ICT-UNPA-221-2019
ISSN: 1852-4516
Aprobado por Resolución N° 1142/19-R-UNPA
http://doi.org/10.22305/ict-unpa.v11.n3.804

Análisis del Algoritmo Genético Celular para el Problema de Ensamblado de Cadenas de ADN

Analysis of Cellular Genetic Algorithm for the DNA Fragment Assembly Problem

Sabrina Pastrana[1], Ana Carolina Olivera[2,3], Pablo Javier Vidal[1,2,3]
sabrina.pastrana@gmail.com, acolivera@conicet.gov.ar, pjvidal@conicet.gov.ar
[1]Universidad Nacional de la Patagonia Austral. Unidad Académica Caleta Olivia. Ruta N 3 Acceso Norte - (9011) Caleta Olivia - Santa Cruz - Argentina
[2]Universidad Nacional de Cuyo. Facultad de Ingeniería. Mendoza. Argentina
[3]CONICET. Universidad Nacional de Cuyo. Instituto Universitario para las Tecnologías de la Información y las Comunicaciones. (M5502JMA). Mendoza. Argentina

Recibido: 09/05/2019. Aceptado: 25/09/2019

RESUMEN

En este trabajo, presentamos un análisis del comportamiento de un Algoritmo Genético Celular (cGA) aplicado al Problema de Ensamblado de cadenas de Ácido Desoxirribonucleico (ADN- FAP). Se construyeron 12 configuraciones para algoritmo basadas en dos operadores de mutación y dos tamaños de población. Luego de analizar los resultados obtenidos por las distintas configuraciones sobre instancias de la literatura se muestra claramente que los valores obtenidos considerando la utilización de operadores no adaptados al problema son promisorios a nivel de cubrimiento y alineamiento de cadenas de ADN.

Palabras clave: Algoritmo Genético Celular, Problema de Ensamblado de cadenas de Ácido Desoxirribonucleico.

ABSTRACT

In this work, an analysis of the behavior of Cellular Genetic Algorithm applied to the Deoxyribonucleic Acid Fragment Assembly Problem (DNA-FAP) is presented. Twelve configurations of the algorithm were constructed based on two mutation operators and two sizes of population. After to analyse the results obtained for the different configurations over literature's instances it is showed clearly that the values obtained -consider the use of non-problem ready operators- are promising with respect to coverage level and alignment of DNA fragments.

Keywords: Cellular Genetic Algorithm; DNA Fragment Assembly Problem; Algorithms parametrization.

1. INTRODUCCIÓN

Si bien las herramientas informáticas de análisis de datos genéticos han madurado mucho desde su aparición, aun en la actualidad sigue siendo un gran reto computacional la interpretación del

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-SinDerivar 4.0 Internacional.

enorme volumen de datos brutos que se origina al secuenciar el genoma de cualquier organismo vivo. El genoma contiene la nómina completa de genes, y toda la información biológica requerida para construir y mantener cada una de las instancias de dicho organismo.
La mayor parte de los genomas presentes en la naturaleza están constituidos por ácido desoxirribonucleico (DNA, por sus siglas en inglés) aunque ciertos virus poseen como material genético, ácido ribonucleico (RNA, por sus siglas en inglés). En ambos casos, se trata de moléculas poliméricas construidas por cadenas de sub-unidades denominadas nucleótidos. Para el caso del DNA, estos nucleótidos se denominan desoxirribonucleótidos (de ahí la D), y para el caso del RNA, ribonucleótidos (de ahí la R).
El DNA está compuesto por una mezcla de cuatro nucleótidos: Adenina, que se representa con una A, Guanina (G), la Citosina (C) y Timina (T). Una molécula de DNA está formada por dos cadenas de estos nucleótidos polimerizados, lo cual se denomina bases, formando una estructura que se describe a menudo como una doble hélice. Las dos cadenas del DNA están estabilizadas entre si por puentes de hidrogeno, que ocurren entre las bases de las dos cadenas. Se dice que estas bases están apareadas unas con otras. Este apareamiento tiene lugar de una forma muy precisa: la A de una cadena se aparea con la T de la otra cadena y la C con la G. La información biológica presente en el DNA se encuentra codificada en el orden preciso de esos nucleótidos dentro de la molécula de DNA, lo que denominamos secuencia de nucleótidos. El objetivo primario es precisamente determinar la secuencia de nucleótidos específica de cada genoma.
En bioinformática, el ensamblaje de secuencias de DNA se refiere al rearmado de múltiples fragmentos en pos de conseguir la secuencia de DNA original, es decir, se alinean y se mezclan múltiples fragmentos de una secuencia de DNA mucho mayor para reconstruir la secuencia original. Establecer las características de un organismo completo en el laboratorio constituye una tarea muy costosa en referencia al tiempo de cómputos que se insume. Las primeras generaciones de montadores de secuencias de DNA empezaron a aparecer en los 80 y primeros años de la década del 90 del siglo XX, los mismos reconstruían las grandes cantidades de fragmentos que eran generados por instrumentos de secuenciación automática. Estos ensambladores de primera generación utilizaban varias estrategias para manejar las secuencias repetitivas y los errores de secuenciación que podían confundir el ensamblado. Sin embargo, no podían manejar genomas mucho más largos que los de una bacteria, y por este motivo fueron siendo reemplazados poco a poco, a medida que el campo de investigación se movía hacia genomas mayores. En la actualidad el uso de las herramientas informáticas ha generado una explosión de posibilidades para el análisis de grandes cantidades de información. Pero aun el problema de evaluar grandes secuencias de información de DNA de seres vivos sigue siendo una limitación, ya que los tiempos computacionales para dicho análisis siguen siendo muy elevados.
Entre los problemas relacionados con el genoma podemos encontrar el estudio del ensamblado de cadenas de DNA (Deoxyribonucleic Acid Fragment Assembly Problem, DNA-FAP), donde dado un conjunto de cientos o miles de fragmentos de DNA, que pueden contener errores, debemos encontrar la secuencia de DNA original a partir de las permutaciones de los fragmentos que mejor representen a dicha secuencia.
Existen herramientas que automatizan el secuenciamiento de DNA tales como PHRAP (P. Green, 2009), TIGR assembler (G. G. Sutton, O. White, M. D. Adams, y A. R. Kerlavage, 1995) y EULER (P. Pevzner, 2000) entre muchas otras. Todas estas herramientas están enfocadas a diferentes problemas encontrados durante el ensamblado de fragmentos.

