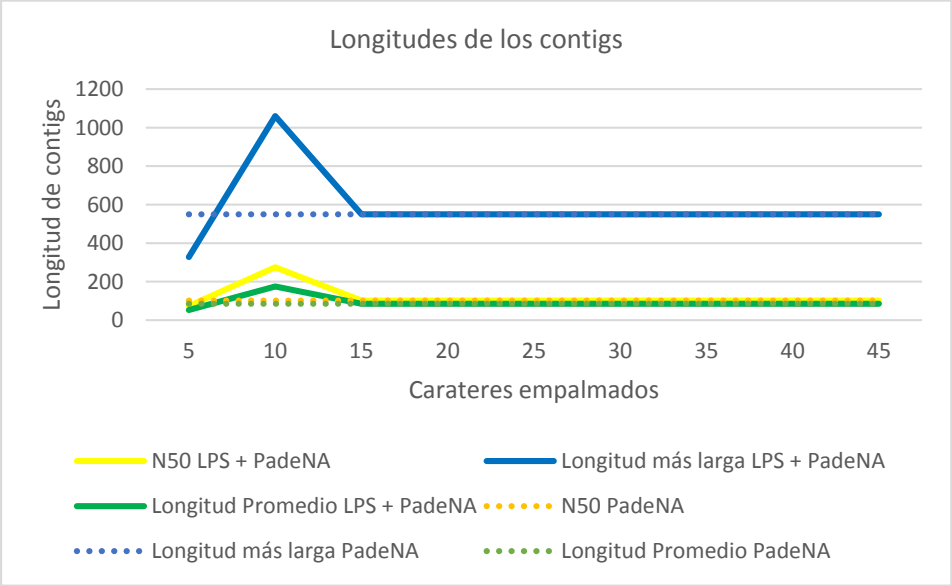

Fig. 48 Longitudes de los contigs

La Fig. 48 muestra mediante líneas discontinuas el valor de N50, promedio de las longitudes de los *contigs* y longitud del *contig* más largo obtenido por PadeNA. Las líneas continuas muestran estos resultados obtenidos para LPS + PadeNA. Podemos observar que para $n = 10$ se obtienen los *contigs* de longitudes más largos confirmando que para este valor de *n* la calidad del ensamble puede ser aceptada como buena.

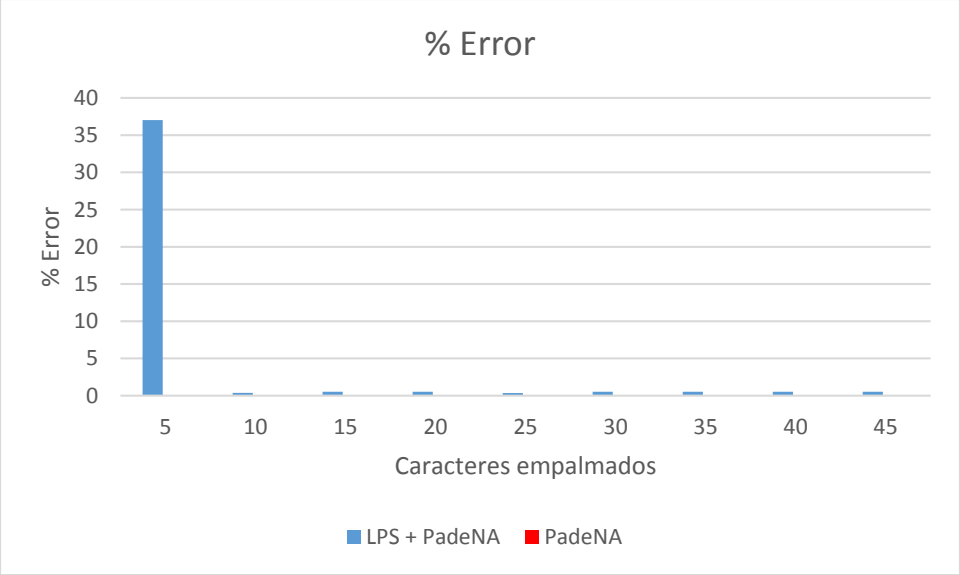

Fig. 49 Por ciento de error

La Fig. 49 muestra el por ciento de error de los *contigs* formados al usar un criterio de comparación de solape exacto. Consideramos que un *contig* es correcto si este es subcadena de la cadena formada por el genoma original. Para valores de $n = 5$, el por ciento de error cometido es grande. Esto se debe a que se exigen pocos caracteres para empalmarse, pudiendo crearse nuevos fragmentos erróneos producto de la mezcla de pares de fragmentos que se solapen. Para $n = 10$ se observa que el por ciento de error se reduce considerablemente, siendo similar al cometido por PadeNA, obteniéndose para el valor de $n = 10$ un ensamble con mejor calidad con la combinación de LPS + PadeNA que con PadeNA sólo.

5.3.4 Ensamblado con *n* caracteres para LPS 1 – 20% caracteres para LPS 2

En la sección 5.3.3 mostramos que el ensamble con mejor calidad obtenido fue para $n = 10$ caracteres en ambas corridas de LPS. Considerando que 10 caracteres es equivalente al 20% de la longitud de fragmentos de tamaño 50 caracteres, entonces, para este experimento tomamos como configuración de parámetros como *n* caracteres para la primera corrida de LPS y 20% de la longitud de los *contigs* para la segunda corrida de LPS. Finalmente, aplicaremos PadeNA para mejorar la calidad del ensamble. Dado que el valor de LPS para la segunda corrida está fijado a 20%, tomemos como convenio el valor de *n* de la primera corrida de LPS para denotar los diferentes ensambles obtenidos en la experimentación.

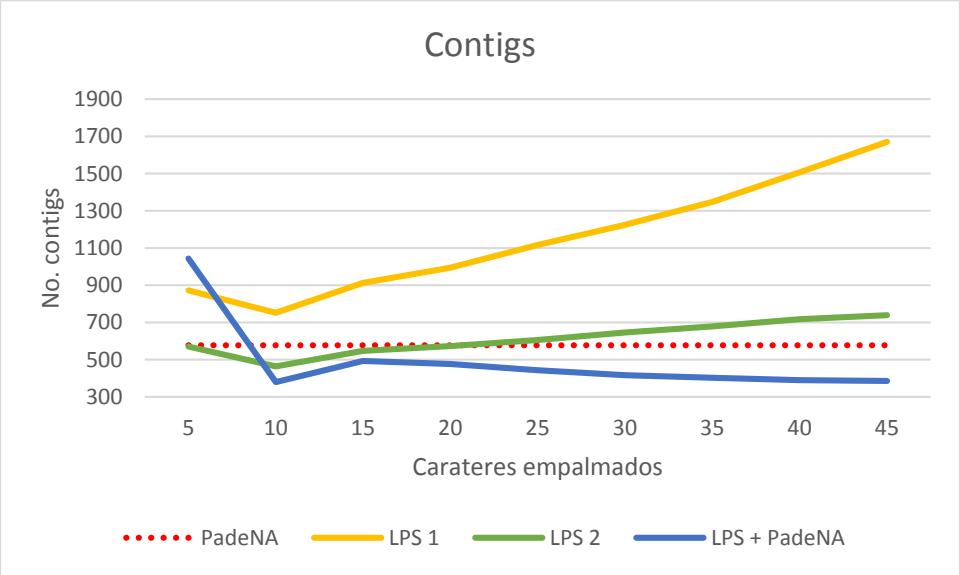

Fig. 50 Número de contigs

La Fig. 50 muestra la cantidad de *contigs* encontrados por ambas corridas de LPS y luego como mejora el resultado cuando usamos PadeNA en conjunto con LPS. Para n = 5 en la primera corrida, la cantidad de contigs obtenidos por LPS + PadeNA es superior que los obtenidos por PadeNA. La menor cantidad de se alcanza para el valor de n = 10 usando LPS en conjunto con PadeNA siendo menor que PadeNA. Para PadeNA la cantidad de *contigs* encontrados son 577. Para la combinación de LPS + PadeNA este resultado mejoró reduciéndose en dos veces obteniéndose 379 *contigs*. Para valores de n = 15, la cantidad de *contigs* encontrados por LPS + PadeNA aumenta ligeramente, pero sigue siendo inferior a PadeNA. Para valores de n mayores de 15 se observa que la cantidad de *contigs* vuelve a reducirse progresivamente decreciendo a medida que aumenta el valor de n acercándose a la cantidad de *contigs* obtenidos con n = 10. El mejoramiento de la calidad del ensamble a partir de n mayor o igual que 15 muestra que a pesar de que exigimos muchos caracteres para el empalme en la primera corrida de LPS, si decrementamos la cantidad de caracteres en la segunda corrida de LPS, la calidad del ensamble va mejorando progresivamente, ya que permite obtener secuencias con una longitud más larga que usando la misma cantidad de caracteres para ambas corridas de LPS.

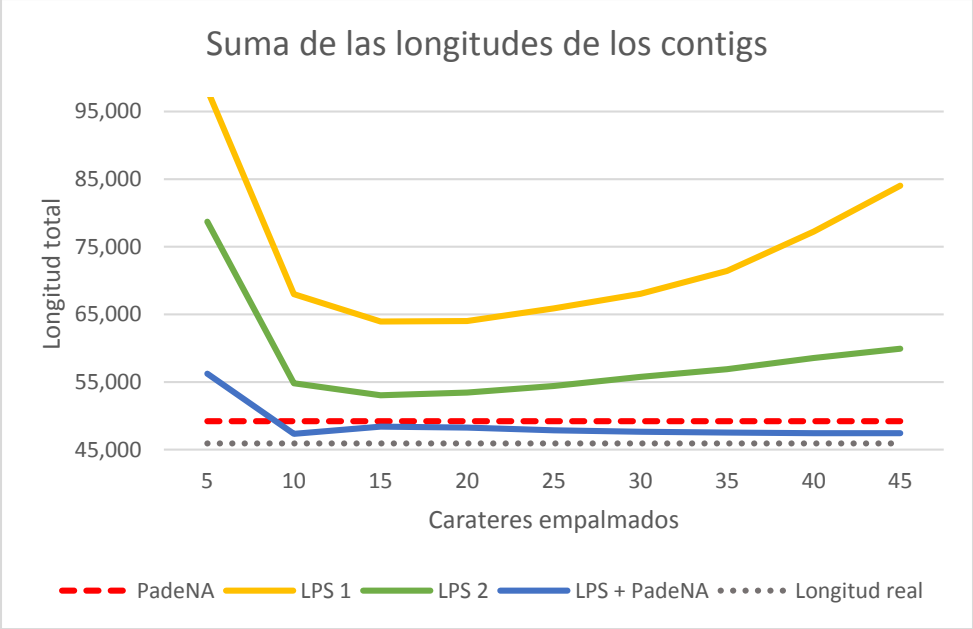

Fig. 51 Suma de las longitudes de los contigs

La Fig. 51 muestra la suma de las longitudes de los *contigs* para las dos corridas de LPS así como para LPS en conjunto con PadeNA. Para $n = 5$ la suma de los *contigs* obtenida mediante LPS + PadeNA es superior a la obtenida con PadeNA. Si observamos la Fig. 50 para este mismo valor de *n*, la cantidad de *contigs* resultantes de LPS + PadeNA es también mayor que los de PadeNA contribuyendo LPS + PadeNA más *contigs* a la suma de sus longitudes que PadeNA. El valor que se acerca más a la longitud del genoma real (45,909 pares de bases) se alcanza para $n = 10$ con LPS + PadeNA siendo 47,334 pares de bases, mejorando a la longitud obtenida por PadeNA con 49,206 pares de bases. Para valores de $n = 15$ la suma de las longitudes de los *contigs* resultantes aumenta, aunque se mantienen inferiores a PadeNA. A partir de $n = 20$ en adelante la suma de las longitudes de los *contigs* decrece progresivamente acercándose al valor obtenido para $n = 10$. Observando la Fig. 50 es esperado este comportamiento debido a que para $n = 10$ obtenemos la menor cantidad de *contigs*, para $n = 15$, la cantidad de *contigs* aumenta ligeramente y para valores de *n* superiores decrece a medida que se aumenta el valor de *n*.

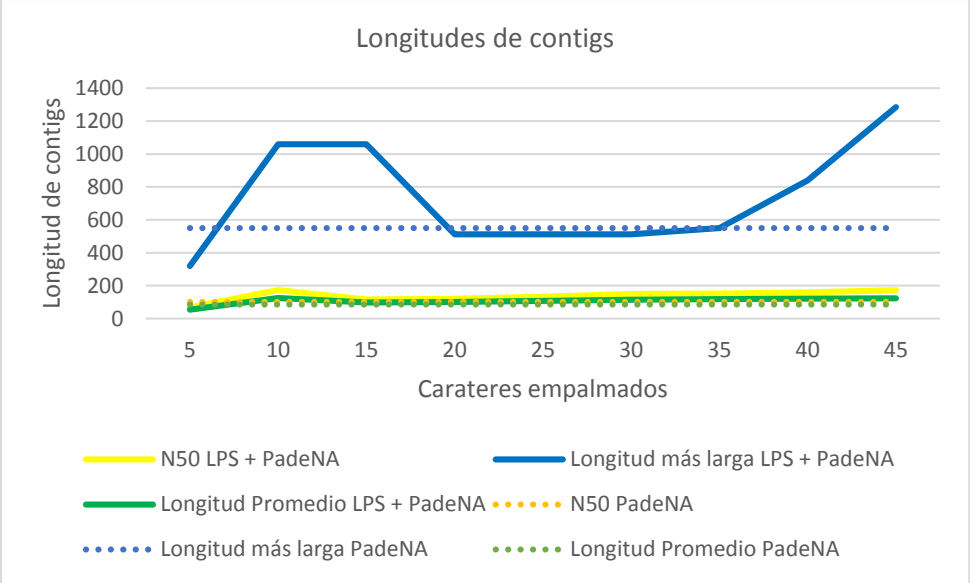

Fig. 52 Longitudes de los contigs

La Fig. 48 muestra mediante líneas discontinuas el valor de N50, promedio de las longitudes de los *contigs* y longitud del *contig* más largo obtenido por PadeNA. Las líneas continuas muestran estos resultados obtenidos para LPS + PadeNA. Podemos observar que para *n* mayor o igual a 10 se obtienen los *contigs* de longitudes más largos confirmando que para estos valores de *n* la calidad del ensamble puede ser aceptada como mejor que las obtenidas por PadeNA.

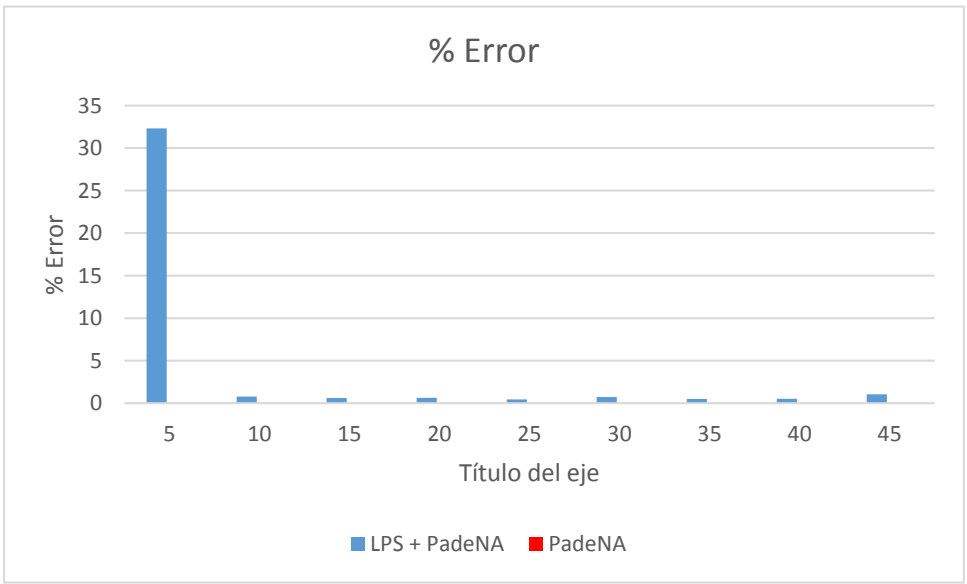

Fig. 53 Por ciento de error

La Fig. 53 muestra el por ciento de error de los *contigs* formados usando un criterio de comparación de solape exacto. Consideramos que un *contig* es correcto si este es subcadena de la cadena formada por el genoma original. Para valores de $n = 5$, el por ciento de error cometido es grande. Esto se debe a que se exigen pocos caracteres para empalmarse, pudiendo crearse nuevos fragmentos erróneos producto de la mezcla de pares de fragmentos que se solapen. Para valores de *n* mayores o iguales a 10 se observa que el por ciento de error se reduce considerablemente, siendo similar al cometido por PadeNA, aunque ligeramente superior a PadeNA, obteniéndose a pesar de ellos según los visto en las Fig. 50 - Fig. 52 ensambles con mejor calidad con la combinación de LPS + PadeNA que sólo con PadeNA.

5.3.5 Resumen de la calidad de los ensambles obtenidos del plásmido Aeromonas hydrophila plasmid pRA3

En este epígrafe mostraremos a modo resumen los resultados obtenidos para las diferentes configuraciones de los caracteres de solapes entre pares de secuencias en los ensambles de LPS + PadeNA mostrados anteriormente. Este epígrafe constituye un resumen de las medidas de por ciento de error, cantidad de *contigs*, suma de las longitudes de los *contigs*, N50, longitud del *contig* más largo, longitud media de los *contigs*, tomadas en los epígrafes anteriores para los ensambles de LPS + PadeNA.

Las líneas horizontal color rojo muestran el valor obtenido por PadeNA. Las líneas color azul oscuro muestran los resultados obtenidos por el ensamble usando *n* caracteres en la primera corrida de LPS y *n* caracteres en la segunda corrida de LPS. Las líneas color naranja muestran los resultados obtenidos por el ensamble usando *n* caracteres en la primera corrida de LPS y 20% caracteres en la segunda corrida de LPS.

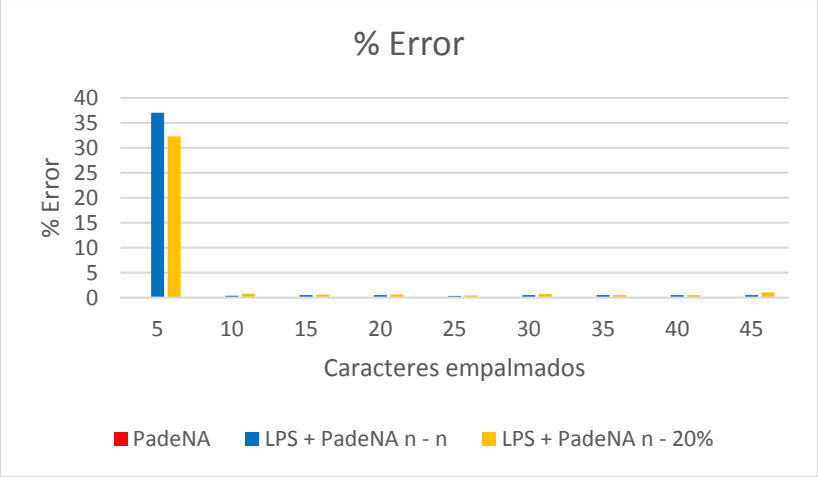

Fig. 54 Por ciento de error

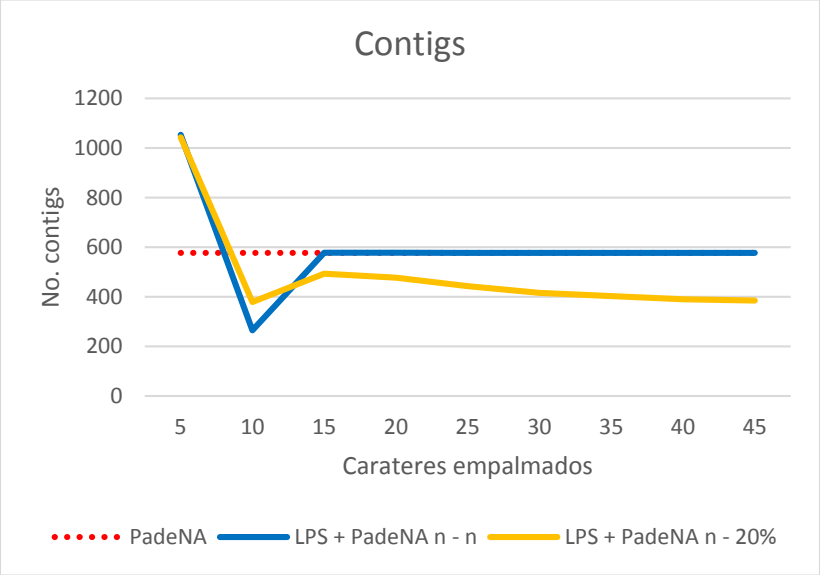

Fig. 55 Número de contigs

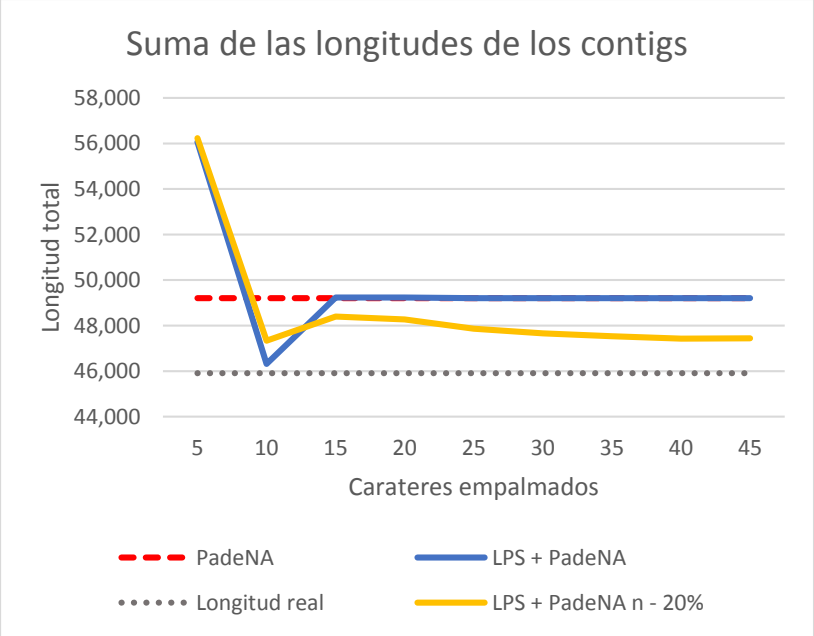

Fig. 56 Suma de las longitudes de los contigs

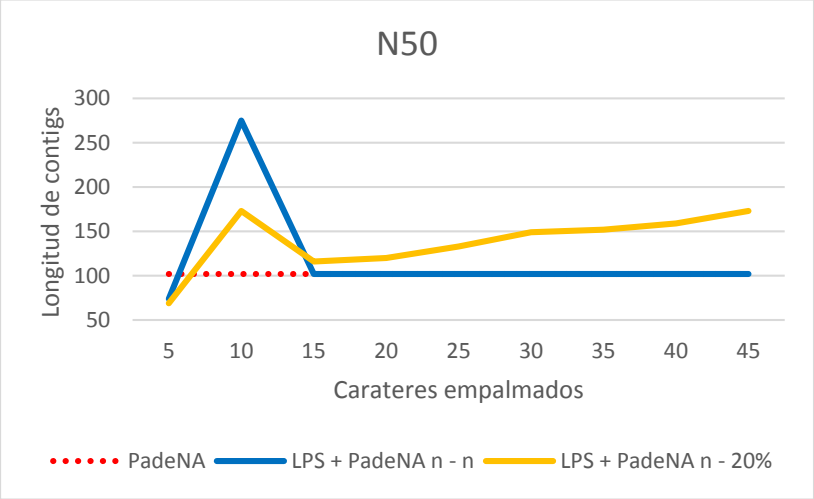

Fig. 57 N50

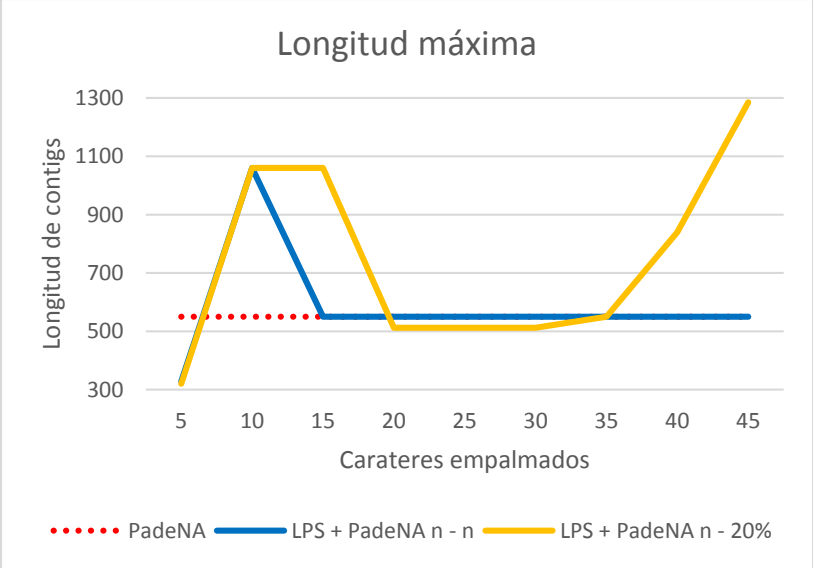

Fig. 58 Longitud del contig más largo

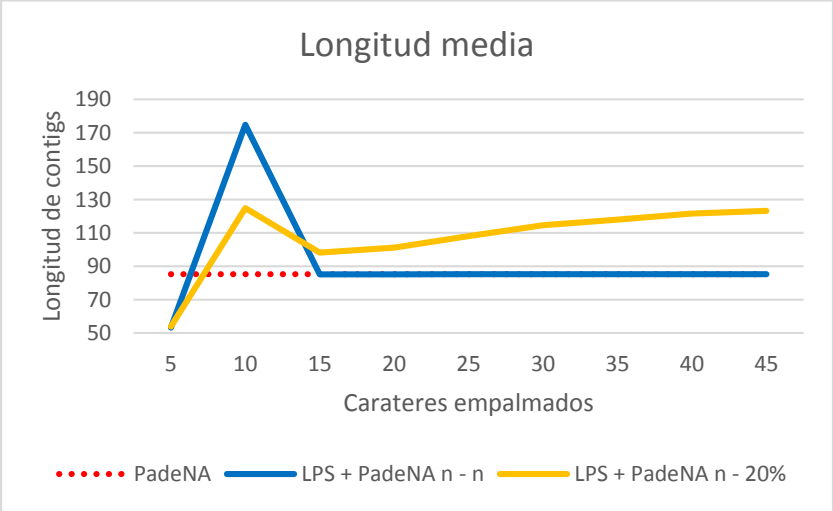

Fig. 59 Longitud media de los contigs

La Fig. 54 muestra los errores cometidos en los ensambles. Para valores de *n* bajos, de 5 caracteres, el error cometido en el ensamble es muy elevado por lo que descartamos esos ensambles.

