UNIVERSIDAD DE CONCEPCIÓN
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA
Y CIENCIAS DE LA COMPUTACIÓN

DESARROLLO Y EVALUACIÓN DE TÉCNICAS METAHEURÍSTICAS HÍBRIDAS PARA EL PROBLEMA DE LA SUBSECUENCIA DE RACHA MÁS LARGA (LRS)

POR

Martín Isla Pino

Memoria de Título presentada a la Facultad de Ingeniería de la Universidad de Concepción para optar al título profesional de Ingeniero(a) Civil Informático(a)

Profesor(es) Guía
Pedro Pinacho Davidson.
Camilo Chacón Sartori.

Agosto 2025
Concepción (Chile)

Dedico este trabajo a mis abuelitas, quienes, a pesar de su delicado estado de salud, siempre han expresado el deseo de verme titulado y han sido una fuente constante de amor y fortaleza.

También lo dedico a mis padres y hermanos, por ser mi apoyo incondicional y fuente de inspiración en cada paso que doy.

A mi pareja, por su amor, comprensión y apoyo constante, que me motivaron a seguir adelante en los momentos más desafiantes.

Y a mis amigos del período universitario, por su compañía, apoyo y los momentos compartidos que enriquecieron esta linda y desafiante etapa.

A todos ellos, gracias por acompañarme en este camino y por creer siempre en mí.

Agradecimientos

Quisiera expresar mi más sincero agradecimiento al profesor Pedro Pinacho, mi profesor guía, por su incondicional apoyo y orientación a lo largo de todo el proceso de elaboración de esta memoria de título. Su paciencia durante las revisiones, así como sus claras y detalladas explicaciones respecto a los contenidos, fueron fundamentales para el desarrollo de este trabajo.

También deseo agradecer a Camilo Chacón, cuya valiosa contribución fue clave para la implementación del método utilizado en esta investigación. El método que aplicamos fue originalmente propuesto por él en un trabajo anterior, y su disposición para aclarar dudas y brindar apoyo directo facilitó enormemente el avance y la comprensión del tema.

Finalmente, quiero agradecer a mi familia y a mi pareja por su apoyo constante, comprensión y motivación durante todo este proceso.

A todos ellos, mi profundo reconocimiento y gratitud.

Este trabajo fue parcialmente financiado por el Proyecto ANID Fondecyt N° 11230359, *"An immune inspired model of Intrusion Prevention System (IPS) for collaborative and distributed environments"*.

Resumen

El problema de la subsecuencia de racha más larga (LRS) es un desafío de optimización combinatoria NP-hard con aplicaciones en bioinformática, particularmente en el reensamblaje de genomas. Esta investigación aborda el LRS mediante el diseño, implementación y evaluación comparativa de diversas estrategias heurísticas y metaheurísticas, con el objetivo de explorar mejoras en la calidad de las soluciones respecto al estado del arte.

Entre las técnicas evaluadas, destaca particularmente una propuesta innovadora que combina el algoritmo BRKGA (Biased Random Key Genetic Algorithm) con modelos de lenguaje (LLMs), demostrando que esta integración supera el rendimiento del BRKGA clásico, actualmente considerado el método más robusto para este problema. Los experimentos revelaron que, si bien el BRKGA tradicional mantiene un buen desempeño general, la versión híbrida con LLMs logra mejores resultados, especialmente en instancias complejas con alfabetos de gran tamaño, donde las técnicas convencionales presentan limitaciones.

Estos resultados destacan la aplicabilidad de este enfoque en nuevas líneas de investigación para el procesamiento de secuencias genómicas, representando un avance significativo en la resolución eficiente del problema, con implicaciones directas en el campo de la bioinformática.

Summary

The Longest Run Subsequence (LRS) problem is an NP-hard combinatorial optimization challenge with applications in bioinformatics, particularly in genome reassembly. This research addresses LRS through the design, implementation, and comparative evaluation of various heuristic and metaheuristic strategies, aiming to improve solution quality beyond the current state-of-the-art.

Among the evaluated techniques, an innovative approach combining the Biased Random Key Genetic Algorithm (BRKGA) with Large Language Models (LLMs) stands out. This hybrid method demonstrates superior performance compared to classical BRKGA, which is currently the most robust solution for this problem. Experimental results reveal that while traditional BRKGA maintains strong overall performance, the LLM-enhanced hybrid version achieves better outcomes—particularly in complex instances with large alphabets, where conventional techniques face limitations.

These findings highlight the applicability of this approach in new research directions for genomic sequence processing, representing a significant advance in efficiently solving the problem, with direct implications for bioinformatics.

Índice general

Índice de tablas

Índice de figuras

Capítulo 1. Introducción

La optimización combinatoria es una disciplina fundamental en la ciencia de la computación, que busca encontrar soluciones óptimas o cercanas a óptimas para problemas complejos. Estos problemas suelen involucrar la selección, disposición o combinación de elementos dentro de un conjunto finito, y su resolución eficiente es crucial en ámbitos como la logística, la ingeniería y la bioinformática, entre otras áreas [4]. En particular, en el ámbito de la bioinformática, la optimización combinatoria es esencial para el análisis de secuencias biológicas como el ADN, lo que permite avanzar en la comprensión de la genética, la evolución y el desarrollo de tratamientos médicos personalizados [5].

1.1. Contexto

Uno de los mayores retos en bioinformática es el proceso de ensamblaje genómico, en el cual las lecturas (*reads*) - fragmentos cortos de ADN obtenidos mediante técnicas de secuenciación - se ensamblan para formar secuencias más largas denominadas *contigs*. Posteriormente, estos contigs requieren ser ordenados correctamente durante la fase de andamiaje (*scaffolding*) para reconstruir la estructura completa del genoma [5]. Cuando se dispone de un genoma de referencia de alta calidad, este proceso se simplifica significativamente. Sin embargo, para aquellas especies que carecen de una buena referencia genómica, es necesario emplear métodos alternativos [6]. Entre estos destaca el uso de homología entre contigs de especies evolutivamente cercanas, dependiendo de estrategias capaces de manejar reordenamientos genómicos, filtrar ruido y resolver regiones repetitivas [7].

Para abordar este problema, Schrinner et al. [7] han propuesto el LRS (Longest Run Subsequence problem), un problema de optimización combinatoria que consiste en hallar la subsecuencia más larga de una cadena donde cada símbolo aparece en un único bloque continuo. En genómica, el LRS ayuda a ordenar contigs, mejorando la precisión del scaffolding en especies sin referencia. Sin embargo, también demostraron que el LRS es NP-hard [7], lo que limita su resolución exacta, especialmente en instancias grandes. Por ello, técnicas aproximadas como heurísticas y metaheurísticas son estrategias prometedoras para obtener soluciones cercanas a

las óptimas en tiempos razonables.

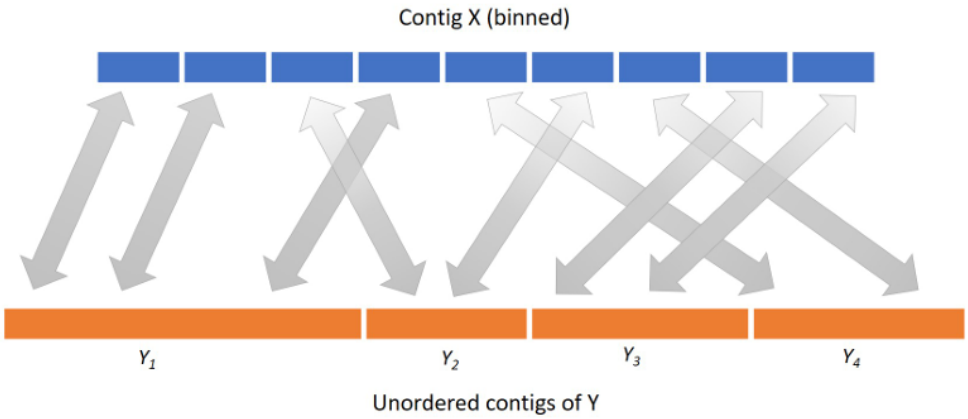


Figura 1.1: Ejemplo de inferencia de orden de contigs mediante el LRS [1].

Una instancia del problema de ordenamiento de contigs está definida por dos conjuntos de contigs, X y Y, donde cada contig representa un fragmento de ADN con un identificador único. El conjunto X contiene los contigs cuyo orden se desea determinar, mientras que Y representa contigs de referencia para el alineamiento.

La Figura 1.1 muestra el proceso de ordenamiento de contigs mediante el método LRS. En este enfoque, los contigs del conjunto X se dividen en segmentos de tamaño uniforme (*bins*) que se asocian a contigs del conjunto Y mediante alineamiento. La secuencia resultante $S = y_1y_1y_2y_1y_4y_2y_4y_3y_3$ puede presentar inconsistencias debido a errores de mapeo. El LRS identifica la subsecuencia válida más larga $S' = y_1y_1y_1y_4y_4y_3y_3$, donde cada contig aparece en bloques consecutivos, permitiendo reconstruir el ordenamiento más probable de X al eliminar asignaciones inconsistentes.

A pesar de los avances recientes, las soluciones actuales para el LRS presentan diversos desafíos que justifican la búsqueda de nuevas estrategias. Los métodos exactos ofrecen soluciones óptimas, pero no escalan de manera eficiente cuando se enfrentan a instancias de gran tamaño debido a su elevado costo computacional. Esto limita su aplicabilidad en escenarios reales donde se requiere procesar grandes volúmenes de datos.

Por otro lado, las metaheurísticas han demostrado un mejor rendimiento en términos de escalabilidad; sin embargo, no siempre logran alcanzar soluciones óptimas, especialmente cuando aumenta la cantidad de símbolos distintos en la cadena de entrada.

Estas limitaciones abren un espacio claro para la exploración de mejoras. En particular, se

plantea que existe un margen de mejora a través de técnicas híbridas modernas, poco exploradas hasta ahora para el LRS. Esta convicción constituye una de las motivaciones centrales de esta investigación, orientada a diseñar y evaluar enfoques híbridos capaces de superar el rendimiento de las soluciones existentes.

1.2. Definición del problema

Dada una cadena de entrada $s = s_1 s_2 \dots s_n$ sobre un alfabeto finito Σ (*sigma*), una racha (*run*) en s es una subcadena maximal $r = s_j s_{j+1} \dots s_k$ donde todos los caracteres son idénticos ($s_i = s_j$ para todo $i \in [j, k]$). Sea $R = \{r_1, \dots, r_m\}$ la secuencia de rachas en s, donde cada r_i se caracteriza por:

- Caracter $c(r_i) \in \Sigma$,
- *Length* $l(r_i) \in \mathbb{Z}^+$ (longitud de una racha).

Una *subsecuencia válida* $R' \subseteq R$ debe cumplir la *condición de contigüidad*:

$$\forall r_i, r_k \in R' \text{ con } i < k \text{ y } c(r_i) = c(r_k), \quad \nexists r_j \in R' \text{ tal que } i < j < k \text{ y } c(r_j) \neq c(r_i).$$

El problema LRS busca una subsecuencia válida R^* que maximice:

$$f(R^*) = \sum_{r_i \in R^*} l(r_i).$$

Ejemplo

Considere la cadena $s = \texttt{AGGCACT}$:

- **Rachas**:

$$\begin{aligned}
r_1 &= \texttt{A} \ (c(r_1) = \texttt{A}, l(r_1) = 1),\\
r_2 &= \texttt{GG} \ (c(r_2) = \texttt{G}, l(r_2) = 2),\\
r_3 &= \texttt{C} \ (c(r_3) = \texttt{C}, l(r_3) = 1),\\
r_4 &= \texttt{A} \ (c(r_4) = \texttt{A}, l(r_4) = 1),\\
r_5 &= \texttt{C} \ (c(r_5) = \texttt{C}, l(r_5) = 1),\\
r_6 &= \texttt{T} \ (c(r_6) = \texttt{T}, l(r_6) = 1).
\end{aligned}$$

- **Soluciones válidas**:

- $\{r_1, r_2, r_3, r_6\} \rightarrow$ Subsecuencia `AGGCT` (length $1 + 2 + 1 + 1 = 5$).
- $\{r_1, r_3, r_5, r_6\} \rightarrow$ Subsecuencia `ACCT` (length $1 + 1 + 1 + 1 = 4$).

- **Solución óptima**:
  - $\{r_1, r_2, r_3, r_5, r_6\} \rightarrow$ Subsecuencia `AGGCCT` (length $1 + 2 + 1 + 1 + 1 = 6$).

1.3. Objetivos

Diseñar, implementar y evaluar estrategias heurísticas y metaheurísticas para la optimización del problema LRS, con el fin de explorar mejoras en la calidad de las soluciones obtenidas con respecto al estado del arte.

1.3.1. Objetivos específicos

- Analizar el problema LRS y su relevancia en el contexto de la bioinformática, identificando sus desafíos computacionales y las metodologías existentes para abordar su resolución.
- Explorar mejoras en el rendimiento mediante ajuste de parámetros diferenciado según características de las instancias. como el largo de la secuencia y la diversidad de caracteres.
- Mejorar la heurística utilizada en los algoritmos existentes e incorporarla en el diseño de metaheurísticas para el LRS.
- Explorar enfoques innovadores como la hibridación de metaheurísticas con grandes modelos de lenguaje.
- Evaluar el impacto de las estrategias implementadas en términos de calidad de la solución.
- Realizar análisis comparativo de los resultados de las soluciones propuestas frente al estado del arte.

Capítulo 2. Marco teórico

En el estudio de problemas computacionales complejos, los algoritmos de optimización juegan un papel fundamental para encontrar soluciones óptimas o aproximadas. Existen varios métodos para obtener soluciones y la elección de estas estrategias depende del balance requerido entre precisión, tiempo de ejecución y recursos disponibles.

2.1. Métodos exactos

Los métodos exactos son algoritmos que garantizan la obtención de soluciones óptimas. Estos algoritmos se fundamentan principalmente en teorías matemáticas y analíticas que permiten reducir el espacio de soluciones sin perder la garantía de optimalidad [8]. Si bien estos métodos proporcionan soluciones óptimas, su aplicabilidad se ve limitada por el crecimiento exponencial de su complejidad computacional en problemas de gran tamaño.

2.1.1. Programación Dinámica(DP)

La Programación Dinámica es una técnica ampliamente utilizada para resolver problemas de optimización que pueden descomponerse en subproblemas más pequeños. Este método explora el espacio de soluciones compuesto por estados elementales para encontrar una solución óptima, pudiendo realizarse la búsqueda tanto en dirección hacia adelante (forward) como hacia atrás (backward) [9].

