Problema de ensamblado de fragmentos de ADN resuelto mediante metaheurísticas y paralelismo

Presentada para cumplir con los
requerimientos del grado de
DOCTOR EN CIENCIAS DE LA COMPUTACIÓN
en la
UNIVERSIDAD NACIONAL DE SAN LUIS
SAN LUIS, ARGENTINA

Autor:
Gabriela F. Minetti
Asesores:
Dr. Enrique Alba
Dr. Mario Guillermo Leguizamón

5 de octubre de 2011

Los abajo firmantes certifican que han leído y recomiendan a la Facultad de Ciencias Físico, Matemáticas y Naturales aceptar la tesis titulada "Problema de ensamblado de fragmentos de ADN resuelto mediante metaheurísticas y paralelismo" por D. Gabriela F. Minetti en cumplimiento parcial de los requerimientos para el grado de Doctor en Ciencias de la Computación.

Fecha: _____

Asesor Científico: _____

Dr. Enrique Alba

Co-Asesor Científico: _____

Dr. Guillermo Leguizamn

A mi hija, mi esposo y mis padres.

Agradecimientos

Este trabajo de tesis doctoral ha significado mucho esfuerzo, compromiso y trabajo de mi parte y, también, de muchas personas que desde su lugar han contribuido a la concreción de este objetivo en mi vida. Es por eso que quiero darles mi más sincero agradecimiento a todas ellas. Empezando por una persona que, casi sin conocerme, me brindó desinteresadamente su conocimiento, experiencia, guía y dedicación, ¡GRACIAS Enrique! Siguiendo por la persona que me brindó todo su apoyo, paciencia y dedicación para que pudiera comenzar este camino hace algunos años y luego recorrerlo con éxito, ¡GRACIAS Guillermo! También quiero agradecer a Gabriel Luque por su colaboración y guía en gran parte de este trabajo.

Tampoco pueden faltar en este reconocimiento mis compañeros de trabajo, con quienes compartí muchas horas de labor, discutí ideas e intercambié opiniones en un ambiente distendido y agradable. Caro, Hugo, Alina, Naty, Fernando y Carlos GRACIAS por acompañarme y escucharme durante todo este proceso. Extiendo mi agradecimiento sincero a Paco que tan atenta y expeditivamente atendió mis requerimientos de disponibilidad de máquinas.

Desde lo más profundo de mi corazón quiero agradecer a **toda mi familia** que siempre estuvo cerca mío para alentarme, ayudarme y hacerme este duro camino un poco más fácil. Especialmente a mis padres y a mi esposo, que estuvieron junto a mí apoyándome y brindándome todo su amor. Pero aún más profundo es mi agradecimiento a mi hija, quien cada día colma mi alma de paz y felicidad con su amor y candidez. ¡GRACIAS ANTO!

Por último, debo mi reconocimiento a las instituciones que permitieron el desarrollo de este trabajo: la Facultad de Ingeniería de la UNLPam donde diariamente desarrollo mi labor, la Universidad de Málaga por los recursos cedidos para desarrollar la parte experimental y la AGENCIA por concederme una beca de doctorado.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Esta tesis aborda el problema de ensamblado de fragmentos del genoma[1] de un organismo mediante la utilización de técnicas metaheurísticas. La obtención de un ensamblado completo y de alta calidad de un genoma tiene implicaciones directas en la Biología y la Medicina. Esta tarea es particularmente compleja cuando se trabaja con genomas de gran tamaño, como es el caso de la mayoría de los eucariotas (animales, plantas y hongos). Razón por la cual, es sumamente necesario contar con algoritmos ensambladores que permitan obtener secuencias genómicas de alta calidad en tiempos razonables y, así, proseguir de manera segura y eficiente con las etapas subsiguientes del proyecto de genómica. En el presente capítulo se introducen los principales cuestionamientos y desafíos que se deben enfrentar en el ámbito de la Bioinformática al explicar cómo esta tesis intenta contribuir a la solución del problema de ensamblado de fragmentos.

1.1. Antecedentes y Motivaciones

La mayoría de los problemas de la vida real muestran un alto grado de vinculación entre los parámetros (epístasis), muchas soluciones localmente óptimas (multimodalidad) y una alta dimensión. Estos problemas complejos están cobrando una mayor notoriedad en la actualidad; esto puede observarse en las áreas de: Comunicaciones, Bioinformática, Planificación, Ambientes Industriales, etc . En éstos y otros campos de investigación a menudo es

[1] Secuencia completa de Ácido Desoxirribonucleico, ADN .

esencial modelar y resolver tareas de optimización, de aprendizaje o de investigación para aplicaciones que no admiten una fácil formulación. De hecho, son frecuentes los casos donde el problema no es diferenciable, tiene un gran número de restricciones u objetivos, no admite las condiciones de contorno, o no está completamente definido.

Cuando es necesario tomar decisiones sobre el valor de ciertas condiciones del problema (por ejemplo: costo, peso, ganancias, tiempo, eficiencia, etc.) y tales decisiones afectan el resultado final de su resolución, entonces se enfrenta un problema de optimización. La optimización es una rama de las Matemáticas Aplicadas para encontrar la mejor o una muy buena solución en la resolución de problemas cuantitativos en muchas disciplinas; tales como: Física, Biología, Ingeniería, Bioinformática y Economía. La Bioinformática, específicamente, es un campo interdisciplinar[2] dedicado a desarrollar técnicas que permitan: analizar secuencias genéticas, identificar y predecir estructuras moleculares, extraer caracterísitcas de microarreglos de datos, etc. Actividades que, en su mayoría, necesitan ser formuladas como problemas de optimización para poder llevarse a cabo.

El conjunto de técnicas en Bioinformática utilizadas en las distintas áreas de la Biología es extenso y de componentes heterogéneos. Se pueden distinguir dos grandes grupos de técnicas algorítmicas. Uno de ellos está conformado por algoritmos diseñados para un uso bioinformático específico; por ejemplo: BLAST [11, 12] y CLUSTALW-pairwise [181] para alinear un par de secuencias de ADN, FASTA [139, 140], PSI-BLAST [12], SSEARCH [140] y HMMER-HSSP [144] para identificar relaciones entre proteínas, PHRAP [80], TIGR assembler [173], STROLL [37, 38], CAP3 (*Contig Assembly Program*) [93] y Celera Assembler [132] para ensamblar fragmentos de un genoma. En tanto que, el otro subconjunto está formado por un grupo de técnicas modernas de uso generalizado, denominadas metaheurísticas. Éstas se utilizan en casi todas las áreas de la Bioinformática y está representado por numerosas familias algorítmicas, a saber: los algoritmos genéticos, la optimización basada en colonias de hormigas y el enfriemiento determinístico en la alineación de secuencias [100, 135, 136, 121] y en el ensamblado de fragmentos [103, 113, 120, 122, 138], diversos algoritmos evolutivos se

[2] El crecimiento inconmensurable en el volumen y la variedad de información generada por avances en la Biología Molecular y en la Tecnología Genética subyacente han marcado la necesidad de involucrar el conocimiento de expertos pertenecientes a otras ciencias tales como las Matemáticas, las Ciencias de la Computación y la Física.

usan en la identificación de relaciones proteínicas [66, 155], la identificación del perfil de la expresión genética [51] y en el análisis de la estructura proteínica [66, 69, 97, 159, 166, 196].

La Bioinfórmática se divide, entonces, en distintos campos. Uno de ellos está directamente relacionado con la identificación de secuencias genómicas y proteómicas. En este campo se distinguen 3 áreas bien definidas: alineación de secuencias, identificación de relaciones proteínicas y ensamblado de ADN, siendo la última el objeto de estudio de esta tesis.

El ensamblado de fragmentos de ADN se formula como un problema de optimización combinatoria NP-duro [142]. Por consiguiente: el tamaño del genoma, el número de fragmentos leídos y secuenciados y la presencia de ruido en los datos son factores altamente influyentes en la capacidad de cualquier algoritmo para llevar a cabo esta tarea. También es ampliamente conocido el hecho que las metaheurísticas son técnicas usadas exitosamente en una amplia gama de problemas de optimización combinatoria NP-duros: encaminamiento, telecomunicaciones, secuenciación de tareas, planificación de recursos, corte y empaquetado, diseño ingenieril, entre muchos otros. 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 pueden prescindir de datos exactos y completos para obtener más y mejores soluciones. Así mismo, las metaheurísticas también han resultado ser eficientes cuando la complejidad del problema es alta y el espacio de soluciones asociado es grande o ambos crecen continuamente. Además son fácilmente paralelizables tanto a nivel algorítmico como de hardware. Estas dos últimas características son muy importantes a la hora de manipular enormes cantidades de información. Tales ventajas y características son difíciles de encontrar o incorporar en los algoritmos diseñados específicamente para resolver un solo tipo de problema.

Todo lo expresado anteriormente parece justificar con creces la elección de las técnicas metaheurísticas para resolver el problema de ensamblado de fragmentos, pero ¿son capaces de cumplir con las siguientes hipótesis?:

H1. Un algoritmo metaheurístico supera el estado del arte en el ensamblado de fragmentos (estado del arte).

H2. Las metaheurísticas son capaces de manipular genomas de gran tamaño sin disminuir la calidad de las soluciones halladas (complejidad).

H2.1 Si H2 se confirma, entonces la complejidad temporal no se transforma en un factor prohibitivo (eficiencia).

H3. Las metaheurísticas son robustas a la hora de operar con ruido en los datos (robustez).

Para estudiar la robustez de un ensamblador se analizan las diferencias entre las soluciones encontradas para las instancias sin y con ruido. Si no se detectan diferencias (estadísticamente significativas), el ensamblador muestra un comportamiento neutro (insensible, indistinto) a pequeñas variaciones en los datos de entrada. Consecuentemente, este ensamblador se considera robusto para resolver instancias ruidosas.

Con el propósito de confirmar estas hipótesis, se propone en este trabajo aplicar técnicas metaheurísticas al problema de ensamblado de fragmentos de ADN, formulando y analizando distintas posibilidades. De esta manera, se intenta sacar el máximo partido a dichas técnicas y ofrecer soluciones de gran calidad con recursos computacionales al alcance de cualquier institución.

1.2. Objetivos y fases

En esta sección se introducen los objetivos a alcanzar y explican el conjunto de fases que cumplimentar para dar respuestas a las hipótesis previamente formuladas. En consecuencia, los principales objetivos de esta tesis doctoral son: aplicar técnicas metaheurísticas al problema de ensamblado de fragmentos de un genoma, analizar los resultados para comprender el comportamiento de estos algoritmos y proponer nuevos métodos para resolver los problemas de manera más eficaz y eficiente. Para llevar a cabo los objetivos planteados en esta tesis se seguirán las fases que establece el método científico [99, 105] según F. Bacon:

1. **Observación**: Observar es aplicar atentamente los sentidos a un objeto o a un fenómeno, para estudiarlo tal como se presentan en realidad. En este caso, la fase de observación consistirá en el estudio del estado-del-arte en Bioinformática, específicamente en el área de ensamblado de fragmentos de ADN. Este estudio permitirá la

comprensión de los métodos que resuelven actualmente este problema, así como también, de las distintas técnicas metaheurísticas, y de los modelos existentes.

2. **Inducción:** La acción y efecto de extraer, a partir de determinadas observaciones o experiencias, el principio particular de cada una de ellas. En este paso, se estudiarán y analizarán los trabajos realizados hasta el momento sobre el objeto de estudio de esta tesis, poniendo de manifiesto sus principales debilidades y carencias.

3. **Hipótesis:** Planteamiento mediante la observación siguiendo las normas establecidas por el método científico. En este caso, se propondrán ideas que pueden resolver los problemas anteriormente manifestados basadas en la aplicación de metaheurísticas y en distintas formas de usarlas para conseguir ensambladores más eficientes que los existentes, en especial a través del uso del paralelismo, la hibridación y la manipulación algorítmica de soluciones en espacios de búsqueda complejos.

4. **Probar la hipótesis por experimentación:** Se implementarán nuevos modelos algorítmicos basándose en las mejoras propuestas en la hipótesis de trabajo, y se llevarán a cabo los experimentos.

5. **Demostración o refutación de la hipótesis:** Se analizarán los resultados obtenidos con los nuevos modelos desarrollados, comparándolos mediante estudios de significación estadística con los resultados publicados en la literatura.

6. **Conclusiones:** Se presentará las conclusiones a las que se arribó tras este trabajo de investigación, y sugerirán algunas líneas de trabajo futuro que surjan del presente estudio.

El método científico se sustenta en dos pilares fundamentales: la reproducibilidad y la falsabilidad, que establece que toda proposición científica tiene que ser susceptible de ser falsada.

En relación a la reproducibilidad, en todo momento se presentarán los detalles necesarios para que cada experimento pueda ser reproducido por cualquier otro investigador interesado en cualquiera de las aplicaciones propuestas en este trabajo. En cuanto a la falsabilidad, en todos los estudios se ofrecerán los resultados obtenidos de forma clara, estructurada y sencilla como prueba de las inferencias realizadas. Debido a la naturaleza estocástica de los

algoritmos a utilizar, se realizan un mínimo de 30 experimentos independientes. Además, para asegurar la relevancia estadística de las conclusiones, se aplica un conjunto de análisis estadísticos a los datos en todos los estudios realizados.

Con el fin de llevar a cabo los objetivos globales, arriba enunciados, se los ha descompuesto en objetivos parciales más concretos:

- Diseñar, implementar y evaluar modelos algorítmicos que resuelvan el problema en cuestión, potenciando las ventajas y atenuando o eliminando las desventajas de los existentes.

- Incorporar y evaluar técnicas descentralizadas y paralelas en los modelos algorítmicos obtenidos anteriormente para: (*i*) aumentar la calidad en las soluciones halladas; dado que este tipo de técnicas realizan una mayor exploración del espacio de búsqueda que los algoritmos centralizados y (*ii*) disminuir el tiempo de convergencia.

- Analizar y evaluar las técnicas desarrolladas a través de métodos estadísticos rigurosos que permitan: (*i*) compararlas con respecto a los enfoques existentes y (*ii*) demostrar la veracidad de los conclusiones obtenidas.

1.3. Contribuciones

En esta sección se enuncian en forma resumida y esquemática los aportes realizados con la presente tesis, a saber:

- Desarrollo de una nueva estrategia voraz para generar soluciones iniciales que incorpora información heurística sobre el problema.

- Diseño y evaluación de ensambladores metaheurísticos basados en trayectoria. Surge ISA un algoritmo basado en enfriamiento simulado que implementa un nuevo procedimiento para generar vecinos. También nacen FVNS y CVNS dos algoritmos basados en la búsqueda en vecindarios que adaptan la definición de la estructura de vecindarios a FAP y se diferencian en la evaluación de la calidad de la solución.

- Diseño y evaluación de ensambladores metaheurísticos basados en población. Se crean y evalúan diferentes algoritmos basados en algoritmos genéticos para intentar resolver FAP.
- Diseño y evaluación de metaheurísticas híbridas, GA+VNS y SAX, surgidas de la combinación de las características ventajosas de los algoritmos de base, con el fin de resolver situaciones específicas del problema en cuestión.
- Implementación de un generador de instancias de mayor complejidad que las encontradas en la literatura, debido a que estas últimas son pocas resultan de escasa complejidad para la prueba de los algoritmos.
- Implementación de un generador de instancias con ruido, de forma tal de poder estudiar la robustez de los algoritmos propuestos.
- Diseño y evaluación de metaheurísticas descentralizadas y paralelas, PH-PALS, con el fin de aumentar la calidad de las soluciones en las situaciones más complejas.

De cada uno de los puntos enumerados anteriormente se ha realizado una importante tarea de divulgación, tanto a nivel nacional como internacional, en conferencias y revistas de impacto (véase apéndice D).

1.4. Organización de la tesis

Este documento de tesis se estructura en cinco partes con el fin de presentar de forma ordenada y coherente los conocimientos adquiridos y resultados obtenidos luego de aplicar cada una de las fases del método científico. En la primera parte se enuncia y describe el conocimiento alcanzado durante la fase de observación, con el propósito de contextualizar y explicar el problema a tratar y los algoritmos utilizados en el ensamblado de fragmentos, haciendo particular hincapié en las metaheurísticas. De la fases 2, 3, 4 y 5 del método científico surgen nuevas propuestas metodológicas y los resultados de la aplicación de dichas propuestas al problema de ensamblado de fragmentos, que son presentados en las partes II y III de este trabajo de tesis. Específicamente en la segunda parte se trabaja con un conjunto de datos de mediana complejidad encontrados en la literatura. En tanto que en

la tercera se trabaja con dos conjuntos de datos de alta complejidad, bien diferenciados entre sí por la inclusión o no de ruido en los datos. En la cuarta parte se muestran las principales conclusiones que se desprenden de este trabajo, así como las líneas de trabajo futuro inmediatas que surgen de este estudio. Finalmente, se presenta la parte V con el propósito de incorporar los apéndices que se consideren necesarios.

Específicamente, la parte I se divide en tres capítulos con la intención de introducir de forma clara y concisa los fundamentos de esta tesis. Por esta razón en primer lugar se describen la formulación del problema y su contexto (capítulo 2), luego los principales algoritmos ensambladores encontrados en la literatura (capítulo 3) y, por último, los algoritmos metaheurísticos que se usarán como ensambladores en este trabajo (capítulo 4). En tanto que, en la segunda parte se dedica un capítulo a la descripción de los puntos de diseño comunes a las metaheurísticas usadas en esta tesis. A continuación, siguiendo la clasificación presentada en el capítulo 3, se desarrollan las propuestas metodológicas y los resultados obtenidos de su aplicación. Esto significa que en el capítulo 6 se tratan las metaheurísticas basadas en trayectoria y en el capítulo 7 las basadas en población. La parte III, como se menciona en él parrafo anterior, está dedicada al tratamiento de instancias del problema de alta complejidad; las cuales se diferencian principalmente por la presencia o no de ruido. Para llevar a cabo dicho tratamiento sólo se aplican aquellos algoritmos que en la segunda parte resultaron más eficientes, a partir de los cuales se generan nuevas propuestas. En función a todo lo explicitado anteriormente, esta tercera parte se divide en tres capítulos. Uno, capítulo 8, dedicado a la resolución de instancias de mayor complejidad pero sin errores y los otro dos, capítulos 9 y 10, destinados especialmente al tratamiento de los casos de ruido. Por último, y con el objetivo de lograr un análisis general, se dedica el capítulo final a las conclusiones sobre todo lo expuesto en el resto del documento y, también, a las principales líneas de trabajo futuro que surgen del presente estudio. A continuación se describe detalladamente el contenido de todos estos capítulos.

Parte I: Fundamentos de esta tesis

El **capítulo 2**, en primer lugar, explica los conceptos relacionados con el dominio del ADN y, así, poder introducir los problemas más importantes de la Bioinformáti-

ca. Seguidamente describe, en detalle, el problema de ensamblado de fragmentos y, finalmente, presenta una reseña de los algoritmos más usados para resolverlo.

El **capítulo 3** proporciona una introducción genérica sobre las metaheurísticas y los algoritmos ensambladores diseñados específicamente para ello.

El **capítulo 4** explica con un alto nivel de detalle las técnicas metaheurísticas concretas que se han diseñando y se usan a lo largo de la tesis.

Parte II: Resolución del problema de ensamblado usando metaheurísticas

El **capítulo 5** describe los puntos de diseño comunes a las metaheurísticas utilizadas en esta tesis. También introduce el paquete de software utilizado para implementar los algoritmos propuestos, hace una reseña de las instancias del problema que aparecen en la literatura y que son utilizadas para probar tales algoritmos y, por último, describe tópicos del diseño experimental comunes a todas las pruebas realizadas.

Los **capítulos 6 y 7** detallan las propuestas metodológicas propuestas para resolver el problema usando metaheurísticas basadas en trayectoria (ISA, PALS, FVNS y CVNS) y en población ($GA2o_{50}$, $GA2o_{100}$, GAG_{50} y GAG_{100} y GA+VNS), respectivamente. Muestran los experimentos realizados en cada caso y, finalmente, analizan y comparan los resultados obtenidos.

Parte III: Resolución de instancias complejas del problema de ensamblado de fragmentos

El **capítulo 8** presenta la generación de un conjunto de instancias de mayor complejidad y una nueva metaheurística híbrida (SAX). Las nuevas instancias son utilizadas para estudiar el comportamiento de las propuestas algorítmicas presentadas en la parte II y de SAX.

Los **capítulos 9 y 10** introducen la generación de ruido en tres etapas distintas del problema. Luego analizan el comportamiento de las metaheurísticas ante esta situación incorporando una nueva propuesta híbrida, descentralizada y paralela (PH-PALS).

Parte IV: Conclusiones y Trabajo futuro

Esta tesis finaliza con conclusiones sobre todo lo expuesto en el resto del documento. Asimismo, se describen también las principales líneas de trabajo futuro que surgen del presente estudio.

Parte V: Apéndices En los apéndices A y B se pueden encontrar los resultados experimentales correspondientes a las metaheurísticas presentadas en los capítulos 8 y 9, respectivamente.

En el apéndice C se resumen los resultados obtenidos por las distintas alternativas algorítmicas a partir de las cuales se ideó PH-PALS.

El apéndice D presenta la publicaciones del doctorando realizadas durante la elaboración de la tesis. Estas publicaciones sustentan la calidad de la presente tesis doctoral.

Cabe destacar que antes de iniciarse esta labor existían desarrollos metaheurísticos para resolver FAP que consistían tan solo en el uso de las metaheurísticas básicas, algunas de ellas distribuidas, para resolver un conjunto de instancias del problema reducido y de mediana complejidad. Con el desarrollo de esta tesis se han aportado nuevas técnicas metaheurísticas híbridas, una de las cuales es distribuida. Estos nuevos algoritmos resuelven las instancias anteriores pero también son capaces de enfrentar eficientemente instancias de mayor complejidad con y sin ruido. Todo ello permite establecer objetivamente que se ha realizado una contribución y ensanchamiento del dominio de aplicación y de la eficiencia/eficacia de las metaheurísticas existentes hasta el momento.

Parte I

Fundamentos de esta tesis

Capítulo 2

Bioinformática y el problema de ensamblado de fragmentos

En las últimas décadas, los avances en el campo de la Biología Molecular y en las tecnologías de genómica han provocado un crecimiento muy fuerte en la información biológica generada por la comunidad científica. La secuenciación de genomas y de proteomas, la identificación de genes, la generación de perfiles de expresión genética y otras áreas genéticas han demostrado la necesidad de la participación de expertos matemáticos, ingenieros y físicos para obtener resultados en corto tiempo y de mayor calidad en estas áreas.

La Bioinformática es, entonces, un campo interdisciplinar que involucra a los expertos antes mencionados para: analizar la secuencia genómica, identificar y predecir las estructuras moleculares, determinar el perfil de expresión genética, etc. Esta disciplina incluye técnicas computacionales y aplicaciones que llevan a cabo esas actividades, siendo enorme el número de estas técnicas y aplicaciones y muy significativas las diferencias entre ellas. En este capítulo se explican los conceptos relacionados con el dominio del ADN y así poder introducir los problemas más importantes de la Bioinformática. Luego se describe detalladamente el problema de ensamblado de fragmentos. Por último, se presenta una reseña de los algoritmos más usados para resolver este problema.

2.1. Introducción al dominio de ADN

El *genoma* o secuencia completa de ADN de un organismo constituye la información genética heredable del núcleo celular, es el material genético almacenado en cada una de las células de un organismo. El *ácido desoxirribonucleico* (ADN, DNA son sus siglas en Inglés) contiene la información genética usada en el desarrollo y el funcionamiento de los organismos vivos conocidos y de algunos virus, además de ser responsable de su transmisión hereditaria. EL ADN es un polímero[1] de unidades menores denominados nucleótidos. Se trata de una molécula de gran peso molecular (macromolécula) que está constituida por tres sustancias distintas: ácido fosfórico, un monosacárido aldehídico del tipo pentosa (la desoxirribosa[2]) y una base nitrogenada cíclica que puede ser púrica[3], adenina (A) o citosina (C), o pirimidínica[4] tiamina (T) o guanina (G) (véase figura 2.1). La unión de la base nitrogenada (C, A, G o T) con la pentosa (desoxirribosa) forma un nucleósido; éste, uniéndose al ácido fosfórico, da un nucleótido; la unión de los nucleótidos entre sí en enlace diéster da el polinucleótido, en este caso el ácido desoxirribonucleico. Las bases nitrogenadas se hallan en relación molecular 1:1, la relación adenina + timina / guanina + citosina es de valor constante para cada especie animal. Estructuralmente la molécula de ADN (véase figura 2.2) se presenta en forma de dos cadenas helicoidales alrededor de un mismo eje (imaginario); las cadenas están unidas entre sí por las bases de los pares que la forman. Los emparejamientos son siempre adenina-timina y citosina-guanina. Los extremos de cada una de las hebras (o filamentos) del ADN son denominados $5'P$ (fosfato) y $3'OH$ (hidroxilo) en la desoxirribosa. Las dos cadenas se alinean en forma paralela, pero en direcciones inversas (una en sentido $5' \rightarrow 3'$ y la complementaria en el sentido inverso), pues la interacción entre las dos cadenas está determinada por los puentes de hidrógeno entre sus bases nitrogenadas. Se dice, entonces, que las cadenas son antiparalelas.

Se denomina *secuenciación* a la determinación de la estructura de una secuencia de ADN, es decir, el tipo y orden de sus nucleótidos. El término *secuencia* designa la composición

[1] Los polímeros son macromoléculas (generalmente orgánicas) formadas por la unión de moléculas más pequeñas.

[2] La desoxirribosa es un monosacárido de cinco átomos de carbono

[3] Base púrica por ser derivada de la purina de dos anillos heterocíclicos

[4] Base pirimidínica por ser derivada de la pirimidina de un solo anillo

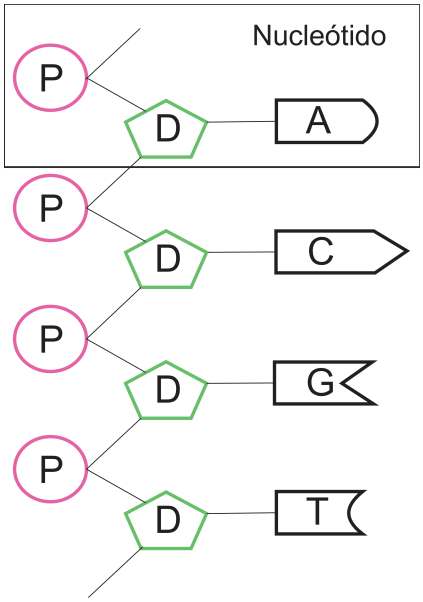


Figura 2.1: Representación esquemática de cuatro eslabones de una cadena de nucleótidos. Ácido fosfórico (P), desoxirribosa (D) y bases nitrogenadas (A, C, G, T).

de nucleótidos de un trozo de ADN o la de *aminoácidos*[5] de una proteína. Ese trozo de ADN puede corresponder a un gen, un genoma, o a una parte de ellos. Históricamente hay dos métodos de secuenciación del ADN: Maxam & Gilbert, o secuenciación química y el método de Sanger, que usa dideoxinucleotidos. Actualmente el segundo método es el más usado en los laboratorios (además de las técnicas de secuenciación masiva). El resultado de la secuenciación es la reconstrucción del ADN en lecturas de fragmentos de alrededor de 700 nucleótidos. Para poder completar un genoma es necesario *ensamblar los fragmentos* y así reconstruir la cadena original completa del ADN secuenciado. Una secuencia, entonces, está dada por el orden de las bases a lo largo de una cadena de ADN. Las bases de nucleótidos del conjunto A, T, C, G forman tri-nucleótidos. Una secuencia de ADN está compuesta esencialmente de dos conjuntos de tri-nucleótidos. Uno de ellos es la parte de codificación de ADN, donde el uso de los tri-nucleótidos, también denominados *codones*, tiene como objetivo la codificación de una proteína. Estas proteínas son responsables en la conducción

[5] Aminoácido: sustancia química orgánica que constituye el componente básico de las proteínas, compuesta por un grupo amino y un grupo carboxílico. Todas las proteínas de los seres vivos están compuestas por la combinación de 20 aminoácidos.

de la maquinaria enzimática de los organismos vivos. El otro conjunto de tri-nucleótidos es la parte no codificada del codón en el ADN, que no está involucrado en la codificación de la proteína.

La información del ADN es mantenida y ordenada de forma lineal y es la que guía la conformación de la estructura tridimensional de la proteína. Inicialmente, la información lineal de los nucleótidos se transcribe a la información lineal de las secuencias de nucleótidos pertenecientes al ácido ribonucléico (ARN); después de eso, las secuencias de nucleótidos del ARN son traducidas a la información lineal de las secuencias de aminoácidos. En el polipéptido[6], la secuencia lineal (estructura primaria) dispone su estructura secundaria y esto determina su organización tridimensional (estructura terciaria). La determinación de una conformación óptima constituye el plegamiento de proteínas. Es un proceso muy complejo, que suministra grandes cantidades de información sobre la presencia de sitios activos[7] y la posible interacción con medicamentos. Las proteínas en diferentes organismos que están relacionados entre sí por la evolución de un ancestro común, se denominan homólogas. Esta relación puede ser reconocida por la comparación de secuencias múltiples.

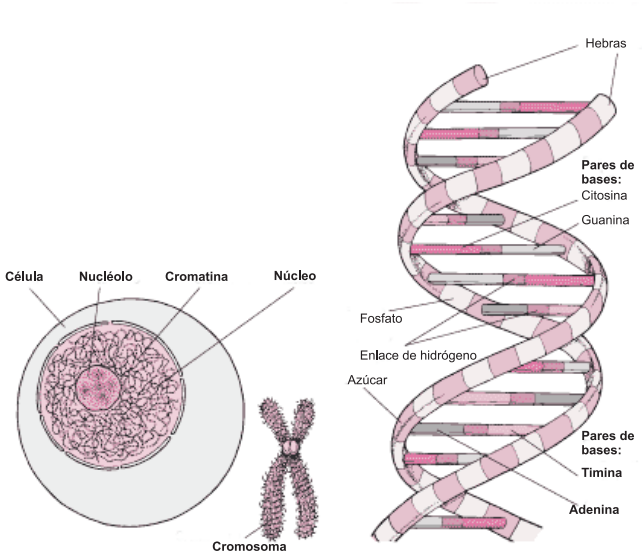


Figura 2.2: Molécula de ADN.

[6] Péptido: unión de varios aminoácidos lo suficientemente grande, que cuando tiene una estructura tridimensional única y estable, es una proteína.

[7] Sitio activo: sitio específico en la superficie de la proteína reconocido por los substratos.

Las herramientas informáticas de análisis de datos genómicos han madurado mucho desde su aparición. Los métodos para alinear las secuencias de ADN se han desarrollado hasta convertirse en los núcleos necesarios de análisis de la secuencia automática. El objetivo de la alineación de secuencias de ADN es alinear las dos bases en dos o más secuencias reduciendo al mínimo el número de discrepancias. El método de Needleman-Wunsch ha sido uno de los métodos originales que hizo su aparición en 1970 [133]. Esta técnica utiliza la programación dinámica para encontrar una alineación óptima para un par de secuencias. Desde Needleman-Wunsch se han incorpardo una serie de nuevos métodos de alineación refinados basados en: programación dinámica [164, 165], BLAST [11, 12], la visualización de secuencias [194], el procesamiento de señales [13, 34, 35, 134], el enfriamiento simulado [121], los algoritmos genéticos [135, 136], los sistemas de colonias de hormigas [100], entre otros.

2.2. Clasificación de los principales problemas de optimización en Bioinformática

En esta sección se introducen las principales áreas de trabajo en Bioinformática. Para esto se ha realizado una extensa revisión del estado del arte que aquí se resume con un enfoque sesgado a las áreas de interés de esta Tesis.

2.2.1. Secuencias genómicas y proteómicas

Para estudiar la información funcional y estructural de una secuencia desconocida de ADN un biólogo compara esta secuencia con otras ya conocidas y bien estudiadas. Si existe similitud entre ambas, podrían tener la misma función. Algunos de los métodos para identificar nuevas secuencias son: alineamiento de secuencias, identificación de proteínas relacionadas y ensamblado de ADN.

- *Alineamiento de secuencias.* Permite comparar y alinear dos (*Pairwise Alignment*) o más secuencias (*Multiple Sequence Alignment*, MSA). Dicho procedimiento busca series de características individuales o patrones de características que se presenten en el mismo orden dentro de una secuencia dada. Las técnicas más conocidas en este área son: Programación Dinámica para alinear pares de fragmentos [133, 165, 164], BLAST

[11, 12], CLUSTALW-pairwise y CLUSTALW-MSA [181], técnicas para la visualización de secuencias [194], procesamiento de señales [13, 34, 35] enfriamiento determinístico (*Deterministic Annealing*) para MSA [121], Algoritmos Genéticos (*Genetic Algorithms*, GAs) tales como SAGA y COFFE [135, 136] y optimización basada en colonia de hormigas (ACSs) [100].

- *Identificación de proteínas relacionadas en familias, superfamilias y pliegues.* Las proteínas pueden presentar similitudes estructurales considerables, aún cuando no existan relaciones evolutivas detectadas entre ellas. Cuando esto ocurre, se dice que la proteína comparte sólo un pliegue (*Fold Level*). Pero en cada pliegue pueden existir secuencias con el mismo origen, a esto se lo denomina superfamilia. Además, dentro de una superfamilia, existen secuencias similares, llamadas familias. A continuación se presenta una serie de algoritmos que permiten detectar esta clase de relaciones: FASTA [139, 140], SAM [94], SAM-T98 hidden Markov model [101], Pfam [168], PSI-BLAST [12], SSEARCH [140], SAM-HSSP [94], T99-HSSP [144], HMMER-HSSP [144], métodos basados en la predicción [64, 85, 149, 162], métodos basados en la estructura [26, 65, 96] y algoritmos evolutivos (EAs) para la derivación de perfiles escasos sin la necesidad de un alineamiento múltiple [155].

- *Problema de ensamblado de fragmentos* (*Fragment Assembly Problem*, FAP). La resolución de este problema conforma la última fase del método de secuenciamiento *shotgun*. El ensamblado de fragmentos de ADN se divide en tres fases diferentes: fase de superposición (encuentra los fragmentos superpuestos), fase de distribución (encuentra el orden de los fragmentos basado en el puntaje de similitud computado) y por último, la fase de consenso (deriva la secuencia de ADN a partir de la distribución anterior). En la siguiente sección se describe detalladamente el método de secuenciamiento antes mencionado y FAP.

2.2.2. Identificación de genes

La identificación automática de genes a partir de grandes secuencias es un objetivo importante para la Bioinformática y abarca las siguientes problemáticas: predicción de genes, análisis de la estructura protéica (*protein structure analysis* o *protein folding problem*), problema de acoplamiento molecular (*molecular docking problem*) y acoplamiento de proteínas (*protein-protein docking*), entre otros.

- *Predicción de genes.* Se examina una secuencia buscando regiones con alto grado de codificación. Predecir con precisión la estructura del gen es un requisito previo para la anotación funcional detallada de genes y genomas. Este problema es uno de los más difíciles en el campo de reconocimiento de patrones; dado que las regiones codificantes normalmente no tienen motivos específicos. La detección potencial de una región genómica codificante, depende de las características asociadas a los genes, y pueden ser muy difíciles de detectar. Algunos métodos para resolver este problema son: estadísticas para la codificación [30, 63, 61, 82], análisis espectral [167, 172, 200], reconocimiento de patrones y aprendizaje automático [62, 191], modelos de aprendizaje probabilístico que usan modelos Markov (*Hidden Markov models*, HMM) [154, 156, 182, 187, 199] y motores de inferencia difusa [15] que identifican el borde entre regiones codificadas y no codificadas.

- *Análisis de la estructura proteínica.* Aquí se estudian las propiedades de la conformación espacial de las moléculas de proteína que provienen de su secuencia de aminoácidos. Este tipo de análisis puede ser dividido en:

    1. *Estructura primaria de la proteína.* Dicha estructura corresponde a la secuencia lineal de aminoácidos de la proteína en estudio, e indica qué aminoácidos componen la cadena de polipéptidos y el orden en que dichos aminoácidos se encuentran. La función de una proteína depende de su secuencia y de la forma que ésta adopte. Las redes neuronales artificiales (*Artificial Neural Networks*, ANNs) [31, 112, 184, 190] y los GAs [135, 136] son algunas de las técnicas usadas para identificar esta estructura.

2. *Estructura secundaria de la proteína.* Para predecir completamente la estructura 3D de una proteína (estructura terciaria de la proteína) es necesario identificar la conformación local de la cadena de polipéptidos, llamada estructura secundaria [41]. Esta estructura representa la disposición de la secuencia de aminoácidos en el espacio. Es decir, se forma al enrollarse la estructura primaria helicoidalmente sobre sí misma. Existen dos tipos de estructura secundaria: la hélice alfa y la conformación beta. Los siguientes son métodos y técnicas usadas para predicir la estructura secundaria: Método Estadístico [41], Vector de Momentos (*Composition moment vector*) [153], Predicción Estadística [129, 151] y ANNs [145, 150, 152].

3. *Estructura terciaria de la proteína y pliegues.* Esta estructura hace referencia a la estructura tridimensional de la proteína. Generalmente la función proteínica depende de dicha estructura. El objetivo es determinar el estado de mínima energía para un pliegue de una cadena de polipéptidos. El proceso de plegamiento (conocido en Inglés como *folding*) involucra la minimización de una función de energía. Para realizar ese trabajo se han utilizado los siguientes métodos: ANNs [24], GAs [66, 97, 159] y Predicción Estadística [21].

- *Problema de acoplamiento molecular.* El problema consiste en determinar cómo interactúan las moléculas entre sí y qué papel clave juegan en el entendimiento de la función celular. Resolver este problema ayuda a predecir una asociación segura o la que mejor se adapte independientemente de la estructura molecular, permitiendo cambios conformacionales de las moléculas individuales durante la construcción de un complejo molecular. El acoplamiento se utiliza con frecuencia para anticipar la orientación de las moléculas pequeñas candidatas a formar medicamentos con sus proteínas objetivos con el fin de predecir a su vez, la afinidad y la actividad de la molécula pequeña. Algunas de las técnicas más conocidas para resolver este problema son los algoritmos evolutivos [69, 97, 166, 196].

- *Acoplamiento de proteínas.* Es la determinación de la estructura molecular de complejos formados por dos o más proteínas. Este campo está altamente orientado al trabajo computacional y comparte métodos con el problema de Acoplamiento Molecular. El

cual suele referenciarse como *small-molecule docking*, para distinguirlo del problema de Acoplamiento de Proteínas. Algunas de las técnicas usadas para este último acoplamiento son los métodos de: Espacio Recíproco [102], Montecarlo [79] y de Confiabilidad Interactiva por medio de Interacciones entre Caminos Alternativos (*Interaction Reliability by Alternative Path, IRAP, Interactions*) [36].

- *Otros.* En este punto se presentan diferentes técnicas usadas para identificar más de un tipo distinto de estructura proteínica, a saber: Cristolografía de Rayos X (la cual no es una técnica bioinformática), Espectroscopía de la Resonancia Magnética Nuclear (NMR) (tampoco es una técnica bioinformática), Modelado de dinámicas moleculares (MD) [198], técnica de búsqueda *Primary Hill climbing* (usa características de *Simulated Annealing*) [32, 163], Metaheurísticas usadas en modelos *HP* (*H-hidrofobic amino acid, P-polar*)[22, 108] y Máquinas Vectoriales de Soporte combinadas con la idea de reglas de asociación (GSVM-AR) para predecir la homología entre dos proteínas dadas [179].

2.2.3. Identificación del perfil de la expresión genética

Este proceso consiste en determinar cuándo y dónde los genes son expresados. La expresión[8] de un gen es a menudo regulada por la expresión de otro gen. Un análisis detallado de toda esta información permitiría un entendimiento sobre la red de interconexiones de diferentes genes y de sus roles funcionales [114]. La tecnología usada para este propósito se denomina *MicroArray Data*. Dicha tecnología posibilita el estudio simultáneo de decenas de miles de secuencias distintas de ADN. Los métodos más conmúnmente usados son: procesamiento de imágenes [95, 68, 170, 185], descubrir patrones por medio del análisis de clusters [3, 44, 86, 87], análisis de perfiles temporales y regulación genética, [52, 58, 91, 197], *Non-Dominated Sorting-GA*, NSGA-II [51], eliminación recursiva de característcas con máquinas de vectores de soporte (*Recursive feature elimination with support vector machines*, RFE-SVMs) [92] y optimización basada en cúmulos de partículas (*Particle Swarm Optimization*, PSO) [7].

[8]Expresión genética es la traducción de la información genética de la forma de ácido nucleico a la forma de proteínas.

2.2.4. Otros problemas

A continuación se presentan otros problemas que no encajan claramente en la clasificación descripta anteriormente pero igualmente importantes:

- *Interferencia del ARN (Ácido Ribonucleico).* Es una técnica genética descubierta recientemente con aplicaciones terapéuticas y genómicas. El método computacional usado para resolver el problema es el de ramificación y poda (*Branch and Bound*) con restricciones [201].

- *Análisis de los mecanismos de control.* Los mecanismos utilizados por los organismos vivos para adapatarse a diferentes situaciones pueden ser modelados computacionalmente. *Artificial C. elegans*, un modelo de circuito neuronal, es una técnica útil para modelar los movimientos en un organismo determinado [174].

2.3. Problema de ensamblado de fragmentos de ADN (FAP)

Antes de explicar detalladamente el problema de ensamblado de fragmentos es necesario describir el proceso de secuenciación que ha sido introducido en la sección anterior. Uno de los métodos más usados es el *Shotgun Sequencing* de Sanger [157] y consiste en (véase figura 2.3):

1. Generar múltiples copias de la secuencia de ADN original y dividir cada una de ellas en miles de fragmentos aleatorios (ver pasos 1 y 2 de la figura 2.3).

2. Estos fragmentos son leídos por una máquina de secuenciación de ADN. En esta fase es donde se realiza la secuenciación propiamente dicha, que identifica cada una de las bases nucleótidas presentes en el fragmento (ver pasos 3 y 4 de la figura 2.3).

3. Un ensamblador une los fragmentos leídos que se superponen, reconstruyendo la secuencia original (ver pasos 5 y 6 de la figura 2.3).

En los siguientes párrafos se describe la segunda fase de este método. En tanto que la tercera etapa se explica en la siguiente sección, ya que, como se mencionó con anterioridad, se trata del problema de ensamblado de fragmentos objeto de estudio de esta tesis.

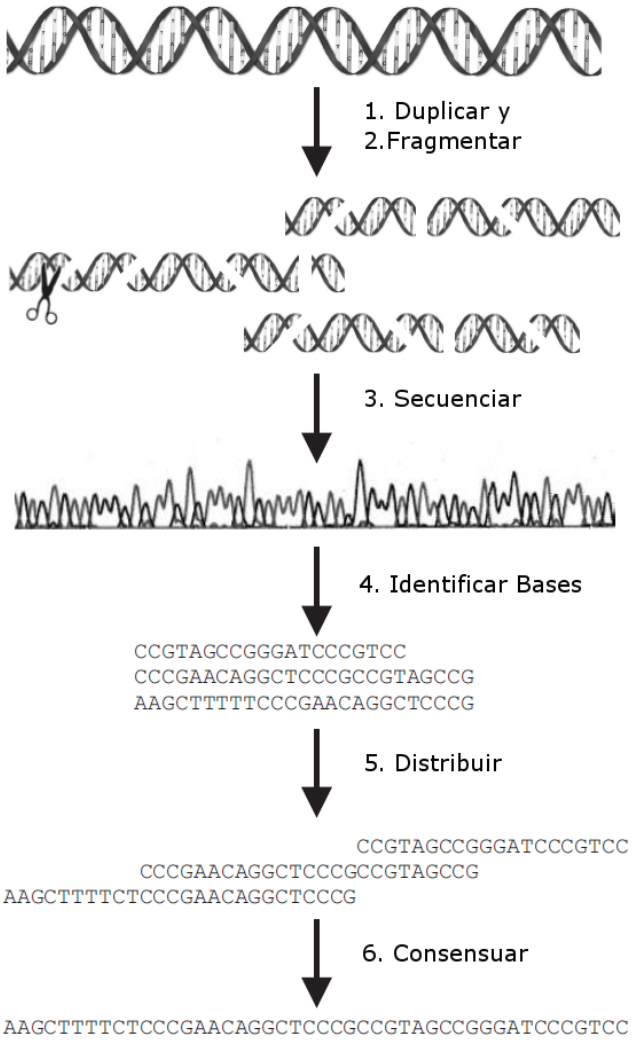


Figura 2.3: Representación gráfica de secuenciamiento y ensamblado de ADN.

La secuenciación de ADN utilizando el *método didesoxi* de Sanger emplea nucleótidos modificados (didesoxinucleótidos), que no poseen el hidróxilo (OH) en el extremo 3′. El ADN se sintetiza in vitro utilizando un molde de la cadena que se desee secuenciar, un exceso de sustratos nucleótidos desoxi, una pequeña cantidad de didesoxi específico (A, T, C o G), un cebador o *primer* y polimerasa.

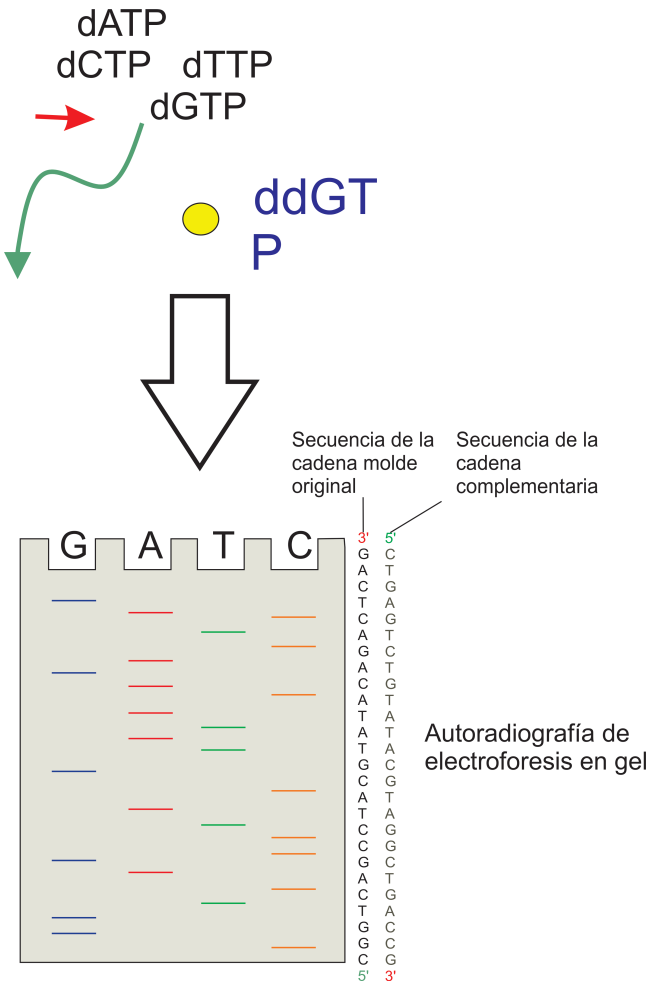


Figura 2.4: Secuenciación de ADN.

En la figura 2.4 se ilustra el proceso de secuenciación. Para llevar a cabo dicho proceso se coloca en un tubo los cuatro desoxinucleótidos (dATP, dCTP, dTTP, dGTP) más un didesoxinucleótido (ddGTP), además del primer, el molde y la polimerasa (círculo). El resultado

de la reacción de síntesis son cadenas de distinta longitud separadas por electroforesis en gel. La incorporación de un nucleótido didesoxi hará que termine el proceso de síntesis. Esto se debe a que la polimerasa necesita un grupo hidróxilo en la posición $3'$ para poder agregar el siguiente nucleótido (si este grupo no está presente, la polimerasa no puede continuar con la síntesis). Una vez sintetizado el ADN, se realiza una corrida en gel sembrando las productos de las reacciones correspondientes al agregado de cada uno de los nucleótidos didesoxi. De esta manera, se pueden ver distintas bandas correspondientes a tamaños diferentes. Si se leen los carriles de abajo hacia arriba (es decir, de menor a mayor tamaño), se tendrá la secuencia del ADN elegido (en orientación $5' \rightarrow 3'$).

2.3.1. Descripción del problema

Una vez que todos los fragmentos del genoma original han sido secuenciados, es necesario ensamblarlos para obtener la secuencia de ADN original. El ensamblado de fragmentos de ADN es un problema resuelto en las primeras fases del proyecto del genoma y por lo tanto muy importante, ya que los demás pasos dependen de su precisión. El proceso de ensamblado de fragmentos (figura 2.5) consiste en: una primera fase de superposición, una segunda de distribución y una tercera de consenso.

Fase de superposición

Encuentra y determina el tamaño de la superposición (o solapamiento) entre los fragmentos. Esta fase consiste en encontrar la mejor correspondencia o la más larga entre el sufijo de una secuencia y el prefijo de otra. En este paso, se comparan todos los posibles pares de fragmentos para determinar su similitud. Por lo general, un algoritmo de programación dinámica aplicada a la alineación semiglobal se utiliza en este paso. Es muy probable que los fragmentos con un alto grado de solapamiento estén uno seguido de otro en la secuencia destino.

El número de bases que se superponen entre dos fragmentos alineados, se llama *puntaje de solapamiento*. Con el fin de obtener el mencionado puntaje, cada posible orientación de los dos fragmentos es evaluada y luego se eligen: la superposición, la orientación y la alineación que maximice el número de bases coincidentes entre ambos fragmentos. Si no

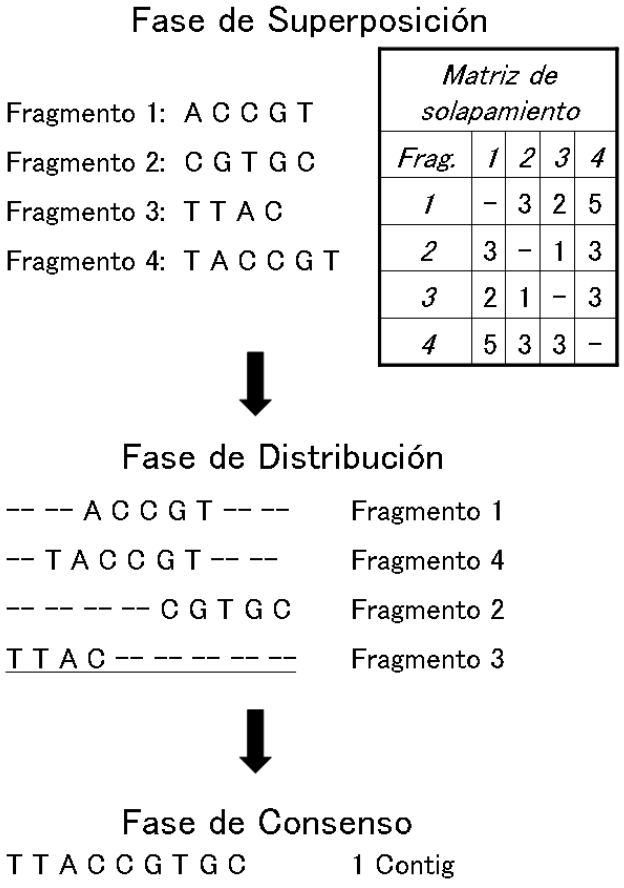


Figura 2.5: Fases en el ensamblado de fragmentos.

existe coincidencia entre dos fragmentos y ambos aparecen contiguos en la etapa del consenso entonces habrá un vacío en la secuencia final.

Fase de distribución

Encuentra el orden de los fragmentos basado en el puntaje de similitud computado en la fase anterior. Además de la complejidad que implica resolver el problema de optimización combinatoria de ordenamiento, este paso resulta el más complicado ya que es difícil conocer el solapamiento real dados los siguientes inconvenientes:

1. *Orientación desconocida.* Después de cortar a la secuencia original en muchos fragmentos, se pierde la orientación. No se sabe qué cadena debe ser seleccionada. Si un

fragmento no tiene ningún tipo de solapamiento con otro, todavía es posible que su complemento sí presente esa coincidencia.

2. *Errores en la identificación de bases.* Hay tres tipos de errores en la identificación de bases: inserción, sustitución y eliminación. Se producen debido a errores experimentales durante la electroforesis (el método utilizado en los laboratorios para leer las secuencias de ADN). Los errores afectan la detección de solapamientos entre fragmentos. Por lo tanto, la determinación del consenso requiere de múltiples alineaciones en las regiones de alta.

3. *Cobertura incompleta.* Se produce cuando el algoritmo no es capaz de ensamblar un determinado conjunto de fragmentos en un solo *contig*. Un *contig* es una secuencia en la que la solapamiento entre los fragmentos adyacentes es mayor o igual a un umbral predefinido (parámetro de corte denominado $cutoff$).

4. *Regiones repetidas o* Repeats. Son secuencias que aparecen dos o más veces en la secuencia de ADN. Las regiones repetidas han causado problemas en muchos proyectos de secuenciación de genomas, y se torna dificultosa para los programas actuales de ensamblado poder manejarlos perfectamente.

5. *Quimeras y contaminación.* Las quimeras surgen cuando dos fragmentos que no son adyacentes o superpuestos en la molécula se unen en un solo fragmento. La contaminación se produce debido a la depuración incompleta del fragmento desde el vector de ADN.

Fase de consenso

Deduce la secuencia de ADN usando el ordenamiento de fragmentos obtenidos anteriormente. La técnica más comúnmente utilizada en esta fase es aplicar la regla de la mayoría en la construcción del consenso. Para medir la calidad de un consenso, podemos ver la distribución de la cobertura. La cobertura en una posición de base se define como el número de fragmentos en esa posición. Se trata de una medida de la redundancia de los datos del fragmento; que denota el número promedio de fragmentos que se espera aparezca un nucleótido

en la secuencia de ADN. Se calcula como el número de bases leídas en los fragmentos sobre la longitud de la cadena de ADN [161]:

$$Cobertura = \frac{\sum_{i=1}^{n} longitud\ \ del\ \ fragmento\ \ i}{longitud\ \ de\ \ la\ \ secuencia} \tag{2.1}$$

donde n es la número de fragmentos. Cuanto mayor sea la cobertura, menor es el número de espacios en blanco y mejor es el resultado.

Resumiendo, el conjunto de fragmentos de ADN en una secuencia de consenso que corresponde a la secuencia padre constituye el "problema de ensamblado de fragmentos" [161]. Es un problema de permutación NP-duro [142], por lo tanto, no existe (asumiendo P $\neq$ NP) un algoritmo exacto que resuelva este problema en tiempo polinómico. Desde el punto de vista de la optimización combinatoria, la construcción de un consenso es similar a la de un recorrido en un problema del viajante de comercio (*Travelling Salesman Problem*, TSP). Esto es porque cada fragmento tiene una ubicación específica en la formación de una secuencia en la etapa de consenso. Aunque los puntos terminales de un recorrido de TSP sean irrelevantes ya que su solución es un recorrido circular de ciudades, en el caso de FAP estos puntos son importantes ya que ellas representan los extremos opuestos de la secuencia original de ADN. En TSP el ordenamiento de las ciudades es la solución final al problema. En cambio para FAP, el ordenamiento de fragmentos es sólo un resultado intermedio que será utilizado en la fase de consenso.

2.3.2. Trabajos relacionados

La mayoría de los algoritmos ensambladores de secuencias, tales como: PHRAP [80], CAP3 [93], Celera assembler [132], TIGR Assembler [173], STROLL [39, 38] y EULER [142], están basados en alguna variación de un algoritmo voraz. En este enfoque voraz (o *greedy*), los fragmentos son ensamblados al combinar el par de fragmentos con mayor superposición. Si bien este método es muy rápido, las regiones repetidas ocasionan inconvenientes en dicho proceso.

Sin embargo, las técnicas metaheurísticas que están siendo usadas para resolver este problema brindan mejores resultados, dado que pueden resolver problemas de muy alta dimensión con fuertes restricciones de manera eficiente. Entre ellas se encuentran: los algo-

ritmos evolutivos [43, 55, 103, 113, 120, 138], la optimización basada en colonia de hormigas [122], los métodos de enfriamiento simulado [43, 42], etc.

Parsons et al. en [138] analizan la aplicación de algoritmos genéticos (*Genetic Algorithms*, GAs) a instancias de FAP con secuencias de 10Kbp. Para llevar a cabo este estudio usan diferentes representaciones de los individuos y por ende distintos operadores genéticos. Además proponen dos nuevas funciones de fitness para evaluar la calidad de las soluciones. La primera de estas funciones es la usada en este trabajo de tesis y se describe detalladamente en el capítulo 5.

En [103], Kim y Mohan proponen un GA adaptativo, jerárquico y paralelo (PHAGA) para resolver FAP. En PHAGA las soluciones se representan con una lista doblemente enlazada de secuencias, el tamaño de la población es un parámetro determinado por las características de los datos de entrada, mientras que las subsecuencias con un alto nivel de confianza son congeladas para futuras iteraciones del algoritmos. De esta forma, el algoritmo reduce el espacio de búsqueda progresivamente durante la evolución.

En [113], Li y Khuri analizan la resolución de FAP mediante el uso de GAs. Para ello emplean la representación por permutación, el operador de mutación *swap* y los operadores de cruce basados en el orden (OX) y en la recombinación de ejes (EX). Para la experimentación utilizan instancias con hasta 77Kbp y concluyen que el comportamiento de los GAs empeora cuando resuelven las instancias de mayor tamaño. En [120], Luque, Alba y Khuri amplían el trabajo el trabajo realizado en [113] al distribuir y paralelizar los GAs empleados en este último. De esta forma, logran incrementar la calidad de las soluciones para las instancias con 77Kbp.

Cotta et al., en [43], resuelven FAP usando tres metaheurísticas basadas en población (GAs, búsqueda dispersa -*Scatter Search*, SS- y CHC) y una basada en trayectoria, (enfriamiento simulado -*Simulated Annealing*, SA-). A partir de los experimentos realizados, los autores encuentran que el comportamiento de SA es superior al de las técnicas basadas en población.

Una alternativa a los algoritmos genéticos son los GAs celulares (cGAs), Alba y Dorronsoro en [6] proponen la combinación de cGA con PALS (cGA+PALS) para dar solución a instancias de FAP con 77Kbp. Aunque este algoritmo híbrido muestra un mejor compor-

tamiento que PALS (un método de búsqueda local especialmente diseñado para resolver FAP) incrementa considerablemente el tiempo de ejecución con respecto a este último. Otra propuesta basada en cGAs es el algoritmo SACMA desarrollado por Dorronsoro et al. en [55]. SACMA es un cGA que adapta el formato de la población dependiendo de la velocidad de convergencia. Este algoritmo incrementa la calidad de los resultados obtenidos por cGA+PALS pero también el tiempo de ejecución.

En [122], Meksangsouy y Chaiyaratana proponen un algoritmo basado en la optimización basada en colonia de hormigas para resolver instancias de FAP con hasta 36Kbp. Desafortunadamente, este algoritmo no puede resolver eficientemente aquellas instancias que cuentan con fragmentos de gran tamaño.

A pesar de la intensa investigación realizada en este campo, la precisión, la eficiencia y la resolución de instancias de gran tamaño (100Kbp o más) son temas de investigación abiertos en lo que respecta a la resolución de FAP mediante el uso de metaheurísticas. Por eso, en este trabajo se proponen mecanismos para incrementar la precisión y eficiencia de estas técnicas como son: las estrategias de inicio heurísticas, la incorporación de diferentes mecanismos de generación de soluciones vecinas, la hibridación de técnicas que hasta el momento han aportado buenos resultados (por ejemplo: GAs y SA) y la utilización de modelos distribuidos y paralelos. Además estas nuevas propuestas son aplicadas exitosamente a instancias con más de 400Kbp que, por otro lado, presentan una complejidad extra, ruido en los datos.

2.4. Conclusiones

En este capítulo se han revisado las distintas problemáticas presentes en el campo de la Bioinformática. El tratamiento se ha centrado en los problemas de secuenciamiento que conforman los fundamentos de cualquier proyecto genómico. El objetivo ha sido introducir el problema de ensamblado de fragmentos de ADN, abordado en esta tesis, así como dar una breve revisión de los ensambladores metaheurísticos que lo resuelven.

Capítulo 3

Algoritmos ensambladores

El conjunto de técnicas bioinformáticas utilizadas en las distintas áreas de la Biología Molecular, en particular en el ensamblado de fragmentos de ADN, es extenso y de componentes heterogéneos. Es posible distinguir dos grandes grupos de técnicas algorítmicas. El primero está formado por un grupo de técnicas modernas de uso generalizado, denominadas metaheurísticas. Por otro lado, el segundo subconjunto está conformado por algoritmos especialmente diseñados para un uso bioinformático específico, por ejemplo para alinear un par de secuencias de ADN.

En el primer caso, las técnicas metaheurísticas, han sido intensamente usadas en diferentes campos y se han adaptado a muchos usos bioinformáticos. La razón es que pueden resolver problemas de grandes dimensiones con fuertes restricciones de manera eficiente. Ejemplos de esto son: los algoritmos evolutivos, los métodos de Montecarlo guiados, la optimización basada en colonias de hormigas, los algoritmos meméticos, la optimización basada en cúmulo de partículas, entre otras. Todas ellas son metaheurísticas con estructura interna y búsqueda guiada no exhaustiva que se usan en diferentes campos, tales como: sistemas industriales, diseño en ingeniería, logística, comunicaciones, etc.

Aunque este tipo de algoritmos tiene un uso más generalizado, resultan eficaces [1] y eficientes [2] cuando la complejidad del problema y su respectivo espacio de soluciones son extensos o crecen continuamente. Esta es una característica muy importante y extrema-

[1] Alcanzan el resultado esperado (óptimo o quasi-óptimo).

[2] Obtienen el resultado esperado o los más cercanos a él con el mínimo uso de recursos computacionales.

damente necesaria a la hora de manipular enormes cantidades de información biológica. Además, dichas técnicas no necesitan datos precisos y completos para hallar soluciones de muy buena calidad. Por otro lado, estas técnicas inteligentes no son exhaustivas ni deterministas. Estas características reducen considerablemente el esfuerzo computacional empleado y pueden producir resultados múltiples para una misma situación. Además estas metaheurísticas presentan otra ventaja significativa, en el área de la Bioinformática, ya que resultan sumamente eficientes en la resolución de problemas de optimización combinatoria; como por ejemplo los problemas de: alineamiento de secuencias, ensamblado de fragmentos, análisis proteínico, entre muchos otros. Las características y ventajas mencionadas anteriormente son difíciles de encontrar o de incorporar en las técnicas del segundo grupo.

En el segundo caso los algoritmos han sido diseñados y modelados específicamente para manejar información biológica. Es el caso de herramientas como: CAP3 (ensambla fragmentos de ADN) [93], CLUSTALW-pairwise (compara un par de secuencias de ADN) [181], CLUSTAL-MSA (compara múltiples secuencias de ADN) [181], FASTA (permite identificar proteínas relacionadas entre sí) [139, 140], BLAST y sus variantes (conjunto de herramientas de búsqueda local para alinear secuencias) [11, 12], Vector de momentos de composición (permite predecir la estructura secundaria de la proteína) [153], Modelado de la dinámica molecular (identifica más de un tipo de estructura proteíca) [198], IRAP (determina cómo interactúan las proteínas entre sí, acoplamiento de proteínas [36], entre otras.

3.1. Metaheurísticas

La obtención de soluciones óptimas para muchos problemas de optimización, en el campo industrial y científico es intratable. Esto significa que un *método exacto* necesita un tiempo no polinomial para garantizar la optimalidad de la solución. Pero, para esta clase de problemas, también denominados NP-duros, se requiere de métodos que generen soluciones de alta calidad en un tiempo razonable para su uso aunque no se garantice encontrar una solución óptima global. A estos últimos se los denomina *métodos aproximados* o *heurísticos*.

Dentro de los métodos aproximados se pueden encontrar, tres clases de algoritmos: *algoritmos de aproximación* propiamente dichos, los *algoritmos heurísticos ad-hoc* y los *al-*

goritmos metaheurísticos. Los primeros proveen calidad de solución y límites en el tiempo de ejecución demostrables. Los segundos, los algoritmos heurísticos ad-hoc, son hechos a medida y diseñados para resolver un problema y/o instancia específica. En tanto que, los algoritmos metaheurísticos son técnicas de propósito general que pueden utilizarse en la mayoría de los problemas de optimización. En general, las últimas dos clases de algoritmos permiten hallar soluciones de alta calidad para instancias de gran tamaño de un problema con un desempeño aceptable.