Para valores de *n* = 10, en ambas combinaciones de ensamble, se obtienen los mejores valores de calidad. Esto se ve reflejado en las Fig. 54 - Fig. 59. La suma de las longitudes de los *contigs* se acerca a la longitud real de la secuencia original. La mejor combinación de valores

obtenida para este caso es 10 caracteres de solape para la primera corrida de LPS y 10 caracteres de solape para la segunda corrida de LPS. Estos resultados son alcanzados con un error menor de 1%. Para este caso se obtiene una buena mejoría debido a que la cantidad de caracteres necesarios para el solape entre secuencias son lo suficientes para crear nuevas secuencias a partir de las mezclas de este par empalmado tal que no sean erróneos y lo suficientemente para generar nuevas secuencias con longitudes más largas. Recordemos que el objetivo de aplicar LPS es generar secuencias de longitudes más largas para tratar los posibles *repeats*. A medida de que exijamos más caracteres para el solape, las nuevas secuencias creadas serán de longitud menor, pudiendo reducir en la calidad de ensamble. Esto puede ser visto para $n = 15$ en ambos esquemas de configuración de LPS. Para este valor la calidad del ensamble decae de manera tal que es similar al obtenido con PadeNA solamente (Fig. 55 -Fig. 59). A partir de este valor en adelante para los esquemas de la misma cantidad de caracteres del solape para las dos corridas de LPS, se observa que el ensamble tiene una calidad similar a PadeNA. Sin embargo, para el esquema de n caracteres en la primera corrida de LPS y 20% en la segunda corrida, a partir de $n = 20$ vemos que el ensamble mejora sucesivamente para los restantes valores de n (Fig. 55 -Fig. 59). Esto implica que a pesar de exigir muchos caracteres para el solape en la primera corrida, si relajamos esta restricción para la segunda etapa, podemos construir secuencias de longitudes más largas que hagan que mejore la calidad del ensamble. A medida que incrementemos el valor de n la diferencia entre éste y la cantidad de 20% aumenta, resultando en secuencias de longitudes más largas y por tanto un mejor ensamble superando a PadeNA. A pesar de no poder superar a la calidad de los ensambles obtenidos usando el esquema de $n = 10$, la opción de utilizar muchos caracteres en la primera corrida de LPS hace que se obtengan secuencias no tan extensas pero con pocos errores. Luego utilizando un valor bajo como el 20% para generar secuencias más largas con pocos errores, hace que este ensamble sea una solución que mejora el resultado obtenido por LPS.

5.4 Experimento 3 Ensamble del plásmido Bacillus anthracis str. 'Ames Ancestor' plasmid pXO2

En este experimento usaremos a LPS con una secuencia del plásmido Bacillus anthracis str. 'Ames Ancestor' plasmid pXO2, disponible en Accession NC_007323 (Ravel et al., 2009).

El tamaño de esta secuencia es de 94,830 pares de bases. La secuenciación fue realizada sintéticamente *in silico* con cobertura de diez veces la molécula completa. A esta secuenciación se realizó el proceso de ensamblado y se reportan los resultados utilizando las medidas de calidad expuestas previamente (epígrafe 5.1).

En la simulación de la secuenciación se crearon fragmentos de secuencias de longitud uniforme de 50 caracteres, considerando una sola dirección de izquierda a derecha (de 3' a 5'), libre de errores, y todos los caracteres de las secuencias del plásmido están contenidas en al menos un fragmento (cobertura total). El número de fragmentos obtenidos en la secuenciación son 39,204 *reads*.

Primeramente, realizaremos un ensamble usando PadeNA solamente. Los resultados obtenidos por PadeNA serán utilizados como criterio base de comparación para evaluar la calidad de la propuesta de la tesis que es inicialmente aplicar LPS previo a PadeNA. El parámetro fundamental de estos algoritmos es utilizar un valor de *k* para especificar el tamaño de los *k*-mers. En el próximo epígrafe explicaremos cómo buscamos un valor de *k* que brinde un buen ensamble contra el cual nos compararemos utilizando como criterio las medidas de evaluación explicadas previamente.

5.4.1 Encontrando el tamaño de los *k*-mers para PadeNA

En este epígrafe se buscará un valor de *k* para determinar cuál longitud de los *k*-mers arroja un buen resultado. Los resultados obtenidos por el valor de *k* seleccionado serán utilizados como base de la comparación del algoritmo propuesto de LPS combinado con PadeNA.

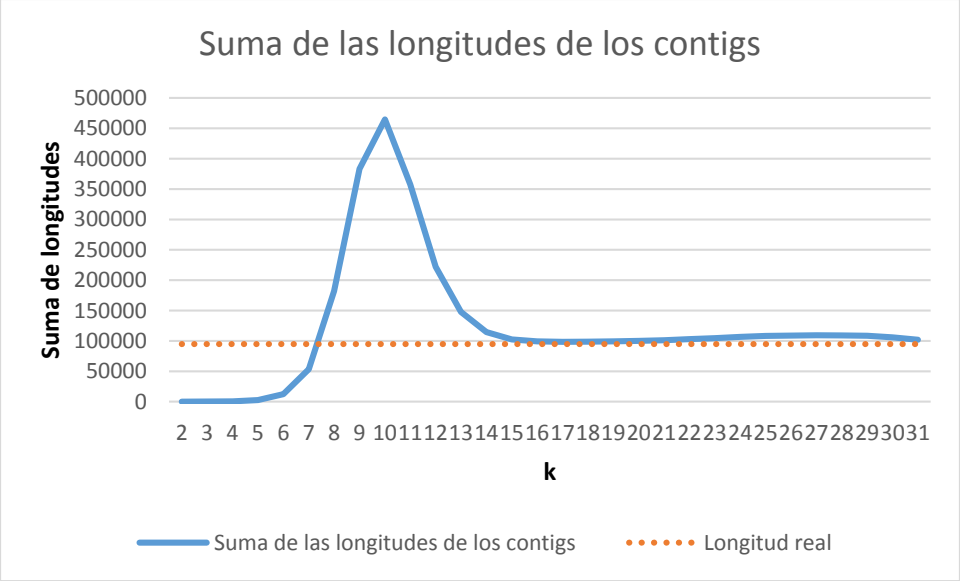

Fig. 60 Suma de las longitudes de los contigs con PadeNA

La Fig. 60 muestra en azul la suma total de los *contigs* puestos uno a continuación de otro. El valor ideal a alcanzar es el que se aproxime al valor real de la longitud de la secuencia original mostrado en rojo el cual es 94,830. Los valores de *k* que dan un ensamble con una medida que se aproxima a este valor están en el rango de 16 a 19 siendo este el intervalo de valores de *k* que nos interesa.

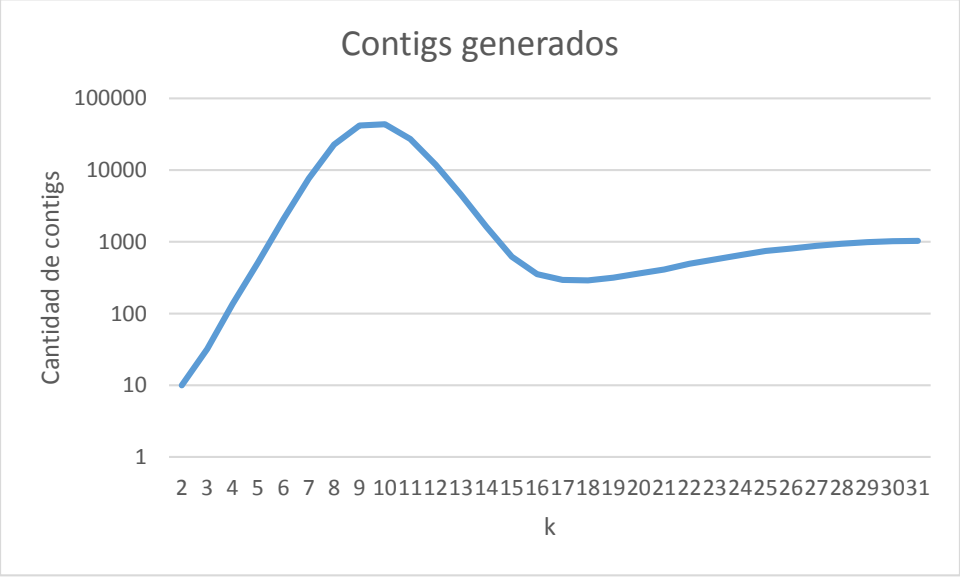

Fig. 61 Contigs generandos con PadeNA

La Fig. 61 contiene la cantidad de *contigs* generados con PadeNA utilizando los diferentes valores de *k*. La tabla muestra los valores en el eje de las abscisas utilizando una escala logarítmica. El resultado ideal es obtener un solo *contig* con una longitud igual a la secuencia original. Observando la Fig. 60 y la Fig. 61 vemos que para valores de *k* entre 2 y 4 se obtienen pocos fragmentos pero la suma de las longitudes está muy por debajo de la longitud de la secuencia original, por lo que estos valores de *k* tan pequeños son descartados. Para valores de *k* en el intervalo de interés comprendido entre 15 a 21 se observa que se obtiene una cantidad de *contigs* no tan elevada como los encontrados en el intervalo de *k* entre 6 y 12 pero inferiores para los restantes valores de *k*. Previamente, teníamos como candidato a tener un buen ensamble con valores de *k* comprendidos en el intervalo entre 16 y 19, el cual está incluido dentro del intervalo 15 a 21 recién deducido de la Fig. 61.

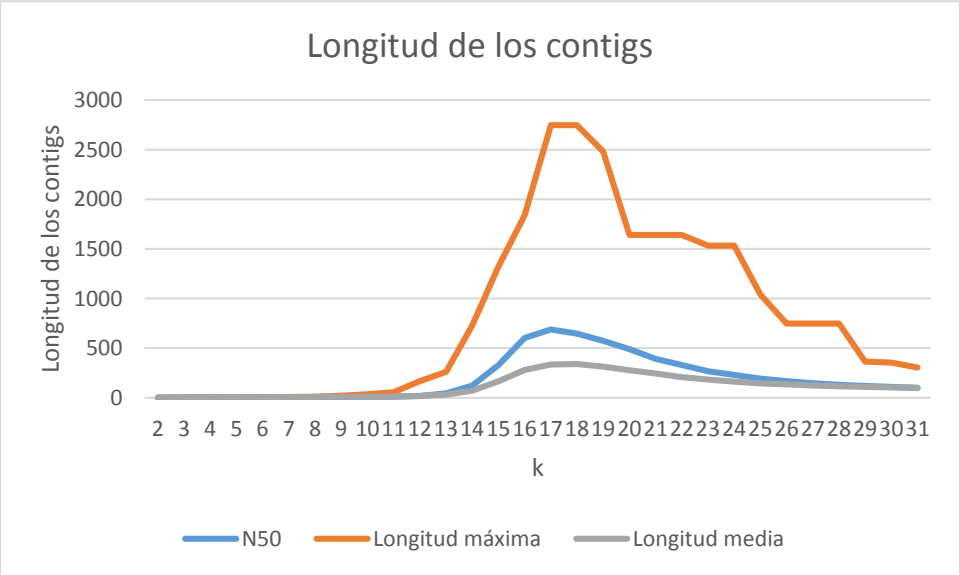

Fig. 62 Medidas de N50, Longitud del contig más largo y longitud media de los contigs con PadeNA

La Fig. 62 muestra las medidas usando las longitudes de los *contigs* N50, longitud del *contig* más largo y la longitud media de todos los *contigs*. Un ensamble que pueda ser interpretado por humanos necesita que los *contigs* encontrados tengan una longitud larga por lo que los valores de *k* serán los que muestren mayor valor de estas medidas. Si tomamos el intervalo de interés establecido previamente entre 16 y 19, podemos observar que las mayores longitudes están dentro de este intervalo destacándose que para *k* = 17 se alcanzan las

longitudes más grandes. Para terminar de asegurarnos que para este valor de *k* la calidad del ensamble es bueno, veremos que el por ciento de error cometido no sea elevado (Fig. 63).

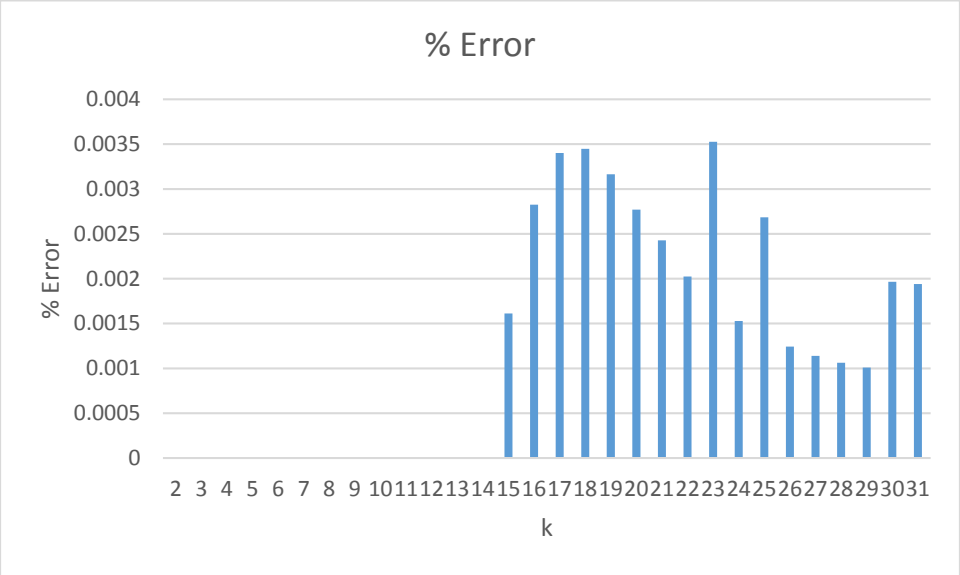

Fig. 63 Por ciento de error con PadeNA

La Fig. 63 muestra los errores en los ensambles. Si tomamos el por ciento de error para *k* = 17, vemos que el error cometido es de apenas 0.0034% siendo un valor bajo. Considerando que para el valor de *k* = 17 se obtuvieron *contigs* con una longitud grande (Fig. 98) y adicionalmente la resta de la suma de las longitudes de los *contigs* con la longitud de la secuencia original es la menor (Fig. 96), nos lleva a tomar las medidas de este ensamble como referencia (Tabla 8) para comparar con la calidad de los ensambles obtenidos usando LPS.

Tabla 8 Evaluación de la calidad del ensamblado con PadeNA para k = 17

	Total de contigs	Contigs correctos	% error	Suma long de contigs	N50	Long máxima	Long media
PadeNA K = 17	294	293	0.003	98589	689	2747	335.33

5.4.2 Ensamblado con LPS

El ensamble con LPS lo realizaremos similarmente a la forma en que hicimos el ensamble con LPS + PadeNA anteriores, donde probamos con los esquemas:

- *n* caracteres para LPS 1 - *n* caracteres para LPS 2: Para este experimento fijamos un valor de *n* para la primera corrida de LPS como para una segunda corrida. Los valores de *n* a probar son $n \in \{5,10,15,20,25,30,35,40,45\}$.
- *n* caracteres para LPS 1 – 20% para LPS 2: En este caso se usará una cantidad de *n* de caracteres para el solape en la primera corrida de LPS y luego se usará el 20% de la longitud del *read* más pequeño para la segunda corrida de LPS. El valor del 20% fue fijado debido a que con este parámetro de LPS se obtuvieron los ensambles con una buena calidad. La selección del valor del 20% se sustenta con los resultados mostrados en la próxima sección 5.4.4.

5.4.3 Ensamblado con *n* caracteres para LPS 1 - *n* caracteres para LPS 2

En este esquema se utilizará la misma cantidad de caracteres de solape para ambas corridas de LPS. Posteriormente, se aplicará PadeNA para mejorar la calidad del ensamble.

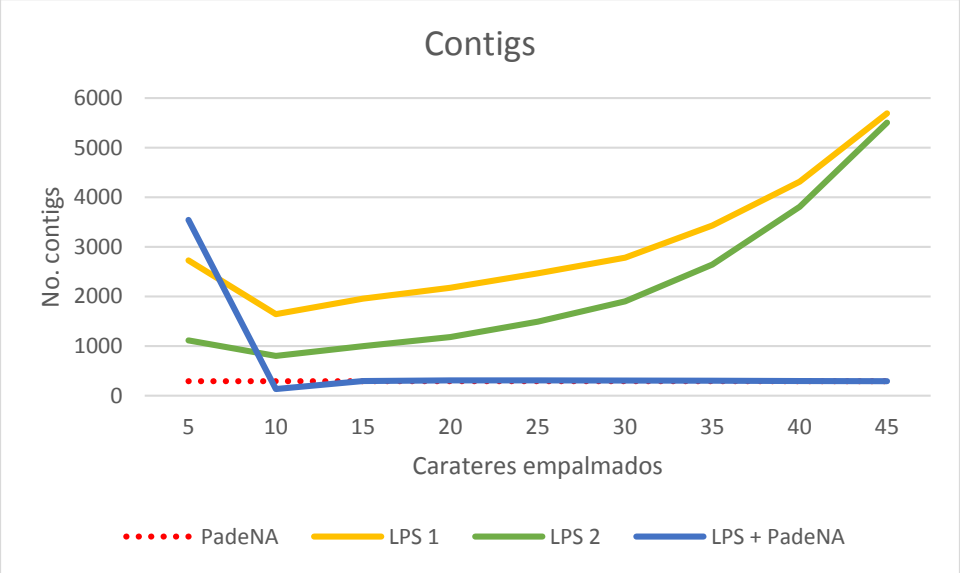

Fig. 64 Número de contigs

La Fig. 64 muestra la cantidad de *contigs* encontrados por ambas corridas de LPS y luego como mejora el resultado cuando usamos PadeNA en conjunto con LPS. Para *n* = 5 la cantidad de *contigs* obtenidos por LPS + PadeNA es superior que los obtenidos por PadeNA. La menor cantidad de se alcanza para el valor de *n* = 10 en ambas corridas de LPS en conjunto con PadeNA siendo menor que PadeNA. Para PadeNA la cantidad de *contigs*

encontrados es de 294. Para la combinación de LPS + PadeNA este resultado mejoró reduciéndose en más de dos veces obteniéndose 135 *contigs*. Para valores de *n* igual o mayores de 15 la cantidad de *contigs* obtenidos mediante la combinación de LPS + PadeNA es similar a los obtenidos por PadeNA.

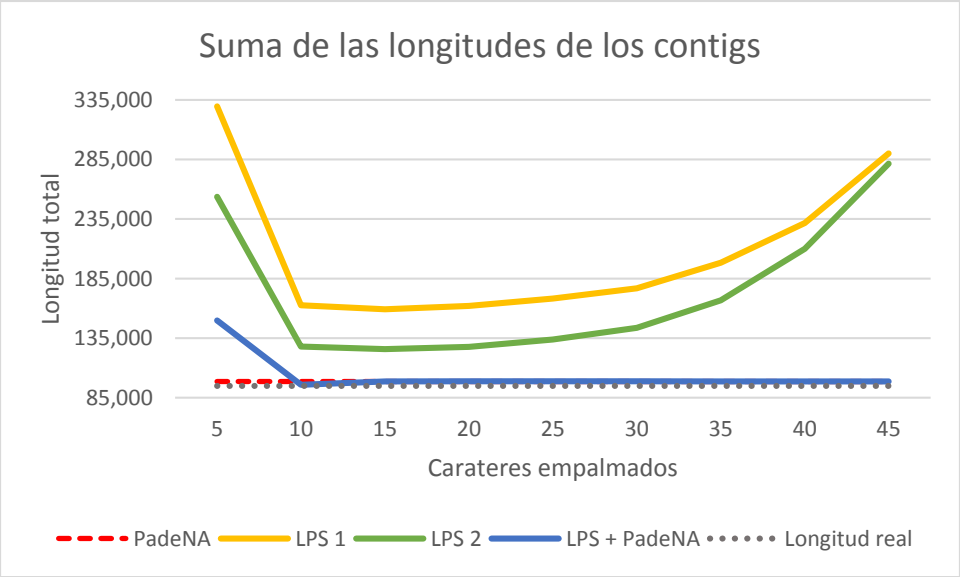

Fig. 65 Suma de las longitudes de los contigs

La Fig. 65 muestra la suma de las longitudes de los *contigs* para las dos corridas de LPS así como para LPS en conjunto con PadeNA. Para *n* = 5 la suma de los *contigs* obtenida mediante LPS + PadeNA es superior a la obtenida con PadeNA. Si observamos la Fig. 64 para este mismo valor de *n*, la cantidad de *contigs* resultantes de LPS + PadeNA es también mayor que los de PadeNA. De esta forma hace que LPS + PadeNA contribuya más *contigs* a la suma de sus longitudes que PadeNA. El valor que se acerca más a la longitud del genoma real (94,830 pares de bases) se alcanza para *n* = 10 con LPS + PadeNA siendo 95,792 pares de bases, mejorando a la longitud obtenida por PadeNA con 98,589 pares de bases. Para valores de *n* iguales o mayores de 15 la suma de las longitudes de los *contigs* resultantes de LPS + PadeNA son similares a las de PadeNA.

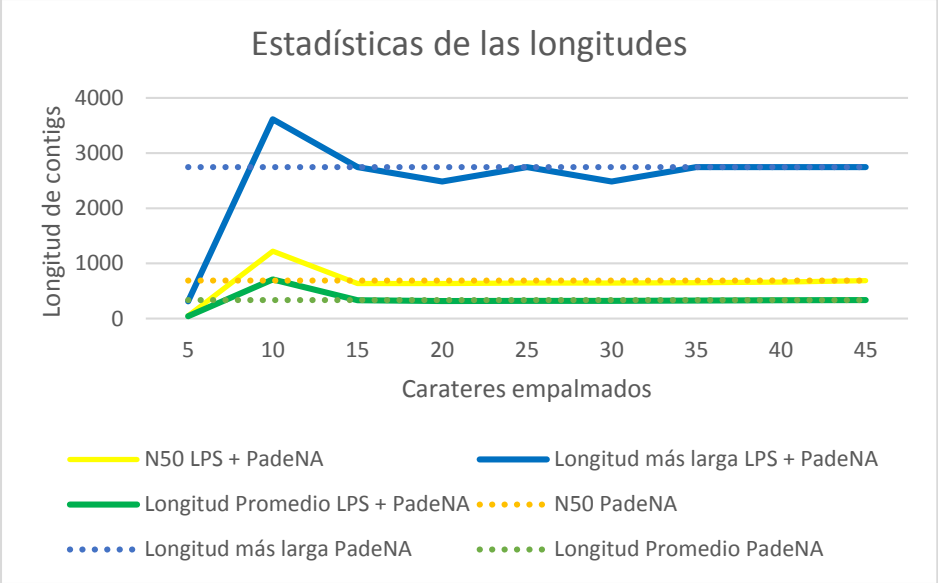

Fig. 66 Longitudes de los contigs

La Fig. 66 muestra mediante líneas discontinuas el valor de N50, promedio de las longitudes de los *contigs* y longitud del *contig* más largo obtenido por PadeNA. Las líneas continuas muestran estos resultados obtenidos para LPS + PadeNA. Podemos observar que para $n = 10$ se obtienen los *contigs* de longitudes más largos confirmando que para este valor de n la calidad del ensamble puede ser aceptada como buena.