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-SinDerivar 4.0 Internacional.

Varios autores abordan el Problema de Ensamblado de Fragmentos de DNA (DNA-FAP, por sus siglas en inglés) con metaheurísticas, en particular, algoritmos inspirados en la naturaleza (P. Vidal y A. C. Olivera, 2014). Las metaheurísticas son procedimientos robustos que encuentran buenos resultados sin tener un conocimiento previo y específico del espacio de búsqueda. El éxito de las metaheurísticas en esta clase de problemas se basa fundamentalmente en que no son exhaustivas ni deterministas. Esto reduce considerablemente el esfuerzo computacional empleado; además, permiten producir múltiples resultados para una misma situación. Por otra parte, esta clase de algoritmos no necesitan de datos exactos y completos para obtener más y mejores soluciones.
En este trabajo, presentamos el análisis del Algoritmo Genético Celular aplicado al DNA-FAP. Los resultados obtenidos demuestran que diferentes configuraciones del algoritmo obtienen valores competitivos a nivel de cobertura y alineamiento de cadenas de ADN.

1.1. Estado del Arte

Varios autores han trabajado ya en el problema del DNA-FAP, utilizando diversas estrategias para resolverlo. Las mismas van desde la utilización de heurísticas, metaheurísticas y algoritmos bio-inspirados.
Minetti, (2011), aborda el problema de ensamblado de fragmentos de DNA y lo resuelve utilizando metaheurísticas y paralelismo. Para esto, compara el comportamiento de distintos algoritmos, incluyendo los resultados de aplicar metaheurísticas basadas en trayectoria y metaheurísticas basadas en población. Luego realiza un estudio combinando estas dos estrategias, obteniendo una manera híbrida de resolver el problema de ensamblado de genomas.
En Minetti G. y Alba E., (2012), los autores proponen otra manera de resolver el problema del DNA-FAP, esta vez el modelo consiste en combinar dos metaheurísticas: un método de trayectoria (como el Recocido Simulado) y un método basado en población (como el Algoritmo Genético), analizando las ventajas de esta hibridización sobre otros ensambladores que existen en la literatura. Concluyen finalmente, que SAX mejora la calidad de los resultados encontrados por otros ensambladores metaheurísticos y no metaheurísticos para resolver la totalidad de las instancias más grandes de este problema.
Díaz B. D., (2006), utiliza Algoritmos Genéticos Celulares para el problema de ensamblado de cadenas de DNA, y plantea la utilización de un esquema canónico. Centra el esfuerzo en el diseño de innovadores algoritmos que mejoran el equilibrio entre la exploración y la explotación que aplican sobre el espacio de búsqueda. Demuestra además que, para un algoritmo evolutivo, si se estructura la población, se obtiene mayor eficiencia y eficacia en la resolución de los problemas abordados ya que la velocidad de convergencia del algoritmo disminuye considerablemente con respecto a la de otros Algoritmos Genéticos. Proponiendo una nueva técnica adaptativa permite al Algoritmo Genético Celular potenciar la exploración o explotación de las zonas más prometedoras del espacio de búsqueda en función de la velocidad de convergencia de la población.
Minetti G. y Alba E., (2008), abordan el problema de ensamblado de fragmentos de DNA enfocándose en la inicialización de la población y luego en los operadores de recombinación. En este estudio los resultados indican que el uso de una población inicial generada heurísticamente y el uso de un operador especifico de recombinación obtiene resultados competitivos.
Mallen-Fullerton y Fernández-Anaya (2013), presentan otro algoritmo inspirado en la naturaleza basado en la Optimización de Enjambre de Partículas y la Evolución Diferencial. Comparan estos algoritmos utilizando un conjunto de instancias de estudio mostrando sus ventajas.