2.1.2. Programación Lineal Entera (ILP)

La Programación Lineal Entera es un modelo de optimización matemática que extiende la Programación Lineal al requerir que algunas o todas las variables de decisión tomen valores enteros [8]. Para resolver problemas ILP en la práctica, se emplean solvers avanzados como CPLEX, Gurobi o SCIP, que implementan técnicas híbridas (Branch-and-Cut, Planos de Corte, relajaciones, etc.). Un problema de ILP se formula de la siguiente manera:

$$
\begin{aligned}
\text{Maximizar} \quad & \mathbf{c}^T\mathbf{x} \quad \text{(Función objetivo lineal)} \\
\text{sujeto a} \quad & A\mathbf{x} \leq \mathbf{b} \quad \text{(Restricciones lineales)} \\
& \mathbf{x} \geq \mathbf{0} \quad \text{(No-negatividad)} \\
& x_j \in \mathbb{Z} \quad \forall j \in J \subseteq \{1, \ldots, n\} \quad \text{(Integralidad)}
\end{aligned}
$$

donde:

- $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$ es el vector de variables de decisión (enteras o binarias)
- $\mathbf{c} = [c_1, c_2, \ldots, c_n]$ es el vector de coeficientes de la función objetivo
- $A \in \mathbb{R}^{m \times n}$ es la matriz de restricciones
- $\mathbf{b} = [b_1, b_2, \ldots, b_m]^T$ es el vector de términos independientes
- J es el conjunto de índices de variables que deben ser enteras

2.2. Métodos Heurísticos y Metaheurísticos

Los métodos heurísticos y metaheurísticos representan un enfoque alternativo que sacrifica garantías de optimalidad a cambio de mayor eficiencia computacional. Las heurísticas construyen soluciones mediante reglas específicas del problema o realizan búsquedas locales en el espacio de soluciones. Por otro lado, las metaheurísticas son estrategias que guían el proceso de búsqueda, independientemente del problema específico [10]. Su principal ventaja radica en la capacidad de encontrar soluciones de buena calidad (con respecto al óptimo conocido o al mejor valor alcanzado en experimentos) en tiempos computacionales razonables para problemas complejos donde los métodos exactos resultan inviables.

2.2.1. Greedy

Los algoritmos Greedy son métodos heurísticos que constituyen un paradigma de diseño algorítmico que resuelve problemas de optimización mediante una secuencia de decisiones localmente óptimas en cada etapa, sin retroceder sobre elecciones previas [11]. La esencia de este enfoque radica en su propiedad de optimalidad local: en cada iteración, selecciona la alternativa que maximiza o minimiza inmediatamente la función objetivo, bajo el supuesto de que esta estrategia acumulativa conducirá a una solución globalmente óptima.

2.2.2. Biased random key genetic algorithm (BRKGA)

BRKGA es una metaheurística evolutiva que, en lugar de trabajar directamente con las soluciones, transforma cada opción posible en una combinación de números entre [0,1] y con estos valores utiliza estrategias de selección y cruce sesgadas hacia soluciones élite, manteniendo diversidad mediante la inclusión de individuos totalmente aleatorios denominados mutantes [12]. El Algoritmo 1 presenta el pseudocódigo generalizado de esta metaheurística junto con la tabla 2.1 definición de los parámetros.

Algorithm 1 BRKGA pseudo-code

Require: a problem instance
Ensure: values for parameters -p, -pe, -pm, -rhoe
1: $P \leftarrow$ GenerateInitialPopulation(p)
2: Decode(P)
3: Evaluate(P) { evaluates the fitness of the solution}
4: **while** computation time limit not reached **do**
5: $P_e \leftarrow$ EliteSolutions(P, pe)
6: $P_m \leftarrow$ Mutants(P, pm)
7: $P_c \leftarrow$ Crossover(P, pe, rhoe)
8: Decode($P_m \cup P_c$)
9: Evaluate($P_m \cup P_c$)
10: $P \leftarrow P_e \cup P_m \cup P_c$
11: **end while**
12: **return** best solution in P

Tabla 2.1: Parámetros del algoritmo BRKGA

Parámetro	**Descripción**
-p	Tamaño de la población
-pe	Proporción de la población élite
-pm	Proporción de la población mutante
-rhoe	Probabilidad de herencia élite

2.2.3. Greedy Randomized Adaptive Search Procedure (GRASP)

GRASP es una metaheurística iterativa que combina una fase de construcción greedy aleatorizada, donde se generan soluciones iniciales mediante un proceso greedy con cierto grado de aleatoriedad y una fase de búsqueda local para refinar dichas soluciones [13] explicada en el Algoritmo 2 y la tabla 2.2 con la definición de los parámetros.

Algorithm 2 GRASP pseudo-code

Require: Data D, parameters: -max_iter, -d,

Ensure: Best solution S^{bsf}

```
S^bsf ← ∅
for i ← 1 to max_iter do
    S ← construct_solution(D,d)
    S ← local_search(S)
    if |S| > |S^bsf| then
        S^bsf ← S
    end if
end for
return S^bsf
```

Tabla 2.2: Parámetros del algoritmo GRASP

Parámetro	**Descripción**
-max_iter	Número máximo de iteraciones
-d	Tasa de determinismo [0,1] (0=aleatorio, 1=greedy puro)

2.2.4. Ant Colony Optimization (ACO)

ACO es una metaheurística inspirada en el comportamiento de hormigas reales, utiliza agentes artificiales que construyen soluciones mediante la exploración probabilística del espacio de búsqueda, guiados por feromonas (indicadores de calidad de solución). La ventaja de este enfoque es que ajusta dinámicamente la exploración mediante el rastro de feromonas y combina explotación con exploración. Suele tener una convergencia lenta que requiere múltiples iteraciones para estabilizar las feromonas [14]. A continuación se presenta el Algoritmo 3 en forma de pseudocódigo y la tabla 2.3 con la definición de los parámetros.

Algorithm 3 ACO

Require: Data D, parameters: -n_ants,-d , -l_rate

Ensure: Best solution S^{bsf}

```
Initialize pheromone matrix τ in D
S^bsf ← ∅
while time limit not reached do
    for k ← 1 to n_hormigas do
        S_k ← construct_solution(D, τ,d)
        if |S_k| > |S^bsf| then
            S^bsf ← S_k
        end if
    end for
    τ ← update_pheromones(τ, S^bsf,-l_rate)
end while
return S^bsf
```

Tabla 2.3: Parámetros del algoritmo ACO

Parámetro	**Descripción**
-n_ants	Número de hormigas por iteración
-d	nivel de determinismo
-l_rate	Tasa de evaporación de feromonas

2.3. Métodos Híbridos

Los métodos híbridos surgen como una estrategia que combina las fortalezas de los enfoques exactos y heurísticos o metaheurísticos [10]. Las estrategias de hibridación pueden incluir el uso de métodos exactos para resolver subproblemas dentro de una estructura metaheurística, la generación de soluciones iniciales subóptimas mediante heurísticas para alimentar algoritmos exactos, o la combinación paralela de diferentes enfoques. Los métodos híbridos son particularmente valiosos en problemas de gran escala donde se requiere un equilibrio entre calidad de solución y tiempo de cómputo.

2.3.1. Construct, Merge, Solve & Adapt (CMSA)

CMSA es un algoritmo híbrido que combina metaheurísticas con técnicas exactas para resolver problemas de optimización combinatoria. La idea principal de CMSA propuesta en [15]

consiste en iterativamente construir soluciones parciales, fusionarlas en un subproblema reducido, resolver este subproblema utilizando un método exacto como CPLEX (solver de ILP), y adaptar el subproblema basándose en las soluciones encontradas. Este enfoque permite aprovechar las fortalezas tanto de las metaheurísticas para explorar el espacio de búsqueda como de los métodos exactos para refinar soluciones localmente. A continuación se presenta el Algoritmo 4 en forma de pseudocódigo y la tabla 2.4 con la definición de los parámetros.

Algorithm 4 CMSA pseudo-code

Require: Data D, params: -n_a, -max_age, -cplex_time
Ensure: Best solution S^{bsf}

$S^{bsf} \leftarrow \emptyset$, $C' \leftarrow \emptyset$
for $d_i \in D$ **do**
  $age[d_i] \leftarrow 0$
end for
while time limit not reached **do**
  $T \leftarrow$ construct(n_a, D)
  $C' \leftarrow$ merge(C', T)
  $S^{cplex} \leftarrow$ solve(C',cplex_time)
  if $|S^{cplex}| > |S^{bsf}|$ **then**
    $S^{bsf} \leftarrow S^{cplex}$
  end if
  $C' \leftarrow$ adapt(C', S^{cplex},max_age)
end while
return S^{bsf}

Notación

- S^{bsf}: Mejor solución encontrada (*Best-So-Far*)
- C': Conjunto de componentes de la subinstancia
- T: Conjunto de soluciones temporales construidas en cada iteración
- S^{cplex}: Solución óptima local obtenida por CPLEX

2.3.2. Learned CMSA

Como se define en [16] LCMSA es una extensión de CMSA que incorpora un mecanismo de aprendizaje para mejorar la construcción de soluciones. Inspirado en algoritmos evolutivos,

Tabla 2.4: Parámetros del algoritmo CMSA

Parámetro	Descripción
-n_a	Número de soluciones construidas por iteración
-max_age	Edad máxima antes de eliminar un componente
-cplex_time	Tiempo límite para el solver ILP (en segundos)

LCMSA utiliza una población de soluciones que evoluciona mediante operadores de recombinación y mutación. Esto permite que el algoritmo aprenda qué componentes son más prometedores a lo largo del tiempo. A continuación se presenta el Algoritmo 5 en forma de pseudocódigo y la tabla 2.5 con la definición de los parámetros.

Algorithm 5 LCMSA with BRKGA

Require: Data D,
CMSA params (-n_a, -max_age, -cplex_time),
BRKGA params (-p, -pe, -pm, -rhoe)
Ensure: Best solution S^{bsf}
$S^{bsf} \leftarrow \emptyset$, $C' \leftarrow \emptyset$
$P \leftarrow$ BRKGA.init_population(p, D)
while time limit not reached **do**
  $P \leftarrow$ BRKGA.evolve(p, pe, pm, rhoe)
  $T \leftarrow$ select_elite(P,n_a)
  $C' \leftarrow$ merge(C', T)
  $S^{ILP} \leftarrow$ solve(C',cplex_time)
  if $|S^{ILP}| > |S^{bsf}|$ **then**
    $S^{bsf} \leftarrow S^{ILP}$
  end if
  $C' \leftarrow$ adapt(C', S^{ILP},max_age)
  $P \leftarrow$ inject(P, S^{bsf})
end while
return S^{bsf}

2.3.3. BARRAKUDA

En [17] se define BARRAKUDA como un algoritmo evolutivo híbrido basado en la estructura de BRKGA e integra un componente exacto mediante la resolución de subinstancias con un solver ILP (como CPLEX). En cada iteración, BARRAKUDA selecciona un subconjunto de soluciones de la población BRKGA, genera una subinstancia combinando sus componentes y aplica el solver exacto para intentar mejorar las soluciones. Luego, la solución de alta calidad

encontrada por el solver es integrada nuevamente en la población. A continuación se presenta el Algoritmo 6 en forma de pseudocódigo y la tabla 2.5 con la definición de los parámetros.

Algorithm 6 BARRAKUDA

Require: Data D, params: $psize$, pe, pm, $probelite$, na, $exmode$, t_{ILP}
Ensure: Best solution S^{bsf}

```
P ← GenerateInitialPopulation(psize, D)
Evaluate(P)
S^bsf ← best solution in P
while time limit not reached do
    P_e ← EliteSolutions(P, pe)
    P_m ← Mutants(P, pm)
    P_c ← Crossover(P, P_e, probelite)
    Evaluate(P_m ∪ P_c)
    P ← P_e ∪ P_m ∪ P_c
    P_ILP ← ChooseSolutions(P, exmode, na)
    C' ← ⋃_{S∈P_ILP} S
    S^cplex ← Solve(C', t_ILP)
    if |S^cplex| > |S^bsf| then
        S^bsf ← S^cplex
    end if
    P ← IntegrateSolverSolution(P, S^cplex)
end while
return S^bsf
```

Tabla 2.5: Parámetros del algoritmo BARRAKUDA y LCMSA

Parámetro	**Descripción**
$psize$	Tamaño de la población
pe	Porcentaje de individuos de élite
pm	Porcentaje de mutantes por iteración
$probelite$	Probabilidad de herencia de un gen del padre élite
na	Número de soluciones seleccionadas para subinstancia ILP
$exmode$	Modo de extracción: élite o aleatorio
t_{ILP}	Tiempo límite para el solver ILP (en segundos)

2.4. Integración de Metaheurísticas y Modelos de Lenguaje para Optimización Combinatoria

Chacón et al. [2] proponen un enfoque que integra metaheurísticas con grandes modelos de lenguaje (LLMs) para abordar problemas de optimización combinatoria. El método híbrido desarrollado utiliza los LLMs como herramientas de reconocimiento de patrones. Pero en lugar de generar soluciones completas, los LLMs analizan métricas clave de las instancias del problema mediante prompts estructurados que incluyen información relevante de la instancia.

El proceso genera dos tipos de parámetros: los valores alpha, que ponderan la importancia de cada métrica, y los valores beta, que representan los rangos ideales para cada característica. Estos parámetros alimentan un modelo lineal que calcula las probabilidades de pertenencia a la solución óptima de cada elemento candidato, sesgando así el proceso de búsqueda de la metaheurística (véase figura 2.1).

La implementación se realizó sobre un algoritmo BRKGA aplicado al problema de maximización de influencia en redes sociales (k-d DSP). Los resultados experimentales demostraron mejoras en la calidad de las soluciones obtenidas .

El LLM genera un conjunto de salida con un valor α y β para cada métrica, como se muestra en la ecuación (2.1):

$$\text{Output} = \{\alpha_1, ..., \alpha_5, \beta_1, ..., \beta_5\} \tag{2.1}$$

Componentes de la Fórmula

- **Valores Alpha (α_i)**: Pesos normalizados que reflejan la importancia relativa de cada métrica. La ecuación (2.2) detalla la restricción de acuerdo a los 5 valores alpha que se obtienen:

$$\sum_{i=1}^{5} \alpha_i = 1 \quad \text{con} \quad 0 < \alpha_i < 1 \tag{2.2}$$

  donde α_i corresponde al peso de la i-ésima métrica en el orden predefinido.

- **Valores Beta (β_i)**: Parámetros de ajuste independientes que indican el valor óptimo esperado para cada métrica normalizada ($0 < \beta_i < 1$).