Las metaheurísticas surgen en los años setenta como una nueva clase de algoritmos no deterministas. Se conoce como no deterministas a aquellos algoritmos que pueden obtener diferentes resultados a partir de los mismos datos de entrada. El propósito de las metaheurísticas es combinar diferentes métodos heurísticos a un nivel más alto y así explorar el espacio de búsqueda de forma eficiente y efectiva. Pero no es sino hasta el año 1986 que Glover acuña este término [71]. Antes de que la comunidad científica aceptara completamente el término metaheurística, a estos métodos se los conoce como heurísticos modernos [148]. En la actualidad existe un número importante de algoritmos metaheurísticos, entre ellos se encuentran: la *optimización basada en colonias de hormigas* (*Ant Colony Optimization*, ACO), los *algoritmos evolutivos* (Evolutionary Algorithms, EAs), la *búsqueda local iterada* (*Iterated Local Search*, ILS), el *enfriamiento simulado* (*Simulated Annealing*, SA) y la *búsqueda tabú* (*Tabu Search*, TS). Se pueden encontrar diferentes revisiones de metaheurísticas en [5, 23, 73, 177]. Al sintetizar las distintas nociones de metaheurísticas dadas en la literatura es posible extraer un conjunto de propiedades fundamentales que caracterizan a esta clase de métodos:

- Las metaheurísticas son estrategias o plantillas generales que "guían" el proceso de búsqueda.
- El objetivo es una exploración del espacio de búsqueda eficiente para encontrar soluciones (casi) óptimas.
- Las metaheurísticas son métodos inteligentes no exhaustivos ni deterministas.
- Pueden incorporar mecanismos para evitar las áreas del espacio de búsqueda no óptimas.

- El esquema básico de cualquier metaheurística es general y no depende del problema a resolver, por ende no necesitan contar con datos precisos y completos para obtener más información (o soluciones) de muy buena calidad.

- Las metaheurísticas hacen uso del conocimiento del problema que se trata de resolver en forma de operaciones heurísticas específicas que son controladas por una estrategia de más alto nivel.

Resumiendo esos puntos, se logra acordar que una metaheurística es una estrategia de alto nivel que usa diferentes métodos para explorar el espacio de búsqueda. En otras palabras, las metaheurísticas son métodos algorítmicos con estructura interna y búsqueda guiada no exhaustiva que se usan en diferentes campos, tales como: sistemas industriales, diseño en ingeniería, logística, comunicaciones, etc. Aunque este tipo de algoritmos tiene un uso más generalizado, resultan eficaces y eficientes cuando la complejidad del problema y su respectivo espacio de soluciones son extensos o crecen continuamente. Esta es una característica muy importante y extremadamente necesaria a la hora de manipular enormes cantidades de información biológica. Además las metaheurísticas presentan otra ventaja significativa en el área de la Bioinformática, ya que resultan sumamente eficientes en la resolución de problemas de optimización combinatoria; como es el caso del problema de ensamblado de fragmentos.

Esta sección está dedicada a establecer los fundamentos necesarios sobre los algoritmos de optimización utilizados para abordar el problema de ensamblado de fragmentos presentado en el capítulo anterior. Se realiza un planteamiento clásico de optimización para definir el concepto de metaheurística y establecer su clasificación. A continuación se introducen conceptos de optimización

3.1.1. Definición formal

La optimización en el sentido de encontrar la mejor solución, o al menos una solución lo suficientemente buena, para un problema es un campo de vital importancia en el mundo real y, en particular, en Bioinformática. Constantemente se están resolviendo pequeños problemas de optimización, como el camino más corto para ir de un lugar a otro, la organización

de una agenda, etc. En general, estos problemas son lo suficientemente pequeños y puede resolverse sin ayuda adicional, pero conforme se hacen más grandes y complejos, el uso de los ordenadores para su resolución es inevitable. Se comienza definiendo formalmente el concepto de optimización. Asumiendo, sin pérdida de generalidad, el caso de la minimización, es posible definir un problema de optimización como sigue:

Definición 3.1.1. Problema de optimización. Un problema de optimización se formaliza como un par (S, f), donde $S \neq \emptyset$ representa el espacio de soluciones (o de búsqueda) del problema, mientras que f es una función denominada función objetivo o función de *fitness*, que se define como:

$$f : S \rightarrow \mathbb{R}. \tag{3.1}$$

Así, resolver un problema de optimización consiste en encontrar una solución, $i^* \in S$, que satisfaga la siguiente desigualdad:

$$f(i^*) \leq f(i), \forall i \in S. \tag{3.2}$$

Asumir el caso de maximización o minimización no restringe la generalidad de los resultados, puesto que se puede establecer una igualdad entre tipos de problemas de maximización y minimización de la siguiente forma [17, 77]:

$$max\{f(i)|i \in S\} \equiv min\{-f(i)|i \in S\} \tag{3.3}$$

En función del dominio al que pertenezca S, se pueden definir problemas de *optimización binaria* ($S \subseteq \mathbb{B}^*$), *entera* ($S \subseteq \mathbb{N}^*$), *continua*($S \subseteq \mathbb{R}^*$), o *heterogénea* ($S \subseteq (\mathbb{B} \cup \mathbb{N} \cup \mathbb{R})^*$).

Una metaheurística se define formalmente como una tupla de elementos que, dependiendo de cómo se especifiquen, dan lugar a una técnica concreta u otra. Esta definición formal ha sido desarrollada en [118] y posteriormente extendida en [40].

Definición 3.1.2. Metaheurística. Una metaheurística $\mathcal{M}$ es una tupla formada por los siguientes ocho componentes:

$$\mathcal{M} = \langle \Gamma, \Xi, \mu, \lambda, \Phi, \sigma, \mathcal{U}, \tau \rangle, \tag{3.4}$$

donde:

- $\Gamma = \{x_1, x_2, ..., x_n\}$, es el conjunto de soluciones que manipula la metaheurística. Este conjunto contiene al espacio de búsqueda y en la mayoría de los casos coincide con él.
- $\Xi = \{(\xi_1, D_1), (\xi_2, D_2), ..., (\xi_v, D_v)\}$, es un conjunto de v pares. Cada par está formado por una variable de estado de la metaheurística y el dominio de dicha variable.
- $\mu \in \mathbb{N}$, es el número de soluciones con las que trabaja $\mathcal{M}$ en un paso.
- $\lambda \in \mathbb{N}$, es el número de nuevas soluciones generadas en cada iteración de $\mathcal{M}$.
- $\Phi : \Gamma^\mu \times \prod_{i=1}^{v} D_i \times \Gamma^\lambda \to [0, 1]$ representa el operador que genera nuevas soluciones a partir de las existentes. Esta función debe cumplir para todo $x \in \Gamma^\mu$ y para todo $t \in \prod_{i=1}^{v} D_i$,

$$\sum_{y \in \Gamma^\lambda} \Phi(x, t, y) = 1. \tag{3.5}$$

- $\sigma : \Gamma^\mu \times \Gamma^\lambda \times \prod_{i=1}^{v} D_i \times \Gamma^\mu \to [0, 1]$ es una función que permite seleccionar las soluciones que serán manipuladas en la siguiente iteración de $\mathcal{M}$. Esta función debe cumplir para todo $x \in \Gamma^\mu, z \in \Gamma^\lambda$ y $t \in \prod_{i=1}^{v} D_i$,

$$\sum_{y \in \Gamma^\mu} \sigma(x, z, t, y) = 1 \tag{3.6}$$

$$\forall y \in \Gamma^\mu, \sigma(x, z, t, y) = 0 \quad \vee \sigma(x, z, t, y) > 0 \quad \wedge \tag{3.7}$$

$$\wedge (\forall i \in \{1, ..., \mu\} \quad (\exists j \in \{1, ..., \mu\}, y_i = x_j) \quad \vee \quad (\exists j \in \{1, ..., \lambda\}, y_i = z_j)).$$

- $\mathcal{U} : \Gamma^\mu \times \Gamma^\lambda \times \prod_{i=1}^{v} D_i \times \prod_{i=1}^{v} D_i \to [0, 1]$ representa el procedimiento de actualización de las variables de estado de la metaheurística. Esta función debe cumplir para todo $x \in \Gamma^\mu, z \in \Gamma^\lambda$ y $t \in \prod_{i=1}^{v} D_i$,

$$\sum_{u \in \prod_{i=1}^{v} D_i} \mathcal{U}(x, z, t, u) = 1 \tag{3.8}$$

- $\tau : \Gamma^{\mu} \times \prod_{i=1}^{v} D_i \rightarrow \{falso, verdadero\}$ es una función que decide la terminación del algoritmo.

La definición anterior considera el comportamiento estocástico típico de las técnicas metaheurísticas. En concreto, las funciones Φ, σ y $\mathcal{U}$ deben interpretarse como probabilidades condicionadas. Por ejemplo, el valor de $\Phi(x, t, y)$ se interpreta como la probabilidad de que se genere un nuevo vector de soluciones $y \in \Gamma^{\lambda}$ dado que actualmente el conjunto de soluciones con el que la metaheurística trabaja es $x \in \Gamma^{\mu}$ y su estado interno viene definido por las variables de estado $t \in \prod_{i=1}^{v} D_i$. Puede observarse que las restricciones que se le imponen a las funciones Φ, σ y $\mathcal{U}$ permite considerarlas como funciones que devuelven estas probabilidades condicionadas.

Definición 3.1.3. Estado de una metaheurística. Sea $\mathcal{M} = \langle \Gamma, \Xi, \mu, \lambda, \Phi, \sigma, \mathcal{U}, \tau \rangle$ una metaheurística y $\Theta = \{\theta_1, \theta_2, ..., \theta_{\mu}\}$ el conjunto de variables que almacenarán las soluciones con las que trabaja la metaheurística. Se utilizará la notación $first(\Xi)$ para referirse al conjunto de las variables de estado de la metaheurística, $\{\xi_1, \xi_2, ..., \xi_{\mu}\}$. Un estado s de la metaheurística es un par de funciones $s = (s_1, s_2)$ con

$$s_1 : \Theta \rightarrow \Gamma, \tag{3.9}$$

$$s_2 : first(\Xi) \rightarrow \bigcup_{i=1}^{v} D_i \tag{3.10}$$

donde s_2 cumple

$$s_2(\xi_i) \in D_i \forall \xi_i \in first(\Xi). \tag{3.11}$$

Se denotará con $\mathcal{S}_{\mathcal{M}}$ el conjunto de todos los estados de una metaheurística $\mathcal{M}$.

Con la definición anterior es posible representar todas las variables intervinientes en un momento dado de la ejecución de una metaheurísticas y conocer su contenido. s_1 brinda datos completos sobre la codificación y decodificación de las soluciones utilizadas hasta ese momento. En tanto que s_2 representa los datos de variables de estado, como pueden ser el número de iteración actual, el error promedio, la mejor solución encontrada, etc. Por último, una vez definido el estado de la metaheurística, es posible definir su dinámica.

Definición 3.1.4. Dinámica de una metaheurística. Sea $\mathcal{M} = \langle \Gamma, \Xi, \mu, \lambda, \Phi, \sigma, \mathcal{U}, \tau \rangle$ una metaheurística y $\Theta = \{\theta_1, \theta_2, ..., \theta_\mu\}$ el conjunto de variables que almacenarán las soluciones con las que trabaja la metaheurística. Se denotará $\bar{\Theta}$ a la tupla $(\theta_1, \theta_2, ..., \theta_\mu)$ y con $\bar{\Xi}$ a la tupla $(\xi_1, \xi_2, ..., \xi_v)$. Se extenderá la definición de estado para que pueda aplicarse a tuplas de elementos, esto es, definimos $\bar{s} = (\bar{s_1}, \bar{s_2})$ donde

$$\bar{s_1} : \Theta^n \to \Gamma^n, \tag{3.12}$$

$$\bar{s_2} : first(\Xi)^n \to (\bigcup_{i=1}^{v} D_i)^n, \tag{3.13}$$

y además

$$\bar{s_1}(\theta_{i_1}, \theta_{i_2}, ..., \theta_{i_n}) = (s_1(\theta_{i_1}), s_2(\theta_{i_2}), ..., s_n(\theta_{i_n})), \tag{3.14}$$

$$\bar{s_2}(\xi_{i_1}, \xi_{i_2}, ..., \xi_{i_n}) = (s_1(\xi_{i_1}), s_2(\xi_{i_2}), ..., s_n(\xi_{i_n})), \tag{3.15}$$

para $n \geq 2$. Se llamará r a un estado sucesor de s si existe $t \in \Gamma^\lambda$ tal que $\Phi(\bar{s_1}(\bar{\Theta}), \bar{s_2}(\bar{\Xi}), t) > 0$ y además

$$\sigma(\bar{s_1}(\bar{\Theta}), t, \bar{s_2}(\bar{\Xi}), \bar{r_1}(\bar{\Theta})) > 0 \;\; y \tag{3.16}$$

$$\mathcal{U}(\bar{s_1}(\bar{\Theta}), t, \bar{s_2}(\bar{\Xi}), \bar{r_2}(\bar{\Theta})) > 0. \tag{3.17}$$

Se denotará con $\mathcal{F}_\mathcal{M}$ la relación binaria "ser sucesor de" definida en el conjunto de estados de una metaheurística $\mathcal{M}$. Es decir, $\mathcal{F}_\mathcal{M} \subseteq \mathcal{S}_\mathcal{M} \times \mathcal{S}_\mathcal{M}$ y $\mathcal{F}_\mathcal{M}(s, r)$ si r es un estado sucesor de s.

La definición de dinámica de una metaheurística formaliza la transición de estados durante la ejecución. Un cambio de estado se produce en $\bar{s_1}$ con la generación de una o varias soluciones, a partir de la o las actuales, por medio de la aplicación del operador Φ, seguida de la aceptación y/o rechazo de la o las soluciones recientemente generadas para formar parte de la próxima iteración al aplicarse la función σ. Esto genera la necesidad de actualizar las variables de estado, $\bar{s_2}$, para lo que se usa el procedimiento $\mathcal{U}$. Utilizando entonces, la definición 3.1.4 es posible formalizar toda una ejecución de una metaheurística, como se muestra a continuación.

Definición 3.1.5. Ejecución de una metaheurística. Una ejecución de una metaheurística $\mathcal{M}$ es una secuencia finita o infinita de estados, $s_0, s_1, ...$ en la que $\mathcal{F}_{\mathcal{M}}(s_i, s_{i+1})\ \forall i \geq 0$ y además:

- si la secuencia es infinita se cumple $\tau(s_i(\bar{\Theta}), s_i(\bar{\Xi})) = falso\ \forall i \geq 0$ y
- si la secuencia es finita se cumple $\tau(s_k(\bar{\Theta}), s_k(\bar{\Xi})) = verdadero$ para el último estado s_k, y además, $\tau(s_i(\bar{\Theta}), s_i(\bar{\Xi})) = falso\ \forall i \geq 0$ tal que $i < k$.

En las próximas secciones es posible comprobar cómo esta formulación general se puede adaptar a las técnicas concretas (obviando aquellos parámetros no fijados por la metaheurística o que dependen de otros aspectos como el problema o la implementación concreta).

3.1.2. Principales conceptos en común de las metaheurísticas

Como se ha mencionado anteriormente una metaheurística es una plantilla general no determinista que debe rellenarse con datos específicos del problema (representación de las soluciones, operadores para manipularlas, función de costo, etc.), que permiten abordar problemas con espacios de representación (o de búsqueda) de gran tamaño. Por lo tanto es de especial interés el correcto equilibrio (generalmente dinámico) entre *diversificación* (o exploración) e *intensificación* (o explotación).

El término diversificación alude a la exploración del espacio de representación (Γ), mientras que intensificación se refiere a la explotación de algún área concreta de ese espacio. El equilibrio entre estos dos aspectos contrapuestos es de gran importancia, ya que por un lado deben identificarse rápidamente las regiones prometedoras del espacio de búsqueda global y por el otro lado no se debe malgastar tiempo en las regiones que ya han sido exploradas o que no contienen soluciones de alta calidad.

Por otra parte, existen dos puntos de diseño relacionados con todas las metaheurísticas: la representación de las soluciones manipuladas por los algoritmos y la definición de la función objetivo que guía la búsqueda. Ambas cuestiones de diseño son explicadas en las siguientes subsecciones.

3.1.2.1. Representación

Para diseñar una metaheurística es fundamental codificar o representar la solución. La codificación juega un rol importante en la eficiencia y eficacia de cualquier metaheurística y constituye un paso esencial en su diseño. La codificación debe ser apta y relevante para el problema de optimización que se enfrenta. Es más, la eficiencia de una representación está relacionada con los operadores de búsqueda aplicados sobre dicha representación (vecindario, recombinación, etc.). De hecho, al diseñar una representación es necesario considerar cómo se evaluará y cómo operarán los operadores de búsqueda.

Para un problema dado pueden existir muchas alternativas de representación. Pero es conveniente que una representación cumpla con las siguientes características:

- **Completitud.** Esto significa que debe ser posible representar todas las soluciones asociadas al problema.
- **Alcanzabilidad.** Esta característica indica que debe existir al menos un camino de búsqueda entre dos soluciones cualesquiera. En otras palabras, tiene que ser posible alcanzar cualquier solución del espacio de búsqueda, especialmente el óptimo global.
- **Eficiencia.** Esta última característica implica que el operador de búsqueda pueda manipular fácilmente a la solución candidata. De esta forma se reducirían las complejidades espaciales y temporales de los operadores aplicados a la representación.

De acuerdo a su estructura las representaciones pueden clasificarse en: lineales o no lineales. Las **representaciones lineales** pueden ser vistas como cadenas de símbolos de un alfabeto dado. En muchos problemas de optimización clásicos, tales como problemas de satisfacción de restricciones o programas lineales $-0, 1$, donde la variable de decisión denota la presencia o ausencia de un elemento o una decisión si/no, puede usarse una *codificación binaria*. Dicha codificación consiste en asociar un valor binario a cada variable de decisión, entonces, una solución se representa por medio de un vector de variables binarias.

La codificación binaria usa un alfabeto binario que consiste en dos símbolos diferentes, que puede generalizarse a una codificación basada en *valores discretos* de un alfabeto n-ario. En este caso, cada variable toma su valor de un alfabeto n-ario y la codificación es un

vector de valores discretos. Este tipo de representación puede usarse en problemas donde las variables toman un número finito de valores, tales como los problemas de *optimización combinatoria*.

Muchos de los problemas de encaminamiento, planificación y secuenciamiento son considerados como problemas de *permutación*, tales como el problema del viajante de comercio o el problema de ensamblado de fragmentos de ADN, se pueden representar por medio de una permutación $\pi = (\pi_1, \pi_2, ..., \pi_n)$. Cualquier elemento del problema (ciudades para el TSP o fragmentos para el FAP) deben aparecer sólo una vez en la representación.

Para problemas de *optimización continua*, la codificación natural se basa en *valores reales*. Este tipo de representación es comúnmente usada para problemas de *optimización continua no lineal*, donde la codificación se basa en vectores de valores reales.

Las **representaciones no lineales** usan en general estructuras de datos más complejas, tales como grafos. Particularmente, la estructura de árbol es la más ampliamente usada. Esta última se usa principalmente para codificar las soluciones de problemas de optimización estructurados jerárquicamente, como es el caso de la programación genética. Un árbol puede representar una expresión aritmética, una fórmula lógica de primer orden o un programa. Otras representaciones no lineales se basan en máquinas de estado finito y en grafos.

Durante el diseño de una representación también es importante considerar los puntos críticos que surgen entre la codificación y decodificación de las soluciones. Ellos son:

- *La factibilidad de la solución decodificada.* Se refiere al fenómeno en que una solución decodificada se encuentre en la región factible de un problema dado. La no factibilidad proviene de la naturaleza de los problemas de optimización con restricciones, donde todos los métodos deben manejar restricciones. En muchos problemas de optimización, las regiones factibles pueden ser representadas como un sistema de ecuaciones o inecuaciones (lineales o no).

- *La legalidad de la solución codificada.* Se refiere al fenómeno en que una codificación represente una solución a un problema dado. La ilegalidad del cromosoma se origina en la naturaleza de la técnica de codificación. Para muchos problemas de optimización combinatoria se utilizan codificaciones específicas al problema, que al aplicarse

conjuntamente con operadores no diseñados para tal representación producen soluciones ilegales. Es decir, a partir de una codificación ilegal no se puede decodificar una solución para el problema y por ende, no es posible evaluarla.

- *La unicidad de la decodificación.* La función de traducción entre la solución codificada y la decodificada puede pertenecer a uno de los siguientes casos:

  1. Traducción 1 a 1. Corresponde a la clase tradicional de representación y es la ideal. Aquí cada codificación representa a una única solución, y cada solución es representada por una única codificación. Por lo tanto no hay redundancia ni reducción en el espacio de búsqueda.
  2. Traducción n a 1. En este caso varias codificaciones pueden representar a una solución. Esta redundancia en la codificación aumenta el tamaño del espacio de búsqueda pudiendo impactar negativamente en la efectividad de la metaheurística.
  3. Traducción 1 a n (o *codificación indirecta*). En esta clase una sola codificación puede representar varias soluciones. Generalmente en estos casos se observa falta de detalles en la codificación, algunos datos sobre la solución no están explícitamente representados. Esto reduce el espacio de búsqueda mejorando la eficiencia de la metaheurística.

  Es necesario considerar estos problemas cuidadosamente cuando se diseña una nueva codificación no binaria para construir un algoritmo metaheurístico efectivo.

3.1.2.2. Función objetivo

La *función objetivo*, también denominada *función de costo*, *función de evaluación* y *función de utilidad*, f, formula el objetivo a alcanzar. Cuando la función objetivo asocia a cada solución del espacio de búsqueda un valor real que describe la calidad o aptitud de la solución, $f : S \to \mathbb{R}$, entonces se trata de la (*función de fitness*). Esta función representa un valor absoluto y permite un ordenamiento de todas las soluciones del espacio de búsqueda.

La función objetivo es un elemento importante en el diseño de una metaheurística, ya que ésta guiará la búsqueda hacia buenas soluciones del espacio de búsqueda. Si la fun-

ción objetivo no está definida apropiadamente, puede guiar la búsqueda hacia soluciones inaceptables.

Para algunos problemas de optimización la definición de la función objetivo es simple, ya que especifica la función objetivo formulada originalmente. Ejemplo de esto son los problemas de encaminamiento vehicular, del viajante de comercio y diferentes problemas de optimización continua.

En otros casos, la definición de esta función es una tarea difícil y constituye un punto crucial, ya que se transforma para lograr una mejor convergencia de la metaheurística. Es decir, la nueva función objetivo guiará la búsqueda de manera más eficiente. Este es el caso de los problemas de satisfacción de restricciones.

El diseño de la representación y de la función objetivo pueden estar relacionados, dado que en algunos problemas la representación (genotipo para los EAs) se decodifica para generar la mejor solución (fenotipo para los EAs) posible. En esta situación la correspondencia entre la representación y la función objetivo no es simple; dado que se necesita especificar una función decodificadora para generar la mejor solución desde una representación dada de acuerdo a la función objetivo.

3.1.3. Clasificación de las metaheurísticas

Hay diferentes formas de clasificar y describir las técnicas metaheurísticas [33, 46]. Dependiendo de las características que se seleccionen se pueden obtener diferentes taxonomías: basadas en la naturaleza y no basadas en la naturaleza, con memoria o sin ella, con una o varias estructuras de vecindario, etc. Una de las clasificaciones más populares las divide en *metaheurísticas basadas en trayectoria* y *basadas en población*. Las primeras manipulan en cada paso un único elemento del espacio de búsqueda, mientras que las segundas trabajan sobre un conjunto de ellos (población). Esta taxonomía se muestra de forma gráfica en la figura 3.1, que además incluye las principales metaheurísticas.

3.1.3.1. Metaheurísticas basadas en trayectoria

En esta sección repasaremos brevemente algunas metaheurísticas basadas en trayectoria. La principal característica de estos métodos es que parten de una solución y, mediante la

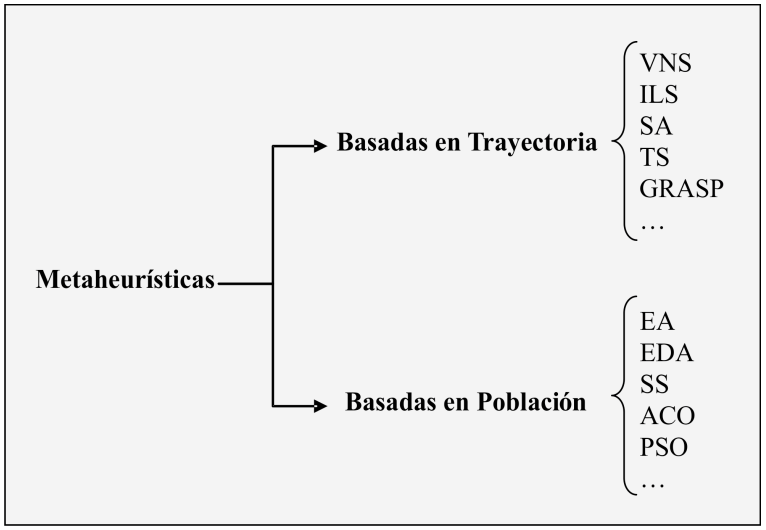


Figura 3.1: Clasificación de las metaheurísticas.

exploración del vecindario, van actualizando la solución actual, formando una trayectoria. Según la notación de la Definición 3.1.2, esto se formaliza con $\mu = 1$. La mayoría de estos algoritmos surgen como extensiones de los métodos de *búsqueda local* (*Local Search*, LS) simples a los que se les añade algún mecanismo para escapar de los mínimos locales. Esto implica la necesidad de una condición de parada (función τ) más elaborada que la de encontrar un mínimo local. Normalmente la búsqueda termina cuando se alcanza un número máximo predefinido de iteraciones, se encuentra una solución con una calidad aceptable, o se detecta un estancamiento del proceso. Por otra parte, las características más importantes, en esta clase de metaheurísticas radica en la definición de la función utilizada para seleccionar las soluciones que serán manipuladas en la siguiente iteración (función σ) y en la actualización de las variables de estado (procedimiento $\mathcal{U}$). Como puede observarse en el enfriamiento simulado y en la búsqueda tabú.

Enfriamiento simulado (SA) El enfriamiento simulado es una de las técnicas más antiguas entre las metaheurísticas y posiblemente es el primer algoritmo con una estrategia explícita para escapar de los mínimos locales [104]. Los orígenes del algoritmo se encuentran en un mecanismo estadístico, denominado Metropolis [123]. La idea del SA es simular el proceso de enfriamiento del metal y del cristal. Para evitar quedar atrapado en un mínimo local, es posible elegir una solución cuyo valor de *fitness* sea peor que el de la solución

actual. Por lo que la función σ opera de la siguiente manera: en cada iteración se elige, a partir de la solución actual s, una solución s_0 del vecindario $N(s)$. Si s_0 es mejor que s (es decir, tiene un mejor valor en la función de *fitness*), se sustituye s por s_0 como solución actual. Si la solución s_0 es peor, entonces es aceptada con una determinada probabilidad que depende de la temperatura actual T y de la diferencia de *fitness* entre ambas soluciones, $f(s_0) - f(s)$ (caso de minimización). En tanto que el procedimiento $\mathcal{U}$ está relacionado con la actualización de la temperatura, lo que se explica detalladamente en el capítulo 4.

Búsqueda tabú (TS) La búsqueda tabú es una de las metaheurísticas que se han aplicado con más éxito a la hora de resolver problemas de optimización combinatoria. Los fundamentos de este método fueron introducidos en [71], y están basados en las ideas formuladas en [70]. Un buen resumen de esta técnica y sus componentes se puede encontrar en [74]. La idea básica de la búsqueda tabú es el uso explícito de un historial de la búsqueda (una memoria a corto plazo), tanto para escapar de los mínimos locales como para implementar su estrategia de exploración y evitar buscar varias veces en la misma región. Esta memoria a corto plazo se implementa con una *lista tabú*, donde se mantienen las soluciones visitadas más recientemente para excluirlas de los próximos movimientos. En cada iteración se elige (función σ) la mejor solución entre las permitidas y la solución es añadida a la lista tabú. Desde el punto de vista de la implementación, mantener una lista de soluciones completas no suele ser práctico debido a su ineficiencia. Por lo tanto, en general, se suelen almacenar los movimientos que han llevado al algoritmo a generar esa solución o los componentes principales que definen la solución (procedimiento $\mathcal{U}$). En cualquier caso, los elementos de esta lista permiten filtrar el vecindario, generando un conjunto reducido de soluciones elegibles denominado $N_a(s)$. El almacenamiento de los movimientos en vez de las soluciones completas es bastante más eficiente, pero introduce una pérdida de información. Para evitar este problema, se define *un criterio de aspiración* que permite realizar movimientos incluso si está prohibido debido a la lista tabú. El criterio de aspiración más ampliamente permite realizar movimientos tabúes que generen soluciones cuyo *fitness* sea mejor que el de la mejor solución encontrada hasta el momento.

Procedimiento de búsqueda voraz aleatorio y adaptativo (GRASP) El procedimiento de búsqueda voraz aleatorio y adaptativo (*Greedy Randomized Adaptive Search Procedure*) [60] es una metaheurística simple que combina heurísticos constructivos con búsqueda local. GRASP es un procedimiento iterativo, compuesto de dos fases: primero la construcción de una solución y después un proceso de mejora. La solución mejorada es el resultado del proceso de búsqueda. El mecanismo de construcción de soluciones es un heurístico constructivo aleatorio. Va añadiendo paso a paso diferentes componentes c a la solución parcial s^p, que inicialmente está vacía. Los componentes que se añaden en cada paso son elegidos aleatoriamente de una lista restringida de candidatos (RCL). Esta lista es un subconjunto de $N(s^p)$, el conjunto de componentes permitidos para la solución parcial s^p. Para generar esta lista, los componentes de la solución en $N(s^p)$ se ordenan de acuerdo a alguna función dependiente del problema (η). La lista RCL está compuesta por los α mejores componentes de ese conjunto. En el caso extremo de $\alpha = 1$, siempre se añade el mejor componente encontrado de manera determinista, con lo que el método de construcción es equivalente a un algoritmo voraz. En el otro extremo, con $\alpha = |N(s^p)|$ el componente a añadir se elige de forma totalmente aleatoria de entre todos los disponibles. Por lo tanto, α es un parámetro clave que incluye en cómo se va a muestrear el espacio de búsqueda. La segunda fase del algoritmo consiste en aplicar un método de búsqueda local para mejorar la solución generada. Este mecanismo de mejora puede ser una técnica de mejora simple o algoritmos más complejos como SA o TS.

Búsqueda con vecindario variable (VNS) La búsqueda con vecindario variable (*Variable Neighborhood Search*, VNS) es una metaheurística propuesta en [131] que aplica explícitamente una estrategia para cambiar entre diferentes vecindarios durante la búsqueda. Este algoritmo es muy general y con muchos grados de libertad a la hora de diseñar variaciones e instanciaciones particulares. El primer paso a realizar es definir un conjunto de vecindarios. Esta elección puede hacerse de muchas formas: desde ser elegidos aleatoriamente hasta utilizar complejas ecuaciones deducidas del problema. Cada iteración consiste en tres fases: la elección del candidato, una fase de mejora y, finalmente, el movimiento. En la primera fase, se elige aleatoriamente un vecino s' de s usando el $k - ésimo$ vecindario. Esta solución s' es utilizada como punto de partida de la búsqueda local de la segunda fase.

Cuando termina el proceso de mejora, se compara la nueva solución s'' con la original s. Si es mejor, s'' se convierte en la solución actual y se inicializa el contador de vecindarios ($k \leftarrow 1$); si no es mejor, se repite el proceso pero utilizando el siguiente vecindario ($k \leftarrow k + 1$). La búsqueda local es el paso de intensificación del método y el cambio de vecindario puede considerarse como el paso de diversificación. Como puede observarse, VNS también proporciona un marco de búsqueda genérico donde distintas metaheurísticas pueden ser elegidas para la segunda fase, pero su principal característica radica en la forma en que explora los distintos vecindarios pertenecientes a Γ^{μ}.

Búsqueda local iterada (ILS) La búsqueda local iterada [117, 171] es una metaheurística basada en un concepto simple pero muy efectivo. En cada iteración, la solución actual es perturbada y a esta nueva solución se le aplica un método de búsqueda local para mejorarla. Este nuevo mínimo local obtenido por el método de mejora puede ser aceptado como nueva solución actual si pasa un *test de aceptación*. La importancia del proceso de perturbación es obvia: si es demasiado pequeña puede que el algoritmo no sea capaz de escapar del mínimo local; por otro lado, si es demasiado grande, la perturbación puede hacer que el algoritmo sea como un método de búsqueda local con un reinicio aleatorio. Por lo tanto, el método de perturbación debe generar una nueva solución que sirva como inicio a la búsqueda local, pero que no debe estar muy lejos del actual para que no sea una solución aleatoria. El criterio de aceptación (o función σ) actúa como contra-balance, ya que filtra la aceptación de nuevas soluciones dependiendo de la historia de búsqueda y de las características del nuevo mínimo local.

3.1.3.2. Metaheurísticas basadas en población

Los métodos basados en población se caracterizan por trabajar con un conjunto de soluciones (denominado población) en cada iteración (es decir, generalmente $\mu > 1$ y/o $\lambda \geq 1$), a diferencia de los métodos basados en trayectoria, que únicamente manipulan una solución del espacio de búsqueda por iteración. Si bien la generación de nuevas soluciones a partir de las existentes (aplicación del operador Φ) es dependiente de la codificación y del problema, ésta es una de de las características diferenciadoras de esta clase de algoritmos. Ejemplo de esto son: los operadores de crossover y mutación en los algoritmos genéticos, el

método de combinación de soluciones en la búsqueda dispersa o el proceso de construcción de una solución influenciado por los rastros de feromona artificial en la optimización basada en colonia de hormigas.

Algoritmos evolutivos (EAs) Los algoritmos evolutivos están inspirados en la teoría de la evolución natural. Esta familia de técnicas sigue un proceso iterativo y estocástico que opera sobre una población de soluciones, denominadas en este contexto individuos. Inicialmente, la población es generada aleatoriamente (quizás con ayuda de un heurístico de construcción). El esquema general de un algoritmo evolutivo comprende tres fases principales: *selección*, *reproducción* (aplicación del operador Φ) y *reemplazo* (función σ). El proceso completo es repetido hasta que se cumpla un cierto criterio de terminación (aplicación de la función τ), normalmente después de un número dado de iteraciones. En la fase de selección se escogen generalmente los individuos más aptos de la población actual para ser posteriormente recombinados en la fase de reproducción. Los individuos resultantes de la recombinación se alteran mediante un operador de mutación. Finalmente, a partir de la población actual y/o los mejores individuos generados (de acuerdo a su valor de *fitness*) se forma la nueva población, dando paso a la siguiente generación del algoritmo.

Algoritmos de estimación de la distribución (EDAs) Los algoritmos de estimación de la distribución (*Estimation of Distribution Algorithms*, EDAs) [124] muestran un comportamiento similar a los algoritmos evolutivos presentados en la sección anterior y, de hecho, muchos autores consideran los EDA como otro tipo de EA. Los EDAs operan sobre una población de soluciones tentativas como los algoritmos evolutivos pero, a diferencia de estos últimos, que utilizan operadores de recombinación y mutación para mejorar las soluciones, los EDA infieren la *distribución de probabilidad* del conjunto seleccionado y, a partir de ésta, generan nuevas soluciones que formarán parte de la población. Es decir, se basan principalmente en sustituir los operadores genéticos por la estimación y posterior muestreo de una distribución de probabilidad aprendida a partir de los individuos seleccionados. Los modelos gráficos probabilísticos son herramientas comúnmente usadas en el contexto de los EDAs para representar eficientemente la distribución de probabilidad. Algunos autores [111, 141, 169] han propuesto las redes bayesianas para representar la distribución de pro-

babilidad en dominios discretos, mientras que las redes gaussianas se emplean usualmente en los dominios continuos [193].

Búsqueda dispersa (SS) La búsqueda dispersa (*Scatter Search*, SS) [72] es una metaheurística cuyos principios fueron presentados en [70] y que actualmente está recibiendo una gran atención por parte de la comunidad científica [78, 109, 110]. Esta metaheurística es un método evolutivo que, a diferencia de los algoritmos genéticos, no se basa en la aleatorización sobre un conjunto relativamente grande de soluciones sino en elecciones sistemáticas y estratégicas sobre un conjunto pequeño. Por lo que SS usa un subconjunto muy selectivo del espacio de soluciones Γ. El marco de la búsqueda dispersa es flexible, permitiendo el desarrollo de implementaciones alternativas con diferentes grados de sofisticación. Como ya se mencionó este algoritmo mantiene un conjunto relativamente pequeño de soluciones tentativas (llamado conjunto de referencia o *RefSet*) que se caracteriza por contener soluciones de calidad y diversas (distantes en el espacio de búsqueda). Para la definición completa de SS hay que concretar cinco componentes: *creación de la población inicial, generación del conjunto de referencia, generación de subconjuntos de soluciones, método de combinación de soluciones* y *método de mejora.*

Optimización basada en colonias de hormigas (ACO) Los algoritmos de optimización basados en colonias de hormigas son parte de la inteligencia basada en cúmulo de partículas (*swarm*). Esta clase de algoritmos están compuestos por individuos simples que cooperan a través de la auto-organización, es decir, sin ningún tipo de control central sobre los miembros del enjambre. En particular los algoritmos ACO [53, 54] están inspirados en el comportamiento de las hormigas reales cuando buscan comida. Este comportamiento es el siguiente: inicialmente, las hormigas exploran el área cercana a su nido de forma aleatoria. Tan pronto como una hormiga encuentra comida, la lleva al nido. Mientras que realiza este camino, la hormiga va depositando una sustancia química denominada feromona. Esta sustancia ayudará al resto de las hormigas a encontrar la comida. La comunicación indirecta entre las hormigas mediante el rastro de feromona las capacita para encontrar el camino más corto entre el nido y la comida. Este comportamiento es el que intenta simular este método para resolver problemas de optimización. La técnica se basa en dos pasos principales: *proceso de construcción de una solución influenciado por los rastros de feromona artificial*

(operador Φ) y *actualización de dichos rastros* (procedimiento $\mathcal{U}$). El algoritmo no fija ninguna planificación o sincronización a priori entre las fases, pudiendo ser incluso realizadas simultáneamente.

Optimización basada en cúmulos de partículas (PSO) Los algoritmos de optimización basados en cúmulos de partículas (*Particle Swarm Optimization*, PSO) también forman parte de la inteligencia colectiva. Los algoritmos PSO están inspirados en el comportamiento social del vuelo de las bandadas o el movimiento de los bancos de peces [57]. El algoritmo PSO mantiene un conjunto de soluciones, también llamadas partículas, que son inicializadas aleatoriamente en el espacio de búsqueda. Cada partícula posee una posición y una velocidad que cambian conforme avanza la búsqueda. El operador Φ de esta metaheurística está definido por el movimiento que realiza una partícula. Dicho movimiento incluye la velocidad y las posiciones donde la propia partícula y las partículas de su vecindario encontraron buenas soluciones. Datos importantes a la hora de: seleccionar soluciones (σ), actualizar las variables de estado ($\mathcal{U}$) o de decidir la terminación del algoritmo (τ). En el contexto de PSO, el vecindario de una partícula se define como un conjunto de partículas del cúmulo. No debe confundirse con el concepto de vecindario de una solución utilizado previamente en este capítulo. El vecindario de una partícula puede ser global, en el cual todas las partículas del cúmulo se consideran vecinas, o local, en el que sólo las partículas más cercanas se consideran vecinas.

3.1.4. Metaheurísticas híbridas

En los últimos años, el interés en las *metaheurísticas híbridas* ha aumentado considerablemente en el área de optimización. Los mejores resultados encontrados en muchos problemas de optimización clásicos o de la vida real son obtenidos por algoritmos híbridos [176]. Combinaciones de algoritmos tales como metaheurísticas basadas en población y/o en trayectoria, *programación matemática, programación con restricciones* y *técnicas de aprendizaje automático* (machine learning) proveen algoritmos de búsqueda muy poderosos. Cuatro diferentes tipos de combinaciones se pueden considerar:

- Combinar dos o más metaheurísticas diferentes.

- Combinar metaheurísticas con métodos exactos procedentes de la programación matemática, los cuales son ampliamente usados en investigación operativa.
- Combinar metaheurísticas con métodos provenientes de la programación con restricciones desarrollados en la comunidad de la inteligencia artificial.
- Combinar metaheurísticas con técnicas de aprendizaje automático y de minería de datos.

La hibridación de metaheurísticas implica una serie de cuestiones importantes calificadas como diseño e implementación. La primera categoría se refiere a un algoritmo híbrido en sí mismo, se planteen cuestiones tales como la funcionalidad y la arquitectura del algoritmo, mientras que la implementación incluye la plataforma de hardware, el modelo de programación y el ambiente en el cual el algoritmo se ejecuta. Ambas cuestiones son claves a la hora de catalogar a las metaheurísticas híbridas como se muestra en los siguientes parágrafos.

3.1.4.1. Clasificación de las metaheurísticas híbridas

La taxonomía presentada a continuación es la propuesta por Talbi en [176] y en [177] y, como ya se ha mencionado, marca la diferencia entre las cuestiones de diseño usadas para introducir la hibridación y las cuestiones de implementación dependientes del modelo de ejecución del algoritmo.

Diseño Bajo este aspecto la hibridación se clasifica en jerárquica y plana.

- ***Clasificación jerárquica.*** En el tope de la jerarquía se pueden distinguir hibridaciones de alto y bajo nivel (ver figura 3.2). La hibridación de bajo nivel apunta a la composición funcional de un único método de optimización. En esta clase de híbridos, una función dada de una metaheurística es reemplazada por otra metaheurística. En los algoritmos híbridos de alto nivel, las metaheurísticas híbridas son auto contenidas. Esto significa que no existe una relación directa con el trabajo interno de una metaheurística.

  En la *hibridación de relevos* (o en cadena), un conjunto de metaheurísticas se aplican una después de otra, usando la salida de la previa como entrada de la

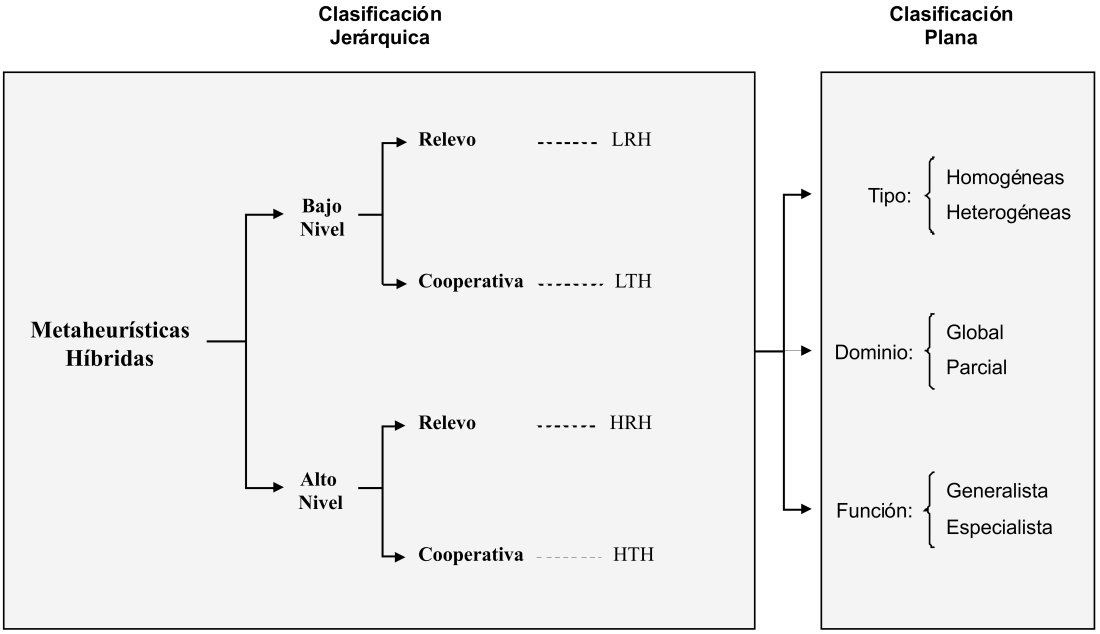


Figura 3.2: Clasificación de las metaheurísticas híbridas en términos de diseño.

actual. La *hibridación cooperativa* representa a los modelos de optimización en los cuales muchos agentes cooperantes evolucionan en paralelo; cada agente lleva a cabo una búsqueda en un espacio de soluciones.

De esta taxonomía jerárquica se derivan cuatro clases de metaheurísticas híbridas:

1. *Hibridación de relevos a bajo nivel (LRH).* Esta clase de híbridos representa algoritmos en los cuales una metaheurística dada es embebida en otra metaheurística basada en trayectoria. Un ejemplo de esto es embeber una búsqueda local dentro de un algoritmo SA, la idea es que la LS ayude a mejorar la solución inicial o soluciones intermedias durante el proceso de búsqueda de SA.

2. *Hibridación cooperativa a bajo nivel (LTH).* Como se sabe, dos objetivos en conflicto gobiernan el diseño de una metaheurística: Exploración y Explotación. La exploración es necesaria para asegurar que cada parte del espacio sea lo suficientemente examinado para proveer una estimación confiable del

óptimo global. La explotación es importante dado que el refinamiento de la solución actual produce a menudo una mejora en dicha solución. Las metaheurísticas basadas en población generalmente son más fuertes en la exploración del espacio de búsqueda que en la explotación de las soluciones encontradas.

Por lo tanto, la mayoría de las metaheurísticas basadas en población eficientes se han acoplado con metaheurísticas basadas en trayectoria tales como LS, SA y TS, cuya fortaleza radica en la explotación de una solución dada. De esta manera estas dos clases de algoritmos se complementan en sus fortalezas y debilidades. En los híbridos LTH, una metaheurística basada en trayectoria es embebida en otra basada en población. Un ejemplo de esta clase de híbridos es el reemplazo de un operador genético en un algoritmo evolutivo con una metaheurística basada en trayectoria.

3. *Hibridación de relevos a alto nivel (HRH).* En esta clase de híbridos, las metaheurísticas auto-contenidas son ejecutados en una secuencia. Por ejemplo, la solución inicial de una metaheurística basada en trayectoria puede ser generada por otro algoritmo de optimización, como son los algoritmos voraces cuya complejidad computacional es menor que la de las heurísticas iterativas y que en general es determinista. Este esquema también puede aplicarse a las metaheurísticas basadas en población, pero es necesario un algoritmo voraz aleatorio para generar una población diversa.

   Otros algoritmos híbridos HRH, ampliamente usados, surgen de la combinación de metaheurísticas basadas en población con las basadas en trayectoria. De esta forma una vez que las metaheurísticas basadas en población localizan regiones de alta performance del complejo y gran espacio de búsqueda, puede ser útil intensificar la búsqueda en dichas regiones aplicando las basadas en trayectoria. Estas últimas también se utilizan para generar diversidad en una población de soluciones que se vuelva bastante uniforme provocando un estancamiento en los valores de *fitness*.

4. *Hibridación cooperativa a alto nivel (HTH).* Este esquema involucra distintos algoritmos auto-contenidos que realizan una búsqueda en paralelo y cooperan para encontrar un óptimo. Esta colaboración consiste en proveer información a los otros para ayudarlos en sus respectivos procesos de búsqueda. Un ejemplo característico es el modelo isla para los algoritmos genéticos.

- ***Clasificación plana.*** Aquí las metaheurísticas híbridas son clasificadas de la siguiente forma:

1. *Homogéneos versus heterogéneos.* Los *híbridos homogéneos* son algoritmos combinados que usan la misma metaheurística. Un ejemplo típico de esto es el modelo isla para GAs. Las metaheurísticas híbridas homogéneas pueden diferir en la inicialización de los parámetros (configuraciones paramétricas distintas al de los algoritmos intervinientes) y de los componentes de búsqueda (diferentes estrategias para cualquier componente de la metaheurística), mientras que en los *híbridos heterogéneos*, las metaheurísticas usadas son diferentes. Esta clase de híbridos se puede ejemplificar con el algoritmo heterogéneo HTH basado en algoritmos genéticos y la búsqueda tabú propuesto en [45] para resolver un problema de diseño.
2. *Global versus Parcial.* Desde otro punto de vista, se pueden distinguir dos formas de cooperación: global y parcial. En los *híbridos globales*, todos los algoritmos exploran el mismo espacio de búsqueda conjunto. Aquí, el objetivo es explorar el espacio más a fondo. Todos los híbridos mencionados anteriormente son híbridos globales en el sentido de que todos resuelven el problema de optimización completo.

   En los *híbridos parciales*, el problema a ser resuelto es descompuesto *a priori* en sub-problemas, y cada uno con su propio espacio de búsqueda, entonces cada algoritmo está dedicado a buscar en uno de estos sub-espacios de soluciones. Los sub-problemas están relacionados entre ellos, lo que implica restricciones entre los óptimos encontrados, por ende los algoritmos necesitan comunicarse para respetar estas restricciones y para construir una solución global viable al problema.

3. *Generalistas versus especialistas.* Todos los híbridos descriptos anteriormente son *híbridos generalistas*, en el sentido que todos ellos resuelven el mismo problema de optimización. Los *híbridos especialistas* combinan algoritmos que resuelven problemas distintos. Tal es el caso del modelo genérico CO-SEARCH [178] que maneja la cooperación de un agente de búsqueda local, un agente diversificador y otro intensificador; que intercambian información por medio de una memoria adaptativa.

Implementación En la figura 3.3 se muestra la taxonomía concerniente a las cuestiones de implementación en metaheurísticas híbridas. Esta clasificación es discutida en los siguientes items:

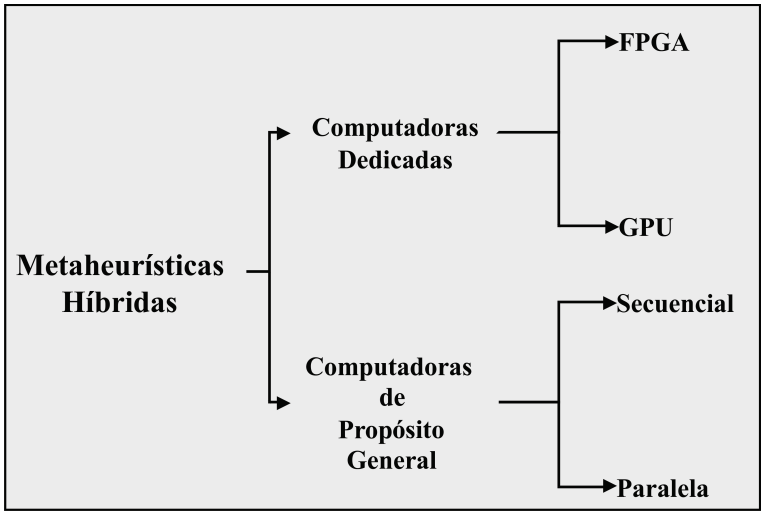


Figura 3.3: Clasificación de las metaheurísticas híbridas en términos de implementación.

- ***Computadoras de propósito general versus dedicadas.*** Las aplicaciones ejecutadas en computadoras específicas difieren de las ejecutadas en computadoras de propósito general, en que las primeras usualmente resuelven un conjunto pequeño de problemas, pero a menudo a tasas de eficiencia mucho más altas y menor costo.

- ***Secuencial versus paralelo.*** La mayoría de las metaheurísticas híbridas propuestas en la literatura son secuenciales. Pero en función al tamaño del problema, se consideran implementaciones paralelas de algoritmos híbridos, siendo el modelo híbrido HTH el más fácilmente paralelizable. Los modelos paralelos se detallan en la próxima sección.

3.1.5. Metaheurísticas paralelas

Los problemas de optimización de la vida real, a menudo, son NP-duros y grandes consumidores de tiempo de CPU y de memoria. Aunque el uso de metaheurísticas permiten reducir significativamente la complejidad computacional del proceso de búsqueda, para muchos problemas de optimización este consumo de recursos sigue siendo muy costoso. Esto puede suceder cuando la función objetivo y las restricciones asociadas con el problema hacen un uso intensivo de los recursos y, además, el tamaño del espacio de búsqueda es enorme.

El uso de la computación distribuida y paralela se ha convertido en una estrategia efectiva para atacar estos problemas de optimización tan complejos. El paralelismo puede usarse en el diseño y la implementación de metaheurísticas por las siguientes razones:

- ***Velocidad de la búsqueda.*** Uno de los principales objetivos de paralelizar una metaheurística es reducir el tiempo de búsqueda.

- ***Aumentar la calidad de las soluciones obtenidas.*** Algunos modelos metaheurísticos paralelos permiten incrementar la calidad de la búsqueda, ya que intercambian información entre metaheurísticas cooperativas en términos de la búsqueda en el espacio de soluciones del problema asociado.

- ***Mejorar la robustez.*** Una metaheurística paralela puede ser más robusta al resolver de manera efectiva diferentes problemas de optimización y distintas instancias de un problema dado. La robustez también puede medirse en términos del grado de reacción de la metaheurística frente a variaciones en la configuración de sus parámetros.

- ***Resolver problemas a gran escala.*** Las metaheurísticas paralelas permiten resolver instancias de gran tamaño de los problemas de optimización, así como también, resolver con mayor precisión los modelos matemáticos asociados a dichos problemas.

A continuación se presentan los principales modelos paralelos para metaheurísticas [177].

3.1.5.1. Modelo paralelo a nivel del algoritmo

En este modelo paralelo se usan tanto metaheurísticas colaborativa como independientes. Se trata de una paralelización interalgorítmica independiente del problema a resolver. Si las diferentes metaheurísticas son independientes, la búsqueda es equivalente a la ejecución secuencial de las mismas en términos de la calidad de soluciones. Sin embargo el modelo cooperativo (o colaborativo) altera el comportamiento de la metaheurística mejorando la calidad de las soluciones.

En el caso de usar metaheurísticas independientes, cada una puede ser inicializada con diferentes soluciones o poblaciones de soluciones. Además pueden utilizarse distintas configuraciones paramétricas, tales como el tamaño de la lista tabú para TS, las probabilidades de transición para colonias de hormigas, las probabilidades de mutación y crossover para los EAs y así siguiendo. Más aún, es posible realizar distintos diseños para cada componente de una metaheurística: codificación, operadores de búsqueda, función objetivo, restricciones, condición de terminación, entre otros. El paradigma maestro−esclavo se adapta muy bien a este modelo paralelo. Un esclavo implementa una metaheurística; mientras que el maestro define los diferentes parámetros a ser usados por los primeros y determina la mejor solución encontrada a partir de las soluciones obtenidas por los diferentes esclavos. Además de incrementar la velocidad del algoritmo, este modelo paralelo incrementa la robustez del mismo.

En cuanto al modelo cooperativo para metaheurísticas paralelas, los diferentes algoritmos intercambian información sobre la búsqueda con el objetivo de computar soluciones de mayor calidad y robustez. En el diseño de esta clase de metaheurísticas es necesario considerar:

- ***Cuándo se intercambia información.*** El criterio de intercambio puede ser *ciego* (periódico o probabilístico) o *inteligente* (adaptativo). El intercambio periódico ocurre en cada algoritmo después de un número fijo de iteraciones. El intercambio probabilístico consiste en comunicarse después de cada iteración con una cierta probabilidad. Los intercambios adaptativos están guiados por la evolución de alguna característica de la búsqueda durante la ejecución del algoritmo.

- ***Cuál es la topología de intercambio.*** La topología determina la fuente y destino de la información a intercambiar. Las topologías más usadas son la de anillo, malla e hipercubos. Los anillos pueden ser direccionales (por ej. un grafo dirigido) o bidireccionales (por ej. un grafo no dirigido). En un hipercubo de orden k, existen 2^k y cada nodo tiene k vecinos, puede usarse un grafo completo o uno aleatorio. En el caso del grafo completo cada nodo está conectado a todos los otros nodos, mientras que, en el caso del aleatorio un nodo envía información a un subconjunto de nodos elegidos al azar. Diferentes estrategias pueden usarse para seleccionar a los vecinos.

- ***Cuál es la información a intercambiar.*** En general esta información puede estar compuesta por:

  - **Soluciones.** Esta información consta de una selección de las soluciones generadas y almacenadas durante la búsqueda. Por lo general, contiene las mejores soluciones que han sido encontradas. También es conveniente enviar la calidad de las soluciones, para no calcularlas nuevamente en el destino. En el caso de las metaheurísticas basadas en trayectoria, generalmente, se intercambia la mejor solución hallada. En cambio para las basadas en población, el número de soluciones a cambiar puede ser un valor absoluto o un porcentaje de la población. Si bien cualquier mecanismo de selección puede utilizarse para escoger las soluciones, la estrategia más usada consiste en elegir las mejores soluciones para un criterio dado.

  - **Memoria de búsqueda.** Esta información se relaciona con elementos de la memoria de búsqueda asociados con la metaheurística involucrada. Por ejemplo, la memoria a largo o a corto plazo utilizada en TS, o el rastro de feromona en ACO.

- ***Cuál es la política de integración.*** Esta política se refiere a cuál es la estrategia a usar para actualizar las variables locales utilizando la información recibida. Por ejemplo, en un EA la solución recibida reemplazará la peor solución de la población.

En el modelo cooperativo también es posible aplicar una estrategia heterogénea, esto significa que los componentes de la búsqueda tienen características diferentes. Los siguientes son posibles niveles de heterogeneidad:

- ***A Nivel de los parámetros.*** Una misma metaheurística forma parte del modelo paralelo, pero se usan distintas configuraciones de parámetros [46, 89, 183, 189].
- ***A Nivel de la búsqueda.*** En este nivel la heterogeneidad se introduce al emplear diferentes componentes de la búsqueda. Ejemplo de esto es el uso de distintos operadores de mutación y cruce en los EAs y de vecindarios diferentes en las metaheurísticas basadas en trayectoria [2, 14, 116].
- ***A nivel del modelo de optimización.*** Cada metaheurística optimiza un problema distinto al utilizar, distintas funciones objetivos y/o restricciones [115].
- ***A nivel de algoritmo.*** Esta es la clase más general en la cual diferentes algoritmos metaheurísticos forman parte del modelo paralelo [25, 147].

3.1.5.2. Modelo paralelo a nivel de la iteración

El modelo paralelo a nivel de la iteración se focaliza en la paralelización de cada iteración de las metaheurísticas basándose en general en la distribución de las soluciones manejadas. Este modelo es muy útil en aquellos casos donde la evaluación de la función objetivo es la tarea que consume más tiempo en la metaheurística.

En las metaheurísticas basadas en trayectoria se paraleliza la generación y la evaluación del vecindario o soluciones candidatas, dado que es el paso de computación más intensivo. Para esta clase de metaheurísticas paralelas, el vecindario es descompuesto en particiones distintas; las cuales son generadas y evaluadas en paralelo haciendo posible la exploración de grandes vecindarios.

En las metaheurísticas basadas en población la paralelización se enfoca en cada elemento de la población (por ejemplo: individuos en EAs, hormigas en ACO, partículas en PSO, soluciones en SS) como una unidad independiente. El modelo paralelo a nivel de iteración involucra la distribución de la población. Las operaciones aplicadas a cada elemento de la población son realizadas en paralelo.

3.1.5.3. Modelo paralelo a nivel de la solución

En este modelo se paralelizan las operaciones dependientes del problema aplicadas a las soluciones. En general el interés está puesto en la paralelización de una sola solución. Este modelo es particularmente interesante cuando la evaluación de la función objetivo o de las restricciones hacen uso intensivo de los recursos (tiempo y/o memoria). Dos diferentes modelos paralelos a nivel de la solución pueden considerarse:

- ***Descomposición funcional.*** Aquí se divide la función objetivo y/o las restricciones en diferentes funciones parciales, las cuales son evaluadas en paralelo reduciendo, así, el tiempo de cómputo. Por definición este modelo es síncrono.
- ***Partición de datos.*** Para algunos problemas, la función objetivo puede requerir el acceso a grandes bases de datos. Debido a la restricción sobre los requisitos de memoria, la base de datos se distribuye en diferentes sitios y se explota el paralelismo en los datos al evaluar la función objetivo

3.2. Ensambladores de uso común

Los paquetes que se introducen en esta sección forman parte de las técnicas más populares para resolver el FAP, todas ellas están basadas en algoritmos voraces y diseñadas para resolver únicamente este problema. Estas técnicas hacen frente a las distintos inconvenientes del problema de ensamblado de fragmentos pero no a todos ellos. En las siguientes subsecciones se describen tres de las más usadas en la literatura. Finalmente se introducen otro grupo de ensambladores, también diseñados específicamente para solucionar FAP.

3.2.1. PHRAP

PHRAP es un ensamblador que realiza la comparación, la alineación y ensamblado de grandes conjuntos de secuencias de ADN [80]. PHRAP compara las secuencias mediante la búsqueda de pares de fragmentos perfectamente congruentes o secuencia de las regiones que cumplen con ciertos criterios. Si se encuentra una coincidencia, intenta extender la alineación en contigs. Además, utiliza los valores de calidad producidos por PHRED. Este último,

PHRED, lee los archivos de cromatogramas de las secuencias de ADN y los análisis de los picos para identificar las bases y asignar puntuaciones de calidad a cada base identificada.

PHRAP no necesita recortar las secuencias, puede trabajar con secuencias repetidas o usar datos sobre repeticiones calculadas internamente y es capaz de manejar grandes conjuntos de datos. Además, recorta las regiones de baja calidad de los fragmentos y usa la base de valores de calidad en la evaluación de los solapamientos y en la generación de contigs.

3.2.2. Familia CAP (Contig Assembly Program)

CAP realiza tres tareas diferentes [93]: la detección de fragmentos superpuestos, la formación de contigs y la generación de la secuencia consenso. En la primera, este ensamblador calcula los solapamientos entre cada par de fragmentos de entrada. Después de eso, ensambla los contigs usando una técnica voraz, esto es, añadiendo el fragmento con mayor superposición (encontrado en la fase previa) de uno en uno. Por último, CAP genera secuencias de consenso con los contigs formados previamente, al unir cada par de fragmentos alineados en un alineamiento múltiple de fragmentos.

CAP2 mejora a CAP, ya que filtra los fragmentos que no se superponen con ningún otro, denominados *singlets*, identifica fragmentos quiméricos y se ocupa de secuencias repetidas. CAP3 es una versión ampliada de CAP2 y CAP. CAP3, un ensamblador muy especializado, tiene la capacidad de: recortar las regiones de fragmentos de poca calidad, utilizar los valores de calidad de las bases en el cálculo de los solapamientos entre los fragmentos y usar las restricciones hacia adelante-atrás (que especifican los pares de fragmentos opuestos) para lograr un correcto ensamblado uniendo contigs sin errores.

3.2.3. Celera Assembler

Celera Assembler es un programa científico desarrollado para la investigación del ADN [132]. Puede reconstruir largas secuencias de ADN usando los fragmentos obtenidos por el método de secuenciamieto *whole-genome shotgun*. Es decir, este ensamblador utiliza algoritmos de grafos y cadenas sofisticadas basados en el paradigma de ensamblado superposición-distribución-c

Puede manejar millones de fragmentos y hace un uso extensivo de restricciones hacia adelante-atrás para hacer frente al problema de las regiones repetidas.

Este ensamblador ha permitido descubrir genomas eukarioticos grandes, genomas diploides y genomas provenientes de muestras del medioambiente. También ha contribuido en la identificación de la primera secuencia diploide de un individuo humano y en el ensamblado metagenómico del Muestreo Oceánico Global.

El Celera Assembler es un miembro de la clase de software ensamblador llamado "*whole-genome shotgun assemblers*". Celera Assembler está escrito mayormente en código C para el sistema operativo Unix. Aunque requiere de grandes recursos computacionales para resolver genomas complejos, puede ensamblar genomas bacteriales en una laptop.

3.2.4. Otros

Como se ha mencionado al inicio de este capítulo son muchas las técnicas específicamente diseñadas para resolver FAP, además de las descriptas anteriormente se pueden hallar:

- *TIGR Assembler* [173] es una herramienta ensambladora clásica desarrollada por TIGR (*Institute for Genomic Research*) para construir una secuencia de ADN consensuada usando fragmentos pequeños de la misma. TIGR Assembler ha sido utilizado en una serie de proyectos relacionados con el genoma microbiano. Este software garantiza desarrollos y mejoras continuos para afrontar los nuevos desafíos en los proyectos de genomas del ser humano, del ratón y del maíz. TIGR Assembler es comparable a PHRAP y a otros ensambladores basados en algoritmos voraces.

- *STROLL* [37] es un ensamblador de fragmentos desarrollado por el equipo del *Brookhaven National Laboratory*. STROLL originalmente proporcionó apoyo computacional para la estrategia de secuenciación *primer walking*, pero se ha convertido en un ensamblador de propósito general que soporta una amplia variedad de tecnologías de secuenciamiento. STROLL mejora la precisión del secuenciamiento y resuelve regiones repetidas al incorporar: (1) la comparación por pares para discriminar superposiciones, regiones repetidas y quimeras, (2) las alineaciones múltiples incrementales para unir un

fragmento en un ensamblado parcial y (3) la generación de consenso para determinar la base de consenso y brindar un nivel de confianza asociado [38].

- *EULER* [142] se diferencia del resto de los ensambladores ya que formula al problema como el problema del *SuperString* más Corto (*Shortest Superstring Problem*). De esta forma Euler cambia el paradigma "superposición-distribución-consenso" por el "Supercamino Euleriano" (*Eulerian Superpath*). Cabe aclarar que ambos paradigmas pertenecen a la clase de problemas NP-duros.

  Euler pasa por alto el problema de las regiones repetidas porque el enfoque del Supercamino Euleriano transforma las regiones repetidas imperfectas en diferentes caminos en el grafo de Bruijn. Como resultado, Euler no reconoce regiones repetidas a menos que sean perfectas y extensas.