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-SinDerivar 4.0 Internacional.

Por último, Vidal P. y Olivera C., (2018) proponen un algoritmo de luciérnaga diseñado especialmente para ser ejecutado sobre una arquitectura de unidades de procesamiento gráfico de manera tal de acelerar el proceso computacional buscando resolver el problema de DNA-FAP.
En particular, en este trabajo realizamos una comparación de diferentes operadores y configuraciones en un Algoritmo Genético Celular utilizando para esto dos instancias de estudio: x60189_4 de 39 fragmentos y la instancia bx842596_4 de 442.
Pudimos demostrar que diferentes configuraciones del algoritmo, obtienen valores competitivos a nivel de cobertura y alineamiento de ADN con respecto al estado actual de la literatura.

2. PROBLEMA DE ENSAMBLADO DE FRAGMENTOS DE ADN

Estructuralmente, el ADN está compuesto por larguísimas sucesiones de moléculas: Adenine (A), Thynine(T), Guanine (G) y Cytosine (C) llamadas comúnmente bases. Dado lo inmenso de estas cadenas estudiarlas de forma global es casi imposible. Por lo cual, las cadenas son copiadas y fragmentadas para poder ser analizadas. A esto se lo conoce comúnmente como shotgun. Shotgun no mantiene el orden en que fueron fragmentadas ni ninguna otra información de la cadena original. Una vez que estos fragmentos fueron estudiados es necesario volver a la cadena original, rearmando de manera inteligente todas las subcadenas. Debido al proceso de copia y corte de las cadenas de ADN los fragmentos pueden contener errores o faltar piezas. Al proceso de rearmado se lo conoce como ensamblado de fragmentos de ADN (Vidal P. and Olivera C., 2018). Pequeñas secuencias deben ser nuevamente ensambladas en orden solapando porciones de las mismas. Estos pequeños fragmentos van uniéndose formando fragmentos más grandes, denominados *contigs*.

La mayor a de los algoritmos de ensamblado realizan los siguientes pasos:

- *Solapamiento (Overlap).* Consiste en encontrar el mejor solapado entre los sufijos y los prefijos de todos los fragmentos. La práctica común consiste en filtrar pares de fragmentos que no compartan subcadenas significativas.
- *Alineamiento (Layout).* Es encontrar el orden en que estaban los fragmentos en la secuencia original. Constituye la parte más costosa del proceso de ensamblado dada la dificultad de decidir si dos fragmentos están solapados (sus diferencias están causadas por errores de copia) o en realidad son dos copias distintas de una repetición. Las subcadenas repetidas son el mayor desafío para ensamblar cadenas de genoma.
- *Consenso (Consensus).* Se refiere a derivar la secuencia de ADN a partir de la disposición establecida en el paso anterior.

Para medir la calidad del consenso se observa el llamado cubrimiento (*Coverage*). El cubrimiento de una posición base está definido como el número de fragmentos que comparten esa posición. Esta es una medida de redundancia de un fragmento de dato y denota el número de fragmentos, en promedio, donde un nucleótido se espera que aparezca en el ADN. Esto se calcula como el número de bases leídas por fragmento sobre el tamaño del ADN obtenido. En la Fig. 1 se pueden observar los pasos usuales de las técnicas de ensamblado.

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-SinDerivar 4.0 Internacional.

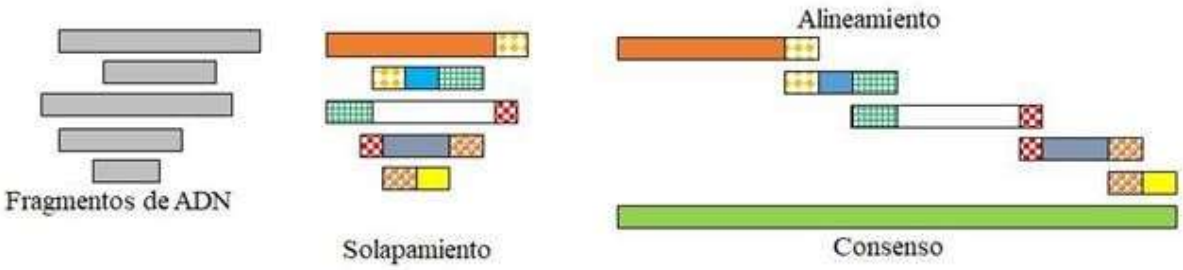

Figura 1: Pasos en el ensamblado de fragmentos de ADN

Obtener un alineamiento es la parte más costosa del proceso de ensamblado. La dificultad se encuentra en establecer si dos fragmentos están realmente solapados, son partes totalmente distintas o bien son dos fragmentos del mismo corte solo que, uno de ellos posee errores. En orden de evaluar una solución para el DNA-FAP se define la siguiente función:

$$f(i) = \sum_{a=0}^{l-1} w_{a,a+1} \quad (1)$$

3. ALGORITMO GENÉTICO CELULAR

Un Algoritmo Genético Celular (cGA) es una variante del Algoritmo Genético, basado en una clase de población descentralizada en la que las soluciones tentativas evolucionan en vecindades superpuestas. En un cGA, los individuos son situados en una malla toroidal bidimensional (normalmente bidimensional, aunque el número de dimensiones puede ser extendido fácilmente a tres o más), y se les permite recombinarse con individuos cercanos (Díaz B.D., 2006). Estas vecindades superpuestas ayudan en la exploración del espacio de búsqueda debido a que, por medio de una lenta difusión de las soluciones a través de la población, proporcionan exploración, mientras que la explotación tiene lugar dentro de cada vecindario mediante los operadores genéticos utilizados. El vecindario más utilizado se denomina L5 (también llamado vecindario de NEWS) integrado por los individuos Norte, Este, Oeste y Sur que se muestran en la Figura 2.

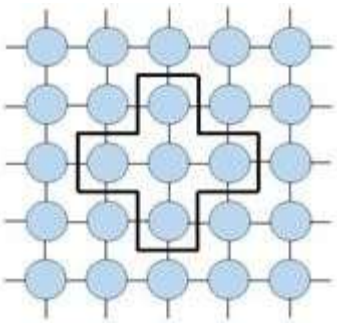

Figura 2: Disposición de la población en un cGA

En este vecindario, los individuos sólo pueden interactuar con sus vecinos en el bucle reproductor que aplican los operadores de variación. Este bucle reproductor es aplicado dentro del vecindario de cada individuo y consiste generalmente en seleccionar dos individuos del

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-SinDerivar 4.0 Internacional.

vecindario como padres de acuerdo a un cierto criterio, aplicarles los operadores de variación (recombinación y mutación), y reemplazar el individuo considerado por el descendiente recientemente creado siguiendo un determinado criterio, por ejemplo, si el descendiente representa una mejor solución que el individuo considerado. A continuación, se muestra el proceso de la selección del vecindario y su evolución a través de las iteraciones hacia la población de las soluciones.

Para lograr la población de soluciones propiamente dicha, se debe entonces iterar siguiendo los siguientes pasos:

- Selección del vecindario inicial (inicialización): Por lo general, la población inicial se genera de manera aleatoria, aunque pueden utilizarse diversos criterios para este objetivo como, por ejemplo, incluir conocimiento del problema o cualquier otro tipo de información.
- Evaluación de la solución: Luego de inicializada la población, y por cada solución obtenida, se debe evaluar la calidad de la misma, y esto se logra calculando el valor Fitness de las soluciones candidatas.
- Selección de la mejor solución: Por medio de este proceso, se impone la supervivencia de las mejores soluciones, es decir, aquellas con mayor valor de Fitness obtenido.
- Cruce o recombinación de la solución: Este proceso combina partes de soluciones padres, para crear nuevas soluciones, hijos de los anteriores, que se supone que son mejores soluciones. Los mismos no serán idénticos a sus antecesores, pero contendrán parte de información de ellos.
- Mutación: La mutación, realiza la modificación de manera aleatoria de una única solución.
- Reemplazo: Luego de conseguir la población descendiente, mediante todos los anteriores pasos descritos, se debe reemplazar la misma, siguiendo algún criterio determinado.

En la Figura 3 se muestra el proceso de iteración mencionado:

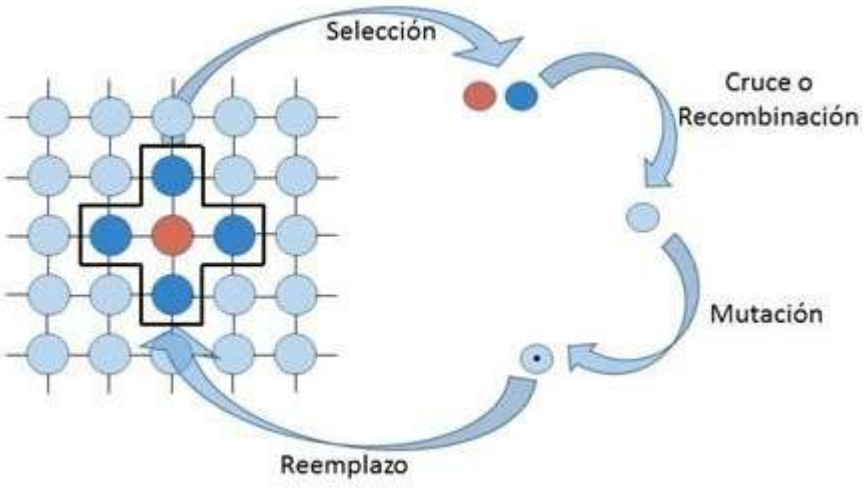

Figura 3: Iteración de un cGA

El siguiente es un pseudocódigo de un cGA canónico. Se puede ver que tras la generación y evaluación (líneas 1 y 2 respectivamente) de la población inicial, los operadores genéticos tales