- **Función de Probabilidad ($p_{\mathbf{LLM}}$)**: Combina ambos parámetros para calcular la probabilidad de selección de un elemento candidato v_j que se calcula mediante la ecuación

(2.3):

$$p_{\text{LLM}}(v_j) := \sigma\left(\sum_{i=1}^{5} \alpha_i \cdot \left(1 - |\beta_i - m_{j,i}^{\text{eval}}|\right)\right) \tag{2.3}$$

donde:

- $m_{j,i}^{\text{eval}}$: Valor normalizado de la i-ésima métrica para el nodo v_j
- σ: Función sigmoide que mapea el resultado al intervalo $[0, 1]$ incorporando no linealidad en las probabilidades.
- El término $|\beta_i - m_{j,i}^{\text{eval}}|$ mide la desviación del valor ideal

Esta formulación permite al LLM cuantificar la importancia relativa de las métricas, permitiendo así orientar estratégicamente la búsqueda del algoritmo BRKGA. Para automatizar el proceso se utiliza la herramienta Optipattern, que automatiza la generación de Prompts, el procesamiento de métricas y el cálculo de probabilidades. El código fuente de Optipattern está públicamente accesible en el repositorio: `https://github.com/camilochs/optipattern`.

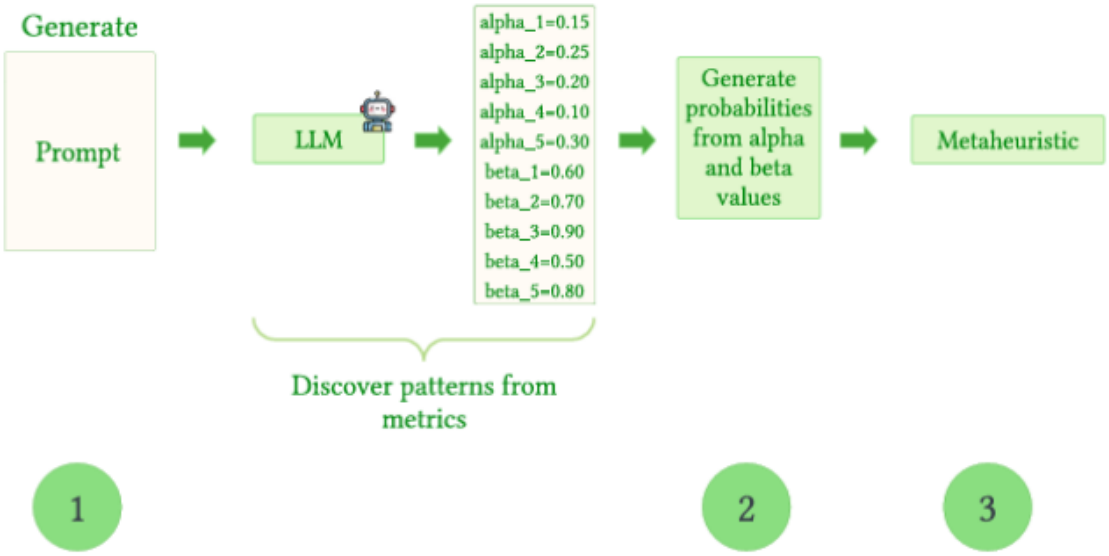


Figura 2.1: Resumen visual del proceso integración de metaheurísticas para analizar instancias del problema y descubrir patrones ocultos que se convierten luego en información útil que guía a la metaheurística [2].

Capítulo 3. Estado del arte de soluciones

El LRS ha sido ampliamente estudiado, donde se han propuesto diversas técnicas que buscan equilibrar la exactitud de las soluciones con el tiempo de cálculo, dada su complejidad algorítmica. En esta sección, se analizan las estrategias actuales y los avances más relevantes del estado del arte para abordar el LRS, analizando sus fundamentos, ventajas prácticas y limitaciones. Según el marco teórico presentado en la Sección 2, los métodos propuestos y utilizados actualmente para la resolución del LRS pueden clasificarse en métodos exactos y aproximados.

3.1. Métodos Exactos

Donde se destaca el uso de algoritmos como programación dinámica (DP) y programación lineal entera (ILP).

3.1.1. Programación Dinámica

Este enfoque propuesto por Schrinner et al. [7] para el LRS construye una tabla bidimensional $D[i, F]$, donde i representa el índice de la racha, F el conjunto de caracteres utilizados y P representa el estado parcial de la subsecuencia construida, indicando qué caracteres han sido asignados. La recursión (Ecuación (3.1) y (3.2)) maximiza la longitud de la subsecuencia válida considerando las últimas ocurrencias de cada caracter para evitar violaciones de las restricciones de rachas únicas. Este enfoque garantiza optimalidad, demostrando en los resultados que encuentra la solución óptima para instancias con alfabetos pequeños ($|\Sigma| \leq 20$). Sin embargo, el espacio de estados crece exponencialmente con el tamaño del alfabeto, haciéndolo inviable para $|\Sigma| > 24$. Se define la recursión DP mediante las ecuaciones:

$\sigma(r_i) \in \Sigma$ y \$ representa el inicio de la secuencia.

$$D[0, F] = \begin{cases} 0 & \forall F \subseteq \Sigma \\ -\infty & \text{si } F = \emptyset \end{cases} \tag{3.1}$$

Para $i > 0$ y $F \subseteq \Sigma$:

$$D[i,F] = \begin{cases} \max\limits_{\sigma \in \Sigma \cup \{\$\}} \begin{cases} D[P_\sigma(i), F] + l(r_i) & \text{si } \sigma = \sigma(r_i) \\ D[P_\sigma(i), F \setminus \{\sigma(r_i)\}] + l(r_i) & \text{si } \sigma(r_i) \in F \\ -\infty & \text{en otro caso} \end{cases} \end{cases} \tag{3.2}$$

3.1.2. Programación Lineal entera

Schrinner et al. [7] también presenta el enfoque ILP que modela el problema LRS como un sistema de maximización lineal donde variables binarias $x_i \in \{0, 1\}$ representan la inclusión ($x_i = 1$) o exclusión ($x_i = 0$) de la racha r_i en la solución. La función objetivo (ecuación (3.3)) maximiza la suma de longitudes de las rachas seleccionadas, mientras que las restricciones (ecuación (3.4)) garantizan la unicidad de las rachas por caracter. Este enfoque funciona eficientemente con $|\Sigma| \geq 30$, siendo adecuado para aplicaciones genómicas reales. Sin embargo, el tiempo de ejecución es exponencial respecto al número de rachas. Por lo tanto, este método es adecuado para problemas con alfabetos grandes y cadenas moderadamente cortas, complementando así las fortalezas de otros enfoques como la programación dinámica.

Sea n el número de rachas en la cadena S, y $\sigma(r_i) \in \Sigma$. Se definen las variables binarias $x_1, \ldots, x_n$ donde:

$$x_i = \begin{cases} 1 & \text{si la racha } r_i \text{ está en la subsecuencia óptima} \\ 0 & \text{en caso contrario} \end{cases}$$

La función objetivo maximiza el length o largo total de la subsecuencia:

$$\text{máx} \sum_{i=1}^{n} x_i l(r_i) \tag{3.3}$$

donde $l(r_i)$ representa el length de la racha r_i.

Para cada par de rachas r_i, r_j con $i < j$ y $\sigma(r_i) = \sigma(r_j)$ (mismo caracter), añadimos restricciones que garantizan que, si ambas rachas son seleccionadas, ningúna racha intermedia con caracter diferente puede ser incluido.

$$(j-i)x_i + \left(\sum_{\substack{i<l<j \\ \sigma(r_l)=\sigma(r_i)}} x_l \right) + (j-i)x_j \leq 2(j-i) \quad \text{para todo } 1 \leq i < j \leq n \tag{3.4}$$

3.1.3. Resultados

Los experimentos realizados evaluaron el rendimiento de los algoritmos DP e ILP en dos escenarios: escalamiento por longitud de cadena y por tamaño de alfabeto. Para esto, se generaron instancias aleatorias. Las pruebas se ejecutaron en un sistema AMD Ryzen 7 2700X con 32 GB de RAM bajo Ubuntu 20.04, implementando los algoritmos en Python 3.7.6 con PuLP 1.6.8 y Snakemake. Los resultados (Figura 3.1) demuestran claramente las limitaciones de los enfoques exactos para resolver LRS. Mientras que el ILP se ve severamente afectado por incrementos en la longitud de la cadena, el DP presenta el patrón opuesto, volviéndose computacionalmente inviable para alfabetos grandes debido a su consumo exponencial de memoria (hasta 9.2 GB para $|\Sigma| = 22$).

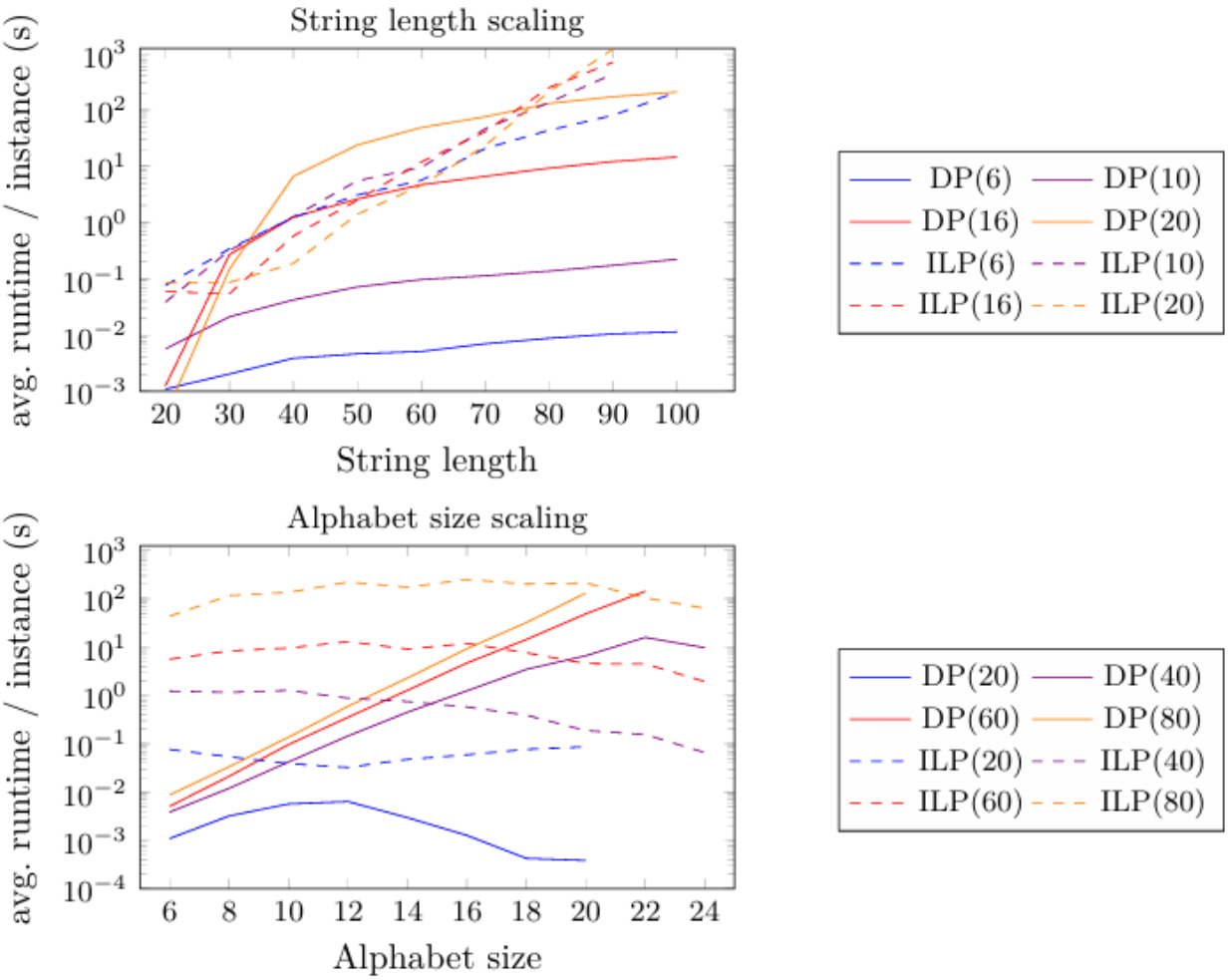


Figura 3.1: Escalamiento del tiempo de ejecución (en escala logarítmica) en función de la longitud de la cadena (gráfico superior) y del tamaño del alfabeto (gráfico inferior). Las curvas representan los algoritmos evaluados (DP e ILP), indicando entre paréntesis los parámetros utilizados: en el gráfico superior se muestra el tamaño del alfabeto y en el gráfico inferior la longitud de la cadena.

3.2. Métodos Aproximados

En el contexto del LRS, cobran especial relevancia los métodos aproximados, entre los cuales destacan algoritmos como ACO y BRKGA.

3.2.1. Ant Colony Optimization

El algoritmo ACO fue implementado por Blum et al. [3] para el LRS, donde su mecanismo basado en feromonas permite guiar inteligentemente la selección de rachas. Este enfoque no solo optimiza el ensamblaje mediante el rastro de feromonas, sino que incorpora, además, una heurística que prioriza las rachas según su length.

El algoritmo utiliza un conjunto T de valores de feromona, donde cada racha $r_i \in R$ (siendo $|R|$ el número total de rachas en la instancia) tiene asociado un valor $\tau_i \in [0, 1]$. ACO genera permutación mediante un proceso probabilístico híbrido: con probabilidad d_{rate} selecciona de manera greedy la racha con máximo valor $\tau_i \times l(r_i)$ (feromona $\times$ length de la racha i), y con probabilidad $1 - d_{rate}$ aplica selección por ruleta basada en estos mismos valores. Este diseño permite combinar efectivamente la explotación de componentes prometedores con la exploración de nuevas regiones del espacio de búsqueda.

3.2.2. Biased Random Key Genetic Algorithm

Blum et al. [3] también propone una variante de Algoritmos Genéticos BRKGA donde los individuos se representan como vectores de claves aleatorias en [0,1], donde cada componente del vector corresponde a una racha en la cadena de entrada. Estos valores, generados aleatoriamente, determinan el orden de procesamiento durante la fase de decodificación. En este proceso, las rachas se ordenan de forma decreciente según sus valores asociados, priorizando aquellos con valores más altos. Se procesan las rachas en ese orden, añadiéndolos a la solución solo si cumplen las condiciones respectivas del LRS.

El proceso evolutivo del BRKGA combina exploración y explotación que privilegia soluciones élite. Comienza con una población inicial aleatoria de vectores de claves. En cada iteración, selecciona un conjunto élite, genera nuevos individuos mediante cruza sesgada e introduce mutantes aleatorios para mantener diversidad. La nueva población combina élites, mutantes y descendientes, equilibrando explotación y exploración tal como se define en el Algoritmo 1

3.2.3. Resultados

El estudio experimental realizado comparó estas técnicas utilizando diversas variables, donde length representa la longitud de la secuencia de entrada y $|\Sigma|$ denota el tamaño del alfabeto. La investigación incorporó además los resultados obtenidos mediante CPLEX, un solver de ILP. Este trabajo estableció el algoritmo BRKGA como nuevo estado del arte para el LRS, como se muestra en la Figura 3.2. Como contribución adicional, se introdujo un dataset de referencia que fue adoptado en este estudio para garantizar comparabilidad.