3.3. Conclusiones

En este capítulo se introduce una larga lista de algoritmos ensambladores. En primer lugar, se introducen técnicas de uso general tales como las metaheurísticas, sobre las cuales se dan las definiciones propuestas por Gabriel Luque y Francisco Chicano en sus respectivas tesis doctorales [40, 118]. También se ha incluido un repaso por las técnicas más importantes y populares dentro de este campo. Estas breves descripciones han sido realizadas siguiendo una clasificación de las metaheurísticas que las dividen en dos clases, atendiendo al número de soluciones tentativas con la que trabajan en cada iteración: metaheurísticas basadas en trayectoria y en población. Tras este repaso se han introducido las metaheurísticas paralelas indicando las distintas formas de paralelismo consideradas en la literatura.

Por último, se han descripto en detalle los algoritmos ensambladores diseñados específicamente para resolver el FAP. Haciendo especial hincapié en los más usados en la literatura: PHRAP, CAP y CELERA Assembler.

Capítulo 4

Algoritmos metaheurísticos básicos de partida

En el capítulo anterior se realiza un compendio de algoritmos ensambladores, poniendo especial énfasis en las técnicas metaheurísticas utilizadas para resolver eficaz y eficientemente el FAP. En este capítulo se presentan de forma detallada las familias de algoritmos metaheurísticos en que se basan los algoritmos propuestos en este trabajo de tesis para ensamblar fragmentos de ADN. Estas familias son: enfriamiento simulado, búsqueda en vecindario variable, búsqueda local guiada por el problema y algoritmos evolutivos. Dentro de la sección dedicada a los algoritmos evolutivos están especialmente detalladas las particularidades de los algoritmos genéticos.

4.1. Metaheurísticas basadas en trayectoria

En esta sección se describen las características de diseño de los ensambladores metaheurísticos basados en trayectoria que se utilizan en este trabajo. Ellos son: enfriamiento simulado, búsqueda en vecindario variables y búsqueda local guiada por el problema.

4.1.1. Enfriamiento Simulado

Enfriamiento simulado (Simulated Annealing, SA) es un método Monte-Carlo simple y de propósito general desarrollado para optimización combinatoria en la década de 1980.

SA es un algoritmo metaheurístico basado en una analogía con los sistemas físicos. Los autores [104] de este método se inspiraron en otro método Monte-Carlo conocido como el algoritmo Metropolis creado para calcular las propiedades de cualquier sustancia que pueda ser considerado como un compuesto de moléculas individuales que interactúan.

En física, el enfriamiento es un proceso térmico para la obtención de los estados de baja energía de un sólido en un baño de calor. En la fase líquida (a muy alta temperatura), todas las partículas se disponen al azar, durante el proceso de calentamiento y enfriamiento lento esas partículas se organizan en una red más estructurada disminuyendo su energía. Cuando la temperatura inicial es bastante alta y el proceso de enfriamiento es suficientemente lento, se alcanza el estado fundamental (y su energía correspondiente es mínima). En otro caso, el metal se congela en un estado meta-estable.

El procedimiento de Metrópolis es una copia exacta de este proceso físico, mientras que el enfriamiento simulado se puede considerar una iteración de algoritmos de Metrópolis, ejecutado al disminuir los valores de un parámetro de control denominado temperatura. En esta clase de algoritmos, las soluciones para un problema de optimización combinatoria se consideran como estados de los sistemas físicos y el costo de una solución es equivalente a la energía de un estado.

En otras palabras, en los primeros pasos (a alta temperatura), SA acepta soluciones con mayores costos en virtud de una cierta probabilidad, con el fin de explorar el espacio de búsqueda y para escapar de óptimos locales. Durante el proceso de enfriamiento esta probabilidad disminuye de acuerdo a la temperatura de enfriamiento, la intensificación de la búsqueda y la reducción de la exploración con el fin de explotar una zona restringida de un espacio de búsqueda. En los últimos pasos la exploración es nula (o casi nula) y la explotación es total (o casi total). En el algoritmo 1, se muestra un pseudo-código de un algoritmo de enfriamiento simulado básico.

El enfriamiento simulado es un sistema dinámico, por ende, es necesario trabajar mediante la simulación de relajaciones sucesivas de un sistema físico ficticio a temperaturas cada vez más bajas. Eso significa que, SA evoluciona a través de una secuencia de transiciones entre estados; las cuales son generadas por probabilidades de transición. En consecuencia, SA puede ser matemáticamente modelado por cadenas de Markov, donde se genera una secuencia

Algoritmo 1 Enfriamiento Simulado

```
k = 0;
inicializar T y S_0; {solución inicial y temperatura }
evaluar S_0 in E_0;
repeat
  repeat
    k = k + 1;
    generar S_1 desde S_0
    evaluar S_1 en E_1;
    {si la nueva solución es mejor que la actual, S_1 es aceptada}
    if (E_1 − E_0) ≤ 0 then
      S_0 = S_1;
      E_0 = E_1;
      {si la nueva solución es peor que la actual,
      S_1 es aceptada bajo la probabilidad: exp((E_1 − E_0)/T)}
    else
      if exp((E_1 − E_0)/T) > random[0, 1) then
        S_0 = S_1;
        E_0 = E_1;
      end if
    end if
  until (k mod Long. Cadena de Markov) == 0
  actualizar T;
until condición de corte sea satisfecha
return S_0;
```

de cadenas usando una probabilidad de transición cuyo cálculo involucra a la temperatura actual.

Sea (S, f) (donde S es un conjunto de soluciones y f es una instancia del problema), N una función de vecindario y $\mathbf{X}(k)$ una variable estocástica que indica el resultado de la $k-$ésima iteración. Entonces la probabilidad de transición en la $k-$ésima iteración para cada par $i, j \in S$ del resultado se define como:

$$P_{ij}(k) = \mathbb{P}\{\mathbf{X}(k) = j | \mathbf{X}(k-1) = i\} \tag{4.1}$$

$$= \begin{cases} G_{ij}(c_k)A_{ij}(c_k) \;\; si \;\; i \neq j \\ 1 - \sum_{l \in S, l \neq i} G_{il}(c_k)A_{il}(c_k) \;\; si \;\; i = j \end{cases}$$

donde $G_{ij}(c_k)$ denota la probabilidad de generación de la solución j a partir de la i, siendo $A_{il}(c_k)$ la probabilidad de aceptación de la solución recientemente generada j. Las probabilidades usadas con más frecuencia son [1]:

$$G_{ij}(c_k) = \begin{cases} |N(i)|^{-1} \;\; si \;\; j \in S_i \\ 0 \;\; si \;\; j \notin S_i \end{cases} \tag{4.2}$$

y

$$A_{ij}(c_k) = \begin{cases} 1 \;\; si \;\; f(j) \leq f(i) \\ exp(\frac{(f(i)-d(j))}{c}) \;\; si \;\; f(j) > f(i) \end{cases} \tag{4.3}$$

Para un valor fijo de c, las probabilidades no dependen de k, en cuyo caso la cadena de Markov resultante es independiente del tiempo u homogénea. Al usar la teoría de las cadenas markovianas resulta relativamente sencillo mostrar que, bajo la condición de vecindarios fuertemente conectados (en cuyo caso la cadena de Markov es irreducible y aperiódica) existe una única distribución estacionaria de los resultados. Esta distribución es la probabilidad de distribución de las soluciones después de un número infinito de iteraciones y asume la siguiente forma [1]:

Teorema[4.1] Dada una instancia (S, f) de un problema de optimización combinatoria y una función de vecindario apropiada, entonces, después de un número suficientemente grande de transiciones en un valor fijo c del parámetro de control, al aplicar las probabilidades de transición de 4.1, 4.2 y, 4.3, enfriamiento simulado se encontrará una solución $i \in S$ con una probabilidad dada por

$$\mathbb{P}_c\{\mathbf{X} = i\} = q_i(c) = \frac{1}{N_0(c)} exp\left(\frac{-f(i)}{c}\right) \tag{4.4}$$

donde **X** es una variable estocástica que denota la solución actual obtenida por SA y

$$N_0(c) = \sum_{j \in S} exp\left(\frac{-f(i)}{c}\right) \tag{4.5}$$

denota una constante de normalización.

Una demostración de este teorema está más allá del alcance de este capítulo y se la puede encontrar en [1]. La distribución de probabilidad de la ecuación 4.4 es denominada estacionaria o distribución de equilibrio y es el equivalente a la distribución de Boltzmann de la ecuación 4.1. A partir de esto es posible formular el siguiente resultado.

Corolario[4.1] Dada una instancia (S, f) de un problema de optimización combinatoria y una función de vecindario apropiada, además de, la probabilidad $q_i(c)$ de que SA encuentre una solución i después de un número infinito de iteraciones en un valor c del parámetro de control dado por el teorema 4.4, entonces

$$\lim_{x \to 0} = q_i^* = \frac{1}{|S^*|}\mathcal{X}_{S^*}(i), \tag{4.6}$$

donde S^* denota el conjunto de soluciones óptimas globales.

El resultado de este corolario es importante dado que garantiza la convergencia asintótica del algoritmo SA a un conjunto de soluciones globalmente óptimas bajo la condición que la distribución estacionaria de (4.4) se alcanza para cada valor de c.

En otras palabras, SA puede encontrar soluciones óptimas con probabilidad igual a 1 si fuese posible realizar un número infinito de transiciones. El algoritmo 1 muestra una implementación eficiente de SA en tiempo finito. Evidentemente, esto será a costa de la garantía de obtener soluciones óptimas. Sin embargo, la práctica demuestra que de esta manera se pueden obtener soluciones de alta calidad.

Hasta este punto se ha introducido el algoritmo SA y se ha explicado cómo se lo puede modelar matemáticamente usando cadenas markovianas. Resta describir cuáles son los principales métodos de actualización de la temperatura, dicho en otras palabras cuáles son las diferentes planificaciones del enfriamiento, conocido en inglés como *cooling schedule*.

La planificación del enfriamiento especifica cómo la temperatura decrece en cada iteración. Existe un amplio rango de métodos para determinar dicha planificación. Miki et al. en [126], por ejemplo, usan algoritmos genéticos para este propósito, pero en este trabajo

solamente se incluyen las tres formas de planificación más simples y usadas en la literatura, a saber:

- ***Enfriamiento proporcional*** [104]:

$$T_{k+1} = \alpha * T_k \tag{4.7}$$

donde α es una constante próxima a uno, aunque siempre está por debajo de dicho valor. Por lo general α se calcula como sigue:

$$\alpha = \frac{k}{k+1} \tag{4.8}$$

- ***Enfriamiento logarítmico*** [83]:

$$T_{k+1} = C * T_k \tag{4.9}$$

En la ecuación 4.9 la cadena converge a un mínimo global de energía, donde la constante C es mayor a la profundidad del mínimo local distinto al global. En general dicha constante se calcula como muestra la ecuación 4.10. La mayor desventaja de este tipo de enfriamiento es el tiempo requerido para alcanzar la temperatura mínima.

$$C = \frac{log(k)}{log(1+k)} \tag{4.10}$$

- ***Enfriamiento exponencial*** [104], también denominado enfriamiento geométrico: la ecuación 4.11 describe este esquema, donde la constante α^k si bien es menor a uno, siempre, es un valor muy cercano a la unidad. Por lo general esta constante toma la forma que se muestra en la ecuación 4.12.

$$T_{k+1} = T_k * \alpha^k \tag{4.11}$$

$$\alpha^k = \frac{e^k}{e^{1+k}} \tag{4.12}$$

Esta metaheurística ha sido utilizada por Churchill y sus colegas en 1993 [42], para ensamblar secuencias de ADN. Al año siguiente Burks et al. en [28] y en [29] usan SA como herramienta estocástica en el ensamblado de fragmentos. En el 2005 Luque, Alba y Khuri, en [120], paralelizan a SA con el mismo objetivo para resolver instancias de mayor tamaño y complejidad.

4.1.2. Búsqueda de vecindario variable

La búsqueda de vecindario variable (Variable Neighborhood Search, VNS) es una metaheurística reciente, propuesta por Mladenović en [130, 131], que explota sistemáticamente la idea de cambio de vecindario, tanto en el descenso hasta los mínimos locales como en el escape de los valles que los contienen.

De esta forma, VNS procede por un método descendiente hacia un mínimo local, entonces explora una serie de distintos vecindarios predefinidos de la solución actual. En cada iteración, uno o más puntos del vecindario actual se usan como punto de inicio para métodos descendientes locales que finalizan en un mínimo local. La búsqueda salta al nuevo mínimo si y solo si éste es mejor que el actual. En este sentido, no es un método que sigue una trayectoria por una misma región como lo hace SA o TS y tampoco especifica movimientos prohibidos. Pero a pesar de su sencillez, VNS demuestra ser eficaz explotando sistemáticamente las siguientes características:

1. Un mínimo local con respecto a una estructura de vecindario no lo es necesariamente para otra estructura.

2. Un mínimo global es uno local con respecto a todas las posibles estructuras de vecindario.

3. En muchos problemas, los mínimos locales relacionados a uno o varios vecindarios están relativamente cercanos unos de otros.

Esta última propiedad, empírica, implica que un óptimo local a menudo provee información sobre uno global. Sin embargo, usualmente no se conoce el óptimo global como tal.

A diferencia de la mayor parte de las metaheurísticas, el esquema básico de VNS y sus extensiones son simples y requieren pocos o ningún parámetro. La idea básica consiste en cambiar la estructura de vecindario cuando la búsqueda local queda atrapada en un óptimo local. Una estructura de vecindario, en un espacio de soluciones S es una asignación de $N : S \to 2^S$, que asocia a cada $x \in S$ un entorno de soluciones, o vecindario, $N(x) \subset X$. Las metaheurísticas de búsqueda local aplican una transformación o movimiento a la solución de búsqueda y por tanto utilizan, explícita o implícitamente, una estructura de vecindarios.

Se denota con N_k $(k = 1, ...k_{max})$ al conjunto finito de estructuras de vecindarios pre-seleccionadas y con $N_k(x)$ al conjunto de soluciones en el $k-$ésimo vecindario de x. Los entornos N_k pueden ser inducidos por una o más métricas introducidas en el espacio de soluciones S. La mayoría de las heurísticas de búsqueda local usan sólo una estructura de vecindarios.

La búsqueda de entorno variable básica (*Basic Variable Neighbourhood Search*, BVNS) combina cambios determinísticos y aleatorios de la estructura de vecindarios. El algoritmo 2 muestra los pasos de una VNS básica.

La condición de terminación puede establecerse considerando diferentes criterios, por ejemplo: un tiempo máximo de CPU, un número máximo de iteraciones o un número máximo de iteraciones entre dos mejoras. A menudo los vecindarios sucesivos, N_k, están anidados. Obsérvese que la solución x_0 se genera al azar en el paso denominado `agitación` (o *shaking* en Inglés) para evitar el ciclado, que puede ocurrir si se usa cualquier regla determinística.

La búsqueda de entorno variable descendente (*Variable Neighbourhood Descent*, VND), la VNS general VNS (GVNS) y la VNS reducida (RVNS) son extensiones de la VNS básica. VND se trata de una extensión determinista basada en la primera característica de VNS, por lo tanto, puede ser ventajoso combinar distintas heurísticas por descenso. Aquí una búsqueda local determina una mejor solución del entorno de la solución actual. La clásica búsqueda voraz descendente consiste en reemplazar iterativamente la solución actual por el resultado de la búsqueda local, mientras se produzcan mejoras. Por último se realiza un cambio de estructura de vecindario de forma determinista cada vez que se llega a un mínimo local. Entonces, la solución final proporcionada por el algoritmo es un mínimo local con respecto a todas las k_{max} estructuras de entornos, por lo tanto la probabilidad de alcanzar un mínimo global es mayor que usando una sola estructura.

En tanto que, RVNS es un método puramente estocástico: las soluciones de los vecindarios pre-seleccionados son elegidas aleatoriamente. Su eficiencia está basada principalmente en la tercera característica antes descripta dado que, usualmente, los vecindarios están anidados (cada uno contiene al previo), entonces un punto es elegido al azar en el primer vecindario. Si este punto es mejor que el actual la búsqueda continua en ese vecindario. De lo contrario se procede a buscar en el próximo vecindario. Después de que todos los vecin-

Algoritmo 2 VNS Básica

{*Inicialización:*}
Seleccionar el conjunto de estructuras de vecindarios $N_k, k = 1, ..., k_{max}$;
Encontrar una solución inicial x;
Elegir una condición de terminación;
while La condición de terminación no se cumpla **do**
  $k = 1$;
  while $k <= k_{max}$ **do**
    {*Agitación:*}
    Generar aleatoriamente $x' \in N_k(x)$;
    {*Búsqueda Local:*}
    Obtener el óptimo local x'' al aplicar algún método de búsqueda local a x';
    {*Mejorar o no:*}
    if x'' es mejor que x **then**
      $x = x''$;
      $k = 1$;
    else
      $k = k + 1$;
    end if
  end while
end while

darios hayan sido considerados, se vuelve a comenzar con el primero, hasta que se satisfaga la condición de finalización. La RVNS es útil para instancias muy grandes de problemas en las que la búsqueda local es muy costosa.

La GVNS combina las dos extensiones de VNS previamente explicadas. Esto es, GVNS no utiliza una búsqueda local simple sino que usa VND, además mejora la solución inicial encontrada utilizando RVNS. Una descripción detallada de todas estas extensiones puede encontrarse en [5, 84].

4.1.3. Búsqueda local guiada

La búsqueda local guiada (*Problem Aware Local Search*, PALS) es un método de búsqueda local propuesto por Alba y Luque en [8]. Se trata de un nuevo algoritmo orientado a resolver de forma precisa grandes instancias del problema de ensamblado de fragmentos.

Los ensambladores clásicos utilizan funciones de *fitness* que favorecen soluciones en las que se produce una fuerte superposición entre los fragmentos adyacentes, al usar ecuaciones como las formuladas por Parsons et al. en [138] (ver ecuación 5.1 en el capítulo 5). Pero el objetivo de PALS es obtener un orden de los fragmentos que minimize el número de contigs, con el objetivo de alcanzar un único contig, es decir una secuencia de ADN completa compuesta de todos los fragmentos superpuestos. De esta forma, el número de contigs es usado como un criterio de alto nivel para juzgar completamente la calidad de los resultados dada la dificultad de capturar la dinámica del problema en otras funciones matemáticas. Los valores de contigs son calculados al aplicar un paso final de refinamiento con una heurística voraz usualmente utilizada en este dominio [113]. En muchos casos se ha encontrado que una solución con mejor *fitness* que otra genera un mayor número de contigs, convirtiéndose en una solución peor. Todo esto sugiere que el *fitness*, calculado como la sumatoria de superposiciones entre fragmentos, debiera complementarse con el número actual de contigs. Ésta es la estrategia que implementa PALS para cumplir con su objetivo.

Además, el cálculo del número de contigs es una operación que demanda mucho tiempo de ejecución y esto, definitivamente, es una desventaja importante para cualquier algoritmo. Una solución a esto consiste en relajar dicho cálculo, para ello PALS estima el número de contigs al medir los contigs que se crean o destruyen cuando soluciones tentativas son manipuladas. Para lo cual los autores proponen una variación del método 2-opt de Lin [115] para el área Bioinformática. El algoritmo 3 muestra el pseudo-código de PALS.

Este algoritmo trabaja con una única solución, representada por una permutación; la cual es generada por `Inicializar` y modificada iterativamente mediante la aplicación estructurada de movimientos. Un movimiento es una perturbación (`AplicarMovimiento`) que, dada una solución s y dos posiciones i y j, intercambia estas dos posiciones.

Pero la clave de PALS se encuentra en el cálculo de la variación en la superposición (Δ_f) y en el número de contigs (Δ_c) entre la solución actual y la resultante de aplicar un

Algoritmo 3 Búsqueda local guiada

```
Inicializar S; {Genera la solución inicial}
repeat
  L = ∅;
  for i = 0 to N do
    for j = 0 to N do
      Δc, Δf = CalcularDelta(S, i, j); {Ver alg. 4}
      if Δc ≤ 0 then
        L = L∪ <i, j, Δc, Δf>; {Incorpora los movimientos candidatos a L}
      end if
    end for
  end for
  if L <> ∅ then
    <i, j, Δc, Δf> = Extraer(L); {Selecciona un movimiento perteneciente a L}
    ApplicarMovimineto(S, i, j); {Modifica la solución}
  end if
until no existan más cambios
return S;
```

movimiento (ver algoritmo 4). Este cálculo requiere mucho menos procesamiento y tiempo de cómputo que la función de *fitness* presentada en el capítulo 5, ya que sólo necesita analizar los fragmentos afectados por el movimiento tentativo ($i, j, i-1$ y $j+1$). Esto significa, remover el puntaje de superposición de los fragmentos afectados a la solución actual y agregar a Δ_f el puntaje correspondiente a la solución modificada (ecuaciones de las líneas 4-5 del algoritmo 4), además de verificar si algún contig es roto (la primer sentencia if del algoritmo 4) o si dos contigs se unen (la última sentencia if del algoritmo 4) al aplicar el operador de movimiento.

En cada iteración, PALS realiza estos cálculos para todos los posibles movimientos, almacenado los movimientos candidatos en una lista L. Son candidatos aquellos movimientos que reducen el número de contigs. Una vez culminado este paso se selecciona un movimiento de la lista L y se lo aplica generando una nueva solución. El algoritmo llega a su fin cuando no es posible generar más movimientos candidatos.

Algoritmo 4 Función CalcularDelta

$\Delta_c = 0$;
$\Delta_f = 0$;
{Calcula la variación en el solapamiento}
$\Delta_f = \Delta_f$ - $w_{s[i-1],s[i]}$ - $w_{s[j],s[j+1]}$; {Remueve el puntaje de solapamiento de la solución actual}
$\Delta_f = w_{s[i-1],s[j]} + w_{s[i],s[j+1]}$; {Suma el puntaje de solapamiento de la solución modificada}
{Verifica si un contig es roto, y si es así, incrementa el número de contigs}
if ($w_{s[i-1],s[i]}$ > cutoff) || ($w_{s[j],s[j+1]}$ > cutoff) **then**
  $\Delta_c = \Delta_c + 1$;
end if
if ($w_{s[i-1],s[j]}$ > cutoff) || ($w_{s[i],s[j+1]}$ > cutoff) **then**
  $\Delta_c = \Delta_c$ - 1;
end if
return Δ_c, Δ_f;

4.2. Metaheurísticas basadas en población

Aquí se describen las características de diseño del ensamblador metaheurístico basado en población utilizado en esta tesis. Se trata de los algoritmos genéticos pertenecientes a la familia de los algoritmos evolutivos. Por ende, se explican los GAs en el contexto de los EAs.

4.2.1. Algoritmos evolutivos

Las técnicas de la computación evolutiva son métodos estocásticos que emulan el proceso evolutivo de las especies naturales. La idea de diseñar métodos de simulación para resolver problemas utilizando los conceptos de auto-replicación, auto-modificación y la evolución se propuso a principios de la década de 1960 por los pioneros en la ciencia de la información y la inteligencia artificial, tales como, Von Neumannn, Barricelli, Rechenberg y otros. Sin embargo, la primer propuesta algorítmica del método evolutivo data de 1975 y fue realizada por Holland en [90], quien sugirió un procedimiento de búsqueda genética y presentó los antecedentes de la computación evolutiva.

Entre una amplia gama de métodos metaheurísticos modernos para optimización, los algoritmos evolutivos (*Evolutionary Algorithms*, EAs) se han convertido en métodos flexibles y robustos para la resolución de problemas complejos de optimización en muchas áreas de

aplicación como la Industria, las Matemáticas, la Economía, las Telecomunicaciones y la Bioinformática, entre otras [19, 50, 77].

Los EAs son técnicas de optimización que trabajan sobre poblaciones de soluciones y que están diseñadas para buscar valores óptimos en espacios complejos. Están basados en procesos biológicos que se pueden apreciar en la naturaleza, como la selección natural [47] o la herencia genética. Parte de la evolución está determinada por la selección natural de individuos diferentes compitiendo por recursos en su entorno. Algunos individuos son mejores que otros y es deseable que aquellos individuos que son mejores sobrevivan y propaguen su material genético.

La reproducción sexual permite el intercambio del material genético de los cromosomas, produciendo así descendientes que contienen una combinación de la información genética de sus padres. Éste es el operador de recombinación utilizado en los EAs, también llamado operador de cruce. La recombinación ocurre en un entorno en el que la selección de los individuos que tienen que emparejarse depende, principalmente, del valor de la función de *fitness* del individuo, es decir, qué tan bueno es el individuo comparado con los de su entorno.

Como en el caso biológico, los individuos pueden sufrir mutaciones ocasionalmente (operador de mutación). La mutación es una fuente importante de diversidad para los EAs. En un EA, se introduce normalmente una gran cantidad de diversidad al comienzo del algoritmo mediante la generación de una población de individuos aleatorios. La importancia de la mutación, que introduce aún más diversidad mientras el algoritmo se ejecuta, es objeto de debate. Algunos se refieren a la mutación como un operador de segundo plano, que simplemente reemplaza parte de la diversidad original que se haya podido perder a lo largo de la evolución, mientras que otros ven la mutación como el operador que juega el papel principal en el proceso evolutivo.

Un EA procede de forma iterativa mediante la evolución de los individuos pertenecientes a la población actual. Esta evolución es normalmente consecuencia de la aplicación de operadores estocásticos de variación sobre la población, como la selección, recombinación y mutación, con el fin de calcular una generación completa de nuevos individuos. El criterio de terminación consiste normalmente en alcanzar un número máximo de iteraciones (pro-

gramado previamente) del algoritmo, o encontrar la solución óptima al problema (o una aproximación a la misma) en caso de que se conozca de antemano.

Ahora se pasará a analizar detalladamente el funcionamiento de un algoritmo evolutivo, cuyo pseudo-código se muestra en el algoritmo 5. El algoritmo evolutivo es probabilístico y mantiene una población de individuos, $P(t) = \{x_1^t, \ldots, x_\mu^t\}$ en la iteración t, donde μ es el tamaño de la población [125]. Cada individuo (o cromosoma) representa una solución potencial al problema en cuestión, y, un algoritmo evolutivo es implementado como alguna estructura de datos S. Cada solución x_i^t, es evaluada para dar alguna medida de su *fitness*. Entonces, una nueva población (iteración $t+1$) es formada al seleccionar los individuos con mejor *fitness* (`SeleccionarNuevaPoblación`). Algunos miembros de la nueva población sufren transformaciones por medio de operadores "genéticos" (`Alterar`) para construir nuevas soluciones. Existen transformaciones unarias m_i (mutación), que crean nuevos individuos al realizar pequeños cambios en un único individuo ($m_i : S \to S'$), y transformaciones de orden más alto c_j (crossover), que generan nuevos individuos al combinar partes de (dos o más) soluciones distintas ($c_j : S \times \ldots \times S \to S'$) previamente elegidas (`SeleccionarPadres`). Después de algunas generaciones el programa converge, esperándose que el mejor individuo represente una solución razonablemente cercana al óptimo.

Algoritmo 5 Algoritmo Evolutivo

```
t = 0;
Inicializar(P(t));
Evaluar(P(t));
while (no se cumpla la condición de terminación do
   P'(t)=SeleccionarPadres(P(t));
   P'(t)=Alterar(P'(t));
   Evaluar(P'(t));
   P(t+1)=SeleccionarNuevaPoblación(P(t),P'(t));
   t = t + 1;
end while
return la mejor solución;
```

Como puede verse, el algoritmo comprende tres fases principales: selección, reproducción y reemplazo; las cuales son detalladas a continuación:

- ***Selección:*** partiendo de la población inicial $P(t)$ de μ individuos, se crea una nueva población temporal ($P'(t)$) de λ individuos. Generalmente los individuos más aptos (aquellos correspondientes a las mejores soluciones contenidas en la población) tienen un mayor número de instancias que aquellos que tienen menos aptitud (selección natural). Existen diversos mecanismos de selección de individuos para la reproducción. Ellos incluyen a:

  - *Selección proporcional* o *Selección por ruleta* [90]. La idea básica es determinar la probabilidad de selección para cada cromosoma proporcionalmente a su valor de *fitness*. Los individuos son ubicados en segmentos continuos de una recta, de forma tal que el segmento correspondiente a cada individuo es proporcional en tamaño a su *fitness*. Se genera un número aleatorio y se selecciona el individuo cuyo segmento alcanza dicho número. El proceso se repite hasta que se obtiene el número deseado de individuos.
  - *Muestreo estócastico universal* (*Stochastic Universal Sampling* - SUS) [19, 20]. Aquí también, los individuos son ubicados en segmentos continuos sobre una línea, de manera tal que el segmento correspondiente a cada individuo es igual en tamaño a su *fitness*. Los punteros a cada segmento están separados en distancias iguales sobre la línea dependiendo de la cantidad de individuos a ser seleccionados.
  - *Métodos de ranking*. En este tipo de selección las soluciones son ordenadas de mejor a peor y las probabilidades de selección son fijas durante todo el proceso de evolución.
  - *Selección por torneo* dada por Goldberg [76]. Este método aleatorio elige un conjunto de individuos (el tamaño de este conjunto se denomina tamaño del torneo) y toma el mejor de ellos para formar el conjunto de padres para la reproducción. Un tamaño de torneo común es 2 y se denomina *torneo binario*.

- ***Reproducción:*** en esta fase se aplican los operadores reproductivos a los individuos de la población P' para producir una nueva población. Típicamente, esos operadores se corresponden con la recombinación, o crossover, de parejas y con la mutación de los nuevos individuos generados. En general, estos operadores de variación no son

deterministas, es decir, no siempre se tienen que aplicar a todos los individuos y en todas las generaciones del algoritmo, sino que su comportamiento está determinado por su probabilidad asociada.

- ***Reemplazo:*** finalmente, los individuos de la población original son sustituidos por los individuos recién creados. Este reemplazo usualmente intenta mantener los mejores individuos (*elitismo*) eliminando los peores. Dependiendo de si para realizar el reemplazo se tiene en cuenta la antigua población, $P(t)$, se pueden obtener dos tipos de estrategia de reemplazo:

  1. $(\mu; \lambda)$ si el reemplazo se realiza utilizando únicamente los individuos de la nueva población $P(t)$.
  2. $(\mu + \lambda)$ si el reemplazo se realiza seleccionando μ individuos de la unión de $P(t)$ y $P(t+1)$.

  De acuerdo con los valores de μ y λ es posible definir distintos esquemas de selección:

  1. *Selección por estado estacionario (steady-state).* En este tipo de selección $\lambda = 1$, es decir que únicamente se genera un hijo en cada paso de la evolución.
  2. *Selección generacional.* En este caso $\lambda = \mu$ y se está frente a una selección por generaciones en la que genera una nueva población completa de individuos en cada paso.
  3. *Selección ajustable.* Cuando $1 \leq \lambda \leq \mu$ se tiene una selección intermedia en la que se calcula un número ajustable (*generation gap*) de individuos en cada paso de la evolución. Los anteriores son casos particulares de éste.
  4. *Selección por exceso.* Si en cambio $\lambda > \mu$ se trata de una selección por exceso típica de los procesos naturales reales.

Estos algoritmos establecen un equilibrio entre la explotación de buenas soluciones (fase de selección) y la exploración de nuevas zonas del espacio de búsqueda (fase de reproducción), basado en el hecho de que la política de reemplazo permite la aceptación de nuevas soluciones que no mejoran necesariamente las existentes. A efectos prácticos, es fundamental para

el buen funcionamiento de un EA tener controlada en todo momento la diversidad de la población. Se entiende a la diversidad en sentido general como "diversidad de individuos" y en particular como "diversidad de aptitudes".

Con poca diversidad de individuos hay poca variedad de segmentos, a causa de ello el operador de recombinación pierde casi por completo la capacidad de intercambio de información útil entre individuos, contribuyendo así al estancamiento de la búsqueda. La necesidad de tener controlada la diversidad de aptitudes radica en la imposibilidad práctica de trabajar con una población infinita. Como resultado final la búsqueda se estanca y, la situación puede empeorar si la población es finita llevando al algoritmo a una convergencia prematura. Resumiendo, para que la selección sea efectiva, la población debe contener en todo momento una cierta variedad de aptitudes.

Por otra parte cuando la población es finita -es decir, siempre- tampoco se puede tener gran disparidad de aptitudes, pues ello suele afectar muy negativamente a la diversidad de la población.

Distintos sistemas evolutivos surgen a partir de esta idea de algoritmo evolutivo. Las principales diferencias entre ellos están ocultas en un nivel inferior: el uso de una estructura de datos apropiada (para la representación del cromosoma) junto con un conjunto expandido de operadores genéticos. Además de los algoritmos géneticos, que se describen en la próxima sección, existen muchas variantes de algoritmos evolutivos, entre las que se encuentran:

- ***Las estrategias evolutivas*** han sido desarrolladas como un método para resolver problemas de optimización de parámetros [48, 160]; consecuentemente, un cromosoma representa a un individuo como un par de vectores, $v = (x, \sigma)$. Donde x simboliza un punto en el espacio de búsqueda y σ es un vector de desviaciones estándares. La idea principal detrás de estas estrategias es permitir el control de los parámetros, tal como, la varianza de la mutación; para así auto-adaptarse en lugar de modificar sus valores por algún algoritmo determinístico.

- ***La programación evolutiva*** ha sido desarrollada por Fogel [67]. Apunta a la evolución de la inteligencia artificial en el sentido de desarrollar la habilidad de predecir cambios en un medio ambiente. Dicho contexto ha sido descripto como una secuencia

de símbolos (desde un alfabeto finito) y un supuesto algoritmo evolutivo que produce un nuevo símbolo, como salida. La salida debería maximizar la función de retribución (*payoff*), la cual mide la exactitud de la predicción. La idea de la programación evolutiva es evolucionar tales algoritmos.

- ***La programación genética*** ha sido desarrollada por Koza [106, 107]. Propone que el programa deseado debería evolucionarse a sí mismo durante un proceso de búsqueda. En otras palabras, en vez de resolver un problema y construir un programa evolutivo para solucionarlo, sería necesario buscar el espacio de programas computacionales posibles para resolver el problema de la mejor manera.

4.2.1.1. Algoritmos genéticos

Los algoritmos genéticos (*Genetic Algorithms*, GAs) son un tipo de algoritmo evolutivo pensado inicialmente para trabajar con soluciones representadas mediante cadenas binarias denominadas cromosomas. No obstante, a lo largo de los años se han usado otras representaciones no binarias, como permutaciones [120], vectores de enteros [56], reales [88] e incluso estructuras de datos complejas y muy particulares de problemas determinados [175]. Los GAs son los más conocidos entre los EAs, esto ha provocado la aparición en la literatura de gran cantidad de variantes, de forma que es muy difícil dar una regla clara para caracterizarlos. A pesar de esto, podemos decir que la mayoría de los GAs se caracterizan por poseer una representación lineal de las soluciones, no suelen incluir parámetros de auto-adaptación en los individuos y dotan de mayor importancia al operador de recombinación (*crossover*) que al de mutación.

El *crossover* es un operador $c_{\{p_c\}} : S^2 \rightarrow S$ que con probabilidad p_c selecciona dos padres $\overrightarrow{s} = (s_1, ..., s_l)$ y $\overrightarrow{v} = (v_1, ..., v_l)$, donde l es la longitud del cromosoma, y los recombina para formar dos nuevos individuos. Las probabilidades de crossover comúnmente propuestas son $p_c = 0.6$ [98], $p_c = 0.95$ [81] y $p_c \in [0{,}75, 0{,}95]$ en [158]. El tradicional crossover de un punto introducido por Holland elige aleatoriamente una posición de corte $j \in \{1, .., l-1\}$ dentro del cromosoma e intercambia los genes a la derecha de esta posición entre ambos

individuos [90], resultando en:

$$\overrightarrow{s}' = (s_1, ..., s_{j-1}, s_j, v_{j+1}, ..., v_l)$$
$$\overrightarrow{v}' = (v_1, ..., v_{j-1}, v_j, s_{j+1}, ..., s_l)$$

Este operador de *crossover* tiene un desempeño claramente inferior al de otros operadores de *crossover*. El *crossover* de un punto sufre una fuerte dependencia de la probabilidad de intercambio sobre las posiciones de los bits. Existe una asimetría entre la aplicación del *crossover* y la mutación: la unidad básica de la mutación es el gen, la del *crossover* es el individuo; en consecuencia, la probabilidad de que un individuo se cruce no depende de su longitud, mientras que la probabilidad de que contenga genes mutados es más alta cuanto más largo sea.

El operador de mutación tradicional ha sido introducido por Holland como un operador secundario que ocasionalmente cambia un único bit del individuo al invertirlo [90]. La probabilidad de mutación p_m, por bit, pertenece al intervalo $[0, 1]$ y es usualmente muy baja en los GAs. Algunas configuraciones comunes son $p_m = 0.001$ [98], $p_m = 0.01$ [81] y p_m en el intervalo $[0.005, 0.01]$ en [158].

Mientras que el operador de *crossover* combina segmentos útiles de diferentes padres con el fin de obtener cadenas más grandes de alto *fitness*, la mutación sirve como un operador para introducir alelos perdidos en la población y así introducir diversificación en la búsqueda.

La forma del operador de mutación, $m_{p_m} : S \to S$, usando la inversión del bit por cada posición que sufre mutación, produce una cadena de bits $\overrightarrow{x}' = (x'_1, ..., x'_l) = m'_{p_m}(\overrightarrow{x})$ de acuerdo a:

$$x'_i = \begin{cases} x_i \;\; si \;\; x_i > p_m \\ 1 - x_i \;\; si \;\; x_i \leq p_m \end{cases} \tag{4.13}$$

donde $x_i \in [0, 1]$. Bajas probabilidades de mutación, garantizan que un individuo producido por mutación no difiera genéticamente demasiado de sus ancestros.

4.3. Conclusiones

En este capítulo se han descripto en detalle los algoritmos metaheurísticos que se utilizan como punto de partida en el diseño de los ensambladores metaheurísticos propuestos en esta

tesis, clasificándolos por el número de soluciones que manejan (basados en trayectoria o en población).

En primer lugar, se introduce el concepto de SA, se definen las probabilidades de transición y la de aceptación de la solución candidata y se describen los tres esquemas, más conocidos, de enfriamiento. Además se muestra y describe el pseudo-código de un algoritmo SA básico.

En segundo lugar, la metaheurística VNS y sus principales variantes son descriptas. A continuación se explica PALS un método de búsqueda local especialmente diseñado para resolver FAP.

Por último, se describen las técnicas evolutivas específicamente los algoritmos genéticos. Esto incluye una pormenorizada explicación de cada uno de los procedimientos (operadores genéticos, de selección y de reemplazo) que forman parte de esta clase de metahuerísticas.

Parte II

Resolución del problema de ensamblado usando metaheurísticas

Capítulo 5

Algoritmos propuestos: partes comunes en su diseño

Para llevar a cabo la implementación de las metaheurísticas elegidas para resolver el problema de ensamblado de fragmentos ha sido necesario considerar determinados puntos de diseño comunes a todas ellas, tales como: representación, función de evaluación, generación de las soluciones iniciales y plataforma de desarrollo del software.

En primer lugar, resulta indispensable diseñar una representación de una solución para FAP que cumpla con las condiciones de completitud, alcanzabilidad y eficiencia que se describen en el capítulo 3. Además, es imprescindible escoger una forma de evaluación apropiada a tal representación.

Por otra parte, se propone un algoritmo de muy bajo costo computacional para obtener *semillas*, esto es, el uso de soluciones iniciales de buena calidad. Con esta pequeña mejora, las metaheurísticas obtendrán soluciones de excelente calidad para las distintas instancias del FAP. Por último, pero no menos importante, es necesario decidir sobre qué plataforma y qué herramientas de software se usarán para desarrollar los algoritmos metaheurísticos.

Igualmente importante es seleccionar el conjunto de instancias del problema sobre las que se estudiará el desempeño de las metaheurísticas. La importancia de una adecuada selección de instancias radica en el hecho que los algoritmos necesitan probarse sobre casos

de diferentes complejidades y compararse con los resultados de otros algoritmos publicados en la literatura afín.

Los puntos de diseño mencionados arriba, las instancias usadas, además de las características comunes del diseño experimental se describen a lo largo de este capítulo. En la primera sección se explica la representación elegida, mientras que en la segunda sección se enuncia la función de *fitness* utilizada. En la tercera sección se describe el método de generación de semillas, también conocido como estrategias de inicialización, y en la cuarta la biblioteca utilizada para desarrollar los algoritmos. En la siguiente sección se introducen las instancias usadas por estos algoritmos, en tanto que en la penúltima se explican las características comunes de la experimentación. Por último, en la sección 5.7 se exponen las conclusiones del capítulo.

5.1. Representación de la solución

Dado que FAP es un problema de optimización combinatoria, en este trabajo, una solución a dicho problema se representa por medio de una permutación de enteros. Esta permutación representa una secuencia de números de fragmentos, donde los fragmentos sucesivos se superponen. Por lo tanto, cada fragmento es representado por un identificador entero único, $f \in \{1, 2, 3, \ldots, n\}$ donde n es el número total de fragmentos y la ubicación de f en la permutación indica la posición de f en la distribución de los fragmentes. En la figura 5.1 se muestra una permutación de 5 fragmentos y dos contigs, donde los tres primeros (4, 2 y 1) forman un contig y los dos últimos conforman el segundo.

El uso de este tipo de representaciones requiere la utilización y/o diseño de operadores específicos para asegurar la legalidad de las soluciones generadas. A fin de mantener una solución legal, las dos condiciones que deben cumplirse son: todos los fragmentos deben estar presentes en la permutación y todos ellos deben aparecer sólo una vez en dicha permutación.

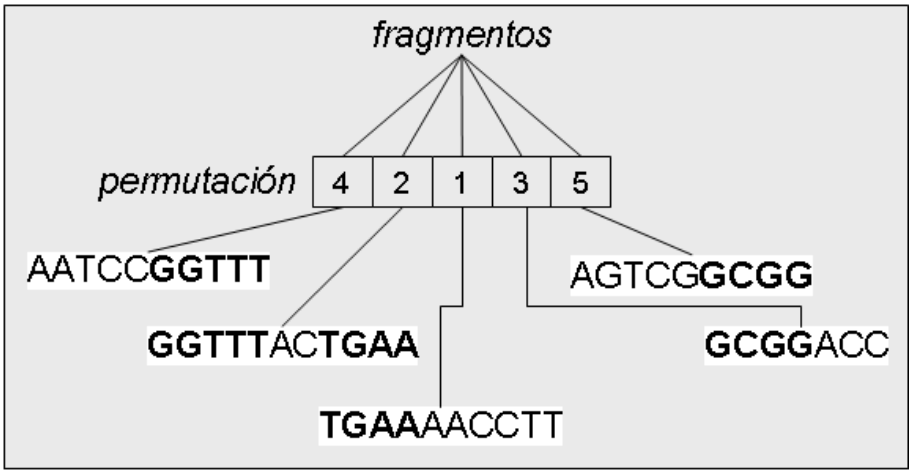


Figura 5.1: Ejemplo de una permutación de 5 fragmentos y 2 contigs.

5.2. Función de evaluación

Como se dijo en el capítulo 2, en la literatura existen diferentes funciones de *fitness* para evaluar el ordenamiento obtenido en la segunda fase de la resolución de FAP, en esta tesis se usa la primera función de evaluación propuesta por Parsons et al. en [138]. Dicha función, $F(S)$, maximiza la suma del puntaje de solapamiento entre fragmentos adyacentes pertenecientes a una solución dada, como se muestra en la ecuación 5.1.

$$F(S) = \sum_{i=1}^{n-1} w(i,j) \tag{5.1}$$

donde $w(i,j)$ corresponde al puntaje de solapamiento entre los fragmentos i y j. El puntaje (w) se calcula usando un algoritmo de alineación semiglobal, esto es, una técnica de programación dinámica clásica usada en la alineación de pares.

5.3. Generación de semillas

Habitualmente, la generación de las soluciones con las cuales una metaheurística inicia su búsqueda es aleatoria. Más aún, por lo general en la literatura este paso no es tratado con

la importancia que corresponde. Comúnmente, durante la búsqueda de soluciones se intenta equilibrar la exploración y la explotación con el fin de evitar una búsqueda aleatoria o, bien, caer en óptimos locales. Sin embargo, considerar "buenas" soluciones iniciales, también denominadas *semillas*, en los primeros pasos del método puede ayudar a aprovechar las regiones prometedoras del espacio de búsqueda. Por lo tanto, atacar el problema clásico de intensificación versus diversificación en esta fase permite lograr un equilibrio más preciso y mejorar la calidad de los resultados finales.

En esta tesis se propone un algoritmo voraz para generar soluciones iniciales. La idea detrás de este enfoque es generar soluciones mediante la adición de componentes de la solución (fragmentos), adecuadamente seleccionados, a una solución parcial, inicialmente vacía. Los pasos a seguir son:

1. Se inicia al generar una solución parcial con un solo fragmento elegido al azar.

2. El siguiente fragmento que insertar debe satisfacer las siguientes condiciones: (i) no debe haber sido insertado aún, (ii) debe formar un contig con el último fragmento insertado y, (iii) si existe más de un fragmento que cumplan las dos condiciones anteriores se escoge aquel que forme el contig de mayor solapamiento.

3. Si no es posible insertar ningún fragmento de acuerdo a las condiciones anteriormente mencionadas, se selecciona al azar uno de los restantes fragmentos disponibles.

4. Los pasos 2) o 3), respectivamente, se repiten hasta que todos los fragmentos formen parte de la permutación.

5.4. Biblioteca MALLBA

La biblioteca MALLBA[1] surge en el seno del Proyecto MALLBA, integrado por grupos de investigación pertenecientes a tres universidades españolas (Málaga, La Laguna y Barcelona). Esta biblioteca ofrece un conjunto de técnicas de resolución para problemas de optimización. Sus cualidades son eficiencia, reusabilidad, genericidad, portabilidad y facilidad de uso [10]. Para lograrlas, MALLBA utiliza una novedosa aproximación a la programación genérica con

[1] http://neo.lcc.uma.es/software/mallba/project.php

objetos. Una visión general de la arquitectura de la biblioteca se muestra en la Figura 1. Como puede apreciarse en la misma, existen tres niveles de interfaz que dan lugar a otras tantas visiones de la misma. A continuación se describirán estas diferentes visiones de la biblioteca.

Todo el código MALLBA ha sido desarrollado en C++; para cada algoritmo de optimización se proporciona un conjunto de clases que, en función de su dependencia del problema objetivo, pueden agruparse en dos categorías: clases provistas y clases requeridas. En la Figura 5.2 se muestra la arquitectura de la biblioteca MALLBA diseñada en UML.

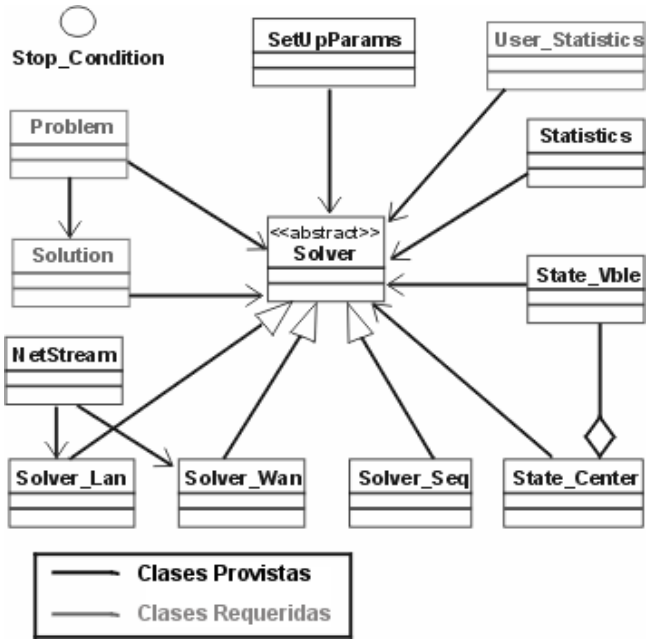


Figura 5.2: Diseño UML de la biblioteca MALLBA.

- ***Clases provistas.*** Las clases englobadas dentro de esta categoría son las responsables de implementar toda la funcionalidad básica del algoritmo correspondiente. En primer lugar, la clase `Solver` encapsula el motor de optimización del algoritmo que se trate. Este motor de optimización es plenamente genérico, interactuando con el problema a través de las clases que el usuario debe proporcionar. La clase `Solver` proporciona métodos para ejecutar el esquema de resolución elegido y métodos para consultar su

progreso o sus cambios de estado. La información que necesita dicha clase es una instancia del problema a resolver y la configuración paramétrica correspondiente. Para que sea posible utilizar diferentes motores de optimización, la clase `Solver` define una única interface y provee varias subclases para llevar a cabo diferentes implementaciones secuenciales y paralelas (`Solver_Seq`, `Solver_Lan` y `Solver_Wan`). En segundo lugar, existe la clase `SetupParams`, encargada de contener los parámetros propios de la ejecución del algoritmo, por ejemplo: el número de iteraciones, el tamaño de la población en un algoritmo genético, el mecanismo de gestión de la cola de subproblemas en un algoritmo de ramificación y poda, etc. Otra clase provista es `Statistics`, cuya finalidad es la recolección de estadísticas propias del algoritmo empleado.

- ***Clases Requeridas.*** Estas clases son las responsables de proporcionar detalles sobre todos los aspectos dependientes del problema a resolver. No obstante, debe reseñarse que a pesar de esta dependencia del problema, la interfaz de las mismas es única y prefijada, permitiendo de esta manera que las clases provistas puedan ser usadas sin necesidad de relacionarse directamente con los detalles del problema. Entre las clases requeridas están la clase `Problem` (que debe proporcionar los métodos necesarios para manipular los datos propios del problema), la clase `Solution` (que encapsula el manejo de soluciones al problema tratado), la clase `UserStatistics` (que permite al usuario recoger aquella información estadística de su interés que no estuviera siendo monitorizada por la clase provista Statistics) así como otras clases que dependen del esqueleto algorítmico elegido (por ejemplo, en el caso de un algoritmo de enfriamiento simulado es necesario especificar el operador de movimiento empleado mediante la clase `Move`).

De esta manera, el usuario de MALLBA se encuentra abocado a implementar las estructuras de datos dependientes del problema, así como de dotar de un comportamiento específico a todos los métodos incluidos en las interfaces de las clases requeridas.

5.5. Instancias de FAP usadas en la literatura

Para llevar a cabo los diferentes experimentos se han elegidos tres secuencias, ampliamente usadas en la literatura, extraídas desde National Center for Biotechnology Information[2] (NCBI): MHC humano clase II con repeticiones de fibronectina tipo II HUMMHCFIB, con número de acceso X60189; una apolipoproteína humana HUMAPOBF, con número de acceso M15421; el genoma completo de bacteriófago lambda, con número de acceso J02459; una secuencia de Neurospora crassa BAC, con número de acceso BX842596 (GI38524243). Se ha usado GenFrag [59] para generar los diferentes conjuntos de datos a partir estas secuencias, que se muestran en la tabla 5.1. GenFrag es una aplicación UNIX/C application creada para aceptar una secuencia de ADN como entrada y generar un conjunto de fragmentos superpuestos como salida, para poder chequear cualquier aplicación de ensamblado de fragmentos.

Tabla 5.1: Información sobre el conjunto de datos. Los números de acceso son usados como nombres de instancias.

Instancias	*Cobertura*	*Longitud media de los fragmentos*	*Número de fragmentos*	*Longitud de la secuencia original*
x60189_4	4	395	39	3835
x60189_5	5	386	48	
x60189_6	6	343	66	
x60189_7	7	387	68	
m15421_5	5	398	127	10089
m15421_6	6	350	173	
m15421_7	7	383	177	
j02459_7	7	405	352	20000
bx842596_4 (o *38524243_4*)	4	708	442	77292
bx842596_7 (o *38524243_7*)	7	703	773	

En la tabla 5.1 se describen las características numéricas de cada instancia usada en esta parte de la tesis. En la primera columna se presenta el nombre con el cual se identifican. Luego se especifica la cobertura, esto es: una medida de la redundancia de los datos del frag-

[2] http://www.ncbi.nlm.nih.gov/

mento (véase sección). En la tercera columna se presenta el número promedio de bases que contienen los fragmentos (longitud media de los fragmentos). A continuación se muestra la cantidad de veces que la secuencia ha sido fragmentada (número de fragmentos). Por último, se indica la longitud de la secuencia original de ADN; es decir, la longitud de la secuencia antes de generar las múltiples copias de la misma. Además, para las dos últimas instancias se especifican dos maneras diferentes de identificarlas: el número de acceso (BX842596) y el número con que NCBI registra la instancia (GI38524243). La razón de esto radica en el hecho que estas dos conjuntos de datos son, habitualmente, citados en la literatura y en este trabajo con el segundo número.

5.6. Características comunes del diseño experimental

Debido a la naturaleza no determinista de los algoritmos estudiados, las comparaciones sobre los resultados de una única ejecución no son consistentes, por lo que deben realizarse sobre un conjunto grande de resultados, obtenidos tras llevar a cabo un elevado número de ejecuciones independientes del algoritmo sobre el problema dado. Para poder comparar los resultados obtenidos es necesario recurrir a estudios estadísticos que permitan asegurar que los resultados y comparaciones presentados son significativos.

Con el fin de obtener resultados estadísticamente significativos, se emplean t-tests o análisis de varianza (ANOVA), según se comparen 2 ó más algoritmos. El uso de estos tests estadísticos permite determinar si los efectos observados en los resultados obtenidos son significativos o si, por el contrario, son debidos a errores en el muestreo realizado. Pero el t-test o el test ANOVA no pueden aplicarse sobre cualquier distribución de datos, puesto que en el caso de una distribución no normal de datos puede ser erróneo el resultado. En este caso, lo correcto sería aplicar el test Mann-Whitney U o el test de Kruskal-Wallis respectivamente.

Para obtener los resultados en esta tesis se han realizado un mínimo de 30 ejecuciones independientes del algoritmo sobre cada instancia del problema dado. En este trabajo se considera en todo momento un nivel de confianza del 95 % (nivel de significancia α del 5 % o la probabilidad p del test por debajo de 0.05) en los tests estadísticos. Esto significa que si $p < \alpha$ los algoritmos presentan diferencias significativas entre sí; de lo contrario se cumple la

hipótesis nula del test indicando que estos son similares. Las conclusiones obtenidas a partir de la aplicación de estos tests se incorporan en las tablas de resultados experimentales, donde el signo '+' muestra que los algoritmos son significativamente distintos y el signo '-', indica lo contrario.

En los casos donde se comparen más de dos algoritmos y, además, la calidad media y el tiempo de estos conjuntos difieran, se efectúan comparaciones múltiples. Las conclusiones arrojadas por tales comparaciones se muestran en las tablas de resultados experimentales, donde por cada instancia se sombrean con un mismo color las celdas correspondientes a opciones algorítmicas estadísticamente similares.

5.7. Conclusiones

En este capítulo se introducen aquellos elementos que son de uso común en todos los algoritmos propuestos en este trabajo. Ellos son: la representación por permutación donde cada fragmento es representado por un entero unívoco, la función de evaluación propuesta por Parsons et al. que maximiza el puntaje de solapamiento, las estrategias de inicialización que utilizan diferentes enfoques heurísticos para generar soluciones iniciales, la biblioteca MALLBA que brinda el marco adecuado para el desarrollo de metaheurísticas, las instancias del problema que son habitualmente utilizadas en la literatura y, por último, los tests estadísticos que permiten corroborar las inferencias realizadas sobre los resultados obtenidos. De esta forma se concentra en este capítulo aquellos conceptos involucrados en la experimentación llevada a cabo en esta tesis.

Capítulo 6

Resolución de FAP usando metaheurísticas basadas en trayectoria

Como se ha dicho anteriormente las metaheurísticas son herramientas eficientes y eficaces para resolver problemas de optimización combinatoria de grandes dimensiones. Por ende, el objetivo es aplicarlas en la resolución del problema de ensamblado de fragmentos, objeto de estudio de esta tesis, que presenta ambas características. Para ello es necesario adaptarlas al problema, comparar sus resultados, analizar sus ventajas y desventajas y realizar estudios estadísticos que den soporte a tales comparaciones y análisis.

En este capítulo se resuelve FAP usando algoritmos basados en metaheurísticas que trabajan con una única solución, a saber: ISA un algoritmo basado en el enfriamiento simulado (SA) y en la mutación por Inversión, PALS una búsqueda local guiada y, por último, FVNS y CVNS basados en la búsqueda en vecindarios variables (VNS). Si bien VNS pertenece a la familia de metaheurísticas basadas en trayectoria y trabaja con una única solución, no recorre estrictamente la trayectoria de dicha solución; sino que recorre una trayectoria distinta en cada vecindario. Por esta razón este capítulo se divide en dos partes: una dedicada a ISA y PALS y otra destinada a FVNS y CVNS. Por último se discuten, en forma

global, los resultados obtenidos por todas las técnicas usadas en este capítulo arribando a una conclusión general.

6.1. Resolución de FAP mediante ISA y PALS

SA es una metaheurística fácilmente adaptable a este tipo de problemas ya que es posible adecuar su diseño a una representación por permutación. Esto significa, por un lado, poder usar como función de evaluación la descripta en el Capítulo 5, pero por otro lado, tener que diseñar una función que genere vecinos legales a partir de la solución actual. Sin embargo, la facilidad de adaptación al problema no es la única razón por la que SA es elegido, sino que también se lo escoge por su capacidad de explorar el espacio de búsqueda en etapas tempranas localizando las regiones más prometedoras para luego explotarlas. De esta forma, nace ISA que permite alcanzar valores de *fitness* muy altos y números de contigs óptimos cuando resuelve FAP.

PALS, en cambio, es un método de búsqueda local diseñado específicamente para resolver FAP, siendo una sola la decisión de diseño a tomar: el criterio de selección de los movimientos a realizar. Dados los buenos resultados reportados en la literatura [5], PALS se convierte en un excelente punto de comparación; motivo por el cual se lo incluye en este trabajo de tesis.

A continuación se describen los detalles de diseño correspondientes a cada uno de estos algoritmos:

- ***ISA.*** El diseño de la función usada para crear una solución vecina a partir de la actual se basa en el procedimiento de la mutación por inversión, operador comúnmente usado en problemas de optimización combinatoria. Entonces, un vecino se genera al seleccionar aletoriamente dos posiciones dentro de la permutación actual e invertir la sub-permutación entre estas dos posiciones, como se ilustra en la Figura 6.1.

  El decaimiento de la temperatura se planifica usando el enfriamiento proporcional descripto en la ecuación 4.7. En tanto que el número de iteraciones entre dos cambios consecutivos de temperatura (longitud de la cadena de Markov) es fijado en 10.

- ***PALS.*** La opción de PALS implementada en este trabajo selecciona los mejores movimientos de la lista L para realizar los cambios en la solución actual. Esto significa

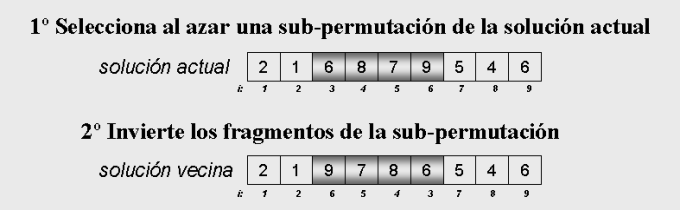


Figura 6.1: Ilustración de la función generadora de soluciones vecinas.

que los movimientos que se deben aplicar son elegidos considerando primero los valores de Δ_c que minimicen el número de contigs y luego los respectivos valores de Δ_f que maximicen la función de *fitness*.

Finalmente, es necesario determinar el criterio de terminación para ambos algoritmos. En este caso, el número de evaluaciones de la función de *fitness* y él de iteraciones no son considerados medidas equitativas para comparar ambos algoritmos. Dado que ISA realiza una evaluación en cada iteración y PALS no realiza ninguna durante el proceso de búsqueda. En el caso del número de iteraciones se puede observar que el esfuerzo computacional realizado por ambas durante una iteración difiere enormemente; ya que ISA solamente invierte los fragmentos de una sub-permutación, mientras que, PALS necesita comparar cada fragmento con cada uno de los que se ubican a continuación de éste. En cambio, si el tiempo de ejecución es el criterio de terminación, se establece el mismo límite al esfuerzo computacional empleado por cada algoritmo. Motivo por el cual se decide asignar a ambos algoritmos el mismo tiempo de ejecución, un máximo de 60 segundos. De esta forma, es posible realizar una comparación justa entre ambas técnicas.

6.1.1. Análisis de resultados

En esta sección se presenta la experimentación realizada con el objetivo de analizar y comparar el comportamiento de estas dos técnicas metaheurísticas a la hora de resolver las instancias del problema de ensamblado de fragmentos presentadas en el capítulo 5. Con el fin de ofrecer resultados con soporte estadístico, por cada algoritmo se realizan 30 ejecuciones independientes usando los parámetros mostrados en la tabla 6.1. Para lo cual se usan computadoras con procesadores AMD Phenom (64 bits) a 2.4 GHz y 2 GB de RAM, con

un sistema operativo perteneciente la distribución Slackware de Linux con una versión del kernel 2.6.27.7-smp.

Tabla 6.1: Valores paramétricos usados por ISA y PALS.

	Parámetro	***Valor***
ISA	Longitud de la cadena de Markov	10
	Temperatura Inicial	0.99
PALS	Selección del movimiento	Los mejores movimientos
ISA y PALS	Condición de terminación	60"

Los resultados se analizan considerando los siguientes dos aspectos: calidad de las soluciones y el esfuerzo computacional. Por lo cual en la tablas 6.2 y 6.3 se presentan datos sobre: el mejor valor de *fitness* hallado y el promedio del mismo, el porcentaje de veces que se encontró el número óptimo de contigs, y el tiempo promedio (segundos) en que se tardó en encontrar la mejor solución. En estas tablas también se incorporan los resultados arrojados por el Test estadístico Mann-Whitney U.

Tabla 6.2: Resultados experimentales de ISA y PALS. Los mejores valores están remarcados en negro.

Instancias	*Mejor fitness*		*Fitness medio*		
	ISA	PALS	ISA	PALS	Test Mann-Whitney U
x60189_4	**11478**	**11478**	11332.90	**11416.17**	+
x60189_5	**14027**	14021	**13872.40**	13758.47	+
x60189_6	**18301**	**18301**	**18147.93**	17890.43	+
x60189_7	**21271**	21210	**20913.10**	20832.83	-
m15421_5	**38583**	38526	**38474.17**	38402.30	+
m15421_6	**48048**	**48048**	47891.70	**47925.13**	-
m15421_7	55048	**55067**	**54702.47**	54525.20	+
j024589_7	**116257**	115320	**115164.87**	114575.13	+
82524243_4	**226538**	225783	**225647.07**	224833.10	+
82524243_7	436739	**438215**	433482.17	**436645.40**	+
Promedio	**98629.00**	98596.90	97962.88	**98080.42**	

En primer lugar se estudia el comportamiento de estos ensambladores desde el punto de vista de la calidad de las soluciones, observándose diferentes hechos que muestran la superioridad de ISA con respecto a PALS para resolver este conjunto de instancias de FAP, a saber:

- Los valores de *fitness* máximos obtenidos por ISA son mejores que los encontrados por PALS en el 50 % de las instancias, en un 30 % obtienen el mismo valor y en el 20 % restante PALS mejora a ISA. Al considerar la media de los valores máximos de *fitness* obtenidos por ambos, ISA supera a PALS en un 70 % de las instancias. En tanto que en el 30 % (3 instancias) restante PALS encuentra mejores *fitness* pero, sorprendentemente, en dos de estos tres casos no logra encontrar un único contig. Por lo que un mejor valor de *fitness* no siempre implica un menor número de contigs, como puede observarse en la figura 6.2 para la instancia *38524243_7*; donde cada barra representa el valor de *fitness* hallado para la mejor solución en cada ejecución de PALS y el número en el eje horizontal la cantidad de contigs asociada. Por último, se verifica que las diferencias detectadas entre ISA y PALS son estadísticamente significativas.

- ISA obtiene una distribución de los fragmentos óptima (un único contig) en todas las instancias y en todas las ejecuciones. Esta situación no se repite con el uso de PALS, ya que en dos instancias, *m15421_5* y *j02459_7* encuentra 1 único contig sólo en el 96 y 86 % de las ejecuciones, respectivamente, y en otras cuatro, *m15421_6*, *m15421_7*, *38524243_4* y *38524243_7*, no lo logra en ejecución alguna. Además, PALS, solamente alcanza el 100 % de los éxitos en las instancias de menor complejidad y sólo en estos casos logra resultados estadísticamente similares a los alcanzados por ISA. Por todo esto se deduce que ISA no presenta problemas a la hora de incrementar la complejidad del problema, situación que no es bien resuelta por PALS.

En un segundo lugar, se analiza el comportamiento de estos ensambladores teniendo en cuenta el tiempo de ejecución. Si se examina el tiempo promedio que se requiere para encontrar la mejor solución en las 3 últimas columnas de la tabla 6.3, se detecta que PALS necesita menos tiempo de cómputo que ISA para encontrar sus mejores permutaciones en todas las instancias, diferencia que resulta estadísticamente significativa. Pero, si además se

Tabla 6.3: Resultados experimentales de ISA y PALS (cont.). Los mejores valores están remarcados en negro.