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-SinDerivar 4.0 Internacional.

como selección de los padres, recombinación, mutación y el reemplazo del individuo padre por el hijo (de líneas 7 a 11), son aplicados a cada uno de los individuos dentro del entorno de sus vecindarios iterativamente hasta alcanzar una condición de finalización (en este caso *max_iter*). Los individuos que formarán parte de la población de la siguiente generación (los nuevos hijos generados o bien los individuos de la población actual, dependiendo del criterio de reemplazo declarado en la línea 11 van almacenándose en una población auxiliar que tras cada generación reemplaza a la población actual. Por tanto, en este modelo todos los individuos son actualizados simultáneamente en la población (síncrono).

Algoritmo 1 cGA($DNA_{Instance}$): sol

```
1:  GenerarPoblacionInicial(P)
2: Evaluar(P )
3:  for s ←1 to max_iter
do 4:  for n ← 1 to
max_n do
5:          for m ← 1 to max_m do
6:              vecinos ← CalcularVecindario(P, posición(n,
m)) 7:          padre ← Seleccionar(vecinos)
8:              hijo ← Recombinar(padres)
9:              hijo ← Mutar(hijo)
10:             Evaluar(hijo)
11:             Remplazar(posición(n, m), P',
hijo) 12:     end for
13:      end for
14:    P ←  P'
15: end for
16: return sol
```

3.1. Representación de la Solución

Las soluciones se han codificado como permutaciones (vectores) de longitud N, etiquetadas de 1 a N. Decimos que cada ordenación posible de los elementos del vector o arreglo, sin repetirlos, es una permutación. Por último, cada elemento de la permutación representa a un fragmento de la cadena de ADN y su índice denota su posición en el vector.

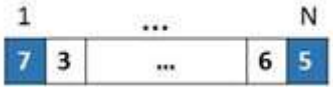

Figura 4: Representación de la solución como una permutación

3.2. Operadores de Cruce

Los operadores de cruce de un AG tienen por finalidad crear una generación nueva de individuos, pidiendo información a sus ancestros (padres). Por lo general dos padres son seleccionados para generar nuevas y mejores soluciones al problema. El operador de

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-SinDerivar 4.0 Internacional.

cruzamiento, tiene como objetivo principal la exploración del espacio de soluciones en busca de posibles mejores soluciones. Su utilidad se basa en el hecho de que las nuevas soluciones pueden ser mejores que sus "progenitoras" si combina las buenas características si se combinan las buenas características de ambas partes.

3.2.1. OX: Order Crossover

El operador de cruce OX construye los hijos eligiendo una subsecuencia de un padre y preservando el orden relativo de los genes del otro padre. Los genes son copiados en el mismo orden, partiendo del segundo punto de corte, omitiendo los conflictivos (que ya aparecieron). En la Figura 5 se muestran los pasos que sigue este operador para la generación de los nuevos individuos.

Paso 1: Se seleccionan los padres. De un padre se elige una subsecuencia de sus genes, y del otro padre, se preserva el orden relativo de los genes que no se repiten en el padre 1.

Paso 2: Los genes del segundo padre (que no se repiten), se copian en el hijo a partir del segundo corte, completando con ellos, los genes que se eliminaron del padre 1.

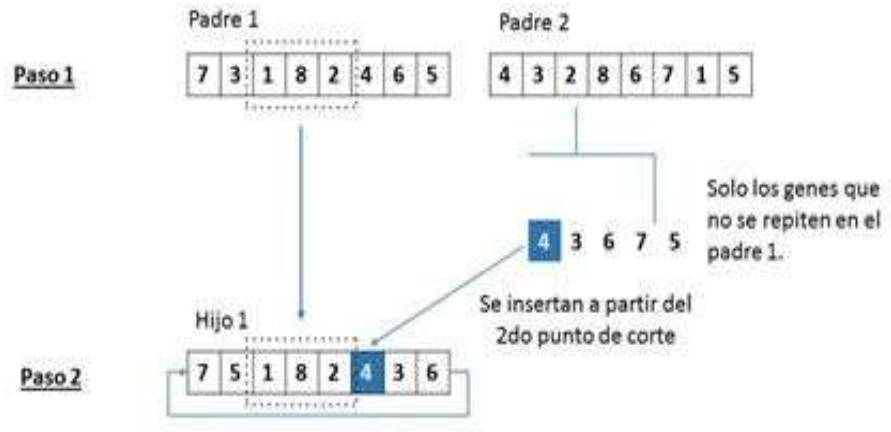

Figura 5: Operador de cruce para representación de orden (Cruce OX)

3.3. Operadores de Mutación

La mutación de un individuo provoca que alguno de sus genes, generalmente uno solo, varíe su valor de forma aleatoria.