Las métricas para evaluar y comparar las técnicas fueron:

- Average: Promedio del último puntaje obtenido para las 30 instancias idénticamente distribuidas en cada celda.
- Time: Tiempo promedio de ejecución hasta el último registro.
- Gap: Diferencia porcentual relativa al óptimo conocido.

Figura 3.2: Comparación de desempeño entre BRKGA, ACO y CPLEX en el dataset de referencia. Los valores representan promedios sobre 30 instancias por combinación length-|Σ|. Resultados obtenidos por Blum et al. [3]. En negrita mejores resultados por fila

		CPLEX			ACO		BRKGA	
Length	|Σ|	average	time	gap	average	time	average	time
100	2	**59.03**	0.11	8.55	**59.03**	0.01	**59.03**	0.00
	4	40.97	1.00	44.16	40.70	1.35	**41.07**	0.05
	8	33.33	4.57	64.98	30.57	3.04	**33.67**	0.52
	16	**34.20**	6.46	57.59	27.63	2.62	33.77	2.45
	32	**42.03**	4.76	33.13	28.73	3.30	39.53	4.65
200	2	**115.17**	1.19	11.22	**115.17**	0.14	**115.17**	0.00
	4	70.63	8.22	77.08	71.77	4.56	**72.77**	0.12
	8	51.83	14.12	135.37	49.27	8.93	**55.03**	2.52
	16	46.00	12.59	165.92	35.90	6.48	**49.97**	6.06
	32	**53.90**	15.96	123.77	27.87	7.16	52.90	11.25
300	2	**165.47**	8.37	12.54	**165.47**	0.10	**165.47**	0.00
	4	95.77	9.82	103.99	101.53	5.54	**102.63**	0.40
	8	64.43	13.85	204.36	66.53	15.56	**74.63**	3.72
	16	55.07	13.99	253.98	50.00	11.56	**65.23**	10.00
	32	61.03	21.54	211.73	45.33	9.73	**63.73**	19.81
500	2	269.27	14.76	21.87	**271.67**	0.42	**271.67**	0.02
	4	146.73	27.21	125.02	161.00	7.04	**162.73**	1.44
	8	84.87	28.42	292.47	99.97	21.77	**109.70**	7.34
	16	61.63	25.91	441.28	66.93	21.72	**88.63**	23.13
	32	66.03	32.57	401.36	57.33	17.16	**81.43**	31.73
1000	2	458.90	43.60	>10000.00	**530.23**	0.85	**530.23**	0.03
	4	196.57	50.00	>10000.00	297.73	35.99	**300.83**	9.97
	8	85.73	40.95	>10000.00	175.73	56.10	**189.93**	20.81
	16	49.73	28.08	>10000.00	105.63	41.94	**139.60**	47.00
	32	42.67	23.21	>10000.00	67.90	41.13	**119.20**	71.98
2000	2	0.00	200.00	>10000.00	**1041.73**	16.07	**1041.73**	1.53
	4	0.00	198.53	>10000.00	567.57	74.70	**572.10**	5.88
	8	0.00	178.90	>10000.00	317.13	99.79	**342.13**	55.61
	16	0.00	106.48	>10000.00	180.77	103.18	**230.80**	107.27
	32	0.00	52.91	>10000.00	115.43	83.32	**177.47**	154.92
5000	2	0.00	500.00	>10000.00	2567.37	41.68	**2567.47**	1.14
	4	0.00	500.00	>10000.00	1350.33	274.30	**1360.47**	24.65
	8	0.00	500.00	>10000.00	720.67	266.67	**763.17**	221.00
	16	0.00	500.00	>10000.00	394.30	301.26	**476.20**	343.23
	32	0.00	500.00	>10000.00	220.67	223.14	**322.00**	419.01

Nota: El tiempo de ejecución máximo para cada algoritmo fue *length*/10 segundos por instancia.

3.2.4. Evaluación de alternativas

El análisis de las soluciones existentes revela información importante que orienta esta investigación. Los métodos exactos demuestran ser poco adecuados para este tipo de problemas debido a la complejidad algorítmica del modelo y a las características particulares de las secuencias analizadas. Concretamente, su rendimiento tiene una baja notable al trabajar con alfabetos de tamaño $|\Sigma| \geq 22$ y secuencias de longitud $length \geq 100$.

En contraste, las metaheurísticas como BRKGA y ACO presentan un mejor comportamiento en instancias grandes, aunque con ciertas limitaciones. Este trabajo se centra particularmente en metaheurísticas híbridas, ya que demuestran un rendimiento considerable y las posibilidades de mejora son amplias.

Capítulo 4. Solución Propuesta

Ante los desafíos planteados en la optimización del LRS, este capítulo presenta una solución metodológica que combina distintas técnicas. La propuesta se fundamenta en cuatro enfoques diseñados para abordar las distintas dimensiones del problema: (1) el ajuste adaptativo de hiperparámetros que considera las características de las instancias; (2) el rediseño de heurísticas; (3) el desarrollo de algoritmos híbridos que combinan métodos exactos con aproximaciones metaheurísticas; y (4) la integración de modelos de lenguaje como mecanismo de sesgo. En este capítulo se realiza el marco metodológico de los experimentos para llevar a cabo dichos enfoques.

4.1. Metodología

Para evaluar el desempeño de las estrategias propuestas, se siguió una metodología basada en cuatro componentes principales. Primero, se desarrollaron algoritmos con comportamiento anytime behavior [18] que, dada una instancia de entrada y los parámetros requeridos (dependiendo del algoritmo) registran tanto la mejor solución encontrada como el momento exacto de su hallazgo. El tiempo de ejecución se definió proporcionalmente al tamaño de la instancia dividido por 5; por ejemplo, si length es 200, el tiempo de ejecución será de 40 segundos. Esto permite una exploración más exhaustiva en instancias complejas que son las que poseen mayor oportunidad de mejora.

Segundo, para el ajuste de hiperparámetros se utilizó irace[1], un paquete de R que implementa el método Iterated Race, una generalización de Iterated F-Race para la configuración automática de algoritmos de optimización [19]. El proceso empleó 35 instancias generadas aleatoriamente, representando todas las combinaciones length-$|\Sigma|$ del dataset. En los casos de ajuste segmentado (big length, short length, big $|\Sigma|$, short $|\Sigma|$), cada configuración usó exclusivamente las instancias correspondientes a su categoría.

El ajuste de hiperparámetros se realiza en Luthier, un clúster de computación de alto rendimiento instalado en la Dirección de Tecnologías de la Información (DTI) de la Universidad de Concepción en marzo de 2019. Está compuesto por 31 servidores idénticos, cada uno equipado

con procesadores Intel Xeon E3-1270 v6 (3.8 GHz), 64 GB de RAM y sin GPUs dedicadas. El sistema incluye componentes de comunicación y administración (KVM), un servidor NAS básico para almacenamiento y sistemas UPS para respaldo de energía. Diseñado para tareas de procesamiento paralelo, Luthier ofrece capacidad de cálculo escalable para proyectos de investigación que requieren alto rendimiento, aunque su arquitectura se centra en CPU sin aceleración gráfica.

Cada ejecución de irace se limitó a 3000 experimentos con un tiempo máximo de length/5 segundos por instancia, como se estableció previamente. Debido al elevado costo computacional, todos los experimentos se ejecutaron en el clúster Luthier de la Universidad de Concepción.

Tercero, la evaluación de un algoritmo se realiza ejecutando el algoritmo con la configuración de hiperparámetros obtenida en el dataset completo, y se calcula el promedio cada 30 instancias que representan una combinación length-$|\Sigma|$, lo que facilita el proceso comparativo visualmente.

Para comparar el rendimiento de algoritmos en caso de una potencial mejora se utiliza scmamp [2] (Statistical Comparison of Multiple Algorithms in Multiple Problems), un paquete de R diseñado para simplificar el análisis de los resultados obtenidos. Aunque el paquete scmamp se centra en la comparación de múltiples algoritmos, también puede adaptarse a otros escenarios, permitiendo generar gráficos de diferencias críticas (Critical Difference, CD), los cuales resultan especialmente útiles para visualizar las comparaciones de los algoritmos, como se menciona en [20].

El resumen metodológico se presenta en la Figura 4.1 como ayuda visual para comprender a grandes rasgos el proceso con cada algoritmo.

[1] https://mlopez-ibanez.github.io/irace/
[2] https://github.com/b0rxa/scmamp

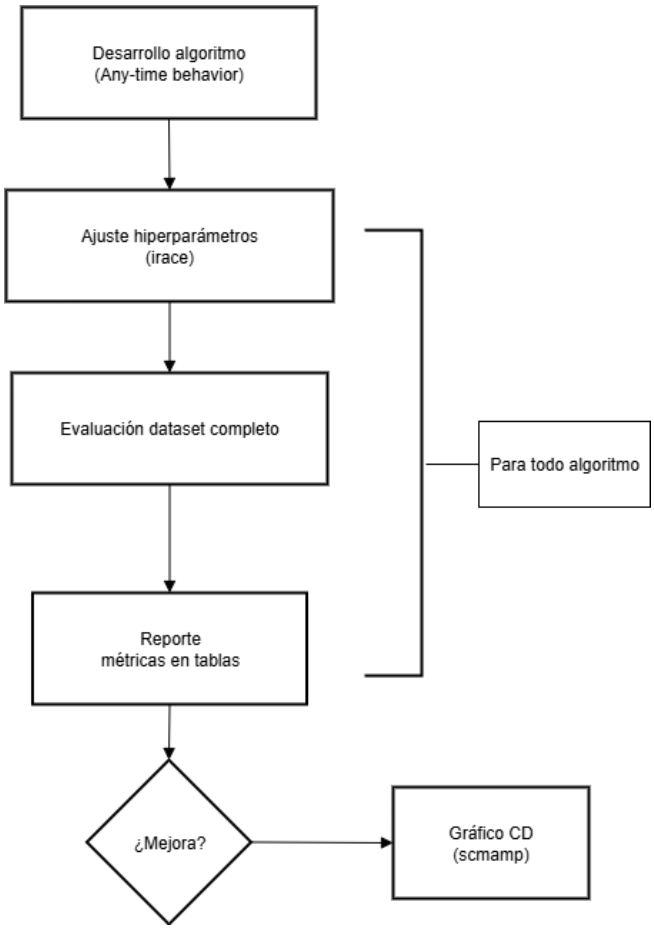


Figura 4.1: Diagrama de flujo metodología.

4.2. Diseño de experimentos

A continuación se presentan los objetivos y principales características de los experimentos a realizar teniendo en cuenta las características principales del dataset a utilizar.

El dataset consta de 1,050 instancias sintéticas con las siguientes propiedades:

- **length**: Tamaño de la secuencia genómica $\{100, 200, 300, 1{,}000, 2{,}000, 5{,}000\}$.
- **Sigma ($|\Sigma|$)**: Cardinalidad del alfabeto (caracteres distintos), con valores $\{2, 4, 8, 16, 32\}$.

Para evaluar el comportamiento de los algoritmos, el dataset incluye 30 instancias por cada combinación posible de length y $|\Sigma|$ (por ejemplo: 30 instancias con length=100 y $|\Sigma|$=2, 30 con length=100 y $|\Sigma|$=4, y así sucesivamente).

4.2.1. Hiperparámetros segmentados

Este experimento busca comparar el rendimiento de los algoritmos BRKGA y LCMSA mediante un enfoque de ajuste de hiperparámetros diferenciado por características estructurales de las instancias. La segmentación se realiza considerando dos dimensiones clave:

Por length: Short length (100, 200, 300, 500) Big length (1000, 2000, 5000)
Por $|\Sigma|$: Short $|\Sigma|$ (2, 4, 6, 8) Big $|\Sigma|$ (16, 32)

La división del dataset en subconjuntos por length y $|\Sigma|$ responde a la necesidad de evaluar cómo los algoritmos se comportan bajo diferentes condiciones de complejidad. Las instancias de short length permiten analizar el rendimiento en problemas de escala reducida donde los métodos convencionales pueden ser efectivos, mientras que las de big length ante problemas de mayor dimensión y mayor complejidad. Por otro lado, la segmentación por $|\Sigma|$ posibilita estudiar el impacto de la variabilidad, considerando que valores altos de $|\Sigma|$ pueden generar espacios de búsqueda más complejos y dispersos. Este enfoque permite identificar configuraciones óptimas de hiperparámetros adaptadas a las características específicas de cada tipo de instancia.

4.2.2. Nueva heurística

Este experimento tiene como objetivo mejorar el rendimiento del algoritmo ACO mediante el desarrollo de una nueva versión de su heurística. Primero se implementó la heurística original del ACO para establecer una línea base de comparación. Luego, se diseñó una versión mejorada de esta heurística, incorporando nuevos criterios de selección. Ambas versiones se integraron en los algoritmos ACO y GRASP para evaluar su comportamiento en diferentes contextos metaheurísticos en el dataset completo.

4.2.3. Hibridación de Metaheurísticas con Métodos Exactos

Este experimento tiene como objetivo evaluar el desempeño de diferentes metaheurísticas, considerando que BRKGA es un algoritmo genético robusto. Se buscó desarrollar y analizar algoritmos metaheurísticos alternativos como CMSA y BARRAKUDA, los cuales combinan estrategias de exploración y explotación. Para ello, se siguió la metodología propuesta anteriormente, desarrollo del algoritmo, ajuste de hiperparámetros con irace y evaluación en el dataset.

4.2.4. Integración BRKGA+LLMs

Este experimento evalúa el uso de LLMs para mejorar el algoritmo BRKGA. Primero, se seleccionaron métricas clave del problema para calcular probabilidades de selección de elementos

a incorporar en la solución (rachas), desarrollando una adaptación de la herramienta Optipattern para el LRS en específico. Se comenzó probando con LLMs gratuitos en un conjunto representativo de instancias, analizando su impacto en la calidad de las soluciones.

Cuando los resultados iniciales muestran mejoras significativas, se procede a utilizar LLMs de paga para obtener parámetros alpha y beta más precisos. Cada versión del algoritmo (con diferentes LLMs) sigue la metodología establecida: ajuste de hiperparámetros con irace y evaluación en el dataset completo para posterior comparación. Adicionalmente, para comprobar si los LLMs realmente mejoran el algoritmo, se propone comparar su desempeño contra versiones que usan probabilidades aleatorias. Implementando dos pruebas: una con valores fijos y otra con valores que cambian en cada ejecución, mencionadas como probabilidades aleatorias estáticas o dinámicas posteriormente.