Instancias	*%Contigs óptimos*			*Tiempo de la mejor solución*		
	ISA	PALS	Test Mann-Whitney U	ISA	PALS	Test Mann-Whitney U
x60189_4	**100.00**	**100.00**	-	0.01	**0.00**	+
x60189_5	**100.00**	**100.00**	-	0.04	**0.00**	+
x60189_6	**100.00**	**100.00**	-	0.10	**0.01**	+
x60189_7	**100.00**	**100.00**	-	0.07	**0.00**	+
m15421_5	**100.00**	96.67	-	0.54	**0.03**	+
m15421_6	**100.00**	0.00	+	0.79	**0.07**	+
m15421_7	**100.00**	0.00	+	1.15	**0.09**	+
j024589_7	**100.00**	86.67	+	8.77	**0.69**	+
82524243_4	**100.00**	0.00	+	16.88	**1.23**	+
82524243_7	**100.00**	0.00	+	30.46	**1.64**	+
Promedio	**100.00**	58.33		5.88	**0.38**	

considera que el tiempo total de ejecución de ambos algoritmos es igual y que la calidad de los resultados favorece a ISA, es posible inferir que PALS converge rápidamente a óptimos locales en las instancias de mayor complejidad.

Por último y para cerrar este estudio se presenta la tabla 6.4, donde se resume cuantitativamente todo el análisis anterior a partir de lo cual ISA se considera mejor ensamblador que PALS para este conjunto de instancias. El puntaje asociado a cada algoritmo se obtiene usando un promedio ponderado donde el 50 % del peso total corresponde al conjunto de las tres medidas de calidad (número de instancias donde: el *fitness* más alto, el mayor *fitness* medio, el 100 % de los contigs óptimos son hallados por cada algoritmo), y el 50 % restante al porcentaje de reducción del tiempo empleado para hallar la mejor solución con respecto al más lento (en este caso ISA).

Tabla 6.4: *Categorización* de los ensambladores metaheurísticos propuestos en esta sección.

Algoritmo	*Fitness máx.*	*Mejor fitness medio*	*100 % contigs óptimos*	*Reducción de tpo.*	*Puntaje*	*Rango*
ISA	**8**	**7**	**10**	0.00 %	4.17	1^o
PALS	5	3	4	**93.00 %**	2.46	2^o

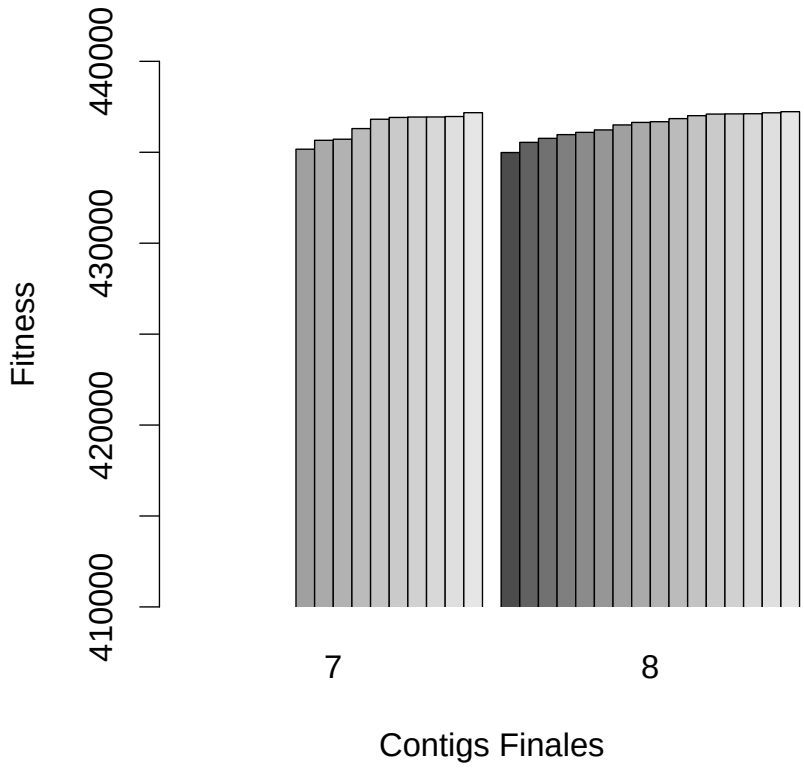


Figura 6.2: *Fitness* y número de contigs obtenidos por PALS para la instancia *38524243_ 7*.

6.1.2. Discusión

ISA se adapta al problema en cuestión usando una representación por permutación y como operador de movimiento se utiliza el procedimiento por inversión. En tanto que PALS ha sido creada para resolver este problema y por lo tanto no necesita adecuación alguna.

En general, las pruebas estadísticas indican diferencias importantes entre las soluciones obtenidas por ambos algoritmos, en especial si se comparan los valores de *fitness* y el tiempo mínimo requerido para encontrar aquellas soluciones con mejores valores de *fitness*. En cuanto al número de contigs finales, las diferencias se encuentran en las instancias de mayor tamaño y complejidad. Estas diferencias favorecen ampliamente a ISA, ya que es capaz de encontrar la distribución óptima de fragmentos en todas las instancias y PALS no. También

obtiene mejores valores de *fitness* en más de la mitad de las instancias. Aunque los tiempos requeridos para hallar estos valores son superiores a los empleados por PALS, ISA no se estanca en óptimos locales como sí lo hace PALS.

6.2. Resolución de FAP mediante FVNS y CVNS

La búsqueda en vecindarios variables brinda el marco adecuado para implementar un algoritmo ensamblador; en particular, la versión de VNS básica. Desde la cual surgen dos algoritmos ensambladores, FVNS y CVNS, que utilizan un mismo marco de trabajo; por ende comparten: el método de creación las estructuras de vecindarios, el procedimiento de generación de la solución inicial, el número de vecindarios y el algoritmo de búsqueda local. La diferencia radica en la función de optimización utilizada, esto es: FVNS maximiza la función de *fitness* presentada en el capítulo 5 y CVNS minimiza el número de contigs usando la estimación propuesta en [8].

En el algoritmo 6 se muestra el pseudo-código que encuadra a estas dos versiones y a continuación se describen los detalles de las mismas:

- *Estructura de los vecindarios.* Los vecinos pertenecientes a un mismo vecindario son generados al intercambiar los fragmentos entre dos posiciones de una solución inicial. Donde el primer fragmento seleccionado y su ubicación dentro de la permutación inicial son las características que identifican a un vecindario. Este fragmento es elegido aleatoriamente al inicio de cada ejecución.

- *Número de vecindarios*, k_{max}. La cantidad de vecindarios varía de acuerdo al tamaño de cada instancia. Es decir, es proporcional al número de fragmentos de cada instancia. De esta forma, el proceso de VNS crea más subespacios de búsqueda cuando el número de fragmentos aumenta. Esta característica permite adecuar el esfuerzo computacional de VNS y su eficacia a la complejidad del problema, diversificando la búsqueda cuando la complejidad aumenta. Esta idea surge después de muchas pruebas donde se comprobaron diferentes maneras de establecer el número de vecindarios. Una opción sería usar el tamaño total de la permutación como número de vecindarios pero, el tiempo de ejecución en las instancias más grandes crece de manera desproporcionada y esto

Algoritmo 6 Algoritmo base de FVNS y CVNS

```
{Inicialización:}
Generar solución inicial x;
Establecer max como porcentaje del número de fragmentos;
Seleccionar el conjunto de estructuras de vecindarios N_k, k = 1, ..., k_max;
iter = 0;
while no se cumpla la condición de terminación do
  k = 0;
  while (k < k_max) and (iter < iter_max) do
    {Agitación:}
    Generar aleatoriamente x' ∈ N_k(x);
    {Búsqueda Local:}
    Obtener el óptimo local x'' al aplicar la heurística 2−opt a x';
    {Mover o no:}
    if x'' es mejor que x then
      x = x'';
      k = 1;
    else
      k = k + 1;
    end if
  end while
  iter = iter + 1;
end while
```

no se refleja en la calidad de los resultados. Entonces, se decide chequear diferentes porcentajes (10, 20, 25, 50 y 100 %) de tamaños de instancias como número de vecindarios, y finalmente se arribó a que un 10 % es un buen valor compromiso entre calidad y tiempo.

- *Agitación (Shaking)*. Para generar una solución a partir de un vecindario N_i: dos fragmentos pertenecientes a la copia de una solución inicial son intercambiadas. La posición del primer fragmento, como se ha mencionado, representa la estructura del vecindario, mientras que la ubicación del segundo es elegida aleatoriamente en cada

iteración. Esta nueva solución puede modificarse y mejorarse por medio de la utilización de un método de búsqueda local.

- *Búsqueda Local.* Se trabaja con un versión modificada de la heurística 2-opt (ver el Algoritmo 7). La modificación consiste en reducir el número total de iteraciones, dada la complejidad de este método y su aplicación en VNS.

Algoritmo 7 2-opt(x) modificado

```
i = 0;
j = i + 2;
while i < número de fragmentos do
  x'=x; {x es la solución inicial}
  x'_i=x_j;
  x'_j=x_i;
  if x' es mejor que x then
    x=x';
  end if
  j = j + 1;
  if j > (número de fragmentos - 1) then
    i = i + 1;
    j = i + 2;
  end if
end while
return x;
```

- *Función de Fitness*, $F(S)$. $F(S)$ es usada para evaluar y comparar soluciones. En este caso una solución a es mejor que otra b si $F(S)_a > F(S)_b$. En FVNS, se considera a FAP como un problema de maximización con $F(S)$ como función objetivo.

- *Estimación de Contigs*, Δ_c. Alba y Luque, en [8], proponen evaluar la solución candidata considerando si el número de contigs es incrementado o decrementado cuando la búsqueda local es aplicada. De esta manera, sólo una parte de la permutación, modificada por la búsqueda local, es considerada en la evaluación. Esta evaluación, Δ_c, es

muy simple ya que suma 1 si se rompe un contig o resta 1 si se unen dos contigs. Consecuentemente solo se evalúan las posiciones modificadas por el operador de variación. Tomando esto en cuenta, una solución es mejor que otra si su número de contigs es menor. En otras palabras, CVNS considera a FAP como un problema de minimización, donde el número de contigs es la función objetivo.

- *Condición de Terminación.* En el algoritmo se han establecido dos puntos donde son necesarios dos diferentes criterios de terminación. El primer punto está relacionado con la cantidad de veces que el algoritmo itera (primera sentencia *while* en el algoritmo 6). El segundo está relacionado con el número de iteraciones realizadas en cada vecindario (segunda sentencia *while* del algoritmo 6).

  Dadas las características del problema, es necesario fijar el número total de iteraciones en uno para reducir el tiempo empleado en cada ejecución. Para el segundo caso, ha sido necesario limitar el número de veces en que los vecindarios son explorados; dado que la combinación entre el número de vecindarios (proporcional al tamaño de la permutación) y ciertos tamaños de instancias producen bucles extremadamente extensos. Para lo cual se establece un número máximo de iteraciones calculado de la siguiente manera:

$$iter_{max} = (k_{max}/t) + 1 \tag{6.1}$$

  donde k_{max} es el número de vecindarios y t, es una constante entera igual a 10. Esta constante se elige luego de probar sistemáticamente la fragmentación de k_{max} con distintos valores de t.

6.2.1. Análisis de resultados

A continuación se muestran y comparan los resultados obtenidos al ejecutar las dos versiones de VNS propuestas; las cuales usan un 10 % del tamaño de la permutación como k_{max} y 10 como valor de t. En la tabla 6.5 se especifican k_{max} e $iter_{max}$ por instancia. Por cada algoritmo e instancia se realizan 30 ejecuciones independientes. La plataforma de hardware utilizada ha sido un Pentium IV de 2.4 GHz y 1 GB de RAM. En tanto que el

sistema operativo corresponde a la distribución SuSE de Linux con una versión del kernel 2.4.19-4GB.

Tabla 6.5: Valores de los parámetros k_{max} e $iter_{max}$ para cada instancia.

Parámetros	*Instancias*									
	X60189				M15421			J02459	38524243	
k_{max}	5	6	8	8	14	18	19	36	45	78
$iter_{max}$	2	2	2	2	3	3	3	4	5	9

En primer lugar, se considera necesario analizar los dos enfoques algorítmicos propuestos: *FVNS* que optimiza el solapamiento entre los fragmentos adyacentes en el diseño y *CVNS* que optimiza el número de contigs. Para ello, se comparan los resultados desde diferentes puntos de vista y se realizan pruebas estadísticas para corroborar las inferencias aquí presentadas. En la tabla 6.6, se presentan el mejor valor de *fitness* y el número de contigs obtenidos por los algoritmos FVNS y CVNS en las 30 ejecuciones realizadas para cada instancia. En la tabla 6.7 se muestra un resumen de los resultados obtenidos por FVNS y CVNS para todas las ejecuciones en cada caso. Ellos son: la media del valor de *fitness*, el porcentaje de la cantidad óptima de contigs, el promedio del total empleado en tiempo (en segundos) cuando el número óptimo de contigs fue encontrado. Por cada conjunto de datos, se presentan los resultados del test estadístico. Además, se muestra el porcentaje del tiempo reducido por CVNS con respecto a FVNS.

Observando la tabla 6.6 es posible inferir que un buen valor de *fitness* no necesariamente está relacionado con el número óptimo de contigs. Las razones de esto son:

- Para algunas instancias, como las pertenecientes a los conjuntos *x60189*, *m15421_5* y *j02459_7*, ambos algoritmos encuentran el número óptimo de contigs independientemente de si su calidad de *fitness* es alta o no.

- Cuando el algoritmo FVNS encuentra el valor más alto de *fitness* en la instancia *x60189_6*, su correspondiente número de contigs es mayor a uno.

- En otras instancias, como *m15421_5*, CVNS optimiza el número de contigs y sus respectivos valores de *fitness* son menores al mejor encontrado. Esto es claramente

Tabla 6.6: Contraste entre el mejor valor de *fitness* y su respectivo número de contigs, y entre mejor número de contigs y su respectivo *fitness* obtenidos por FVNS y CVNS para todos los casos.

Instancias	*Máximo valor de fitness*		*Mínimo número de contigs*	
	FVNS	CVNS	FVNS	CVNS
x60189_4	**9920**/**1**	2988/**1**	**1** / 8629	**1**/1138
x60189_5	**12714**/**1**	3034/**1**	**1** /12065	**1**/3034
x60189_6	**15757**/2	3343/**1**	**1** /15306	**1** / 970
x60189_7	**18749**/**1**	4892/**1**	**1** /17832	**1**/2305
m15421_5	**33720**/**1**	3418/2	**1** /33495	**1**/2090
m15421_6	**40018**/2	3946/2	2 /38573	2/2862
m15421_7	**46241**/2	4724/2	2 /45375	2/4724
j02459_7	**96816**/2	5716/2	**1** /96517	**1**/5337
38524243_4	**184768**/7	7271/7	6 /182460	6/4336
38524243_7	**356565**/4	7551/4	4/**356565**	2/5513

visible en la figura 6.3, donde las barras muestran el valor de *fitness* obtenidos por CVNS en cada ejecución y el valor debajo de cada grupo de barras corresponde al número final de contigs. Particularmente, en esta figura es posible ver que algunos de los más altos valores de *fitness* están asociados a números de contigs mayores a uno. Por otro lado, los contigs óptimos están relacionados con algunos de los valores más bajos de *fitness*.

Por otra parte, a partir de las tablas (6.6 y 6.7), se observa que FVNS supera significativamente a CVNS en todos los casos cuando se comparan los valores de *fitness*. Esta diferencia es corroborada mediante la aplicación del Test Mann-Whitney U con $\alpha = 0{,}05$, donde $p < \alpha$. Incluso cuando el análisis del número óptimo de contigs, obtenidos por ambos algoritmos (ver tabla 6.7), presenta un comportamiento muy similar. Por ejemplo, para la instancia *x60189_4* ambos enfoques obtienen 1 contig (el óptimo) en cada ejecución, pero para la instancia *x60189_6*, FVNS alcanza el óptimo sólo el 93 % de las veces mientras que CVNS lo encuentra en cada ejecución. En tanto que para *j02459_7* FVNS obtiene 1 contig en más ejecuciones que CVNS (46.67 % contra 33 % respectivamente). Éstas no son diferencias importantes y así lo demuestran los resultados del test estadístico donde los valores de p son mayores que α.

Tabla 6.7: Resultados experimentales de CVNS y FVNS. Los mejores valores están remarcados en negro.

Instancias	*Fitness Medio*			*%Contigs Óptimos*			*Tiempo Total Medio*			
	FVNS	CVNS	Test Mann-Whitney U	FVNS	CVNS	Test Mann-Whitney U	FVNS	CVNS	Test Mann-Whitney U	*% del Tiempo Reducido*
x60189_4	**9290.47**	1549.80	+	**100.00**	**100.00**	–	0.047	**0.002**	+	94.98
x60189_5	**11994.07**	1970.73	+	**100.00**	90.00	–	0.167	**0.005**	+	96.69
x60189_6	**15211.20**	2074.90	+	93.33	**100.00**	–	0.846	**0.020**	+	97.65
x60189_7	**17966.80**	2770.70	+	96.67	**100.00**	–	1.018	**0.022**	+	97.86
m15421_5	**32904.00**	2055.20	+	**66.67**	53.33	–	46.752	**0.472**	+	98.99
m15421_6	**39035.37**	2362.50	+	0.00	0.00	–	230.05	**1.690**	+	99.26
m15421_7	**45596.20**	2832.20	+	0.00	0.00	–	269.07	**1.810**	+	99.32
j02459_7	**95034.50**	3144.80	+	**46.67**	33.33	–	11949.845	**55.759**	+	99.53
38524243_4	**182569.62**	3462.14	+	0.00	0.00	–	48301.810	**170.380**	+	99.64
38524243_7	**356565.00**	5781.10	+	0.00	0.00	–	865222.000	**2528.020**	+	99.70
Promedio	**33379.08**	2345.10		**62.92**	59.59		1999.780	**9.380**		97.20

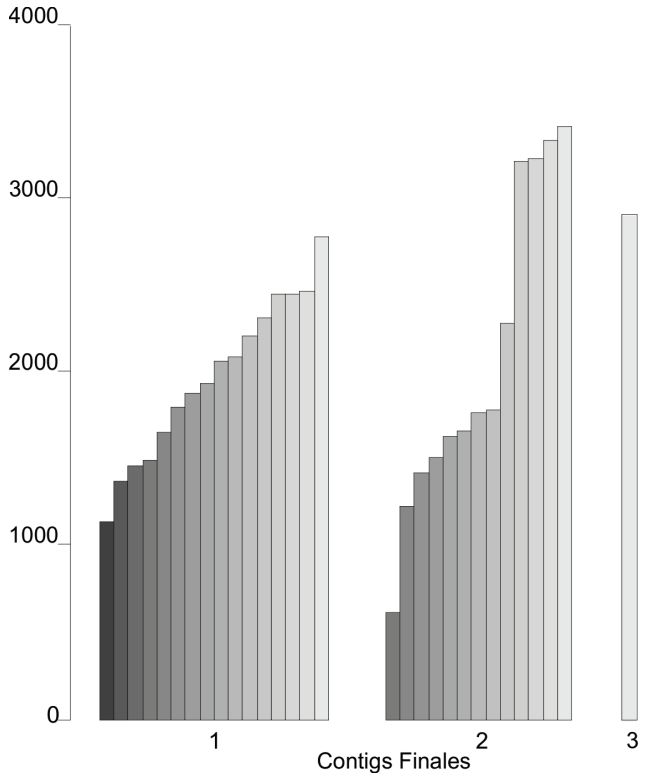


Figura 6.3: *Fitness* y número de contigs obtenidos por CVNS para la instancia *m15421_5*.

Otra cuestión importante a considerar es el tiempo computacional empleado por ambos enfoques. CVNS emplea mucho menos tiempo que FVNS para optimizar el número final de contigs, en promedio, CVNS reduce en un 98 % el tiempo de ejecución con respecto a FVNS, siendo el tiempo medio para obtener un único contig de 56 segundos. Esta diferencia, también, es corroborada estadísitcamente donde $p < \alpha$. La razón principal de esta reducción radica en el número de veces que la función de *fitness*, $F(S)$, es computada por cada algoritmo. Es decir, CVNS sólo calcula el *fitness* en dos oportunidades: al inicio y al final del algoritmo; mientras que el número de evaluaciones realizadas por FVNS es igual a la cantidad de iteraciones realizadas por 2-opt multiplicado por el número de iteraciones de VNS. Esto significa que, como mínimo, FVNS realiza miles de evaluaciones de $F(S)$ dependiendo del tamaño de la instancia.

Finalmente para concluir este análisis se muestra en la tabla 6.8 un resumen cuantitativo de lo anteriormente analizado; a partir de lo cual se considera a FVNS mejor ensamblador que CVNS. Considerando que: ambas técnicas tienen distintas direcciones de optimización, el número óptimo de contigs no siempre implica un valor de *fitness* alto y directa o indirectamente se busca una solución con un único contig, se compara estos algoritmos teniendo en cuenta los contigs óptimos obtenidos por ambos. Entonces el puntaje asociado a cada algoritmo se obtiene usando un promedio ponderado donde el 50 % del peso total corresponde al conjunto de indicadores de calidad (número de instancias donde: el *fitness* más alto, el mayor *fitness* medio, el 100 % de los contigs óptimos son hallados por cada algoritmo), y el 50 % restante al porcentaje de reducción del tiempo empleado para hallar la mejor solución con respecto al más lento (en este caso FVNS).

Tabla 6.8: *Categorización* de los ensambladores metaheurísticos propuestos en esta sección.

Algoritmo	*Fitness máx.*	*Mejor fitness medio*	*100 % contigs óptimos*	*Reducción de tpo.*	*Puntaje*	*Rango*
FVNS	**10**	**10**	2	0.00 %	3.34	1^o
CVNS	0	0	**3**	97.20 %	0.97	2^o

6.2.2. Discusión

Las dos versiones de VNS propuestas están específicamente adaptadas a FAP pero difieren en la orientación de la optimización. Una de ellas, *FVNS*, maximiza la función de *fitness* propuesta por Parsons la cual suma el puntaje de solapamiento entre fragmentos adyacentes en una solución dada. Al maximizar dicho puntaje, a la mejor solución hallada le corresponderá el puntaje más alto. La otra variante, *CVNS*, evalúa la solución candidata considerando, solamente, si el número de contigs es incrementado o decrementado cuando 2-opt realiza un movimiento. De esta forma el objetivo consiste en minimizar el número final de contigs.

La calidad de los resultados obtenidos por ambos algoritmos es muy alta ya que encuentran el número óptimo de contigs en casi todas las instancias, cuyos números de fragmentos varían en un rango de [39..352]. Aunque es importante destacar que CVNS alcanza estos resultados empleando un 98 % menos de tiempo de CPU que FVNS.

6.3. Comparación con otros ensambladores

En esta sección se comparan el desempeño de ISA, PALS y ambos enfoques VNS en contrapartida con los algoritmos ensambladores de la literatura: CAP3 [93] y PHRAP [80]. Dadas las características de CAP3 y PHRAP esta comparación solo puede realizarse contrastando el número de contigs finales. Además, no es posible contrastar los tiempos de ejecución, ya que, en general, los autores no proveen esta información.

En primer lugar, se destaca la superioridad de ISA sobre todos estos ensambladores en todas las instancias; ya que la distribución óptima de fragmentos es una constante para este algoritmo. Luego, se comparan PALS y los enfoques VNS propuestos con el resto de los ensambladores mencionados arriba (ver tabla 10.3). En este sentido, PALS y ambas versions de VNS tienen un comportamiento mejor o igual que PMA, CAP3 y PHRAP.

Tabla 6.9: Mejor número de contigs para los algoritmos ISA, PALS, FVNS, CVNS, y para los otros sistemas especializados. El símbolo − indica que esa información no se proporciona.

	ISA	*PALS*	*FVNS*	*CVNS*	*CAP3*	*PHRAP*
x60189_4	1	1	1	1	1	1
x60189_5	1	1	1	1	1	1
x60189_6	1	1	1	1	1	1
x60189_7	1	1	1	1	1	1
m15421_5	1	1	1	1	1	1
m15421_6	1	2	2	2	2	2
m15421_7	1	2	2	2	1	2
j02459_7	1	1	1	1	1	1
38524243_4	1	6	6	6	-	6
38524243_7	1	3	4	2	-	-

6.4. Conclusiones

En este capítulo se presenta el diseño de cuatro algoritmos metaheurísticos basados en trayectoria utilizados como ensambladores de fragmentos de ADN: ISA, PALS, FVNS y CVNS. Los dos primeros siguen una única trayectoria en el espacio de soluciones durante la búsqueda, mientras que los dos últimos establecen distintas trayectorias al cambiar de vecindarios durante este proceso.

Para categorizar a los ensambladores, aquí propuestos, es necesario comparar sus respectivos resultados experimentales. Para esto, en la tabla 6.10 se resumen los principales datos presentados en las secciones anteriores, contrastando el comportamiento y desempeño de cada uno de ellos. El puntaje asociado a cada algoritmo se desprende del promedio ponderado, donde el 50 % del peso total corresponde al conjunto de las tres medidas de calidad (número de instancias donde: el *fitness* más alto, el mayor *fitness* medio, el 100 % de los contigs óptimos son hallados por cada algoritmo) y el 50 % restante al item que refleja el costo temporal (porcentaje medio de reducción del tiempo mínimo empleado por ISA versus el requerido por PALS, y por FNVS versus CVNS). Del mencionado resumen y de todo lo expuesto en este capítulo es posible observar para este conjunto de instancias que:

Tabla 6.10: *Categorización* de las ensambladores metaheurísticos basados en trayectoria propuestos en este capítulo.

	ISA	PALS	FVNS	CVNS
Fitness Máximo	**8**	5	0	0
Fitness Medio	**7**	3	0	0
100 % Contigs óptimos	**10**	4	2	3
Reducción de Tiempo (ISA vs. PALS y FVNS vs. CVNS)	0.00 %	**93.00 %**	0.00 %	**97.20 %**
Puntaje	**4.17**	2.47	0.67	0.97
Rango	**1°**	**2°**	**4°**	**3°**

- ISA es el ensamblador metaheurístico más eficaz y eficiente presentado en este capítulo. Además, si se considera la evaluación realizada en la sección 6.3 donde ISA es superior a otros ensambladores presentados en la literatura, entonces resulta que ISA confirma la hipótesis **H1** (estado del arte) propuesta en el capítulo introductorio. Esto significa que puede obtener soluciones de mayor calidad que las proporcionadas hasta el momento por otros ensambladores.

- PALS emplea un tiempo de cómputo menor al requerido por ISA para encontrar sus mejores permutaciones en todas las instancias. Pero, si se considera que el tiempo total de ejecución de ambos algoritmos es igual y que la calidad de los resultados obtenidos por ISA es superior, se deduce que PALS converge rápidamente a óptimos locales en las instancias de mayor complejidad.

- La diferencia fundamental entre ambas versiones de VNS se encuentra en el tiempo de ejecución empleado por cada una. Esto se relaciona directamente con el costo computacional a la función de optimización utilizada en cada caso. Por otra parte, FVNS y CVNS sólo logran obtener una distribución óptima de fragmentos en las mismas cinco instancias, quedando así en amplia desventaja con respecto a ISA y PALS.

Capítulo 7

Resolución de FAP usando metaheurísticas basadas en población

Los métodos poblacionales más utilizados para resolver problemas de optimización combinatoria son los algoritmos evolutivos, entre ellos se destacan los algoritmos genéticos. Este tipo de métodos proveen una manera natural e intrínseca para explorar el espacio de búsqueda. Motivos por los cuales el uso de GAs resulta muy atractivo para resolver el FAP. Aún así, el comportamiento de esta clase de algoritmos depende principalmente de cómo es manipulada la población. Por esta razón, se ha decidido estudiar: cómo diferentes estrategias de inicio afectan la obtención de los resultados finales, cómo distintos operadores de recombinación afectan la búsqueda y cómo el desempeño de un GA puede beneficiarse incorporando a otra meteheurística durante la búsqueda.

Para llevar a cabo los dos primeros casos de estudio se desarrollan (sección 7.1) dos nuevas estrategias de inicio usando técnicas heurísticas y se analiza el efecto de distintos operadores de cruce en la resolución del FAP. En un intento de incrementar la calidad de los resultados obtenidos hasta el momento, en la Sección 7.2 se hibrida al GA incorporando una versión de VNS básico como tercer operador genético. En la tercera sección se comparan los resultados obtenidos por los algoritmos propuestos en este capítulo con los obtenidos por los ensambladores propuestos en la literatura. Finalmente, se sintetizan y concluyen los resultados expuestos durante el desarrollo del presente capítulo.

7.1. Resolución de FAP mediante $GA2o_{50}$, $GA2o_{100}$, GAG_{50} y GAG_{100}

Dado que la representación y la formulación del problema de ensamblado de fragmentos de ADN se asemejan a las del problema del viajante de comercio (tal como se explicó en el capítulo 2) y que, además, los GAs han demostrado adaptarse efectivamente a este último [180, 195] resultan, entonces, una opción prometedora para resolver el FAP. Además, esto trae aparejada la ventaja de la reutilización de los operadores genéticos existentes diseñados para una representación por permutación y para el TSP.

A continuación se especifican los detalles de diseño de los GAs propuestos en este trabajo para resolver el FAP:

- *Generación de las soluciones iniciales.* Con el fin de atacar el clásico problema de intensificación-diversificación encontrado en la fase de creación de la población inicial y, por ende, de aumentar la calidad de los resultados finales se proponen tres diferentes estrategias de creación de soluciones iniciales (o semillas): *método aleatorio*, *heurística 2-opt* y *la técnica voraz.*

  1. Método aleatorio. Ésta es la forma tradicional de generar poblaciones, donde los fragmentos son agregados aleatoriamente a la solución.
  2. Heurística 2-opt. Éste es un método de búsqueda local simple que selecciona aleatoriamente dos fragmentos no adyacentes y los intercambia. Este proceso se repite hasta que se satisfaga el criterio de terminación. En este trabajo se aplica la heurística 2-opt a una solución generada al azar.
  3. Técnica voraz. Las soluciones son generadas usando la técnica voraz descripta en el Capítulo 5.

  Tomando como base estas ideas para crear una solución inicial, se proponen cinco diferentes estrategias para generar la población inicial:

  1. Todas soluciones de la población son generadas aleatoriamente. En esta tesis, esta variante se denomina *GA*.

2. La mitad de la población es creada al azar y la otra mitad usando la heurística 2-opt. Esta versión se identifica con el nombre $GA2o_{50}$, donde *2o* representa la heurística aplicada y el número 50 indica el porcentaje de la población generado con 2-opt.
3. La población entera se crea usando la heurística 2-opt y se sigue la misma técnica para identificar a estas versiones, denominándose $GA2o_{100}$.
4. El 50 % de la población es creada aleatoriamente y el resto es generada usando la técnica voraz. Esta versión de GA se identifica como GAG_{50}, donde la segunda G representa a la técnica *greedy* o voraz y el número, al igual que en las dos anteriores, indica el porcentaje de individuos creados con esta técnica.
5. El total de la población es generado por medio de la técnica voraz. Consecuentemente, el nombre de esta versión es GAG_{100}.

- *Operadores de Recombinación.* Con el propósito de hallar un operador de cruce que logre un buen compromiso entre calidad y eficiencia, se analizan diferentes operadores. Todos ellos son operadores de recombinación especialmente diseñados para la representación por permutación y así evitar soluciones ilegales. Los operadores considerados en este estudio son: *Partial Mapped Crossover* (PMX) [75], *Order Crossover* (OX) [49], *Cycle Crossover* (CX) [137], y *Edge Recombination* (ER) [192].

  - *Partial Mapped Crossover.* Goldberg y Lingle presentan PMX en [75]. Es posible ver a este operador como una extensión del crossover de dos puntos. PMX usa un procedimiento de reparación especial para resolver la ilegitimidad causada por el crossover de dos puntos sobre una permutación. Es decir, las bases de PMX son el crossover de dos puntos más un procedimiento de reparación. PMX opera de la siguiente manera:

    1. Selecciona uniformemente dos posiciones a lo largo de la permutación. Las sub-permutaciones definidas por las dos posiciones se denominan secciones de transferencia.
    2. Intercambia dos sub-permutaciones entre los padres para producir 2 hijos.

3. Determina las relaciones de transferencia entre las dos secciones transferidas.
4. Legaliza los hijos con las relaciones de transferencia.

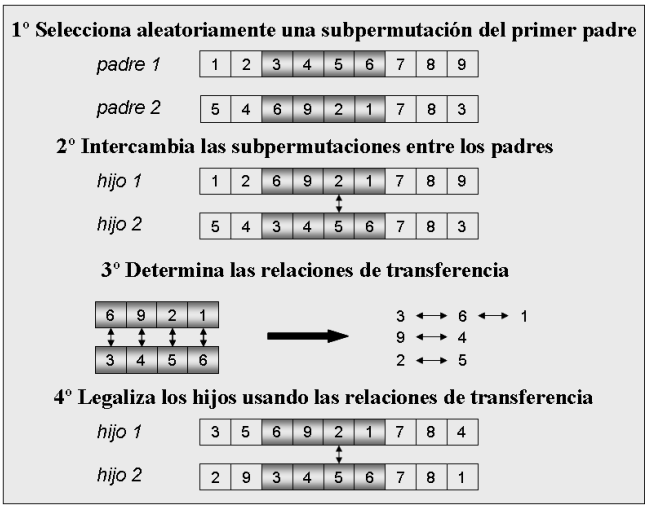


Figura 7.1: Ilustración del operador PMX.

La Figura 7.1 ilustra el procedimiento de PMX. Los fragmentos 1, 2 y 9 están duplicados en el hijo 1, mientras que se pierden los fragmentos 3, 4 y 5. De acuerdo a las relaciones de transferencia determinadas en el paso 3, los fragmentos repetidos 1, 2 y 9 deberían reemplazarse por los fragmentos faltantes 3, 5 y 4, respectivamente, mientras mantiene la sub-permutación intercambiada sin modificaciones.

- *Order Crossover*. OX es propuesto por Davis en [49]. Éste puede verse como una variación de PMX con un procedimiento de reparación diferente. OX trabaja como sigue:

  1. Selecciona al azar una sub-permutación desde un padre.
  2. Produce un hijo al copiar la sub-permutación dentro sus correspondientes ubicaciones.
  3. Borra las ciudades que están en la sub-permutación del segundo padre. La secuencia de ciudades resultantes son las requeridas por los hijos.

4. Ubica los fragmentos en las posiciones libres del hijo de izquierda a derecha, de acuerdo al orden de la secuencia en el segundo padre.

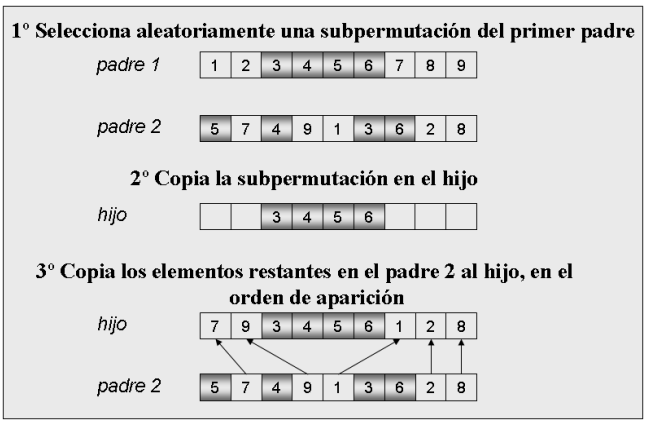


Figura 7.2: Ilustración del operador OX.

La Figura 7.2 ilustra este procedimiento. Ésta muestra un ejemplo de cómo se obtiene un cromosoma hijo. Con idénticos pasos, es posible producir el segundo hijo *(2,5,4,9,1,3,6,7,8)* desde los mismos padres.

- *Cycle Crossover*. Oliver, Smith y Holland proponen CX en [137] y usa algunos de los fragmentos desde un padre y selecciona los fragmentos restantes desde el otro padre. Los fragmentos provenientes del primer progenitor no se seleccionan aleatoriamente, sino que se eligen si definen un ciclo de acuerdo a las posiciones correspondientes entre padres. CX funciona de la siguiente forma:

  1. Encuentra el ciclo definido por las posiciones correspondientes de las ciudades entre los padres.
  2. Copia las ciudades en el ciclo a un hijo con las posiciones correspondientes de un padre.
  3. Determina las ciudades restantes para el hijo al eliminar las ciudades que ya están en el ciclo desde el otro padre.
  4. Completa al hijo con las ciudades restantes.

Este procedimiento se ilustra en la Figura 7.3, siguiendo los mismos pasos se puede crear el segundo vástago.

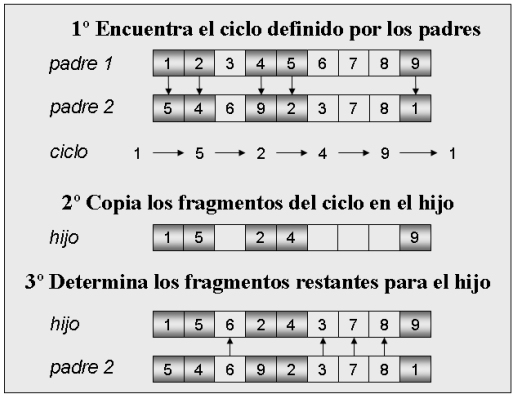


Figura 7.3: Ilustración del operador CX.

- *Edge Recombination.* Whitley, Starweather y Fuquay introducen este operador en [192]. ER transfiere más del 95 % de los arcos desde los padres a un único hijo y explora la información sobre los arcos en una permutación, por ejemplo para la permutación *(3,1,2,8,7,4,6,9,5)* los arcos son: (3 1), (1 2), (2 8), (8 7), (7 4), (4 6), (6 9), (9 5) y (5 3).

  Luego todos los arcos -no los fragmentos- poseen valores (puntaje de solapamiento en el FAP). Entonces la función objetivo, que debe ser maximizada, es el total de arcos que constituyen una permutación legal. La posición de un fragmento en una permutación no es importante. Tampoco lo es la dirección de un arco, porque los arcos (3 1) y (1 3), por ejemplo, indican sólo que los fragmentos 1 y 3 están directamente conectados entre sí. La idea general detrás del crossover ER es que un hijo debería construirse exclusivamente desde los arcos presentes. Esto se realiza con ayuda de la lista de arcos creada desde los dos padres. La lista de arcos provee, por cada fragmento i, todos los otros fragmentos conectados al fragmento i en al menos uno de los padres. En todo grafo, por cada fragmento i existen al menos dos y a lo sumo 4 fragmentos en la lista. Un ejemplo de esto, se observa en la figura 7.4.

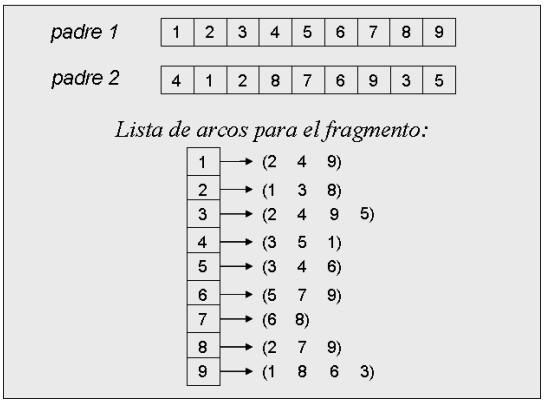


Figura 7.4: Ilustración del operador EX.

La construcción del hijo comienza con la selección de un fragmento de inicio desde uno de los padres. Entonces, se elige el fragmento conectado al inicial con el menor número de arcos. Si estos números son iguales, se realiza una elección aleatoria entre ellos. Cada selección incrementa la posibilidad de completar una permutación con todos los arcos seleccionados desde los padres. Con una selección al azar, la chance de tener un arco erróneo debería ser mucho más grande. Se asume que el fragmento 1 es seleccionado. Este fragmento está directamente conectado con otros tres: 2, 4 y 9. El próximo fragmento se elige desde uno de estos tres. En este ejemplo, los fragmentos 2 y 4 tienen tres arcos, y el 9 tiene 4. Se hace una selección aleatoria entre los fragmentos 2 y 4; se asume que se ha seleccionado el 4 y la permutación parcial resultante es *(1,4)*. Nuevamente los candidatos para el próximo fragmento de la permutación, que se está construyendo, son 3 y 5, ya que ellos están directamente conectados al último fragmento (4), entonces se selecciona el 5. Siendo la permutación parcial resultante: *(1,4,5)*. Continuando con este procedimiento se obtiene el hijo: *(1,4,5,6,7,8,2,3,9)* el cual se forma completamente por arcos tomados desde ambos padres.

- *Operador de Mutación.* En este trabajo se utiliza el bien conocido operardor de mutación *swap*. Este operador, también denominado mutación por intercambio recíproco, selecciona dos posiciones aleatorias y luego intercambia los fragmentos sobre estas posiciones.

- *Operadores de Selección.* Se emplean dos diferentes métodos de selección. La selección por *torneo binario* es utilizada para elegir las soluciones a ser recombinadas. Pero, para construir la población correspondiente a la próxima generación, se utiliza el método $(\mu + \lambda)$.

  La selección por torneo es un método estocástico, introducido por Goldberg en [76], que elige un conjunto de permutaciones (el tamaño de este conjunto se denomina tamaño del torneo) y toma el mejor de ellos para formar el conjunto de padres para la reproducción. Un tamaño de torneo común es 2, y se denomina torneo binario.

  La selección $(\mu+\lambda)$, usada originalmente en las estrategias evolutivas, es introducida por Bäck y Hoffmeister en los GAs [18, 16]. Con esta estrategia μ padres y λ hijos compiten por sobrevivir y los μ mejores son seleccionados para formar la población en la próxima generación.

- *Condición de Terminación.* Se utiliza un número fijo de generaciones (1000) como criterio de terminación. Este valor y la configuración paramétrica aplicada a todas las versiones de GA se han escogido en base a un análisis preliminar realizado sobre diferentes valoraciones de los parámetros. La mencionada configuración se muestran en la Tabla 7.1

7.1.1. Análisis de resultados

A continuación, se presenta una síntesis de los resultados obtenidos por los cinco GAs propuestos (GA, $GA2o_{50}$, $GA2o_{100}$,GAG_{50}, GAG_{100}) con sus variantes (los operadores de recombinación PMX, OX, CX y EX) en todas las instancias del problema. De la misma forma que en el capítulo anterior, el propósito es ofrecer resultados significativos desde el punto de vista estadístico. Por lo tanto, para cada algoritmo, se realizan 30 ejecuciones independientes, utilizando los parámetros que se muestran en la Tabla 7.1. Para llevar a cabo esta experimentación se han usado computadoras con las siguientes características: Pentium 4 de 2.4 GHz y 1 GB RAM; cuyo sistema operativo corresponde a la distribución SuSE de Linux con la versión de kernel 2.4.19-4GB.

Tabla 7.1: Valores paramétricos usados por los diferentes GAs.

Parámetro	***Valor***
μ	512
λ	512
Op. de recombinación	PMX, OX, CX, y EX
y su probabilidad	0.7
Op. de mutación.	*Swap*
y su probabilidad	0.2
Selección de padres	Torneo binario
Reemplazo	Las mejores μ soluciones de $(\mu + \lambda)$
Condición de terminación	1000 generaciones

En la tablas 7.2, 7.3, 7.4, 7.5, 7.6 y 7.7 se muestran los resultados más importantes relacionados con la calidad de la solución (medida en función del *fitness* y el número de contigs), el tiempo de ejecución (expresado en segundos) y los resultados del test Kruskal-Wallis (KW) y de las comparaciones múltiples. En el caso de las comparaciones múltiples se indican sus resultados sombreando con un mismo color las celdas correspondientes a opciones algorítmicas estadísticamente similares. Por ejemplo, en la tabla 7.2 para la instancia *m15421_5*, se distinguen tres grupos disímiles de GAs: uno formado por los que ejecutan PMX y OX, otro constituido por él que usa CX y el último conformado por él que utiliza EX.

Tabla 7.2: Resultados experimentales de GA con los distintos operadores genéticos. Los mejores valores están remarcados en negro.

Instancias	*Mejor fitness*				*Fitness medio*				*Test*
	PMX	OX	CX	EX	PMX	OX	CX	EX	*KW*
x60189_4	10503	10869	10787	**11478**	9488.47	9829.93	9478.15	**11309.93**	+
x60189_5	13097	12871	12482	**13885**	11873.83	12238.34	11693.09	**13556.93**	+
x60189_6	15709	16345	15444	**18301**	14600.28	15178.07	14556.17	**17379.13**	+
x60189_7	19036	19504	18831	**21271**	17306.10	18076.34	17596.10	**20447.31**	+
m15421_5	32307	36976	**38097**	36976	30738.00	35981.27	**37366.71**	35982.72	+
m15421_6	37546	38613	37048	**44690**	35528.57	36673.30	34073.43	**42545.17**	+
m15421_7	44719	44727	**53183**	51480	41608.82	42957.07	**52555.28**	49307.00	+
j02459_7	82060	83102	69428	**94846**	78935.00	80015.28	67455.03	**92296.20**	+
38524243_4	149092	150302	117351	**173126**	144472.63	146251.75	114981.04	**166742.88**	+
38524243_7	234518	223231	153893	**259402**	226791.41	218377.16	148422.85	**254408.20**	+

Desde el punto de vista de la calidad de las soluciones es posible inferir varias conclusiones:

Tabla 7.3: Resultados experimentales de $GA2o_{50}$ y $GA2o_{100}$ con los distintos operadores genéticos. Los mejores valores están remarcados en negro.

Algoritmos	*Instancias*	*Mejor fitness*				*Fitness medio*				*Test*
		PMX	OX	CX	EX	PMX	OX	CX	EX	*KW*
$GA2o_{50}$	*x60189_4*	**11478**	**11478**	10657	**11478**	11375.07	11375.30	10230.27	**11478.00**	+
	x60189_5	14021	14021	**14308**	14027	13798.79	13777.69	13820.63	**13988.20**	+
	x60189_6	**18301**	**18301**	18176	**18301**	18012.31	18044.68	17978.77	**18293.03**	+
	x60189_7	21260	21271	**21324**	21271	20882.07	20934.59	20832.30	**21221.00**	+
	m15421_5	38184	38061	37677	**38563**	37339.28	37491.07	37414.75	**37967.13**	+
	m15421_6	38219	39665	37129	**44582**	36236.37	37870.07	35631.17	**43057.03**	+
	m15421_7	53498	53447	53559	**53838**	52530.07	52647.86	52761.08	**53041.89**	+
	j02459_7	110455	110558	110358	**110767**	109190.69	109513.62	**109927.67**	109274.69	+
	38524243_4	149004	153548	124141	**178233**	144925.75	149706.00	121114.50	**170452.54**	+
	38524243_7	234383	235977	162410	**274040**	228226.08	228907.00	158443.82	**262596.56**	+
$GA2o_{100}$	*x60189_4*	**11478**	**11478**	10776	**11478**	11402.50	11400.30	10354.80	**11478.00**	+
	x60189_5	14021	14027	**14235**	14027	13861.28	13827.34	13821.53	**13986.24**	+
	x60189_6	18301	18301	**18314**	18301	17972.55	17986.17	18011.63	**18286.44**	+
	x60189_7	21271	21197	**21405**	21271	20856.59	20858.41	20865.03	**21213.20**	+
	m15421_5	37992	38014	38004	**38643**	37324.69	37585.34	37834.75	**37931.58**	+
	m15421_6	37920	39402	38011	**44402**	36065.00	37599.20	35799.03	**43158.27**	+
	m15421_7	53206	53616	53382	**54353**	52679.10	52917.38	52880.00	**53148.27**	+
	j02459_7	110263	111023	**111324**	110649	109139.17	**109816.82**	109345.92	109477.07	+
	38524243_4	148469	154611	124677	**176636**	145201.33	150596.63	121239.04	**170726.79**	+
	38524243_7	231556	232641	161698	**266538**	228334.36	229708.94	159218.00	**262233.67**	+

- Si se considera el mejor valor de *fitness* y su media se observa que el operador de cruce EX le permite a *GA* obtener mejores resultados en el 90 % de las instancias (veáse tabla 7.2). Esta situación comienza a cambiar lentamente cuando la generación de la población inicial deja de ser totalmente aleatoria. En $GA2o_{50}$ y en $GA2o_{100}$ se detecta que PMX, OX y CX encuentran mejores *fitness* que EX en el 40 y 50 % de las instancias, correspondientemente (véanse tablas 7.3 y 7.4). En tanto que para GAG_{50} y GAG_{100} estos porcentajes suben al 90 y 100 %, respectivamente; situación que se repite cuando se estudia el *fitness* medio. En este sentido se observa que el operador EX resulta ser el más apropiado para *GA*, $GA2o_{50}$, y $GA2o_{100}$, diferenciándose significativamente del comportamiento de los otros operadores (véase sombreado en las tablas 7.2 y 7.3). Sin embargo para los algoritmos GAG_{50} y GAG_{50}, los dos operadores que muestran un comportamiento mejor son OX y CX, hecho que generalmente los distingue del resto de los operadores (véase sombreado en la tabla 7.4).

- Siguiendo con el análisis de los promedios de los valores de *fitness* (véase tablas 7.2, 7.3 y 7.4) se observa que: la utilización de estrategias de inicialización de la población, tales como la heurística 2-opt y la técnica voraz, son beneficiosas para el proceso de búsqueda. Esto es claramente visible en las instancias de mayor tamaño, como es el caso de *j02459_7* donde las mencionadas estrategias permiten un importante aumento de la calidad de la solución (18 % cuando la heurística 2-opt es aplicada y un 20 % cuando la opción voraz es usada). Si se considera el porcentaje de aplicación de la estrategia, $GA2o_{50}$ alcanza mejores resultados en instancias de moderada complejidad. También se observa que la aplicación de las estrategias 2-opt o voraz al total de las soluciones iniciales resulta lo más conveniente. Pero la aplicación de la heurística 2-opt es la que obtiene, generalmente, los resultados de mayor calidad.

- Al analizar el número final de contigs es posible ver que existen cuatro instancias (*m15421_6*, *m15421_7*, *38524243_4* y *38524243_7*) donde ninguna variante del GA obtiene la distribución óptima (véanse tablas 7.5, 7.6 y 7.7). Aún cuando el uso de las estrategias heurísticas hayan incrementado el puntaje de solapamiento entre un 2 y un 60 % (véanse tablas 7.3 y 7.4).

En relación al tiempo de ejecución, según lo observado en las tablas 7.5, 7.6 y 7.7 el uso de las dos heurísticas (2-opt y voraz) como estrategias de inicialización de la población no provoca un incremento en el tiempo de ejecución. Este resultado es esperado dado que este paso se ejecuta sólo una vez y su tiempo es insignificante con respecto al tiempo de ejecución de las 1000 generaciones del GA. Esto es un resultado interesante, ya que es un método sencillo para aumentar el rendimiento numérico del algoritmo. A pesar de su costo insignificante, los beneficios de 2-opt en la calidad son muy altos, lo cual es un hallazgo notable. El tiempo de ejecución de los algoritmos que utilizan 2-opt y la técnica voraz son similares. Además, la aplicación de ambas heurísticas es mejor que el inicio totalmente aleatorio de la población (GA). El motivo de esto radica en que, para el 70 % de las instancias, los GAs evolucionan durante unas pocas generaciones para encontrar su mejor solución (ver la Figura 7.5) cuando crea la población por medio de las heurísticas 2-opt y voraz. Por otra parte, cuando estas opciones no se aplican en el comienzo, el proceso de la evolución necesita un mayor número de generaciones para encontrar sus mejores resultados.

Tabla 7.4: Resultados experimentales de GAG_{50} y GAG_{100} con los distintos operadores genéticos. Los mejores valores están remarcados en negro.

Algoritmos	*Instancias*	*Mejor fitness*				*Fitness medio*				*Test*
		PMX	OX	CX	EX	PMX	OX	CX	EX	*KW*
GAG_{50}	*x60189_4*	11478	11478	**11571**	11478	11478.00	11478.00	**11571.00**	11478.00	+
	x60189_5	13531	13553	**13713**	13551	13293.97	13275.50	**13369.30**	13271.23	+
	x60189_6	17811	**17866**	17561	17563	17379.20	**17414.90**	17338.13	17359.87	+
	x60189_7	20884	20884	**21026**	20884	20684.33	20737.23	**20788.67**	20722.80	+
	m15421_5	37619	**37932**	37717	37773	37360.07	**37506.83**	37370.90	37399.77	+
	m15421_6	46936	47152	47180	**47961**	46699.27	46788.87	46784.87	**47140.00**	+
	m15421_7	52726	52702	52785	**52833**	52218.37	52277.93	**52334.37**	52272.37	+
	j02459_7	110593	110869	**110977**	110565	109975.10	**110223.87**	110102.87	109945.73	+
	38524243_4	217081	218250	217529	**218446**	216265.46	**217186.55**	216329.10	216296.42	+
	38524243_7	415868	**417702**	416915	416353	415205.86	416011.13	**416187.75**	415371.67	+
GAG_{100}	*x60189_4*	11478	11478	**11571**	11478	11478.00	11478.00	**11571.00**	11478.00	+
	x60189_5	13534	13501	**13774**	13535	13283.60	13326.70	**13349.43**	13304.53	+
	x60189_6	**18055**	17738	17479	17561	17457.13	**17473.75**	17350.53	17373.07	+
	x60189_7	20829	20884	**20999**	20884	20649.30	20668.77	**20804.37**	20707.60	+
	m15421_5	37597	**37914**	37814	37741	37345.57	**37528.27**	37396.13	37417.57	+
	m15421_6	46979	47151	47053	**47964**	46719.97	46838.80	46759.00	**47262.37**	+
	m15421_7	52731	**52793**	52680	52552	52254.53	52285.27	**52296.90**	52252.03	+
	j02459_7	110440	**110786**	110418	110692	110064.57	**110303.47**	110101.20	110034.13	+
	38524243_4	217455	**219155**	218266	217443	216431.75	**217371.57**	216577.91	216229.43	+
	38524243_7	416307	417107	416761	**417431**	415557.17	**416734.25**	415642.10	415832.89	+

Tabla 7.5: Resultados experimentales de GA con los distintos operadores genéticos (cont.). Los mejores valores están remarcados en negro.

Instancias	*% Contigs óptimos*				*Test*	*Tpo. medio de la mejor solución*				*Test*
	PMX	OX	CX	EX	*KW*	PMX	OX	CX	EX	*KW*
x60189_4	**100 %**	**100 %**	**100 %**	**100 %**	-	0.93	**0.83**	1.12	1.48	-
x60189_5	**100 %**	**100 %**	73 %	**100 %**	-	1.42	**1.40**	1.86	2.45	-
x60189_6	**100 %**	**100 %**	93 %	**100 %**	-	**2.24**	2.53	3.25	8.87	+
x60189_7	**100 %**	**100 %**	93 %	**100 %**	-	**2.47**	2.89	3.33	7.04	+
m15421_5	55 %	64 %	64 %	**80 %**	-	8.45	7.60	**2.32**	64.57	+
m15421_6	0 %	0 %	0 %	0 %	-	13.65	12.65	**11.53**	83.22	+
m15421_7	0 %	0 %	0 %	0 %	-	13.62	12.84	**5.60**	122.26	+
j02459_7	**67 %**	13 %	47 %	33 %	-	31.32	27.43	**21.92**	337.12	+
38524243_4	0 %	0 %	0 %	0 %	-	41.11	34.67	**25.46**	525.47	+
38524243_7	0 %	0 %	0 %	0 %	-	85.94	69.59	**43.86**	1595.69	+

Otro resultado (esperado) es que la aplicación de la recombinación EX causa un gran aumento en el tiempo de ejecución, diferenciándose significativamente del rendimiento del resto de los operadores (véase sombreado en las tablas 7.5, 7.6 y 7.7). Esto se debe al hecho de que la complejidad de este operador es de $O(n^2)$, mientras que el resto de operadores es de $O(n)$.

Tabla 7.6: Resultados experimentales de $GA2o_{50}$ y $GA2o_{100}$ con los distintos operadores genéticos (cont.). Los mejores valores están remarcados en negro.

Algoritmos	*Instancias*	*% Contigs óptimos*				*Test*	*Tpo. medio de la mejor solución*				*Test*
		PMX	OX	CX	EX	*KW*	PMX	OX	CX	EX	*KW*
$GA2o_{50}$	*x60189_4*	**100 %**	**100 %**	**100 %**	**100 %**	-	0.17	**0.15**	0.18	0.20	-
	x60189_5	**100 %**	**100 %**	**100 %**	**100 %**	-	0.35	**0.33**	0.47	1.12	-
	x60189_6	**100 %**	**100 %**	**100 %**	**100 %**	-	0.63	**0.54**	0.69	1.94	-
	x60189_7	**100 %**	**100 %**	**100 %**	**100 %**	-	**0.71**	0.72	0.78	2.60	+
	m15421_5	**100 %**	**100 %**	82 %	73 %	-	2.85	2.51	**2.45**	21.93	+
	m15421_6	0 %	0 %	0 %	0 %	-	26.15	24.02	**23.29**	91.10	+
	m15421_7	0 %	0 %	0 %	0 %	-	7.49	6.25	**5.87**	64.27	+
	j02459_7	**40 %**	20 %	27 %	20 %	-	24.25	20.63	**21.92**	270.61	+
	38524243_4	0 %	0 %	0 %	0 %	-	95.00	88.55	**78.63**	566.77	+
	38524243_7	0 %	0 %	0 %	0 %	-	190.18	174.52	**145.37**	1692.23	+
$GA2o_{100}$	*x60189_4*	**100 %**	**100 %**	**100 %**	**100 %**	-	**0.17**	**0.17**	0.20	0.18	-
	x60189_5	**100 %**	**100 %**	**100 %**	**100 %**	-	0.37	**0.36**	0.37	0.77	-
	x60189_6	**100 %**	**100 %**	**100 %**	**100 %**	-	0.64	**0.58**	0.59	1.56	+
	x60189_7	**100 %**	**100 %**	**100 %**	**100 %**	-	**0.68**	0.69	0.72	2.77	+
	m15421_5	**100 %**	**100 %**	**100 %**	**100 %**	-	7.66	2.84	3.00	**0.76**	+
	m15421_6	0 %	0 %	0 %	0 %	-	39.61	37.71	**36.90**	100.18	+
	m15421_7	0 %	0 %	0 %	0 %	-	7.66	**6.11**	6.15	64.27	+
	j02459_7	13 %	33 %	20 %	27 %	-	23.94	20.41	**13.38**	450.12	+
	38524243_4	0 %	0 %	0 %	0 %	-	150.11	142.90	**133.22**	613.36	+
	38524243_7	0 %	0 %	0 %	0 %	-	296.30	278.78	**249.39**	1754.92	+

Tabla 7.7: Resultados experimentales de GAG_{50} y GAG_{100} con los distintos operadores genéticos (cont.). Los mejores valores están remarcados en negro.

Algoritmos	*Instancias*	*% Contigs óptimos*				*Test*	*Tpo. medio de la mejor solución*				*Test*
		PMX	OX	CX	EX	*KW*	PMX	OX	CX	EX	*KW*
GAG_{50}	*x60189_4*	**100 %**	**100 %**	**100 %**	**100 %**	-	**0.21**	0.22	0.22	0.49	-
	x60189_5	**100 %**	**100 %**	**100 %**	**100 %**	-	0.42	**0.41**	0.44	0.95	-
	x60189_6	**100 %**	**100 %**	**100 %**	**100 %**	-	**0.66**	0.69	0.82	2.57	+
	x60189_7	**100 %**	**100 %**	**100 %**	**100 %**	-	**0.62**	0.62	0.73	2.97	+
	m15421_5	**97 %**	90 %	70 %	80 %	-	3.18	**2.57**	2.77	27.27	+
	m15421_6	0 %	0 %	0 %	0 %	-	7.95	6.24	**5.85**	60.40	+
	m15421_7	0 %	0 %	0 %	0 %	-	6.75	6.08	**5.32**	88.77	+
	j02459_7	**20 %**	10 %	17 %	10 %	-	29.02	25.94	**18.70**	444.45	+
	38524243_4	0 %	0 %	0 %	0 %	-	51.65	46.04	**35.32**	439.51	+
	38524243_7	0 %	0 %	0 %	0 %	-	241.98	228.36	**194.68**	1437.62	+
GAG_{100}	*x60189_4*	**100 %**	**100 %**	**100 %**	**100 %**	-	**0.25**	**0.25**	**0.25**	0.52	-
	x60189_5	**100 %**	**100 %**	**100 %**	**100 %**	-	**0.44**	0.47	0.45	1.12	-
	x60189_6	**100 %**	**100 %**	**100 %**	**100 %**	-	0.82	**0.77**	0.78	3.01	+
	x60189_7	**100 %**	**100 %**	**100 %**	**100 %**	-	**0.74**	0.76	0.85	2.74	+
	m15421_5	87 %	**97 %**	77 %	83 %	-	4.04	3.20	**3.12**	29.82	+
	m15421_6	0 %	0 %	0 %	0 %	-	8.90	7.10	**6.75**	61.24	+
	m15421_7	0 %	0 %	0 %	0 %	-	8.07	6.84	**6.11**	85.40	+
	j02459_7	**23 %**	10 %	20 %	20 %	-	39.42	36.64	**29.00**	447.33	+
	38524243_4	0 %	0 %	0 %	0 %	-	73.13	67.95	**56.57**	466.49	+
	38524243_7	0 %	0 %	0 %	0 %	-	241.69	227.37	**195.07**	1392.24	+

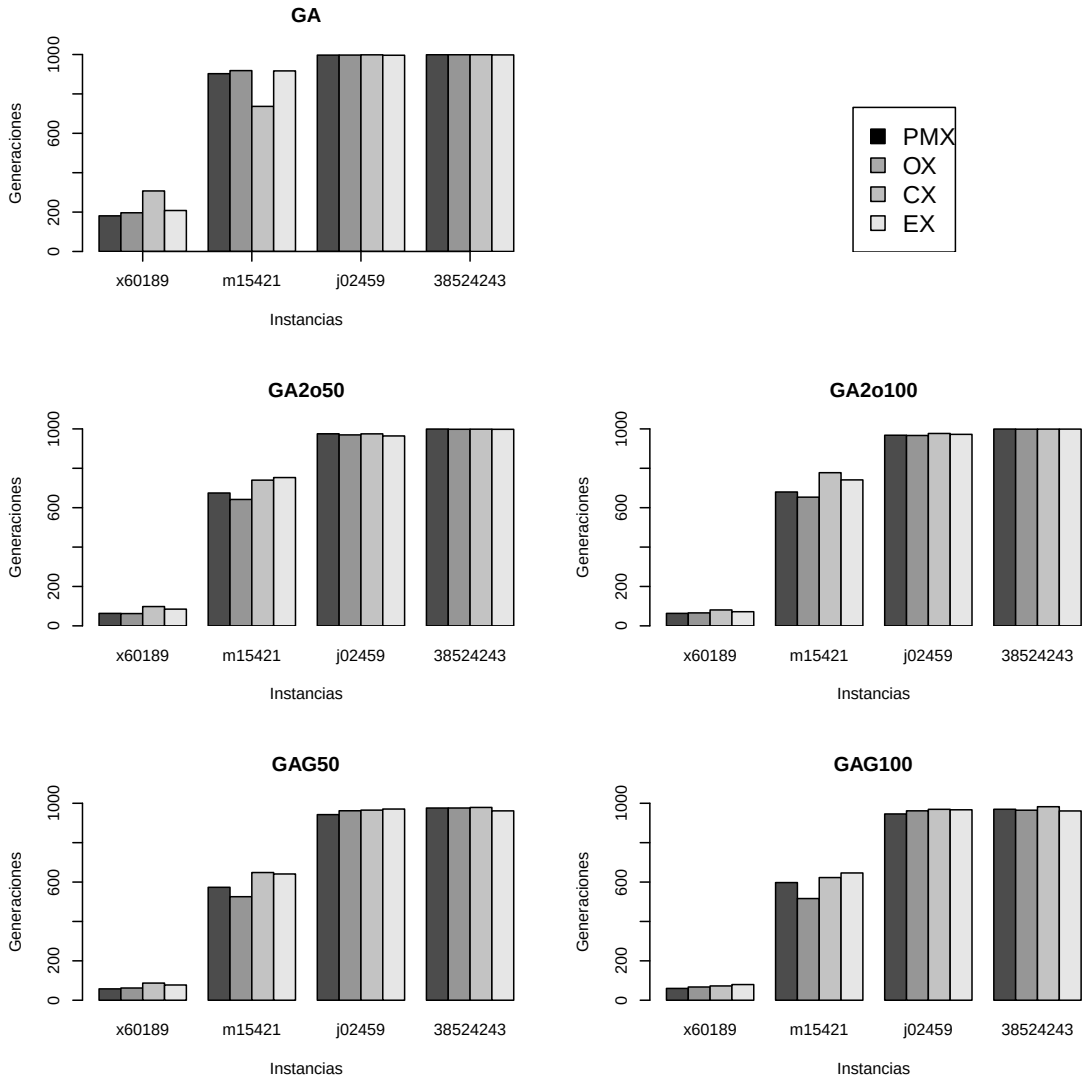


Figura 7.5: Número medio de generaciones para encontrar el mejor *fitness* obtenido por cada algoritmo propuesto en todas las instancias.