Aunque se pueden seleccionar los individuos directamente de la población actual y mutarlos antes de introducirlos en la nueva población, la mutación se suele utilizar de manera conjunta con el operador de cruce. Primeramente, se seleccionan dos individuos de la población para realizar el cruce. Si el cruce tiene éxito entonces uno de los descendientes, o ambos, se muta con cierta probabilidad P_m.

La probabilidad de mutación es muy baja, generalmente menor al 1%. Esto se debe sobre todo a que los individuos suelen tener un ajuste menor después de mutados. Sin embargo, se realizan mutaciones para garantizar que ningún punto del espacio de búsqueda tenga una probabilidad nula de ser examinado.

Tal y como se ha comentado, la mutación más usual es el reemplazo aleatorio. Este consiste en variar aleatoriamente un gen de un cromosoma. Si se trabaja con codificaciones binarias consistirá simplemente en negar un bit. También es posible realizar la mutación intercambiando

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-SinDerivar 4.0 Internacional.

los valores de dos alelos del cromosoma.

3.3.1. Inserción

El operador de Inserción es un operador de mutación donde un determinado elemento $x_{(i)}$ (componente de un vector, para nuestro problema) de la solución x, pasa a ocupar otra posición
j. El resto de los elementos de la solución quedan desplazados, como se puede apreciar en la Figura 6.

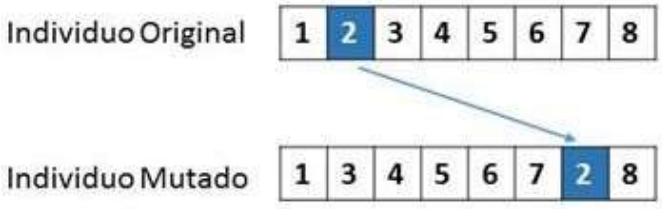

Figura 6: Operador de Mutación Inserción

3.3.1. Swap

El operador de Intercambio (swap), es aquel en el que dos elementos de la solución intercambian su posición (Figura 7).

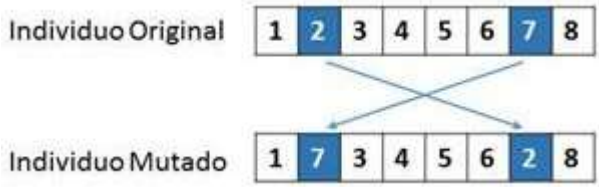

Figura 7: Operador de Mutación Swap o Intercambio

4. EXPERIMENTOS

Para este trabajo se utilizaron dos instancias construidas con GenFrag [M. L. Engle and C. Burks, 1993], que es un aplicativo UNIX/C el cual recibe una secuencia de ADN como entrada y devuelve un conjunto de fragmentos superpuestos como salida. Estos fragmentos generados se pueden utilizar para chequear cualquier aplicación de ensamblado de fragmentos ADN. Su nombre, el número de fragmentos de ADN y el óptimo pueden observarse en la Tabla 1.

Tabla 1. Instancias del DNA-FAP.

Instancia	Número Fragmentos	Óptimo
x60189_4	39	11478
bx842596_4	442	227920

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-SinDerivar 4.0 Internacional.

Para realizar un análisis paramétrico del cGA utilizado para resolver el DNA-FAP se han establecido 12 configuraciones diferentes. Las configuraciones se pueden observar en la Tabla 2, como podemos observar en la columna uno se coloca un nombre a la configuración para identificarla más adelante. En la columna dos y tres se encuentran el max_n y el max_m utilizados para la malla toroidal (tamaño de la población). En la columna cuatro se establece el número de evaluaciones a realizar por cada configuración. Se han utilizado 3 tamaños de población para los experimentos: 6x6 = 36 individuos, 8x8 = 64 individuos y 10x10 = 100 individuos. Para las configuraciones cGA_1 a cGA_6 el número de evaluaciones se estableció en 1.000.000 mientras que para las configuraciones cGA_7 a cGA_12 es de 500.000. Las columnas cinco y seis contienen el tipo de cruzamiento (OX) y su probabilidad (0,9), respectivamente. Por último, en las columnas siete y ocho se presentan los operadores de mutación utilizados (Swapp, inserción) y su correspondiente probabilidad de mutación (0,01)

Tabla 2: Valores de los parámetros para las diferentes configuraciones del cGA.

Configuraciones	Tamaño P		Num. Evaluaciones	Op. Cruce	Prob. Cruce	Op. Mutación	Prob. Mutación
	max_n	max_m					
cGA_c1	6	6	1.000.000	OX	0,9	Swapp	0,01
cGA_c2						Inserción	
cGA_c3	8	8				Swapp	
cGA_c4						Inserción	
cGA_c5	10	10				Swapp	
cGA_c6						Inserción	
cGA_c7	6	6	500.000			Swapp	
cGA_c8						Inserción	
cGA_c9	8	8				Swapp	
cGA_c10						Inserción	
cGA_c11	10	10				Swapp	
cGA_c12						Inserción	

Para cada una de estas 12 configuraciones se realizaron 30 ejecuciones para asegurar la independencia estadística de los resultados. Las mismas se realizaron sobre una computadora con Sistema Operativo Linux Ubuntu Release 16.04, cuya memoria interna es de 6Gb, y la capacidad de su disco duro de 65GB.