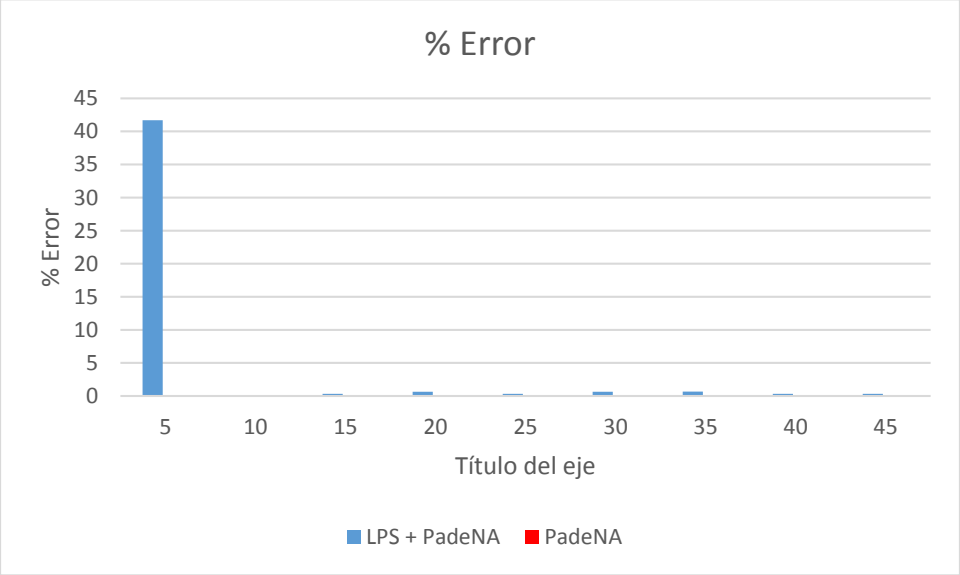

Fig. 67 Por ciento de error

La Fig. 67 muestra el por ciento de error de los *contigs* formados usando un criterio de comparación de solape exacto. Consideramos que un *contig* es correcto si este es subcadena de la cadena formada por el genoma original. Para valores de n = 5, el por ciento de error cometido es grande. Esto se debe a que se exigen pocos caracteres para empalmarse, pudiendo crearse nuevos fragmentos erróneos producto de la mezcla de pares de fragmentos que se solapen. Para n = 10 se observa que el por ciento de error se reduce considerablemente, siendo similar al cometido por PadeNA, obteniéndose para el valor de n = 10 un ensamble con mejor calidad con la combinación de LPS + PadeNA que con PadeNA sólo.

5.4.4 Ensamblado con *n* caracteres para LPS 1 – 20% caracteres para LPS 2

En la sección 5.4.3 mostramos que el ensamble con mejor calidad obtenido fue para n = 10 caracteres en ambas corridas de LPS. Considerando que 10 caracteres es equivalente al 20% de la longitud de fragmentos de tamaño 50 caracteres, entonces, para este experimento tomamos como configuración de parámetros como n caracteres para la primera corrida de LPS y 20% de la longitud de los *contigs* para la segunda corrida de LPS. Finalmente, aplicaremos PadeNA para mejorar la calidad del ensamble. Dado que el valor de LPS para la segunda corrida está fijado a 20%, tomemos como convenio el valor de n de la primera corrida de LPS para denotar los diferentes ensambles obtenidos en la experimentación.

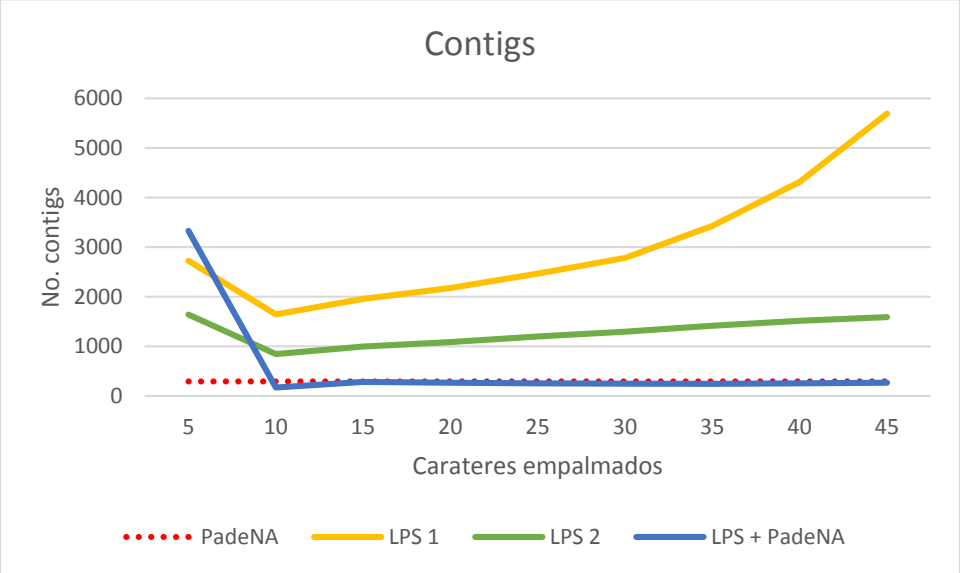

Fig. 68 Número de contigs

La Fig. 68 muestra la cantidad de *contigs* encontrados por ambas corridas de LPS y luego como mejora el resultado cuando usamos PadeNA en conjunto con LPS. Para $n = 5$ en la primera corrida, la cantidad de *contigs* obtenidos por LPS + PadeNA es superior que los obtenidos por PadeNA. La menor cantidad de *contigs* se alcanza para el valor de $n = 10$ usando LPS en conjunto con PadeNA siendo menor que PadeNA. Para PadeNA la cantidad de *contigs* encontradas es de 294. Para la combinación de LPS + PadeNA este resultado mejoró reduciéndose en dos veces obteniéndose 170 *contigs*. Para valores de $n = 15$, la cantidad de *contigs* encontrados por LPS + PadeNA aumenta ligeramente, pero sigue siendo inferior a PadeNA. Para valores de *n* mayores de 15 se observa que la cantidad de *contigs* vuelve a reducirse progresivamente decreciendo a medida que aumenta el valor de *n* acercándose a la cantidad de *contigs* obtenidos con $n = 10$. A diferencia de lo ocurrido en el ensamble para Bacillus cereus (epígrafe 5.2.5) y para Aeromonas hydrophila pRA3 (epígrafe 5.3.4) con *n* caracteres para LPS en la primera corrida y 20% de la longitud de los *contigs* para la segunda corrida de LPS, donde la calidad del ensamble comenzaba a mejorar a partir de un valor de *n* mayor a 15, en este ensamble la calidad mejora a partir de *n* mayor a 15; pero hasta $n = 35$. A partir de $n = 40$ en adelante, la calidad del ensamble comienza a ser parecido a PadeNA. Destaquemos que la longitud de Bacillus cereus es de 15,274 pares de bases y Aeromonas hydrophila pRA3 de 45,909 pares de bases, valores mucho menores a la longitud de Bacillus anthracis str. 'Ames Ancestor' plasmid pXO2 con 94,830 pares de bases. Al aplicar LPS, generamos nuevos fragmentos con una longitud mayor a los *reads* que son pasados como parámetro de entrada. Para un valor alto de *n*, los nuevos fragmentos tienen una longitud mucho menor que para valores de *n* de menor valor. Si comparamos la longitud de los fragmentos obtenidos usando un valor alto de *n* en relación al tamaño de la secuencia original y además comparamos la longitud de los *reads* iniciales con la longitud de la secuencia original, obtenemos que esta relación tiende a ser parecida obteniéndose con LPS un ensamble parecido al de PadeNA.

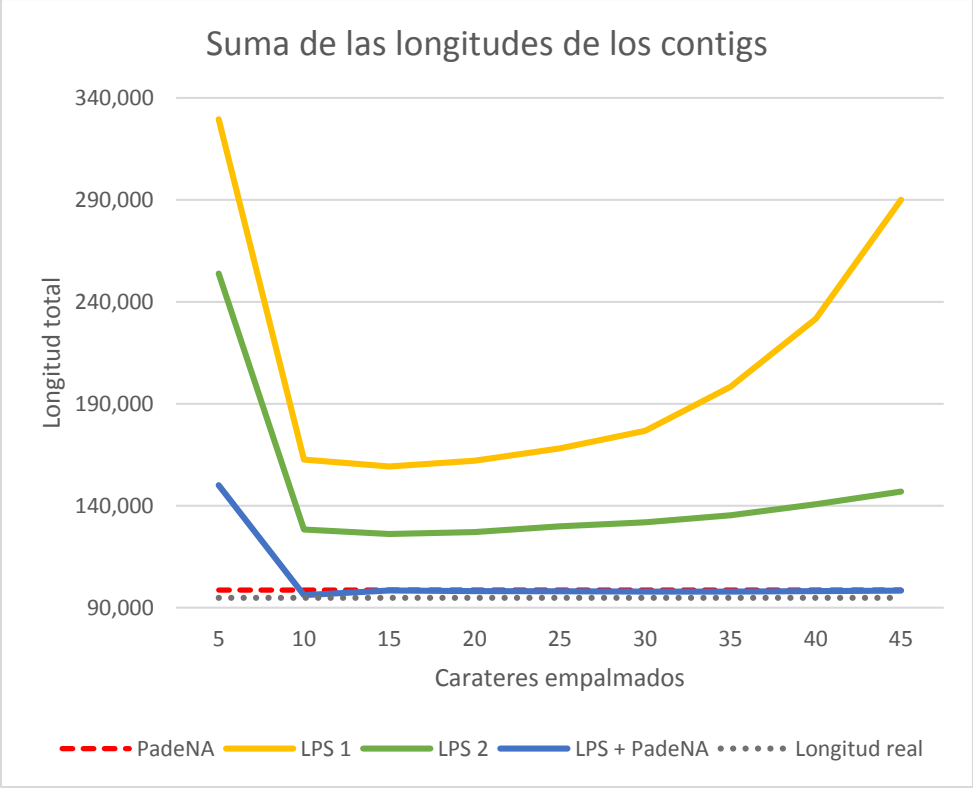

Fig. 69 Suma de las longitudes de los contigs

La Fig. 69 muestra la suma de las longitudes de los *contigs* para las dos corridas de LPS así como para LPS en conjunto con PadeNA. Para $n = 5$ la suma de los *contigs* obtenida mediante LPS + PadeNA es superior a la obtenida con PadeNA. Si observamos la Fig. 68 para este mismo valor de *n*, la cantidad de *contigs* resultantes de LPS + PadeNA es también mayor que los de PadeNA contribuyendo LPS + PadeNA más *contigs* a la suma de sus longitudes que PadeNA. El valor que se acerca más a la longitud del genoma real (94,830 pares de bases) se alcanza para $n = 10$ con LPS + PadeNA siendo 96,138 pares de bases, mejorando a la longitud obtenida por PadeNA con 98,589 pares de bases. Para valores de $n = 15$ la suma de las longitudes de los *contigs* resultantes aumenta, aunque se mantiene inferior a la de PadeNA. Similar al comportamiento mostrado por la cantidad de contigs en la Fig. 68, desde de $n = 20$ hasta $n = 35$ la suma de las longitudes de los *contigs* decrece progresivamente acercándose al valor obtenido para $n = 10$. A partir de $n = 40$ en adelante la suma de las longitudes de los *contigs* aumenta teniendo un valor parecido a la suma de las longitudes de los *contigs* obtenidos por PadeNA.

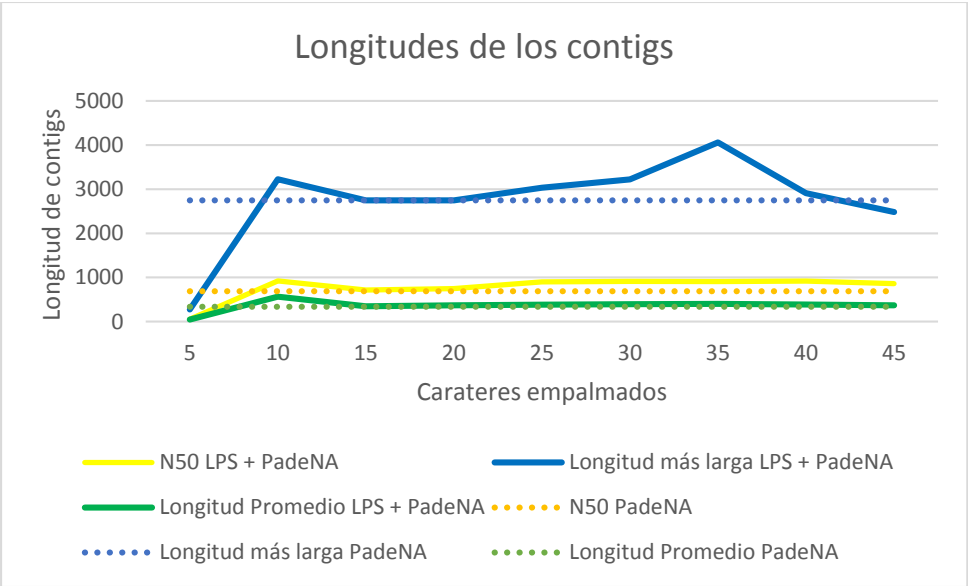

Fig. 70 Longitudes de los contigs

La Fig. 70 muestra mediante líneas discontinuas el valor de N50, promedio de las longitudes de los *contigs* y longitud del *contig* más largo obtenido por PadeNA. Las líneas continuas muestran estos resultados obtenidos para LPS + PadeNA. Observando las Fig. 68 y Fig. 69, junto a la Fig. 70, corroboramos que para $n = 10$ se obtiene un ensamble de mayor calidad con LPS + PadeNA comparado con PadeNA. Para $n = 15$, la calidad de este ensamble decae. A partir de $n = 20$ hasta $n = 35$, la calidad del ensamble aumenta progresivamente. Para valores de $n = 40$ en adelante, la calidad del ensamble con LPS + PadeNA vuelve a decaer acercándose a PadeNA.

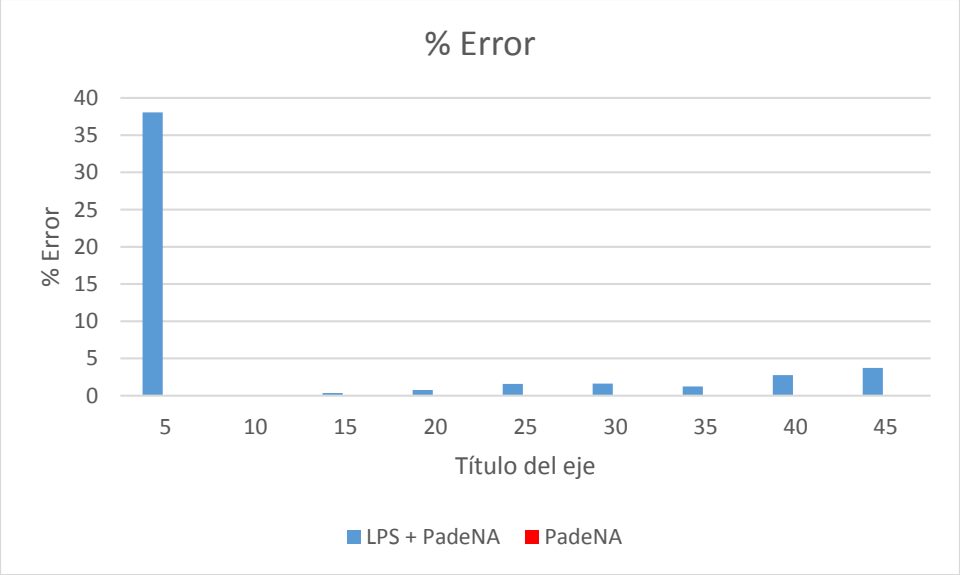

Fig. 71 Por ciento de error

La Fig. 71 muestra el por ciento de error de los *contigs* formados usando un criterio de comparación de solape exacto. Consideramos que un *contig* es correcto si este es subcadena de la cadena formada por el genoma original. Para valores de $n = 5$, el por ciento de error cometido es grande. Esto se debe a que se exigen pocos caracteres para empalmarse, pudiendo crearse nuevos fragmentos erróneos producto de la mezcla de pares de fragmentos que se solapen. Para valores de *n* mayores o iguales a 10 se observa que el por ciento de error se reduce considerablemente, siendo similar al cometido por PadeNA. Para valores de $n = 40$ en adelante, observamos que el error crece ligeramente respecto a los otros valores de *n*. Esto es de esperarse dado que para estos valores de *n* la longitud de los nuevos fragmentos son más pequeños en comparación con la longitud de la secuencia original, debido a que se exigen más elementos para el empalme entre pares de secuencias.

5.4.5 Resumen de la calidad de los ensambles obtenidos del plásmido Bacillus anthracis str. 'Ames Ancestor' plasmid pXO2

En este epígrafe solamente mostraremos a modo resumen los resultados obtenidos para las diferentes configuraciones de los caracteres de solapes entre pares de secuencias en los ensambles de LPS + PadeNA mostrados anteriormente. Este epígrafe solo constituye un resumen de las medidas de por ciento de error, cantidad de *contigs*, suma de las longitudes

de los *contigs*, N50, longitud del *contig* más largo, longitud media de los *contigs*, tomadas en los epígrafes anteriores para los ensambles de LPS + PadeNA.

Las líneas horizontal color rojo muestran el valor obtenido por PadeNA. Las líneas color azul oscuro muestran los resultados obtenidos por el ensamble usando *n* caracteres en la primera corrida de LPS y *n* caracteres en la segunda corrida de LPS. Las líneas color naranja muestran los resultados obtenidos por el ensamble usando *n* caracteres en la primera corrida de LPS y 20% de caracteres en la segunda corrida de LPS.

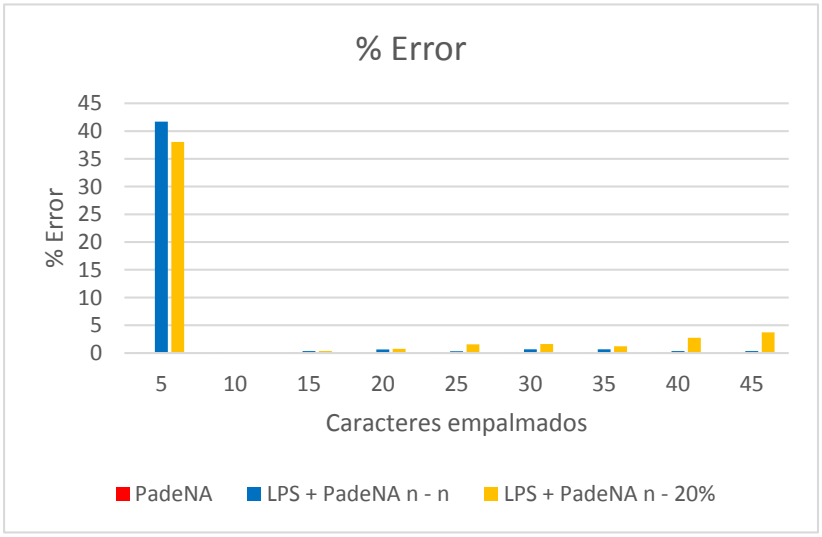

Fig. 72 Por ciento de error

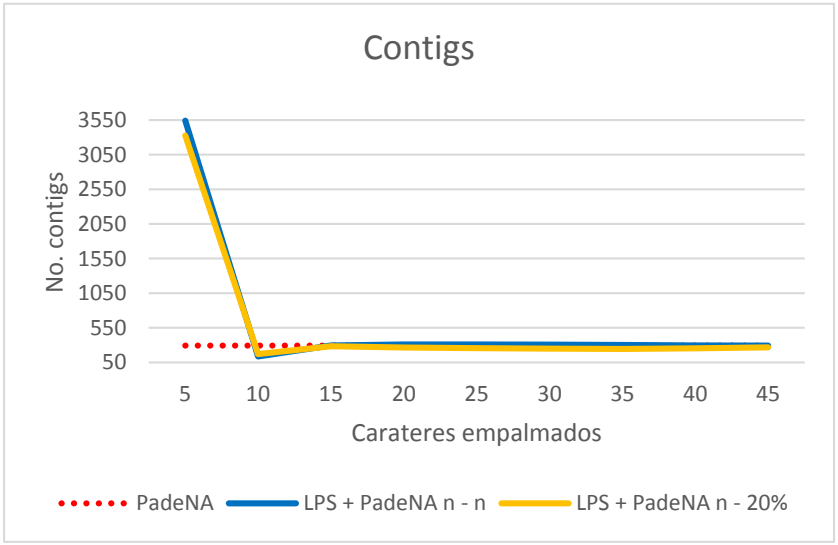

Fig. 73 Número de contigs

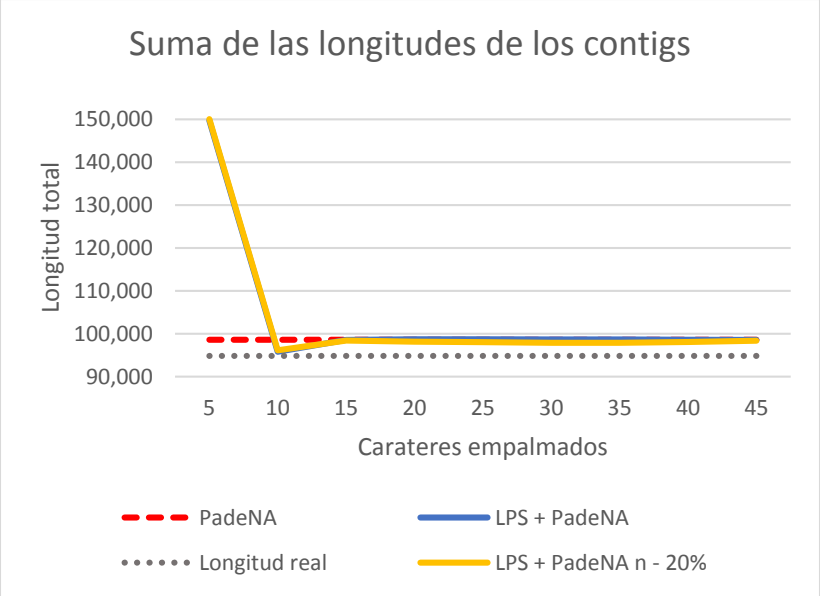

Fig. 74 Suma de las longitudes de los contigs

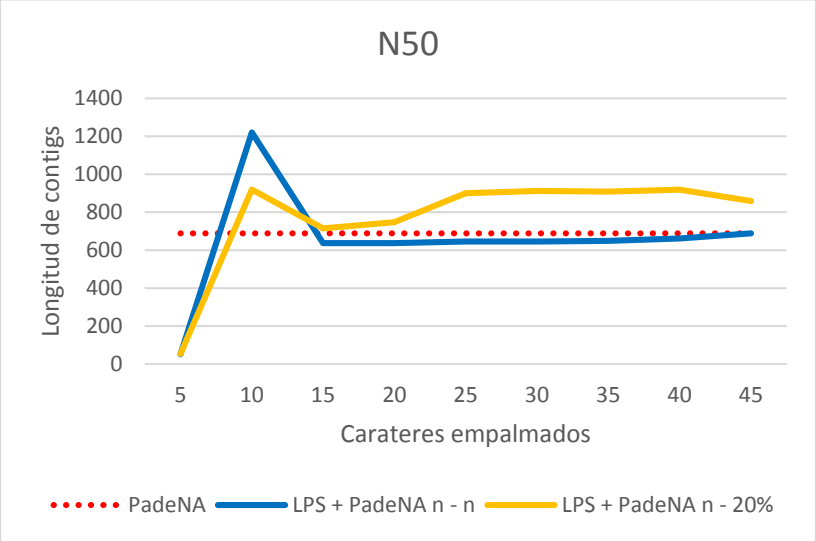

Fig. 75 N50

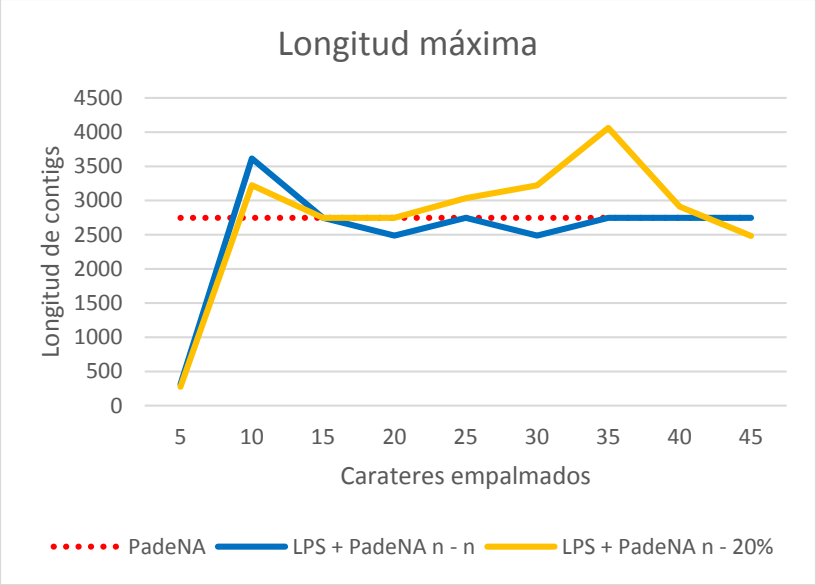

Fig. 76 Longitud del contig más largo

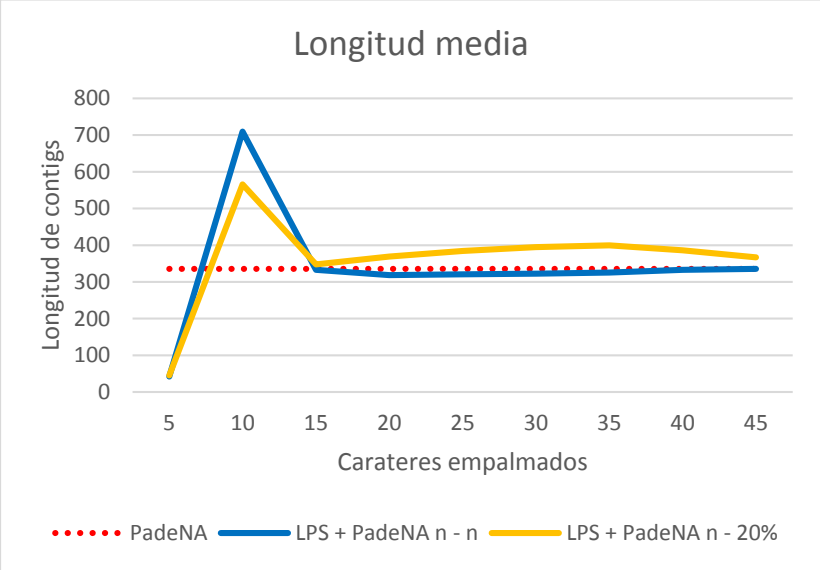

Fig. 77 Longitud media de los contigs

La Fig. 72 muestra los errores cometidos en los ensambles. Para valores de *n* bajos, de 5 caracteres, el error cometido en el ensamble es muy elevado por lo que descartamos esos ensambles. Esto se debe a que se exigen pocos caracteres para empalmarse, pudiendo crearse nuevos fragmentos erróneos producto de la mezcla de pares de fragmentos que se solapen.