Con el fin de analizar la diversidad fenotípica, se presenta la figura 7.6 donde se muestra la diversidad de cada estrategia de inicio, teniendo en cuenta a EX y a *m15421_5* como un operador de cruce y una instancia, respectivamente, que representan al resto de los operadores e instancias. En esta figura se observa que *GA* mantiene una mayor cantidad de soluciones diferentes durante la ejecución. Mientras tanto el resto de los algoritmos pierde

casi por completo su diversidad en las primeras cien generaciones. Aunque el uso de semillas en la generación de la población inicial provoca la pérdida temprana de diversidad, también permite que en menos de 100 generaciones se alcancen valores de *fitness* significativamente superior a los obtenidos por el GA luego de 1000 generaciones. No obstante, esta mejora no es suficiente para lograr el número óptimo de contigs en todas las instancias. Cabe aclarar que la diversidad fenotípica se ha calculado como media del *fitness* de la población cada cien generaciones.

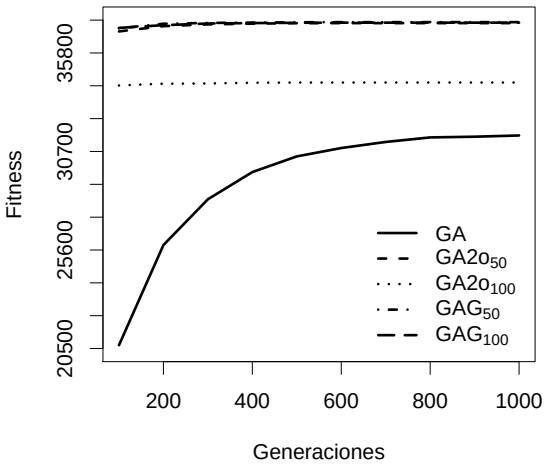


Figura 7.6: Diversidad Fenotípica obtenida al usar cada propuesta de inicio de la población considerando EX y la instancia *m15421_5*

7.1.2. Discusión

Con el objetivo de resolver de forma eficiente el problema de ensamblado de fragmentos, se han propuesto diferentes enfoques basados en algoritmos genéticos. Se han estudiado los efectos inducidos por el uso de diferentes estrategias de inicio para generar la población inicial. Luego, para mejorar el desempeño del GA, se han analizado varios operadores de recombinación para la representación de permutación. Las estrategias de inicio estudiadas son:

la estrategia aleatoria, el 2-opt y un método heurístico voraz que fue diseñado especialmente para FAP. Los operadores estudiados son: PMX, OX, CX y EX.

A partir del estudio y análisis de todas estas componentes del algoritmo genético se puede observar que la heurística 2-opt resulta muy beneficiosa para generar la población ya que permite mejorar la calidad del *fitness* y, así, obtener el número óptimo de contigs (uno) sin aumentar el tiempo de ejecución. Desde el punto de vista del operador de recombinación, el operador EX ha logrado los mejores resultados, aunque es el que emplea el mayor tiempo de ejecución en especial cuando el GA usa las estrategias de inicio aleatoria y 2-opt. Por otro lado, los operadores OX y CX son las mejores opciones cuando la población inicial se genera usando la estrategia voraz.

7.2. Resolución de FAP mediante GA+VNS

En la sección anterior se ha visto cómo los operadores de crossover más eficientes (CX y OX) y los no tanto (EX) pierden diversidad fenotípica en generaciones tempranas de la evolución sin poder encontrar el número óptimo de contigs para las instancias de mayor tamaño. Con la idea de obtener un GA que amplíe su diversidad fenotípica y de esta forma alcance mejores resultados, se ha considerado hibridarlo con VNS. De esta forma, se incorpora al GA un método explorativo que garantizaría la búsqueda en distintos vecindarios. Entonces, para que el GA no quede atrapado en óptimos locales, VNS es aplicado como un operador genético más, bajo una determinada probabilidad. El diseño de este nuevo operador se corresponde con el diseño del algoritmo FVNS presentado en el capítulo anterior; ya que el GA utilizado en este trabajo maximiza el puntaje de solapamiento entre fragmentos adyacentes de una solución dada. El esquema algorítmico correspondiente a este nuevo híbrido se muestra en el algoritmo 8.

Cabe aclarar que por una cuestión de mayor legibilidad en las tablas y las figuras, en el resto de este capítulo se referenciará a FVNS solo como VNS. En la próxima sección se analiza el comportamiento del GA con VNS como un operador genético adicional considerando la calidad de los resultados y el tiempo de CPU empleado.

Algoritmo 8 Algoritmo Genético Híbrido: GA+VNS

```
t = 0;
Inicializar(P(t));
Evaluar(P(t));
while (no se cumpla la condición de terminación do
  P'(t)=SeleccionarPadres(P(t));
  P'(t)=AplicarCrossover(P'(t), p_c); {p_c es la probabilidad de aplicar el crossover}
  P'(t)=AplicarMutación(P'(t), p_m); {p_m es la probabilidad de aplicar la mutación}
  P'(t)=AplicarVNS(P'(t), p_v); {p_v es la probabilidad de aplicar la VNS}
  Evaluar(P'(t));
  P(t+1)=SeleccionarNuevaPoblación(P(t),P'(t));
  t = t + 1;
end while
return la mejor solución;
```

7.2.1. Análisis de resultados

Los resultados analizados en esta sección corresponden a los obtenidos por el algortimo híbrido propuesto en sus 5 diferentes configuraciones parámetricas. Además, dichos resultados se comparan con los alcanzados por GA y VNS en la sección 7.1. Nuevamente para cada algoritmo, se realizan 30 ejecuciones independientes. Para llevar a cabo esta experimentación se usan computadoras con las siguientes características: procesador Pentium 4 de 2.4 GHz y 1 GB RAM; cuyo sistema operativo corresponde a la distribución SuSE de Linux con la versión de kernel 2.4.19-4GB.

Con el objetivo de analizar extensivamente esta nueva propuesta algorítmica, y así determinar su eficiencia, diferentes configuraciones paramétricas se han establecido para el algoritmo genético y para el operador VNS. En la tabla 7.8, se muestran los valores paramétricos que usa el GA: las tres versiones de GAs usan la misma configuración pero con diferentes condiciones de terminación. Para el operador VNS, se utiliza un 33 % del tamaño del cromosoma como k_{max} y t toma el valor 10. También se experimenta con diferentes configuraciones de este operador: la primera, VNS_1, sólo realiza una iteración, y la segunda, VNS_2, itera como máximo $iter_{max}$ veces. A partir de estas configuraciones de GA y VNS surgen los siguientes algoritmos híbridos: $GA_1 + VNS_1$, $GA_2 + VNS_1$, $GA_3 + VNS_1$, $GA_1 + VNS_2$ y $GA_3 + VNS_2$. Los valores paramétricos aquí usados, han sido optimizados manualmente comparándolos con diferentes valores. Es necesario resaltar que el algoritmo

híbrido $GA_2 + VNS_2$ es descartado de esta experimentación con el fin de no ser reiterativos. Dado que los resultados preliminares reflejan la misma semejanza con $GA_3 + VNS_2$, que la dada entre $GA_2 + VNS_1$ y $GA_3 + VNS_1$.

Tabla 7.8: Configuraciones paramétricas del GA

Parámetros	***Configuraciones***		
	GA_1	GA_2	GA_3
Condición de terminación	1000 generaciones	2000	5000
μ	512		
λ	512		
Op. de recombinación	OX		
y su probabilidad	0.7		
Op. de mutación.	Swap		
y su probabilidad	0.2		
Tercer op.	VNS		
y su probabilidad	0.01		
Selección de padres	Torneo binario		
Reemplazo	Las mejores μ soluciones de $(\mu + \lambda)$		

En primer lugar el comportamiento del GA es comparado con el desempeño de VNS. En la Tabla 7.9, es posible detectar que GA generalmente logra soluciones numéricamente mejores que VNS, aunque, el tiempo de CPU (expresado en segundos) empleado por VNS es significativamente menor en 5 de las 7 instancias. Esta es otra de las razones por lo que resulta interesante aplicar VNS como un operador de GA, siendo su principal objetivo escapar de óptimos locales incrementando, así, la calidad de los resultados obtenidos por el GA.

En un segundo lugar, se analiza estadísticamente el comportamiento de las cinco configuraciones desde tres puntos de vista: calidad de la solución (mejor *fitness*, *fitness* medio, porcentaje de contigs óptimos hallados), tiempo medio para encontrar la mejor solución y el tiempo total requerido por el algoritmo (los dos últimos expresados en segundos). Los resultados alcanzados son:

- Considerando la mejor solución hallada se deduce que para todas las instancias, el comportamiento de cada versión del algoritmo híbrido no difiere significativamente

Tabla 7.9: Medias de los mejores valores de *fitness* y del tiempo total de CPU empleado por VNS y GA. Los mejores valores están remarcados en negro.

Instancias	*Mejor fitness medio*		*%Contigs óptimos*		*Tiempo total medio*	
	VNS	*GA*	*VNS*	*GA*	*VNS*	*GA*
x60189_4	9290.47	**9829.93**	**100 %**	**100 %**	**0.05**	15.63
x60189_5	11994.07	**12238.34**	**100 %**	**100 %**	**0.17**	29.01
x60189_6	**15211.20**	15178.07	93 %	**100 %**	**0.85**	79.50
x60189_7	17966.80	**18076.34**	97 %	**100 %**	**1.01**	92.33
m15421_5	32904.00	**35981.27**	67 %	64 %	46.71	**43.90**
m15421_6	**39035.37**	36673.30	0 %	0 %	230.05	**141.75**
m15421_7	**45596.20**	42957.07	0 %	0 %	**269.07**	4068.09

uno del otro, dado que sus respectivos valores probabilísticos (p) son mayores a 0.05. Estas similitudes son fácilmente detectadas en la tablas 7.10, 7.11 y 7.12.

Tabla 7.10: Resultados experimentales de GA+VNS bajo las distintas configuraciones. Los mejores valores están remarcados en negro.

Instancias	*Mejor fitness*				
	GA_1+VNS_1	GA_2+VNS_1	GA_3+VNS_1	GA_1+VNS_2	GA_3+VNS_2
x60189_4	**11478**	**11478**	**11478**	**11478**	**11478**
x60189_5	14016	**14021**	14021	14016	14016
x60189_6	18122	**18301**	**18301**	**18301**	**18301**
x60189_7	**21271**	**21271**	**21271**	**21271**	**21271**
m15421_5	38126	**38252**	38216	38238	38216
m15421_6	47624	47760	47542	47649	**47824**
m15421_7	54209	54358	**54405**	54181	**54405**

Tabla 7.11: Resultados experimentales de GA+VNS bajo las distintas configuraciones (cont). Los mejores valores están remarcados en negro.

Instancias	*Fitness medio*					*Test*
	GA_1+VNS_1	GA_2+VNS_1	GA_3+VNS_1	GA_1+VNS_2	GA_3+VNS_2	*KW*
x60189_4	**11,478.00**	**11,478.00**	**11,478.00**	**11,478.00**	**11,478.00**	-
x60189_5	13,783.88	13,748.40	**13,804.25**	13,771.83	13,792.05	-
x60189_6	17,977.54	17,965.42	17,953.80	17,969.48	**17,985.93**	-
x60189_7	20,980.54	20,988.28	20,974.10	20,985.03	**20,991.13**	-
m15421_5	37,775.82	37,906.30	37,898.70	37,846.35	**37,921.80**	-
m15421_6	47,239.04	47,289.86	**47,374.40**	47,194.37	47,334.20	-
m15421_7	53,499.08	53,558.03	**53,750.38**	53,408.18	**53,750.38**	+

Tabla 7.12: Resultados experimentales de GA+VNS bajo las distintas configuraciones (cont.). Los mejores valores están remarcados en negro.

Instancias	*% Contigs óptimos*					*Test*
	GA_1+VNS_1	GA_2+VNS_1	GA_3+VNS_1	GA_1+VNS_2	GA_3+VNS_2	*KW*
x60189_4	**100 %**	**100 %**	**100 %**	**100 %**	**100 %**	-
x60189_5	**100 %**	**100 %**	**100 %**	**100 %**	**100 %**	-
x60189_6	**100 %**	**100 %**	**100 %**	**100 %**	**100 %**	-
x60189_7	**100 %**	**100 %**	**100 %**	**100 %**	**100 %**	-
m15421_5	**92 %**	90 %	90 %	90 %	90 %	-
m15421_6	0 %	0 %	0 %	0 %	0 %	-
m15421_7	0 %	0 %	0 %	0 %	0 %	-

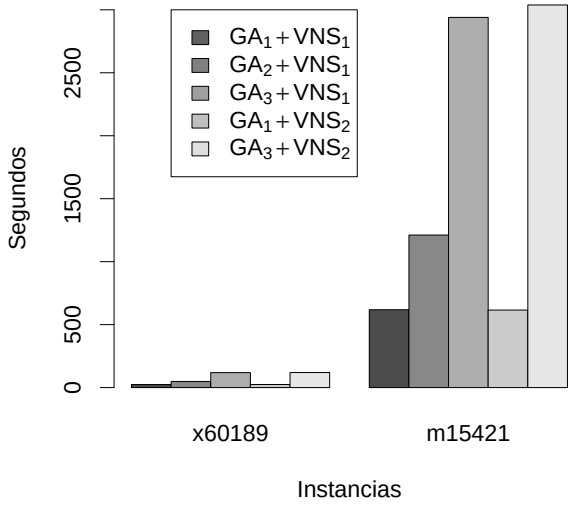
- Tomando en cuenta el tiempo de ejecución empleado para hallar la mejor solución, se puede inferir que para todas las instancias con número de acceso *x60189* los diferentes algoritmos híbridos se comportan de manera similar. En tanto que, para todas las instancias identificadas con *m15421* se distinguen dos grupos bien diferenciados: el primero itera durante 1000 generaciones ($GA_1 + VNS_1$ y $GA_1 + VNS_2$) y el segundo realiza 5000 generaciones ($GA_2 + VNS_1$, $GA_3 + VNS_1$ y $GA_3 + VNS_2$). Esto puede observarse en la tabla 7.13, donde el primer conjunto de instancias necesita menos de 31 segundos para hallar la mejor solución; mientras que el segundo requiere más de 200 segundos.

Tabla 7.13: Resultados experimentales de GA+VNS bajo las distintas configuraciones (cont.). Los mejores valores están remarcados en negro.

Instancias	*Tpo. medio de la mejor solución*					*Test*
	GA_1+VNS_1	GA_2+VNS_1	GA_3+VNS_1	GA_1+VNS_2	GA_3+VNS_2	*KW*
x60189_4	1.25	1.24	1.18	1.29	**1.08**	-
x60189_5	4.34	6.96	6.84	**4.29**	7.26	-
x60189_6	**10.87**	15.18	17.95	12.01	16.71	-
x60189_7	18.13	24.64	30.09	**17.71**	25.67	-
m15421_5	**207.95**	414.83	740.57	216.27	740.57	+
m15421_6	500.7	928.4	1219.83	**431.18**	1704.67	+
m15421_7	679.84	1231.84	2523.01	**676.34**	2523.01	+

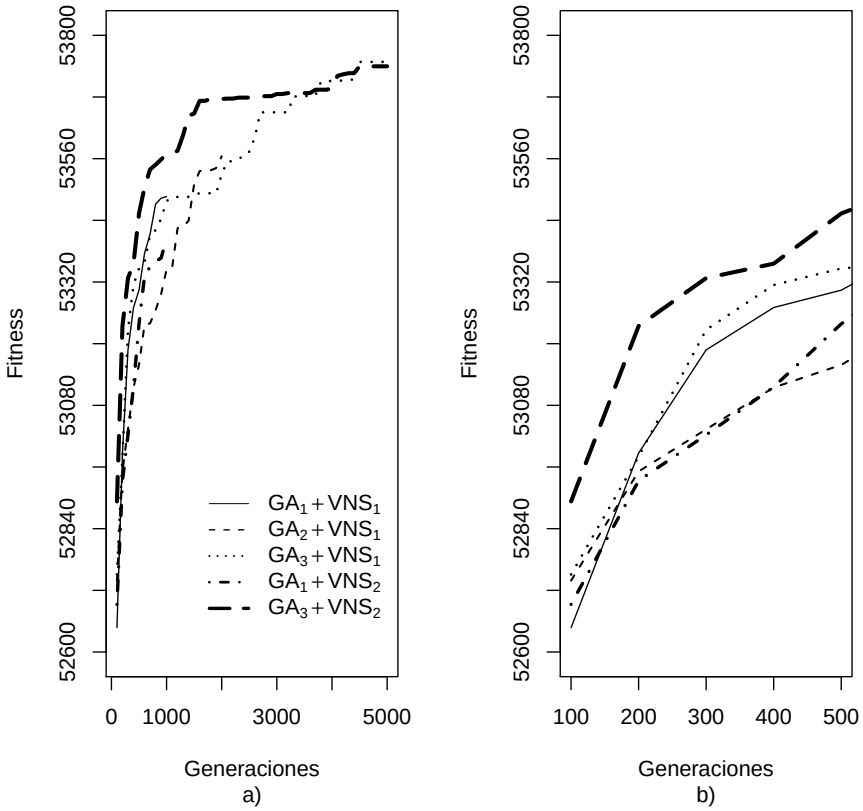
- Al analizar el tiempo total consumido por cada método híbrido (ver figura 7.7), nuevamente, se distinguen dos grupos. Uno de ellos está conformado por $GA_1 + VNS_1$ y $GA_1 + VNS_2$, mientras que el segundo conjunto está constituido por el resto de los enfoques. La razón de esto radica en que el primer grupo de algoritmos itera un máximo de 1000 generaciones (empleando menos de 40 segundos para las instancias *x60189* y de 865 para el grupo *m15421*); mientras que $GA_2 + VNS_1$, $GA_3 + VNS_1$ y $GA_3 + VNS_2$ realiza entre 2000 y 5000 generaciones (requiriendo menos de 195 segundos para las instancias *x60189* y entre 627 y 4100 segundos para las restantes). Esto significa que, para las instancias *x60189* y *m15421*, el incremento en el esfuerzo computacional está relacionado con el número de generaciones, en tanto que el tiempo empleado por el operador VNS no es importante con respecto al tiempo total de ejecución del algoritmo híbrido.

Figura 7.7: Tiempo total medio empleado por cada variante híbrida



Como se observa a lo largo de este apartado, las instancias *j02459_7*, *38524243_4* y *38524243_7* no se incluyeron en esta experimentación; ya que el tiempo de ejecución del GA híbrido sobre estas instancias supera cualquier espera razonable (más de cinco días por

Figura 7.8: Diversidad fenotípica obtenida al usar cada configuración del GA híbrido para la instancia *m15421_7*. a) Diversidad fenotípica en la totalidad de las generaciones. b) Ampliación de a) considerando sólo las primeras 500 generaciones.



una generación). El motivo de esto radica en la aplicación de VNS a instancias con más de 300 fragmentos; dado que esto implica que VNS deba explorar entre 110 y 255 vecindarios según el caso iterando entre 12 y 26 veces en cada uno de ellos.

A continuación se estudia la diversidad fenótipica, también conocida como la evolución poblacional, durante la ejecución de cada algoritmo híbrido. Para ello se usa una instancia representativa de las demás, *m15421_7* y se mide el *fitness* promedio cada 100 generaciones. Al observar la figura 7.8, es posible detectar que todas las variantes algorítmicas tienen un comportamiento similar: antes de las mil generaciones el *fitness* crece rápidamente, y luego

la mejora de las soluciones es muy lenta. De hecho, estas mejoras no son estadísticamente diferentes ni significativas como pudo observarse en la Tabla 7.10.

En general se puede concluir que: el operador VNS es necesario para evitar una convergencia prematura pero, si se consideran sus diferentes números de iteraciones, no es posible hallar diferencias entre los resultados obtenidos al usar VNS_1 y VNS_2. En tanto que, el número máximo de iteraciones del GA sí es un parámetro que incide en la calidad de las soluciones y en el tiempo de ejecución. Esto se debe a que un mayor número de iteraciones permite alcanzar mejores resultados pero, como es de esperarse, el tiempo de ejecución requerido para ello se incrementa también.

7.2.2. Discusión

El algoritmo genético propuesto en la sección 7.2 se ha hibridado en dos puntos diferentes: el primero está relacionado con una estrategia de inicio para generar la primera población y el segundo se relaciona con un método descendente utilizado como un operador genético adicional. Se ha utilizado, entonces, un método voraz como una estrategia de inicio, que ha sido especialmente diseñado para este problema (ver Sección 7.1). Además, se ha usado a la metaheurística VNS como operador genético para explorar distintos vecindarios predefinidos de la solución actual con una búsqueda local.

Se han analizado diferentes configuraciones paramétricas del GA y del operador VNS. A partir de los experimentos realizados es posible deducir varias conclusiones. En un primer lugar, los enfoques híbridos mejoran la calidad de los resultados cuando se comparan con las versiones no híbridas. En segundo lugar, las versiones híbridas que evolucionan más generaciones obtiene mejores resultados, aunque la mejora más importante se realiza durante las primeras 1000 generaciones. Además, un número mayor de generaciones significa más tiempo de ejecución. Aunque la aplicación del operador VNS contribuye a evitar una convergencia prematura, permitiendo obtener el número óptimo de contigs para 6 de los 7 casos de FAP, no es una alternativa válida cuando el número de fragmentos de una instancia es superior a 300. Dado que la mayoría de las instancias propuestas en los capítulos venideros son bastante mayores no se considerará en los estudios futuros de la tesis la aplicación de VNS.

Tabla 7.14: Número final de contigs obtenidos por los algoritmos propuestos y por otros sistemas especializados. El símbolo − indica que esa información no se proporciona.

	x60189_(4-7)	*m15421_5*	*m15421_6*	*m15421_7*	*j02459_7*	*38524243_4*	*38524243_7*
GA	1	1	2	2	1	6	2
$GA2o_{50}$	1	1	2	2	1	6	2
$GA2o_{100}$	1	1	2	2	1	6	2
GAG_{50}	1	1	2	2	1	6	4
GAG_{100}	1	1	2	2	1	6	3
$GA_1 + VNS_1$	1	1	2	2	-	-	-
$GA_2 + VNS_1$	1	1	2	2	-	-	-
$GA_3 + VNS_1$	1	1	2	2	-	-	-
$GA_1 + VNS_2$	1	1	2	2	-	-	-
$GA_3 + VNS_2$	1	1	2	2	-	-	-
CAP3	1	3	2	1	1	8	-
PHRAP	1	1	2	2	1	6	-

7.3. Comparación con otros ensambladores

Una vez estudiados los algoritmos propuestos y visto cómo los diferentes métodos alternativos influyen en su rendimiento, es el turno de comparar estos resultados con los obtenidos por otros ensambladores que se encuentran en la literatura: CAP3 [93] y PHRAP [80]. La comparación se realiza en términos del número final de contigs ensamblados y se resumen en la Tabla 7.14

En la tabla 7.14 se pueden observar, nuevamente que la generación inteligente de la población (a través de las heurísticas) beneficia el proceso de búsqueda obteniendo mejores resultados (un número menor de contigs) que el inicio aleatorio. Además, estos resultados son competitivos con respecto a los presentados en la literatura. De hecho, los métodos que usan una población generada por medio de una heurística alcanzan la solución óptima (un solo contig) para 6 de los 10 casos de FAP estudiados. En tanto que, los GAs hibridados con VNS obtienen una distribución óptima de los fragmentos en 5 de las 7 instancias abordadas. En los casos donde los diferentes GAs propuestos en este capítulo no encuentran la solución óptima, sus resultados son similares a los paquetes comerciales muy conocidos, como CAP3 o PHRAP.

7.4. Conclusiones

En este capítulo se presentan diferentes algoritmos basados en GAs para resolver FAP. En primer lugar se implementa y analiza la incorporación de tres estrategias de inicio (aleatoria, 2-opt y voraz) y la utilización de cuatro operadores de cruce (PMX, OX, CX y EX). En segundo lugar se desarrolla y estudia la inserción de un operador adicional (VNS) en un GA creando el híbrido GA+VNS.

La aplicación de las técnicas heurísticas como estrategias de inicio en el GA permite una mejor orientación de la búsqueda genética, además no afectan la eficiencia del GA, ya que su costo es insignificante con respecto a la ejecución del resto del algoritmo. Por otra parte, el operador de cruce EX inserta en la solución recientemente creada más del 90 % de la información proveniente de sus dos padres, circunstancia que le permite a un GA obtener soluciones de mayor calidad que con PMX, OX o CX. Sin embargo, el tiempo de ejecución puede incrementarse más de 90 veces en las instancias más grandes, situación no deseable en optimización combinatoria. En cambio, la aplicación del algoritmo voraz como estrategia de inicio le permite a OX y a CX obtener soluciones de igual o mayor calidad que las obtenidas por EX sin incrementar el tiempo de ejecución del GA.

La aplicación de GA+VNS produce un aumento del *fitness* medio, con respecto al GA sin hibridar, que va desde un 5 % a un 22 % con una media del 14 %. En cuanto al tiempo total medio empleado $GA_1 + VNS_1$ y $GA_1 + VNS_2$ son las variantes híbridas recomendadas para solucionar los dos grupos de instancias resueltos.

Hasta el momento se ha concluido por separado sobre los algoritmos propuestos en las dos primeras secciones de este capítulo, ahora es necesario realizar una contrastación entre todos ellos que permita *categorizarlos*. Para lograr esto, en la tabla 7.15 se presenta un resumen de los principales datos que miden el comportamiento y el desempeño de cada uno de ellos. El puntaje asociado a cada algoritmo se obtiene usando un promedio ponderado donde: un 45 % del peso total corresponde a la calidad de la solución (columnas 3 a 5), otro 45 % se asocia al esfuerzo computacional (columnas 6 a 9) y el 10 % restante al número de instancias resueltas (columna 10). En este capítulo se decide incorporar esta última medida comparativa dado que los GA híbridos presentados en la segunda sección no resuelven tres de las instancias usualmente presentadas en la literatura; siendo ésta una desventaja notable con respecto al

resto de los GAs propuestos en este capítulo. En cuanto a la calidad se considera por cada algoritmo el número de instancias donde encuentra: el valor de *fitness* máximo (columna 3), el mayor *fitness* medio (columna 4), un solo contig en todas las ejecuciones (columna 5). Con respecto al esfuerzo computacional se tiene en cuenta por cada algoritmo el número de instancias donde: se requiere el menor tiempo para encontrar la mejor solución (columna 6) y el porcentaje de reducción o incremento del tiempo total de ejecución con respecto al GA estándar.

Tabla 7.15: *Categorización* de las versiones de GAs propuestas en este capítulo. El signo '+' que acompaña a los valores de la séptima columna indican una reducción del tiempo total medio de ejecución; en tanto que el signo '-' representa un aumento del mismo.

Algoritmo	*Operador de cruce*	*Fitness máx.*	*Mejor fitness medio*	*100 % Contigs óptimos*	*Tpo. mín. de la mejor sol.*	*% Tiempo reducido*	*Instancias resueltas*	*Puntaje*	*Rango*
GAG_{50}	OX	0	0	4	3	+99.99 %	10	2.50	1°
GAG_{50}	PMX	0	0	4	3	+99.99 %	10	2.50	1°
GAG_{100}	OX	0	4	4	0	+99.99 %	10	2.42	2°
GAG_{50}	CX	1	1	4	3	-99.13 %	10	2.35	3°
GAG_{100}	PMX	3	0	4	0	+99.99 %	10	2.27	4°
$GA2o_{50}$	OX	0	0	5	1	+93.81 %	10	2.19	5°
$GA2o_{50}$	EX	0	4	5	0	-90.75 %	10	2.15	6°
$GA2o_{100}$	OX	0	0	6	0	+99.99 %	10	2.12	7°
$GA2o_{100}$	PMX	0	0	6	0	+86.57 %	10	2.09	8°
$GA2o_{50}$	PMX	0	0	5	0	+99.99 %	10	1.97	9°
$GA2o_{100}$	EX	2	1	6	0	-203.07 %	10	1.89	10°
$GA_3 + VNS_1$		0	2	4	0	+99.73 %	7	1.82	11°
$GA2o_{100}$	CX	2	0	6	0	-203.07 %	10	1.74	12°
$GA2o_{50}$	CX	1	0	5	0	-90.75 %	10	1.70	13°
$GA_1 + VNS_1$		1	0	4	0	+99.94 %	7	1.67	14°
$GA_3 + VNS_2$		0	1	4	0	+99.72 %	7	1.67	14°
GAG_{100}	CX	1	1	4	0	-122.74 %	10	1.62	15°
GAG_{100}	EX	2	0	4	0	-122.74 %	10	1.62	15°
GA	PMX	0	0	4	0	0.00 %	10	1.60	16°
GA	OX	0	0	4	0	0.00 %	10	1.60	16°
GA	CX	0	0	1	2	0.00 %	10	1.60	16°
GA	EX	0	0	4	0	0.00 %	10	1.60	16°
GAG_{50}	EX	1	0	4	0	-99.13 %	10	1.53	17°
$GA_1 + VNS_2$		0	0	4	0	+99.94 %	7	1.52	18°
$GA_2 + VNS_1$		0	0	4	0	+99.89 %	7	1.52	19°

En general, en la *categorización* mostrada en la tabla 7.15 se observa que los primeros 10 puestos están ocupados por las versiones de GA que utilizan distintas estrategias heurísticas para crear la población inicial. En particular, el primer puesto es obtenido por GAG_{50} cuando utiliza OX y PMX. Si bien con el uso de PMX la calidad de de las soluciones e inferior a la alcanzada cuando se aplican OX y CX, la utilización de esta estrategia de inicio hace que la aplicación del operador PMX requiera menos tiempo de ejecución que cuando no la usa. Situación que no ocurre con el uso del operador CX. A partir del onceavo lugar aparecen los GAs hibridados con VNS mostrando que, aunque están al mismo nivel de los ensambladores de la literatura, se encuentran en inferioridad de condiciones con respecto a sus antecesores por su incapacidad de resolver las tres instancias más grandes analizadas en este capítulo. Resumiendo el ensamblador GAG_{50} que usa OX logra el mejor compromiso entre calidad y costo temporal. Aunque se rechaza hipótesis **H1** (estado del arte, véase pág. 3), es decir, los ensambladores propuestos en este capítulo no superan a los de la literatura, la calidad de los resultados encontrados es la misma.

Tabla 7.16: *Categorización* de los 4 mejores ensambladores metaheurísticos propuestos en la parte II.

Algoritmo	*Fitness máx.*	*Mejor fitness medio*	100 % *Contigs óptimos*	% *Tiempo reducido*	*Puntaje*	*Rango*
SA	8	7	10	0.00 %	4.17	**1°**
PALS	5	2	4	93.00 %	2.30	**2°**
GAG_{50}-OX	1	1	4	99.99 %	1.50	**3°**
GAG_{50}-PMX	1	1	4	99.99 %	1.50	**3°**

Con el objetivo de cerrar la segunda parte de esta tesis se contrastan las mejores opciones algorítmicas basadas en población con las correspondientes basadas en trayectoria y, de esta forma determinar el conjunto de ensambladores metaheurísticos que mejor se adapta a la resolución de instancias de baja y mediana complejidad del problema de ensamblado de fragmentos. Para ello se confrontan, en la tabla 7.16, nuevamente los indicadores de calidad y los de costo computacional correspondientes a ISA, PALS y las dos versiones de GAG_{50} que han obtenido el primer puesto en este capítulo. Como puede observarse en esta última tabla las metaheurísticas basadas en trayectoria presentan un mejor desempeño en la búsqueda

de soluciones para FAP que las basadas en población. En particular ISA sobresale sobre los restantes ensambladores al encontrar en un mayor número de instancias el mejor valor de *fitness* y su promedio más alto, además de, conseguir la distribución óptima de fragmentos en todas las ejecuciones para todos los casos.

Parte III

Resolución de instancias complejas del problema de ensamblado de fragmentos

Capítulo 8

Resolución de instancias de mayor tamaño

Para realizar un estudio significativo es necesario analizar más instancias del problema de ensamblado de fragmentos. Centrarse únicamente en un grupo reducido (veáse Tabla 5.1 en el capítulo 5), que sólo contiene un par de instancias de gran tamaño y de alta complejidad, podría derivar en conclusiones específicas de ese grupo y no del problema en general.

Con la colaboración del Dr. Gabriel Luque, se ha desarrollado un generador de instancias que crea una serie de fragmentos de ADN a partir de secuencias genómicas con la complejidad requerida. La complejidad relacionada al problema de ensamblado de fragmentos está dada por:

- El número de fragmentos a ordenar, que determina el tamaño de la permutación. Es decir, la principal dificultad de FAP se debe a la alta dimensionalidad del espacio de búsqueda asociado: dados k fragmentos existen $2^k k!$ combinaciones. Esto convierte a la segunda fase de FAP (distribución) en un problema con una explosión exponencial de soluciones. Por otra parte, un aumento en el número de fragmentos, también, incrementa la complejidad espacial y temporal de los algoritmos asociados a la primera y a la tercera fase de la resolución de FAP (superposición y consenso), ya que ellos necesitan almacenar y procesar permutaciones más extensas.

- El número de bases en cada fragmento. El incremento en el número de bases nucleótidas implica un aumento del esfuerzo computacional empleado en la primera y en la tercera fase de FAP. Esto se debe a que los algoritmos asociados a ambas fases requieren contrastar un mayor número de bases entre dos fragmentos distintos.
- La longitud de la secuencia original de ADN. Éste es un factor altamente influyente en la determinación del número de fragmentos y de bases en cada una de ellos. Dado que la capacidad de lectura de la máquina secuenciadora define el número de bases del fragmento e indirectamente el número de fragmentos.

Este generador de instancias es una aplicación fácilmente configurable que genera instancias con la dificultad deseada. El uso del generador de instancias elimina la posibilidad de ajustar los algoritmos para una instancia particular, permitiendo una comparación más justa entre diferentes opciones algorítmicas. Con un generador de instancias es posible evaluar a los algoritmos sobre un gran número de casos de prueba del problema y, consecuentemente, serán más confiables y precisas las afirmaciones sobre el desempeño de un algoritmo con respecto al problema en general.

Los objetivos de este capítulo son: analizar el comportamiento de los ensambladores metaheurísticos que hasta el momento han mostrado un mejor desempeño (ISA, PALS y), desarrollar una nueva metaheurística híbrida para resolver instancias de mayor tamaño de FAP, denominada SAX y confirmar la segunda hipótesis planteada en esta tesis (complejidad y eficiencia, véase pág. 3). Los ensambladores ISA, PALS y GAG_{50} han logrado una mejor calidad de los resultados, ya sea medida por medio del *fitness* o por el número final de contigs. Además, han sido capaces de resolver el conjunto completo de instancias analizado con un esfuerzo computacional mínimo y han confiramdo la primera hipótesis formulada en este trabajo (estado del arte, véase pág. 3). El objetivo perseguido con SAX es combinar la potencialidad de ISA como ensamblador de fragmentos con operadores de recombinación genéticos como intensificadores de la búsqueda. El resto de este capítulo, entonces, se divide en tres secciones: en la primera se describe con detalle el generador de instancias desarrollado, en la segunda se describe SAX y se estudia la aplicación de ISA, PALS, GAG_{50} y SAX a las nuevas instancias y finalmente, en la última sección, se presentan las conclusiones sobre lo actuado.

8.1. Generación de un nuevo conjunto de instancias

El procedimiento utilizado para generar artificialmente instancias de FAP, denominado DNAgen, se basa en el primer paso del método de secuenciamiento propuesto por Sanger (*Shotgun Sequencing*) [157]. Esto es, dada una secuencia de ADN original, DNAgen realiza múltiples copias de la misma generando una gran secuencia que es dividida aleatoriamente en cientos de fragmentos.

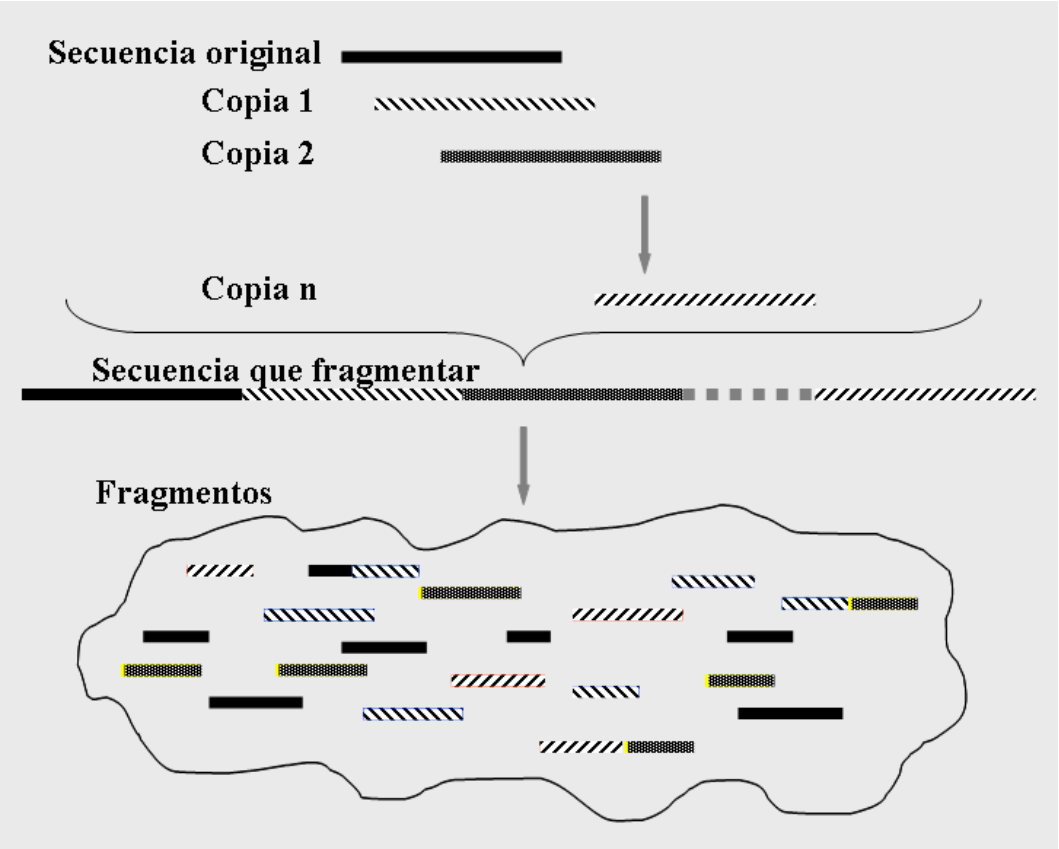


Figura 8.1: Esquema de funcionamiento de DNAgen.

Con el propósito de simular las características que influyen en la complejidad de una instancia (números de fragmentos, número de bases en cada fragmentos y la longitud de la secuencia original), DNAgen permite manipular los siguientes parámetros: el factor de multiplicación, el número mínimo y máximo de bases que un fragmento puede tener y la

longitud de la secuencia original. El primero determina el número de copias a realizar de dicha secuencia, el segundo regula la extensión de los fragmentos y el tercero influye en el número de fragmentos a generar para esa instancia. Cabe aclarar que la cantidad de bases de un fragmento es un número elegido al azar en un rango previamente establecido. Por ejemplo, para generar la instancia *acin7* mostrada en la tabla 8.1, se utiliza la secuencia original de ADN ACIN02000007 cuya longitud es de 426840 bases; la cual es replicada 18 veces generando así una secuencia de 7683120 bases. Esta nueva secuencia ha sido dividida aleatoriamente en fragmentos cuyas longitudes pertenecen al rango [500, 1500]; obteniendo de esta forma una instancia con 901 fragmentos. La figura 8.1 muestra el esquema de funcionamiento de DNAgen.

Entonces, cuanto más extensa sea la secuencia original y mayor el factor de multiplicación, la secuencia resultante revestirá mayor complejidad, ya que constará de un mayor número de fragmentos. Sin embargo, si el rango de bases permitidas por fragmento es muy amplia, puede causar un número bajo de fragmentos y, por ende, la optimización combinatoria no resultaría complicada. Por el contrario, si es muy chico generaría demasiados fragmentos pudiendo generar instancias muy complejas, incluso, a partir de secuencias de corta longitud. Además, es necesario considerar que este rango no debe exceder los límites de lecturas de las máquinas secuenciadoras que se intentan simular.

En función a lo planteado en el párrafo anterior, se han realizado diversas pruebas sobre diferentes factores de multiplicación y rangos de amplitud de fragmentos. El objetivo de dichas pruebas ha sido hallar una combinación de los mismos que permita obtener instancias artificiales de FAP de alta complejidad, comparables con las generadas por una máquina secuenciadora. De estas pruebas surge que un factor de multiplicación adecuado es 18 y la longitud de un fragmento debe estar entre las 500 y 1500 bases para las secuencias de ADN elegidas.

Las secuencias originales de ADN utilizadas para generar las nuevas instancias han sido seleccionadas y descargadas desde el sitio web NCBI[1]. Corresponden a la bacteria microbiana humana ATCC 49176 cuyos números de acceso van desde ACIN02000001 hasta ACIN02000026. Particularmente, se han utilizado las secuencias más extensas de este ge-

[1] http://www.ncbi.nlm.nih.gov/

noma para fragmentarlas usando el generador DNAgen. Estas nuevas instancias se pueden encontrar en el sitio web LISI[2] y sus características se muestran en la tabla 8.1.

Tabla 8.1: Información sobre el nuevo conjunto de instancias.

Instancias	***Cobertura***	***Longitud media de los fragmentos***	***Número de fragmentos***	***Longitud de la secuencia original***
acin1	26	182	307	2170
acin2	3	1002	451	147200
acin3	3	1001	601	200741
acin5	2	1003	751	329958
acin7	2	1003	901	426840
acin9	7	1003	1049	156305

8.2. Resolución de instancias de mayor tamaño mediante ISA, PALS, GAG_{50} y SAX

En esta sección se utilizan los algoritmos ISA, PALS y GAG_{50}, según fueron introducidos en los capítulos 6 y 7 para resolver este nuevo conjunto de prueba de FAP. En cuanto a las diferentes versiones de GAG_{50}, se escogió la que utiliza el operador de cruce OX. Además, se propone a continuación una nueva metaheurística denominada SAX.

SAX es un algoritmo híbrido cooperativo de bajo nivel, que combina a ISA con un operador genético de cruce. Este algoritmo unifica el buen desempeño de ISA con la información heurística obtenida al aplicarse un operador de cruce. La idea detrás de los incorporación de esta información heurística es procurar un algoritmo que guíe la búsqueda hacia regiones donde se combina altos puntajes de solapamiento con el número óptimo de contigs. Como muestra el algoritmo 9, SAX usa la mutación por inversión para crear dos nuevas y diferentes soluciones (S_1 y S_2) a partir de la actual (S_0). S_0 es creada usando la estrategia de inicio voraz propuesta en el capítulo 5. S_1 y S_2 son recombinadas utilizando el operador de crossover OX. Esta nueva solución reemplaza a la actual si es mejor que ella o si es aceptada bajo la distribución de Boltzman. A diferencia de ISA, SAX incrementa la exploración al

[2]http://mdk.ing.unlpam.edu.ar/ lisi/

Algoritmo 9 SAX

```
k = 0;
inicializar T y S_0; {solución inicial y temperatura }
evaluar S_0 en E_0;
repeat
  repeat
    k = k + 1;
    generar S_1 desde S_0 por inversión;
    {Pc es la probabilidad de crossover}
    if (random(0,1)< Pc) then
      generar S_2 desde S_0 por inversión;
      generar S_3 aplicando OX sobre S_1 y S_2;
      S_1 = S_3;
    end if
    evaluar(S_1) en E_1;
    {si la nueva solución, S_1, es mejor que la actual, entonces S_1 es aceptada}
    if (E_1 - E_0) ≥ 0 then
      S_0 = S_1;
      E_0 = E_1;
      {si la nueva solución es peor que la actual, S_1 es aceptada bajo la probabilidad de Boltzman}
    else
      if exp((E_1 - E_0)/T) > random[0,1) then
        S_0 = S_1;
        E_0 = E_1;
      end if
    end if
  until (k mod Long. Cadena de Markov) == 0
  actualizar T;
until condición de corte sea satisfecha
return S_0;
```

generar dos soluciones vecinas a partir de la actual, pero también incrementa la explotación al combinar ambas soluciones. Con el propósito de ilustrar la descripción de SAX, en la figura 8.2 se muestra el esquema de funcionamiento de esta nueva metaheurística híbrida.

Un punto crítico en la comparación de estos cuatro algoritmos es la determinación del criterio de terminación de los mismos. Esto es porque una iteración de un algoritmo basado en trayectoria significa operar con una solución y en uno basado en población se utilizan más de una. Por lo tanto, el número de iteraciones no es un criterio apropiado para medir y consecuentemente comparar el esfuerzo computacional realizado por estos algoritmos. Tam-

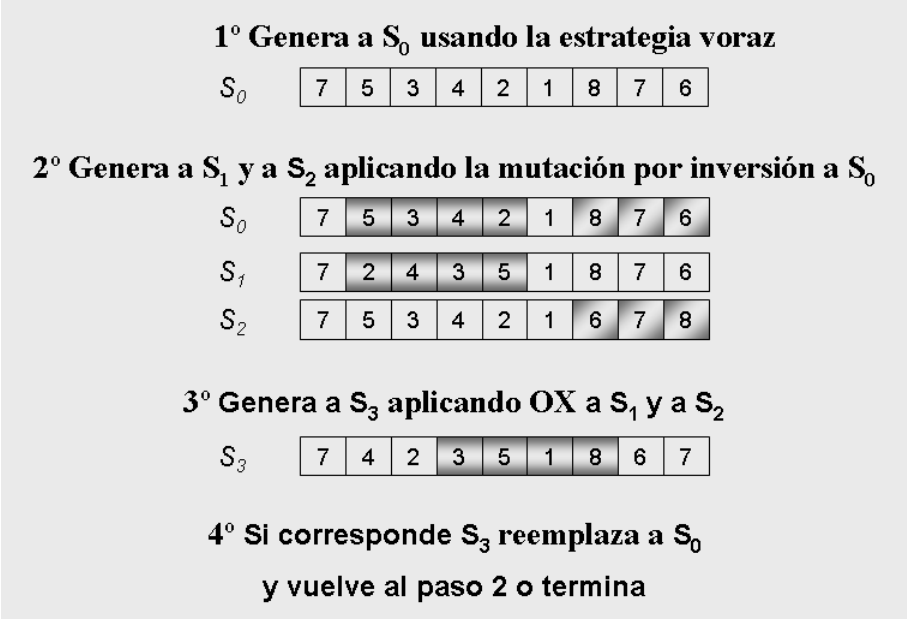


Figura 8.2: Esquema de funcionamiento de SAX.

poco lo es el número de evaluaciones de la función de *fitness* dado que PALS, a diferencia de ISA, GAG_{50} y SAX, no la calcula durante la búsqueda sino que estima este valor. Por ende, el criterio elegido es el tiempo de ejecución. Es decir, todos los algoritmos realizan su búsqueda durante una misma cantidad de tiempo (60 segundos). De esta forma es posible medir lo que cada ensamblador puede realizar en un determinado período y compararse con otros bajo las mismas condiciones. En la tabla 8.2 se resume la configuración paramétrica de cada uno de estos ensambladores.

8.2.1. Análisis de Resultados

Los resultados analizados en esta sección corresponden a los obtenidos por ISA, PALS, GAG_{50} y SAX al resolver las instancias del problema de ensamblado de fragmentos presentadas en este capítulo. Con el fin de ofrecer resultados con soporte estadístico, por cada algoritmo se realizan 30 ejecuciones independientes usando los parámetros mostrados en la tabla 8.2. Para esto se utilizan computadoras con procesadores AMD Phenom (64 bits) a 2.4

Tabla 8.2: Valores paramétricos usados por ISA, PALS, GAG_{50} y SAX.

	Parámetro	***Valor***
ISA y SAX	Longitud de la cadena de Markov	10
	Temperatura inicial	0.99
PALS	Selección del movimiento	Los mejores movimientos
GAG_{50}	μ	256
	λ	256
	Op. de mutación.	*Swap*
	y su probabilidad	0.2
	Selección de padres	Torneo binario
	Reemplazo	Las mejores μ soluciones de $(\mu + \lambda)$
GAG_{50} *y SAX*	Op. de recombinación	Order Crossover (OX)
	y su probabilidad	0.7 para GAG_{50} y 1.0 para SAX
ISA, PALS, GAG_{50} *y SAX*	Condición de terminación	60"

GHz y 2 GB de RAM, con un sistema operativo perteneciente a la distribución Slackware de Linux con una versión del kernel 2.6.27.7-smp.

Los resultados se analizan considerando los siguientes dos aspectos: calidad de las soluciones y el esfuerzo computacional. Por ende, en las tablas 8.3 y 8.4 se presentan datos sobre: el mejor valor de *fitness* hallado y el promedio del mismo, el porcentaje de veces que se encontró el número óptimo de contigs, el tiempo promedio en que se tardó en encontrar la mejor solución y los resultados del test estadístico Kruskall-Wallis y de las comparaciones múltiples.

Tabla 8.3: Resultados experimentales de ISA, PALS, GAG_{50} y SAX. Los mejores valores están remarcados en negro.

Instancias	*Mejor fitness*				*Fitness medio*				*Test*
	ISA	PALS	GAG_{50}	SAX	ISA	PALS	GAG_{50}	SAX	*KW*
acin1	44511	**46876**	45565	46865	44258.94	**46758.20**	45377.03	46636.97	+
acin2	138699	**144634**	143444	144567	136719.33	**144112.00**	142499.33	143972.87	+
acin3	152177	**156776**	154947	155789	150193.90	**156507.50**	153980.90	154991.57	+
acin5	143864	**146591**	145332	145880	142770.50	**146563.80**	145193.87	145309.40	+
acin7	155117	**158004**	155873	157032	154977.53	**157972.80**	155801.40	156939.40	+
acin9	311035	**325930**	313203	314354	308603.93	**324620.30**	312005.55	311863.23	+

Tabla 8.4: Resultados experimentales de ISA, PALS, GAG_{50} y SAX (cont.). Los mejores valores están remarcados en negro.

Instancias	*% Contigs óptimos*				*Test*	*Tpo. medio de la mejor solución*				*Test*
	ISA	PALS	GAG_{50}	SAX	*KW*	ISA	PALS	GAG_{50}	SAX	*KW*
acin1	**100.00 %**	0.00 %	0.00 %	**100.00 %**	+	6.29	**0.39**	29.56	55.38	+
acin2	**100.00 %**	0.00 %	0.00 %	**100.00 %**	+	15.44	**2.42**	58.30	59.42	+
acin3	**100.00 %**	0.00 %	0.00 %	**100.00 %**	+	17.73	**6.50**	59.42	59.49	+
acin5	**100.00 %**	0.00 %	0.00 %	**100.00 %**	+	26.11	**19.30**	59.82	59.73	+
acin7	**100.00 %**	0.00 %	0.00 %	**100.00 %**	+	35.57	**33.16**	59.98	59.63	+
acin9	**100.00 %**	0.00 %	0.00 %	**100.00 %**	+	43.78	39.84	**2.27**	59.77	+

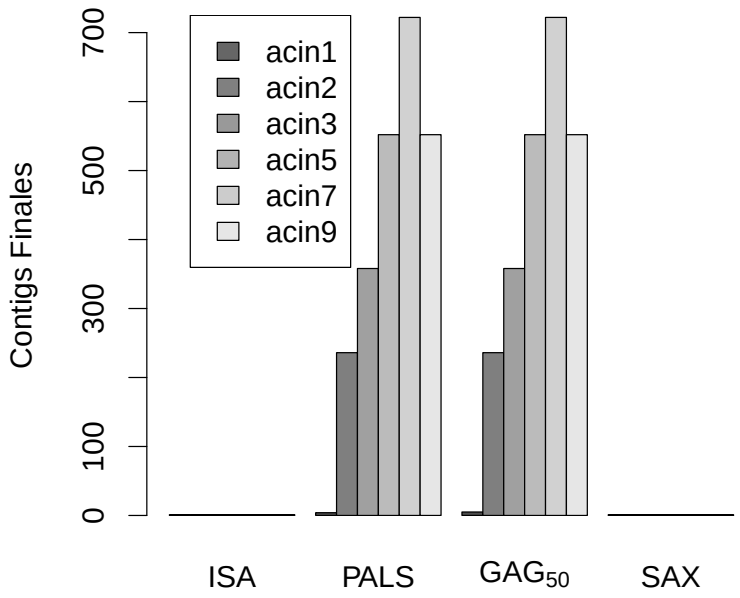


Figura 8.3: Números de contigs obtenidos por ISA, PALS, GAG_{50} y SAX en cada instancia.

En primer lugar se estudia el comportamiento de estos ensambladores considerando la calidad de las soluciones. En este sentido, se analizan el *fitness* y el porcentaje de contigs óptimos encontrados por instancia (véanse tablas 8.3 y 8.4) y se observa que:

- ISA logra la distribución óptima de los fragmentos (1 único contig) en todos los casos, pero obtiene los valores de *fitness* más bajos con respecto a PALS, GAG_{50} y SAX.

- PALS y GAG_{50} no logran esta distribución para ninguna instancia y obtienen soluciones con igual número de contigs y muy distantes al óptimo, como puede observarse en la figura 8.3. En esta figura se muestra el número final de contigs obtenido por cada uno de los tres ensambladores en cada instancia.

- Si bien PALS no logra soluciones con un único contig, obtiene los valores de *fitness* más altos. Esto indica que cada uno de los contigs encontrados tiene un alto grado de consenso, pero aún falta un reordenamiento más eficiente de estos cadenas para unirlas en una única secuencia.

- SAX junto a PALS y GAG_{50} forman parte del grupo de algoritmos estadísticamente similares con valores de fitness más altos pero, SAX al igual que ISA, logra soluciones con 1 único contig en todos los casos.

En función a lo expresado anteriormente, se deduce que el número óptimo de contigs no se asocia necesariamente con un alto puntaje de solapamiento. ISA para lograr un reordenamiento de los fragmentos que permita encontrar la secuencia total de ADN necesita sacrificar el puntaje de solapamiento. Esto sucede cuando ISA acepta soluciones peores a la actual dependiendo de una probabilidad de aceptación. De esta forma, ISA escapa de óptimos locales, explora nuevas regiones del espacio de búsqueda y encuentra la distribución óptima de fragmentos. Por otra parte SAX, que también acepta soluciones peores a la actual, logra ambos objetivos mediante la utilización del operador de cruce OX, que le permite realizar una mejor explotación de las nuevas regiones alcanzadas. En otras palabras, SAX alcanza la distribución óptima de fragmentos con un alto puntaje de solapamiento. A partir de todo esto y siendo PALS una metaheurística basada en trayectoria, como ISA y SAX, se propone a futuro incorporar a PALS un mecanismo para aceptar temporalmente soluciones peores y de esta forma mejorar la distribución de fragmentos.

En un segundo lugar, se analiza el esfuerzo computacional promedio de cada ensamblador para encontrar su mejor solución (ver tabla 8.4):

- ISA muestra una relación directa entre el número de fragmentos de la instancia y el tiempo necesario para encontrar la mejor solución. Es decir, a mayor cantidad de fragmentos, más tiempo de ejecución se requiere para encontrar la mejor solución.

- Se detecta que PALS es el ensamblador que emplea menos tiempo, pero dado los resultados obtenidos se infiere que este ensamblador converge rápidamente a un óptimo local.

- En el caso de GAG_{50}, se observa que en la mayoría de los casos este tiempo es casi igual al tiempo máximo de ejecución; deduciéndose así que este ensamblador necesitaría más tiempo para encontrar una mejor solución, pero esto no asegura que logre encontrar el óptimo.

- SAX requiere tiempos de ejecución similares y cercanos al tiempo de ejecución total para encontrar su mejor solución en todas las instancias, pero a diferencia de GAG_{50} este tiempo es suficiente para encontrar el óptimo.

En resumen ISA y PALS necesitan más tiempo para encontrar sus mejores soluciones cuando el tamaño de la instancia crece. En cambio, los ensambladores que utilizan el operador OX (GAG_{50} y SAX) generalmente requieren el tiempo máximo de ejecución para hallar sus mejores resultados.

8.3. Comparación con otros ensambladores

En esta sección se compara el desempeño de ISA, PALS, GAG_{50} y SAX en contrapartida con CAP3 [93]. La comparación con PHRAP[80] no es posible, ya que el proceso para generar las instancias es artificial y por ende no tiene en cuenta la generación de los cromatogramas asociados que son necesarios para ejecutar los otros ensambladores.

A fin de llevar a cabo dicha comparación, en la tabla 8.5 se presenta el número mínimo de contigs alcanzados por ISA, PALS, GAG_{50} y CAP3 al resolver el nuevo conjunto de instancias introducidas en este capítulo. En la mencionada tabla, se observa como ISA y SAX encuentran el número óptimo de contigs en todos los casos; mientras que el resto de los ensambladores (incluyendo CAP3) no lo logra en ninguno de los casos. Además, éstos

Tabla 8.5: Mejor número de contigs para los algoritmos ISA, PALS, GAG_{50}, SAX y CAP3.

Instancias	*ISA*	*PALS*	GAG_{50}	*SAX*	*CAP3*
acin1	**1**	4	5	**1**	9
acin2	**1**	236	236	**1**	239
acin3	**1**	358	358	**1**	361
acin5	**1**	552	552	**1**	556
acin7	**1**	722	722	**1**	727
acin9	**1**	552	552	**1**	552

muestran un crecimiento significativo en el esfuerzo de resolución cuando resuelven instancias de más de 400 fragmentos. Por otra parte, se observa que CAP3 obtiene soluciones finales con un número mayor de contigs finales que PALS y GAG_{50}.

8.4. Conclusiones

En este capítulo se introduce un generador de instancias correspondientes al problema de ensamblado de fragmentos, con el cual se crea un conjunto de instancias de alta complejidad. De esta forma, se reduce al mínimo la posibilidad de ajustar los algoritmos a una instancia o a un conjunto reducido de instancias de mediana complejidad. Esto permite una comparación equitativa de los diferentes ensambladores. Para llevar a cabo dicha comparación, se han elegido las tres metaheurísticas (ISA, PALS y GAG_{50}) que en la parte II brindaron un mejor desempeño sobre todo el conjunto de instancias y confirmaron la primer hipótesis planteada en esta tesis (estado del arte, véase pág. 3), además de SAX una nueva metaheurística híbrida propuesta en este capítulo.

Cuando estos ensambladores se comparan considerando la calidad de los resultados, ISA y SAX obtienen una secuencia final de un único contig. ISA lo logra sacrificando el puntaje de solapamiento total (o *fitness*), mientras que SAX lo consigue sin disminuir el *fitness* al aplicar OX. PALS y GAG_{50} obtienen valores de *fitness* altos pero esto no implica que alcancen una secuencia de un único contig. En efecto, no lo logran para ninguna de las instancias quedando atrapados en soluciones de muy buen *fitness* pero sin una distribución óptima de los fragmentos. Por ende, se propone para el futuro una versión de PALS que permita escapar de los óptimos locales aplicando un mecanismo que acepte temporalmente

soluciones peores. Por otra parte, en este capítulo y en los anteriores se observa que esta función de *fitness* sólo considera el puntaje de solapamiento y que la variación de esta medida no es inversamente proporcional al número de contigs.

Adicionalmente, estos cuatro ensambladores metaheurísticos confirman la hipótesis **H1** (estado del arte, véase pág. 3), al obtener soluciones de mayor calidad que CAP3. Pero también confirman a **H2** (complejidad, véase pág. 3) ya que son capaces de manipular instancias de gran complejidad sin perder calidad: ISA y SAX encuentran la distribución de fragmentos óptima en todas las ejecuciones, mientras que PALS y GAG_{50} mantienen altos valores de *fitness*. Además confirman la hipótesis **H2.1** (eficiencia, véase pág. 4), dado que estos resultados son alcanzados en menos de 60".

Capítulo 9

Resolución de instancias con ruido en los datos

En el desarrollo de esta tesis y en la literatura [5, 8, 9, 113, 119, 120, 122, 143], varios ensambladores metaheurísticos tales como SA, VNS, GA, ACO y PALS resuelven el problema de ensamblado de fragmentos de forma relativamente eficiente. En términos generales, las técnicas antes mencionadas han reportado resultados sobre todo con las instancias sin ruido del problema. Por el contrario, existen pocos estudios que han tratado el ruido en algunas etapas del FAP, véase, por ejemplo [103, 188, 128]. Dado que en un proyecto de genómica la mayoría de las tareas dependen de la precisión (y eficiencia) de las secuencias obtenidas por el ensamblador, un tratamiento adecuado de los casos ruidosos es imprescindible. Por otra parte, también es conocido que los errores en los datos (ruido) pueden producirse antes o durante el proceso de ensamblado. Razones por las cuales, en este trabajo, primero se identifican las fuentes importantes de ruido y luego, se analiza y compara el comportamiento de los ensambladores metaheurísticos cuando resuelven instancias con errores provenientes de tales fuentes.

Los principales errores surgen durante:

1. ***El método de secuenciación.*** En este caso, el ADN se rompe en fragmentos de tamaño aleatorio y el error aparece como un cambio en algunas bases correctas de sus extremos (Ruido en las bases nucleótidas,-NB[1]-).

2. ***La fase de superposición.*** El ruido en esta etapa se produce cuando por error se alteran algunos valores en la matriz de solapamiento (Matriz de solapamiento ruidosa, -NS-).

3. ***La evaluación de las secuencias reconstruidas.*** El ensamblador evalúa la calidad de las soluciones halladas. Los errores surgen al alterarse directamente el cálculo del *fitness* (*Fitness* ruidoso, -NF-).

Los ensambladores metaheurísticos elegidos, nuevamente, son ISA, PALS, GAG_{50} y SAX dado el buen desempeño demostrado en los capítulos anteriores. Para llevar a cabo dicho estudio se requiere contar con un conjunto de instancias ruidosas, con las características antes mencionadas. Por otra parte, es necesario comparar la calidad de las soluciones obtenidas para las instancias sin y con ruido. De esta forma, es posible analizar la robustez de las metaheurísticas involucradas. Por esta razón, resulta imprescindible generar instancias ruidosas a partir de las usadas hasta el momento para cada una de las fuente de ruido; permitiendo así una comparación válida de los resultados obtenidos. En los siguientes apartados se explica cómo se genera cada uno de estos conjuntos de instancias.

Específicamente, en la sección 9.1 se describe la creación de instancias a partir del ruido suscitado en la etapa de secuenciación. En tanto que en la sección 9.2, se expone la generación de instancias con ruido en la matriz de solapamiento. A continuación, en la sección 9.3, se explica cómo el ruido es simulado durante el cálculo del *fitness*. En el apartado se introduce la configuración paramétrica de todos los algoritmos usados en la experimentación. Seguidamente, se analizan los resultados de la experimentación: en primer lugar, se estudia y compara el desempeño de estos algoritmos para resolver instancias sin

[1] La notación NB, NF y NS provienen del artículo "Metaheuristic Assemblers for DNA Sequences Containing Noisy Information" y en este trabajo se ha decidido mantener dicha nomenclatura. NB son las siglas de *Noisy Bases*, NS las de *Noisy Score* y NF las de *Noisy Function*.

ruido, con ruido en las bases (NB), con ruido en la matriz (NS) y con ruido en el *fitness* (NF). En segundo lugar, se analizan los distintos experimentos realizados sobre las instancias NS con diferentes intensidades de ruido. Luego de la discusión sobre lo experimentado se procede a comparar con otros ensambladores propuestos en la literatura. Por último, se presentan las conclusiones.

9.1. Simulación de ruido durante la secuenciación

Esta fuente de ruido está presente cuando los errores aparecen en las bases de un fragmento. Esto se debe a que la salida de datos desde un equipo de secuenciación puede contener errores, lo que provoca: denominaciones erróneas, inserciones y eliminaciones de bases en la secuencia de ADN leída.

Considerando que los porcentajes de error producidos por una máquina secuenciadora de última generación, son menores al cinco por ciento y los errores se producen en los extremos de los fragmentos, se adoptó el siguiente criterio: los fragmentos que modificar son seleccionados al azar bajo una probabilidad de 0.05. Luego se realizan inserciones, eliminaciones o modificaciones de las bases ubicadas en el sector inicial o final del fragmento escogido. Se establece que el número máximo de bases para modificar cada extremo es proporcional al tamaño de los fragmentos y varía entre un 0 y un 3 %. Este procedimiento ha sido aplicado al conjunto de instancias presentados en las tablas 5.1 y 8.1 (ver capítulos 5 y 8, respectivamente), generando de esta forma un conjunto de 16 nuevas instancias denominadas instancias ruidosas o con ruido y cuya nomenclatura es: NB-nombreInstanciaOriginal. Por ejemplo, *NB-x60189_4* corresponde a la instancia *x60189_4* con ruido en las bases nucleótidas.

Como se mencionó antes, este tipo de error no se produce durante la resolución del FAP, sino con anterioridad a la misma. Por ende, es preciso iniciar y completar la fase de superposición para cada una de las nuevas instancias. De esta forma se obtienen las respectivas matrices de solapamiento usadas en las etapas siguientes de distribución y consenso.