5. RESULTADOS

La información de cada experimento se muestra en las tablas 3 y 4. En la primera columna se listan las configuraciones, luego en la segunda se muestra el mejor resultado obtenido, en la tercer columna el peor valor y la media en la cuarta. La desviación estándar en la quinta columna. Y en la última, el tiempo de ejecución en segundos.
Del análisis de la información de estas tablas, se puede observar que para el caso de estudio de 39 fragmentos (x60189_4), el mejor comportamiento del algoritmo fue con la configuración cGA_c6.
Haciendo el mismo análisis, para el grupo de configuraciones con el que se analizó el caso de estudio de 442 fragmentos (bx842596_4) la configuración que obtuvo el mejor valor al ejecutar el algoritmo, fue la cGA_c5.

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-SinDerivar 4.0 Internacional.

Tabla 3. Resultados de las diferentes configuraciones del cGA para la instancia x60189_4.

Configuraciones	mejor	peor	media	std.	avg. time (seg.)
cGA_c1	10678,00	9603,00	10059,13	297,07	34629,97
cGA_c2	11197,00	10103,00	10558,77	249,16	37906,20
cGA_c3	10960,00	9372,00	10334,70	333,88	36313,80
cGA_c4	11264,00	*10190,00*	10686,07	**234,02**	35278,73
cGA_c5	11137,00	9890,00	10550,77	322,80	38472,20
cGA_c6	**11285,00**	**10226,00**	*10789,90*	*240,75*	35835,00
cGA_c7	10724,00	9343,00	9985,37	357,08	**11505,17**
cGA_c8	10963,00	10039,00	10485,17	262,05	16223,50
cGA_c9	11175,00	9393,00	10414,30	418,72	*14769,43*
cGA_c10	*11265,00*	10051,00	10657,93	285,19	18566,20
cGA_c11	11028,00	9634,00	10447,03	293,95	14858,63
cGA_c12	11200,00	10159,00	**10796,70**	259,12	19381,13

Tabla 4. Resultados de las diferentes configuraciones del cGA para la instancia bx842596_4.

Configuraciones	mejor	peor	media	std.	avg. time (seg.)
cGA_c1	143582,00	122239,00	133286,43	4676,59	**29304,37**
cGA_c2	143749,00	136617,00	139601,53	**1797,45**	177638,10
cGA_c3	158253,00	143976,00	150709,93	3294,35	88553,30
cGA_c4	135880,00	127721,00	131964,90	2131,90	253734,33
cGA_c5	**166973,00**	**151845,00**	**159501,70**	4034,26	53764,00
cGA_c6	129630,00	117246,00	124505,80	2380,80	256301,67
cGA_c7	134865,00	114739,00	125964,17	5456,81	*46630,27*
cGA_c8	126630,00	118638.00	123078,73	*1961,58*	126508,87
cGA_c9	156861,00	137920,00	147048,03	5053,44	55972,97
cGA_c10	122073,00	110747,00	114923,43	2352,68	130755,97
cGA_c11	*166710,00*	*149182,00*	*155437,50*	3912,46	53361,63
cGA_c12	112746,00	102169,00	108049,33	2323,21	134443,37

Si analizamos los dos operadores de mutación utilizados en los experimentos para la instancia x60189_4 (*swapp* e inserción) se observa que aquellas configuraciones cuyo operador de mutación es la inserción obtienen el mejor valor. Sin embargo, para la instancia bx842596_4 no ocurre lo mismo, el operador de inserción sólo obtiene el mejor valor, en las configuraciones cuyo tamaño de población es de 6x6, en el resto de las configuraciones, el valor para este operador es notablemente menor; consiguiendo los mejores valores con el operador swapp. Para el caso del tiempo de ejecución, en la instancia x60189_4, las configuraciones cGA_c1 a cGA_c6 tienen un tiempo de ejecución superior a las configuraciones cGA_c7 a cGA_c12, lo cual tiene sentido ya que el número de evaluaciones es el doble. Para la instancia bx842596_4 el tiempo de ejecución para las poblaciones de tamaño 6x6 y operador de mutación *swapp* el tiempo de ejecución con 500.000 evaluaciones es superior al obtenido con el mismo tamaño de población con 1.000.000 de evaluaciones. Por el contrario, en las poblaciones de tamaño 8x8 y 10x10 el tiempo de ejecución es mayor al incrementar el número de evaluaciones. Comparando el tiempo de cómputo que consume la

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-SinDerivar 4.0 Internacional.

ejecución del algoritmo según el operador de mutación utilizado, se observa que para la instancia de 39 fragmentos la diferencia entre los tiempos para las configuraciones es despreciable. No sucede lo mismo para la instancia de 442 fragmentos, ya que se observa que el tiempo de ejecución se incrementa considerablemente para el operador de inserción, en todas las configuraciones analizadas. Dado lo observado podemos concluir que la instancia de estudios de mayor cantidad de fragmentos, bx842596_4, nos permite analizar con mayor detalle el comportamiento del algoritmo según el operador de mutación utilizado. El mejor valor de aptitud en x60189_4, lo arroja el operador inserción. En tanto que el mejor valor de aptitud en bx842596_4, es obtenido con el operador de mutación *swapp*.