Para valores de *n* = 10, en ambas combinaciones de ensamble, se obtienen los mejores valores de calidad. Esto se ve reflejado en las Fig. 71 - Fig. 77. La suma de las longitudes de los *contigs* se acerca a la longitud real de la secuencia original. La mejor combinación de valores obtenida para este caso es 10 caracteres de solape para la primera corrida de LPS y 10 caracteres de solape para la segunda corrida de LPS. Estos resultados son alcanzados con un error menor de 1%. Para este caso se obtiene una buena mejoría debido a que la cantidad de caracteres necesarios para el solape entre secuencias son lo suficientes para crear nuevas secuencias a partir de las mezclas de este par empalmado tal que no sean erróneos y lo suficiente para generar nuevas secuencias con longitudes más largas. Recordemos que el objetivo de aplicar LPS es generar secuencias de longitudes más largas para tratar los posibles *repeats*. A medida de que exijamos más caracteres para el solape, las nuevas secuencias creadas serán de longitud menor, pudiendo reducir en la calidad de ensamble. Esto puede ser visto para *n* = 15 en ambos esquema de configuración de LPS. Para este valor la calidad del ensamble decae de manera tal que es similar al obtenido con PadeNA solamente (Fig. 73 - Fig. 77). A partir de este valor en adelante para los esquemas de la misma cantidad de caracteres del solape para las dos corridas de LPS, se observa que el ensamble tiene una calidad similar a PadeNA. Sin embargo, para el esquema de *n* caracteres en la primera corrida de LPS y 20% en la segunda corrida, a partir de *n* = 20 vemos que el ensamble mejora sucesivamente para de *n* hasta *n* = 35 (Fig. 73 -Fig. 77). Esto implica que a pesar de exigir muchos caracteres para el solape en la primera corrida, si relajamos esta restricción para la segunda etapa, podemos construir secuencias de longitudes largas que hagan que mejore la calidad del ensamble. Sin embargo, para valores altos de *n* (a partir de *n* mayor o igual a 40), la calidad del ensamble comienza a decaer (Fig. 73 - Fig. 77). Una posible explicación de este comportamiento se debe a que los fragmentos formados tienen una longitud mucho menor para los fragmentos usando valores de *n* de menor valor. Si comparamos la longitud de los fragmentos obtenidos usando un valor alto de *n* y la longitud de los *reads* originales en relación al tamaño de la secuencia original, obtenemos que esta relación tiende a ser parecida, resultando en un ensamble similar al obtenido con PadeNA.

5.5 Experimento 4 Ensamble del plásmido Acaryochloris marina MBIC11017 plasmid pREB8

En este experimento usaremos a LPS con una secuencia del plásmido Acaryochloris marina MBIC11017 plasmid pREB8, disponible en Accession CP000845 (Swingley et al., 2008). El tamaño de esta secuencia es de 120,693 pares de bases. La secuenciación fue realizada sintéticamente *in silico* con cobertura de diez veces la molécula completa. A esta secuenciación se realizó el proceso de ensamblado y se reportan los resultados utilizando las medidas de calidad expuestas previamente (epígrafe 5.1).

En la simulación de la secuenciación se crearon fragmentos de secuencias de longitud uniforme de 50 caracteres, considerando una sola dirección de izquierda a derecha (de 3' a 5'), libre de errores, y todos los caracteres de las secuencias del plásmido están contenidas en al menos un fragmento (cobertura total). El número de fragmentos obtenidos en la secuenciación son 49,800 *reads*.

Primeramente, realizaremos un ensamble usando PadeNA solamente. Los resultados obtenidos por PadeNA serán utilizados como criterio base de comparación para evaluar la calidad de la propuesta de la tesis que es inicialmente aplicar LPS previo a PadeNA. El parámetro fundamental de estos algoritmos es utilizar un valor de *k* para especificar el tamaño de los *k*-mers. En el próximo epígrafe se expondrá cómo buscar un valor de *k* que brinde un buen ensamble contra el cual nos compararemos utilizando como criterio las medidas de evaluación explicadas previamente.

5.5.1 Encontrando el tamaño de los *k*-mers para PadeNA

En este epígrafe se buscará un valor de *k* para determinar cuál longitud de los *k*-mers arroja un buen resultado. Los resultados obtenidos por el valor de *k* seleccionado serán utilizados como base de la comparación del algoritmo propuesto de LPS combinado con PadeNA.

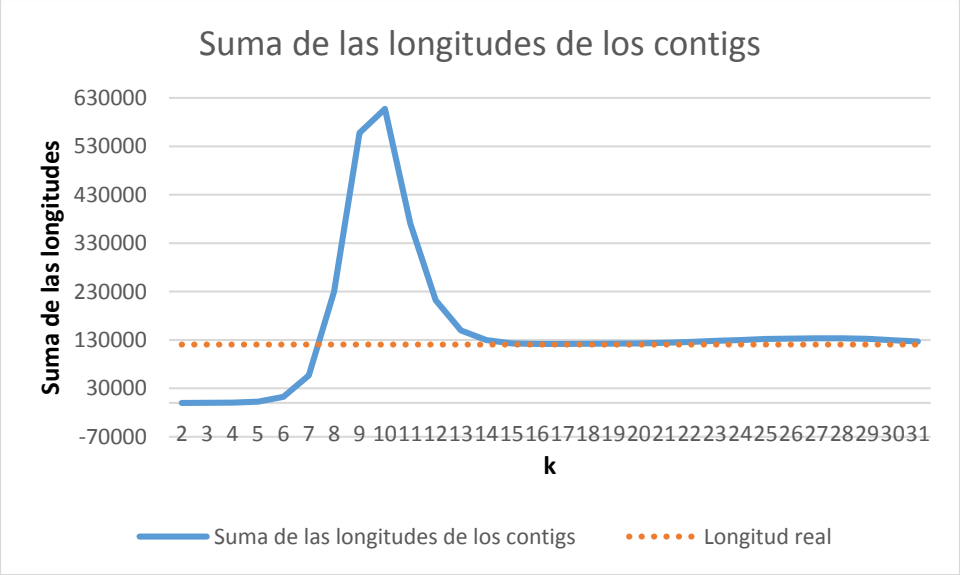

Fig. 78 Suma de las longitudes de los contigs con PadeNA

La Fig. 78 muestra en azul la suma total de los *contigs* puestos uno a continuación de otro. El valor ideal a alcanzar es el que se aproxime al valor real de la longitud de la secuencia original mostrado en rojo el cual es 120,693. Los valores de *k* que dan un ensamble con una medida que se aproxima a este valor están en el rango de 16 a 20 siendo este el intervalo de valores de *k* que nos interesa.

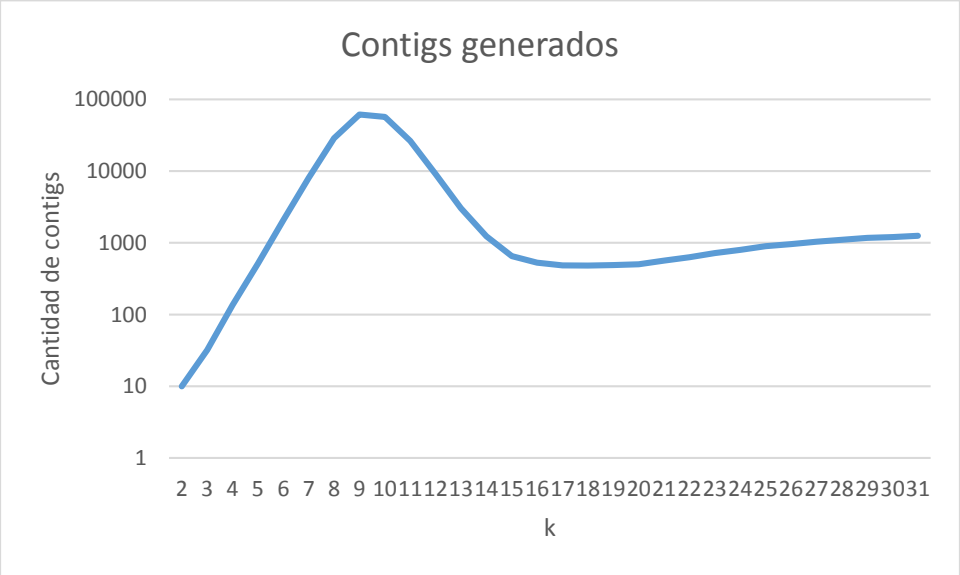

Fig. 79 Contigs generandos con PadeNA

La Fig. 79 contiene la cantidad de *contigs* generados con PadeNA utilizando los diferentes valores de *k*. La tabla muestra los valores en el eje de las abscisas utilizando una escala logarítmica. El resultado ideal es obtener un solo *contig* con una longitud igual a la secuencia original. Observando la Fig. 78 y la Fig. 79 vemos que para valores de *k* entre 2 y 7 se obtienen pocos fragmentos pero la suma de las longitudes está muy por debajo de la longitud de la secuencia original, por lo que estos valores de *k* tan pequeños son descartados. Para valores de *k* en el intervalo de interés comprendido entre 16 a 20 se observa que se obtiene una cantidad de *contigs* no tan elevada como los encontrados en el intervalo de *k* entre 8 y 15 pero inferiores para los restantes valores de *k*.

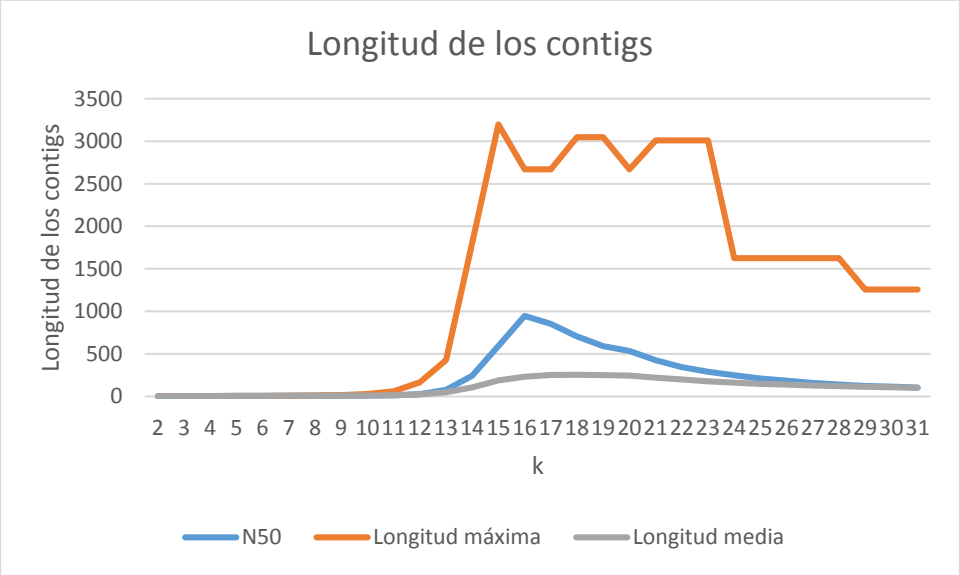

Fig. 80 Medidas de N50, Longitud del contig más largo y longitud media de los contigs con PadeNA

La Fig. 80 muestra las medidas usando las longitudes de los *contigs* N50, longitud del *contig* más largo y la longitud media de todos los *contigs*. Un ensamble que pueda ser interpretado por humanos necesita que los *contigs* encontrados tengan una longitud larga por lo que los valores de *k* serán los que muestren mayor valor de estas medidas. Si tomamos el intervalo de interés establecido previamente entre 16 y 20, podemos observar que dentro de este intervalo se obtienen altos valores para estas medidas.

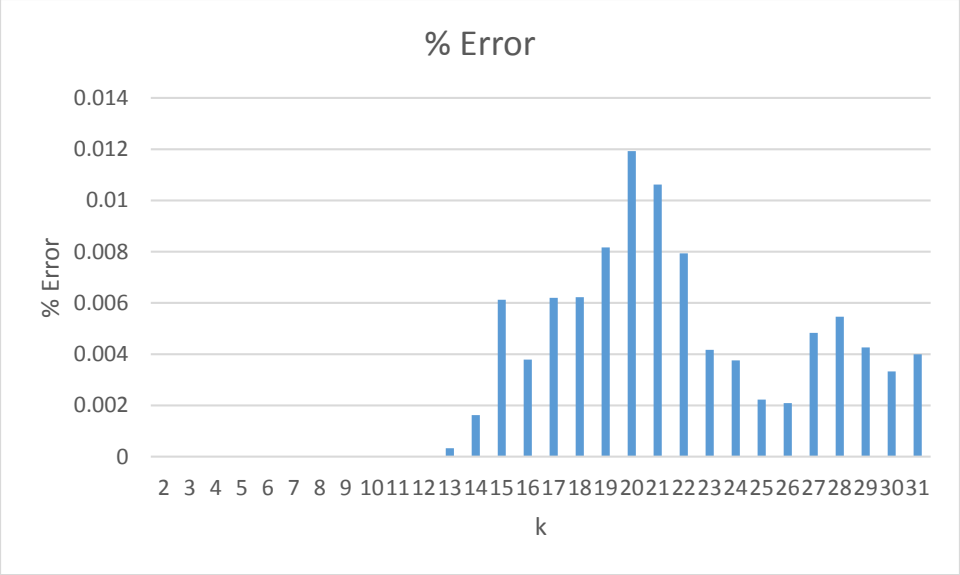

Fig. 81 Por ciento de error con PadeNA

La Fig. 81 muestra los errores en los ensambles. Vemos que los errores cometidos por PadeNA son bajos siendo inferiores a 0.012% pudiendo escogerse cualquier valor de *k* para el ensamble, en particular podemos tomar el valor dentro del intervalo de interés entre 16 y 20.

Dentro del intervalo de interés, escogemos el valor de *k* = 17, con el cual se obtiene un ensamble cuya suma de las longitudes es el que más se acerca a la longitud del genoma real (Fig. 78) siendo de 121,730 pares de bases para la longitud de PadeNA y 120,693 pares de bases para la longitud real. Usaremos la calidad del ensamble obtenido con este valor de *k* (Tabla 9) para comparar con la calidad de los ensambles obtenidos usando LPS.

Tabla 9 Evaluación de la calidad del ensamblado con PadeNA para k = 17

	Total de contigs	Contigs correctos	% error	Suma long de contigs	N50	Long máxima	Long media
PadeNA K = 17	484	481	0.006	121730	852	2669	251.5

5.5.2 Ensamblado con LPS

El ensamble con LPS lo realizaremos similarmente a la forma en que hicimos el ensamble con LPS + PadeNA anteriores, donde probamos con los esquemas:

- *n* caracteres para LPS 1 - *n* caracteres para LPS 2: Para este experimento fijamos un valor de *n* para la primera corrida de LPS como para una segunda corrida. Los valores de *n* a probar son $n \in \{5,10,15,20,25,30,35,40,45\}$.
- *n* caracteres para LPS 1 – 20% para LPS 2: En este caso se usará una cantidad de *n* de caracteres para el solape en la primera corrida de LPS y luego se usará el 20% de la longitud del *read* más pequeño para la segunda corrida de LPS. El valor del 20% fue fijado debido a que con este parámetro de LPS se obtuvieron los ensambles con una buena calidad. La selección del valor del 20% se sustenta con los resultados mostrados en la próxima sección 5.5.3.

5.5.3 Ensamblado con *n* caracteres para LPS 1 - *n* caracteres para LPS 2

En este esquema se utilizará la misma cantidad de caracteres de solape para ambas corridas de LPS. Posteriormente, se aplicará PadeNA para mejorar la calidad del ensamble.

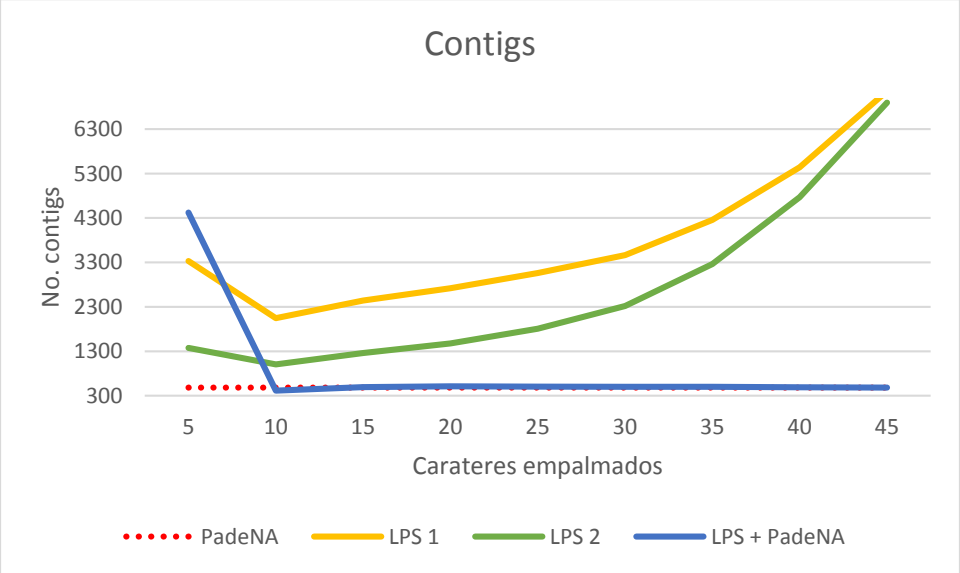

Fig. 82 Número de contigs

La Fig. 82 muestra la cantidad de *contigs* encontrados por ambas corridas de LPS y luego como mejora el resultado cuando usamos PadeNA en conjunto con LPS. Para *n* = 5 la cantidad de *contigs* obtenidos por LPS + PadeNA es superior que los obtenidos por PadeNA. La menor cantidad de *contigs* se alcanza para el valor de *n* = 10 en ambas corridas de LPS en conjunto con PadeNA siendo menor que PadeNA. Para PadeNA la cantidad de *contigs*

encontrados es de 484. Para la combinación de LPS + PadeNA este resultado mejoró reduciéndose en más de dos veces obteniéndose 413 *contigs*. Para valores de *n* igual o mayores de 15 la cantidad de *contigs* obtenidos mediante la combinación de LPS + PadeNA es similar a los obtenidos por PadeNA.

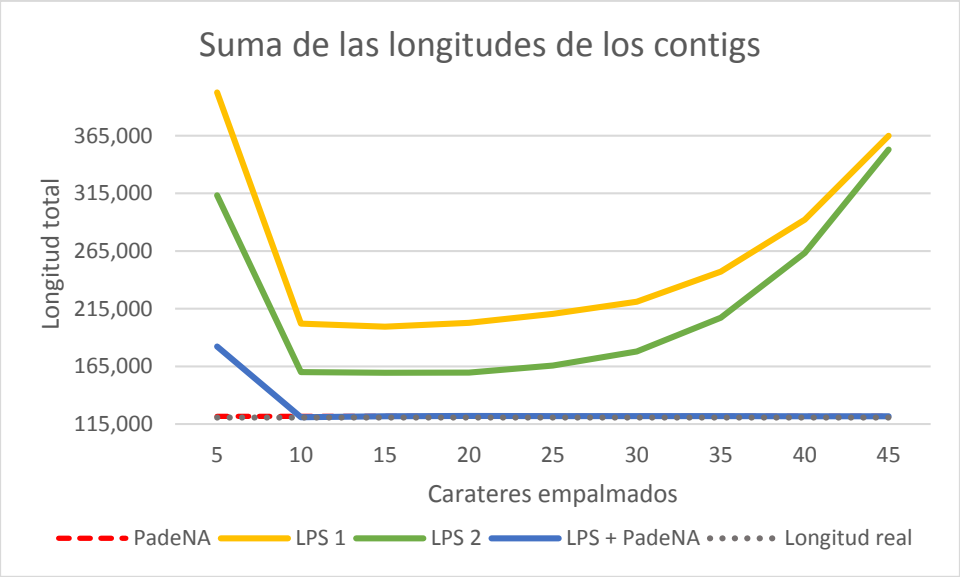

Fig. 83 Suma de las longitudes de los contigs

La Fig. 83 muestra la suma de las longitudes de los *contigs* para las dos corridas de LPS así como para LPS en conjunto con PadeNA. Para *n* = 5 la suma de los *contigs* obtenida mediante LPS + PadeNA es superior a la obtenida con PadeNA. Si observamos la Fig. 82 para este mismo valor de *n*, la cantidad de *contigs* resultantes de LPS + PadeNA es también mayor que los de PadeNA. De esta forma hace que LPS + PadeNA contribuya más *contigs* a la suma de sus longitudes que PadeNA. El valor que se acerca más a la longitud del genoma real (120,693 pares de bases) se alcanza para *n* = 10 con LPS + PadeNA siendo 121,730 pares de bases, mejorando a la longitud obtenida por PadeNA con 120,903 pares de bases. Para valores de *n* iguales o mayores de 15 la suma de las longitudes de los *contigs* resultantes de LPS + PadeNA son similares a las de PadeNA.

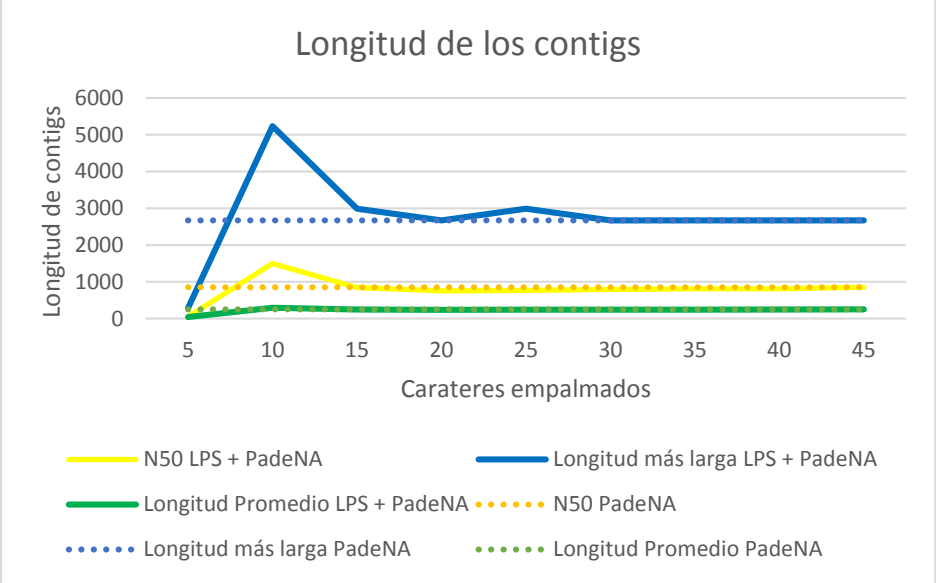

Fig. 84 Longitudes de los contigs

La Fig. 84 muestra mediante líneas discontinuas el valor de N50, promedio de las longitudes de los *contigs* y longitud del *contig* más largo obtenido por PadeNA. Las líneas continuas muestran estos resultados obtenidos para LPS + PadeNA. Podemos observar que para $n = 10$ se obtienen los *contigs* de longitudes más largos confirmando que para este valor de n la calidad del ensamble puede ser aceptada como buena.