El ruido en las bases nucleótidas afecta, entonces, todas las fases del ensamblado de fragmentos. En primer lugar, la matriz de solapamiento se calcula sobre fragmentos que contienen errores, esto genera puntajes que no representan con precisión la superposición

entre esos fragmentos. En consecuencia, el ensamblador metaheurístico utilizará valores de fitness que no evidencian la calidad real de la solución pudiendo guiar la búsqueda hacia regiones de baja calidad. Una distribución de fragmentos de escasa calidad impacta negativamente en la fase de consenso al disminuir las posibilidades de encontrar una secuencia con un único contig. Además, los errores en las bases pueden provocar toma de decisiones erróneas durante la última fase de ensamblado, generando así errores en la reconstrucción de la secuencia original.

9.2. Simulación de ruido en la fase de superposición

Este tipo de errores se produce durante el cálculo del solapamiento entre los dos fragmentos. Un error en el cálculo de superposición puede representar pseudo mutaciones, eliminaciones e inserciones en los fragmentos, por lo que los diseños obtenidos podrían conducir dificultades durante las fases de distribución y de consenso. En otras palabras, esta clase de errores afecta la tarea del ensamblador metaheurístico, dado que la evaluación de la solución se basa en un puntaje de solapamiento erróneo. De esta forma, la búsqueda puede dirigirse hacia regiones donde no es posible encontrar distribuciones de fragmentos de buena calidad, reduciendo las posibilidades de hallar una secuencia óptima durante el consenso.

Esta fuente de ruido se origina al cambiar, uniformemente, el puntaje de solapamiento de las instancias sin ruido. Esto se logra incluyendo ruido en la matriz de puntajes que representa la superposición entre pares de fragmentos; es decir que, de una manera única y sencilla son representados todos los tipos posibles de mutaciones, eliminaciones e inserciones, y los errores que se encuentran en la naturaleza a través de sus efectos resultantes en los datos numéricos de este problema. A partir de la aplicación de este proceso, a las instancias introducidas en los capítulos 5 y 8, se crea un conjunto de 16 nuevas instancias ruidosas.

Con el fin de analizar el comportamiento algorítmico de los ensambladores considerando diferentes intensidades de ruido, se generan cinco conjuntos de 16 instancias ruidosas cada uno con diferentes porcentajes de error: 5, 10, 15, 20 y 25 %. El porcentaje máximo de intensidad es definido en función a la mayor cantidad de errores permitidos en esta fase; a partir del cual se definen sistemáticamente el resto de las intensidades con el propósito de realizar

una amplia cobertura de la cantidad de errores cometidos. Estas instancias son identificadas de la siguiente forma: NSintensidad-nombreInstanciaOriginal, por ejemplo, *NS10-acin1* representa a la instancia *acin1* con una intensidad del 10 % de ruido en la matriz de solapamiento. En particular, las instancias NS10 son usadas para comparar el comportamiento de los ensambladores metaheurísticos ante las tres fuentes de ruido aquí estudiadas.

9.3. Simulación de ruido durante el cálculo del *fitness*

Muchos factores pueden provocar evaluaciones poco precisas o erróneas (ruidosas). Un ejemplo de esto son los resultados obtenidos al evaluar instancias con ruido en la matriz de superposición. Otra fuente ruido, como ya se observó, ocurre durante el suecuenciamiento, por esta razón el *fitness* resultante no representa el valor real. Pero la fuente de ruido NF está directamente relacionada a la imprecisión en el cálculo del *fitness*. Tales errores podrían guiar erróneamente a la búsqueda, dado que se selecciona una solución de acuerdo a su valor de *fitness*. Por consiguiente, las soluciones encontradas por el ensamblador podrían no ser de buena calidad y esto se vería reflejado en la secuencia final obtenida en la fase de consenso.

Miller y Goldberg establecen en [127] que: una función de *fitness* ruidosa puede calcularse como la suma del *fitness* real más un componente ruidoso aleatorio. Para modelar esta fuente de ruido, el componente de ruido aleatorio debe ser generado a partir de una distribución normal no sesgada. En la ecuación 9.1 se ilustra la ecuación de *fitness* ruidosa usada en este trabajo:

$$NF(l) = F(l) + BM \tag{9.1}$$

donde BM es el componente ruidoso aleatorio, generado por medio del método Box-Muller [27], que garantiza la distribución normal no sesgada. BM se calcula, entonces, como muestra la ecuación 9.2.

$$BM = \sqrt{-2\ln(U)} * \sin(2\pi V) \tag{9.2}$$

donde U y V son variables uniformemente distribuidas.

Para esta fuente de ruido no es necesario generar nuevas instancias, dado que el ruido no afecta los datos de entrada sino que se produce durante el proceso de búsqueda llevado a cabo

por el ensamblador. Sin embargo, con el fin de mantener consistencia con las otras fuentes de ruido y claridad en la exposición de los resultados, se han renombrado las instancias como NF-nombreInstanciaOriginal. Por ejemplo, *NF-j02459_7* indica la instancia *j02459_7* con función de *fitness* ruidosa.

9.4. Resolución de instancias ruidosas mediante ISA, PALS, GAG_{50} y SAX

Para resolver este nuevo conjunto de instancias ruidosas de FAP, en esta sección se utilizan los algoritmos ISA, PALS, GAG_{50} y SAX propuestos en los capítulos 6, 7 y 8. En cuanto a las diferentes versiones de GAG_{50}, se escoge nuevamente la que utiliza el operador de cruce OX.

Otra vez, el punto crítico para la comparación de estos cuatro algoritmos es la condición de terminación y por las mismas razones descriptas en el capítulo anterior se elige el tiempo de ejecución como criterio de parada. Éste y el resto de los parámetros involucrados en estos cuatro algoritmos se resumen en la tabla 9.1.

Tabla 9.1: Valores paramétricos usados por ISA, PALS, GAG_{50} y SAX.

	Parámetro	***Valor***
ISA y SAX	Longitud de la cadena de Markov	10
	Temperatura inicial	0.99
PALS	Selección del movimiento	Los mejores movimientos
GAG_{50}	μ	256
	λ	256
	Op. de mutación.	*Swap*
	y su probabilidad	0.2
	Selección de padres	Torneo binario
	Reemplazo	Las mejores μ soluciones de $(\mu + \lambda)$
GAG_{50} *y SAX*	Op. de recombinación	Order Crossover (OX)
	y su probabilidad	0.7 para GAG_{50} y 1.0 para SAX
ISA, PALS, GAG_{50} *y SAX*	Condición de terminación	60"

9.4.1. Análisis de Resultados

En este apartado se analiza el desempeño de ISA, PALS, GAG_{50} y SAX cuando resuelven instancias de FAP con errores. En primer lugar, se estudia cómo se conducen estos ensambladores ante instancias con ruido proveniente de diferentes fuentes de ruido. Luego, se estudia el comportamiento de estos algoritmos al resolver instancias con distintas intensidades de ruido en la matriz de solapamiento.

Para comprender mejor el análisis realizado es necesario considerar tres cuestiones. La primera es que dadas las características de las fuentes de ruido NB y NF, no es factible simular diferentes intensidades de ruido en estos casos. La segunda consiste en tener en cuenta que PALS es inmune al ruido en el cálculo del *fitness*, ya que esta operación no es realizada durante la búsqueda; por ende PALS no es aplicado a la fuente de ruido NF. Por último, para las soluciones finales de los casos ruidosos, sólo se consideran los valores reales de *fitness* y de contigs, por lo que resulta necesario volver a calcular estos valores utilizando la matriz de solapamiento sin ruido.

Con el fin de ofrecer resultados con soporte estadístico, por cada algoritmo se realizan 30 ejecuciones independientes usando los parámetros mostrados en la tabla 9.1. Esta experimentación se lleva a cabo sobre computadoras con procesadores AMD Phenom (64 bits) a 2.4 GHz y 2 GB de RAM, con un sistema operativo perteneciente a la distribución Slackware de Linux con una versión del kernel 2.6.27.7-smp. Dado que en cada experimentación se emplean un total de 64 y 70 instancias respectivamente, los resultados experimentales no son tabulados como en los capítulos anteriores sino que se muestra un resumen de los mismos (error porcentual medio del mejor *fitness* con respecto al máximo encontrado hasta el momento, porcentaje promedio de contigs óptimos, y tiempo medio empleado para encontrar la mejor solución) en diagramas de caja, conocido en Inglés como *Box plot* [186]. La tabulación mencionada anteriormente se puede encontrar en el apéndice A junto con los resultados del test de varianza.

En los diagramas de caja (véase figura 9.1) [146], las valores por encima (o por debajo) del límite superior (o inferior) se consideran atípicos. El lado superior de la caja representa el tercer cuartil y por debajo de él se encuentran como máximo el 75 % de los datos. La mediana coincide con el segundo cuartil dividiendo a la distribución de datos en partes iguales. El

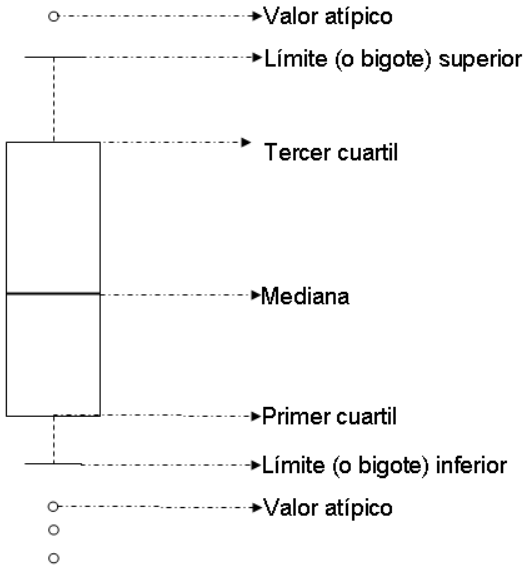


Figura 9.1: Explicación de un diagrama de caja.

primer cuartil coincide con el lado inferior de la caja y por debajo de él se encuentran como máximo el 25 % de los datos. Por último, se encuentra el extremo inferior del bigote que separa los resultados atípicos de los agrupados en el cuerpo principal de datos. Cuanto más extensos la caja y los bigotes, más dispersa es la distribución de datos. Por ejemplo, en la figura 9.2, particularmente en la gráfica correspondiente a las instancias NB, la longitud de las cajas y de los bigotes es igual a la longitud del eje y; esto indica que los errores de las soluciones varían a lo largo del rango $[0; 1]$. En cambio, para las instancias sin ruido, la variación del error es mínima ya que pertenecen al intervalo [0;0.1].

Por otro lado, esta clase de diagramas facilita la comparación entre los resultados obtenidos por un algoritmo para las instancias sin y con ruido, permitiendo de esta forma establecer si tal algoritmo es resistente o no frente al ruido. Esto es, si para un mismo algoritmo las cajas correspondientes a los resultados de las instancias sin ruido y a los provenientes de una fuente con ruido son similares, entonces sus resultados no difieren estadísticamente y el algoritmo es resistente a dicha fuente de error. Siguiendo la definición de robustez dada en el inicio de esta tesis (véase pág. 4), una metaheurística resistente al ruido es un algoritmo

robusto. Por otra parte, este tipo de gráficos también es usado para realizar comparaciones múltiples, en este caso se comparara el comportamiento de distintos algoritmos. Por ende, a partir de estos diagramas es posible comparar el comportamiento de los algoritmos con rigurosidad estadística.

9.4.1.1. Comparación del comportamiento algorítmico en las tres fuentes de ruido

En esta sección, se compara el comportamiento de ISA, PALS, GAG_{50} y SAX considerando las tres fuentes de ruido (bases nucleótidas, matriz de solapamiento y *fitness*). Para esto se usan las instancias NB, NF y NS10, también se han normalizado los valores de *fitness* encontrados en las 30 ejecuciones para cada instancia. En la figura 9.2 se resumen los resultados normalizados obtenidos por estos cuatro ensambladores para cada conjunto de instancias sin y con ruido. En tanto que, el porcentaje de contigs óptimos encontrados para cada caso se muestra en la figura 9.3; mientras que el esfuerzo computacional empleado por cada uno de estos algoritmos, se presenta en la figura 9.4.

La inclusión del comportamiento de ISA, PALS, GAG_{50} y SAX en las instancias sin ruido, se debe a la necesidad de medir el grado de robustez de estos algoritmos. En otras palabras, una metaheurística se considera robusta si las soluciones obtenidas no sufren alteraciones importantes ante la presencia de variaciones en los datos, como es la producida por el ruido.

Se comienza con el estudio de la calidad de las soluciones teniendo en cuenta el error porcentual de la mejor solución encontrada cuando se compara con el valor de *fitness* máximo encontrado. Dicho estudio muestra que:

- Las mejores soluciones, aquellas con errores próximos o iguales a cero, surgen al resolverse las instancias sin ruido, y las NF (véanse los gráfico *a* y *c* de la figura 9.2). Para las instancias sin ruido, ISA y PALS encuentran los *fitness* máximos (error igual a 0.0) en todas las instancias; mientras que SAX y GAG_{50} hallan soluciones con errores mínimos (menores a 0.05). En el segundo caso, ISA y SAX encuentran soluciones con errores cercanos a cero en todas las instancias; mientras que GAG_{50} resuelve la mitad

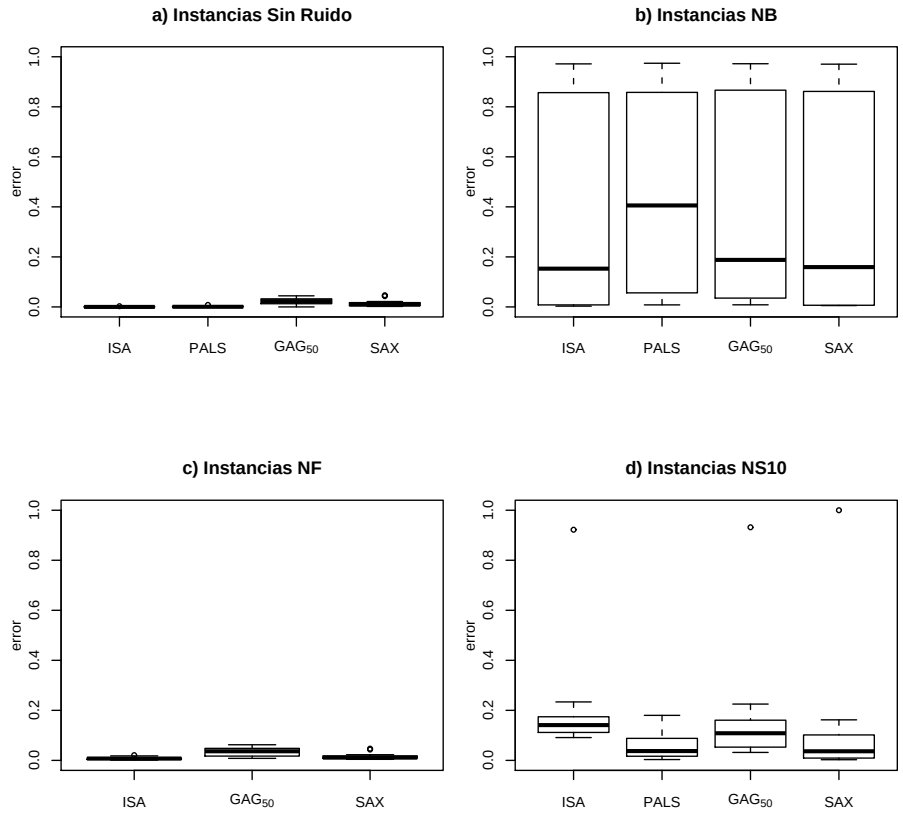


Figura 9.2: Diagramas de cajas correspondiente al error porcentual de la media de los mejores *fitness* encontrados por ISA, PALS, GAG_{50} y SAX en las instancias sin ruido (a), NB (b), NF (c) y NS10 (d).

de las instancias hallando soluciones con errores menores a 0.15 y en el resto los errores varían entre 0.15 y 0.25.

- Si la fuente de ruido está en la matriz de solapamiento (NS10) los algoritmos que obtienen mejores *fitness* son PALS y SAX (véase el gráfico d de la figura 9.2); ya que todas sus soluciones tienen errores menores a 0.20. En tanto que ISA y GAG_{50} encuentran soluciones con errores menores e iguales a 0.30.

- Sin duda alguna, la fuente de ruido que resulta más difícil de enfrentar es NB (véase el gráfico b de la figura 9.2); es decir, cuando el ruido se produce en la fase de

secuenciación los ensambladores encuentran las soluciones de menor calidad. Si bien los cuatro algoritmos encuentran soluciones con errores que varían entre 0 y 1, PALS presenta la mediana más alta. Esto significa que para la mitad de las instancias, PALS halla soluciones con errores menores o iguales a 0.40; mientras que para el resto de los ensambladores, los errores son menores a 0.20.

En cambio, cuando la calidad de los resultados se analiza considerando la cantidad de veces que la distribución óptima es alcanzada por cada ensamblador se observa que:

- ISA y SAX logran un único contig para todas las instancias sin ruido en todas las ejecuciones; mientras que PALS y GAG_{50} sólo lo logran para un cuarto de las instancias. Sin embargo, esto no se repite cuando se resuelven las instancias NF (véanse los gráficos *a* y *c* de la figura 9.3). Específicamente, cuando este tipo de instancias ruidosas son resueltas, ISA no encuentra la distribución óptima en el 50 % de las instancias (mediana igual a cero) y en el resto lo hace como mínimo una vez. En tanto que para SAX, la mediana sube al 20 %; esto significa que sólo en un cuarto de las instancias, SAX no encuentra la distribución óptima de contigs. Con respecto a GAG_{50}, también se observa una reducción en la calidad de los resultados al disminuir el tercer cuartil.

- Para la mitad de las instancias NS10, ninguno de los ensambladores encuentra el contig óptimo ya que el porcentaje es cero (véase gráfico *d* de la figura 9.3). PALS logra el 100 % de contigs óptimos en un cuarto de los casos NS10 (tercer cuartil igual a límite superior). SAX y GAG_{50}, en ese orden, presentan comportamientos ligeramente inferiores a PALS. En tanto que, ISA es el ensamblador que brinda el peor desempeño para esta fuente de ruido.

- En el caso de las instancias NB, los cuatro ensambladores no encuentran distribuciones óptimas para la mitad de ellas (véase gráfico *b* de la figura 9.3). En tanto que, para las restantes el porcentaje de contigs óptimos hallados varía entre el 3 y el 100 %. Siendo SAX, el único ensamblador que logra el número máximo de óptimos y lo hace en un cuarto de las instancias (tercer cuartil igual al límite superior).

Resumiendo el análisis sobre la calidad de las soluciones, se infiere que: ISA, PALS, GAG_{50} y SAX muestran un mejor comportamiento cuando el ruido se produce en la primera

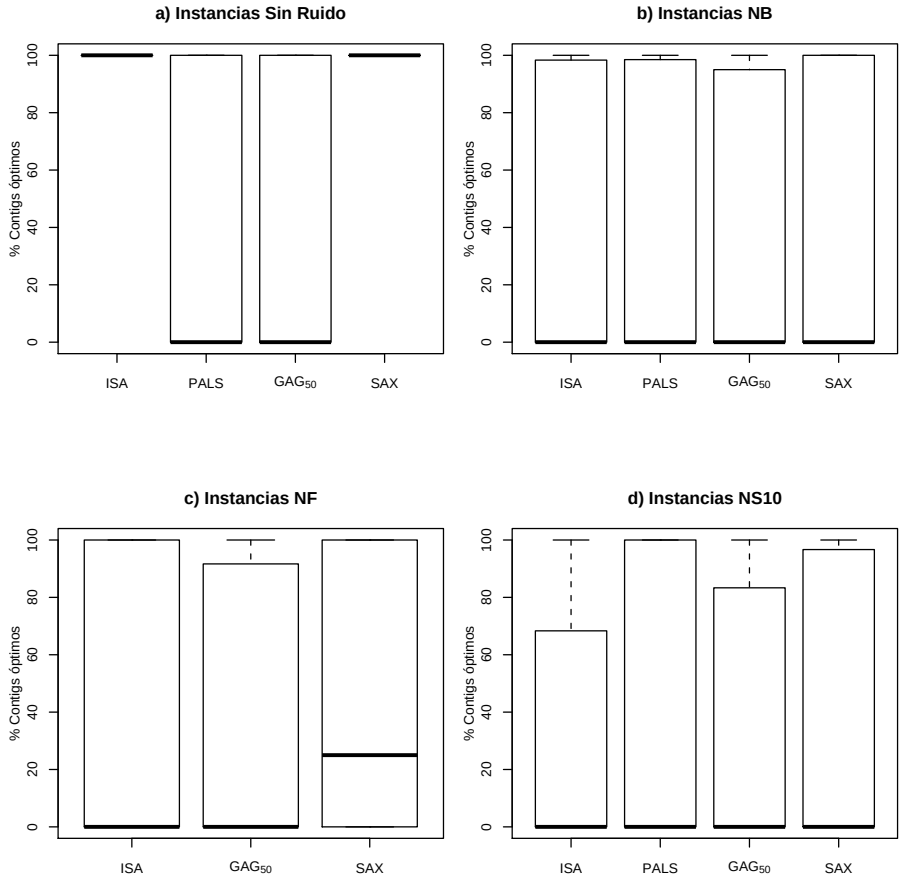


Figura 9.3: Diagramas de cajas correspondiente al porcentaje medio de veces que ISA, PALS, GAG_{50} y SAX encuentran el número de contigs óptimos en las instancias sin ruido (a), NB (b), NF (c) y NS10 (d).

y en la última fase del proceso de ensamblado (superposición y consenso). En particular, ISA y SAX son los ensambladores de mejor desempeño cuando el ruido se produce durante el cálculo del *fitness*. En tanto que, PALS y SAX, en ese orden, son los ensambladores que mejor resuelven las instancias con ruido en la matriz de solapamiento; pero dada la diferencia de los resultados obtenidos (*fitness* y contigs) por los cuatro ensambladores con respecto a los casos sin y con ruido, las metaheurísticas robustas son PALS y GAG_{50}. Por último, para las instancias NB, no existen diferencias importantes entre los algoritmos, sin embargo ISA sobresale ligeramente sobre el resto. Aunque esto no significa que sea una

metaheurística robusta cuando el ruido se presenta en las bases nucleótidas, de hecho al analizar los diagramas de caja de las figuras 9.2 y 9.3 se detecta que PALS y GAG_{50} son los ensambladores más robustos para este caso.

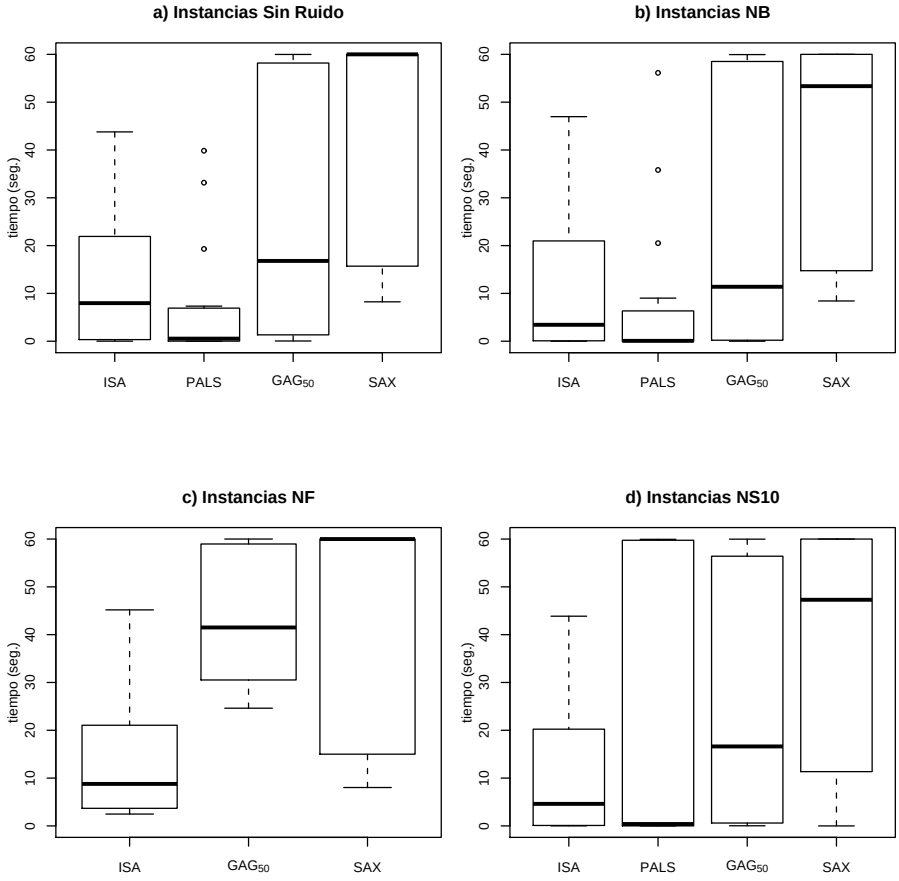


Figura 9.4: Diagramas de cajas correspondiente al tiempo promedio empleado por ISA, PALS, GAG_{50} y SAX para encontrar su mejor solución en las instancias sin ruido (a), NB (b), NF (c) y NS10 (d).

Por último, se analiza la figura 9.4, donde se observa que el esfuerzo computacional requerido por cada ensamblador es diferente. El esfuerzo de PALS es el menor (menos de 10 segundos) en dos de los cuatro escenarios, pero para las tres fuentes de ruido necesita menos de un segundo para hallar su mejor solución en la mitad de los casos de prueba. En

tanto que, ISA necesita menos de 25 segundos para encontrar la mejor solución en el 75 % de las instancias. SAX demanda 60 segundos en el 50 % de las instancias, y en otro 25 % requiere entre 15 y 60 segundos. GAG_{50}, al igual que SAX, es uno de los ensambladores que más tiempo necesita para encontrar su mejor solución en la mitad de las instancias pero, a diferencia de SAX, no siempre requiere del tiempo máximo de ejecución.

9.4.1.2. Análisis de instancias con diferentes intensidades de ruido en la matriz de solapamiento

En la sección anterior se analizó en detalle el comportamiento de los ensambladores ISA, PALS, GAG_{50} y SAX para resolver instancias de FAP con errores provenientes de distintas fuentes de ruidos. Ahora, se estudia el comportamiento de los algoritmos teniendo en cuenta diferentes porcentajes de ruido en la matriz de solapamiento. Para ello, se han generado cuatro nuevas series de instancias con intensidades de ruido diferente en la matriz de solapamiento, lo que resulta en los casos con 5, 10, 15, 20 y 25 % de ruido (por ejemplo, NS05 identifica un conjunto de casos con un 5 % de ruido en la mencionada matriz). Cada una de estas series está conformada por 16 instancias. Los algoritmos se han ejecutado 30 veces por cada nueva instancia; de esta forma, es posible aplicar las pruebas estadísticas a los resultados obtenidos. Con el objetivo de comparar este gran volumen de resultados de una manera adecuada, los valores de *fitness* se han normalizado teniendo en cuenta el valor máximo en cada caso, y los contigs finales se analizan considerando el porcentaje de contigs óptimos hallados por cada algoritmo en cada instancia (ver figuras 9.5 y 9.6). Por último, se analiza el esfuerzo computacional empleado por cada uno de estos algoritmos, a partir de los datos resumidos en la figura 9.7. Dado que en esta sección se comparan distintas intensidades de ruido para una misma fuente, los diagramas de caja no se agrupan por tipo de ruido sino por algoritmo; facilitando así el análisis y la comparación de los ensambladores.

Cuando se estudia la calidad de los resultados considerando los valores de *fitness* resumidos en la figura 9.5 se observa que:

- En general, el error con respecto al mejor *fitness* encontrado aumenta para ISA y GAG_{50} cuando la intensidad del ruido crece. Esto significa que, la dificultad de reso-

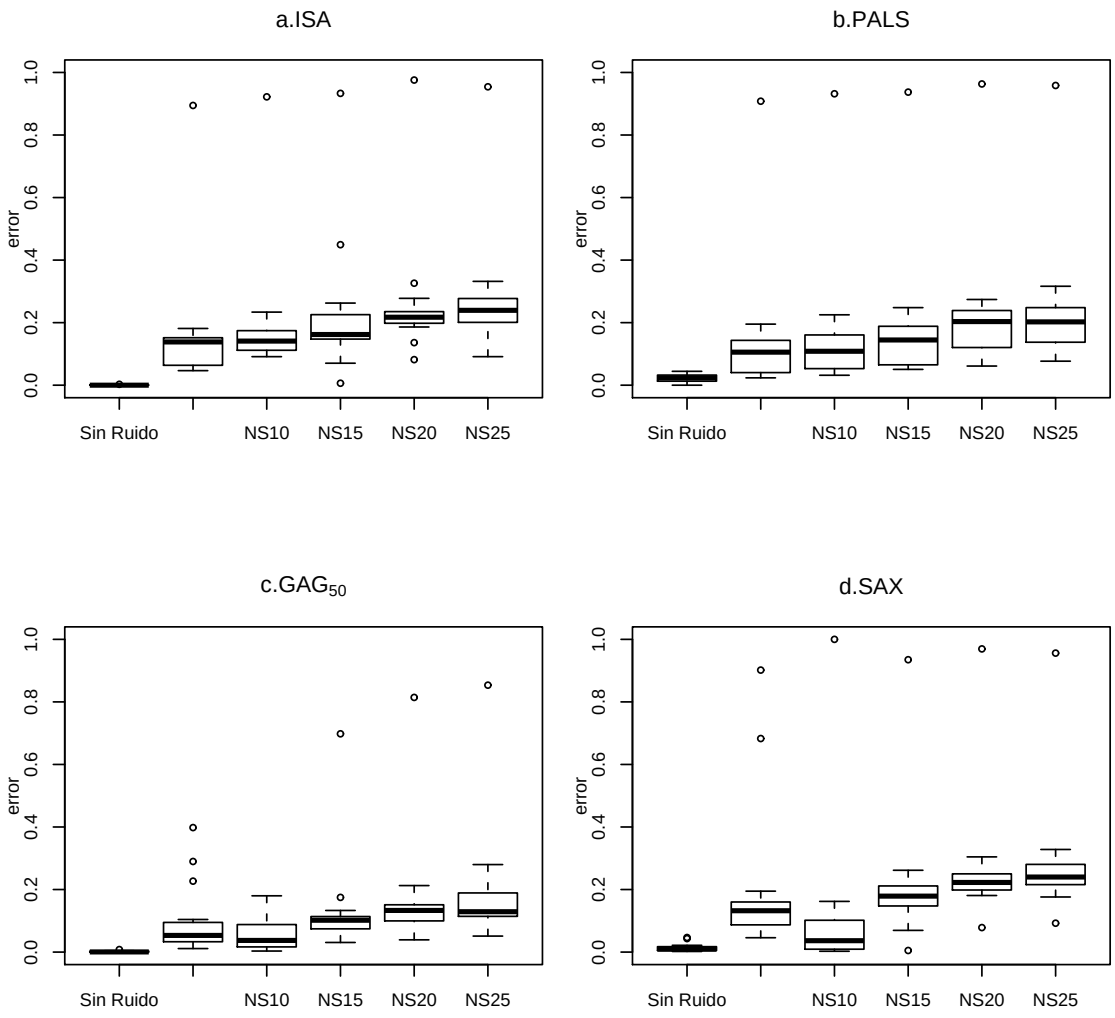


Figura 9.5: Diagramas de cajas correspondiente al error porcentual de la media de los mejores *fitness* encontrados por ISA (a), PALS (b), GAG_{50} (c) y SAX (d) en las instancias sin ruido, NS05, NS10, NS15, NS20 y NS25.

lución en ISA y GAG_{50} crece junto a la intensidad de ruido. En tanto que para PALS y SAX esta característica se hace presente a partir de las instancias NS15.

- En cuanto a la robustez, ISA presenta mayores diferencias entre los resultados provenientes de las instancias sin y con ruido que el resto de los algoritmos. Esto indica

que ISA es el ensamblador menos robusto a la hora de resolver casos con ruido en la matriz de solapamiento, independientemente de su intensidad. En cambio, PALS, GAG_{50} presentan falencias en la robustez a partir de NS20 y SAX a partir de NS15.

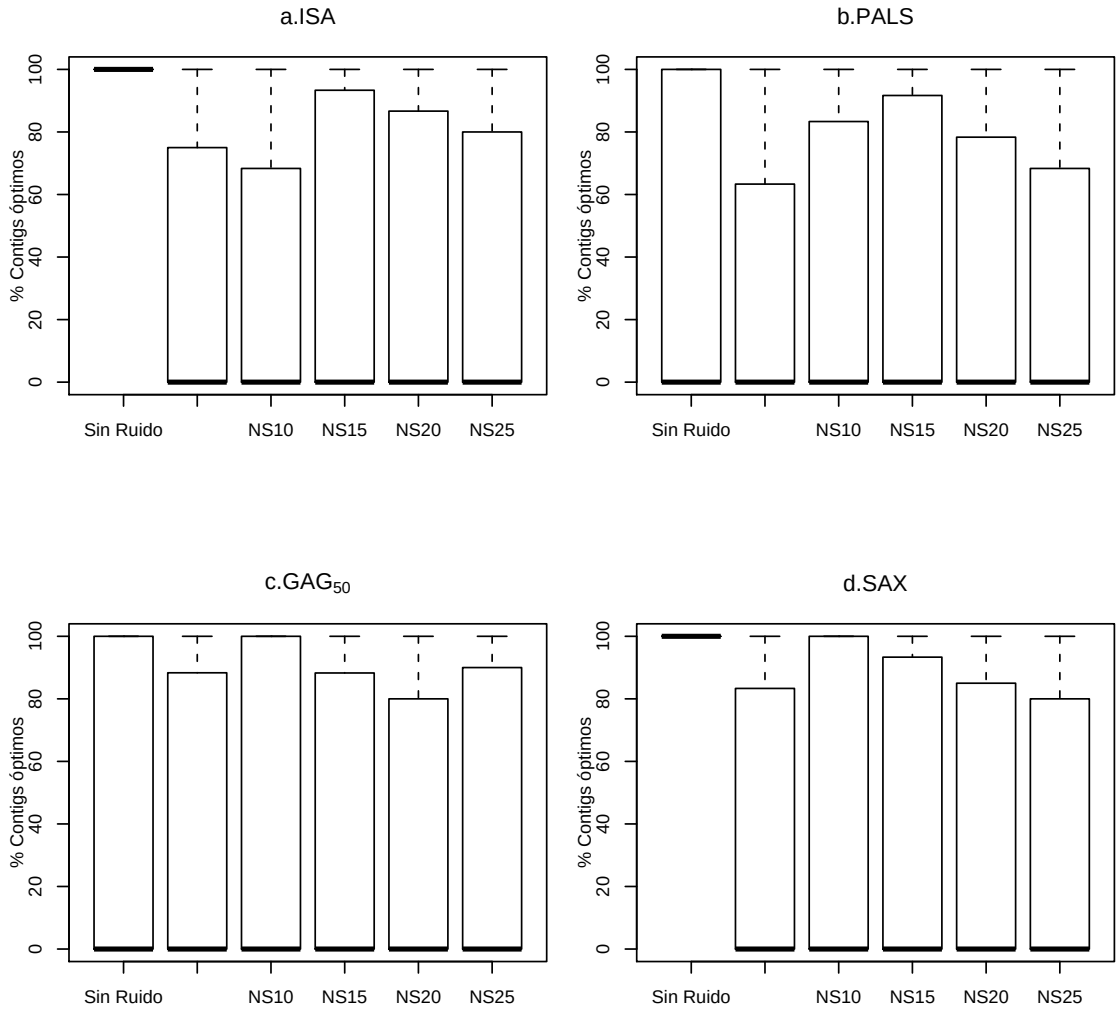


Figura 9.6: Diagramas de cajas correspondiente al porcentaje medio de veces que ISA (a), PALS (b), GAG_{50} (c) y SAX (d) encuentran el número de contigs óptimos en las instancias sin ruido, NS05, NS10, NS15, NS20 y NS25.

A continuación se analiza la calidad desde el punto de vista de los contigs finales alcanzados por cada ensamblador siguiendo la figura 9.6:

- Los cuatro ensambladores tienen dificultades con las instancias ruidosas, bajo todas sus intensidades, dado que no logran encontrar una distribución óptima en la mitad de los casos. Sin embargo, PALS y SAX alcanzan esta distribución de fragmentos en todas las ejecuciones realizadas sobre 4 de las 16 instancias NS10 (25 % de los casos). Además, son los que más contigs óptimos alcanzan para el resto de las instancias bajo las restantes intensidades de ruido (diferencias menores entre el tercer cuartil y el bigote superior).

- En cuanto al análisis de robustez, los diagramas de caja de la figura 9.6 indican que ISA y SAX presentan diferencias significativas entre los resultados obtenidos para las instancias sin y con ruido. Por ende, muestran no ser robustos para resolver instancias con errores en la matriz de solapamiento. En cambio, para PALS y GAG_{50}, este estudio marca que no existen diferencias importantes entre los resultados que provienen de los casos sin y con ruido. Por consiguiente PALS y GAG_{50} son considerados robustos a la hora de resolver instancias NS.

Del estudio realizado sobre la calidad de los resultados se concluye que: PALS es el mejor ensamblador para resolver los problemas de ensamblados de fragmentos con diferentes intensidades de ruido en la matriz de solapamiento; ya que demuestra tener el mejor comportamiento y ser robusto a la hora de evaluar el *fitness* y el número de contigs. Específicamente, el error relativo de la media de los mejores *fitness* encontrados por PALS es de 0.12 contra un error de 0.22 en ISA, 0.18 en GAG_{50} y 0.21 en SAX; mientras que el porcentaje de contigs óptimos hallados por PALS es del 35 % contra un 31 % de ISA, un 29 % de GAG_{50} y un 32 % de SAX.

Con el fin de analizar el esfuerzo computacional, se muestra en la figura 9.7 el tiempo medio para encontrar el mejor valor de aptitud para cada algoritmo. Como puede observarse, el tiempo de cálculo empleado por ISA, GAG_{50} y SAX no varían significativamente entre los casos sin y con ruido, siendo ISA el algoritmo con el menor tiempo de ejecución. Sin embargo, si comparamos el esfuerzo de cálculo realizado por PALS, podemos ver que en la

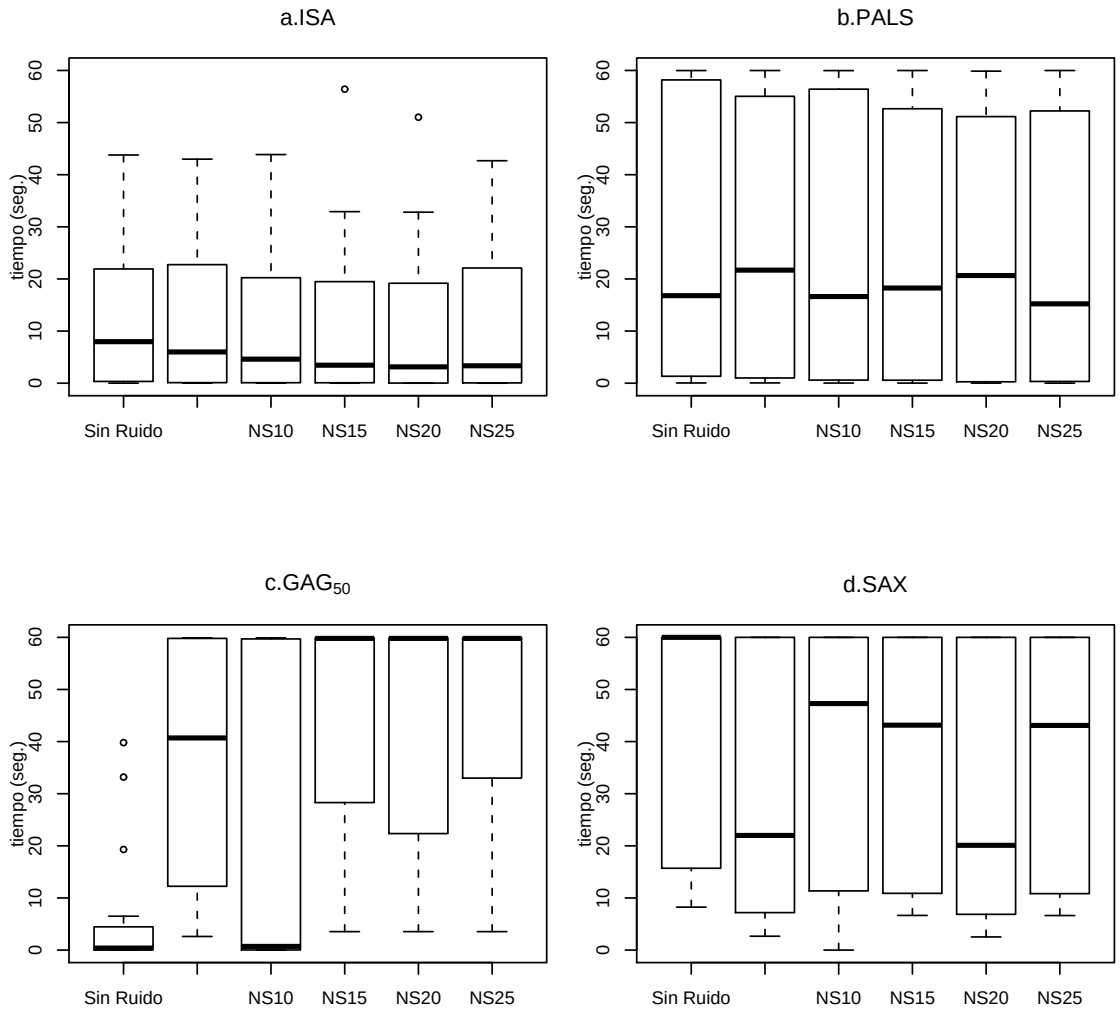


Figura 9.7: Diagramas de cajas correspondiente al tiempo promedio empleado por ISA (a), PALS (b), GAG_{50} (c) y SAX (d) para encontrar su mejor solución en las instancias sin ruido, NS05, NS10, NS15, NS20 y NS25.

resolución de casos ruidosos el esfuerzo computacional se multiplica hasta seis veces, con respecto al empleado en las instancias sin ruido. Además, PALS y SAX son los que emplean más tiempo para resolver los casos ruidosos. La razón de este incremento en PALS es el mayor número de iteraciones que realiza este ensamblador (véase tabla 9.2). Sin embargo,

en SAX la razón del incremento está dada por la generación de dos soluciones más y la aplicación del operador de crossover en cada iteración.

En general, puede inferirse que si la intensidad del ruido crece, los ensambladores tienden a converger en óptimos locales estancándose la búsqueda en esos puntos.

Tabla 9.2: Número promedio de iteraciones utilizadas por ISA, PALS, GAG_{50} y SAX para encontrar la mejor solución.

Instancias	ISA	PALS	GAG_{50}	SAX
Sin Ruido	534171.56	183.31	1894.42	337811.70
NS05	444212.85	15519.19	1281.09	215223.78
NS10	448167.98	1260.07	1621.07	314098.50
NS15	406158.74	28156.61	1560.74	320202.95
NS20	422861.10	20348.40	1569.32	263381.87
NS25	421732.19	33963.67	1515.76	288771.01

9.5. Comparación con otros algoritmos

En esta sección se compara el comportamiento de los ensambladores metaheurísticos aquí propuestos en contrapartida con el de los algoritmos ensambladores propuestos en la literatura: CAP3[93] y PHRAP (http://www.phrap.org). Con el fin de realizar dicha comparación, se muestran en la tabla 9.3 el mejor número de contigs logrados por ISA, PALS, GAG_{50}, SAX y CAP3 de casos sin y con ruido. Como CAP3 toma la información directamente de los fragmentos (y no de la matriz de solapamiento), sólo es posible usar las instancias NB para evaluarlo. Se presentan también los resultados alcanzados por PHRAP sólo para varias instancias sin ruido: *x60189*, *m15421*, *j02459* y *38524243*. La razón de esto radica en que el proceso para generar las instancias ruidosas estudiadas en este trabajo es artificial y, por ende, no tiene en cuenta la generación de los cromatogramas asociados necesarios para la aplicación de PHRAP.

Tabla 9.3: Número final de contigs obtenidos por ISA, PALS, GAG_{50}, SAX, CAP3 y PHRAP. El símbolo - indica que esta información no se proporciona.

Instancias	*ISA*	*PALS*	GAG_{50}	*SAX*	*CAP3*	*PHRAP*
x60189	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
m15421	**1.00**	1.67	1.67	**1.00**	2.00	1.50
j02459	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
38524243	**1.00**	4.50	4.50	**1.00**	5.00	4.00
acin	**1.00**	404.00	404.17	**1.00**	407.33	-
NB-x60189	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	-
NB-m15421_5	**1.67**	**1.67**	**1.67**	**1.67**	3.33	-
NB-j02459	3.00	3.00	2.00	1.97	**1.00**	-
NB-38524243	6.00	**4.50**	**4.50**	5.00	6.50	-
NB-acin	**404.00**	556.33	404.16	**404.00**	412.33	-

Al analizar la tabla 9.3, es posible detectar comportamientos similares entre PALS, GAG_{50}, CAP3 y PHRAP, especialmente entre los tres primeros, dado que no pueden encontrar diseños óptimos para los mismos grupos de instancias, por ejemplo: *m15421_6*, *m15421_7* y los grupos *38524243* y *Acin* completos. Además, esto se repite en las instancias con ruido (*NB-m15421_6*, *NB-m15421_7* y los grupos *38524243* y *Acin*). Por otra parte, se observa que ISA y SAX encuentran la distribución óptima en todas las instancias sin ruido. En cambio, para las instancias NB el comportamiento presentado por ISA y SAX es similar al del resto de los ensambladores. En otras palabras, la menor variabilidad de los resultados obtenidos por el resto de los ensambladores (PALS, GAG_{50} y CAP3) para los casos sin y con ruido muestra un comportamiento robusto. En resumen, PALS, GAG_{50}, y CAP3 son los algoritmos más robustos para lidiar con el ruido, mientras que ISA y SAX son los mejores ensambladores para la resolución de casos sin ruido. Además, es necesario enfatizar que el número de contigs obtenidos por CAP3 es igual o mayor que los obtenidos por los algoritmos metaheurísticos. Esto indica que, los ensambladores aquí propuestos brindan mejores distribuciones de fragmentos que CAP3. De esta forma, se confirma nuevamente la hipótesis **H1** (estado del arte, véase pág. 3).

9.6. Conclusiones

En este capítulo se presenta un análisis exhaustivo sobre cómo instancias sin y con ruido de FAP son resueltas por cuatro algoritmos ensambladores eficientes: ISA, PALS, GAG_{50} y SAX. Se analiza y compara el comportamiento de los algoritmos para las tres diferentes fuentes de ruido: en las bases nucleótidas, en la matriz de solapamiento y en la función de *fitness*. Para hacer este estudio se han generado conjuntos de instancias para cada fuente de ruido.

La idea detrás de este estudio es identificar qué ensamblador es más robusto para tratar el ruido en los datos de entrada. Para eso es necesario un análisis de la precisión de los resultados de los casos sin y con ruido, el tiempo de ejecución, y el comportamiento en función del tamaño de la instancia.

ISA y SAX son los ensambladores más precisos para resolver las instancias sin ruido, ya que sólo obtienen un contig para cada caso. Por el contrario, para los casos ruidosos estos ensambladores no son capacaces de resolver la mayoría de los casos ruidosos de manera satisfactoria. Esto muestra que la robustez no es una característica de estos dos algoritmos.

En general, PALS es el mejor ensamblador para resolver las instancias con ruido, ya que encuentra mejores soluciones que el resto de los ensambladores. Además, PALS este ensamblador encuentra soluciones estadísticamente similares para las instancias con y sin ruido, probando de esta forma ser un algoritmo robusto. A pesar de que no puede encontrar un único contig para las instancias más grandes. La diferencia con ISA y SAX, es que PALS trabaja con información sobre el *fitness* y el número de contigs. De esta manera, puede evaluar los casos ruidosos con mayor precisión que ISA y SAX.

Aunque GAG_{50} es un ensamblador que no alcanza el nivel de calidad logrado por ISA, PALS y SAX encuentra soluciones de calidad estadísticamente similar para las instancias con y sin ruido. Esto significa que GAG_{50} es un ensamblador robusto.

Por lo expuesto anteriormente, en este capítulo se confirman las tres hipótesis planteadas al inicio de esta tesis. En cuanto a **H1** (estado del arte, véase pág. 3), los ensambladores metaheurísticos ISA, PALS, GAG_{50} y SAX pueden obtener soluciones de mayor calidad que las proporcionadas hasta el momento por otros ensambladores (CAP3 y PHRAP). Además, estos cuatro ensambladores también confirman las hipótesis **H2** y **H2.1** (complejidad y efi-

ciencia, véase pág. 3), porque ISA y SAX manipulan instancias con más de 900 fragmentos sin ruido encontrando en menos de 60 segundos la distribución óptima de fragmentos. Además, los cuatro ensambladores (ISA, PALS, GAG_{50} y SAX) logran en menos de un minuto mantener la buena calidad del *fitness* en las instancias sin y con ruido más grandes. Por último y con respecto a **H3** (robustez, véase pág. 4), PALS y GAG_{50} han mostrado un comportamiento robusto a la hora de resolver instancias ruidosas.

Capítulo 10

Resolución de instancias con ruido en los datos usando paralelismo

A partir del análisis realizado en el capítulo 9 surge que PALS es la metaheurística más robusta a la hora de solucionar las instancias de FAP con ruido. Además, es uno de los algoritmos que mejores soluciones encuentra. Aunque, al igual que los otras ensambladores, PALS no logra obtener soluciones finales con el número óptimo de contigs, especialmente cuando resuelva instancias de gran tamaño. En los capítulos 6, 8 y 9, también se muestra que la falencia de PALS es la rápida convergencia a óptimos locales.

En un intento de lograr un ensamblador robusto que solucione eficientemente instancias ruidosas de gran tamaño, se propone una nueva metaheurística que aproveche las fortalezas de PALS y mitigue sus debilidades. Este nuevo ensamblador, al igual que PALS, estima el número de contigs y el *fitness* de cada posible movimiento de fragmentos pero, a diferencia de PALS, evita la convergencia prematura a óptimos locales.

En primer lugar, se consideró necesario forzar al algoritmo a realizar todas las posibles permutaciones de los fragmentos. Al desarrollar esta idea surgieron distintas posibilidades, a saber:

- Preservar en una lista el mejor movimiento por cada fragmento i, para su posterior aplicación. Se determinó que esto carece de sentido; ya que una vez hecho un movimiento, la solución se modifica y es posible que este intercambio altere el resto de los

movimientos escogidos. Al tratarse del intercambio de sólo un par de fragmentos, es posible que no se alteren los elementos de los restantes movimientos preservados, pero sí sus respectivos valores Δ_c y Δ_f. Entonces, ya no se sabría si tales movimientos continúan siendo los mejores. Por esto, la aplicación de esta alternativa no logró obtener resultados satisfactorios (véase apéndice C).

- Preservar y aplicar siempre el mejor movimiento de todas las combinaciones posibles para una solución. El empleo de esta opción permitió alcanzar una mejora mínima en los resultados, pero aún insuficiente para escapar de óptimos locales (véase apéndice C).

Con el propósito de escapar de las regiones de óptimos locales, se pensó que una forma para explorar otras áreas del espacio de búsqueda era: aceptar algún movimiento que no resultase el mejor posible bajo cierta probabilidad. Con esto sólo se lograron resultados insatisfactorios. Luego de analizar estos resultados y el algoritmo utilizado, se observó que para elegir un nuevo mejor movimiento se priorizaba la variación en los contigs (Δ_c) y a partir de esto se evaluaba la variación en el *fitness* (Δ_f). Por otra parte, la variabilidad del *fitness* es mucho mayor a la de los contigs, por ende, se pensó que cambiar las prioridades sería una buena estrategia para salir del estancamiento. Esta variante no resultó como se esperaba, ya que no aportó beneficios suficientes como para considerar su aplicación (véase apéndice C).

Otra alternativa fue la implementación de un algoritmo paralelo heterogéneo que seguía el diseño de un modelo isla. Donde una isla que ejecutaba ISA se conectaba con otra que ejecutaba la versión modificada de PALS propuesta al inicio de este capítulo. De esta manera, se formaba un anillo unidireccional que mantenía una comunicación asíncrona entre las islas con una frecuencia de migración establecida por un cierto número de iteraciones. Dado que, el tiempo computacional empleado por ambos algoritmos para ejecutar un mismo número de iteraciones era muy dispar, la migración se daba en muy pocas ocasiones. Esto produjo un mayor estancamiento de PALS. Por otra parte, cuando ISA recibía una nueva solución la temperatura tomaba su valor inicial y esto impedía obtener buenas soluciones. Más tarde, se estableció una comunicación síncrona entre las islas. Esto mostró una mejoría en la calidad

de los resultados, pero el aumento en el tiempo empleado para hallarlos sacó de competencia a esta alternativa (véase apéndice C).

Analizando todo el camino recorrido, es posible identificar algunos puntos claves a tener en cuenta para un buen diseño de un algoritmo basado en PALS. Tales puntos son:

- La posibilidad de analizar todas las permutaciones de fragmentos aprovechando el bajo costo computacional que esto implica, dado que no se realiza la evaluación de la función de *fitness* ni el conteo de contigs en cada paso.
- La detección del estancamiento del algoritmo y la aplicación de algún procedimiento que permita explorar otras regiones.
- La utilización de las ventajas del paralelismo y la distribución para intercambiar información y, de esta forma, proveer otro mecanismo que permita escapar de óptimos locales sin incrementar el tiempo computacional empleado.

Todo esto ha dado origen a una nueva metaheurística híbrida y paralela basada en PALS, denominada PH-PALS, para resolver instancias ruidosas del problema de ensamblado de fragmentos. La utilización de algoritmos paralelos conlleva al uso de medidas de rendimiento específicas de esta clase de algoritmos, por ende en la sección siguiente se introducen tales medidas. En la sección 10.2, se describe detalladamente este nuevo ensamblador, se analizan la calidad de los resultados obtenidos al aplicar PH-PALS y se lo compara con PALS, también se analiza el rendimiento logrado por PH-PALS. A continuación, en la sección 10.3 se contrasta el comportamiento de PH-PALS con el de otros algoritmos de la literatura. Por último, en la sección 10.4, se enuncian las conclusiones a las que se arriba.

10.1. Medidas de rendimiento

Existen distintas métricas para medir el desempeño de los algoritmos paralelos. Una de las más importantes y también más usadas es el *speedup*.

La medida de *speedup* compara dos tiempos dando como resultado la relación entre los tiempos de ejecución secuencial y el paralelo. Por lo tanto, es necesario aclarar qué se entiende por tiempo. En un sistema mono-procesador, una medida típica de rendimiento

es el tiempo de CPU para resolver el problema; se considera que este tiempo es el que el procesador ha estado ejecutando el algoritmo, excluyendo el tiempo de lectura de los datos del problema, la escritura de los resultados y cualquier tiempo debido a otras actividades del sistema. En el caso paralelo, el tiempo no es la suma de los tiempos de CPU de cada procesador, ni el más extenso de ellos. Puesto que el objetivo del paralelismo es la reducción del tiempo total de ejecución, este tiempo debería incluir claramente cualquier sobrecarga debida al uso de algoritmos paralelos. Por lo tanto, la elección más prudente para medir el rendimiento de un código paralelo es el tiempo real para resolver el problema. Esto es, medir el tiempo desde que comienza hasta que termina el algoritmo completo.

El *speedup* contrasta el tiempo de la ejecución secuencial con el tiempo correspondiente al caso paralelo a la hora de resolver un problema. Si se denota por T_n al tiempo de ejecución para un algoritmo usando n procesadores, el *speedup* es la relación entre la ejecución más rápida en un sistema mono-procesador T_1 y el tiempo de ejecución en m procesadores T_n:

$$s_m = \frac{T_1}{T_n} \tag{10.1}$$

Para los algoritmos no deterministas, como son los metaheurísticos, no es posible utilizar esta métrica directamente. En cambio, debe contrastarse el tiempo medio secuencial con el tiempo medio paralelo:

$$s_m = \frac{[T_1]}{[T_n]} \tag{10.2}$$

A partir de esta definición se puede distinguir entre: *speedup* sublineal ($s_n < n$), lineal ($s_n = n$) y superlineal ($s_n > n$). El problema con esta medida es que no existe un acuerdo entre los investigadores sobre el significado de T_1 y T_n. En [4], Alba define el *speedup* dependiendo del significado de esos valores. De esta forma, surge la taxonmomía presentada en la tabla 10.1.

El *speedup* fuerte (tipo I) compara el tiempo de ejecución paralelo respecto al mejor algoritmo secuencial. Esta es la definición más exacta de *speedup*, pero generalmente no se utiliza debido a lo complicado que resulta encontrar el algoritmo actualmente más eficiente. El *speedup* débil (tipo II) compara el algoritmo paralelo desarrollado por el investigador con su propia versión secuencial. En este caso, se pueden usar dos criterios de terminación: la

Tabla 10.1: Taxonomía de las medidas de *speedup* propuesta por Alba en [4].

I. *Speedup* fuerte
II. *Speedup* débil
A. *Speedup* con parada por calidad de soluciones
1. Versus panmixia
2. Ortodoxa
B. *Speedup* con esfuerzo predefinido

calidad de soluciones y el máximo esfuerzo. El autor de esta taxonomía descarta esta última definición debido a que compara algoritmos que no producen soluciones de similar calidad. Para el *speedup* débil con criterio de parada basado en la calidad de soluciones se proponen dos variantes: comparar el algoritmo paralelo con la versión secuencial canónica (tipo II.A.1) o comparar el tiempo de ejecución del algoritmo paralelo en un procesador con el tiempo que tarda el mismo algoritmo pero en m procesadores (tipo II.A.2). En la primera variante es claro que se están comparando dos algoritmos de comportamiento diferentes.

Puesto que el algoritmo propuesto en este capítulo no evalúa la solución en cada iteración, no es posible establecer como criterio de terminación la calidad de las mismas. Por ende, la única alternativa es determinar un número máximo de iteraciones como criterio de terminación.

10.2. Resolución de instancias ruidosas mediante PH-PALS

Con el propósito de aprovechar las fortalezas de PALS y mejorar la precisión de los resultados, se propone una nueva metaheurística basada en PALS. Este nuevo ensamblador metaheurístico, PH-PALS, aplica tres diferentes mecanismos para escapar de los óptimos locales:

1. Extensión de la búsqueda local a todos los posibles movimientos de fragmentos en una solución.

2. Aplicación de ISA cuando el mejor movimiento elegido permanece invariable luego de un cierto número de movimientos (umbral).

3. Utilización del modelo isla para ampliar la exploración en el espacio de búsqueda e intensificar la explotación en cada región del mismo.

La idea perseguida en el primer punto es forzar la aplicación de la búsqueda local a todos los posibles movimientos, pero sólo mantener el mejor. En tanto que, el objetivo del segundo es introducir una hibridación auto-adaptativa, ya que ISA es aplicado si después de un cierto número de movimientos analizados (umbral) no se modifica el movimiento escogido para modificar la solución actual. Siguiendo la clasificación de metaheurísticas híbridas, se trata de una hibridación por relevos de bajo nivel (véase sección 3.1.4.1, en el capítulo 3).

Finalmente este algoritmo híbrido (véase algoritmo 10), denominado H-PALS, es distribuido en pequeñas islas. En cada una de estas islas, el algoritmo toma ventaja de la arquitectura multi-núcleo del procesador. Es decir, el número de soluciones (o individuos) de cada isla coincide con el número de núcleos en el procesador. En consecuencia, cada isla genera tantos *threads* como núcleos posea el procesador, donde cada *thread* resuelve sólo una permutación. De esta forma, surge PH-PALS un algoritmo híbrido con dos niveles diferentes de paralelismo: uno de ellos dado por el modelo isla, una isla por máquina, y el otro dado sobre cada isla, una permutación es procesada en cada uno de los núcleos. El primer nivel de paralelismo da origen a una hibridación cooperativa de alto nivel.

En un modelo isla, un conjunto de soluciones (o población) es procesado independientemente de otros conjuntos. De esta forma, PH-PALS intenta evitar la convergencia prematura al preservar la diversidad dado el semi aislamiento de las poblaciones. Las islas intercambian soluciones entre ellas con una cierta frecuencia de migración y sobre una determinada topología de comunicación. De esta manera, PH-PALS permite la cooperación al explotar áreas prometedoras encontradas por otras islas. Específicamente, PH-PALS dispone las islas en un anillo unidireccional y utiliza una comunicación asíncrona; mientras que la frecuencia de migración se produce cuando cada individuo en una isla ha sido procesado durante 25000 iteraciones. Además la isla emisora elige la mejor solución para enviar y la receptora selecciona y reemplaza a su peor individuo si el recibido es mejor. En la figura 10.1 se ilustra el modelo de diseño para PH-PALS, donde el *master* realiza las siguientes tareas: *(1)* generación de n islas, *(2)* recepción del mejor individuo desde cada isla y *(3)* selección y devolución del mejor individuo recibido. En tanto que cada isla (i) lleva a cabo estas tareas: *(1)* generación de m

Algoritmo 10 H-PALS

```
k = 0;
inicializar S; {Genera la solución inicial}
repeat
  L = ∅;
  best Δc=best Δf=0;
  for i = 0 to N do
    for j = 0 to N do
      Δc, Δf = CalcularDelta(S, i, j); {ver alg. 11}
      if (Δc ≤ 0) && (Δc < best Δc) && (Δf > best Δf) then
        L = L∪ <i, j, Δc, Δf>; {Agregar movimiento candidato a L}
        best Δc=Δc
        best Δf=Δf
        t = 0;
      else
        t = t + 1;
      end if
    end for
    {K es el número máximo de iteraciones por individuo}
    if (umbral ≤ t) && (k < K) then
      AplicaISA(S, k); {Aplica ISA a la solución actual}
      L = ∅;
      best Δc=best Δf = t = 0;
    end if
  end for
  if L <> ∅ then
    <i, j, Δc, Δf> = Extraer(L); {Selecciona el mejor movimiento en L}
    AplicaMovimiento(S, i, j); {Modifica la solución}
    k = k + 1;
  end if
until k >= K
return S;
```

individuos y m *threads*, *(2)* cada *thread* ejecuta H-PALS sobre un individuo, *(3)* selección del mejor individuo recibido y el envío del mismo a la isla siguiente $(i+1)$, *(4)* recepción del mejor individuo desde la isla anterior $(i-1)$, *(5)* reemplazo del peor individuo si el entrante es mejor y *(6)* si todos los *threads* finalizan, seleccionar el mejor individuo y enviarlo al *master*.

En los experimentos realizados hasta el momento, se detecta una mejor desempeño de las metaheurísticas al utilizar la estrategia voraz, propuesta en el capítulo 5, para generar

Algoritmo 11 CalcularDelta

$\Delta_c = 0$;
$\Delta_f = 0$;
{Calcula la variación del puntaje de solapamiento}
$\Delta_f = w_{s[i-1],s[j]} + w_{s[i],s[j+1]}$; {Incrementa el puntaje de solapamiento de la solución modificada}
$\Delta_f = \Delta_f$ - $w_{s[i-1],s[i]}$ - $w_{s[j],s[j+1]}$; {Decrementa el puntaje de solapamiento de la solución actual}
{Verifica si un contig es dividido, y si es así, incrementa el número de contigs}
if ($w_{s[i-1],s[i]} >$ cutoff) || ($w_{s[j],s[j+1]} >$ cutoff) **then**
  $\Delta_c = \Delta_c + 1$;
end if
if ($w_{s[i-1],s[j]} >$ cutoff) || ($w_{s[i],s[j+1]} >$ cutoff) **then**
  $\Delta_c = \Delta_c$ - 1;
end if
return Δ_c, Δ_f;

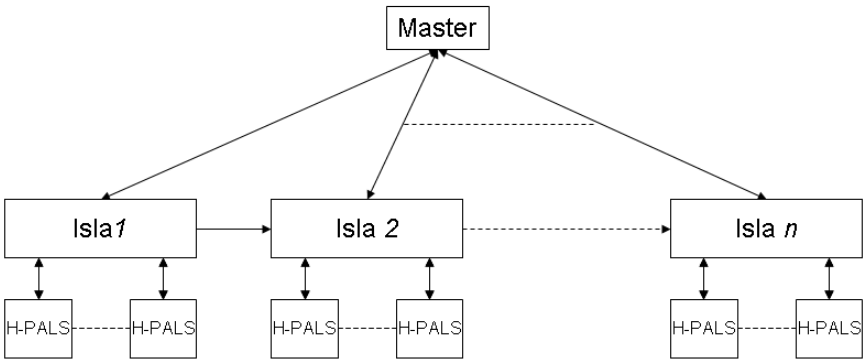


Figura 10.1: Modelo de PH-PALS

la solución con la que inician la búsqueda. Por esta razón, PH-PALS usa la mencionada estrategia para generar los primeros individuos que formarán parte de la población de cada isla.

Por otra parte, también se ha considerado una versión panmíctica de este algoritmo, denominada PanH-PALS, donde una población de $m \times n$ individuos es procesada en un solo procesador. En este caso, cada individuo es procesado separadamente de los otros usando H-PALS durante 500000 iteraciones.