6. CONCLUSIONES

En este trabajo se propuso la aplicación de técnicas metaheurísticas para solucionar el problema de ensamblado de fragmentos de ADN. El objetivo es entonces encontrar el ordenamiento de fragmentos que permita obtener la secuencia original de ADN. Para esto, se ha realizado el estudio del comportamiento de un Algoritmo Genético Celular (cGA) con dos operadores de mutación diferentes (*Swapp* e Inserción) y un Operador de Cruce (*Order Crossover*).

Se analizó en detalle cada resultado obtenido luego de cada configuración propuesta para el algoritmo. Los resultados fueron promisorios. Para la instancia de 39 fragmentos los valores obtenidos por todas las configuraciones han sido similares y cercanas al óptimo 11478, no así para la instancia de estudio de 442 fragmentos lo que permite plantear a futuro el desarrollo de nuevos operadores especialmente para problema abordado.

En la instancia de 39 fragmentos, los resultados entre las diferentes configuraciones son similares. Sin embargo, para la instancia de 442 fragmentos claramente el operador *Swapp* permitió obtener los mejores resultados. Además, las tablas revelan claramente que las configuraciones con mayor tamaño de población y mayores evaluaciones obtienen los mejores resultados.

7. AGRADECIMIENTOS

Vidal y Olivera agradecen al CONICET, al Instituto Universitario para las Tecnologías de la Información y las Comunicaciones, y a la Facultad de Ingeniería de la Universidad Nacional de Cuyo (UNCuyo), Argentina. Este Trabajo se enmarca en el proyecto Tipo 1 B081 de la UNCuyo. Vidal agradece a la Universidad Nacional de la Patagonia Austral por el proyecto B29/220.

REFERENCIAS

DÍAZ, B. D. (2006). *Diseño e implementación de algoritmos genéticos celulares para problemas complejos*. UNIVERSIDAD DE MÁLAGA. Tesis de Maestría.

ENGLE, M. L., y Burks, C. (1993). Artificially generated data sets for testing DNA sequence assembly algorithms. *Genomics*, *16*(1), 286–288. https://doi.org/10.1006/geno.1993.1180

GREEN, P. (2009). *Phrap, version 1.090518, http://phrap.org*.

MALLÉN-FULLERTON, G. M., y Fernández-Anaya, G. (2013, June). DNA fragment assembly using optimization. *IEEE congress on evolutionary computation* (p. 1570-1577). https://doi.org/10.1109/cec.2013.6557749

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-SinDerivar 4.0 Internacional.

MINETTI, G., Alba, E., y Luque, G. (2008). Seeding strategies and recombination operators for solving the DNA fragment assembly problem. *Information Processing Letters*, *108*(3), 94-100. https://doi.org/10.1016/j.ipl.2008.04.005

MINETTI, G., Leguizamón, G., y Alba, E. (2012). SAX: a new and efficient assembler for solving dna fragment assembly problem. *41 Jornadas Argentinas de Informática. Simposio Argentino de Inteligencia Artificial*.

MINETTI, G. F. (2011). *Problema de ensamblado de fragmentos de adn resuelto mediante metaheurísticas y paralelismo*. UNIVERSIDAD NACIONAL DE SAN LUIS, ARGENTINA. Tesis de Maestría.

PEVZNER, P. (2000). *Computational molecular biology: An algorithmic approach*. MIT Press.

SUTTON, G. G., White, O., Adams, M. D., y Kerlavage, A. R. (1995). TIGR assembler: A new tool for assembling large shotgun sequencing projects. *Genome Science and Technology*, *1*(1), pp. 9–19. https://doi.org/10.1089/gst.1995.1.9

VIDAL, P., y Olivera, A. (2018). Ensamblado de fragmentos de adn utilizando un novedoso algoritmo de luciérnaga en GPU. *DYNA*, *85*(204). https://doi.org/10.15446/dyna.v85n204.60078

VIDAL, P., y Olivera, A. C. (2014). A parallel discrete firefly algorithm on GPU for permutation combinatorial optimization problems. En G. Hernández y cols. (Eds.), *High performance computing. 485*, 191-205. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-662-45483-1_14

Esta obra está bajo una Licencia Creative Commons Atribución-NoComercial-SinDerivar 4.0 Internacional.