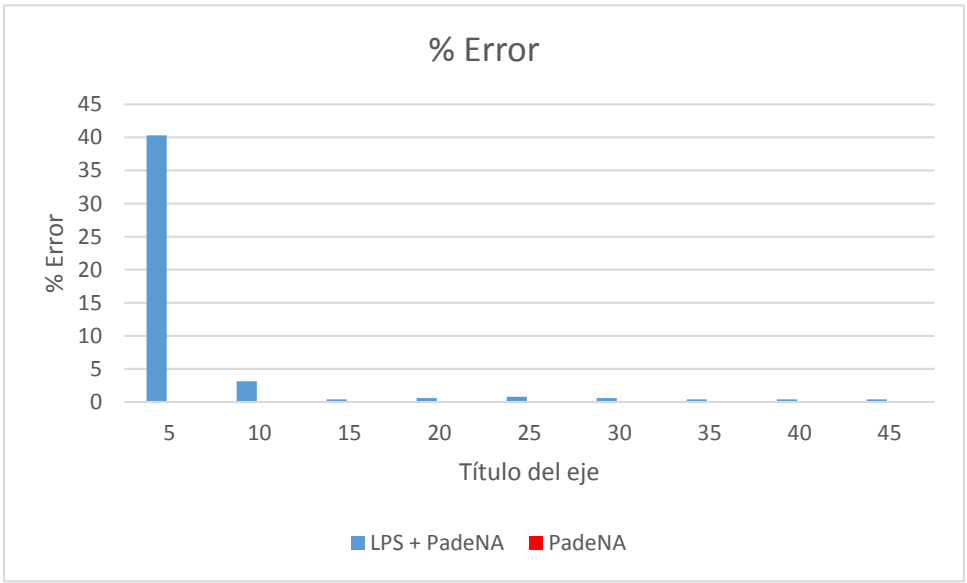

Fig. 85 Por ciento de error

La Fig. 85 muestra el por ciento de error de los *contigs* formados usando un criterio de comparación de solape exacto. Consideramos que un *contig* es correcto si este es subcadena de la cadena formada por el genoma original. Para valores de n = 5, el por ciento de error cometido es grande. Esto se debe a que se exigen pocos caracteres para empalmarse, pudiendo crearse nuevos fragmentos erróneos producto de la mezcla de pares de fragmentos que se solapen. Para n = 10 se observa que el por ciento de error se reduce considerablemente a 3.14%, pero es superior a PadeNA. El incremento del error en este ensamble comparado con los mostrados en los epígrafes 5.2.3, 5.3.3 y 5.4.3 se debe a que para este ensamble se tiene una mayor cantidad de *reads* inicialmente (49,800 *reads*) comparado con las secuenciaciones anteriores (6,500 , 18,580, 39,204 *reads* respectivamente), siendo n =10 caracteres pocos para el solape entre pares de secuencias sin que se generen nuevas secuencias erróneas. A pesar de tener un valor de error del 3.14%, solo son 13 *contigs* de los 413 siendo un ensamble de buena calidad superando a PadeNA en cuanto a cantidad de *contigs* (Fig. 82), suma de las longitudes de los *contigs* (Fig. 83), N50, longitud del *contigs* más largo, longitud media de los *contigs* (Fig. 84), decayendo en cuanto a por ciento de error (Fig. 85). Para valores de n mayor o igual a 15 podemos observar que el porciento de error cometido se reduce siendo similar a PadeNA (Fig. 85).

5.5.4 Ensamblado con *n* caracteres para LPS 1 – 20% caracteres para LPS 2

En la sección 5.5.3 mostramos que el ensamble con mejor calidad obtenido fue para n = 10 caracteres en ambas corridas de LPS. Considerando que 10 caracteres es equivalente al 20% de la longitud de fragmentos de tamaño 50 caracteres, entonces, para este experimento tomamos como configuración de parámetros como n caracteres para la primera corrida de LPS y 20% de la longitud de los *contigs* para la segunda corrida de LPS. Finalmente, aplicaremos PadeNA para mejorar la calidad del ensamble. Dado que el valor de LPS para la segunda corrida está fijado a 20%, tomemos como convenio el valor de n de la primera corrida de LPS para denotar los diferentes ensambles obtenidos en la experimentación.

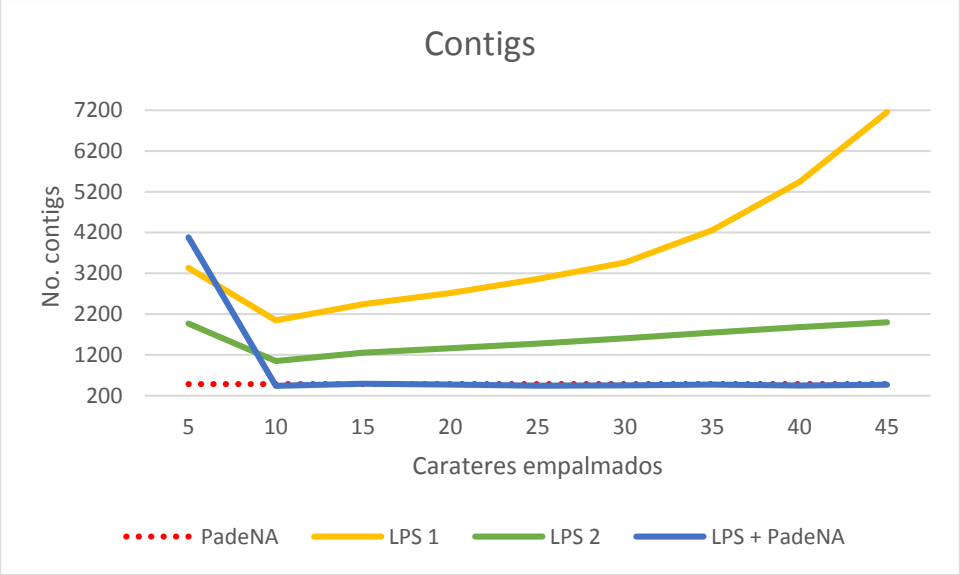

Fig. 86 Número de contigs

La Fig. 86 muestra la cantidad de *contigs* encontrados por ambas corridas de LPS y luego como mejora el resultado cuando usamos PadeNA en conjunto con LPS. Para n = 5 en la primera corrida, la cantidad de contigs obtenidos por LPS + PadeNA es superior que los obtenidos por PadeNA. La menor cantidad de se alcanza para el valor de n = 10 usando LPS en conjunto con PadeNA siendo menor que PadeNA. Para PadeNA la cantidad de *contigs* encontradas es de 484. Para la combinación de LPS + PadeNA este resultado se redujo a 440 *contigs*. Para valores de n = 15 en adelante, el comportamiento del ensamble es similar al ensamble obtenido por PadeNA.

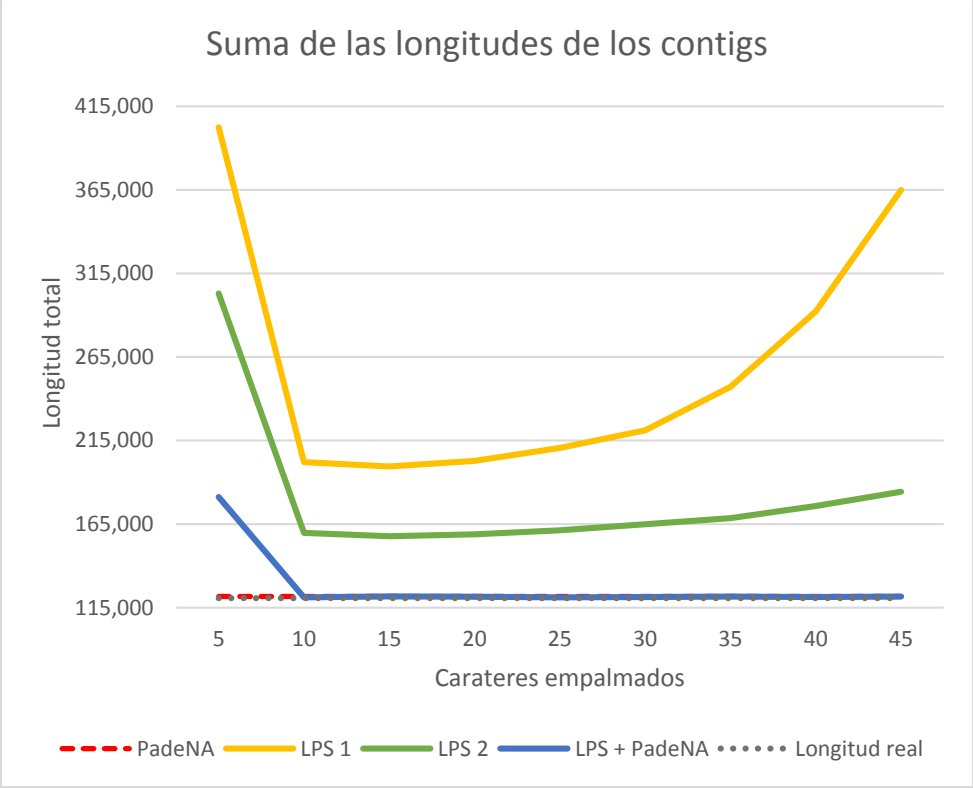

Fig. 87 Suma de las longitudes de los contigs

La Fig. 87 muestra la suma de las longitudes de los *contigs* para las dos corridas de LPS así como para LPS en conjunto con PadeNA. Para $n = 5$ la suma de los *contigs* obtenida mediante LPS + PadeNA es superior a la obtenida con PadeNA. Si observamos la Fig. 86 para este mismo valor de *n*, la cantidad de *contigs* resultantes de LPS + PadeNA es también mayor que los de PadeNA contribuyendo LPS + PadeNA más *contigs* a la suma de sus longitudes que PadeNA. El valor que se acerca más a la longitud del genoma real (120,693 pares de bases) se alcanza para $n = 10$ con LPS + PadeNA siendo 121,140 pares de bases, mejorando a la longitud obtenida por PadeNA con 121,730 pares de bases. Para valores de $n = 15$ el comportamiento del ensamble es similar al obtenido por PadeNA.

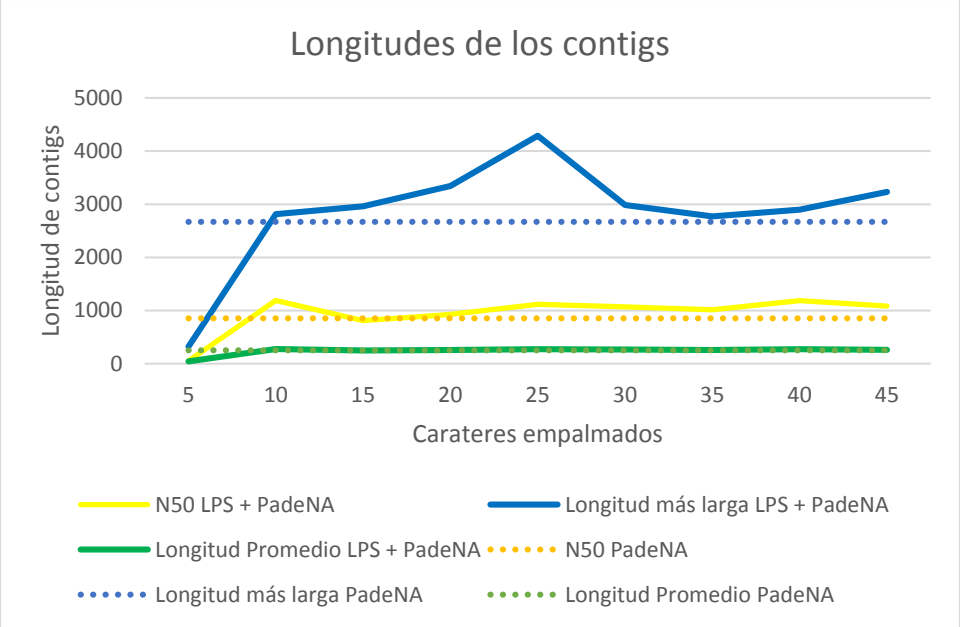

Fig. 88 Longitudes de los contigs

La Fig. 88 muestra mediante líneas discontinuas el valor de N50, promedio de las longitudes de los *contigs* y longitud del *contig* más largo obtenido por PadeNA. Las líneas continuas muestran estos resultados obtenidos para LPS + PadeNA. Observando las Fig. 86 y Fig. 87, junto a la Fig. 88, corroboramos que para n = 10 se obtiene un ensamble de mayor calidad con LPS + PadeNA comparado con PadeNA. Se ha de destacar que para n = 25 se obtienen *contigs* con longitudes largas, de manera similar al ensamble mostrado en el epígrafe 5.4.4 en la Fig. 70.

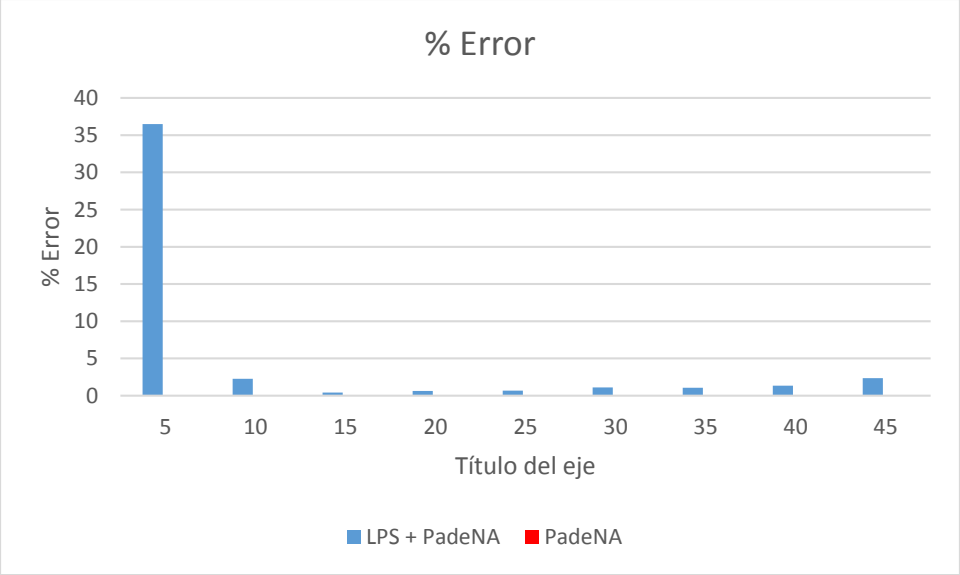

Fig. 89 Por ciento de error

La Fig. 89 muestra el por ciento de error de los *contigs* formados usando un criterio de comparación de solape exacto. Consideramos que un *contig* es correcto si este es subcadena de la cadena formada por el genoma original. Para valores de $n = 5$, el por ciento de error cometido es grande. Esto se debe a que se exigen pocos caracteres para empalmarse, pudiendo crearse nuevos fragmentos erróneos producto de la mezcla de pares de fragmentos que se solapen. Para valores de n mayores o iguales a 10 se observa que el por ciento de error se reduce a 2.72%, pero es superior al error cometido por PadeNA de 0.006%. Similar al error del ensamble mostrado en el epígrafe 5.4.3, para esta cantidad de *reads* iniciales, tener $n = 10$ caracteres para el empalme puede llevar a generar nuevas secuencias erróneas formadas a partir de la mezcla del par de fragmentos empalmados. Si se observa en la Fig. 89, a partir de $n = 15$, el error decae y éste vuelve a crecer para valores altos de $n = 45$, donde los nuevos fragmentos formados tienen menor longitud al usarse gran parte de estos para el empalme.

5.5.5 Resumen de la calidad de los ensambles obtenidos del plásmido Acaryochloris marina MBIC11017 plasmid pREB8

En este epígrafe solamente mostraremos a modo resumen los resultados obtenidos para las diferentes configuraciones de los caracteres de solapes entre pares de secuencias en los

ensambles de LPS + PadeNA mostrados anteriormente. Este epígrafe solo constituye un resumen de las medidas de por ciento de error, cantidad de *contigs*, suma de las longitudes de los *contigs*, N50, longitud del *contig* más largo, longitud media de los *contigs*, tomadas en los epígrafes anteriores para los ensambles de LPS + PadeNA.

Las líneas horizontal color rojo muestran el valor obtenido por PadeNA. Las líneas color azul oscuro muestran los resultados obtenidos por el ensamble usando *n* caracteres en la primera corrida de LPS y *n* caracteres en la segunda corrida de LPS. Las líneas color naranja muestran los resultados obtenidos por el ensamble usando *n* caracteres en la primera corrida de LPS y 20% caracteres en la segunda corrida de LPS.

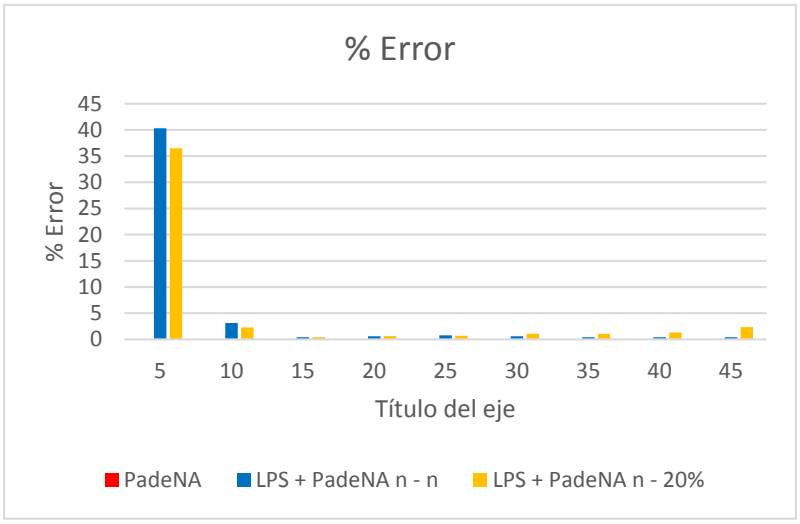

Fig. 90 Por ciento de error

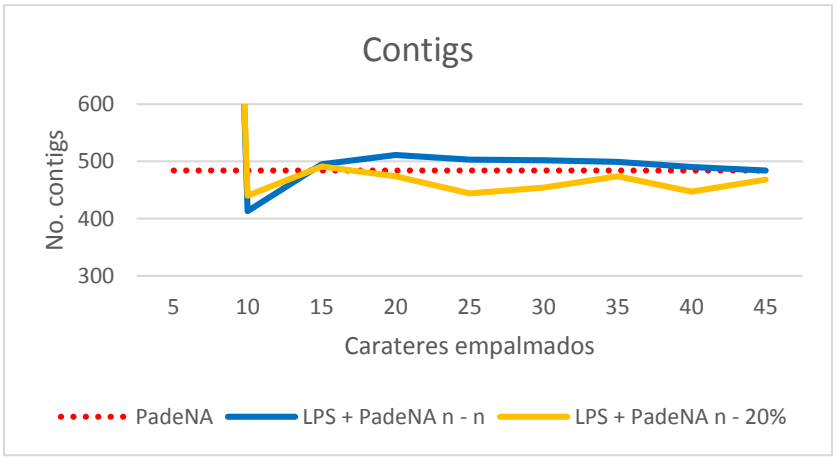

Fig. 91 Número de contigs

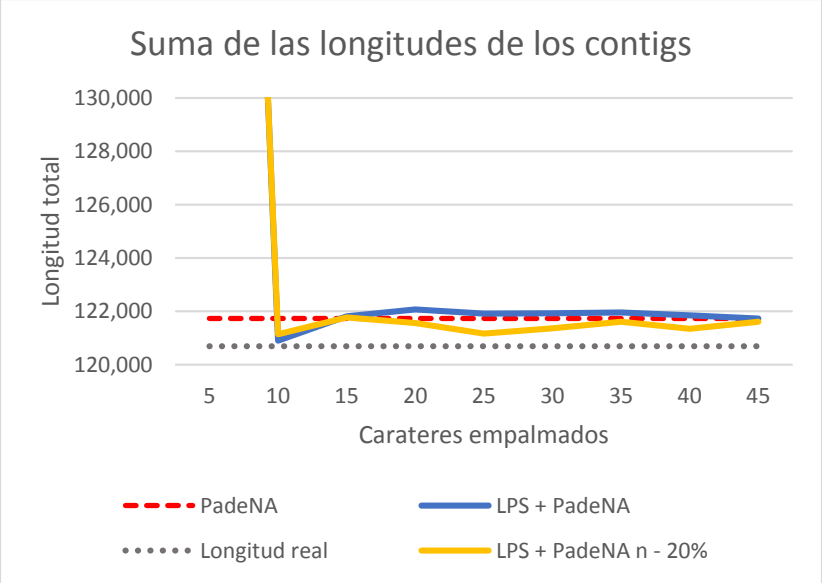

Fig. 92 Suma de las longitudes de los contigs

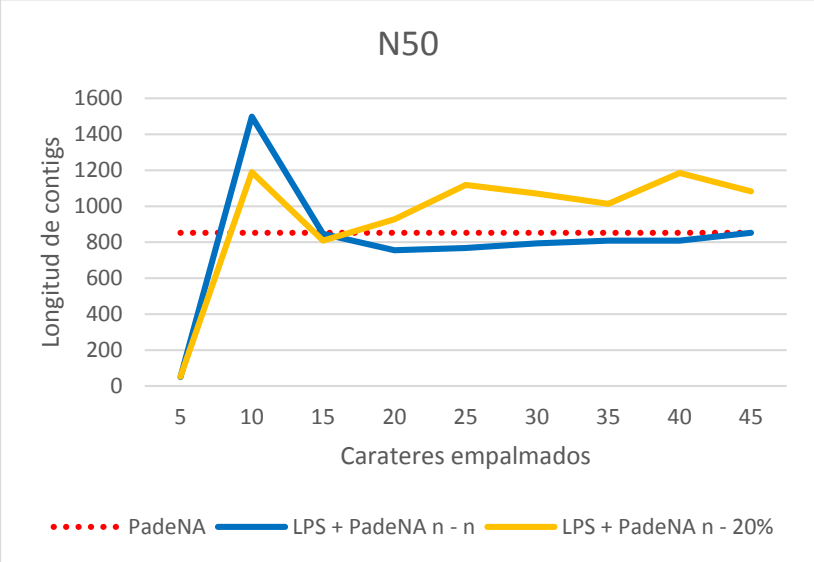

Fig. 93 N50

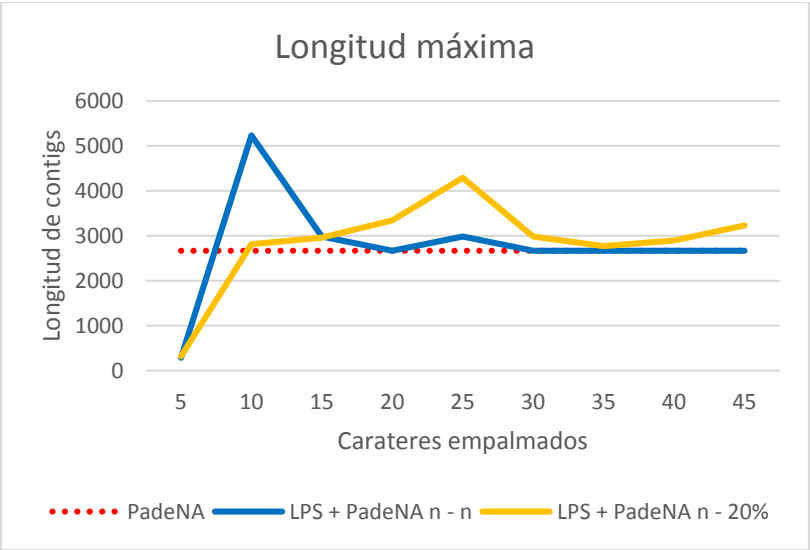

Fig. 94 Longitud del contig más largo

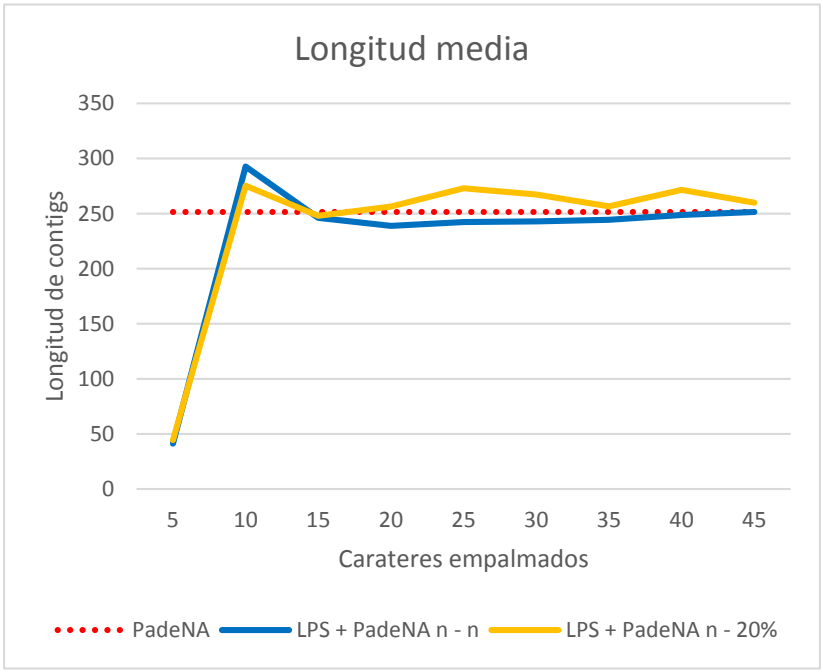

Fig. 95 Longitud media de los contigs

La Fig. 90 muestra los errores cometidos en los ensambles. Para valores de *n* bajos, de 5 caracteres, el error cometido en el ensamble es muy elevado por lo que descartamos esos ensambles. Esto se debe a que se exigen pocos caracteres para empalmarse, pudiendo crearse nuevos fragmentos erróneos producto de la mezcla de pares de fragmentos que se solapen.