10.2.1. Análisis de Resultados

En este apartado se estudia el comportamiento de PALS, PanH-PALS y PH-PALS cuando resuelven instancias ruidosas de FAP. Para ello, se analiza cómo estos ensambladores conducen la búsqueda de soluciones ante instancias con ruido proveniente de diferentes fuentes de ruido (NB, NF y NS10, véase capítulo 9).

Esta experimentación se lleva a cabo sobre un cluster de 12 computadoras con procesadores Intel Core 2 Duo a 3 GHz, conectadas por medio de Fast Ethernet, con un sistema operativo perteneciente a la distribución SuSe de Linux con una versión del kernel 2.4.19-4GB. Para estudiar el rendimiento de PH-PALS sobre un número diferente de procesadores, PH-PALS se distribuye en 3, 6, 9 y 12 islas. En cada procesador, una isla procesa dos permutaciones en paralelo, una por núcleo. Por otra parte, la versión panmíctica también ha sido ejecutada. Con el objetivo de llevar a cabo dicha experimentación, dar soporte estadístico y validar las conclusiones, por cada versión algorítmica se realizan 30 ejecuciones independientes usando los parámetros mostrados en la tabla 10.2. Al igual que en el capítulo anterior, los resultados experimentales no son tabulados aquí dado que en cada experimentación se emplean un total de 64 instancias. Estas tablas pueden encontrarse en el apéndice B. En cambio, los resultados son compendiados en gráficos que facilitan la comprensión del desempeño de cada uno de los algoritmos ejecutados.

Tabla 10.2: Valores paramétricos usados por ISA, H-PALS, PH-PALS y PanH-PALS.

	Parámetro	***Valor***
ISA	Longitud de la cadena de Markov	10
	Temperatura inicial	0.99
H-PALS	Selección del movimiento	El mejor movimiento
	Umbral	$Nmero\ De\ Fragmentos \times 5$
	Número máx. de iteraciones (K)	500000
PH-PALS	Número de individuos por isla (m)	2
	Número de islas (n)	3, 6, 9 y 12
	Frecuencia de migración	25000 iteraciones
	Política de migración	El mejor individuo es enviado El peor individuo es reemplazado si el entrante es mejor
PanH-PALS	Tamaño de la población	$n \times m$

A continuación, se analiza la calidad de los resultados obtenidos por PanH-PALS y PH-PALS para las instancias sin y con ruido de FAP y se los compara con los alcanzados por PALS en el capítulo anterior. Luego, se estudia el desempeño de PH-PALS al analizar la medida de *speedup*.

En la figura 10.2, se muestran los resultados obtenidos por PALS, PanH-PALS y PH-PALS bajo las tres fuentes de ruido y se comparan con los obtenidos para los casos sin ruido. Es necesario recordar que, PALS no evalúa la función de *fitness* durante la búsqueda, por ende resolver las instancias NF con PALS carece de sentido.

Al analizar la figura 10.2, se detecta que:

- Para las instancias más pequeñas (*x60189*), PALS, PanH-PALS y PH-PALS encuentran el número óptimo de contigs para todos los casos sin y con ruido.

- Para las instancias un poco más grandes (*m15421*), los tres ensambladores continúan mostrando un buen comportamiento ya que encuentran soluciones con dos contigs como máximo. Además, PALS, PanH-PALS y PH-PALS presentan un patrón de comportamiento similar al resolver instancias sin ruido y NB. En cambio, si se trata de las instancias NF-*m15421*, PanH-PALS obtiene soluciones ligeramente mejores que PH-PALS. Finalmente, para los casos *NS10-m15421* PALS y PH-PALS encuentran soluciones de mayor calidad que PanH-PALS.

- Para las instancias con número de acceso *j02459* y 352 fragmentos, se encuentran más diferencias que similitudes en el comportamiento de los tres ensambladores. Sin embargo, todos ellos hallan soluciones con tres o menos contigs. Para las instancias sin ruido, NF y NS10, PALS y PH-PALS encuentran soluciones óptimas; mientras que PanH-PALS sólo logra soluciones con dos y tres contigs. En cambio, para los casos NB, PALS y PanH-PALS hallan el número óptimo de contigs; en tanto que, PH-PALS no puede encontrar la solución óptima.

- Para instancias de mayor tamaño (*38524243*), ninguno de los tres ensambladores encuentra la solución óptima. Específicamente, para los casos sin ruido, NF y NS10, los tres ensambladores se comportan de manera similar, ya que en promedio logran soluciones con cinco contigs. Sin embargo, para las instancias NB, PALS es la mejor

opción; mientras que PanH-PALS y PH-PALS encuentran soluciones con 35 o más contigs.

- Para las instancias más grandes (*acin*), el número de contigs encontrado por PanH-PALS y PH-PALS se reduce más del 50 % en comparación con los resultados encontrados por PALS, aunque ninguno de ellos alcance el número óptimo de contigs. En particular, para las instancias sin ruido y NF, PanH-PALS muestra un comportamien-

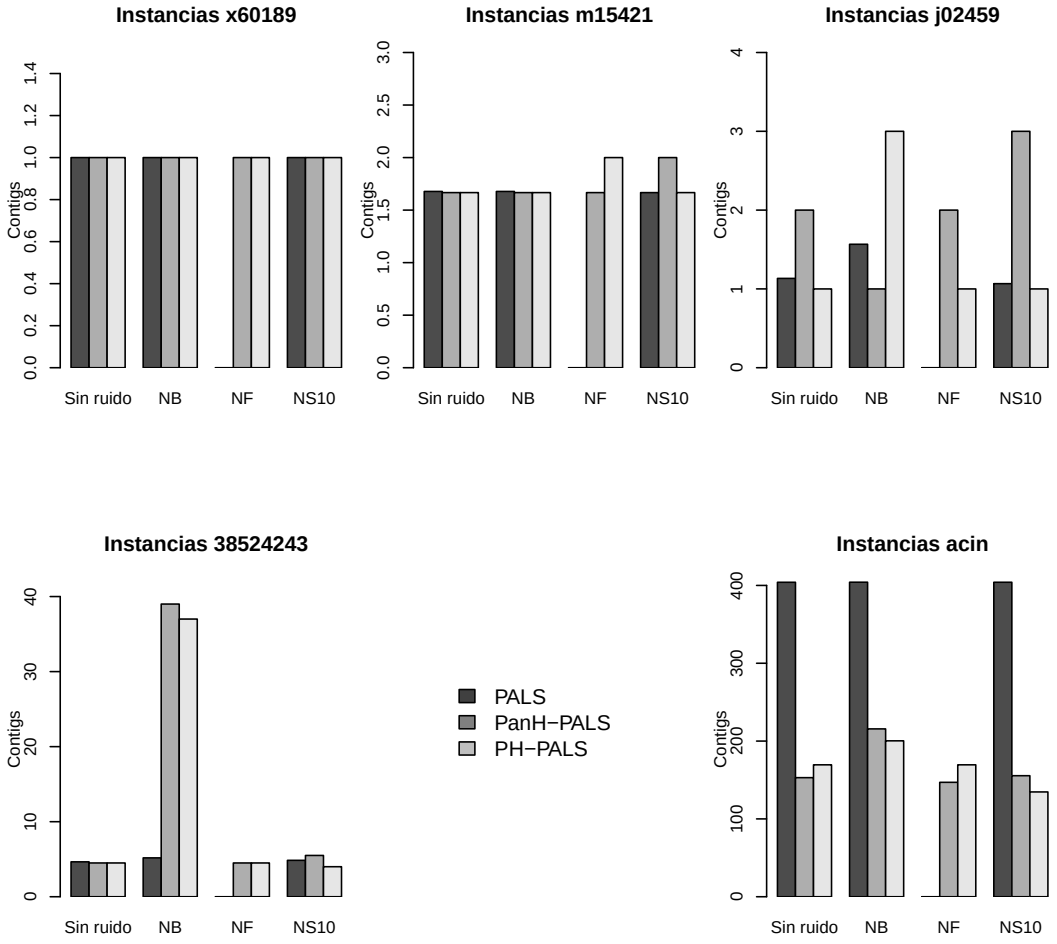


Figura 10.2: Número promedio de contigs encontrados por PALS, PanH-PALS y PH-PALS para cada grupo de instancias sin y con ruido. (PALS no se aplica a las instancias NF.)

to mejor que PH-PALS, pero para las instancias NB y NS10, PH-PALS se comporta mejor que PanH-PALS.

En resumen, cuando PanH-PALS y PH-PALS resuelven las instancias más grandes, el objetivo planteado al inicio de este capítulo es cumplido. Es decir, estos ensambladores son capaces de escapar de los óptimos locales y reducir el número de contigs en las soluciones correspondientes a las instancias de mayor tamaño. Para las restantes instancias ambas propuestas se comportan como PALS; excepto para las instancias NB-38524243. Además, tanto PanH-PALS como PH-PALS al igual que PALS son ensambladores robustos ya que los resultados para los casos sin y con ruido son similares.

La figura 10.3 muestra el desempeño de PH-PALS considerando la medida de *speedup* débil propuesto en [4], dado que PH-PALS es un algoritmo no determinista. Por otra parte, PanH-PALS y PH-PALS son dos algoritmos claramente diferentes. El segundo intercambia información útil para guiar la búsqueda hacia regiones prometedoras, mientras que el primero no lleva a cabo el cruce de información. Con el propósito de realizar un estudio profundo del *speedup*, PH-PALS se ejecuta sobre 3, 6, 9 y 12 procesadores usando 3, 6, 9 y 12 islas, respectivamente.

Se analiza el *speedup* de PH-PALS para las instancias sin ruido y las correspondientes a las tres fuentes de ruido (NB, NF y NS10). De este estudio (veáse figura 10.3), se observa que:

- En general, el *speedup* crece al incrementar el número de procesadores, mostrando que la eficiencia de PH-PALS aumenta con el incremento del número de procesadores. Excepto para las instancias *NB-x60189* y *NF-j02459*, donde el *speedup* se reduce para 9 y 12 procesadores, respectivamente. Por otra parte, si se observan los datos sobre costo temporal en el apéndice B se detecta que PH-PALS, cuando es ejecutado sobre 12 procesadores, generalmente disminuye el tiempo de ejecución total en más de un 21 % con respecto a PanH-PALS.

- Un *speedup* sublineal en todos los casos, dado que es menor a los respectivos número de procesadores. En general, PH-PALS alcanza su mayor *speedup* cuando resuelve las

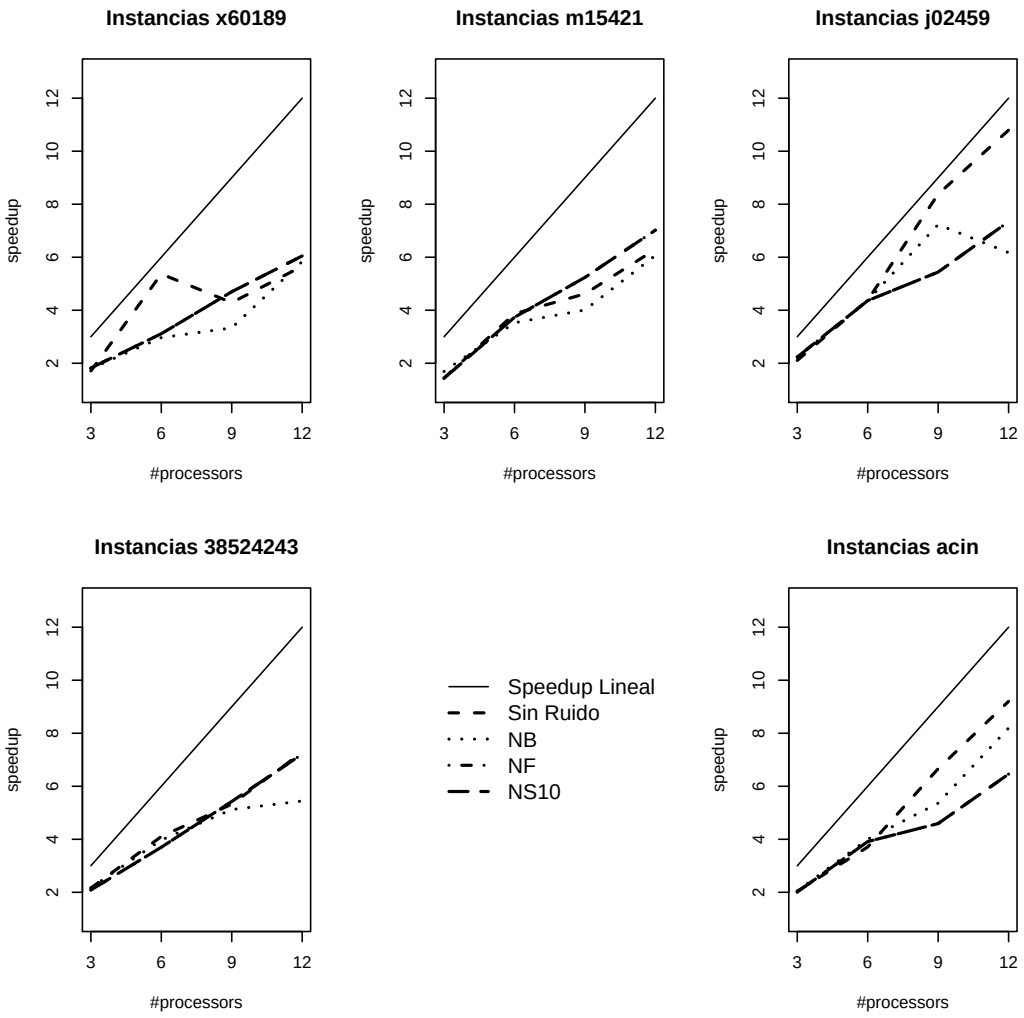


Figura 10.3: Medida de *speedup* para PH-PALS bajo diferente número de procesadores.

instancias sin ruido. Aunque, las diferencias entre las instancias sin y con ruido no son estadísticamente significativas.

10.3. Comparación con otros algoritmos

En esta sección se compara el comportamiento del ensamblador propuesto en este capítulo con otros propuestos en la literatura como CAP3 [93] y PHRAP (http://www.phrap.org). También se lo contrasta con los propuestos en esta tesis: ISA, PALS, GAG_{50} y SAX.

En la tabla 10.3 se muestra el número promedio de contigs alcanzados por PH-PALS, ISA, PALS, GAG_{50}, SAX, CAP3 y PHRAP para las instancias sin ruido y las NB. Sólo es posible la comparación con las instancias NB dado que CAP3 toma información desde los fragmentos y no de la matriz de ruido. Además, los ensambladores como CAP3 y PHRAP no utilizan la función de *fitness* usada por las metaheurísticas, por lo tanto tampoco es posible resolver las instancias NF. Con respecto a PHRAP, sólo se presentan los resultados para varias instancias sin ruido: *x60189*, *m15421*, *j02459* y *38524243*. La razón de esto radica, como se mencionó en capítulos anteriores, en que el proceso para generar instancias ruidosas en este trabajo no considera los cromatogramas asociados, los cuales son necesarios para aplicar PHRAP.

Tabla 10.3: Número promedio de contigs obtenidos por PH-PALS, ISA, PALS, GAG_{50}, SAX, CAP3 y PHRAP. El símbolo - indica que esta información no se proporciona. Los mejores valores están remarcados en negro.

Instancias	*PH-PALS*	*ISA*	*PALS*	GAG_{50}	*SAX*	*CAP3*	*PHRAP*
x60189	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
m15421	1.67	**1.00**	1.67	1.67	**1.00**	3.33	1.50
j02459	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
38524243	4.50	**1.00**	4.50	4.50	**1.00**	5.00	4.00
acin	169.5	**1.00**	404.00	404.17	**1.00**	407.33	-
NB-x60189	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	-
NB-m15421	**1.67**	**1.67**	**1.67**	**1.67**	**1.67**	3.33	-
NB-j02459	2.00	3.00	3.00	2.00	1.97	**1.00**	-
NB-38524243	37.00	6.00	**4.50**	**4.50**	5.00	6.50	-
NB-acin	**200.33**	404.00	556.33	404.16	404.00	412.33	-

Al analizar la tabla 10.3, se detecta que PH-PALS supera a PALS, GAG_{50} y CAP3 cuando resuelve las instancias sin ruido *acin*; ya que el número promedio de contigs se reduce aproximadamente un 58 %. Para el resto de las instancias sin ruido, PH-PALS se comporta de manera similar a estos ensambladores. En cambio, para las instancias *NB-acin* PH-PALS supera a todos estos ensambladores, incluyendo a ISA y a SAX, al reducir el número medio de contigs entre un 62 y un 74 %. Para el resto de las instancias NB, excepto para *NB-38524243*, PH-PALS alcanza la calidad lograda por los restantes ensambladores.

10.4. Conclusiones

En este capítulo se presentó una nueva metaheurística híbrida y paralela basada en PALS, denominada PH-PALS, para resolver instancias ruidosas de FAP. Este nuevo desarrollo se hibrida con ISA y se distribuye en islas. El uso de ISA le permite a PH-PALS escapar de óptimos locales; mientras que la paralelización incrementa su rendimiento. Se establecen dos diferentes niveles, uno dado por las islas que se ejecutan en procesadores distintos (uno por isla), y otro dado dentro de cada isla ya que cada individuo se ejecuta en un núcleo distinto del procesador.

La calidad de los resultados encontrados por PH-PALS supera ampliamente a ISA, PALS, GAG_{50}, SAX y CAP3, cuando las instancias ruidosas de mayor tamaño son resueltas al reducir el número medio de contigs más de un 62 %. PH-PALS también supera aproximadamente en un 58 % a PALS, GAG_{50} y CAP3 cuando las instancias más complejas sin ruido son resueltas. Para las instancias restantes, PH-PALS muestra el mismo comportamiento que los otros ensambladores.

PH-PALS confirma las tres hipótesis planteadas al inicio de esta tesis (véase pág. 3). Primero , respecto a **H1** (estado del arte) su comportamiento supera ampliamente al mostrado por CAP3 (algoritmo propuesto en la literatura), al reducir entre un 50 y un 74 % el número promedio de contigs obtenidos. Segundo, respecto a **H2** (complejidad) y **H2.1** (eficienciaa) PH-PALS demuestra ser capaz de manipular eficientemente las instancias de mayor complejidad de FAP. Por último, respecto a **H3** PH-PALS demuestra ser una metaheurística robusta en todos los escenarios de ruido y para cada una de las instancias.

Parte IV

Conclusiones y Trabajo futuro

Conclusiones

En esta tesis doctoral se ha propuesto la aplicación de técnicas metaheurísticas para resolver el problema de ensamblado de fragmentos de ADN. Este problema consiste en tres fases: una primera fase de superposición, una segunda de distribución y una tercera de consenso. La primera encuentra y determina el tamaño de la superposición (o solapamiento) entre los fragmentos. La segunda halla el orden de los fragmentos basado en el puntaje de similitud computado en la fase anterior. La tercera y última fase, consenso, deduce la secuencia de ADN usando el ordenamiento de fragmentos obtenidos anteriormente. Particularmente, el trabajo en esta tesis se enfoca a resolver el problema combinatorio NP-duro que surge al llevarse a cabo la fase de distribución. El objetivo es entonces encontrar el ordenamiento de fragmentos que permita obtener la secuencia original de ADN. El ensamblado de fragmentos de ADN es un problema resuelto en las primeras fases del proyecto del genoma y por lo tanto muy importante, ya que los demás pasos dependen de su precisión.

Antes de comenzar las investigaciones desarrolladas en este trabajo, se ha realizado una amplia exploración del estado del arte en algoritmos ensambladores, específicamente en aquellos basados en metaheurísticas. Tras este estudio inicial, se desprende que los GAs son una de las metaheurísticas más utilizadas para resolver este problema. También las metaheurísticas basadas en trayectoria, tales como SA, PALS y VNS, forman parte de la literatura asociada a FAP. A partir de esto, surgen diversas contribuciones que se describen a continuación.

- **Generación de semillas.** Se ha desarrollado una nueva estrategia voraz para generar soluciones iniciales, que incorpora información heurística sobre el problema en soluciones representadas por una permutación. Se comprobó que introducir *semillas*

en los primeros pasos del método permite, desde el inicio, guiar a la búsqueda hacia regiones prometedoras y, así, encontrar soluciones de calidad visiblemente superior a la obtenida cuando el inicio es aleatorio. Dado que en promedio el porcentaje de esta mejora es del 19 % y el costo de su aplicación es insignificante se aconseja siempre utilizar semillas generadas mediante dicha técnica voraz como estrategia de inicio.

- **Generación de instancias de mayor complejidad.** Con el objetivo de realizar un estudio del comportamiento de los ensambladores propuestos que permita concluir sobre el problema en general, resulta necesario analizar un número superior de instancias y también de mayor complejidad que las proporcionadas por la literatura. Por lo tanto, se ha implementado DNAgen un generador de instancias con el cual se obtuvieron un nuevo conjunto de instancias de alta complejidad. Con la utilización de DNAgen se crea un nuevo grupo de instancias, denominadas *acin*. Estas instancias se caracterizan por la alta complejidad dada por el tamaño de las mismas, es decir que están conformadas por entre 307 y 1049 fragmentos con una longitud promedio de 1000 bases cada uno.

  Pero este estudio es realmente significativo cuando se abordan problemas reales originados por el ruido en los datos. En este trabajo se han identificado tres fuentes de ruido (el proceso de secuenciación, la fase de solapamiento y el cálculo de fitness), en función a esto se han generado nuevos grupos de instancias acorde a dichas fuentes.

  La incorporación de todas estas nuevas instancias (sin y con ruido) a la experimentación ha permitido concluir de manera general sobre el problema con respecto a la precisión, eficiencia y robustez de los algoritmos propuestos en esta tesis.

- **Diseño, desarrollo y evaluación de nuevos ensambladores metaheurísticos basados en trayectoria.** ISA y SAX son dos nuevos algoritmos basados en SA, el primero utiliza un procedimiento de inversión para generar vecinos y el segundo se hibrida con el operador genético OX. FVNS y CVNS son dos ensambladores basados en VNS, el primero maximiza el puntaje de solapamiento y el segundo minimiza el número de contigs. PH-PALS es un ensamblador paralelo basado en PALS que se

hibrida con ISA. El comportamiento de cada uno de estos algoritmos es contrastados con PALS, una metaheurística especialmente diseñada para resolver FAP.

A partir de la aplicación de todos ellos a FAP se recomienda: aplicar ISA y SAX para resolver las instancias sin ruido, ya que son los ensambladores que resuelven con mayor precisión y eficiencia estas instancias al encontrar siempre la solución óptima en menos de 60 segundos. También se sugiere el uso de PH-PALS para resolver instancias ruidosas (cualquiera sea la fuente de ruido), dado que encuentra soluciones que reducen en más del 50 % el número de contigs con respecto a las halladas por el resto de los ensambladores propuestos. Además, logra una reducción importante del costo temporal al utilizar dos niveles de paralelismo.

Por otra parte, se detecta que PALS requiere menos tiempo de ejecución que ISA y SAX. Pero, dado que la calidad de los resultados obtenidos por éste es inferior a la lograda por los dos ensambladores basados en SA, se infiere que PALS converge rápidamente a óptimos locales en las instancias de mayor complejidad (sin y con ruido).

Con respecto a las hipótesis planteadas al inicio de esta tesis, ISA, PALS, SAX y PH-PALS confirman las hipótesis **H1** (estado del arte) y **H2** (complejidad y eficiencia), mientras que sólo PALS y PH-PALS confirman la **H3** (robustez).

- **Diseño, desarrollo y evaluación de nuevos ensambladores metaheurísticos basados en población.** Se han propuesto nuevos ensambladores basados en GAs que utilizan distintas estrategias de inicio (aleatoria, 2-opt y voraz), además de aplicar diferentes operadores de cruce (PMX, OX, CX y EX). De esta forma surgen los siguientes algoritmos: $GA2o_{50}$, $GA2o_{100}$, GAG_{50} y GAG_{100}. También se propone GA+VNS, que nace de hibridar a GAG_{50} con FVNS como un tercer operador genético. De la aplicación de todos ellos para resolver FAP surge que: el ensamblador, basado en población, que logra un mejor compromiso entre calidad y costo computacional es GAG_{50} cuando utiliza OX. Pero, cuando se compara el comportamiento de GAG_{50} con el de los ensambladores basados en trayectoria, se detecta que estos últimos superan a la calidad obtenida por GAG_{50} en más del 7 % en las instancias de mediana y alta complejidad (sin y con ruido). En cuanto al costo temporal, sólo ISA y PH-PALS se diferencian de

GAG_{50} requiriendo tiempos de ejecución significativamente menores al empleado por este GA. Finalmente, GAG_{50} confirma las 3 hipótesis planteadas.

En resumen, para resolver FAP usando ensambladores metaheurísticos se recomienda el uso de estrategias de inicio heurísticas, como es la técnica voraz propuesta en esta tesis. De esta forma, el ensamblador mejora la calidad promedio de los resultados finales en un 19 % siendo insignificante el costo computacional resultante de aplicar esta estrategia.

Por otra parte, se sugiere la utilización de ensambladores metaheurísticos basados en trayectoria, dado que en general son más precisos y eficientes que los basados en población. En particular, para las instancias sin ruido se recomienda el uso de ISA y SAX por obtener siempre la distribución óptima de fragmentos en menos de 60 segundos. En tanto que para las instancias ruidosas, el ensamblador que brinda mayor precisión y eficiencia en los resultados es PH-PALS. Es decir, este ensamblador logra soluciones con al menos un 50 % más de calidad que el resto de los algoritmos. Además, se trata de un ensamblador robusto.

Trabajos futuros

Diversas son las líneas de trabajo que surgen de las investigaciones desarrolladas en esta tesis doctoral. A continuación se sintetizan las principales:

- En reiteradas ocasiones se comprobó, que sacrificar el puntaje de solapamiento medido por la función de *fitness* permitía hallar una secuencia de ADN con un solo contig. En otras palabras, se comprobó que esta función de *fitness* no contemplaba una de las características más importantes del problema en cuestión. Entonces, se propone diseñar una medida de calidad que, por un lado, considere el número de contigs pero, por otro, evite las características de una función deceptiva.

- La incorporación de información heurística al inicio de las metaheurísticas ha dado muy buenos frutos, por ende incorporar esta clase de información durante la búsqueda es una tarea pendiente que promete mejorar la calidad de los resultados. Esto permitiría tomar mejores decisiones sobre las regiones de búsqueda a explorar y explotar, y así se evitarían óptimos locales y reduciría el tiempo de ejecución.

- En cuanto a la manipulación de instancias ruidosas queda mucho trabajo por delante. En primer lugar, se propone incrementar el *speedup* de PH-PALS con el fin de obtener como mínimo un *speedup* lineal. También es de interés diseñar una versión de PH-PALS heterogénea, que permita una exploración simultánea y controlada del espacio de búsqueda con el objetivo de seguir reduciendo el número de contigs hasta hallar el óptimo. Otra propuesta relacionada con la mejora en la calidad de las soluciones es el intercambio de información útil entre los individuos en una isla. Esto podría llevarse a cabo por medio de la aplicación de algún operador de cruce entre ellos.

Parte V

Apéndices

Apéndice A

Resultados experimentales de ISA, PALS, GAG_{50} y SAX para las instancias con ruido

En este apéndice se muestran las tablas de resultados experimentales obtenidos por las metaheurísticas propuestas en el capítulo 9 (ISA, PALS, GAG_{50} y SAX), para resolver instancias con distintas fuentes de ruido. Los errores o ruido pueden ocurrir durante: la secuenciación generando errores en las bases nucleótidas (NB), la fase de superposición creando errores en la matriz de solapamiento (NS) y en el cálculo del *fitness* obteniendo valores de *fitness* ruidosos (NF).

Dadas las características de las fuentes de ruido NB y NF no es factible simular diferentes intensidades de ruido en estos casos (véase capítulo 9). En cambio, sí es posible simular distintas intensidades cuando la fuente de ruido está presente en la matriz de solapamiento. Por esta razón, se estudian los siguientes porcentajes de ruido: 5, 10, 15, 20 y 25; cuyos conjuntos de instancias son identificados con NS05, NS10, NS15, NS20 y NS25 respectivamente.

En las tablas A.1 y A.2 se muestran los resultados experimentales obtenidos para las instancias sin ruido, en las tablas A.3 y A.4 los conseguidos para las instancias NB. En tanto que, en las tablas A.5 y A.6 se presentan los resultados logrados para las instancias NF y,

por último, los obtenidos para los 5 conjuntos de instancias NS en las tablas A.7, A.8, A.9, A.10, A.11, A.12, A.13, A.14, A.15 y A.16.

Tabla A.1: Resultados experimentales de ISA, PALS, GAG_{50} y SAX para las instancias sin ruido. Los mejores valores están remarcados en negro.

Instancias	*Mejor fitness*				*Fitness medio*				*Test*
	ISA	PALS	GAG_{50}	SAX	ISA	PALS	GAG_{50}	SAX	*KW*
x60189_4	**11478**	**11478**	**11478**	**11478**	11332.90	11416.17	**11478.00**	11380.47	+
x60189_5	**14027**	14021	13553	**14027**	13872.40	13758.47	13275.50	**13999.20**	+
x60189_6	**18301**	**18301**	17866	**18301**	**18147.93**	17890.43	17414.90	18131.33	+
x60189_7	**21271**	21210	20884	21268	20913.10	20832.83	20737.23	**21070.37**	-
m15421_5	38583	38526	37932	**38726**	38474.17	38402.30	37506.83	**38557.97**	+
m15421_6	**48048**	**48048**	47152	**48048**	47891.70	**47925.13**	46788.87	47894.03	-
m15421_7	55048	55067	52702	**55072**	54702.47	54525.20	52277.93	**54789.90**	+
j02459_7	**116257**	115320	110869	115301	**115164.87**	114575.13	110223.87	114160.87	+
38524243_4	**226538**	225783	218250	223029	**225647.07**	224833.10	217186.55	221677.70	+
38524243_7	436739	**438215**	417702	417680	433482.17	**436645.40**	416011.13	415455.13	+
acin1	44511	**46876**	45565	46865	44258.94	**46758.20**	45377.03	46636.97	+
acin2	138699	**144634**	143444	144567	136719.33	**144112.00**	142499.33	143972.87	+
acin3	152177	**156776**	154947	155789	150193.90	**156507.50**	153980.90	154991.57	+
acin5	143864	**146591**	145332	145880	142770.50	**146563.80**	145193.87	145309.40	+
acin7	155117	**158004**	155873	157032	154977.53	**157972.80**	155801.40	156939.40	+
acin9	311035	**325930**	313203	314354	308603.93	**324620.30**	312005.55	311863.23	+

Tabla A.2: Resultados experimentales de ISA, PALS, GAG$_{50}$ y SAX para las instancias sin ruido (cont.). Los mejores valores están remarcados en negro.

Instancias	*% Contigs óptimos*				*Test*	*Tpo. medio de la mejor solución*				*Test*
	ISA	PALS	GAG$_{50}$	SAX	*KW*	ISA	PALS	GAG$_{50}$	SAX	*KW*
x60189_4	**100.00 %**	**100.00 %**	**100.00 %**	**100.00 %**	-	0.01	**0.00**	0.22	0.21	-
x60189_5	**100.00 %**	**100.00 %**	**100.00 %**	**100.00 %**	-	0.04	**0.00**	0.41	0.99	-
x60189_6	**100.00 %**	**100.00 %**	**100.00 %**	**100.00 %**	-	0.10	**0.01**	0.69	1.46	+
x60189_7	**100.00 %**	**100.00 %**	**100.00 %**	**100.00 %**	-	0.07	**0.00**	0.62	5.32	+
m15421_5	**100.00 %**	96.67 %	90.00 %	**100.00 %**	-	0.54	**0.03**	2.57	8.17	+
m15421_6	**100.00 %**	0.00 %	0.00 %	**100.00 %**	+	0.79	**0.07**	6.24	12.99	+
m15421_7	**100.00 %**	0.00 %	0.00 %	**100.00 %**	+	1.15	**0.09**	6.08	14.07	+
j02459_7	**100.00 %**	86.67 %	10.00 %	**100.00 %**	-	8.77	**0.69**	25.94	58.17	+
38524243_4	**100.00 %**	0.00 %	0.00 %	**100.00 %**	+	16.88	**1.23**	46.04	58.96	+
38524243_7	**100.00 %**	0.00 %	0.00 %	**100.00 %**	+	30.46	**1.64**	47.36	59.25	+
acin1	96.77 %	0.00 %	0.00 %	**100.00 %**	+	6.29	**0.39**	29.56	55.38	+
acin2	**100.00 %**	0.00 %	0.00 %	**100.00 %**	+	15.44	**2.42**	58.30	59.42	+
acin3	**100.00 %**	0.00 %	0.00 %	**100.00 %**	+	17.73	**6.50**	59.42	59.49	+
acin5	**100.00 %**	0.00 %	0.00 %	**100.00 %**	+	26.11	**19.30**	59.82	59.73	+
acin7	**100.00 %**	0.00 %	0.00 %	**100.00 %**	+	35.57	**33.16**	59.98	59.63	+
acin9	**100.00 %**	0.00 %	0.00 %	**100.00 %**	+	43.78	39.84	**2.27**	59.77	+

Tabla A.3: Resultados experimentales de ISA, PALS, GAG$_{50}$ y SAX para las instancias NB. Los mejores valores están remarcados en negro.

Instancias	*Mejor fitness*				*Fitness medio*				*Test*
	ISA	PALS	GAG$_{50}$	SAX	ISA	PALS	GAG$_{50}$	SAX	*KW*
x60189_4	**11478**	**11478**	**11478**	**11478**	11330.57	10600.20	**11473.37**	11415.97	+
x60189_5	**14027**	**14027**	13530	14027	13855.27	13754.40	13276.73	**13938.03**	-
x60189_6	**18301**	18131	17816	**18301**	18158.37	17779.50	17414.60	**18189.70**	+
x60189_7	**21271**	21162	20884	**21271**	**20930.97**	20856.70	20698.03	**20992.97**	-
m15421_5	38694	38593	**47617**	38721	**38515.97**	38393.10	37381.10	38515.00	+
m15421_6	**47697**	47696	47138	47694	47647.69	47644.50	46468.27	**47652.53**	+
m15421_7	**55086**	54779	52567	55069	**54723.33**	54482.20	52204.13	54721.10	-
j02459_7	4403	3100	**7244**	5318	3298.03	3017.27	3235.20	**3346.51**	-
38524243_4	**143515**	142802	139493	141750	142187.93	**142267.86**	137713.00	140800.43	+
38524243_7	**314599**	310688	297332	301568	**311525.90**	308637.80	295219.03	301568.00	+
acin1	46975	46954	45645	**47068**	46676.07	46683.00	45383.27	**46755.43**	+
acin2	25183	**25465**	25051	22048	24109.10	**24150.11**	23255.47	21962.63	+
acin3	**20894**	19669	20848	19950	19908.62	**20007.00**	19503.27	19913.70	-
acin5	16263.2759	**16349**	15556	14905	16263.28	**16431.20**	14046.63	14864.70	+
acin7	**19960.4828**	18856	18052	17920	**19960.48**	19657.10	16074.10	17846.00	+
acin9	**52071.3793**	51083	47617	49029	**52071.38**	51996.50	46567.44	47877.30	+

Tabla A.4: Resultados experimentales de ISA, PALS, GAG$_{50}$ y SAX para las instancias NB (cont.). Los mejores valores están remarcados en negro.

Instancias	*% Contigs óptimos*				*Test*	*Tpo. medio de la mejor solución*				*Test*
	ISA	PALS	GAG$_{50}$	SAX	*KW*	ISA	PALS	GAG$_{50}$	SAX	*KW*
x60189_4	**100.00 %**	**100.00 %**	**100.00 %**	**100.00 %**	-	0.02	**0.00**	0.04	0.81	-
x60189_5	**100.00 %**	**100.00 %**	**100.00 %**	**100.00 %**	-	0.01	**0.00**	0.20	3.11	+
x60189_6	**100.00 %**	**100.00 %**	**100.00 %**	**100.00 %**	-	0.01	**0.00**	0.22	0.75	-
x60189_7	**100.00 %**	**100.00 %**	**100.00 %**	**100.00 %**	-	0.02	**0.00**	0.21	2.77	+
m15421_5	96.67 %	0.00 %	90.00 %	**100.00 %**	-	**0.01**	0.03	1.90	2.27	+
m15421_6	3.33 %	**97.00 %**	0.00 %	0.00 %	+	**0.00**	0.07	3.30	4.00	+
m15421_7	0.00 %	0.00 %	0.00 %	0.00 %	-	**0.01**	0.09	4.40	6.54	+
j02459_7	46.67 %	**50.00 %**	26.67 %	33.33 %	-	0.97	**0.01**	5.54	5.88	+
38524243_4	0.00 %	0.00 %	0.00 %	0.00 %	-	**0.37**	1.95	57.81	58.77	+
38524243_7	0.00 %	0.00 %	0.00 %	0.00 %	-	0.29	**0.00**	59.22	59.62	+
acin1	0.00 %	0.00 %	0.00 %	0.00 %	-	**0.01**	0.37	32.60	35.36	+
acin2	0.00 %	0.00 %	0.00 %	0.00 %	-	**0.83**	3.66	59.32	59.06	+
acin3	0.00 %	0.00 %	0.00 %	0.00 %	-	**0.87**	9.02	59.65	59.50	+
acin5	0.00 %	0.00 %	0.00 %	0.00 %	-	**0.89**	20.52	59.96	59.71	+
acin7	0.00 %	0.00 %	0.00 %	0.00 %	-	**0.87**	35.82	55.99	59.74	+
acin9	0.00 %	0.00 %	0.00 %	0.00 %	-	**0.84**	56.13	18.38	59.84	+

Tabla A.5: Resultados experimentales de ISA, PALS, GAG$_{50}$ y SAX para las instancias NF. Los mejores valores están remarcados en negro.

Instancias	*Mejor fitness*			*Fitness medio*			*Test*
	ISA	GAG$_{50}$	SAX	ISA	GAG$_{50}$	SAX	*KW*
x60189_4	**11478**	**11478**	**11478**	11329.17	**11478.00**	11409.57	-
x60189_5	14027	**17807**	14027	13864.83	**14903.58**	13968.30	+
x60189_6	**18301**	17999	**18301**	18138.70	17411.57	**18151.80**	+
x60189_7	21213	20884	**21268**	20888.73	20713.27	**21042.13**	-
m15421_5	**38674**	37681	38672	**38484.50**	37361.30	38360.20	+
m15421_6	**48048**	47137	**48048**	**47867.57**	46675.63	47802.37	+
m15421_7	**54948**	52734	54925	**54614.23**	52313.97	54571.33	+
j02459_7	**116167**	111716	114817	**114992.97**	110612.80	113771.73	+
38524243_4	**226580**	219645	223133	**225253.03**	218060.52	221445.77	+
38524243_7	**435653**	415036	419009	**433297.10**	415036.00	415339.73	+
acin1	**47027**	45499	46754	**46653.13**	45301.17	46452.77	+
acin2	144602	**153492**	144500	**144552.79**	143066.97	143945.20	-
acin3	**156862**	154597	155342	**156293.90**	153658.40	154556.60	+
acin5	**146409**	145299	145598	**146291.63**	145225.83	144947.60	+
acin7	**157661**	155885	156742	**157596.10**	155816.37	156679.90	+
acin9	**324839**	314299	313820	**322862.20**	311993.50	311782.60	+

Tabla A.6: Resultados experimentales de ISA, PALS, GAG$_{50}$ y SAX para las instancias NF (cont.). Los mejores valores están remarcados en negro.

Instancias	*% Contigs óptimos*			*Test*	*Tpo. medio de la mejor solución*			*Test*
	ISA	GAG$_{50}$	SAX	*KW*	ISA	GAG$_{50}$	SAX	*KW*
x60189_4	**100.00 %**	**100.00 %**	**100.00 %**	-	**2.48**	29.57	5.14	+
x60189_5	**100.00 %**	**100.00 %**	**100.00 %**	-	**2.95**	33.06	4.50	+
x60189_6	**100.00 %**	**100.00 %**	**100.00 %**	-	**3.00**	27.31	6.51	+
x60189_7	**100.00 %**	**100.00 %**	**100.00 %**	-	**3.09**	30.73	7.14	+
m15421_5	**100.00 %**	90.00 %	96.67 %	-	**4.26**	30.32	13.51	+
m15421_6	0.00 %	0.00 %	0.00 %	-	**4.94**	36.91	20.16	+
m15421_7	0.00 %	0.00 %	0.00 %	-	**5.02**	32.29	21.03	+
j02459_7	80.00 %	26.67 %	50.00 %	-	**10.20**	54.93	58.87	+
38524243_4	0.00 %	0.00 %	0.00 %	-	**13.94**	58.55	59.53	+
38524243_7	0.00 %	0.00 %	**100.00 %**	+	**26.98**	59.44	59.27	+
acin1	0.00 %	0.00 %	0.00 %	-	**7.37**	46.10	54.29	+
acin2	0.00 %	0.00 %	0.00 %	-	**13.46**	57.52	58.50	+
acin3	0.00 %	0.00 %	0.00 %	-	**17.77**	59.38	59.49	+
acin5	0.00 %	0.00 %	**100.00 %**	+	**24.34**	59.89	58.84	+
acin7	0.00 %	0.00 %	0.00 %	-	**44.61**	60.00	58.89	+
acin9	0.00 %	0.00 %	0.00 %	-	45.18	**24.62**	59.23	+

Tabla A.7: Resultados experimentales de ISA, PALS, GAG$_{50}$ y SAX para las instancias NS05. Los mejores valores están remarcados en negro.

Instancias	*Mejor fitness*				*Fitness medio*				*Test*
	ISA	PALS	GAG$_{50}$	SAX	ISA	PALS	GAG$_{50}$	SAX	*KW*
x60189_4	10633	**11375**	10549	10587	9472.27	**10643.80**	10367.63	10061.73	-
x60189_5	12550	**13807**	12593	13148	11979.87	9963.40	12348.33	**12597.87**	+
x60189_6	16403	**18142**	16909	16124	15752.13	**17133.59**	16338.83	14404.64	+
x60189_7	19752	**20671**	19624	20500	18862.93	**20037.37**	19231.97	17400.27	+
m15421_5	33246	**36911**	33233	33230	32714.50	**34691.20**	33081.33	29615.64	+
m15421_6	7853	**30391**	5135	6067	5071.67	**28919.78**	4418.30	4562.40	+
m15421_7	48220	**53737**	48439	48190	46921.93	**52414.93**	47377.97	42667.45	+
j02459_7	100507	**112467**	100474	100285	99028.63	**110608.10**	99658.20	99032.20	+
38524243_4	**366017**	211968	184253	186368	195474.42	**207218.57**	182335.70	183198.00	+
38524243_7	366017	**409032**	373017	368811	364493.00	336916.94	**373017.00**	365369.50	+
acin1	45280	**46834**	45341	45230	44878.07	**46540.27**	45111.97	44905.60	+
acin2	138927	142669	**142771**	139510	136665.63	139992.33	**140983.20**	137700.70	+
acin3	150949	**153773**	151658	150251	148488.83	**151609.67**	150730.60	148136.00	+
acin5	140943	**145561**	144193	138518	137781.58	**144155.43**	143165.90	136753.30	+
acin7	149908	**153858**	153191	148269	147416.88	152051.27	**152989.30**	146045.90	+
acin9	305646	**320709**	300838	297853	301399.00	**317316.43**	298477.23	293860.50	+

Tabla A.8: Resultados experimentales de ISA, PALS, GAG$_{50}$ y SAX para las instancias NS05 (cont.). Los mejores valores están remarcados en negro.

Instancias	*% Contigs óptimos*				*Test*	*Tpo. medio de la mejor solución*				*Test*
	ISA	PALS	GAG$_{50}$	SAX	*KW*	ISA	PALS	GAG$_{50}$	SAX	*KW*
x60189_4	**100.00 %**	**100.00 %**	**100.00 %**	**100.00 %**	-	**0.01**	3.96	0.05447403	2.66	+
x60189_5	**100.00 %**	**100.00 %**	**100.00 %**	**100.00 %**	-	**0.03**	2.61	0.13130853	7.28	+
x60189_6	96.67 %	**100.00 %**	**100.00 %**	**100.00 %**	-	**0.05**	6.64	0.24088103	3.37	+
x60189_7	**100.00 %**	**100.00 %**	**100.00 %**	**100.00 %**	-	**0.06**	17.80	1.2111515	4.48	+
m15421_5	53.33 %	**76.67 %**	26.67 %	40.00 %	+	**0.21**	20.70	1.27163523	7.09	+
m15421_6	0.00 %	0.00 %	0.00 %	0.00 %	-	**0.15**	59.70	0.78450893	9.50	+
m15421_7	0.00 %	0.00 %	0.00 %	0.00 %	-	**0.83**	6.68	4.43704533	9.45	+
j02459_7	40.00 %	**66.67 %**	23.33 %	30.00 %	-	**7.31**	40.70	40.46648	23.60	+
38524243_4	0.00 %	0.00 %	0.00 %	0.00 %	-	**6.95**	59.80	51.5088767	22.00	+
38524243_7	0.00 %	0.00 %	0.00 %	0.00 %	-	**30.36**	58.50	50.5138467	60.00	+
acin1	0.00 %	0.00 %	0.00 %	0.00 %	-	**5.04**	30.50	38.9446733	51.30	+
acin2	0.00 %	0.00 %	0.00 %	**66.67 %**	+	**18.16**	59.90	58.5664	22.00	+
acin3	0.00 %	0.00 %	0.00 %	0.00 %	-	**22.59**	59.90	59.56179	60.00	+
acin5	0.00 %	0.00 %	0.00 %	0.00 %	-	**22.90**	59.80	59.87411	60.00	+
acin7	0.00 %	0.00 %	0.00 %	0.00 %	-	**34.19**	59.80	59.9881033	60.00	+
acin9	0.00 %	0.00 %	0.00 %	0.00 %	-	42.99	59.70	**2.31949333**	60.00	+

Tabla A.9: Resultados experimentales de ISA, PALS, GAG$_{50}$ y SAX para las instancias NS10. Los mejores valores están remarcados en negro.

Instancias	*Mejor fitness*				*Fitness medio*				*Test*
	ISA	PALS	GAG$_{50}$	SAX	ISA	PALS	GAG$_{50}$	SAX	*KW*
x60189_4	11080	**11478**	11191	**11478**	10108.53	11379.33	10932.17	**11396.17**	-
x60189_5	12518	**13082**	**13082**	12519	12131.97	12442.07	**12551.07**	11753.03	-
x60189_6	15055	18261	15531	**18301**	14515.00	17871.27	15168.60	**18134.23**	+
x60189_7	19013	21104	19178	**21271**	18490.07	20918.23	18894.13	**21073.43**	+
m15421_5	**299141**	38635	34149	38726	33033.27	38335.93	33662.83	**38560.57**	+
m15421_6	5360	**48048**	3917	**48048**	3746.50	47887.60	3275.97	**47918.47**	+
m15421_7	47469	54675	48033	**55089**	45457.33	47114.07	47504.43	**54797.47**	+
j02459_7	97576	**115781**	98739	114672	96037.17	**114702.80**	97596.23	113857.00	+
38524243_4	177262	**225827**	177782	223484	173625.90	**224838.13**	175581.63	221177.03	+
38524243_7	368448	405718	369613	**415705**	362944.83	369540.33	365690.57	**414450.67**	+
acin1	**362216**	46576	44861	45028	**54925.97**	46123.43	44596.53	44396.27	+
acin2	136201	137788	**141006**	135410	128936.97	131298.03	**139692.60**	128015.23	+
acin3	143745	**152277**	150995	143879	138912.59	141360.23	**148470.70**	139467.20	+
acin5	136890	140332	**141641**	135021	133233.80	133060.23	**140841.74**	133102.38	+
acin7	147280	154396	**154524**	147914	142952.88	143702.47	**152982.13**	143382.80	+
acin9	299141	**316920**	298322	291003	294637.11	**313773.33**	295058.37	288036.50	+

Tabla A.10: Resultados experimentales de ISA, PALS, GAG$_{50}$ y SAX para las instancias NS10 (cont.). Los mejores valores están remarcados en negro.

Instancias	*% Contigs óptimos*				*Test*	*Tpo. medio de la mejor solución*				*Test*
	ISA	PALS	GAG$_{50}$	SAX	*KW*	ISA	PALS	GAG$_{50}$	SAX	*KW*
x60189_4	**100.00 %**	**100.00 %**	**100.00 %**	**100.00 %**	-	0.01	**0.00**	0.03	7.98	+
x60189_5	**100.00 %**	**100.00 %**	**100.00 %**	93.33 %	-	0.02	**0.00**	0.07	7.23	+
x60189_6	83.33 %	**100.00 %**	**100.00 %**	**100.00 %**	-	0.03	**0.00**	0.11	10.80	+
x60189_7	**100.00 %**	**100.00 %**	**100.00 %**	**100.00 %**	-	0.05	**0.00**	0.13	11.90	+
m15421_5	53.33 %	**100.00 %**	66.67 %	**100.00 %**	+	0.16	**0.03**	1.07	21.40	+
m15421_6	0.00 %	0.00 %	0.00 %	0.00 %	-	0.13	**0.07**	1.14	32.00	+
m15421_7	0.00 %	0.00 %	0.00 %	0.00 %	-	0.42	**0.09**	2.46	34.60	+
j02459_7	40.00 %	**93.00 %**	63.33 %	56.67 %	-	4.36	**0.70**	35.60	60.00	+
38524243_4	0.00 %	0.00 %	0.00 %	0.00 %	-	5.69	**1.24**	43.99	60.00	+
38524243_7	0.00 %	0.00 %	0.00 %	0.00 %	-	**23.15**	45.99	54.53	60.00	+
acin1	0.00 %	0.00 %	0.00 %	0.00 %	-	**4.86**	23.00	27.69	60.00	+
acin2	0.00 %	0.00 %	0.00 %	**100.00 %**	+	**14.19**	59.70	58.29	60.00	+
acin3	0.00 %	0.00 %	0.00 %	0.00 %	-	**17.32**	59.90	59.46	60.00	+
acin5	0.00 %	0.00 %	0.00 %	0.00 %	-	**25.49**	59.80	59.97	60.00	+
acin7	0.00 %	0.00 %	0.00 %	0.00 %	-	**37.61**	59.80	59.98	60.00	+
acin9	0.00 %	0.00 %	0.00 %	0.00 %	-	43.86	59.70	**5.55**	60.00	+

Tabla A.11: Resultados experimentales de ISA, PALS, GAG$_{50}$ y SAX para las instancias NS15. Los mejores valores están remarcados en negro.

Instancias	*Mejor fitness*				*Fitness medio*				*Test*
	ISA	PALS	GAG$_{50}$	SAX	ISA	PALS	GAG$_{50}$	SAX	*KW*
x60189_4	9759	**10794**	10549	10081	8921.03	10031.21	**10367.63**	9012.90	+
x60189_5	12187	13469	**12593**	12825	11855.37	**12779.00**	12348.33	12107.77	-
x60189_6	15443	17474	**16909**	15956	14522.03	16315.31	**16338.83**	14613.07	+
x60189_7	18818	20566	**19624**	19122	18043.27	19105.24	**19231.97**	18132.70	+
m15421_5	30874	35589	**33233**	31406	30121.90	**34162.07**	33081.33	30274.13	+
m15421_6	4385	**17873**	5135	3986	3211.27	**14513.90**	4418.30	3130.03	+
m15421_7	55081	50405	48439	**55087**	54741.83	49136.20	47377.97	**54809.83**	+
j02459_7	98587	**105891**	100474	97521	96575.77	**103520.37**	99658.20	96197.30	+
38524243_4	170948	**190651**	184253	170977	167083.87	**186952.33**	182335.70	167309.73	+
38524243_7	356565	364756	**373017**	352660	240001.49	365426.17	**373017.00**	351132.20	+
acin1	44074	**46142**	45341	44280	43757.17	**45622.90**	45111.97	43802.60	+
acin2	128185	136861	**142771**	119944	123987.17	130825.77	**140983.20**	115073.33	+
acin3	136075	149720	**151658**	133659	129929.87	146303.60	**150730.60**	128689.10	+
acin5	127984	138637	**144193**	127235	124208.00	136288.37	**143165.90**	123764.30	+
acin7	141228	**154037**	153191	139332	135993.84	151426.50	**152989.30**	134719.10	+
acin9	279013	**307211**	300838	273115	275856.75	**300430.60**	298477.23	267823.00	+

Tabla A.12: Resultados experimentales de ISA, PALS, GAG$_{50}$ y SAX para las instancias NS15 (cont.). Los mejores valores están remarcados en negro.

Instancias	*% Contigs óptimos*				*Test*	*Tpo. medio de la mejor solución*				*Test*
	ISA	PALS	GAG$_{50}$	SAX	*KW*	ISA	PALS	GAG$_{50}$	SAX	*KW*
x60189_4	**100.00 %**	96.55 %	**100.00 %**	**100.00 %**	-	**0.00**	6.66	0.02	6.65	+
x60189_5	**100.00 %**	**100.00 %**	**100.00 %**	**100.00 %**	-	**0.01**	3.54	0.05	8.07	+
x60189_6	**100.00 %**	**100.00 %**	**100.00 %**	**93.33 %**	-	**0.03**	23.60	0.14	8.27	+
x60189_7	**100.00 %**	**100.00 %**	**100.00 %**	96.67 %	-	**0.04**	21.10	0.16	8.37	+
m15421_5	86.67 %	80.00 %	26.67 %	**93.33 %**	+	**0.14**	33.00	1.09	13.40	+
m15421_6	0.00 %	0.00 %	0.00 %	0.00 %	-	**0.13**	59.80	0.99	18.40	+
m15421_7	0.00 %	0.00 %	0.00 %	0.00 %	-	**1.41**	59.90	3.88	19.00	+
j02459_7	30.00 %	**50.00 %**	23.33 %	43.33 %	-	**2.94**	60.00	36.37	48.50	+
38524243_4	0.00 %	0.00 %	0.00 %	0.00 %	-	**3.96**	60.00	44.19	60.00	+
38524243_7	0.00 %	0.00 %	0.00 %	0.00 %	-	**23.42**	60.00	46.35	60.00	+
acin1	0.00 %	0.00 %	0.00 %	0.00 %	-	**4.42**	43.10	32.64	37.80	+
acin2	0.00 %	0.00 %	0.00 %	0.00 %	-	**12.84**	59.90	58.96	60.00	+
acin3	0.00 %	0.00 %	0.00 %	0.00 %	-	**16.64**	59.90	59.57	60.00	+
acin5	0.00 %	0.00 %	0.00 %	0.00 %	-	**22.33**	59.90	59.89	60.00	+
acin7	0.00 %	0.00 %	0.00 %	0.00 %	-	**32.91**	59.80	59.99	60.00	+
acin9	0.00 %	0.00 %	0.00 %	0.00 %	-	**56.41**	59.80	2.47	60.00	+

Tabla A.13: Resultados experimentales de ISA, PALS, GAG$_{50}$ y SAX para las instancias NS20. Los mejores valores están remarcados en negro.

Instancias	*Mejor fitness*				*Fitness medio*				*Test*
	ISA	PALS	GAG$_{50}$	SAX	ISA	PALS	GAG$_{50}$	SAX	*KW*
x60189_4	8403	**10340**	9107	8864	7797.73	**9295.10**	8399.47	8048.50	+
x60189_5	11592	**12604**	11745	11982	11011.20	**11909.73**	11592.83	11228.87	-
x60189_6	14733	**16911**	14346	14717	13913.57	**15911.00**	13627.13	13517.50	+
x60189_7	18078	**20063**	17917	17917	17297.57	**19049.77**	17857.40	17340.70	+
m15421_5	31547	**34623**	31203	31328	30620.10	**33243.67**	30792.70	10205.50	+
m15421_6	2079	**11826**	2882	1905	1164.23	8**919.97**	1767.30	1469.20	+
m15421_7	43993	**48774**	43526	43992	42377.17	**46741.53**	42561.13	42456.50	+
j02459_7	91887	**101318**	92440	90622	90278.43	**99073.40**	91670.03	90032.50	+
38524243_4	168183	**186732**	171507	165681	163667.17	**178395.83**	169884.20	163063.60	+
38524243_7	340420	340630	**345192**	341345	340118.50	340225.37	**341697.50**	337506.60	-
acin1	43860	**45888**	44555	43549	43222.17	**45224.90**	44185.33	43380.10	+
acin2	122712	**134787**	134112	122210	117775.20	130924.30	**132281.33**	118524.70	+
acin3	128183	**139220**	137894	126958	122978.50	**135978.67**	135799.20	122385.70	+
acin5	126440	**139680**	137936	122046	114951.20	**136064.37**	134642.67	116677.00	+
acin7	**257665**	147599	146656	132739	136503.00	**145081.90**	141148.60	127536.40	+
acin9	261457	**297118**	264019	253192	254649.97	**291957.10**	260100.17	248017.20	+

Tabla A.14: Resultados experimentales de ISA, PALS, GAG$_{50}$ y SAX para las instancias NS20 (cont.). Los mejores valores están remarcados en negro.

Instancias	*% Contigs óptimos*				*Test*	*Tpo. medio de la mejor solución*				*Test*
	ISA	PALS	GAG$_{50}$	SAX	*KW*	ISA	PALS	GAG$_{50}$	SAX	*KW*
x60189_4	**100.00 %**	**100.00 %**	**100.00 %**	**100.00 %**	-	**0.01**	6.66	0.01	2.52	+
x60189_5	**100.00 %**	93.33 %	**100.00 %**	90.00 %	-	**0.02**	3.54	0.05	7.64	+
x60189_6	**100.00 %**	**100.00 %**	**100.00 %**	**100.00 %**	-	**0.01**	23.60	0.08	3.32	+
x60189_7	96.67 %	93.33 %	**100.00 %**	**100.00 %**	-	**0.02**	21.10	0.10	9.48	+
m15421_5	76.67 %	66.67 %	56.67 %	**80.00 %**	-	**0.12**	33.00	0.60	4.89	+
m15421_6	0.00 %	0.00 %	0.00 %	0.00 %	-	**0.02**	59.80	0.43	6.75	+
m15421_7	0.00 %	0.00 %	0.00 %	0.00 %	-	**0.56**	59.90	2.97	6.97	+
j02459_7	36.67 %	30.00 %	20.00 %	**40.00 %**	-	**2.62**	60.00	29.23	18.20	+
38524243_4	0.00 %	0.00 %	0.00 %	0.00 %	-	**3.65**	60.00	33.76	22.00	+
38524243_7	0.00 %	0.00 %	0.00 %	0.00 %	-	**21.89**	43.10	50.21	60.00	+
acin1	0.00 %	0.00 %	0.00 %	0.00 %	-	**4.72**	59.90	30.85	38.80	+
acin2	0.00 %	0.00 %	0.00 %	0.00 %	-	**12.48**	59.90	58.58	22.00	+
acin3	0.00 %	0.00 %	0.00 %	0.00 %	-	**16.46**	59.90	59.38	60.00	+
acin5	0.00 %	0.00 %	0.00 %	0.00 %	-	**22.24**	59.80	59.87	60.00	+
acin7	0.00 %	0.00 %	0.00 %	0.00 %	-	**32.81**	59.80	52.09	60.00	+
acin9	0.00 %	0.00 %	0.00 %	0.00 %	-	51.03	**6.66**	12.07	60.00	+

Tabla A.15: Resultados experimentales de ISA, PALS, GAG$_{50}$ y SAX para las instancias NS25. Los mejores valores están remarcados en negro.

Instancias	*Mejor fitness*				*Fitness medio*				*Test*
	ISA	PALS	GAG$_{50}$	SAX	ISA	PALS	GAG$_{50}$	SAX	*KW*
x60189_4	8943	**10529**	8601	8912	8263.03	**9097.80**	8560.67	8121.43	+
x60189_5	12018	**13034**	12044	12021	11475.67	**12317.60**	11859.87	11555.23	+
x60189_6	13903	**15729**	13462	13911	12998.63	**14715.30**	13241.77	13104.43	+
x60189_7	18416	**19472**	18029	18253	17362.97	**18671.97**	18021.53	17349.23	+
m15421_5	29839	**33498**	30007	29840	28572.63	**32393.70**	29592.17	28841.40	+
m15421_6	3200	**8300**	2475	2745	2203.13	**7037.47**	2000.23	2114.47	+
m15421_7	42329	**47424**	42921	42536	41296.20	**45044.93**	42250.67	41074.70	+
j02459_7	92871	**100837**	93380	92155	90389.30	**98131.23**	91574.23	90108.83	+
38524243_4	154778	**166624**	157332	157237	151363.40	**163173.83**	154903.77	152241.33	+
38524243_7	341042	342320	**344316**	338385	341042.00	341247.33	**342047.33**	337426.00	+
acin1	43165	**45270**	43824	43129	42773.80	**44660.57**	43453.23	42721.87	+
acin2	118031	128722	**132350**	117995	118031.00	126021.90	**130770.20**	113635.37	+
acin3	122772	**140973**	139644	121714	117234.00	**136981.06**	136304.87	115794.90	+
acin5	121374	**135405**	131770	118763	113116.10	**131583.90**	128599.93	113525.80	+
acin7	124850	**143208**	140331	118909	115575.27	**141149.77**	135299.90	114264.20	+
acin9	263759	**297566**	267737	258125	251964.28	**294264.87**	263170.67	255307.00	+

Tabla A.16: Resultados experimentales de ISA, PALS, GAG$_{50}$ y SAX para las instancias NS25 (cont.). Los mejores valores están remarcados en negro.

Instancias	*% Contigs óptimos*				*Test*	*Tpo. medio de la mejor solución*				*Test*
	ISA	PALS	GAG$_{50}$	SAX	*KW*	ISA	PALS	GAG$_{50}$	SAX	*KW*
x60189_4	**100.00 %**	**100.00 %**	**100.00 %**	**100.00 %**	-	**0.00**	3.54	0.02	6.63	+
x60189_5	**100.00 %**	**100.00 %**	**100.00 %**	**100.00 %**	-	**0.01**	23.60	0.01	7.15	+
x60189_6	**100.00 %**	**100.00 %**	**100.00 %**	**100.00 %**	-	**0.02**	21.10	**0.02**	8.23	+
x60189_7	**100.00 %**	**100.00 %**	**100.00 %**	**100.00 %**	-	**0.03**	33.00	0.09	8.36	+
m15421_5	60.00 %	**80.00 %**	36.67 %	60.00 %	-	**0.10**	59.80	0.58	13.30	+
m15421_6	0.00 %	0.00 %	0.00 %	0.00 %	-	**0.08**	59.90	0.90	18.40	+
m15421_7	0.00 %	0.00 %	0.00 %	0.00 %	-	**0.36**	60.00	2.25	19.00	+
j02459_7	30.00 %	**33.33 %**	3.33 %	30.00 %	-	**2.51**	44.15	26.05	48.30	+
38524243_4	0.00 %	0.00 %	0.00 %	0.00 %	-	**4.18**	59.50	42.40	60.00	+
38524243_7	0.00 %	0.00 %	0.00 %	0.00 %	-	**21.63**	59.80	45.43	60.00	+
acin1	0.00 %	0.00 %	0.00 %	0.00 %	-	**6.45**	43.10	27.00	37.90	+
acin2	0.00 %	0.00 %	0.00 %	0.00 %	-	**17.58**	59.90	59.03	60.00	+
acin3	0.00 %	0.00 %	0.00 %	0.00 %	-	**22.57**	59.90	59.66	60.00	+
acin5	0.00 %	0.00 %	0.00 %	0.00 %	-	**27.48**	59.90	59.89	60.00	+
acin7	0.00 %	0.00 %	0.00 %	0.00 %	-	**37.90**	59.80	59.99	60.00	+
acin9	0.00 %	0.00 %	0.00 %	0.00 %	-	42.68	59.80	**4.40**	60.00	+

Apéndice B

Resultados experimentales de Pan-H-PALS y PH-PALS para las instancias con ruido

En este apéndice se muestran las tablas de resultados experimentales obtenidos por las metaheurísticas propuestas en el capítulo 10 (PanH-PALS, PH-PALS), para resolver instancias con distintas fuentes de ruido. Los errores o ruido pueden ocurrir durante: la secuenciación generando errores en las bases nucleótidas (NB), la fase de superposición creando errores en la matriz de solapamiento (NS) y en el cálculo del *fitness* obteniendo valores de *fitness* ruidosos (NF). Dado que PH-PALS ha sido ejecutado sobre 3, 6, 9 y 12 procesadores, los resultados provenientes de cada caso son identificados como PH-PALS$_{3p}$, PH-PALS$_{6p}$, PH-PALS$_{9p}$ y PH-PALS$_{12p}$, respectivamente.

En las tablas B.1, B.2, B.3 y B.4 se muestran los resultados experimentales obtenidos para las instancias sin ruido, en las tablas B.5, B.6, B.7 y B.8 los conseguidos para las instancias NB. En tanto que, en las tablas B.9, B.10, B.11 y B.12 se presentan los resultados logrados para las instancias NF. Por último, en las tablas B.13, B.14, B.15 y B.16 se muestran los resultados correspondientes a las instancias NS10

Tabla B.1: Resultados experimentales de PanH-PALS, PH-PALS$_{3p}$, PH-PALS$_{6p}$, PH-PALS$_{9p}$ y PH-PALS$_{12p}$ para las instancias sin ruido. Los mejores valores están remarcados en negro.