4.2.5. Entorno de pruebas

Hardware

Los experimentos se realizaron en el clúster Luthier de la Universidad de Concepción, con las siguientes configuraciones y restricciones para los nodos:

- **Nodos estándar**:
  - Máxima memoria RAM asignada: 6 GB de RAM por nodo
  - Sistema de colas SLURM para gestión de jobs
- **Nodos para algoritmo LCMSA**:
  - Máxima memoria RAM asignada: 30 GB de RAM por nodo (requerido para secuencias con $length \geq 5000$)

Software

Y para el desarrollo, experimentación y evaluación estadística se ocupó el siguiente stack tecnológico:

- Lenguajes: C++, python
- Tuning: irace
- Evaluación estadística: scmamp

Capítulo 5. Desarrollo

Dado que el espacio de mejora se manifiesta principalmente en instancias complejas, se incrementó el tiempo de ejecución por instancia a length/5 segundos, en lugar de los length/10 segundos empleados en estudios anteriores. Este ajuste, aplicado tanto al BRKGA estado del arte como a las implementaciones de esta investigación, permite una exploración más exhaustiva del espacio de búsqueda y una evaluación más robusta del rendimiento algorítmico. La configuración detallada del BRKGA con los nuevos tiempos se presenta en la Tabla 5.1, mientras que los rangos de los parámetros se explican en la Tabla 5.2. Para garantizar la comparabilidad con los resultados previos del estado del arte, se reconfiguraron los hiperparámetros del algoritmo, cuyo código fuente, modificable, está disponible públicamente.

Tabla 5.1: Configuración de hiperparámetros del BRKGA estado del arte para tiempos de ejecución extendidos

Algoritmo	**-p**	**-pe**	**-pm**	**-rhoe**
BRKGA	16	0.25	0.17	0.56

Tabla 5.2: Rango para ajuste de hiperparámetros BRKGA

Parámetro	**Rango**	**Tipo**
-p	(10, 400)	entero
-pe	(0.1, 0.25)	real
-pm	(0.1, 0.3)	real
-rhoe	(0.51, 0.8)	real

el código fuente de todas las implementaciones desarrolladas se encuentra disponible en el repositorio `github.com/martiniip/LRS`.

5.1. Diseño Heurística

Inspirado en la construcción de soluciones de BRKGA con vectores de claves aleatorias, la heurística greedy actual asigna a cada posición del vector un valor basado en la longitud de cada

racha en la secuencia de entrada. Este enfoque simple prioriza las rachas más largas, asumiendo que contribuyen significativamente a la calidad de la solución como se muestra en el Algoritmo 7.

La función Evaluate(ind) construye la solución y asigna el puntaje, recibiendo un individuo (con su vector de puntajes ind.vec) e incorpora las rachas de mayor puntaje a la solución siempre que se cumplan las restricciones del problema. La representación de la solución se realiza mediante valores binarios (0 o 1) en el vector selected runs, donde 1 indica que la racha está incluida en la solución final.

Algorithm 7 Greedy Heuristic for Individual Generation

Require: Problem instance (s, Σ, R) where R is the list of runs
Ensure: Individual *ind* {Contains vec, selected_runs, and score}
1: Initialize *ind*
2: $ind.vec \leftarrow$ vector of size $|R|$
3: **for** each run $r_i \in R$ at position i **do**
4: $\quad ind.vec[i] \leftarrow \text{length}(r_i)$
5: **end for**
6: Evaluate(*ind*)
7: **return** *ind*

La nueva heurística greedy definida en el Algoritmo 8, por su parte, aprovecha el mismo funcionamiento; sin embargo, el principal cambio radica en la manera de asignar los puntajes en el vector del individuo. Aprovechando las características de las rachas vecinas y considerando un vecindario k, agrega una proporción del largo de los vecinos como puntaje, siempre y cuando sean de la misma letra. Este enfoque simple prioriza las rachas más largas y que más se repiten, asumiendo que contribuyen más a la calidad de la solución.

Los valores de los hiperparámetros k y *pond*, que determinan respectivamente el número de vecinos considerados y su ponderación, fueron optimizados utilizando irace. Tras el ajuste, se seleccionaron $k = 240$ y $pond = 0{,}19$ como configuración óptima para el algoritmo greedy.

Algorithm 8 Greedy Heuristic for Individual Generation

Require: Problem instance (s, Σ, R) where R is the list of runs

Ensure: Individual *ind* {Contains vec, selected_runs, letter and score}

```
Initialize ind
ind.vec ← vector of size |R|
for each run r_i ∈ R at position i do
    ind.vec[i] ← length(r_i)
    for each j from 1 to k do
        if (i − j) > 0 & letter.run[i − j] = letter.run_i then
            ind.vec[i] ← ind.vec[i] + pond ∗ length[i − j]
        end if
        if i + j > R.size & letter.run[i + j] = letter.run_i then
            ind.vec[i] ← ind.vec[i] + pond ∗ length[i + j]
        end if
    end for
end for
Evaluate(ind)
return ind
```

5.2. Adaptación de metaheurísticas para el LRS

Se implementó un algoritmo ACO incorporando dos enfoques heurísticos para la construcción de soluciones. La implementación mantiene la misma estructura del código base de ACO descrita en la sección 3.2.1, añadiendo únicamente la funcionalidad para seleccionar dinámicamente la heurística a utilizar.

Se implementó un algoritmo GRASP para el problema LRS que utiliza una de las dos heurísticas en su fase de construcción. Esta fase genera soluciones iniciales aleatorizadas mediante un proceso que combina componentes greedy y aleatorios.

En la fase de búsqueda local, el algoritmo realiza una búsqueda iterativa basada en intercambios simples entre rachas presentes y ausentes en la solución actual. Cada intercambio se evalúa verificando primero el cumplimiento de las restricciones del problema y luego el impacto en la longitud de la solución representada con 0 y 1. Se presenta el ajuste de hiperparámetros de ambos algoritmos con la incorporación de ambas heurísticas así como el rango definido en las tablas 5.3 y 5.4, respectivamente.

Tabla 5.3: Ajuste de hiperparámetros para GRASP y ACO

Algoritmo	**-max_iter**	**-d**	
GRASP (nuevo)	39	0.91	
GRASP (clásico)	69	0.82	
Algoritmo	**-l_rate**	**-n_ants**	**-d**
ACO (nuevo)	0.33	15	0.9
ACO (clásico)	0.21	10	0.86

Tabla 5.4: Rangos para ajuste de hiperparámetros en GRASP y ACO

Parámetro	**Rango**	**Tipo**
-max_iter	(20, 200)	entero
-d	(0.7, 0.99)	real
-l_rate	(0.1, 0.5)	real
-n_ants	(10, 50)	entero

Los métodos híbridos propuestos conservan la representación de soluciones del BRKGA descrita en la subsección 3.2.2, aprovechando su buen rendimiento en este problema. Se incorpora como metaheurística base de los métodos híbridos.

En el algoritmo CMSA, cada iteración emplea una heurística como motor de búsqueda principal para construir soluciones parciales. Estas soluciones se consolidan en un subproblema reducido que conserva únicamente las rachas más prometedoras. Este subproblema se resuelve luego mediante el solver ILP CPLEX, cuyos resultados retroalimentan el proceso al reforzar los componentes que aparecen frecuentemente en las mejores soluciones encontradas.

BARRAKUDA construye una subinstancia nueva en cada iteración a partir de las soluciones élite actuales de la población BRKGA, sin conservar información histórica, a diferencia de LCMSA que acumula conocimiento progresivo del problema al priorizar componentes frecuentes en buenas soluciones a lo largo de su ejecución. Los hiperparámetros ajustados para la adaptación de ambos algoritmos al problema LRS se presentan en la Tabla 5.5, mientras que los rangos asociados a cada parámetro se detallan en la Tabla 5.6.

Tabla 5.5: Ajuste de hiperparámetros para BARRAKUDA y CMSA

Algoritmo	**-p**	**-pe**	**-pm**	**-rhoe**
BARRAKUDA	327	0.22	0.14	0.6
Algoritmo	**-m_age**	**-cpl_t**	**-d**	**-n_sol**
CMSA	3	10	0.81	4

Tabla 5.6: Rango para ajuste de hiperparámetros BARRAKUDA y CMSA

Parámetro	**Rango**	**Tipo**
-p	(10, 400)	entero
-pe	(0.1, 0.25)	real
-pm	(0.1, 0.3)	real
-rhoe	(0.6, 0.9)	real
-m_age	(1, 10)	entero
-cpl_t	(1, 20)	entero
-d	(0.50, 0.99)	real
-n_sol	(1, 10)	entero
-iter	(15, 35)	entero
-na	(1, 10)	entero
-n_c	(0.5, 0.8)	real

Learned CMSA extiende la idea de CMSA mediante un mecanismo de aprendizaje evolutivo. Mantiene una población activa de soluciones BRKGA que evoluciona mediante operadores genéticos. Periódicamente, las soluciones más prometedoras de esta población son reintroducidas en el conjunto inicial, creando un ciclo de mejora continua donde el conocimiento adquirido guía la exploración futura del espacio de búsqueda. Los hiperparámetros en la Tabla 5.7 y sus rangos en la Tabla 5.8

Tabla 5.7: Ajuste de hiperparámetros LCMSA

Algoritmo	**-p**	**-pe**	**-pm**	**-rhoe**	**-m_age**	**-cpl_t**	**-n_c**	**-n_a**
LCMSA	200	0.13	0.24	0.67	3	6	0.59	4

Tabla 5.8: Rango para ajuste de hiperparámetros LCMSA

Parámetro	Rango	Tipo
-p	(10, 400)	entero
-pe	(0.1, 0.25)	real
-pm	(0.1, 0.3)	real
-rhoe	(0.51, 0.8)	real
-m_age	(1, 5)	entero
-cpl_t	(1, 20)	entero
-n_c	(0.5, 0.8)	real
-n_a	(0, 5)	entero

5.3. Adaptación BRKGA+LLMs

Siguiendo el marco teórico establecido. Como se definió en la Sección 2, los parámetros α y β ponderan la importancia relativa de cada métrica en la evaluación de los elementos a seleccionar, en este caso las rachas. Estas ponderaciones, junto con las métricas específicas de la racha r_i, influyen en la probabilidad de elección de una racha en la solución $p_{\text{LLM}}(r_i)$ definida en la ecuación (2.3).

La función Decode del algoritmo 1 fue modificada para incorporar el sesgo de probabilidades representadas por el vector de probabilidades $\vec{L} \in [0, 1]$ en el Algoritmo 9, con el fin de guiar la selección de rachas hacia aquellas con mayor relevancia según el conocimiento del modelo de lenguaje. Esta integración permite que la construcción de soluciones no solo dependa del largo de la cadena y el azar, como se realizaba previamente, sino que también aproveche información de las métricas definidas para favorecer ejecuciones con mayor probabilidad de contribuir a una subsecuencia óptima. El decoder realiza una ordenación ponderada de los índices de las rachas, combinando las claves aleatorias con las probabilidades LLM para priorizar aquellas opciones más prometedoras, y construye de manera greedy una solución válida evitando conflictos de interposición.

Algorithm 9 Hybrid BRKGA-LLM Decoder for LRS

Require: Random-key vector $\vec{v} \in [0,1]^n$, LLM probability vector $\vec{L} \in [0,1]^n$

Ensure: A feasible run-subsequence $\mathcal{S}$ and its score

Let $\pi \leftarrow$ permutation of $\{0, \ldots, n-1\}$ such that for all $i \in \{0, \ldots, n-2\}$:
    $\vec{v}_{\pi(i)} \cdot \vec{L}_{\pi(i)} \geq \vec{v}_{\pi(i+1)} \cdot \vec{L}_{\pi(i+1)}$ {LLM-biased sorting}
$\mathcal{S} \leftarrow \emptyset$ {Initialize empty solution set}
for $i \leftarrow 0$ to $n-1$ **do**
    $r \leftarrow \pi(i)$
    if $\mathcal{S} \cup \{r\}$ is a valid run-subsequence **then**
        $\mathcal{S} \leftarrow \mathcal{S} \cup \{r\}$
    end if
end for
score $\leftarrow \sum_{j \in \mathcal{S}} \text{length}(R_j)$
return $(\mathcal{S}, \text{score})$

5.3.1. Métricas y modelos a utilizar

Como parte de esta investigación, se realizó una adaptación de la herramienta OptiPattern con el objetivo de calcular $p_{\text{LLM}}()$ para cada racha, modificando su arquitectura original para operar específicamente con las rachas como elementos que pueden participar en la solución. La implementación resultante ha sido publicada como software de código abierto, disponible en el repositorio `https://github.com/martiniip/optipatternLRS`, donde se incluyen tanto los módulos de adaptación como ejemplos de configuración para distintos escenarios de prueba. Además, se presenta un template de la estructura general para obtener los valores alpha y beta en el anexo A

La determinación de métricas relevantes para este problema se realizó mediante un Prompt estructurado, utilizando un enfoque similar al empleado por la herramienta Optipattern. Este prompt fue diseñado como entrada para el modelo `meta-llama/llama-3-70b-instruct` (versión free). La estructura utilizada se presenta en el anexo C.

Del listado de métricas propuestas por el modelo, las elegidas fueron las siguientes:

- **Normalized length**: Longitud relativa de una racha, calculada como:

$$\text{normalized length} = \frac{\text{length(run)}}{\text{máx(length(run))}}$$

- **Normalized Future Opportunity**: Mide la proximidad de la siguiente ocurrencia de la

misma letra:

$$\text{normalized Future Opportunity} = \frac{1}{1 + (\text{next\_start} - \text{current\_start})}$$

- **Distance-Next**: Distancia normalizada hasta la siguiente aparición del mismo caracter:

$$\text{distance-next} = \frac{\text{next\_occurrence} - \text{end\_pos}}{\text{length}(s)}$$

- **Local Density**: Densidad o frecuencia de una letra en la cadena:

$$\text{local\_density} = \frac{\text{número de ocurrencias de la letra}}{\text{length}(s)}$$

Con el fin de seleccionar qué LLM usar para obtener los valores α y β, se evaluó el rendimiento de modelos LLM de acceso gratuito como mecanismo de sesgo para el algoritmo BRKGA; se utilizó un subconjunto de instancias representativas del dataset; estas instancias presentan buen rendimiento en BRKGA. Los resultados, presentados en la Tabla 5.9 demuestran que la incorporación del sesgo mediante Llama-3.2-3b-instruct:free produce un puntaje promedio levemente superior al del algoritmo BRKGA en su configuración original. Este hallazgo sugiere que, dado el mejor desempeño obtenido con modelos sin costo, es probable que la utilización de modelos de pago pueda generar mejoras adicionales en los resultados, tal como se propone en el estudio comparativo de [2].