Para valores de *n* = 10, en ambas combinaciones de ensamble, se obtienen los mejores valores de calidad. Esto se ve reflejado en las Fig. 90 -Fig. 95. La suma de las longitudes de los *contigs* se acerca a la longitud real de la secuencia original. La mejor combinación de valores obtenida para este caso es 10 caracteres de solape para la primera corrida de LPS y 10 caracteres de solape para la segunda corrida de LPS. Estos resultados son alcanzados con un error menor de 4%. Para este caso se obtiene una buena mejoría debido a que la cantidad de caracteres necesarios para el solape entre secuencias son lo suficientes para crear nuevas secuencias a partir de las mezclas de este par empalmado tal que no sean erróneos y lo suficiente para generar nuevas secuencias con longitudes más largas. Debemos de destacar que el error cometido en este ensamble es superior al de los ensambles con LPS del plásmido Bacillus Cereus (epígrafe 5.2.3, 5.2.4 y 5.2.5), al del plásmido Aeromonas hydrophila plasmid pRA3 (epígrafe 5.3.3 y 5.3.4) y al del plásmido Bacillus anthracis str. 'Ames Ancestor' plasmid pXO2 (epígrafe 5.4.3 y 5.4.4). Esto se debe a que la cantidad de *reads* en estas secuenciaciones es inferior a la cantidad de *reads* obtenidas para el plásmido Acaryochloris marina MBIC11017 plasmid pREB8. Para *n* = 10 se obtienen buenos ensambles pero comienzan a ser pocos caracteres para el solape entre pares de secuencias sin que se generen nuevas secuencias erróneas. A partir de *n* = 15, el error cometido decrece siendo similar a PadeNA (Fig. 90). Sólo para la configuración de *n* caracteres para la primera corrida de LPS y 20% de la longitud de los *contigs* para la segunda corrida de LPS, se observa un incremento del error a partir de *n* = 40. Esto es de esperarse dado que para estos valores de *n* la longitud de los nuevos fragmentos es más pequeña en comparación con la longitud de la secuencia original, debido a que se exigen más elementos para el empalme entre pares de secuencias.

El factor de la longitud de la nueva secuencia formada, puede observarse en el esquema de *n* elementos para la primera corrida de LPS y *n* elementos para la segunda corrida de LPS a partir de *n* = 15 en adelante. A medida que exigimos más caracteres para el solape, las nuevas secuencias formadas a partir de un par secuencias empalmadas, es menor, decrementando la calidad del ensamble. Si vemos la longitud de la secuencia original (120,693 pares de bases), la cual deseamos reconstruir, en comparación con las secuencias anteriores, (15,274 pares de bases para el plásmido Bacillus cereus, 45,909 pares de bases para el plásmido Aeromonas hydrophila plasmid pRA3 y 94,830 pares de bases para el plásmido Bacillus anthracis str.

'Ames Ancestor' plasmid pXO2), es mayor. Usando fragmentos muy cortos comparado con la longitud de la secuencia que deseamos reconstruir, resulta que la calidad del ensamble decae, incluso puede llegar a ser inferior a PadeNA, aunque muy cercano al ensamble de PadeNA como puede ser visto en las Fig. 91 - Fig. 95.

Sin embargo, usando el esquema de *n* caracteres para la primera corrida de LPS y 20% para la segunda corrida de LPS, las longitudes de los *contigs* obtenidos es mucho mayor que para *n* caracteres para la primera corrida de LPS y n para la segunda corrida de LPS, obteniéndose ensambles que son similares o que mejoren a PadeNA (Fig. 91 - Fig. 95).

5.6 Experimento 5 Ensamble de la bacteria Mycoplasma hominis

En este experimento usaremos a LPS con una secuencia de la bacteria Mycoplasma hominis "ATCC 23114", disponible en Accession NC_013511.1 (Pereyre et al., 2009). Los mycoplasmas son las bacterias que cuentan con un genoma pequeño, teniendo una longitud que oscila entre 580 mil a 1380 mil pares de bases. Para este experimento se utilizó un genoma de referencias con una longitud de 665,445 pares de bases. El objetivo de este experimento es poder evaluar la calidad de los ensambles de LPS + PadeNA utilizando ya un genoma completo. El tamaño de este genoma es 40 veces más grande que la longitud del plásmido utilizado en el experimento mostrado en el epígrafe 5.2.

En este experimento el genoma fue secuenciado sintéticamente *in silico* con cobertura de diez veces la molécula completa.

En la simulación de la secuenciación se crearon fragmentos de secuencias de longitud uniforme de 50 caracteres, considerando una sola dirección (de 3' a 5'), libre de errores, y todos los caracteres de las secuencia del plásmido están contenidas en al menos un fragmento (cobertura total). El número de fragmentos obtenidos fue de 276,140 *reads*.

Primeramente, realizaremos un ensamble usando PadeNA solamente. Los resultados obtenidos por PadeNA serán utilizados como criterio base de comparación para evaluar la calidad de la propuesta de la tesis que es inicialmente aplicar LPS previo a PadeNA. PadeNA es un algoritmo de ensamble basado en el uso de grafos de de Bruijn. El parámetro fundamental de estos algoritmos es utilizar un valor de *k* para especificar el tamaño de los *k*-

mers. En el próximo epígrafe explicaremos como buscamos un valor de *k* que brinde un buen ensamble contra el cual nos compararemos utilizando como criterio las medidas de evaluación explicadas previamente.

5.6.1 Encontrando el tamaño de los *k*-mers para PadeNA

En este epígrafe se buscará un valor de *k* para determinar cuál longitud de los *k*-mers arroja un buen resultado. Los resultados obtenidos por el valor de *k* seleccionado serán utilizados como base de la comparación del algoritmo propuesto de LPS combinado con PadeNA.

Para buscar el valor de *k*, se realiza el ensamblado de la bacteria Mycoplasma hominis con PadeNA, variando los valores de *k* en el rango de 2 a 31. Para cada ensamblado calculamos los valores de calidad descritos previamente. El valor de *k* que tomamos para comparar será el que mejor ensamble resulte. Este mismo valor de *k*, será utilizado en la fase final de LPS + PadeNA.

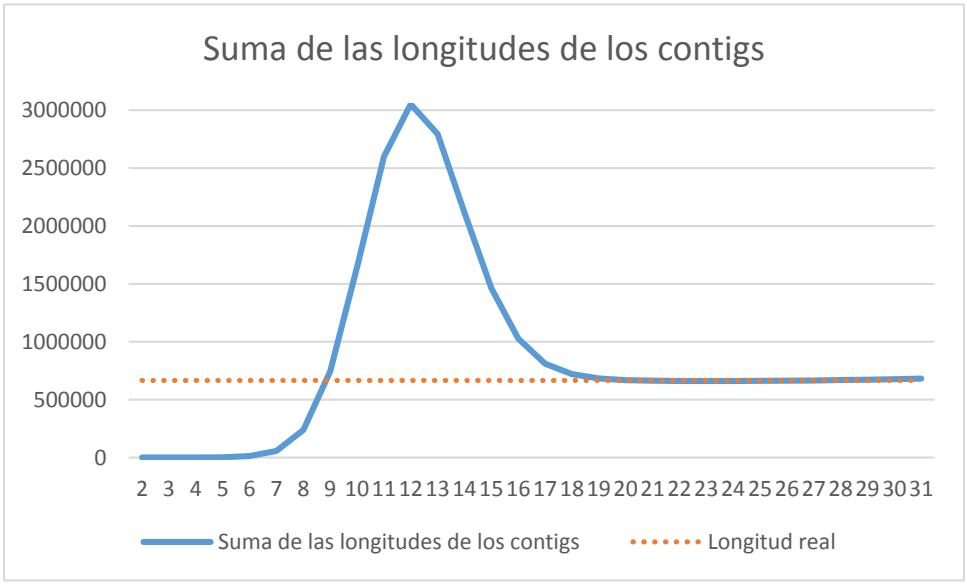

Fig. 96 Suma de las longitudes de los contigs con PadeNA

La Fig. 96 muestra la suma total de los *contigs* puestos uno detrás del otro en azul. El valor ideal a alcanzar es el que se aproxime al valor real de la longitud de la secuencia original mostrado en rojo el cual es 665,445. Los valores de *k* que dan un ensamble con una medida

que se aproxima a este valor están en el rango de 19 a 31 siendo este el intervalo de interés para seleccionar el valor de *k*.

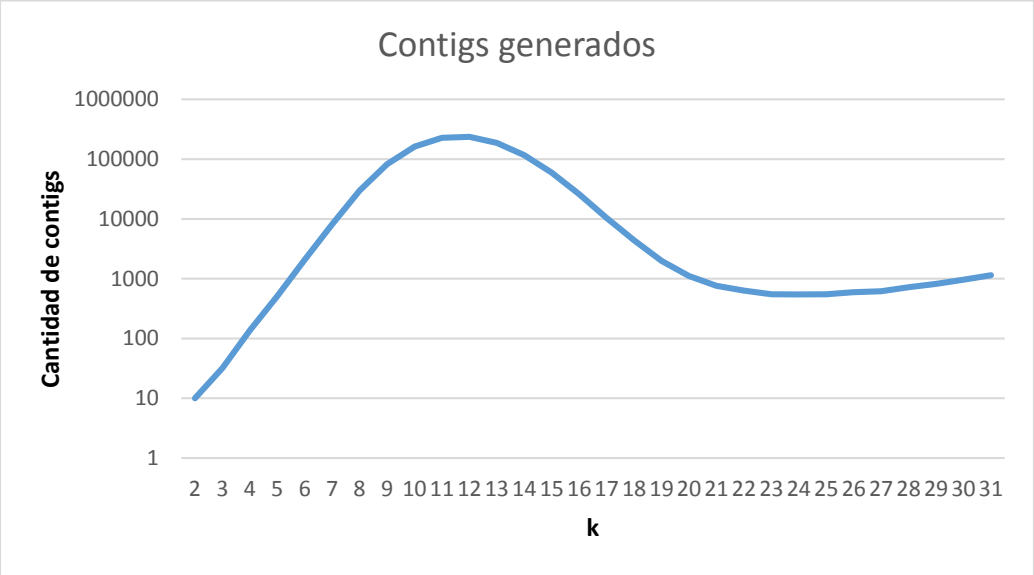

Fig. 97 Contigs generandos con PadeNA

La Fig. 97 contiene la cantidad de *contigs* generados con PadeNA utilizando los diferentes valores de *k*. Para valores de *k* entre 2 y 5 se obtienen pocos fragmentos pero la suma de las longitudes está muy por debajo de la longitud de la secuencia si observamos la Fig. 96. Para valores de *k* en el intervalo de interés comprendido entre 21 a 27 se observa que se obtiene una cantidad de *contigs* inferior que para el resto de valores de *k*, con excepción de valores entre 2 y 6, pero estos los habíamos desechado. Notemos que el intervalo de 21 a 27 está dentro del intervalo inicial de valores candidatos (entre 19 a 31) por lo que nos quedamos con este conjunto de valores de *k* posibles.

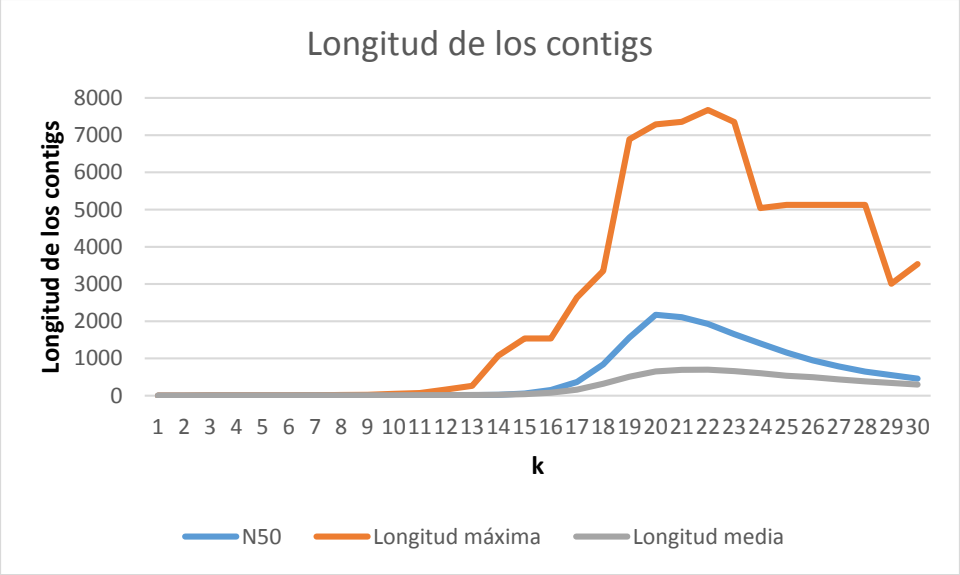

Fig. 98 Medidas de N50, Longitud del contig más largo y longitud media de los contigs con PadeNA

La Fig. 98 muestra los valores de las longitudes de los *contigs* N50, longitud del *contig* más largo y la longitud media de todos los *contigs* respecto al valor *k* de los *k*-mers. Un ensamble que pueda ser interpretado por humanos necesita que los *contigs* encontrados tengan una longitud larga por lo que los valores de *k* serán los que muestren mayor valor de longitud. Centrándonos en el conjunto de valores posibles, el intervalo de valores de *k* posible está entre 21 y 27.

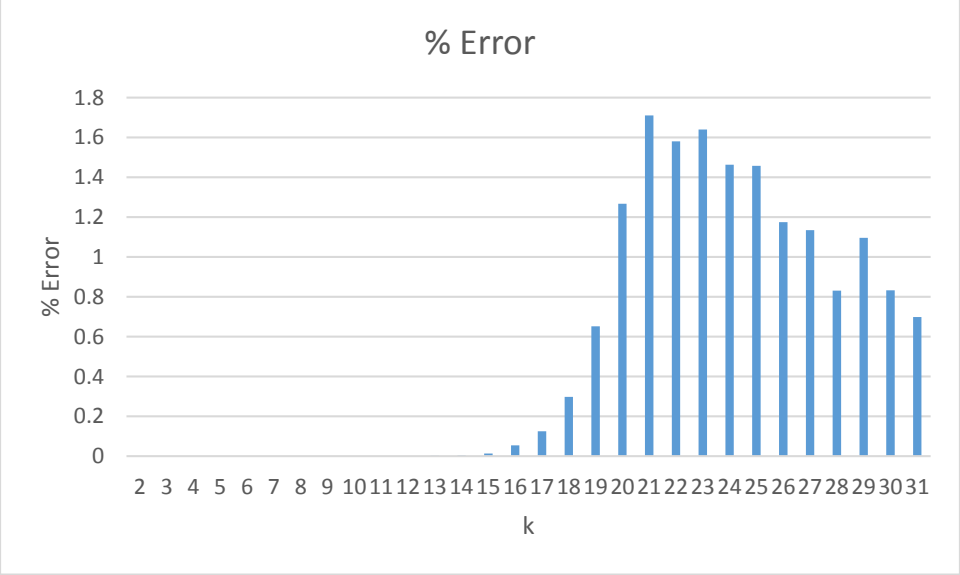

Fig. 99 Por ciento de error con PadeNA

La Fig. 99 muestra los errores en los ensambles. En el intervalo candidato de valores de *k* comprendido entre 21 y 27 vemos que el menor valor de error cometido es para *k* = 27. Así mismo donde se obtiene un ensamble con suma de las longitudes de los *contigs* con menor diferencia a la longitud del genoma real es para el valor de *k* = 27.

En el epígrafe 5.2.1 se realizó el ensamble con PadeNA del plásmido Bacillus cereus, en el cual se obtuvo que la resta entre la suma de las longitudes de los *contigs* y la longitud de la secuencia real es mayor que 2,000 caracteres. A diferencia, en el experimento mostrado en este epígrafe, la resta entre las longitudes es de 160 caracteres siendo muy pequeña en comparación con el anterior experimento. Este mismo valor de *k* = 27 será utilizado en la corrida de LPS + PadeNA que describiremos a continuación (Tabla 10).

5.6.2 Ensamblado con LPS

Similarmente a la forma en que hicimos el ensamble con LPS + PadeNA para el Bacillus cereus, donde probamos con los esquemas:

- *n* caracteres para LPS 1 - *n* caracteres para LPS 2: Para este experimento fijamos un valor de *n* para la primera corrida de LPS como para una segunda corrida. Los valores de *n* a probar son $n \in \{5,10,15,20,25,30,35,40,45\}$.
- *n* caracteres para LPS 1 – 30% para LPS 2: En este caso se usará una cantidad de *n* de caracteres para el solape en la primera corrida de LPS y luego se usará el 30% de la longitud del *read* más pequeño para la segunda corrida de LPS. El valor del 30% fue fijado debido a que con este parámetro de LPS se obtuvieron los ensambles con una buena calidad. La elección del valor del 30% se basa con los resultados mostrados en la próxima sección 5.6.3.

5.6.3 Ensamblado con *n* caracteres para LPS 1 - *n* caracteres para LPS 2

En este esquema se utilizará la misma cantidad de caracteres de solape para ambas corridas de LPS.

La Fig. 100 muestra la cantidad de *contigs* encontrados por ambas corridas de LPS y luego como mejora el resultado cuando usamos PadeNA en conjunto con LPS. La menor cantidad

de *contigs* se alcanza para el valor de *n* = 15 en ambas corridas de LPS en conjunto con PadeNA. Este valor es muy cercano al encontrado por PadeNA siendo este igual a 624 *contigs*, 7 *contigs* más que PadeNA. A partir de *n* = 15, LPS + PadeNA se comporta similar a PadeNA teniendo valores ligeramente superiores respecto a la cantidad de *contigs* encontrados.

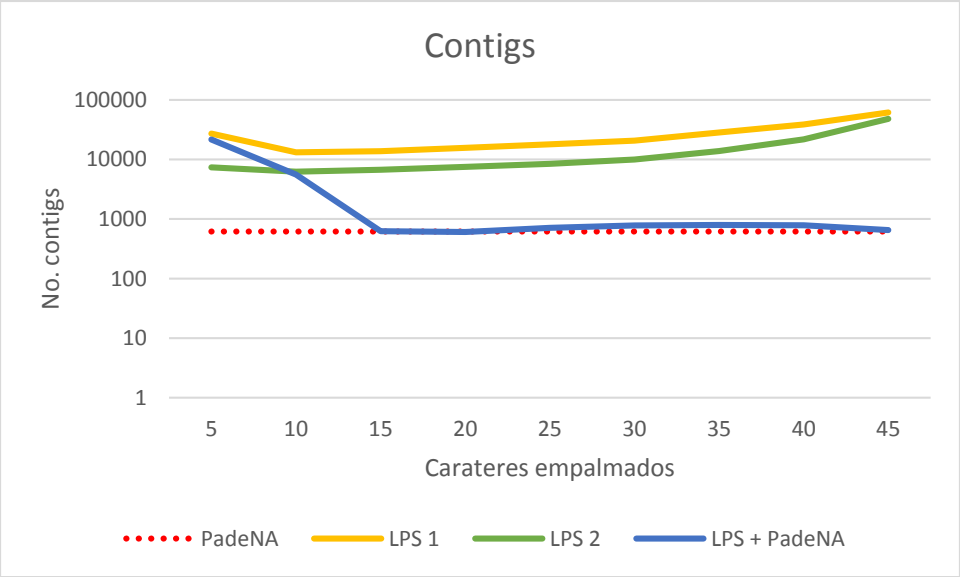

Fig. 100 Número de contigs

La Fig. 101 muestra la suma de las longitudes de los *contigs* para las dos corridas de LPS así como para LPS en conjunto con PadeNA, vemos que el valor que se acerca más a la longitud del genoma real se alcanza para *n* = 15 con LPS + PadeNA siendo este valor de 665,976 pares de bases, ligeramente superior a PadeNA teniendo una buena calidad este ensamble para este valor de *n*. A partir de *n* = 15, LPS + PadeNA se comporta similar a PadeNA teniendo valores y muy próximos a la longitud del genoma original.

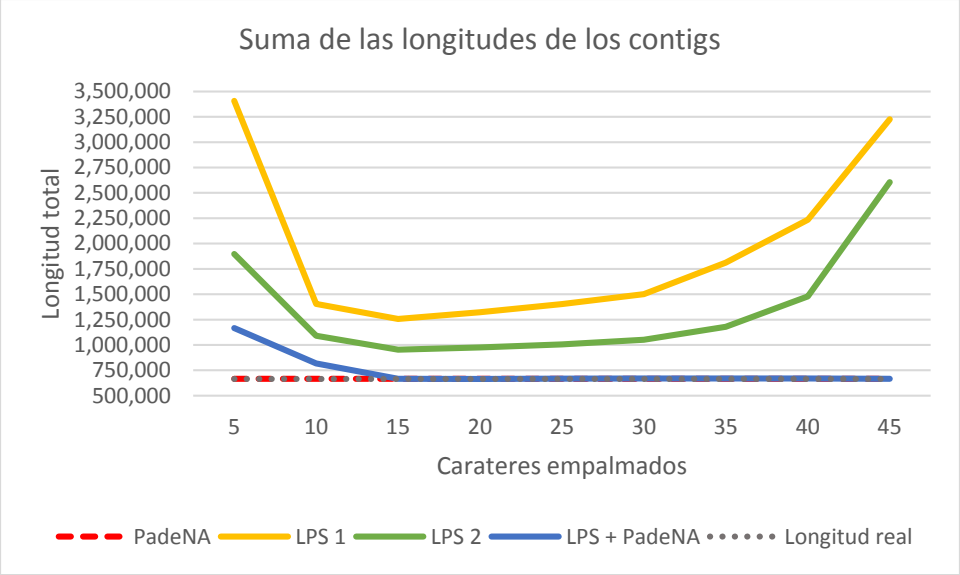

Fig. 101 Suma de las longitudes de los contigs

En las figuras Fig. 100 y Fig. 101 mostramos como aplicar PadeNA a las dos corridas de LPS mejora la calidad del ensamble substancialmente, por lo que solo compararemos los valores de las longitudes de los *contigs* obtenidos por PadeNA con LPS + PadeNA (Fig. 102).

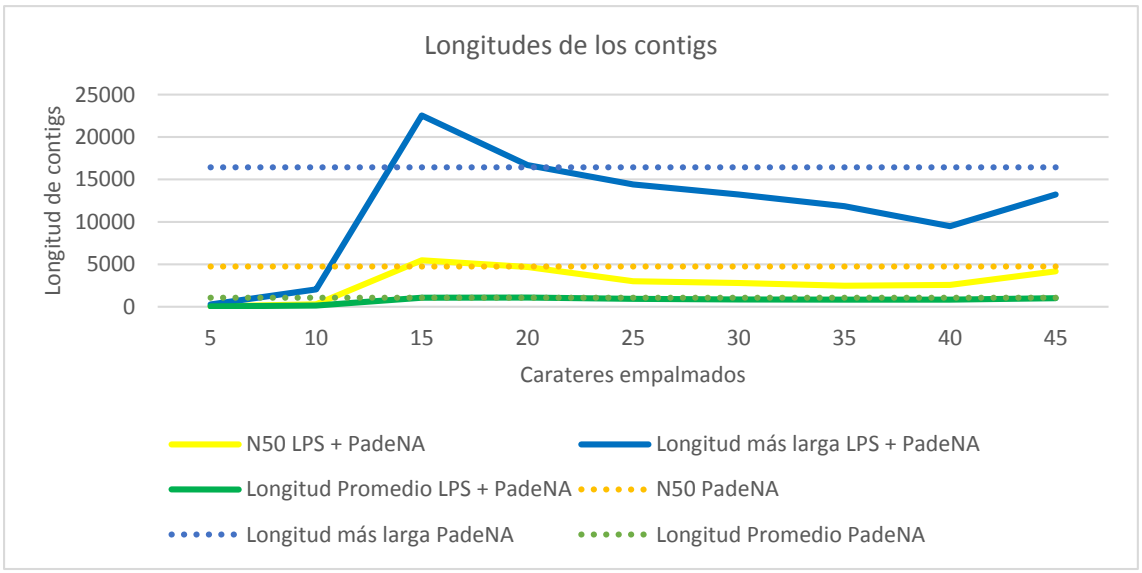

Fig. 102 Longitudes de los contigs

La Fig. 102 muestra mediante líneas discontinuas el valor de N50, promedio de las longitudes de los *contigs* y longitud del *contig* más largo obtenido por PadeNA. Las líneas continuas

muestran estos resultados obtenidos para LPS + PadeNA. Podemos observar que para *n* = 15 se obtienen los *contigs* de longitudes más largos confirmando que para este valor de *n* la calidad del ensamble puede ser aceptada como buena.