Instancias	*Mejor fitness*				
	PanH-PALS	PH-PALS$_{3p}$	PH-PALS$_{6p}$	PH-PALS$_{9p}$	PH-PALS$_{12p}$
x60189_4	11085	11204	**11317**	**11317**	**11317**
x60189_5	12984	12627	12851	**13222**	13186
x60189_6	17023	17046	16960	17461	**17668**
x60189_7	20514	20890	20740	20756	**20997**
m15421_5	36897	36628	37838	**38589**	38022
m15421_6	**51487**	45481	46109	46346	46415
m15421_7	51654	51771	**52074**	52006	51869
j02459_7	107995	108469	108209	108854	**109013**
38524243_4	211314	211576	212661	209382	**214184**
38524243_7	411429	409655	409967	**412572**	411870
acin1	53159	55537	55785	55782	**56798**
acin2	247907	270546	267304	**282990**	281880
acin3	291953	314692	316102	318970	**333929**
acin5	281070	337666	337359	**349003**	348639
acin7	313496	375962	377868	384456	**393863**
acin9	524229	**555023**	548892	550535	359433

Tabla B.2: Resultados experimentales de PanH-PALS, PH-PALS$_{3p}$, PH-PALS$_{6p}$, PH-PALS$_{9p}$ y PH-PALS$_{12p}$ para las instancias sin ruido (cont.). Los mejores valores están remarcados en negro.

Instancias	*Fitness medio*					*Test*
	PanH-PALS	PH-PALS$_{3p}$	PH-PALS$_{6p}$	PH-PALS$_{9p}$	PH-PALS$_{12p}$	*KW*
x60189_4	10840.00	10813.00	10881.33	11069.40	**11090.37**	-
x60189_5	**12939.00**	12580.67	12631.33	12739.60	12733.80	-
x60189_6	16695.33	16866.00	16801.00	**17183.20**	17069.20	-
x60189_7	20352.00	20357.00	**20477.33**	20168.40	20324.40	-
m15421_5	36569.67	36348.00	36848.00	**37255.20**	36856.53	-
m15421_6	**50849.00**	45468.33	45813.33	45915.60	45774.13	+
m15421_7	51389.67	51163.00	**51805.33**	51209.60	51340.27	-
j02459_7	107865.33	**108209.00**	107850.67	108052.00	108046.60	-
38524243_4	210529.00	210932.33	210584.67	209382.00	**211176.20**	+
38524243_7	410969.00	409292.50	409404.67	**412047.30**	409965.43	+
acin1	52117.33	54958.33	55214.67	55098.60	**55568.67**	+
acin2	234042.00	268542.00	264801.33	270549.80	**271859.53**	+
acin3	284182.00	309079.00	312566.67	311395.60	**316028.40**	+
acin5	277234.33	330823.67	330979.00	**337227.40**	336357.87	+
acin7	312218.33	372500.00	375026.33	**377939.80**	375637.67	+
acin9	506756.33	536976.67	**544239.33**	542302.00	342851.14	+

Tabla B.3: Resultados experimentales de PanH-PALS, PH-PALS$_{3p}$, PH-PALS$_{6p}$, PH-PALS$_{9p}$ y PH-PALS$_{12p}$ para las instancias sin ruido (cont.). Los mejores valores están remarcados en negro.

Instancias	*% Contigs óptimos*					*Test*
	PanH-PALS	PH-PALS$_{3p}$	PH-PALS$_{6p}$	PH-PALS$_{9p}$	PH-PALS$_{12p}$	*KW*
x60189_4	**100.00 %**	66.67 %	**100.00 %**	**100.00 %**	**100.00 %**	+
x60189_5	66.67 %	**100.00 %**	**100.00 %**	**100.00 %**	**100.00 %**	+
x60189_6	**100.00 %**	**100.00 %**	**100.00 %**	93.33 %	93.33 %	-
x60189_7	**100.00 %**	**100.00 %**	**100.00 %**	**100.00 %**	**100.00 %**	-
m15421_5	**100.00 %**	0.00 %	33.33 %	6.67 %	6.67 %	+
m15421_6	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	-
m15421_7	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	-
j02459_7	0.00 %	0.00 %	0.00 %	**33.33 %**	**33.33 %**	+
38524243_4	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	-
38524243_7	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	-
acin1	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	-
acin2	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	-
acin3	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	-
acin5	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	-
acin7	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	-
acin9	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	-

Tabla B.4: Resultados experimentales de PanH-PALS, PH-PALS$_{3p}$, PH-PALS$_{6p}$, PH-PALS$_{9p}$ y PH-PALS$_{12p}$ para las instancias sin ruido (cont.). Los mejores valores están remarcados en negro.

Instancias	*Tpo. total medio*					*Test*
	PanH-PALS	PH-PALS$_{3p}$	PH-PALS$_{6p}$	PH-PALS$_{9p}$	PH-PALS$_{12p}$	*KW*
x60189_4	6.30	0.32	0.43	**0.31**	0.33	+
x60189_5	6.30	0.42	0.46	**0.38**	0.39	+
x60189_6	6.30	0.53	0.49	**0.40**	0.49	+
x60189_7	6.30	0.51	0.59	**0.40**	0.53	+
m15421_5	6.30	0.71	0.86	**0.55**	0.67	+
m15421_6	6.30	0.87	1.06	**0.70**	0.78	+
m15421_7	6.30	0.82	0.99	**0.68**	0.79	+
j02459_7	6.30	1.79	1.99	1.94	**1.76**	+
38524243_4	6.20	2.11	2.33	4.47	**1.85**	+
38524243_7	6.30	5.13	5.34	**4.64**	7.82	-
acin1	6.03	1.09	1.16	**0.84**	0.86	+
acin2	6.13	1.85	1.87	**1.48**	1.61	+
acin3	6.25	2.32	2.77	**2.17**	2.28	+
acin5	6.28	3.29	3.64	3.22	**3.08**	+
acin7	6.18	5.55	5.78	5.02	**4.99**	-
acin9	6.30	6.93	7.48	6.76	**3.69**	+

Tabla B.5: Resultados experimentales de PanH-PALS, PH-PALS$_{3p}$, PH-PALS$_{6p}$, PH-PALS$_{9p}$ y PH-PALS$_{12p}$ para las instancias NB. Los mejores valores están remarcados en negro.

Instancias	*Mejor fitness*				
	PanH-PALS	PH-PALS$_{3p}$	PH-PALS$_{6p}$	PH-PALS$_{9p}$	PH-PALS$_{12p}$
x60189_4	10970	11204	**11317**	11081	**11317**
x60189_5	12693	13098	13175	12693	**13388**
x60189_6	16910	17166	17514	16874	**17659**
x60189_7	20682	20352	20751	**20888**	20869
m15421_5	36770	37629	37067	**37793**	37755
m15421_6	45485	45783	45745	45923	**46030**
m15421_7	51402	51727	52288	52366	**52606**
j02459_7	73806	**77694**	76072	76598	76362
38524243_4	133240	137101	136110	135388	**137668**
38524243_7	290247	294856	293991	294572	**297500**
acin1	53970	55168	56555	55743	**57752**
acin2	119054	122821	124608	134216	**137762**
acin3	126901	143158	130856	142382	**146789**
acin5	154398	141097	142702	155439	**167458**
acin7	177486	173476	188277	183891	**190759**
acin9	286439	289657	286971	289199	**297583**

Tabla B.6: Resultados experimentales de PanH-PALS, PH-PALS$_{3p}$, PH-PALS$_{6p}$, PH-PALS$_{9p}$ y PH-PALS$_{12p}$ para las instancias NB (cont.). Los mejores valores están remarcados en negro.

Instancias	*Fitness medio*					*Test*
	PanH-PALS	PH-PALS$_{3p}$	PH-PALS$_{6p}$	PH-PALS$_{9p}$	PH-PALS$_{12p}$	*KW*
x60189_4	10719.67	10751.00	**11279.33**	10960.67	11012.73	-
x60189_5	12625.33	**12897.33**	12860.33	12553.00	12877.73	-
x60189_6	16795.33	16987.33	17035.67	16754.33	**17137.93**	-
x60189_7	19981.33	20223.00	20036.00	**20654.00**	20207.07	-
m15421_5	36305.33	37085.33	36583.67	**37311.20**	36769.00	+
m15421_6	45114.50	45191.67	**45588.50**	45380.67	45576.92	-
m15421_7	51164.33	51450.33	**51945.33**	51926.00	51844.00	-
j02459_7	72968.00	**76350.00**	75334.00	75884.60	75746.20	+
38524243_4	130976.67	134966.00	**135278.67**	134798.33	134104.20	+
38524243_7	288997.67	292462.00	293041.33	**293233.00**	292867.67	+
acin1	53251.33	54693.00	55180.67	55533.00	**55558.50**	+
acin2	112397.00	119527.33	122606.67	122131.33	**125748.00**	+
acin3	121494.67	133241.33	125588.33	**135976.67**	133918.40	+
acin5	147029.00	139574.33	137414.00	**153289.00**	151992.20	+
acin7	167657.67	170150.00	**180266.00**	177549.33	170282.53	+
acin9	286439.00	285014.33	281649.67	**287485.67**	283861.86	+

Tabla B.7: Resultados experimentales de PanH-PALS, PH-PALS$_{3p}$, PH-PALS$_{6p}$, PH-PALS$_{9p}$ y PH-PALS$_{12p}$ para las instancias NB (cont.). Los mejores valores están remarcados en negro.

Instancias	*% Contigs óptimos*					*Test*
	PanH-PALS	PH-PALS$_{3p}$	PH-PALS$_{6p}$	PH-PALS$_{9p}$	PH-PALS$_{12p}$	*KW*
x60189_4	**100.00 %**	**100.00 %**	**100.00 %**	**100.00 %**	**100.00 %**	-
x60189_5	66.67 %	**100.00 %**	**100.00 %**	**100.00 %**	**100.00 %**	+
x60189_6	**100.00 %**	**100.00 %**	**100.00 %**	**100.00 %**	**100.00 %**	-
x60189_7	**100.00 %**	**100.00 %**	**100.00 %**	**100.00 %**	**100.00 %**	-
m15421_5	33.33 %	33.33 %	0.00 %	**46.67 %**	**46.67** %	+
m15421_6	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	-
m15421_7	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	-
j02459_7	**33.33 %**	0.00 %	0.00 %	0.00 %	0.00 %	+
38524243_4	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	-
38524243_7	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	-
acin1	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	-
acin2	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	-
acin3	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	-
acin5	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	-
acin7	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	-
acin9	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	-

Tabla B.8: Resultados experimentales de PanH-PALS, PH-PALS$_{3p}$, PH-PALS$_{6p}$, PH-PALS$_{9p}$ y PH-PALS$_{12p}$ para las instancias NB (cont.). Los mejores valores están remarcados en negro.

Instancias	*Tpo. total medio*					*Test*
	PanH-PALS	PH-PALS$_{3p}$	PH-PALS$_{6p}$	PH-PALS$_{9p}$	PH-PALS$_{12p}$	*KW*
x60189_4	6.30	0.39	**0.35**	0.40	0.36	+
x60189_5	6.30	0.38	0.43	**0.35**	0.44	+
x60189_6	6.30	0.57	0.51	**0.44**	0.62	+
x60189_7	6.30	0.54	0.52	**0.49**	0.60	+
m15421_5	6.30	0.76	0.74	**0.62**	0.82	+
m15421_6	6.30	0.84	0.90	**0.69**	1.12	+
m15421_7	6.30	0.95	0.96	**0.68**	1.05	+
j02459_7	6.15	1.75	1.87	1.45	**1.42**	+
38524243_4	6.30	2.24	2.24	**1.83**	2.84	+
38524243_7	6.30	4.75	4.89	4.29	**4.22**	+
acin1	6.30	1.13	1.15	0.90	**0.84**	+
acin2	6.30	2.11	2.16	1.64	**1.63**	+
acin3	6.30	2.87	2.97	2.30	**2.28**	+
acin5	6.30	3.79	3.68	3.41	**3.37**	+
acin7	6.30	5.60	5.96	5.80	**5.17**	-
acin9	**6.30**	8.32	8.54	8.52	7.85	+

Tabla B.9: Resultados experimentales de PanH-PALS, PH-PALS$_{3p}$, PH-PALS$_{6p}$, PH-PALS$_{9p}$ y PH-PALS$_{12p}$ para las instancias NF. Los mejores valores están remarcados en negro.

Instancias	*Mejor fitness*				
	PanH-PALS	PH-PALS$_{3p}$	PH-PALS$_{6p}$	PH-PALS$_{9p}$	PH-PALS$_{12p}$
x60189_4	**11317**	11204	11204	10859	**11317**
x60189_5	13282	12632	**13315**	12921	13022
x60189_6	16977	**17271**	17177	16616	16932
x60189_7	19882	20219	19782	19744	**20399**
m15421_5	36272	36648	36872	37235	**37605**
m15421_6	45371	45795	45818	46208	**46239**
m15421_7	50796	51922	51599	51608	**52286**
j02459_7	107930	108560	108271	**108854**	108304
38524243_4	210173	211553	210775	209382	**213474**
38524243_7	411429	409655	**409967**	412572	411870
acin1	54221	**56795**	56028	56087	56628
acin2	243598	273882	264827	272751	**279679**
acin3	274515	315832	308059	315288	**323610**
acin5	271366	345749	334914	340831	**348633**
acin7	317565	376815	382931	378154	**393293**
acin9	512869	540374	**549074**	547789	359433

Tabla B.10: Resultados experimentales de PanH-PALS, PH-PALS$_{3p}$, PH-PALS$_{6p}$, PH-PALS$_{9p}$ y PH-PALS$_{12p}$ para las instancias NF (cont.). Los mejores valores están remarcados en negro.

Instancias	*Fitness medio*					*Test*
	PanH-PALS	PH-PALS$_{3p}$	PH-PALS$_{6p}$	PH-PALS$_{9p}$	PH-PALS$_{12p}$	*KW*
x60189_4	10701.00	**10930.33**	10880.00	10526.00	10559.62	-
x60189_5	**12799.67**	12614.67	12638.00	12401.00	12405.07	-
x60189_6	16876.67	**16918.00**	16556.67	16566.33	16401.60	-
x60189_7	**19666.33**	19584.00	19328.67	19381.67	19518.13	-
m15421_5	36037.67	36192.33	36479.67	36638.67	**36669.80**	-
m15421_6	45074.67	45443.67	**45185.33**	45630.67	45311.47	-
m15421_7	50525.00	**51770.67**	50458.33	51122.33	51082.00	-
j02459_7	107726.00	107535.33	107695.00	**108052.00**	107549.93	-
38524243_4	209680.67	210540.00	210041.67	209382.00	**210732.47**	-
38524243_7	410969.00	409292.50	409404.67	**412047.33**	409965.43	+
acin1	52807.33	**55625.67**	55172.00	55331.33	54967.87	+
acin2	234461.00	264286.67	260753.33	**270217.67**	268769.93	+
acin3	264926.00	**310873.00**	301961.67	309136.67	309295.80	+
acin5	266753.00	**339206.33**	328558.67	335509.33	333750.47	+
acin7	304300.33	367377.67	373720.33	**374037.67**	372482.20	+
acin9	505198.67	540059.33	**542810.00**	542514.67	342851.14	+

Tabla B.11: Resultados experimentales de PanH-PALS, PH-PALS$_{3p}$, PH-PALS$_{6p}$, PH-PALS$_{9p}$ y PH-PALS$_{12p}$ para las instancias NF (cont.). Los mejores valores están remarcados en negro.

Instancias	*% Contigs óptimos*					*Test*
	PanH-PALS	PH-PALS$_{3p}$	PH-PALS$_{6p}$	PH-PALS$_{9p}$	PH-PALS$_{12p}$	*KW*
x60189_4	66.67 %	**100.00 %**	**100.00 %**	66.67 %	62.50 %	+
x60189_5	**100.00 %**	**100.00 %**	**100.00 %**	**100.00 %**	86.67 %	-
x60189_6	**100.00 %**	**100.00 %**	33.33 %	**100.00 %**	66.67 %	+
x60189_7	66.67 %	**100.00 %**	**100.00 %**	**100.00 %**	80.00 %	-
m15421_5	**33.33 %**	0.00 %	0.00 %	0.00 %	0.00 %	+
m15421_6	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	-
m15421_7	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	-
j02459_7	0.00 %	**33.33 %**	0.00 %	0.00 %	6.67 %	+
38524243_4	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	-
38524243_7	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	-
acin1	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	-
acin2	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	-
acin3	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	-
acin5	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	-
acin7	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	-
acin9	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	-

Tabla B.12: Resultados experimentales de PanH-PALS, PH-PALS$_{3p}$, PH-PALS$_{6p}$, PH-PALS$_{9p}$ y PH-PALS$_{12p}$ para las instancias NF (cont.). Los mejores valores están remarcados en negro.

Instancias	*Tpo. total medio*					*Test*
	PanH-PALS	PH-PALS$_{3p}$	PH-PALS$_{6p}$	PH-PALS$_{9p}$	PH-PALS$_{12p}$	*KW*
x60189_4	6.30	**0.46**	0.54	0.58	0.50	+
x60189_5	6.30	0.53	0.55	0.69	**0.53**	+
x60189_6	6.30	**0.59**	0.66	0.80	0.70	+
x60189_7	6.30	0.64	**0.63**	0.92	0.67	+
m15421_5	6.30	**0.81**	0.88	1.11	0.99	+
m15421_6	6.30	**1.05**	1.15	1.45	1.18	+
m15421_7	6.30	**0.94**	1.19	1.50	1.26	+
j02459_7	6.30	1.96	2.13	**1.94**	2.95	+
38524243_4	6.30	**2.51**	2.61	4.47	3.51	+
38524243_7	6.30	5.13	5.34	**4.64**	7.82	+
acin1	6.15	**1.10**	1.28	1.26	1.46	+
acin2	6.30	**1.90**	2.01	1.93	2.35	+
acin3	6.20	**2.74**	2.89	2.76	3.39	+
acin5	6.18	**3.33**	3.89	3.56	4.98	+
acin7	6.30	**5.66**	5.78	6.54	7.12	+
acin9	6.30	7.35	7.42	9.60	**3.69**	+

Tabla B.13: Resultados experimentales de PanH-PALS, PH-PALS$_{3p}$, PH-PALS$_{6p}$, PH-PALS$_{9p}$ y PH-PALS$_{12p}$ para las instancias NS10. Los mejores valores están remarcados en negro.

Instancias	*Mejor fitness*				
	PanH-PALS	PH-PALS$_{3p}$	PH-PALS$_{6p}$	PH-PALS$_{9p}$	PH-PALS$_{12p}$
x60189_4	11233	**11317**	11204	11204	**11317**
x60189_5	12193	11390	12600	**12819**	12774
x60189_6	16719	17125	**17549**	17541	17542
x60189_7	20437	20577	20895	19928	**20983**
m15421_5	36536	37017	37505	37223	**37890**
m15421_6	45852	45417	45947	46207	**46281**
m15421_7	43658	44505	44527	44342	**45230**
j02459_7	107781	108357	108400	108562	**108851**
38524243_4	211379	211982	211687	212320	**214461**
38524243_7	411711	412403	412494	412177	**413374**
acin1	53642	54301	55498	55447	**55976**
acin2	242053	267577	**279059**	262577	269025
acin3	275930	312882	316863	309865	**325517**
acin5	288784	326475	326596	332020	**341990**
acin7	303640	360397	356833	362294	**368285**
acin9	484398	533305	524161	529864	**548375**

Tabla B.14: Resultados experimentales de PanH-PALS, PH-PALS$_{3p}$, PH-PALS$_{6p}$, PH-PALS$_{9p}$ y PH-PALS$_{12p}$ para las instancias NS10 (cont.). Los mejores valores están remarcados en negro.

Instancias	*Fitness medio*					*Test*
	PanH-PALS	PH-PALS$_{3p}$	PH-PALS$_{6p}$	PH-PALS$_{9p}$	PH-PALS$_{12p}$	*KW*
x60189_4	11066.33	**11135.00**	10782.00	10836.40	10910.33	-
x60189_5	11513.67	11348.00	11525.80	**11728.40**	11428.86	-
x60189_6	16695.00	16912.67	16886.20	**16961.80**	16897.77	-
x60189_7	20093.33	19725.00	**20606.40**	19535.40	20317.77	-
m15421_5	36158.33	**36923.67**	36549.80	36904.80	36577.67	-
m15421_6	45584.33	45215.00	45538.80	45688.40	**45805.50**	-
m15421_7	43444.67	43988.33	43805.80	43650.40	**44073.07**	-
j02459_7	107594.00	108212.67	107894.67	**108334.20**	108020.87	-
38524243_4	211039.00	210878.67	210601.67	211371.20	**211653.15**	-
38524243_7	411711.00	412305.00	**412494.00**	411191.30	412100.92	-
acin1	52700.33	54049.67	53896.67	53842.00	**54327.87**	+
acin2	236865.00	261868.00	**269769.67**	260167.60	258418.87	+
acin3	270336.33	**309335.33**	304573.00	305585.00	306332.88	+
acin5	273786.00	317386.33	322198.00	321218.00	**324903.80**	+
acin7	299185.33	351793.67	356071.67	356943.60	**357842.27**	+
acin9	482006.00	522687.00	515652.67	520679.20	**527784.93**	+

Tabla B.15: Resultados experimentales de PanH-PALS, PH-PALS$_{3p}$, PH-PALS$_{6p}$, PH-PALS$_{9p}$ y PH-PALS$_{12p}$ para las instancias NS10 (cont.). Los mejores valores están remarcados en negro.

Instancias	*% Contigs óptimos*					*Test*
	PanH-PALS	PH-PALS$_{3p}$	PH-PALS$_{6p}$	PH-PALS$_{9p}$	PH-PALS$_{12p}$	*KW*
x60189_4	**100.00 %**	**100.00 %**	**100.00 %**	**100.00 %**	**100.00 %**	-
x60189_5	**100.00 %**	**100.00 %**	80.00 %	93.10 %	93.10 %	-
x60189_6	66.67 %	**100.00 %**	**100.00 %**	**100.00 %**	**100.00 %**	+
x60189_7	**100.00 %**	**100.00 %**	**100.00 %**	**100.00 %**	**100.00 %**	-
m15421_5	0.00 %	33.33 %	**40.00 %**	40.00 %	40.00 %	+
m15421_6	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	-
m15421_7	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	-
j02459_7	0.00 %	0.00 %	**33.33 %**	13.33 %	13.33 %	+
38524243_4	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	-
38524243_7	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	-
acin1	0.00 %	0.00 %	0.00 %	**6.67 %**	**6.67 %**	+
acin2	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	-
acin3	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	-
acin5	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	-
acin7	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	-
acin9	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	-

Tabla B.16: Resultados experimentales de PanH-PALS, PH-PALS$_{3p}$, PH-PALS$_{6p}$, PH-PALS$_{9p}$ y PH-PALS$_{12p}$ para las instancias NS10 (cont.). Los mejores valores están remarcados en negro.

Instancias	*Tpo. total medio*					*Test*
	PanH-PALS	PH-PALS$_{3p}$	PH-PALS$_{6p}$	PH-PALS$_{9p}$	PH-PALS$_{12p}$	*KW*
x60189_4	6.23	0.35	0.42	0.35	**0.32**	+
x60189_5	6.30	0.37	0.40	0.37	**0.36**	+
x60189_6	6.30	0.48	0.54	0.43	**0.41**	+
x60189_7	6.30	0.47	0.56	**0.42**	0.50	+
m15421_5	6.30	0.75	0.79	**0.55**	0.58	+
m15421_6	6.23	0.85	**1.02**	0.67	0.74	+
m15421_7	6.30	0.99	0.94	**0.67**	0.97	+
j02459_7	6.30	1.87	1.88	**1.52**	1.56	+
38524243_4	6.20	2.18	2.23	**1.81**	2.01	+
38524243_7	6.30	5.11	6.12	5.15	**4.62**	+
acin1	6.18	1.09	1.21	**0.92**	0.93	+
acin2	6.08	1.90	2.05	**1.52**	1.61	+
acin3	6.30	2.39	2.92	**2.21**	3.31	+
acin5	6.30	**3.21**	3.53	3.22	4.81	+
acin7	6.15	5.08	5.63	**5.03**	6.89	+
acin9	**6.30**	7.06	7.49	6.91	8.95	+

Apéndice C

En este apéndice se muestra un resumen de los resultados obtenidos por las distintas alternativas algorítmicas a partir de las cuales se ideó PH-PALS, las cuales son explicadas en la introducción del capítulo 10. Dado que estos algoritmos forman parte del camino de búsqueda de un ensamblador que compartiese las ventajas de PALS pero mejorase sustancialmente los resultados, se los ha ejecutado sólo sobre un conjunto de instancias. Las instancias elegidas para estas pruebas han sido las del grupo *acin* por ser las más complejas. De esta forma, se intenta evitar aquellas técnicas que sólo encontrarían buenos resultados para las instancias de bajo o mediana complejidad.

Con la letra a se identifica a la versión modificada de PALS que preserva y aplica una lista de los mejores movimientos que realizar por cada fragmento i, mientras que con b se representa al algoritmo que preserva y aplica el mejor movimiento. Con el carácter c se distingue al algoritmo basado en PALS que acepta algún movimiento aunque no resultase el mejor posible bajo cierta probabilidad, con d se identifica la versión que prioriza Δ_f sobre Δ_c. Las letras d y e representan las versiones paralelas heterogéneas asíncrona y síncrona, respectivamente. En la primera tabla se muestra el fitness promedio logrado por cada variante algorítmica, en la segunda se presenta el número promedio de contigs alcanzado por cada una y en la última se listan la media del tiempo total de ejecución empleado por las dos versiones paralelas heterogéneas.

Tabla C.1: Fitness promedio encontrado por cada variante algorítmica. Los mejores valores están remarcados en negro.

Instancias	*Opciones algorítmicas*				
	a	*b*	*c*	*d*	*e*
acin1	45591.14	46875.20	46625.97	47157.97	**51117.30**
acin2	143254.33	144620.30	145973.78	154784.23	**230042.20**
acin3	153979.90	156712.22	158993.8	187271.65	**274172.00**
acin5	145087.87	146589.80	142309.87	165423.87	**265234.00**
acin7	155781.55	158004.87	154792.4	214148.78	**298218.33**
acin9	310603.73	325928.66	311604.54	326798.24	**540350.33**

Tabla C.2: Números de contigs promedio logrado por cada variante algorítmica. Los mejores valores están remarcados en negro.

Instancias	*Opciones algorítmicas*				
	a	*b*	*c*	*d*	*e*
acin1	5.55	4.50	5.00	4.20	**3.42**
acin2	236.00	236.00	236.00	236.00	**145.80**
acin3	358.00	358.00	358.00	358.00	**323.40**
acin5	552.00	552.00	552.00	552.00	**478.66**
acin7	722.00	722.00	722.00	722.00	**677.07**
acin9	552.00	552.00	552.00	552.00	**495.50**

Tabla C.3: Tiempo medio de ejecución total (en segundos) empleado por las variantes algorítmica d y e. Los mejores valores están remarcados en negro.

Instancias	*Opciones algorítmicas*	
	d	*e*
acin1	**60.62**	121.64
acin2	**520.22**	988.42
acin3	**1458.21**	2887.26
acin5	**1028.89**	2016.62
acin7	**1458.21**	3251.81
acin9	**2119.51**	4599.34

Apéndice D

Publicaciones que sustentan la tesis doctoral

En este apéndice se presenta el conjunto de trabajos publicados como resultado de las investigaciones desarrolladas a lo largo de esta tesis doctoral. Estas publicaciones avalan el interés, la validez y las contribuciones de esta tesis doctoral en la literatura, dado que estos trabajos se han publicado en foros de prestigio y, por lo tanto, se han sometidos a procesos de revisión por reconocidos investigadores especializados. A continuación se muestran las referencias de todas las publicaciones.

Revistas indexadas

- Gabriela Minetti, Enrique Alba y Gabriel Luque. Seeding strategies and recombination operators for solving the DNA fragment assembly problem. *Information Processing Letters*, Volume 108, Número 3, pág. 97-100, Octubre 2008.

- Gabriela Minetti, Guillermo Leguizamón y Enrique Alba. Assembling DNA Sequences Containing Noisy Information With Metaheuristic Algorithms. En evaluación *Journal of Information Sciences*, Elsevier, 2011.

- Gabriela Minetti, Guillermo Leguizamón y Enrique Alba. A new Parallel and Hybrid Metaheuristic for Solving Noisy DNA Strands. En evaluación *Journal of Information Sciences*, Elsevier, 2011.

Congresos Internacionales

- Gabriela Minetti y Enrique Alba. Metaheuristic Assemblers of DNA strands: Noiseless and Noisy Cases. *2010 IEEE Congress on Evolutionary Computation*. Barcelona, España. Julio 2010.

- Gabriela Minetti, Gabriel Luque, Guillermo Leguizamón y Enrique Alba. A new Hybrid SA for Solving the DNA Fragment Assembly Problem. *XXVIII Internacional Conference of the Chilean Computing Science Society (SCCC)*, pág. 109-116, Noviembre de 2009.

- Gabriela Minetti, Gabriel Luque y Enrique Alba. Variable Neighborhood Search as Genetic Algorithm Operator for DNA Fragment Assembling Problem. *Eighth International Conference on Hybrid Intelligent Systems*, IEEE Computer Society, pág. 714-719, 2008.

Congresos Nacionales

- Gabriela Minetti, Enrique Alba y Gabriel Luque. Variable Neighborhood Search for Solving the DNA Fragment Assembly Problem. *XIII Congreso Argentino de Ciencias de la Computación (CACIC 2007)*, pág. 1359-1371, Octubre de 2007.

Bibliografía

[1] H. L. E. Aarts and J. Korst. *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing*. Wiley, Chichester, 1989.

[2] P. Adamidis and V. Petridis. Co-operating populations with different evolution behaviours. In *3rd IEEE Conference on Evolutionary Computation*, pages 188–191. IEEE Press, Nagoya, Japan, May 1996.

[3] J. Aguilar-Ruiz. Shifting and scaling patterns from gene expression data. *Bioinformatics*, 21(20):3840–3845, 2005.

[4] E. Alba. Parallel evolutionary algorithms can achieve super-linear performance. *Inf. Process. Lett.*, 82(1):7–13, 2002.

[5] E. Alba. *Parallel Metaheuristics: A New Class of Algorithms*. WILEY Series on Parallel and Distributed Computing. Wiley, 2005.

[6] E. Alba and B. Dorronsoro. *Cellular Genetic Algorithms*. Operations Research/Computer Science Interfaces. Springer-Verlag Heidelberg„ 2008.

[7] E. Alba, J. Garcia-Nieto, L. Jourdan, and E.-G. E.-G Talbi. Gene selection in cancer classification using pso/svm and ga/svm hybrid algorithms. In *IEEE Congress on Evolutionary Computation, 2007. CEC 2007*, pages 284–290, Singapore, 2007.

[8] E. Alba and G. Luque. A new local search algorithm for the dna fragment assembly problem. In *Evolutionary Computation in Combinatorial Optimization, EvoCOP'07*, volume 4446 of *Lecture Notes in Computer Science*, pages 1–12. Springer, Valencia, Spain, 2007.

[9] E. Alba and G. Luque. A hybrid genetic algorithm for the dna fragment assembly problem. In C. Cotta and J. van Hemert, editors, *Recent Advances in Evolutionary Computation for Combinatorial Optimization*, volume 153 of *Studies in Computational Intelligence*, pages 101–112. Springer Berlin / Heidelberg, 2008.

[10] E. Alba and the MALLBA Group. Mallba: A library of skeletons for combinatorial optimisation. In R. F. B. Monien, editor, *Proceedings of the Euro-Par*, volume 2400 of *LNCS*, page 927932. Springer-Verlag, Paderborn (GE), 2002.

[11] S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, (1990):403–410, 1990.

[12] S. Altschul, T. Madden, A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. Lipman. Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic Acids Research*, (25):3398–3402, 1997.

[13] D. Anastassiou. Genomic signal processing. *IEEE Signal Processing Magazine*, pages 8–20, July 2001.

[14] D. Applegate, R. Bixby, and V. Chavatal. On the solution of travelling salesman problems. *Documenta Mathematica*, 3:645–656, 1998.

[15] T. Arredondo, P. Neelakanta, and D. D. Groff. Fuzzy attributes of a dna complex: Development of a fuzzy inference engine for codon-junk codon delineation. *Artificial Intelligence in Medicine*, 35(1-2):87–105, 2005.

[16] T. Bäck. Selective pressure in evolutionary algorithms: A characterization of selection mechanisms. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, pages 57–62. IEEE Press, 1994.

[17] T. Bäck. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford University Press, Oxford, UK, 1996.

[18] T. Bäck and F. Hoffmeister. Extended selection mechanisms in genetic algorithms. In *Proceedings of the 4th Int. Conference on Genetic Algorithms*, pages 92–99. Morgan Kaufmann, 1991.

[19] J. E. Baker. Adaptive selection methods for genetic algorithms. In J. Grefenstette, editor, *2nd International Conference On Genetic Algorithms*, pages 100–11. 1987.

[20] J. E. Baker. Reducing bias and inefficiency in the selection algorithm. In J. Grefenstette, editor, *2nd International Conference On Genetic Algorithms*, pages 14–21. 1987.

[21] S. Benner. Patterns of divergence in homologous proteins as indicators of tertiary and quaternary structure. *Advances in Enzyme Regulation*, 28:219–236, 1988.

[22] J. Blazewicz, P. Lukasiak, and M. Milostan. Application of tabu search strategy for finding low energy structure of protein. *Artificial Intelligence in Medicine*, 35(1-2):135–145, 2005.

[23] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 35:268–308, September 2003.

[24] H. Bohr, J. Bohr, S. Brunak, R. Cotterill, and H. Fredholm. A novel approach to prediction of the 3-dimensional structures of protein backbones by neural networks. *FEBS Letters*, (261):43–46, 1990.

[25] A. L. Bouthillier and T. G. Crainic. Co-operative parallel meta-heuristic for vehicle routing problems with time windows. *Computers and Operation Research*, 32(7):685–1708, 2005.

[26] J. Bowie, R. Lüthy, and D. Eisenberg. A method to identify protein sequence that fold into a known three-dimensional structure. *Science*, (253):164–170, 1991.

[27] G. E. P. Box and M. E. Muller. A note on the generation of random normal deviates. In *Annals of Mathematical Statistics*, volume 29, pages 610–611, 1958.

[28] C. Burks, M. Engle, S. Forrest, R. Parsons, C. Soderlund, and P. Stolorz. Stochastic optimization tools for genomic sequence assembly. In M. Adams, C. Fields, and J. Venter, editors, *Automated DNA Sequencing and Analysis*, pages 249–259. Academic Press, 1994.

[29] C. Burks, R. Parsons, and M. Engle. Integration of competing ancillary assertions in genome assembly. In R. Altman, D. Brutlag, P. Karp, R. Lathrop, and D. Searls, editors, *Proceedings Second International Conference on Intelligent Systems for Molecular Biology*, pages 62–69, Menlo Park, CA, 1994. AAA1 Press.

[30] M. Burset and R. Guigo. Evaluation of gene structure prediction programs. *Genomics*, (34):353–367, 1996.

[31] Y. Cai, H. Yu, and K. Chou. Artificial neural network method for predicting HIV protease cleavage sites in protein. *Journal of Protein Chemical*, 17:607–615, 1998.

[32] C. Camacho and D. Thirumalai. Kinetics and thermodynamic of folding in model proteins. *Proceedings of National Academy of Sciences of the United States*, 93(13):63669–63672, 1993.

[33] E. Cantú-Paz. Migration, selection pressure, and superlinear speedups. In *Efficient and Accurate Parallel Genetic Algorithms*, pages 97–120. Kluwer Academic Publishers, 2000.

[34] H. Chang, N. Lo, W. Lu, and C. Kuo. Visualization and comparison of dna sequences by use of three-dimensional trajectories. *Proceedings of the First Asia-Paci.c Bioinformatics Conference APBC2003*, February 2003.

[35] E. Cheever, G. Overton, and D. Searls. Fast fourier transform-based correlation of dna sequences using complex plane encoding. *Comput. Appl. Biosci.*, 7(2):143–154, 1991.

[36] J. Chen, W. Hsu, M. Lee, and S. Ng. Systematic assessment of high-throughput experimental data for reliable protein interactions using network topology. *16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'04)*, pages 368–372, 2004.

[37] T. Chen and S. Skiena. A case study in genome-level fragment assembly. *The Eighth Symposium on Combinatorial Pattern Matching*, pages 206–223, 1997.

[38] T. Chen and S. Skiena. A case study in genome-level fragment assembly. *Bioinformatics*, 16, 2000.

[39] Y. Chen, E.R.Dougherty, and M. Bittner. Ratio-based decisions and the quantitative analysis of cdna microarray images. *Journal of Biomed. Opt*, (2):364–374, 1997.

[40] J. F. Chicano. *Metaheurísticas e Ingeniería del Software*. PhD thesis, University of Málaga, 2007.

[41] P. Chou and G. Fasmann. Prediction of the secondary structure of proteins from their amino acid sequence. *Advances in Enzymology*, (47):45–148, 1978.

[42] G. Churchill, C. Burks, M. Eggert, M. Engle, and M. Waterman. Assembling dna sequence fragments by shuffling and simulated annealing. Technical Report LA-UR-93-2287, Los Alamos National Laboratory, Los Alamos, NM, 1993.

[43] C. Cotta, A. J. Fernández, J. E. Gallardo, G. Luque, and E. Alba. *Metaheuristics in Bioinformatics: DNA Sequencing and Reconstruction*, pages 265–286. John Wiley Sons, Inc., 2008.

[44] C. Cotta, C. Sloper, and P. Moscato. Evolutionary Search of Thresholds for Robust Feature Set Selection: Application to the Analysis of Microarray Data. In *In Proceedings of the EvoWorkshops 2004, LNCS 3005*, pages 21–30. G.R. Raidl et. al (Eds.). Springer-Verlag, 2004.

[45] T. G. Crainic, A. T. Nguyen, and M. Gendreau. Cooperative multi-thread parallel tabu search with evolutionary adaptative memory. In *2nd International Conference on Metaheuristics*, Sophia Antiopolis, France, 1997.

[46] T. G. Crainic and M. Toulouse. Parallel strategies for metaheuristics. In *Handbook of Metaheuristics*, pages 475–513. Kluwer Academic Publishers, 2003.

[47] C. Darwin. *On the Origin of Species by Means of Natural Selection*. Londres, 1859.

[48] D. Dasgupta and Z. Michalewicz. *Evolutionary Algorithms in Engineering Applications*. Springer, Springer-Verlag Berlin Heidelberg, Alemania, 1997.

[49] L. Davis. Applying adaptive algorithms to domains. *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 162–164, 1985.

[50] L. Davis. *Handbook of genetic algorithms*. Van Nostrand Reinhold, 1991.

[51] K. Deb and A. Reddy. Reliable classification of two-class cancer data using evolutionary algorithms. *BioSystems*, 72:111–129, 2003.

[52] J. DeRisi, V. Lyer, and P. Brown. Exploring the metabolic and genetic control of gene expression on a genomic scale. *Science*, (278):680–686, 1997.

[53] M. Dorigo. *Optimization, Learning and Natural Algorithms*. PhD thesis, Politecnico di Milano, Italy, 1992.

[54] M. Dorigo and T. Stützle. The ant colony optimization metaheuristic: Algorithms, applications, and advances, 2003.

[55] B. Dorronsoro, E. Alba, G. Luque, and P. Bouvry. A self-adaptive cellular memetic algorithm for the dna fragment assembly problem. In *IEEE Congress on Evolutionary Computation*, pages 2651–2658, 2008.

[56] G. Dudek. Genetic algorithm with integer representation of unit start-up and shut-down times for the unit commitment problem. *European Transactions on Electrical Power*, 17(5):500–511, 2007.

[57] R. C. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. In *Proceedings of the Sixth International Symposium on Micromachine and Human Science*, pages 39–43, Nagoya, Japan, 1995.

[58] M. Eisen, P. Spellman, P. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns. *Proc. National Academy Sciences USA*, pages 14863–14868, 1998.

[59] M. Engle and C. Burks. Artificially generated data sets for testing dna fa algorithms. *Genomics*, 16, 1996.

[60] P. Festa. Greedy randomized adaptive search procedures. *AIROnews*, 7(4):7–11, 2003.

[61] J. Fickett. Finding genes by computer: the state of the art. *Trends Genetic*, (12):316–320, 1996.

[62] J. Fickett and A. Hatzigeorgiou. Eukaryotic promoter recognition. *Genome Research*, 7(9):861–878, 1997.

[63] J. Fickett and C. Tung. Assessment of protein coding measures. *Nucleic Acids Research*, (20):6641–6450, 1992.

[64] D. Fischer and D. Eisenberg. Protein fold recognition using sequence-derived predictions. *Protein Sci.*, 5:947–955, 1996.

[65] H. Flöckner, F. Domingues, and M. Sippl. Proteins folds from pair interactions: a blind test in fold recogition. *Proteins: Struct. Funct. Genet. 1*, pages 129–133, 1997.

[66] G. Fogel and D. Corne. *Evolutionary Computation in Bioinformatics*. Morgan Kaufmann, San Francisco, 2002.

[67] L. Fogel, A. Owens, and M. J. Walsh. *Artificial Intelligence Thorough Simulated Evolution*. John Wiley, Chichester, UK, 1996.

[68] X. Gan, A. Liew, and H. Yan. Missing value estimation for microarray data based on projection onto convex sets method. *Proceedings of the 17th International Conference on Pattern Recognition*, August 2004. Cambridge, United Kingdom.

[69] D. Gehlhaar, G. Verkhivker, P. Rejto, C. Sherman, D. B. Fogel, L. J. Fogel, , and S. Freer. Molecular recognition of the inhibitor ag-1343 by hiv-1 protease: conformationally flexible docking by evolutionary programming. *Chem. Biol.*, 2:317–324, 1995.

[70] F. Glover. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 1977.

[71] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & OR*, pages 533–549, 1986.

[72] F. Glover. A template for scatter search and path relinking. In *Selected Papers from the Third European Conference on Artificial Evolution*, AE '97, pages 3–54, London, UK, 1998. Springer-Verlag.

[73] F. Glover and G. Kochenberger. *Handbook of Metaheuristics*. Kluwer Academic Publishers, Norwell, MA, 2002.

[74] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[75] D. Golberg and R. Lingle. Alleles, loci and the traveling salesman problem. *Proceedings of the First International Conference on Genetic Algorithms*, pages 154–159, 1986.

[76] D. Goldberg, B. Korb, and K. Deb. Messy genetic algorithms, motivation, analysis, and first results. *Complex Systems*, 3:493–530, 1989.

[77] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989.

[78] F. Gortázar, A. Duarte, M. Laguna, and R. Martí. Black box scatter search for general classes of binary optimization problems. *Comput. Oper. Res.*, 37:1977–1986, November 2010.

[79] J. Gray, S. Moughon, C. Wang, O. Schueler-Furman, B. Kuhlman, C. Rohl, and D. Baker. Protein-protein docking with simultaneous optimization of rigid-body displacement and side-chain conformations. *Journal of Molecular Biology*, 331(1):281–299, 2003.

[80] P. Green. Phrap sequence assembly program. *University of Washington, Seattle*, 1996.

[81] J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Trans. Syst. Man Cybern.*, 16:122–128, January 1986.

[82] R. Guigo. *DNA composition, codon usage and exon prediction*. Academic Press, m.j. bishop edition, 1999.

[83] B. Hajek. Cooling schedules for optimal annealing. *MATHEMATICS OF OPERATIONS RESEARCH*, 13(2):311–329, 1988.

[84] P. Hansen, N. Mladenovic, and J. M. Pérez. Variable neighbourhood search. *Revista Iberoamericana de Inteligencia Artificial*, (19):77–92, 2003. ISSN: 1137-3601.

[85] J. Hargbo and A. Elofsson. A study of hidden markov models that use predicted secondary structures for fold recognition. *Proteins: Struct. Funct. Genet.*, (36):68–87, 1999.

[86] B. Hartke. Global geometry optimisation of clusters using genetic algorithms. *Journal of Physical Chemistry*, 97:9973–9976, 1993.

[87] B. Hartke. Global geometry optimisation of clusters using using a growth strategy optimized by a genetic algorithm. *Chemical Physics Letters*, 240:560–565, 1995.

[88] F. Herrera, M. Lozano, and J. L. Verdegay. Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis. *Artificial Intelligence Review*, 12:265–319, 1998.

[89] T. Hiroyasu, M. Miki, and M. Negami. Distributed genetic algorithms with randomized migration rate. In *Proceedings of the IEEE Conference of Systems, Man and Cybernetics*, volume 1, pages 689–694. IEEE Press, 1999.

[90] J. H. Holland. *Adaptation in Natural and Artificial Systems*. The MIT Press, Cambridge, Massachusetts, first edition, 1975.

[91] N. Holter, M. Mitra, A. Maritan, M. Cieplak, J. Banavar, and N. Fedoroff. Fundamental patterns underlying gene expression pro.les: simplicity from complexity. *Proc. National Academy Sciences USA*, pages 8409–8414, 2000.

[92] T. Huang and V. Kecman. Bias term b in SVMs again. In M. Verleysen, editor, *Proceedings of the ESANN 2004, 12th European Symposium on Artificial Neural Networks*, pages 441–448, 2004.

[93] W. Huang and A. Madan. CAP3: A DNA Sequence Assembly Program. *Genome Research*, 9(9):868–877, 1999.

[94] R. Hughey and A. Krogh. Hidden markov models for sequence analysis: Extension and analysis of the basic method. *CABIOS*, 2(12):95–107, 1996.

[95] A. I. Inc. Genepix pro 3.0, 2001.

[96] D. Jones, W. Taylor, and J. Thornton. A new appoach to protein fold recognition. *Nature*, (358):86–89, 1992.

[97] G. Jones, P. Willett, R. Glen, A. Leach, and R. Taylor. Development and validation of a genetic algorithm for flexible docking. *Journal of Molecular Biology*, (267):727–748, 1997.

[98] K. A. D. Jong. *An analysis of the behavior of a class of genetic adaptive systems*. PhD thesis, Ann Arbor, MI, USA, 1975. AAI7609381.

[99] H. G. Jr. *Scientific Method in Practice*. Cambridge, 1st edition, 2002.

[100] O. Karpenko, J. Shi, and Y. Dai. Prediction of mhc class ii binders using the ant colony search strategy. *Artificial Intelligence in Medicine*, 35(1):147–156, 2005.

[101] K. Karplus, K. Sjölander, C. Barrett, M. Cline, D. Haussler, R. Hughey, L. Holm, and C. Sander. Predicting structures using hidden markov models. *Proteins: Struct. Funct. Genet. Suppl 1*, pages 134–139, 1997.

[102] E. Katchalski-Katzir, I. Shariv, M. Eisenstein, A. Friesem, C. Aflalo, and I. Vakser. Molecular surface recognition: Determination of geometric fit between proteins and their ligands by correlation techniques. *Proceedings of the National Academy of Sciences of the United States of America*, 89(6):2195–2199, 1992.

[103] K. Kim and C. Mohan. Parallel hierarchical adaptive genetic algorithm for fragment assembly. In IEEE, editor, *The 2003 Congress on Evolutionary Computation, 2003. CEC03.*, volume 1, pages 600–607. 2003.

[104] S. Kirkpatrick, C. G. Jr, and M. Vecchi. Optimization by simulated annealing. *Science*, (220):671–680, 1983.

[105] G. Klimovsky. *Las desventuras del conocimiento científico. Una introducción a la epistemología.* A-Z editora, Bs.As., 1997.

[106] J. R. Koza. Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems. Technical Report STAN-CS-90-1314, Stanford University, 1990.

[107] J. R. Koza. *Genetic Programming.* MIT Press, Cambridge, 1992.

[108] N. Krasnogor, B. Blackburne, E. Burke, and J. Hirst. Multimeme algorithms for protein structure prediction. In J. M. Guervos, P. Adamidis, H. Beyer, J. Fernandez-Villacañas, and H. Schwefel, editors, *Parallel Problem Solving from Nature - PPSN VII, 7th International Conference*, number 2439 in Lecture Notes in Computer Science, LNCS, page 769 ff. Springer-Verlag, 2002.

[109] M. Laguna and R. Martí. *Scatter Search: Methodology and Implementations in C.* Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[110] M. Laguna and R. Martí. Experimental testing of advanced scatter search designs for global optimization of multimodal functions. *J. of Global Optimization*, 33:235–255, October 2005.

[111] P. Larrañaaga, R. Etxeberria, J. A. Lozano, and J. M. Peña. Optimization by learning and simulation of bayesian and gaussian networks. Technical Report KZZA-IK-4-99, Department of Computer Science and Artificial Intelligence, University of the Basque Country, 1999.

[112] N. Larsen, J. Engelbrecht, and S. Brunak. Analysis of eukaryotic promoter sequences reveals a systematically occurring ct-signal. *Nucleic Acids Research*, (23):1223–1230, 1995.

[113] L. Li and S. Khuri. A comparison of dna fragment assembly algorithms. In *Proc. of the 2004 International Conference on Mathematics and Engineering Techniques in Medicine and Biological Sciences*, pages 329–335, Las Vegas, 2004.

[114] A. Liew, H. Yan, and M. Yang. Pattern recognition techniques for the emerging field of bioinformatics: A review. *Pattern recognition Society*, page 0000000, 2005.

[115] S.-L. Lin, W. F. Punch, and E. Goodman. Coarse-grain parallel genetic algorithms: Categorization and new approach. In *6th IEEE Symposium on Parallel and Distributed Processing*, pages 28–37, 1994.

[116] E. T. L.M. Gambardella and G. Agazzi. Macs-vrptw: A multiple ant colony system for vehicle routing problems with time windows. In *New Ideas in Optimization*, pages 63–76. McGraw-Hill Ltd., Maidenhead, UK, 1999.

[117] H. R. Lourenço, O. Martin, and T. Stützle. Iterated local search. In *Handbook of Metaheuristics*, pages 321–353. Kluwer Academic Publishers, 2002.

[118] G. Luque. *Resolución de Problemas Combinatorios con Aplicación Real en Sistemas Distribuidos*. PhD thesis, University of Málaga, 2006.

[119] G. Luque and E. Alba. Metaheuristics for the dna fragment assembly problem. *International Journal of Computational Intelligence Research*, 1(2):98–108, 2005.

[120] G. Luque, E. Alba, and S. Khuri. Chapter 16: Assembling dna fragments with a distributed genetic algorithm. In *Parallel Algorithms for Bioinformatics*. Wiley, 2006.

[121] H. Mamitsuka. Finding the biologically optimal alignment of multiple sequences. *Artificial Intelligence in Medicine*, 35(1-2):9–18, 2005.

[122] P. Meksangsouy and N. Chaiyaratana. DNA fragment assembly using an ant colony system algorithm. In *The 2003 Congress on Evolutionary Computation, 2003. CEC03*, volume 3, pages 1756– 1763. IEEE. ISBN: 0-7803-7804-0, 2003.

[123] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of state calculations by fast computing machines. *J. Chem. Phys.*, 21:1087, 1953.

[124] H. Mühlenbein. The equation for response to selection and its use for prediction. *Evolutionary Computation,*, (5):303–346, 1998.

[125] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, third edition, 1999.

[126] M. Miki, T. Hiroyasu, and J. Wako. Adaptive temperature schedule determined by genetic algorithm for parallel simulated annealing. In *The 2003Congress on Evolutionary Computation, CEC03*, volume 1, pages 459– 466, 2003.

[127] B. L. Miller and D. E. Goldberg. Genetic algorithms, selection schemes, and the varying effects of noise. *Evolutionary Computation*, 4:113–131, 1996.

[128] J. R. Miller, A. L. Delcher, S. Koren, E. Venter, B. Walenz, A. Brownley, J. Johnson, K. Li, C. Mobarry, and G. Sutton. Aggressive assembly of pyrosequencing reads with mates. *Bioinformatics*, 24(24):2818–2824, 2008.

[129] S. W. M.J. Rooman. Extracting information on folding from the amino acid sequence: Consensus regions with preferred conformation in homologous proteins. *Biochemestry*, 31(42):10239–10249, 1992.

[130] N. Mladenović and P. Hansen. Variable neighborhood algorithm-a new metaheuristic for combinatorial optimization. In *Optimization Days*, page 112, 1995.

[131] N. Mladenović and P. Hansen. Variable neighborhood search. *Computers and Operations Research*, 24(11):1097–1100, 1997.

[132] E. W. Myers. A whole-genome assembly of drosophila. *Science*, 287:219–2204, 2000.

[133] S. Needleman and C. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Molecular Biology*, 48(3):443–453, 1970.

[134] J. Ning, C. Moore, and J. Nelson. Preliminary wavelet analysis of genomic sequences. *Proceedings of the Second IEEE Computer Society Bioinformatics Conference CSB2003*, pages 509–510, August 2003.

[135] C. Notredame and D. Higgins. SAGA: sequence alignment by genetic algorithm. *Nucleic Acids Research*, 24(8):1515–1524, 1996.

[136] C. Notredame, L. Holm, and D. Higgins. COFFEE: an objective function for multiple sequence alignments. *Bioinformatics*, 14(5):407–422, 1998.

[137] I. Oliver, D. Smith, and J. Holland. A study of permutation crossover operators on the travelling salesman problem. *Proceedings of the First International Conference on Genetic Algorithms*, pages 224–230, 1986.

[138] R. J. Parsons, S. Forrest, and C. Burks. Genetic algorithms, operators, and dna fragment assembly. In *Machine Learning*, pages 11–33. Kluwer Academic Publishers, 1995.

[139] W. Pearson. Comparison of methods for searching protein sequence databases. *Protein Sci.*, 4:1145–1160, 1995.

[140] W. Pearson and D. Lipman. Improved tools for biological sequence analysis. *Proc. Natl Acad. Sci. USA,85*, pages 2444–2448, 1998.

[141] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz. Boa: The bayesian optimization algorithm. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smit, editors, *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99*, volume 1, pages 525–532, Orlando, FL, 1999. Morgan Kaufmann Publishers, San Francisco, CA.

[142] P. Pevzner. *Computational molecular biology: An algorithmic approach*. The MIT Press, 2000.

[143] F. Pinel, B. Dorronsoro, and P. Bouvry. A new parallel asynchronous cellular genetic algorithm for de novo genomic sequencing. In *Proceedings of the 2009 International Conference of Soft Computing and Pattern Recognition*, SOCPAR '09, pages 178–183, Washington, DC, USA, 2009. IEEE Computer Society.

[144] J. Prasad, S. Comeau, S. Vajda, and C. Camacho. Consensus alignment for reliable framework prediction in homology modeling. *Bioinformatics*, pages 1–10, 2003.

[145] N. Qian and T. Sejnowski. Predicting the secondary structure of globular proteins using neural network models. *Journal of Molecular Biology*, (1988):865–884, 1988.

[146] M. R, J. W. Tukey, and W. A. Larsen. Variations of box plots. *The American Statistician*, 32(1):12–16, 1978.

[147] D. J. Ram, T. H. Sreenivas, and K. G. Subramaniam. Parallel simulated annealing algorithms. *Journal fo Parallel and Distributed Computing*, 37:207–212, 1996.

[148] C. R. Reeves. *Modern Heuristic Techniques for Combinatorial Problems.* Halsted Press, New York, 1993.

[149] D. Rice and D. Eisenberg. A 3D-1D substitution matrix for protein fold recognition that includes predicted secondary structure of the sequence. *Journal of Molecular Biology*, (267):1026–1038, 1997.

[150] S. Riis and A. Krogh. Improving prediction of protein secondary structure using structured neural networks and multiple sequence alignments. *Journal of Computational Biology*, 3:163–183, 1996.

[151] B. Robson, J. Garnier, G. Barton, and R. Russel. Protein structure prediction. *Nature*, 316(6412):505–506, 1992.

[152] B. Rost and C. Sander. Prediction of protein secondary structure structure at better than 70 % accuracy. *Journal of Molecular Biology*, (232):584–599, 1993.

[153] J. Ruan, K. Wang, J. Yang, L. Kurgan, and K. Cios. Highly accurate and consistent method for prediction of helix and strand content from primary protein sequences. *Artificial Intelligence in Medicine*, (35):19–35, 2005.

[154] F. O. S. Rogic, A.K. Mackworth. Evaluation of gene-finding programs on mammalian sequences. *Genome Research*, 2(11):817–832, 2001.

[155] M. Sadowski, J. Parish, and D. Westhead. Automated derivation and refinement of sequence length patterns for protein sequences using evolutionary computation. *BioSystems*, (81):247–254, 2005.

[156] S. Salzberg, A. Delcher, A. Kasif, and S. White. Microbial gene identification using interpolated markov models. *Nucleic Acids Research*, 26:544–548, 1998.

[157] F. Sanger, A. Coulson, G. Hong, D. Hill, and G. Petersen. Nucleotide Sequence of Bacteriophage Lambda DNA. *Journal of Molecular Biology*, 162(4):729–773, 1982.

[158] J. D. Schaffer, R. A. Caruana, L. Eshelman, and R. Das. A study of control parameters affecting online performance of genetic algorithms for function optimization. In *Proceedings of the third international conference on Genetic algorithms*, pages 51–60, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[159] S. Schulze-Kremer. Genetic algorithms for protein tertiary structure prediction. *Parallel Problem Solving from Nature II*, pages 391–400, 1992.

[160] H. Schwefel. *Evolution and Optimum Seeking.* John Wiley & Sons, New York, 1994.

[161] J. Setubal and J. Meidanis. *Introduction to Computational Molecular Biology.* International Thomson Publishing, 20 park plaza, Boston, MA02116, 1999.

[162] R. P. Sheridan, J. S. Dixon, and R. Venkataraghavan. Generating plausible protein folds by secondary structure similarity. *Int. J. Pept. Protein Research*, (25):132–143, 1985.

[163] J. Skolnick and A. Kolinski. Simulations of athe folding of a globular protein. *Science*, 250:1621–1625, 1990.

[164] T. Smith and M. Waterman. Comparison of biosequences. *Adv. Appl. Math.*, (2):482–489, 1981.

[165] T. Smith and M. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, (147):195–197, 1981.

[166] F. Solis and R. Wets. Minimization by random search techniques. *Math. Ops. Research*, 6:10–30, 1981.

[167] J. Song, T. Ware, S. Liu, and M. Surette. Comparative genomics via wavelet analysis for closely related bacteria. *EURASIP J. Appl. Signal Process*, (1):5–12, 2004.

[168] E. Sonnhammer, S. Eddy, and R. Durbin. Pfam: a comprehensive database of protein domain families based on seed alignmnets. *Proteins, Struct. Funct. Genet.*, (28):405–420, 1997.

[169] M. Soto, A. Ochoa, S. Acid, and L. M. de Campos. Introducing the polytree aproximation of distribution algorithm. In *Second Symposium on Arti¯cial Intelligence. Adaptive Systems. CIMAF99*, pages 360–367, La Habana, 1999.

[170] P. Spellman, G. Sherlock, M. Zhang, V. Iyer, K. Anders, M. Eisen, P. Brown, D. Botstein, and B. Futcher. Comprehensive identi.cation of cell cycle-regulated genes of the yeast saccharomyces cerevisiae by microarray hybridization. *Molecular Biol. Cell*, (9):3273–3297, 1998. http://cellcycle-www.stanford.edu.

[171] T. Stützle. Local search algorithms for combinatorial problems analysis, algorithms and new applications. Technical report, DISKI Dissertationen zur KÄunstliken Intelligenz, Sankt Augustin, Germany, 1999.

[172] D. Sussilo, A. Kundaje, and D. Anastassiou. Spectrogram analysis of genomes. *EURASIP Journal of Appl. Signal Process*, (1):29–42, 2004.

[173] G. G. Sutton, O. White, M. D. Adams, and A. R. Kerlavage. TIGR Assembler: A new tool for assembling large shotgun sequencing projects. *Genome Science and Technology*, pages 9–19, 1995.

[174] M. Suzuki, T. Tsuji, and H. Ohtake. A model of motor control of the nematode c. elegans with neuronal circuits. *Artificial Intelligence in Medicine*, 35(1-2):75–86, 2005.

[175] E. Talbi and H. Meunier. Hierarchical parallel approach for gsm mobile network design. *J. Parallel Distrib. Comput.*, 66:274–290, February 2006.

[176] E.-G. Talbi. A taxonomy of hybrid metaheuristics. *Journal of Heuristics*, 8:541–564, September 2002.

[177] E.-G. Talbi. *Metaheuristics - From Design to Implementation*. Wiley, 2009.

[178] E.-G. Talbi and V. Bachelet. Cosearch: A parallel cooperative metaheuristic. *Journal of Mathematical Modelling and Algorithms*, 5(2):5–22, 2006.

[179] Y. Tang, Y.-Q. Zhang, Z. Huang, and X. Hu. Granular svm-rfe gene selection algorithm for reliable prostate cancer classification on microarray expression data. *Fifth IEEE Symposium on Bioinformatics and Bioengineering (BIBE'05)*, pages 290–293, 2005.

[180] G. Tao and Z. Michalewicz. *Proceedings of the 5th Parallel Problem Solving from Nature*, chapter Inver-over operator for the ETSP, pages 803–812. Amsterdam, 1998.

[181] J. Thompson, D. Higgins, and T. Gibson. CLUSTALW: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Research*, 22:4673–4680, 1994.

[182] S. Tiwari, S. Ramachandran, A. Bhattacharya, S. Bhattacharya, and R.Ramaswamy. Prediction of probable genes by fourier analysis of genomic sequences. *Comput. Appl. Biosci.*, (13):263–270, 1997.

[183] S. Tongchim and P. Chongstitvatana. Parallel genetic algorithm with parameter adaptation. *Information Processing Letters*, 52(1):47–54, 2002.

[184] G. Towell and J. Shavlik. Knowledge-based artificial neural networks. *Artificial Intelligence*, (70):119–165, 1994.

[185] O. Troyanskaya, M. Cantor, G. Sherlock, P. Brown, T. Hastie, R.Tibshirani, D. Botstein, and R. Altman. Missing values estimation methods for DNA microarrays. *Bioinformatics*, (17):520–525, 2001.

[186] J. Tukey. *Exploratory Data Analysis*. Addison-Wesley, 1977.

[187] E. Uberbacher, Y. Xu, and R. Mural. Discovering and understanding genes in human DNA sequence using GRAIL, Methods Enzymol. *Series title: Computer methods for macromolecular sequence analysis*, (266):259–281, 1996.

[188] A. Valouev, D. C. Schwartz, S. Zhou, and M. S. Waterman. An algorithm for assembly of ordered restriction maps from single DNA molecules. *Proceedings of the National Academy of Sciences*, 103(43):15770–15775, 2006.

[189] R. Venkateswaran, Z. Obradovič, and C. S. Raghavendra. Cooperative genetic algorithm for optimization problems in distributed computer systems. In *Proceedings of the 2nd Online Workshop on Evolutionary Computation*, pages 49–52, 1996.

[190] H. Wang, J. Dopazo, L. de la Fraga, Y. Zhu, and J. Carazo. Self-organizing tree-growing network for the classification of protein sequences. *Protein Sci*, 7:2613–2622, 1998.

[191] T. Werner. Models for prediction and recognition of eukaryotic promoters. *Mammalian Genome*, 2(10):168–175, 1999.

[192] L. D. Whitley, T. Starkweather, and D. Fuquay. Scheduling problems and traveling salesmen: The genetic edge recombination operator. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 133–140. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1989.

[193] J. Whittaker. *Graphical models in applied multivariate statistics*. John Wiley & Sons, Inc., 1990.

[194] Y. Wu, A. Liew, H. Yan, and M. Yang. Db-curve: a novel 2d method of dna sequence visualization and representation. *Journal of Chemical Physics Letters*, (367):170–176, 2003.

[195] M. Yamamura, T. Ono, and S. Kobashi. Character-preserving genetic algorithms for travelling salesman problem. *Journal of Japan Society for Artificial Intelligence*, 6:1049–1059, 1992.

[196] J. Yang and C. Kao. A family competition evolutionary algorithm for automated docking of flexible ligands to proteins. *IEEE Transactions on Information Technology in Biomedicene*, 4(3):225–237, 2000.

[197] L. Yeung, L. Szeto, A. Liew, and H. Yan. Dominant spectral component analysis for transcriptional regulations using microarray time-series data. *Bioinformatics*, 5(20):742–749, 2004.

[198] D. York, T. Darden, L. Pedersen, and M. Anderson. Molecular dynamics simulation of hiv-1 protease in a crystalline environment and in solution. *Biochemestry*, 32(6):1143–1153, 1993.

[199] M. Zhang. Identification of protein coding regions in the human genome by quadratic discriminant analysis. *Proc. Natl. Acad. Sci. USA*, (94):565–568, 1997.

[200] X. Zhang, F. Chen, Y. Zhang, S. Agner, M. Akay, Z. Lu, M. Waye, and S. Tsui. Signal processing techniques in genomic engineering. *Proc. IEEE*, 9(12):1822–1833, 2002.

[201] W. Zhao, M. Fanning, and T. Lane. Efficient rnai-based gene family knockdown via set cover optimization. *Artificial Intelligence in Medicine*, 35(1-2):61–73, 2005.

Índice alfabético