Tabla 5.9: Comparación puntaje promedios de instancias ejecutadas 10 veces, en negrita mejores puntajes por fila

Instancia	DEEPSEEK	DEEPHERMES	META_LLAMA	BRKGA
len_100_sigma16_10.txt	34.600	34.100	34.200	**35.000**
len_1000_sigma8_16.txt	194.000	194.200	**194.700**	194.400
len_200_sigma32_10.txt	**56.900**	54.500	55.200	**56.900**
len_2000_sigma16_16.txt	219.700	220.500	**222.500**	220.700
len_300_sigma8_7.txt	**75.000**	**75.000**	**75.000**	**75.000**
len_500_sigma4_17.txt	**157.000**	**157.000**	**157.000**	**157.000**
len_5000_sigma32_1.txt	333.000	334.500	**337.600**	331.700
PROMEDIO	152.886	152.829	**153.743**	152.957

La elección de modelos LLM de pago a utilizar se basa en la relación Valor-Rank, donde se valoriza que sean de menor valor y alto rank. En la Tabla 5.10 se resaltan los modelos seleccionados con una estimación total de 19,51 dólares, el cálculo de tokens se presenta en el

anexo B. En la Tabla 5.11 se presenta el ajuste de hiperparámetros realizado para la hibridación de BRKGA con los modelos seleccionados.

Tabla 5.10: Cálculos considerando output de 1M de tokens (valores en USD), en amarillo LLMs seleccionados

Modelo	Rank	$/1M input (total input)	$/1M output	Contexto	Total estimado
Gemini-2.5-Pro-Preview-05-06	1	$1.25 ($29.428)	$10	1M	$39.428
Google: Gemini 2.5 Flash Preview 05-20	2	$0.15 ($3.531)	$0.60	1M	$4.131
OpenAI: GPT-4.1	8	$2.00 ($47.084)	$8.00	1M	$55.084
OpenAI: GPT-4.1 Mini	10	$0.40 ($9.417)	$1.60	1M	$11.017
Meta: Llama 4 Maverick	47	$0.16 ($3.767)	$0.60	1M	$4.367
Amazon: Nova Pro 1.0	71	$0.80 ($18.834)	$3.20	300k	$22.034

* Ranking basado en `https://huggingface.co/spaces/lmarena-ai/chatbot-arena-leaderboard` al 27-05-2025

* Cálclulos considerando output de 1M de tokens

* Cálclulos considerando total tokens dataset: 23,542,200

Tabla 5.11: Ajuste de hiperparámetros para hibridaciones de BRKGA con LLMS, mismos rangos de parámetros definidos para BRKGA en la Tabla 5.2

Variante BRKGA	-p	-pe	-pm	-rhoe
Gemini-2.5-flash	21	0.17	0.23	0.60
Llama3.2-3b:free	27	0.19	0.21	0.78
GPT4.1-mini	36	0.23	0.21	0.68
Llama-4-maverick	17	0.12	0.30	0.69

Ya con las métricas obtenidas para cada racha y los LLM seleccionados se utilizó el modelo lineal definido previamente en el marco teórico, que utiliza las métricas calculadas mediante OptiPattern para generar probabilidades de selección. Estas probabilidades, obtenidas para todo el conjunto de datos y para cada uno de los modelos evaluados, están disponibles públicamente en el repositorio que contiene la adaptación de la herramienta.

Capítulo 6. Resultados

Mediante este capítulo se presentan los resultados experimentales obtenidos en la evaluación comparativa de los algoritmos, cabe mencionar que las comparaciones realizadas con el estado del arte BRKGA corresponden a la adaptación mencionada en el capitulo 5.

Los experimentos se diseñaron para medir el desempeño en función de dos parámetros clave: el tamaño de la subsecuencia solución encontrada (avg) y el tiempo para llegar a aquella solución ($Time$). Se implementaron 30 ejecuciones independientes por cada combinación de parámetros, reportándose los valores promedio para facilitar la comparación visual y en negrita cuando un algoritmo tiene mejor puntaje por instancia. A continuación, se detallan los resultados de los experimentos descritos anteriormente.

6.1. Hiperparámetros segmentados

A continuación se presenta la Tabla 6.1 con el rendimiento general para los algoritmos BRKGA y LCMSA, en el dataset de prueba.

Tabla 6.1: Comparación general BRKGA vs LCMSA, reportando puntajes promedio por combinación length-Σ, mejores puntajes en negrita y tiempo en segundos

Length	$\|\Sigma\|$	**BRKGA**		**LCMSA**	
		avg.	**time**	**avg.**	**time**
100	2	**59.03**	0.00	**59.03**	0.00
100	4	**41.07**	0.03	**41.07**	0.02
100	8	**33.87**	1.28	33.67	1.20
100	16	**34.53**	4.65	34.03	4.69
100	32	**39.93**	9.91	39.43	8.40
200	2	**115.17**	0.00	**115.17**	0.00
200	4	**72.77**	0.38	72.73	0.40
200	8	**55.17**	4.27	54.87	3.06
200	16	**50.73**	16.55	49.97	9.93
200	32	**54.70**	25.10	53.07	17.33
300	2	**165.47**	0.01	**165.47**	0.00
300	4	**102.73**	2.88	102.70	0.37
300	8	**74.97**	9.65	74.60	9.54
300	16	**65.80**	28.89	64.73	15.51
300	32	**65.70**	40.60	63.43	22.79
500	2	**271.67**	0.02	**271.67**	0.04
500	4	**162.77**	3.84	162.70	0.87
500	8	109.57	11.51	**109.60**	13.54
500	16	**90.57**	49.01	90.07	38.54
500	32	**84.87**	71.69	83.50	54.77
1000	2	**530.23**	0.05	**530.23**	0.23
1000	4	**301.10**	7.60	300.83	25.18
1000	8	**191.00**	50.82	189.73	51.99
1000	16	**142.70**	119.45	139.97	108.04
1000	32	**122.63**	143.78	119.10	101.46
2000	2	**1041.73**	0.58	1041.70	1.25
2000	4	**572.30**	21.06	571.43	54.97
2000	8	**343.27**	115.35	340.53	125.89
2000	16	**232.33**	276.39	227.07	273.23
2000	32	**182.90**	303.72	176.80	278.96
5000	2	**2567.47**	0.90	**2567.47**	0.25
5000	4	**1361.23**	110.76	1359.73	151.62
5000	8	**765.20**	339.43	761.17	397.56
5000	16	**478.63**	801.50	470.50	708.18
5000	32	**331.10**	863.47	317.77	821.74
Promedio		**311.85**	98.15	310.16	94.33

En términos de los criterios de evaluación generales (avg y time), cuyos valores se resumen en los promedios del final de la Tabla 6.1, el algoritmo BRKGA demostró el mejor rendimiento global. No obstante, su superioridad es más notable en instancias de gran tamaño y con alfabetos extensos. Este resultado justifica la aplicación de estrategias de segmentación de hiperparámetros, las cuales podrían optimizar el rendimiento de LCMSA en estos escenarios.

6.1.1. Comparativa por segmentos

A continuación se presentan los resultados del ajuste de hiperparámetros por segmentos para los algoritmos BRKGA y LCMSA (Tabla 6.2), así como también los resultados de la evaluación en el dataset de prueba para comparar rendimientos (Tabla 6.3).

Tabla 6.2: Ajuste de Hiperparámetros Segmentados para BRKGA y LCMSA, mismos rangos definidos en la Tabla 5.8

Segmentos BRKGA	**-p**	**-pe**	**-pm**	**-rhoe**
biglength	265	0.25	0.23	0.51
shortlength	72	0.18	0.21	0.63
big|Σ|	45	0.19	0.30	0.75
short|Σ|	86	0.13	0.29	0.52

Segmentos LCMSA	**-p**	**-pe**	**-pm**	**-rhoe**	**-m_age**	**-cpl_t**	**-n_c**	**-n_a**
biglength	363	0.25	0.26	0.66	2	7	0.52	1
shortlength	149	0.11	0.29	0.80	3	5	0.70	2
big|Σ|	244	0.16	0.30	0.76	3	6	0.59	2
short|Σ|	161	0.23	0.20	0.56	2	5	0.73	2

Los resultados demuestran que LCMSA no logra superar el desempeño de BRKGA en ninguna configuración (Tabla 6.3), incluso en instancias grandes donde se esperaba mayor ventaja. Contrario a lo hipotetizado, BRKGA mantuvo su superioridad en términos de puntaje en todos los escenarios, no así en términos de tiempo de ejecución. Sin embargo, el análisis revela que la segmentación por tamaño de alfabeto (|Σ|) en BRKGA obtuvo resultados excepcionales, superando incluso promedios del estado del arte, principalmente en instancias complejas. Esta mejora, no obstante, no es comparable, puesto que es el mismo algoritmo; se debe tomar más bien como un sobreajuste a las instancias del dataset, lo que podría ser una buena técnica para sacarle más provecho a un algoritmo en particular.

Tabla 6.3: Comparación de BRKGA y LCMSA con ajuste de hiperparámetros segmentado por características estructurales. Los sufijos (L) y (S) indican ajustes para Longitud (Short: 100-500 azul, Big: 1000-5000 rojo) y Sigma (Short: 2-8 verde, Big: 16-32 amarillo) respectivamente.

Length	$\|\Sigma\|$	**BRKGA (L)**		**LCMSA (L)**		**BRKGA (S)**		**LCMSA (S)**	
		avg.	time	avg.	time	avg.	time	avg.	time
100	2	**59.03**	0.01	**59.03**	0.00	**59.03**	0.00	**59.03**	0.00
100	4	**41.07**	0.23	**41.07**	0.04	**41.07**	0.03	**41.07**	0.06
100	8	33.40	2.05	33.70	0.30	**33.80**	0.99	33.77	0.76
100	16	33.30	6.42	34.23	4.41	**34.50**	3.33	34.40	5.94
100	32	37.80	13.00	39.77	7.85	**41.03**	7.20	40.20	8.62
200	2	**115.17**	0.03	**115.17**	0.00	**115.17**	0.00	**115.17**	0.01
200	4	72.70	1.86	**72.77**	0.11	**72.77**	0.49	72.73	0.12
200	8	54.57	8.37	**55.07**	3.01	**55.07**	4.00	55.03	3.09
200	16	48.73	20.81	50.17	9.83	**51.03**	12.41	50.27	12.22
200	32	54.73	29.54	54.67	20.13	**55.37**	19.48	53.97	20.74
300	2	**165.47**	0.06	**165.47**	0.00	**165.47**	0.01	**165.47**	0.00
300	4	102.63	1.85	**102.73**	1.52	**102.73**	0.94	102.70	0.76
300	8	73.97	12.46	74.80	3.48	**74.83**	6.75	74.70	6.93
300	16	63.30	31.71	65.87	23.64	**66.10**	19.60	64.80	21.67
300	32	65.63	40.54	65.53	29.38	**66.93**	42.97	64.07	29.13
500	2	**271.67**	0.21	**271.67**	0.01	**271.67**	0.01	**271.67**	0.02
500	4	162.67	6.99	162.70	0.60	**162.73**	1.14	162.70	0.57
500	8	108.43	28.35	109.67	6.67	109.67	16.69	**109.73**	19.66
500	16	89.90	61.09	90.00	32.73	**90.97**	44.19	89.77	50.26
500	32	84.20	81.87	84.63	65.83	**87.70**	63.89	83.73	55.76
1000	2	**530.23**	0.06	**530.23**	0.03	**530.23**	0.05	**530.23**	0.03
1000	4	300.90	6.61	300.93	14.67	**301.07**	15.14	300.83	15.53
1000	8	190.87	36.60	190.87	45.97	**191.17**	62.84	190.87	45.40
1000	16	141.97	82.35	142.00	90.63	**143.83**	96.25	140.87	94.96
1000	32	122.47	134.09	122.90	127.20	**124.50**	157.57	119.03	116.83
2000	2	**1041.73**	0.90	**1041.73**	0.20	**1041.73**	0.62	**1041.73**	0.25
2000	4	**572.33**	24.17	571.90	12.82	572.40	16.23	572.00	16.11
2000	8	342.62	91.23	**343.00**	125.15	342.43	167.57	342.47	137.05
2000	16	231.00	218.05	233.20	173.37	**235.20**	224.49	226.27	212.45
2000	32	185.10	294.20	185.00	284.02	**187.00**	340.78	174.13	217.05
5000	2	**2567.47**	1.15	**2567.47**	1.39	2567.00	1.13	**2567.47**	0.98
5000	4	1360.83	178.17	1359.87	15.73	**1361.00**	110.61	1359.23	5.40
5000	8	**766.40**	285.19	755.73	28.02	764.17	427.01	754.40	16.76
5000	16	479.97	632.78	478.20	624.69	**483.37**	656.16	458.97	644.78
5000	32	330.33	802.89	330.33	743.04	**337.97**	845.29	305.80	698.26
Promedio		311.50	89.60	311.49	71.33	**312.59**	96.17	309.41	70.23

6.2. Nueva heurística

En esta sección se presentan los resultados de la construcción de una nueva heurística. Como se evidencia en la Tabla 6.4, los resultados demuestran una superioridad de la nueva heurística propuesta frente a la heurística convencional, principalmente en instancias complejas comenzando desde length 500. Esta mejora en el desempeño, cuantificada mediante el promedio de puntajes avg, justifica plenamente la fase experimental subsiguiente. Concretamente, estos resultados prometedores motivaron la segunda etapa de la investigación, donde se integra esta heurística mejorada dentro de estructuras metaheurísticas más complejas.