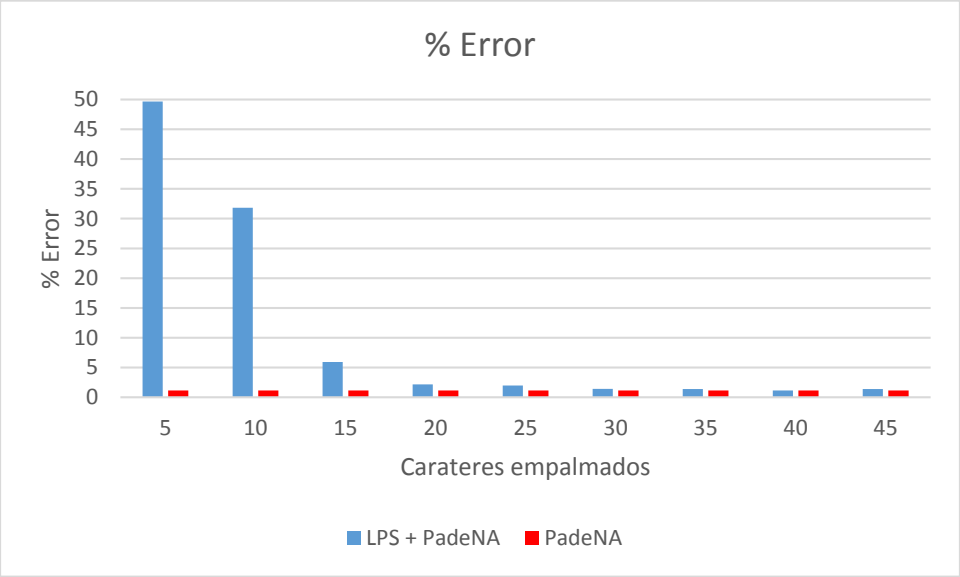

Fig. 103 Por ciento de error

La Fig. 103 muestra el por ciento de error de los *contigs* formados usando un criterio de comparación de solape exacto. Consideramos que un *contig* es correcto si este es subcadena de la cadena formada por el genoma original.

Para valores de *n* de 5 y 10, el por ciento de error cometido es grande. Esto se debe a que se exigen pocos caracteres para empalmarse, pudiendo crearse nuevos fragmentos erróneos producto de la mezcla de pares de fragmentos que se solapen. Este error va decreciendo a medida que se exijan más caracteres para el solape. Para *n* = 15 se observa que el por ciento de error ya se reduce considerablemente, aunque es superior al cometido por PadeNA, pero es aceptable este error para obtener un ensamble aceptable si consideramos los valores de calidad mostrados en Fig. 100, Fig. 101 y Fig. 102. Es de esperar que el error de LPS + PadeNA es superior al de PadeNA ya que los errores cometidos por LPS se acumulan con los errores cometidos por PadeNA.

5.6.4 Ensamblado con *n* caracteres para LPS 1 – 30% caracteres para LPS 2

En esta configuración utilizaremos un número *n* requerido para el solape entre pares de secuencias en la primera corrida de LPS. En la sección anterior 5.6.3 mostramos que el ensamble de mejor calidad se encontró utilizando el valor de *n* = 15. Como el 30% de 50 caracteres es igual a 15 caracteres, se propone utilizar para la segunda corrida de LPS un valor de solape de un 30% de la longitud, en lugar de usar un número n fijo para la segunda corrida de LPS. Finalmente se aplicará PadeNA a los ensambles obtenidos

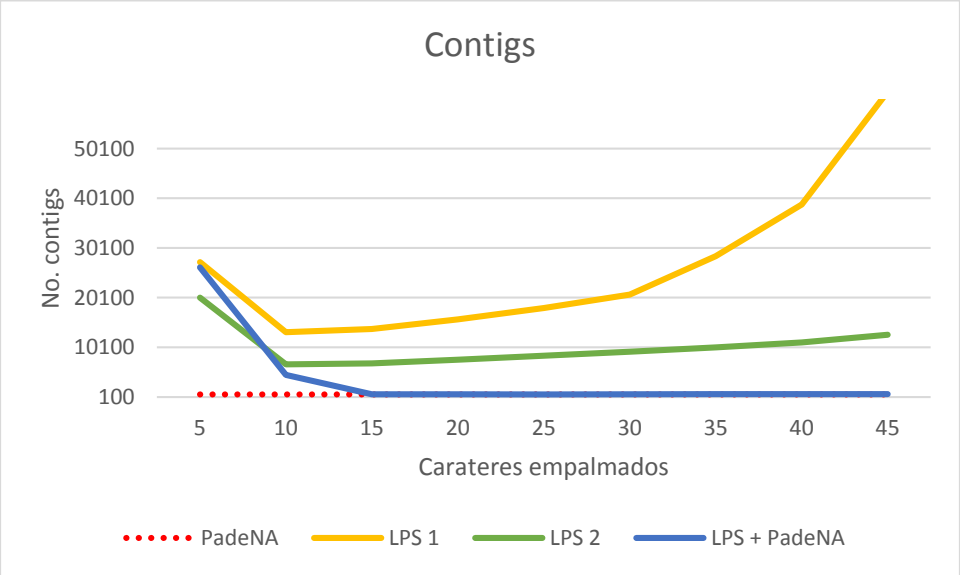

Fig. 104 Número de contigs

La Fig. 104 muestra la cantidad de *contigs* encontrados por ambas corridas de LPS y luego como mejora el resultado cuando usamos PadeNA en conjunto con LPS. A partir de *n* = 15 la cantidad de *contigs* por LPS + PadeNA encontrados es similar a los encontrados por PadeNA. Los dos valores menores se alcanzan para *n* = 25 o 50% para la primera corrida de LPS y 30% para la segunda, con 635 *contigs*, y *n* = 30 o 60% para la primera corrida de LPS y 30% para la segunda, con 639 *contigs*. Los valores obtenidos son 18 y 22 *contigs* más que lo que formó PadeNA respectivamente, lo cual es una diferencia muy pequeña.

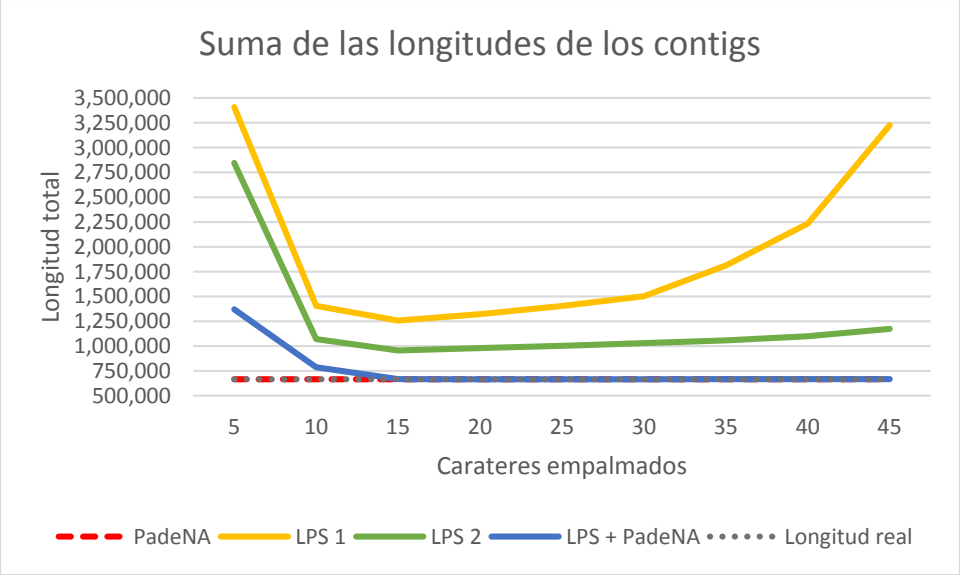

Fig. 105 Suma de las longitudes de los contigs

La Fig. 105 muestra la suma de las longitudes de los *contigs* para las dos corridas de LPS así como para LPS en conjunto con PadeNA. A partir de n = 15 para la primera corrida de LPS y n = 30% para la segunda se observa que LPS + PadeNA se comporta similar a PadeNA. El valor que se acerca más a la longitud del genoma real (665,445) se alcanza para n = 25 o 50% para la primera corrida de LPS y 30% para la segunda, con 665,531 pares de bases, y n = 30 o 60% para la primera corrida de LPS y 30% para la segunda, con 665,581 pares de bases. Ambos valores mejoran la longitud encontrada por PadeNA (665,605 pares de bases).

En las figuras Fig. 104 y Fig. 105 mostramos como aplicar PadeNA a las dos corridas de LPS mejora la calidad del ensamble substancialmente, por lo que en las próximas medidas de esta sección solo compararemos a PadeNA con LPS + PadeNA.

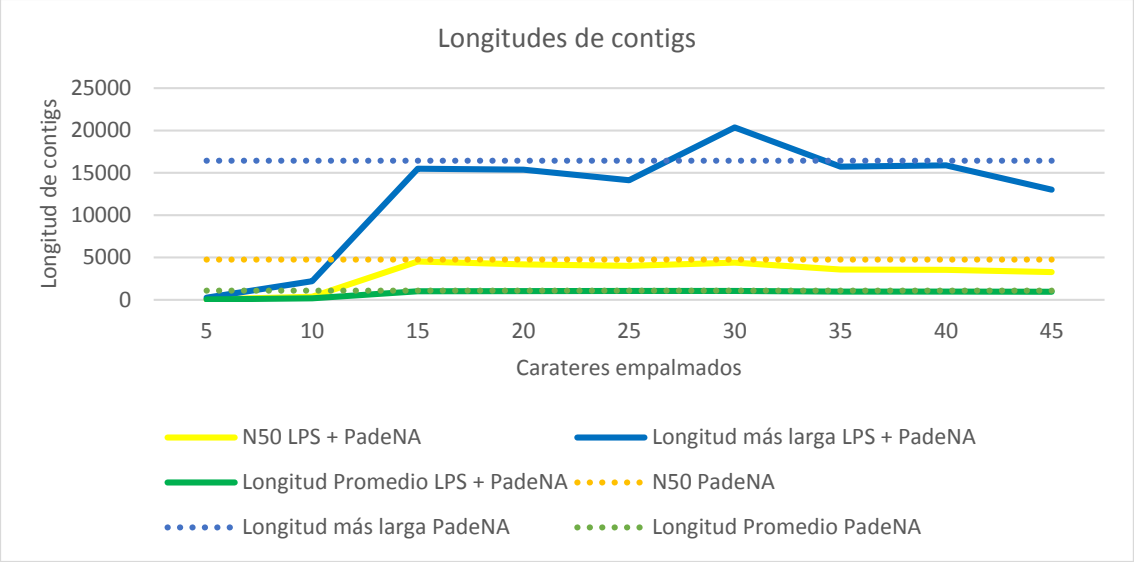

Fig. 106 Longitudes de contigs

La Fig. 106 muestra mediante líneas discontinuas el valor de N50, promedio de las longitudes de los *contigs* y longitud del *contig* más largo obtenido por PadeNA. Las líneas continuas muestran estos resultados obtenidos para LPS + PadeNA. A partir de n = 15 se observa que las longitudes son cercanas a las obtenidas por PadeNA. Previamente teníamos que el mejor ensamble se lograba con un valor de n = 25 y el segundo mejor ensamble se lograba para n = 30 pero con pequeñas diferencias en las medidas mostradas en las figuras Fig. 104 y Fig. 105. Observando la Fig. 106 vemos que el *contig* con longitud mayor se alcanza para n = 30 siendo más fácil de interpretar este resultado al ojo humano que para n = 25 por lo que preferimos este ensamble.

La Fig. 107 muestra el por ciento de error de los *contigs* formados. Para valores de n de 5 y 10, el por ciento de error cometido es grande. Esto se debe a que se exigen pocos caracteres para empalmarse, pudiendo crearse nuevos fragmentos erróneos producto de la mezcla de pares de fragmentos que se solapen. A partir de n = 15 el error decrece considerablemente aunque es superior al cometido por PadeNA debido a que los errores cometidos por LPS se acumulan con los errores cometidos por PadeNA. Para n = 30 encontramos que el error es mucho menor que el obtenido por el ensamble mostrado con un esquema de n caracteres para la primera corrida de LPS y n caracteres para la segunda corrida de LPS mejorando la calidad del ensamble.

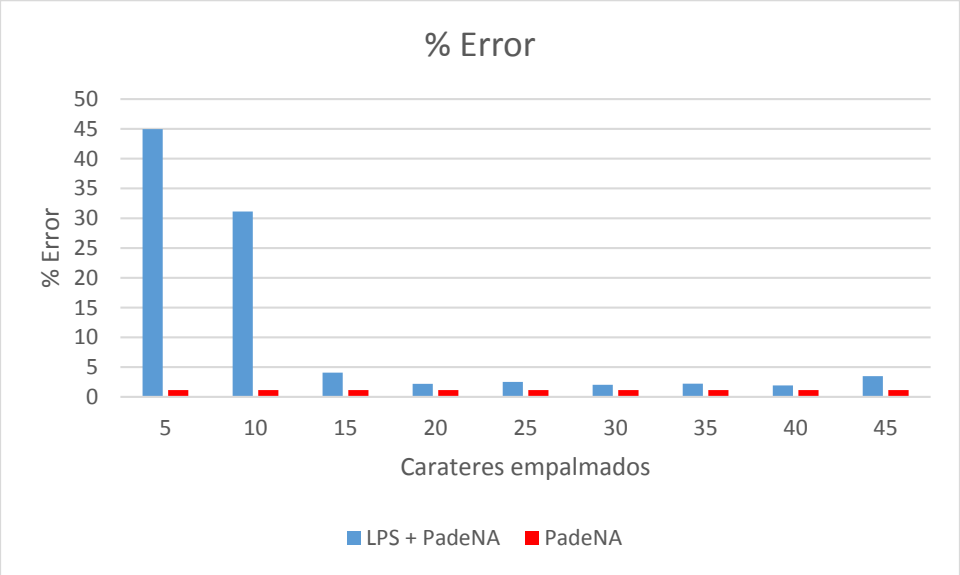

Fig. 107 Por ciento de error

5.6.5 Datos resumidos de la calidad de los ensambles obtenidos de la bacteria Mycoplasma hominis

En esta sección mostraremos los datos obtenidos de los ensambles con mejor calidad resumidos de PadeNA, de LPS con 15 caracteres la primera corrida y 15 para la segunda corrida + PadeNA (denotado por LPS(15) + LPS (15) + PadeNA) y 60% caracteres para primera corrida de LPS + 30% para segunda corrida de LPS + PadeNA (denotado por LPS(60%) + LPS (30%) + PadeNA).

La Tabla 10 muestra los valores de calidad obtenidos por los ensambles usando PadeNA, LPS con 15 caracteres de solape para la primera ejecución y 15 caracteres para la segunda ejecución más PadeNA; y finalmente LPS con 30 caracteres de solape equivalente al 60% de la longitud para la primera ejecución y 30% caracteres de la longitud para la segunda ejecución más PadeNA. Como convenio de la tabla, los valores representados en letra negrita representan el mejor resultado y los subrayados el peor resultado. El objetivo de esta tabla es mostrar que para medidas donde es mejor PadeNA, el comportamiento de ambos ensambles de LPS + PadeNA es similar al de PadeNA, en cambio, donde gana LPS + PadeNA, se nota más la diferencia con PadeNA en la mayoría de las medidas.

Tabla 10 Comparación de los valores de calidad del ensamble de la bacteria mycoplasma hominis

	Contigs	Longitud más larga	N50	Promedio de las longitudes	% Error	Suma de las longitudes de los contigs	Longitud real
PadeNA *k* = 27	**617**	16435	4753	**1078.78**	**1.13**	665605	665445
LPS *n* = 15 + LPS *n* = 15 + PadeNA	624	**22538**	**5472**	1067.27	5.93	665976	665445
LPS *n* = 60% + LPS *n* = 30% + PadeNA	639	20366	4388	1041.60	2.03	**665581**	665445

En estos tres ensambles se aprecia que la cantidad de *contigs* generados son muy parecidos, siendo PadeNA el que menos *contigs* encontró, sin embargo hay que observar otras medidas para ver la calidad de estos *contigs*.

LPS es una algoritmo el cual a partir de *reads* de longitudes cortas genera *reads* de longitudes más largas. Debido a que la generación de *reads* más largos ayuda a que en el proceso de ensamblado se obtengan *contigs* más largos. Las longitudes de los *contigs* obtenidos en ambos casos de estudio (epígrafe 5.2 y epígrafe 0) de LPS + PadeNA superan a PadeNA lo cual ayuda a una más fácil la interpretación de los resultados para un experto humano. Se prefiere tener los *contigs* con una longitud mayor sobre *contigs* pequeños.

Para la medida de N50, se obtiene que LPS(15) + LPS (15) + PadeNA tiene el valor más elevado. Esto se debe a que al exigirse más caracteres para el solape en este último, las nuevas secuencias generadas tendrán una menor longitud repercutiendo en la longitud de los *contigs* reduciendo su tamaño. La medida de N50 de LPS(15) + LPS (15) + PadeNA es superior a PadeNA por el aporte de la longitud del *contig* más largo. Este tiene una diferencia aproximada de 6,000 caracteres comparado con PadeNA. Si también tomamos en cuenta que el promedio de las longitudes de los *contigs* de LPS(15) + LPS (15) + PadeNA es superior que si usamos LPS(60%) + LPS(30%) + PadeNA, obtenemos que la longitud del *contig* que se toma para medir el N50 sea mayor para la primera configuración de LPS. Recordemos por definición de N50 que es la longitud del *contig* donde la suma de las longitudes de los *contigs* más largos es del al menos el 50% de la suma total. Sin embargo para LPS(60%) + LPS(30%) + PadeNA, a pesar de tener la longitud del *contig* más largo es aproximadamente 4,000 caracteres superior a PadeNA, como el promedio de las longitudes de los *contigs* es menor,

hace que se requieran más *contigs* para llegar al *contig* cuya longitud sumada con las longitudes de los *contigs* más largos sea de al menos el 50% de la suma total. La Fig. 108 nos da una idea del comportamiento de la medida N50. La longitud de las barras representa la longitud de los *contigs*. En color azul se muestra la longitud de los *contigs* obtenidos mediante LPS(15) + LPS (15) + PadeNA y en rojo la longitud de los *contigs* obtenidos mediante LPS(60%) + LPS (30%) + PadeNA.

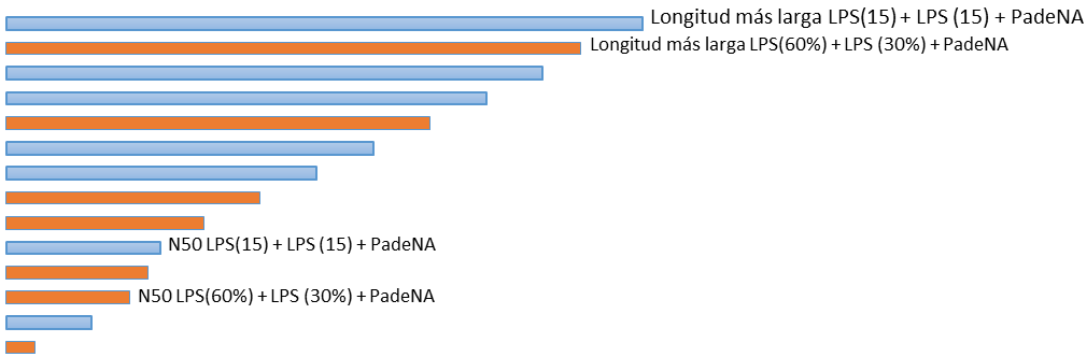

Fig. 108 N50 para LPS(15) + LPS (15) + PadeNA y LPS(60%) + LPS (30%) + PadeNA

Respecto al error cometido por en el ensamble, la diferencia entre las longitudes de LPS(15) + LPS(15) + PadeNA y LPS(60%) + LPS(30%) + PadeNA se debe a que al exigir más caracteres para el solape, es más probable que las secuencias nuevas creadas en las mezclas pertenezcan a la secuencia original, reduciéndose el error final. Esto se aprecia que LPS(15) + LPS(15) + PadeNA tiene el mayor error por exigir menos caracteres para el solape entre secuencias, LPS(60%) + LPS(30%) + PadeNA reduce este error al requerir más caracteres para el solape. Utilizar muchos caracteres para solape en la primera corrida de LPS hace que se generen secuencias con pocos errores, luego disminuir la cantidad de caracteres para el solape comparado hace que se generen secuencias largas con un error menor comparado si usáramos la cantidad de caracteres pequeña en las dos corridas de LPS. Ambas configuraciones de LPS tiene un error superior a PadeNA ya que los errores cometidos por LPS son arrastrados por PadeNA, pero este error se reduce cuando LPS comete menos errores. A pesar de estas diferencias de errores se aprecia que están cerca uno de otro.

A pesar de que LPS(60%) + LPS (30%) + PadeNA tiene valores de N50 y promedio de las longitudes más pequeñas que los demás, la suma total de las longitudes de los *contigs* es la

más cercana a la longitud del genoma real, lo cuál es nuestro objetivo final. Esto indica que los fragmentos de secuencias de este ensamble tienen pocos caracteres que se solapen entre pares de secuencias en comparación con los fragmentos obtenidos mediante PadeNA y LPS(15) + LPS (15) + PadeNA. Si a esto vemos que la longitud del *contig* más largo supera aproximadamente en 4,000 caracteres a PadeNA, además de que la cantidad de *contigs* encontrados sea cercana a los encontrados por PadeNA y que el error cometido es bajo, hace que la calidad de este ensamble sea buena y práctica para la interpretación mediante un experto humano.

5.7 Resumen y conclusiones de los resultados experimentales

En este epígrafe mostraremos en forma de resumen los resultados obtenidos usando LPS combinado con PadeNA de los experimentos anteriormente mostrados en los epígrafes del 5.2 al 5.6.

En la Tabla 11 comparamos los ensambles obtenidos con PadeNA, con los obtenidos con LPS + PadeNA. En la Tabla 11 separamos cada experimento mediante una línea gruesa. En cada experimento comparamos los ensambles obtenidos mediante PadeNA con los ensambles obtenidos con LPS usando el esquema de *n* caracteres para la primera corrida y *n* caracteres para la segunda corrida y el esquema de *n%* para la primera corrida y *n%* para la segunda corrida. Los experimentos se encuentran organizados por la longitud real de la secuencia que se desea obtener. Las secuencias están ordenadas de menor a mayor. Este valor lo podemos ver en la última columna de la Tabla 11. Los valores de calidad de los ensambles mostrados son los mejores valores que se obtuvieron para cada esquema. En letra negrita resaltamos el valor de mejor calidad entre los tres ensambles y subrayamos los valores de peor calidad. La última columna de la tabla correspondiente a 'Longitud real' muestra la longitud del genoma que se desea reconstruir, como es un valor de referencia para comparar con la 'Suma de las longitudes de los *contigs'* obtenidos por los ensambladores, ninguno de estos valores aparecerá subrayado o en negrita y es por eso que esta columna la diferenciamos del resto mediante el color gris. Para el primer ensamble Bacillus cereus ATCC 14579 plasmid pBClin15 no mostramos el esquema de *n* caracteres para la primera corrida y *n*% para la segunda ya que el ensamble con el 20% de caracteres para ambas corridas y el ensamble obtenido con 10 caracteres para la primera corrida y 20% de los caracteres para la

segunda es el mismo. Recordemos que para fragmentos de tamaño 50, el 20% son 10 caracteres.

Tabla 11 Comparación de la calidad de los ensambles

		Contigs	Long máxima	N50	Long media	% Error	Suma de las longitudes de los contigs	Longitud real
Bacillus cereus ATCC 14579 plasmid pBClin15	PadeNA *k* = 17	175	551	183	152	**0**	17199	15274
	LPS *n* = 10 + LPS *n* = 10 + PadeNA	**47**	**1400**	**632**	**408**	0.42	**15780**	15274
	LPS *n* = 20% + LPS *n* = 20% + PadeNA	95	831	357	245	0.21	16284	15274
Aeromonas hydrophila plasmid pRA3	PadeNA *k* = 15	577	550	102	85.27	**0.005**	49206	45909
	LPS *n* = 10 + LPS *n* = 10 + PadeNA	**265**	**1060**	**275**	**174.77**	0.377	**46315**	45909
	LPS *n* = 10 + LPS *n* = 20% + PadeNA	379	**1060**	173	124.89	0.791	47334	45909
Bacillus anthracis str. 'Ames Ancestor' plasmid pXO2	PadeNA *k* = 17	294	2747	689	335.33	0.003	98589	94830
	LPS *n* = 10 + LPS *n* = 10 + PadeNA	**135**	**3614**	**1221**	**709.57**	**0**	**95792**	94830
	LPS *n* = 10 + LPS *n* = 20% + PadeNA	170	3225	920	565.51	**0**	96138	94830
Acaryochloris marina MBIC11017 plasmid pREB8	PadeNA *k* = 17	484	2669	852	251.5	**0.006**	121730	120693
	LPS *n* = 10 + LPS *n* = 10 + PadeNA	**413**	**5233**	**1498**	**292.74**	3.14	**120903**	120693
	LPS *n* = 10 + LPS *n* = 20% + PadeNA	440	2812	1190	275.31	2.72	121140	120693
Mycoplasma hominis "ATCC 23114"	PadeNA *k* = 27	**617**	16435	4753	**1078.78**	**1.13**	665605	665445
	LPS *n* = 15 + LPS *n* = 15 + PadeNA	624	**22538**	**5472**	1067.26	5.93	665976	665445
	LPS *n* = 60% + LPS *n* = 30% + PadeNA	639	20366	4388	1041.60	2.03	**666581**	665445

A partir de las medidas de calidad mostradas, observamos que LPS presenta buenos resultados para reconstruir secuencias de longitudes cortas como son las de Bacillus cereus

ATCC 14579 plasmid pBClin15 y Aeromonas hydrophila plasmid pRA3. La mejora de la calidad de estos ensambles para medidas como cantidad de *contigs*, longitud del *contig* más largo, N50 y promedio de las longitudes, mejora en dos veces a los resultados obtenidos con PadeNA. También se observa que la suma de las longitudes de los *contigs* se aproxima a la longitud de la secuencia original.