Tabla 6.4: Comparación de resultados entre la heurística original y la nueva

Length	$\|\Sigma\|$	**Heurística Original**		**Heurística Nueva**	
		avg.	time	avg.	time
100	2	**55.80**	<0.03	56.53	<0.03
100	4	**35.03**	<0.03	34.03	<0.03
100	8	**22.40**	<0.03	21.43	<0.03
100	16	**16.17**	<0.03	14.67	<0.03
100	32	**15.37**	<0.03	11.23	<0.03
200	2	**109.87**	<0.03	109.40	<0.03
200	4	60.27	<0.03	**60.80**	<0.03
200	8	**38.73**	<0.03	37.87	<0.03
200	16	22.50	<0.03	**23.77**	<0.03
200	32	15.17	<0.03	**16.57**	<0.03
300	2	159.10	<0.03	159.10	<0.03
300	4	**88.00**	<0.03	87.93	<0.03
300	8	49.27	<0.03	**50.57**	<0.03
300	16	**33.27**	<0.03	32.77	<0.03
300	32	**26.53**	<0.03	22.50	<0.03
500	2	260.13	<0.03	**261.07**	<0.03
500	4	139.97	<0.03	**142.27**	<0.03
500	8	80.77	<0.03	**84.30**	<0.03
500	16	47.90	<0.03	**51.23**	<0.03
500	32	35.00	<0.03	**35.63**	<0.03
1000	2	511.57	<0.03	**515.60**	<0.03
1000	4	263.70	<0.03	**274.23**	<0.03
1000	8	144.80	<0.03	**153.97**	<0.03
1000	16	80.07	<0.03	**89.23**	<0.03
1000	32	43.83	<0.03	**55.63**	<0.03
2000	2	1014.37	<0.03	**1025.63**	<0.03
2000	4	517.03	<0.03	**535.67**	<0.03
2000	8	277.80	<0.03	**293.03**	<0.03
2000	16	145.73	<0.03	**162.93**	<0.03
2000	32	85.07	<0.03	**96.47**	<0.03
5000	2	2509.73	<0.03	**2544.70**	<0.03
5000	4	1279.60	0.03	**1306.07**	<0.03
5000	8	655.77	<0.03	**687.60**	<0.03
5000	16	339.37	<0.03	**376.63**	<0.03
5000	32	187.03	<0.03	**202.67**	<0.03
Promedio		267.62	0.008	**275.25**	0.006

Incorporación en metaheurísticas GRASP y ACO

Los resultados de integrar estas heurísticas en los algoritmos ACO y GRASP confirman la mejora sustancial respecto a la heurística tradicional. Como se detalla en la Tabla 6.5, ambas metaheurísticas, al incorporar la nueva heurística, exhiben un mejor desempeño general en comparación con sus versiones que utilizan la heurística convencional. Este avance es particularmente notable en instancias de gran escala, un comportamiento consistente con los resultados reportados en la Tabla 6.4. Estos hallazgos refuerzan la hipótesis de que el éxito de estas metaheurísticas depende de la calidad de la heurística subyacente.

Sin embargo, aunque estos resultados son prometedores, aún se mantienen considerablemente distantes del estado del arte en términos de puntaje. Esta brecha de rendimiento motiva la siguiente fase: el desarrollo de metaheurísticas híbridas que reduzcan su dependencia de heurísticas.

Tabla 6.5: Comparación del desempeño entre implementación de metaheurísticas GRASP y ACO con heurística clásica vs la nueva, en engrita mejores resultados y tiempo en segundos

Length	$\|\Sigma\|$	**GRASP Clásico**		**GRASP Nuevo**		**ACO Clásico**		**ACO Nuevo**	
		avg.	time	avg.	time	avg.	time	avg.	time
100	2	**59.03**	0.00	58.90	0.00	57.03	0.01	57.34	0.01
100	4	**39.77**	0.83	39.84	0.43	37.38	1.09	37.31	0.50
100	8	**30.90**	1.47	30.10	1.68	25.95	1.08	26.75	1.54
100	16	**27.23**	1.89	26.77	2.53	19.17	1.07	19.62	1.99
100	32	**26.93**	1.69	24.52	3.07	17.85	1.33	15.16	1.23
200	2	**114.75**	0.02	114.54	0.03	111.07	0.07	111.83	0.02
200	4	**70.32**	0.47	69.93	1.05	65.95	2.31	67.54	1.36
200	8	**50.75**	4.13	49.82	4.16	43.74	3.26	45.14	3.06
200	16	36.20	4.00	**40.26**	5.66	27.64	3.38	30.08	3.83
200	32	25.28	3.40	**32.83**	5.13	19.25	2.81	21.29	2.88
300	2	**164.03**	0.06	163.84	0.13	161.35	0.20	160.90	0.05
300	4	**99.92**	2.13	99.34	3.19	94.59	3.01	95.45	2.59
300	8	**68.65**	7.35	67.42	7.34	57.20	5.32	60.62	5.14
300	16	50.84	7.88	**51.01**	7.22	37.72	4.84	40.20	5.63
300	32	**43.12**	8.49	40.63	7.83	29.32	5.75	28.12	5.39
500	2	**270.57**	1.64	270.08	0.76	266.36	0.54	267.20	0.70
500	4	**158.64**	3.51	156.85	4.90	150.05	5.85	152.05	4.59
500	8	100.47	9.63	**101.00**	9.69	89.85	9.25	92.69	8.26
500	16	68.68	12.73	**73.32**	14.23	53.54	10.54	58.42	12.67
500	32	**57.37**	14.88	56.26	15.02	38.42	10.23	39.56	9.37
1000	2	526.86	4.80	**528.68**	4.38	520.30	3.38	519.56	3.69
1000	4	**293.09**	17.83	292.12	18.20	280.02	12.35	286.13	14.30
1000	8	175.81	32.90	**178.05**	18.16	157.30	20.34	166.81	19.58
1000	16	110.60	31.17	**118.69**	23.11	89.87	18.41	98.40	20.06
1000	32	69.41	27.68	**83.67**	28.32	52.33	23.89	61.71	18.61
2000	2	1034.05	10.10	**1035.50**	9.37	1027.92	7.82	1030.22	6.50
2000	4	556.99	42.01	**559.31**	28.45	544.18	36.83	553.21	39.15
2000	8	313.51	55.78	**322.22**	55.35	292.22	41.53	306.98	40.67
2000	16	186.07	56.19	**197.31**	65.57	160.80	47.12	173.78	56.65
2000	32	121.13	64.66	**130.95**	57.13	91.40	53.52	104.44	57.81
5000	2	2548.00	34.22	**2558.26**	50.87	2539.86	86.35	2549.63	29.37
5000	4	1329.66	127.39	**1338.33**	119.60	1311.98	160.43	1330.30	138.00
5000	8	710.87	136.08	**727.81**	149.45	682.56	146.69	705.74	140.67
5000	16	397.67	131.60	**421.03**	169.32	358.86	131.71	390.25	158.05
5000	32	230.79	157.47	**252.32**	160.63	197.53	152.19	214.20	170.28
Promedio		290.51	29.03	**294.61**	30.06	277.44	28.99	283.39	28.12

6.3. Hibridación con Métodos Exactos

Para este experimento se desarrollaron y adaptaron las metaheurísticas CMSA y BARRAKUDA definidas con anterioridad en la sección 2.4, y evaluándolos en el dataset para obtener un panorama general del rendimiento. Los resultados son los siguientes:

Tabla 6.6: Comparación de variantes BRKGA con diferentes estrategias (valores promedio y tiempos en segundos)

Length	|Σ|	BRKGA		LCMSA		CMSA		BARRAKUDA	
		avg.	time	avg.	time	avg.	time	avg.	time
100	2	**59.03**	0.00	**59.03**	0.00	57.72	0.01	**59.03**	0.00
100	4	**41.07**	0.03	**41.07**	0.02	38.49	0.31	**41.07**	0.04
100	8	**33.87**	1.28	33.67	1.20	29.52	1.95	33.73	0.68
100	16	**34.53**	4.65	34.03	4.69	26.29	1.86	34.20	4.02
100	32	39.93	9.91	39.43	8.40	27.34	1.92	**40.00**	7.82
200	2	**115.17**	0.00	**115.17**	0.00	112.86	0.08	**115.17**	0.02
200	4	**72.77**	0.38	72.73	0.40	67.93	3.59	72.73	0.26
200	8	**55.17**	4.27	54.87	3.06	47.12	4.04	54.80	5.40
200	16	**50.73**	16.55	49.97	9.93	37.52	6.82	49.63	12.65
200	32	**54.70**	25.10	53.07	17.33	35.40	7.80	54.17	20.82
300	2	**165.47**	0.01	**165.47**	0.00	162.32	1.97	**165.47**	0.03
300	4	**102.73**	2.88	102.70	0.37	95.38	5.90	**102.73**	0.84
300	8	**74.97**	9.65	74.60	9.54	61.75	8.40	74.77	8.53
300	16	**65.80**	28.89	64.73	15.51	47.93	9.52	65.60	15.14
300	32	**65.70**	40.60	63.43	22.79	42.53	12.27	64.80	30.84
500	2	**271.67**	0.02	**271.67**	0.04	267.61	8.47	**271.67**	0.07
500	4	**162.77**	3.84	162.70	0.87	151.95	10.18	162.70	1.28
500	8	109.57	11.51	**109.60**	13.54	93.45	11.09	109.20	16.29
500	16	**90.57**	49.01	90.07	38.54	65.99	8.99	89.00	46.69
500	32	**84.87**	71.69	83.50	54.77	54.99	21.03	84.40	62.70
1000	2	**530.23**	0.05	**530.23**	0.23	521.64	19.01	**530.23**	0.27
1000	4	**301.10**	7.60	300.83	25.18	286.66	22.98	300.97	11.18
1000	8	**191.00**	50.82	189.73	51.99	166.87	31.20	190.60	54.26
1000	16	**142.70**	119.45	139.97	108.04	104.51	20.58	141.63	98.19
1000	32	**122.63**	143.78	119.10	101.46	78.23	27.60	121.37	136.28
2000	2	**1041.73**	0.58	1041.70	1.25	1032.61	46.11	**1041.73**	1.85
2000	4	**572.30**	21.06	571.43	54.97	551.78	75.15	571.73	27.32
2000	8	**343.2**7	115.35	340.53	125.89	308.30	59.94	342.40	142.87
2000	16	**232.33**	276.39	227.07	273.23	178.80	57.01	231.40	233.69
2000	32	**182.90**	303.72	176.80	278.96	117.93	51.09	181.50	299.95
5000	2	**2567.47**	0.90	**2567.47**	0.25	2554.10	104.12	**2567.47**	6.05
5000	4	**1361.23**	110.76	1359.73	151.62	1319.34	149.71	1359.67	22.18
5000	8	**765.20**	339.43	761.17	397.56	706.89	127.55	760.20	326.92
5000	16	**478.63**	801.50	470.50	708.18	398.41	162.61	474.00	693.50
5000	32	**331.10**	863.47	317.77	821.74	234.96	147.19	321.10	860.20
Promedio		**311.85**	98.15	310.16	94.33	288.15	35.09	310.88	89.97

Como se muestra en la Tabla 6.6, el objetivo principal ha sido optimizar la calidad de las soluciones medida a través del puntaje promedio avg más que el tiempo de ejecución. Los resultados presentan dos aspectos notables. Primero, ninguna de las metaheurísticas adaptadas supera al BRKGA original, siendo BARRAKUDA la que alcanza el rendimiento más cercano. Este comportamiento puede atribuirse a su naturaleza híbrida, que combina métodos exactos para resolver subinstancias con un mecanismo de reconstrucción en cada iteración, a diferencia de enfoques como LCMSA que mantienen las subinstancias de forma permanente.

A pesar de no alcanzar los resultados esperados, el hecho de que BRKGA supere a las técnicas híbridas evaluadas sugiere que su potencial podría explotarse mejor mediante una hibridación directa. Por ello, se propone el siguiente experimento de hibridación con LLMs.

6.4. Integración BRKGA+LLMs

Se presenta a continuación la evaluación del rendimiento de distintas variaciones de BRKGA con LLMs en el conjunto de datos completo, con el objetivo de garantizar la comparabilidad con los resultados del algoritmo BRKGA.

Tabla 6.7: Comparación del rendimiento de variantes de BRKGA con integraciones de LLM para el LRS. Los mejores valores en cada fila se resaltan en negrita.

Length	$\lvert\Sigma\rvert$	**BRKGA**		**BRKGA integrates with**							
				GPT-4.1-mini		**Gemini-2.5-Flash**		**Llama-3.2-3b**		**Llama-4-Maverick**	
		avg.	time	avg.	time	avg.	time	avg.	time	avg.	time
	2	**59.03**	0.00	**59.03**	0.00	**59.03**	0.00	**59.03**	0.00	**59.03**	0.00
	4	**41.07**	0.03	**41.07**	0.01	**41.07**	0.01	**41.07**	0.01	**41.07**	0.02
100	8	33.87	1.28	**33.90**	1.69	33.87	1.02	33.87	0.68	**33.90**	0.92
	16	34.53	4.65	34.57	3.43	34.57	3.40	34.53	4.61	**34.60**	3.04
	32	39.93	9.91	40.40	8.67	40.93	10.97	40.83	7.94	**41.10**	8.29
	2	**115.17**	0.00	**115.17**	0.00	**115.17**	0.00	**115.17**	0.00	**115.17**	0.00
	4	**72.77**	0.38	**72.77**	0.09	**72.77**	0.06	**72.77**	0.22	**72.77**	0.17
200	8	**55.17**	4.27	55.10	2.71	**55.17**	3.21	55.10	3.75	55.13	1.87
	16	50.73	16.55	50.83	12.33	50.87	11.80	50.83	13.24	**51.03**	11.30
	32	54.70	25.10	55.37	22.66	54.50	21.15	55.13	23.02	**55.87**	21.21
	2	**165.47**	0.01	**165.47**	0.00	**165.47**	0.00	**165.47**	0.00	**165.47**	0.00
	4	**102.73**	2.88	**102.73**	0.65	**102.73**	0.78	**102.73**	0.63	**102.73**	0.74
300	8	74.97	9.65	75.07	2.14	**75.10**	3.20	75.07	6.44	**75.10**	4.17
	16	65.80	28.89	66.30	17.80	**66.77**	22.42	66.27	19.53	66.63	21.37
	32	65.70	40.60	66.03	33.47	66.50	40.22	66.47	35.35	**67.63**	33.17
	2	**271.67**	0.02	**271.67**	0.01	**271.67**	0.00	**271.67**	0.01	**271.67**	0.02
	4	**162.77**	3.84	162.73	0.47	**162.77**	0.84	**162.77**	0.33	**162.77**	0.45
500	8	109.57	11.51	109.93	15.07	**110.20**	14.01	110.13	13.29	**110.20**	21.53
	16	90.57	49.01	90.60	30.33	90.80	38.49	90.33	48.46	**91.17**	42.28
	32	84.87	71.69	85.50	63.40	85.77	61.78	86.07	63.51	**87.17**	63.28
	2	**530.23**	0.05	**530.23**	0.01	**530.23**	0.02	**530.23**	0.05	**530.23**	0.02
	4	**301.10**	7.60	**301.10**	6.91	**301.10**	6.12	301.00	8.75	301.00	6.18
1000	8	191.00	50.82	**191.87**	47.87	191.23	33.07	191.23	32.18	191.80	34.84
	16	142.70	119.45	142.90	99.31	143.33	99.68	142.77	95.64	**143.37**	107.15
	32	122.63	143.78	123.97	145.17	124.00	141.95	123.97	158.01	**126.23**	140.86
	2	**1041.73**	0.58	**1041.73**	0.07	**1041.73**	0.12	**1041.73**	0.15	**1041.73**	0.12
	4	572.30	21.06	572.67	15.62	**572.50**	6.18	**572.50**	22.29	**572.50**	29.98
2000	8	343.27	115.35	343.17	84.56	343.10	104.27	343.70	117.35	**343.90**	65.82
	16	232.33	276.39	234.63	233.89	235.00	222.05	233.73	193.54	**235.10**	199.04
	32	182.90	303.72	186.80	300.61	188.13	320.46	186.23	299.06	**189.90**	317.78
	2	**2567.47**	0.90	**2567.47**	1.00	**2567.47**	0.50	**2567.47**	0.27	**2567.47**	0.22
	4	1361.23	110.76	**1361.30**	151.09	1361.17	92.46	**1361.30**	117.40	1361.17	78.86
5000	8	765.20	339.43	766.87	330.09	**768.50**	313.15	767.13	319.40	768.07	372.71
	16	478.63	801.50	483.57	630.08	481.33	571.54	483.90	622.15	**486.23**	742.38
	32	331.10	863.47	335.73	797.70	336.93	814.40	336.50	792.55	**338.40**	809.65
Average		311.85	98.15	312.52	87.40	312.61	84.55	312.53	86.28	**313.07**	89.70

Como se muestra en la Tabla 6.7, todos los modelos propuestos superaron a BRKGA tanto en el rendimiento promedio como en el tiempo promedio de convergencia, lo que constituye un primer indicio relevante de la efectividad del enfoque. Las ventajas más significativas se observan en instancias de mayor tamaño, como aquellas con length 2000 y 5000. Además, se aprecia que con $|\Sigma| = 2$ el problema parece ser trivial, ya que todos los métodos convergen a la misma solución. En contraste, la dificultad del problema se vuelve evidente para valores de $|\Sigma| \geq 16$.