Por el contrario, si vemos la última secuencia que reconstruimos correspondiente a Mycoplasma hominis "ATCC 23114". Solo algunos valores de calidad no superan a PadeNA. La mejora la podemos observar en medidas como la longitud del *contig* más largo y el N50. También se logra obtener que la suma de las longitudes de los *contigs* se aproxime más a la longitud de la secuencia que se desea reconstruir. Para el resto de los valores de calidad, podemos observar que se mantienen cercanos a los obtenidos con PadeNA.

Una característica del uso de LPS es que el error cometido aumenta ligeramente respecto a PadeNA, aunque su valor es cercano a PadeNA. Esto se debe a que al aplicar LPS con PadeNA, los errores cometidos en los ensambles de LPS, son arrastrados por PadeNA. Sin embargo, para el ensamble de Bacillus anthracis str. 'Ames Ancestor' plasmid pXO2, al aplicar LPS, se eliminó el error. Como LPS al generar fragmentos más largos que la longitud de los *reads* iniciales, trata de reducir la cantidad de secuencias que sean una repetición. En este caso LPS ayudó a mejorar el resultado de PadeNA al tratar estos casos y no generar fragmentos erróneos.

La generación de fragmentos de secuencias de longitudes más largas a partir de *reads* cortos mediante LPS (del orden de los 100,000 pares de bases), obtiene buenos resultados para reconstruir secuencias de longitudes cortas. Sin embargo, para reconstruir secuencias de longitudes más largas, como para el Mycoplasma hominis "ATCC 23114" con longitud de 665,445 pares de bases, no mejora tanto los resultados, pudiendo ser peor para algunas medidas. Los buenos resultados para reconstruir secuencias cortas se deben a que la relación entre la longitud de los fragmentos generados comparados y la longitud de la secuencia que se desea reconstruir es mucho mayor que si se desea reconstruir una secuencia mucho más larga. Para secuencias largas, los fragmentos generados por LPS no son lo suficientemente largos para mejorar sustancialmente la calidad del ensamble, incluso con los errores propios

que introduce la estrategia de LPS puede hacer que decaiga la calidad de LPS con PadeNA en comparación a los resultados obtenidos usando PadeNA solamente.

En la Tabla 11 mostramos los resultados que obtuvieron la mejor calidad de ensamble para cierto valor *n* de solape entre secuencias. El valor mostrado fue encontrado mediante la exploración de un conjunto de posibles valores y fue determinado comparando los valores de calidad entre los ensambles. Según lo mostrado en los experimentos anteriores. El valor de empalme que obtiene mejores valores de calidad de los ensambles es el que llega a un compromiso de ser un valor bajo ($n = 10$ para los primeros 5 experimentos, o $n = 15$ para el último) que genere un nuevo fragmento suficientemente largo con el cual podamos desambiguar un *repeat*, pero no tan corto tal que no se generen fragmentos erróneos producto de usar pocos caracteres para el solape. Usar muchos caracteres para solape hace que se generen fragmentos muy cortos, acercándose la calidad de los ensambles a la obtenida por PadeNA. Sin embargo, si usamos muchos caracteres para el solape en la primera corrida de LPS y menos para la segunda, es posible obtener secuencias largas con menos errores, dando ensambles de buena calidad. Propondremos como trabajo futuro la obtención del valor óptimo de este parámetro.

6 CONCLUSIONES Y TRABAJO FUTURO

6.1 Conclusiones

LPS es un algoritmo de ensamble de secuencias cortas de ADN el cual permite la creación de secuencias de longitudes más largas utilizando técnicas de *agrupamiento* o *clustering*. Al no utilizarse una matriz para almacenar la cantidad de caracteres que se empalman entre pares de secuencias, esta técnica permite reducir el cálculo al solo necesitar comparar con el elemento representativo o pivote de un *cluster*.

LPS permite tratar las apariciones de los *repeats* creando secuencias más largas reduciendo la posibilidad de apariciones de saltos o *gaps*. También permite la reducción del número de *contigs* y la suma de los *contigs* lo cual es deseado en el resultado de un ensamble.

En los experimentos realizados mostramos que con dos corridas de LPS y el uso de PadeNA posteriormente es posible obtener ensambles con una calidad mejor que la obtenida por PadeNA solo. Para ello fue necesario explorar la configuración del parámetro *n* para la cantidad de caracteres requeridos para el solape.

Una buena configuración de este valor de umbral es utilizar una cantidad pequeña de caracteres para las dos corridas de LPS, por ejemplo utilizar 10 caracteres en el primer experimento y 20% en el segundo, con el propósito de generar secuencias más largas producto de las mezclas de subsecuencias. Si exigimos un umbral más alto para las dos corridas de LPS, el ensamble tiende a ser parecido al obtenido por LPS debido a que mientras más caracteres estén en el solape, más corta será la nueva secuencia generada producto de la mezcla. Sin embargo, si pedimos muy pocos caracteres como umbral, por ejemplo usando 5 caracteres o el 10%, se corre el riesgo de generar nuevas secuencias erróneas obteniéndose un ensamble con una calidad peor que PadeNA.

También es posible obtener un buen ensamble si en la primera corrida de LPS usamos muchos caracteres como umbral para el solape y luego en la segunda corrida, usamos pocos caracteres en el solape. Esto permite en la primera corrida obtener menos fragmentos

erróneos generados en la mezcla con una longitud más larga y luego en la segunda corrida aumentar la longitud de los fragmentos al utilizarse menos caracteres para el solape.

LPS es un método de ensamble para secuencias cortas sin errores y no toma en consideración la dirección de la secuencia por lo que para mejorar la calidad de ensamble requiere de la ayuda de un método de ensamble que acepte secuencias simples (no pareadas) al cual mejora la calidad del ensamble al tratar con el problema de los *repeats*.

6.2 Trabajo Futuro

A partir de los experimentos mostrados observamos que el error cometido por LPS es arrastrado y contribuye a aumentar el error cometido posteriormente por PadeNA. A pesar de que PadeNA cuenta con un mecanismo de reducción de errores, sería conveniente en una extensión de LPS agregar una fase de post-procesamiento para eliminar posibles fragmentos mal ensamblados.

LPS utiliza posteriormente a PadeNA para poder procesar las secuencias en ambas direcciones. Proponemos como trabajo futuro agregar información de la secuencia de ADN complementaria para tratar desde LPS las dos posibles direcciones de las secuencias de ADN.

LPS fue utilizado con *reads* obtenidos libres de errores. Los errores son corregidos posteriormente por PadeNA. Se propone utilizar alguna heurística para descartar los *reads* que se consideren que tengan algún error del conjunto de secuencias a ensamblar.

El parámetro establecido como umbral para el solape entre pares de secuencias juega un papel fundamental para la calidad del ensamble obtenido finalmente. En los experimentos se mostró que tener un valor pequeño de este parámetro hace que se obtengan ensambles con una buena calidad. Si este valor de umbral seleccionado es muy bajo obtendremos un ensamble con muchos errores y si buscamos un valor alto el ensamble obtenido es parecido al obtenido por PadeNA. Se propone realizar un método de estimación de la calidad que tendrá un ensamble para un valor de solape *n* con el objetivo de determinar un valor aceptable para obtener un buen ensamble.

Glosario de Términos

ADN: Ácido desoxirribonucleico, abreviado como ADN, es un ácido nucleico que contiene instrucciones genéticas usadas en el desarrollo y funcionamiento de todos los organismos vivos conocidos y algunos virus, y es responsable de su transmisión hereditaria.

bioinformática: Disciplina que utiliza las matemáticas, estadísticas y algoritmos computacionales para el análisis secuencias de ADN y amino ácidos.

contig: Secuencia continua de ADN que ha sido ensamblada a partir de fragmentos de ADN empalmados.

de novo: Técnica de ensamble de secuencias de ADN en la cual las pequeñas secuencias son usadas para crear una secuencia grande sin información de referencia extra.

ensamblador: Algoritmo usado para ensamblar fragmentos de secuencias cortas mediante la mezcla de éstas con el objetivo de reconstruir la secuencia original.

genoma: Conjunto de genes contenidos en los cromosomas, lo que puede interpretarse como la totalidad de la información genética que posee un organismo o una especie en particular.

in silico: Experimento realizado por una computadora o simulado por una computadora. Se refiere a experimentos a realizarse sobre organismos vivos fuera de organismos vivos.

***k*-mer**: Subcadena, tupla o *k*-grama de longitud *k* en una secuencia de ADN.

número de accession: es un identificador único dado a un ADN o proteína con el cual se puede realizar un seguimiento a lo largo de diferentes versiones en un repositorio.

N50: Es la longitud del *contig* más pequeño del conjunto de *contigs* más largo cuya suma de longitudes es al menos el 50% de la suma de las longitudes de todos los *contigs*.

plásmido: Son moléculas de ADN que son capaces de replicarse independientemente del ADN cromosomal (ADN de los cromosomas).

salto o **gap**: Subregión en la cual no se conoce el valor de la secuencia.

secuencia: Sucesión de letras representando la estructura una molécula de ADN. Las posibles letras son A, C, G, y T, que simbolizan las cuatro subunidades de nucleótidos de una banda ADN - adenina, citosina, guanina, timina.

scaffold: Secuencia no redundante formada por la unión de uno o más *contigs*

REFERENCIAS

Baker, M. (2012). De novo genome assembly: what every biologist should know. *Nat Methods, 9*(4), 333-337. doi: citeulike-article-id:10509881

doi: 10.1038/nmeth.1935

Bankevich, A., Nurk, S., Antipov, D., Gurevich, A. A., Dvorkin, M., Kulikov, A. S., . . . Pevzner, P. A. (2012). SPAdes: A New Genome Assembly Algorithm and Its Applications to Single-Cell Sequencing. *Journal of Computational Biology, 19*(5), 455-477.

Berkhin, P. (2006). A Survey of Clustering Data Mining Techniques. *Grouping Multidimensional Data,* 25--71.

Bradnam, K., Fass, J., Alexandrov, A., Baranay, P., Bechner, M., Birol, I., . . . Ho, I. (2013). Assemblathon 2: evaluating de novo methods of genome assembly in three vertebrate species. *GigaScience, 2*(1), 10.

Bryant, D., Wong, W.-K., & Mockler, T. (2009). QSRA - a quality-value guided de novo short read assembler. *BMC Bioinformatics, 10*(1), 69.

Butler, J., MacCallum, I., Kleber, M., Shlyakhter, I. A., Belmonte, M. K., Lander, E. S., . . . Jaffe, D. B. (2008). ALLPATHS: De novo assembly of whole-genome shotgun microreads. *Genome Research, 18*(5), 810-820. doi: 10.1101/gr.7337908

Compeau, P. E., Pevzner, P. A., & Tesler, G. (2011). How to apply de Bruijn graphs to genome assembly. *Nat Biotechnol, 29*(11), 987-991. doi: 10.1038/nbt.2023

Chaisson, M., Pevzner, P., & Tang, H. (2004). Fragment assembly with short reads. *Bioinformatics, 20*(13), 2067-2074. doi: 10.1093/bioinformatics/bth205

Chaisson, M., & Tesler, G. (2012). Mapping single molecule sequencing reads using basic local alignment with successive refinement (BLASR): application and theory. *BMC Bioinformatics, 13*(1), 238.

Chaisson, M. J., Brinza, D., & Pevzner, P. A. (2009). De novo fragment assembly with short mate-paired reads: Does the read length matter? *Genome Res, 19*(2), 336-346. doi: 10.1101/gr.079053.108

Chaisson, M. J., & Pevzner, P. A. (2008). Short read fragment assembly of bacterial genomes. *Genome Res, 18*(2), 324-330. doi: 10.1101/gr.7088808

Chin, C. S., Alexander, D. H., Marks, P., Klammer, A. A., Drake, J., Heiner, C., . . . Korlach, J. (2013). Nonhybrid, finished microbial genome assemblies from long-read SMRT sequencing data. *Nat Methods, 10*(6), 563-569. doi: 10.1038/nmeth.2474

De Bruijn, N. G. (1946). A combinatorial problem. *Koninklijke Nederlandse Akademie v. Wetenschappen, 49*, 758-764.

Defays, D. (1977). An efficient algorithm for a complete link method. *The Computer Journal, 20*(4), 364-366. doi: 10.1093/comjnl/20.4.364

Deonier, R. C., Tavara, S., & Waterman, M. S. (2005). *Computational Genome Analysis: An Introduction*: Springer.

Deshpande, V., Fung, E. K., Pham, S., & Bafna, V. (2013). Cerulean: A Hybrid Assembly Using High Throughput Short and Long Reads. In A. Darling & J. Stoye (Eds.), *Algorithms in Bioinformatics* (Vol. 8126, pp. 349-363): Springer Berlin Heidelberg.

Dohm, J. C., Lottaz, C., Borodina, T., & Himmelbauer, H. (2007). SHARCGS, a fast and highly accurate short-read assembly algorithm for de novo genomic sequencing. *Genome Res, 17*(11), 1697-1706.

Ester, M., Kriegel, H.-P., Sander, J., & Xu, X. (1996). *A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise*. Paper presented at the Proc. of 2nd International Conference on Knowledge Discovery and.

Euler, L. (1736). Solutio problematis ad geometriam situs pertinentis. *Commentarii Academiae Scientiarum Imperialis Petropolitanae, 8*, 128--140.

Garey, M. R., & Johnson, D. S. (1990). *Computers and Intractability; A Guide to the Theory of NP-Completeness*: W. H. Freeman Co.

Grimaldi, R. P. (1998). *Matemáticas discreta y combinatoria: introducción y aplicaciones*: Pearson Educación.

Guha, S., Rastogi, R., & Shim, K. (1998). *CURE: an efficient clustering algorithm for large databases*. Paper presented at the Proceedings of the 1998 ACM SIGMOD international conference on Management of data, Seattle, Washington, USA.

Hartigan, J. (1975). *Clustering Algorithms*: Books on Demand.

Hernandez, D., François, P., Farinelli, L., Østerås, M., & Schrenzel, J. (2008). De novo bacterial genome sequencing: Millions of very short reads assembled on a desktop computer. *Genome Research*. doi: 10.1101/gr.072033.107

Ivanova, N., Sorokin, A., Anderson, I., Galleron, N., Candelon, B., Kapatral, V., . . . Kyrpides, N. (2003). Genome sequence of Bacillus cereus and comparative analysis with Bacillus anthracis. *Nature, 423*(6935), 87-91. doi: 10.1038/nature01582

Jain, A. K., Murty, M. N., & Flynn, P. J. (1999). Data clustering: a review. *ACM Comput. Surv., 31*(3), 264-323. doi: 10.1145/331499.331504

Jeck, W. R., Reinhardt, J. A., Baltrus, D. A., Hickenbotham, M. T., Magrini, V., Mardis, E. R., . . . Jones, C. D. (2007). Extending assembly of short DNA sequences to handle error. *Bioinformatics, 23*(21), 2942-2944. doi: 10.1093/bioinformatics/btm451

Kaufman, L., & Rousseeuw, P. J. (1990). *Finding groups in data: an introduction to cluster analysis*. New York: John Wiley and Sons.

Kendrew, J., & Lawrence, E. (1995). *Encyclopedia of Molecular Biology*: Wiley.

Kohonen, T. (1985). Median strings. *Pattern Recognition Letters, 3*(5), 309-313. doi: http://dx.doi.org/10.1016/0167-8655(85)90061-3

Kulinska, A., Czeredys, M., Hayes, F., & Jagura-Burdzy, G. (2008). Genomic and functional characterization of the modular broad-host-range RA3 plasmid, the archetype of the IncU group. *Appl Environ Microbiol, 74*(13), 4119-4132. doi: 10.1128/aem.00229-08

Lander, E. S., Linton, L. M., Birren, B., Nusbaum, C., Zody, M. C., Baldwin, J., . . . Chen, Y. J. (2001). Initial sequencing and analysis of the human genome. *Nature, 409*(6822), 860-921. doi: 10.1038/35057062

Levenshtein, V. I. (1966). Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady, 10*, 707. doi: citeulike-article-id:1667643

Liu, L., Li, Y., Li, S., Hu, N., He, Y., Pong, R., . . . Law, M. (2012). Comparison of Next-Generation Sequencing Systems. *Journal of Biomedicine and Biotechnology, 2012*, 11. doi: 10.1155/2012/251364

MacQueen, J. (1967, 1967). *Some methods for classification and analysis of multivariate observations.* Paper presented at the Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics, Berkeley, Calif.

Margulies, M., Egholm, M., Altman, W. E., Attiya, S., Bader, J. S., Bemben, L. A., . . . Rothberg, J. M. (2005). Genome sequencing in microfabricated high-density picolitre reactors. *Nature, 437*(7057), 376-380. doi: http://www.nature.com/nature/journal/v437/n7057/suppinfo/nature03959_S1.html

Medvedev, P., Georgiou, K., Myers, G., & Brudno, M. (2007). Computability of Models for Sequence Assembly. In R. Giancarlo & S. Hannenhalli (Eds.), *Algorithms in Bioinformatics* (Vol. 4645, pp. 289-301): Springer Berlin Heidelberg.

Medvedev, P., Pham, S., Chaisson, M., Tesler, G., & Pevzner, P. (2011). Paired de bruijn graphs: a novel approach for incorporating mate pair information into genome assemblers. *J Comput Biol, 18*(11), 1625-1634. doi: 10.1089/cmb.2011.0151

Miller, J. R., Delcher, A. L., Koren, S., Venter, E., Walenz, B. P., Brownley, A., . . . Sutton, G. (2008). Aggressive assembly of pyrosequencing reads with mates. *Bioinformatics, 24*(24), 2818-2824. doi: 10.1093/bioinformatics/btn548

Miller, J. R., Koren, S., & Sutton, G. (2010). Assembly algorithms for next-generation sequencing data. *Genomics, 95*(6), 315-327. doi: http://dx.doi.org/10.1016/j.ygeno.2010.03.001

Myers, E., Sutton, G., Delcher, A., Dew, I., Fasulo, D., Flanigan, M., . . . Venter, C. (2000). A Whole-Genome Assembly of Drosophila. *Science, 287*(5461), 2196-2204. doi: citeulike-article-id:1145131

doi: 10.1126/science.287.5461.2196

Myers, E. W. (1995). Toward simplifying and accurately formulating fragment assembly. *J Comput Biol, 2*(2), 275-290.

Narzisi, G., & Mishra, B. (2011). Comparing De Novo Genome Assembly: The Long and Short of It. *Plos One, 6*(4), e19175. doi: 10.1371/journal.pone.0019175

Nurk, S., Bankevich, A., Antipov, D., Gurevich, A., Korobeynikov, A., Lapidus, A., . . . Pevzner, P. (2013). Assembling Genomes and Mini-metagenomes from Highly Chimeric Reads. In M. Deng, R. Jiang, F. Sun & X. Zhang (Eds.), *Research in Computational Molecular Biology* (Vol. 7821, pp. 158-170): Springer Berlin Heidelberg.

Peltola, H., Söderlund, H., & Ukkonen, E. (1984). SEQAID: a DNA sequence assembling program based on a mathematical model. *Nucleic Acids Research, 12*(1Part1), 307-321. doi: 10.1093/nar/12.1Part1.307

Pereyre, S., Sirand-Pugnet, P., Beven, L., Charron, A., Renaudin, H., Barre, A., . . . Bebear, C. (2009). Life on arginine for Mycoplasma hominis: clues from its minimal genome and comparison with other human urogenital mycoplasmas. *PLoS Genet, 5*(10), e1000677. doi: 10.1371/journal.pgen.1000677

Perkel, J. M. (2009). Sanger Who? Sequencing the Next Generation. *Science, 324* (5924), 275-279. doi: 10.1126/science.324.5924.275

Pevzner, P. A., & Tang, H. (2001). Fragment assembly with double-barreled data. *Bioinformatics, 17*(suppl 1), S225-S233. doi: 10.1093/bioinformatics/17.suppl_1.S225

Pevzner, P. A., Tang, H., & Tesler, G. (2004). De Novo Repeat Classification and Fragment Assembly. *Genome Research, 14*(9), 1786-1796. doi: 10.1101/gr.2395204

Pevzner, P. A., Tang, H., & Waterman, M. S. (2001). An Eulerian path approach to DNA fragment assembly. *Proceedings of the National Academy of Sciences, 98*(17), 9748-9753. doi: 10.1073/pnas.171285098

Pham, S., Antipov, D., Sirotkin, A., Tesler, G., Pevzner, P., & Alekseyev, M. (2012). Pathset Graphs: A Novel Approach for Comprehensive Utilization of Paired Reads in

Genome Assembly. In B. Chor (Ed.), *Research in Computational Molecular Biology* (Vol. 7262, pp. 200-212): Springer Berlin Heidelberg.

Ravel, J., Jiang, L., Stanley, S. T., Wilson, M. R., Decker, R. S., Read, T. D., . . . Rasko, D. A. (2009). The complete genome sequence of Bacillus anthracis Ames "Ancestor". *J Bacteriol, 191*(1), 445-446. doi: 10.1128/jb.01347-08

Rosen, K. (2011). *Discrete Mathematics and Its Applications*: McGraw-Hill Education.

Simpson, J. T., Wong, K., Jackman, S. D., Schein, J. E., Jones, S. J. M., & Birol, İ. (2009). ABySS: A parallel assembler for short read sequence data. *Genome Research, 19*(6), 1117-1123. doi: 10.1101/gr.089532.108

Skiena, S. S. (1998). *The Algorithm Design Manual: Text*: TELOS, The Electronic Library of Science.

Sneath, P. H. (1957). The application of computers to taxonomy. *J Gen Microbiol, 17*(1), 201-226.

Srinivasulu M, K. R. R., Baji Mohammed, R V S P Kumar. (2012). Data Clustering With Leaders and Subleaders Algorithm *IOSR Journal of Engineering (IOSRJEN), 2*(11), 01-07.

Sutton, G., White, O., Adams, M., & Kerlavage, A. (1995). TIGR Assembler: A New Tool for Assembling Large Shotgun Sequencing Projects. *Genome Science and Technology, 1*(1), 9-19. doi: citeulike-article-id:9757227

doi: 10.1089/gst.1995.1.9

Swingley, W. D., Chen, M., Cheung, P. C., Conrad, A. L., Dejesa, L. C., Hao, J., . . . Touchman, J. W. (2008). Niche adaptation and genome expansion in the chlorophyll d-producing cyanobacterium Acaryochloris marina. *Proc Natl Acad Sci U S A, 105*(6), 2005-2010. doi: 10.1073/pnas.0709772105

Thareja, G., Kumar, V., Zyskowski, M., Mercer, S., & Davidson, B. (2011). *PadeNA: A Parallel De Novo Assembler*. Paper presented at the Bioinformatics. http://dblp.uni-trier.de/db/conf/biostec/bioinformatics2011.html#TharejaKZMD11

Vijaya, P. A., Narasimha Murty, M., & Subramanian, D. K. (2004). Leaders–Subleaders: An efficient hierarchical clustering algorithm for large data sets. *Pattern Recognition Letters, 25*(4), 505-513. doi: http://dx.doi.org/10.1016/j.patrec.2003.12.013

Vijaya, P. A., Narasimha Murty, M., & Subramanian, D. K. (2006). Efficient bottom-up hybrid hierarchical clustering techniques for protein sequence classification. *Pattern Recognition, 39*(12), 2344-2355. doi: http://dx.doi.org/10.1016/j.patcog.2005.12.001

Voelkerding, K. V., Dames, S. A., & Durtschi, J. D. (2009). Next-Generation Sequencing: From Basic Research to Diagnostics. *Clinical Chemistry, 55*(4), 641-658. doi: 10.1373/clinchem.2008.112789

Wang, L., & Jiang, T. (1994). On the complexity of multiple sequence alignment. *J Comput Biol, 1*(4), 337-348.

Warren, R. L., Sutton, G. G., Jones, S. J. M., & Holt, R. A. (2007). Assembling millions of short DNA sequences using SSAKE. *Bioinformatics, 23*(4), 500-501. doi: 10.1093/bioinformatics/btl629

Zerbino, D. R., & Birney, E. (2008). Velvet: Algorithms for de novo short read assembly using de Bruijn graphs. *Genome Research, 18*(5), 821-829. doi: 10.1101/gr.074492.107

Zhang, T., Ramakrishnan, R., & Livny, M. (1996). BIRCH: an efficient data clustering method for very large databases. *SIGMOD Rec., 25*(2), 103-114. doi: 10.1145/235968.233324

Zimin, A., Marçais, G., Puiu, D., Roberts, M., Salzberg, S., & Yorke, J. (2013). The MaSuRCA genome assembler. *Bioinformatics, 29*(21), 2669-2677. doi: doi: 10.1093/bioinformatics/btt476