Para confirmar la significancia estadística de estas mejoras, se presenta el análisis complementario mediante el gráfico de diferencias críticas, como se propone en la metodología.

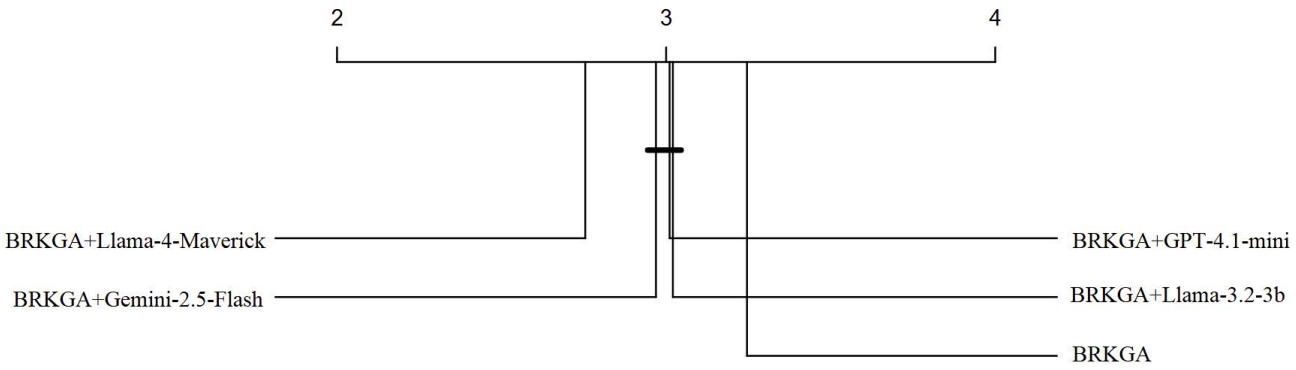


Figura 6.1: Diagrama de Diferencia Crítica (CD). Se muestran los rangos promedio de cinco variantes de BRKGA en 1050 instancias (dataset completo). BRKGA+Llama-4-Maverick y BRKGA+Gemini-2.5-Flash obtuvieron mejores rangos (más bajos, izquierda), mientras que BRKGA (base), BRKGA+Llama-3.2-3b y BRKGA+GPT-4.1-mini tuvieron rangos peores (más altos, derecha). Los algoritmos conectados por la misma línea horizontal no difieren significativamente

En la Figura 6.1 el gráfico de diferencias críticas demuestra que todas las versiones híbridas de BRKGA con LLMs presentan un rendimiento superior al BRKGA convencional. Específicamente, el modelo Meta: Llama 4 Maverick demuestra ser estadísticamente mejor que todas las demás configuraciones, posicionándose como la mejor alternativa. Los demás modelos híbridos, aunque estadísticamente equivalentes entre sí según el análisis representado por la barra de agrupación, superan consistentemente al BRKGA base. Estos resultados confirman que la hibridación con técnicas de LLMs aporta mejoras en el rendimiento del algoritmo.

Resulta aún más valioso el gran desempeño de estas hibridaciones al considerar los tiempos de respuesta de los LLMs para obtener los parámetros α y β. Aunque este cálculo constituye una etapa preliminar no incluida directamente en el algoritmo principal, el tiempo total del cálculo de probabilidades depende de este paso, ya que la obtención de estos parámetros permite al modelo lineal de optimización computar las probabilidades de forma casi instantánea.

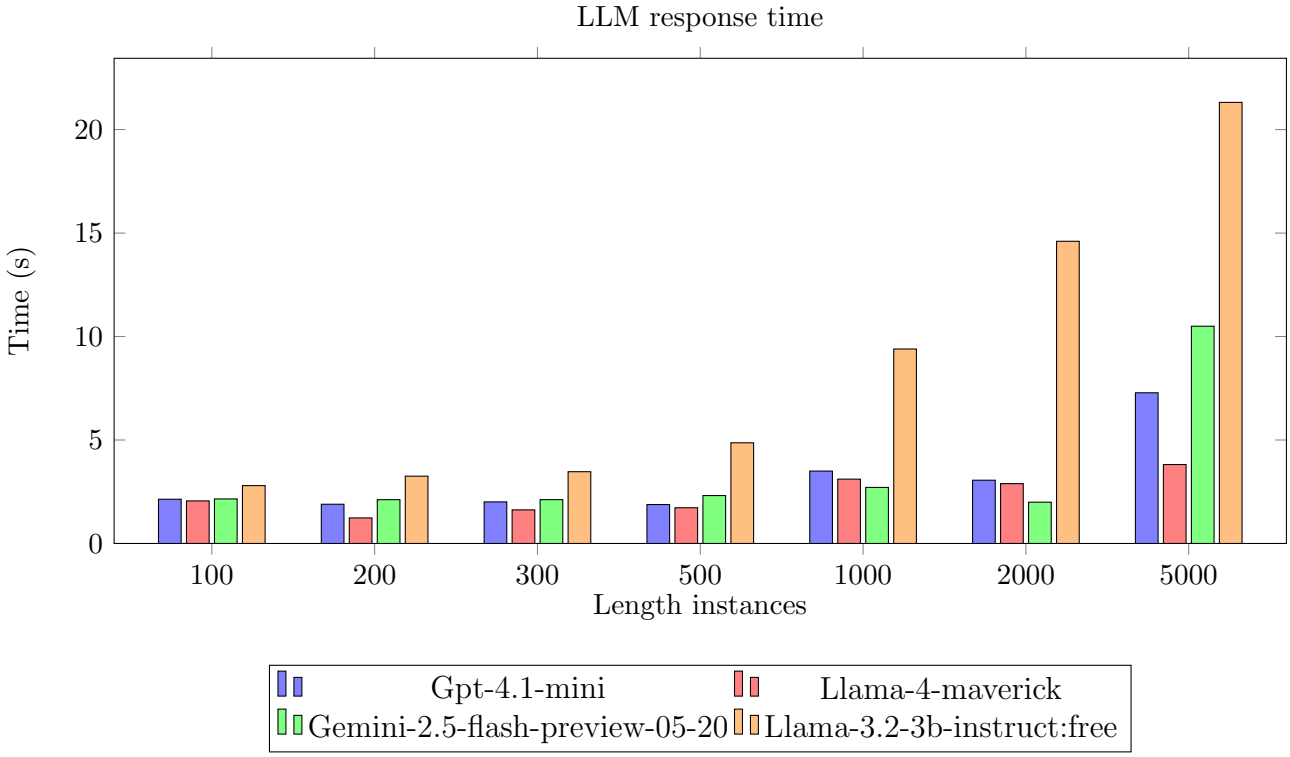


Figura 6.2: Comparación de tiempos de respuesta (en segundos) de modelos LLM para diferentes instancias de longitud (*length*). Se evaluaron cuatro variantes: GPT-4.1-mini, Llama-4-maverick, Gemini-2.5-flash y Llama-3.2-3b-instruct.

Según se aprecia en la Figura 6.2, el mayor tiempo de respuesta (aproximadamente 22 segundos) corresponde al modelo gratuito Llama-3.2-3b-instruct:free en instancias con length=5000. En contraste, los modelos comerciales presentan una mayor eficiencia, con un tiempo promedio por instancia de alrededor de 4 segundos (excluyendo el modelo gratuito).

6.4.1. Pruebas con aleatorización

Para validar estos resultados, se reemplazaron las probabilidades obtenidas por la herramienta Optippattern de cada racha, por probabilidades aleatorias estáticas, es decir, mismas para todos los modelos y dinámicas 6.8 probabilidades aleatorias distintas para cada modelo y manteniendo en ambos casos los mismos hiperparámetros.

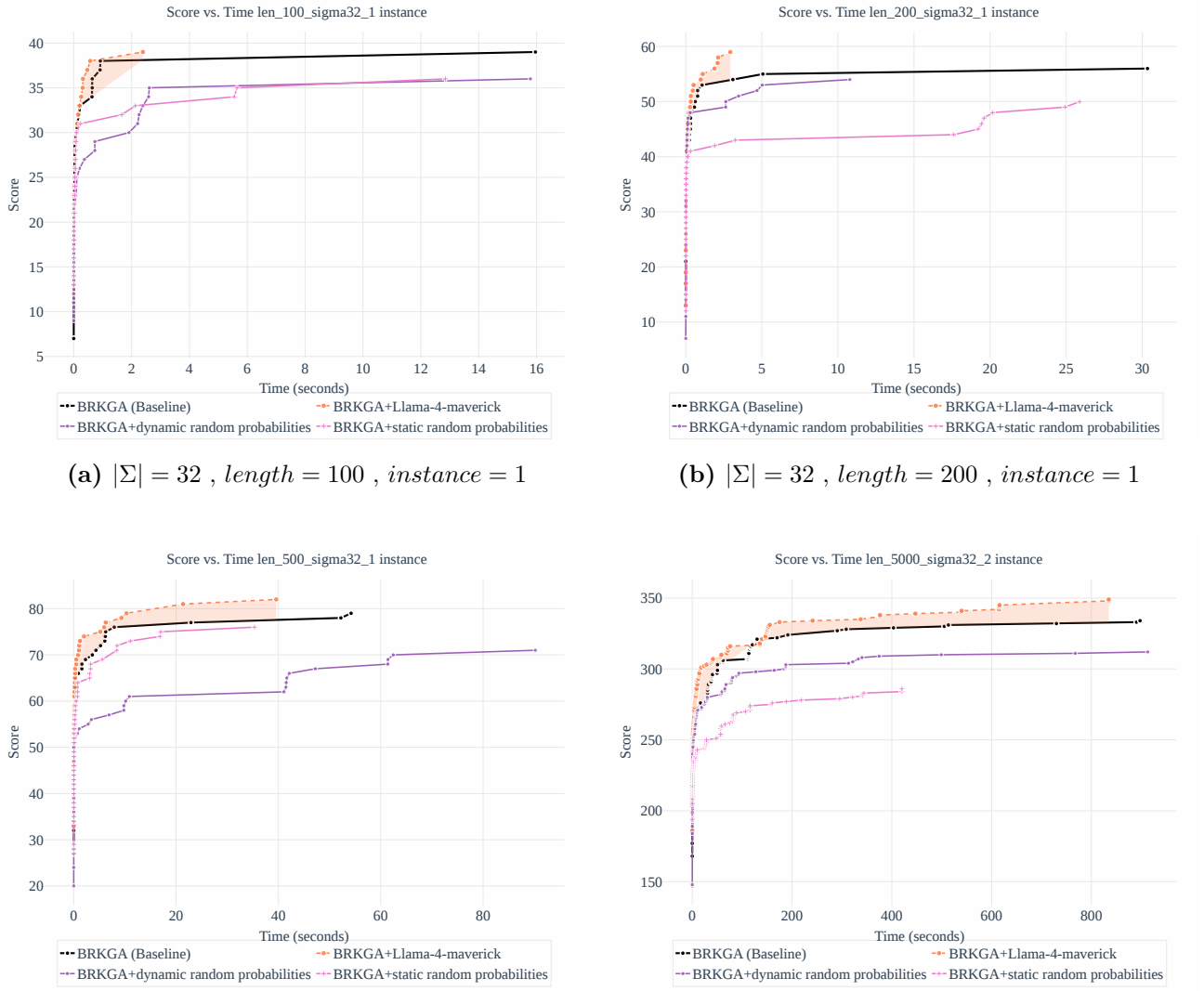
Tabla 6.8: Rendimiento del BRKGA integrado con LLM para el LRS, comparando la calidad de la solución con probabilidades aleatorias estáticas (generadas una vez antes de la ejecución) y dinámicas (actualizadas durante el tiempo de ejecución) para verificar que las mejoras obtenidas con las probabilidades generadas por el LLM no se deben simplemente al azar.

Length	$\|\Sigma\|$	**GPT-4.1-mini**			**Gemini-2.5-Flash**			**Llama-3.2-3b**			**Llama-4-Maverick**		
		LLM	Rand. Static	Rand. Dynamic	LLM	Rand. Static	Rand. Dynamic	LLM	Rand. Static	Rand. Dynamic	LLM	Rand. Static	Rand. Dynamic
100	2	**59.03**	58.93	**59.03**	**59.03**	58.93	**59.03**	**59.03**	58.93	**59.03**	**59.03**	58.93	58.93
	4	**41.07**	40.80	40.83	**41.07**	40.87	40.83	**41.07**	40.87	40.60	**41.07**	40.87	**41.07**
	8	**33.90**	33.13	32.70	33.87	33.00	32.60	33.87	33.03	32.97	**33.90**	33.27	32.83
	16	34.57	31.70	31.53	34.57	32.07	31.23	34.53	32.10	32.57	**34.60**	32.43	32.10
	32	40.40	34.10	33.43	40.93	34.93	34.33	40.83	34.30	34.40	**41.10**	35.60	35.10
200	2	**115.17**	115.10	**115.17**	**115.17**	115.10	115.00	**115.17**	115.10	115.13	**115.17**	115.10	115.03
	4	**72.77**	72.50	72.40	**72.77**	72.57	72.30	**72.77**	72.53	72.53	**72.77**	72.67	72.53
	8	55.10	54.07	53.90	**55.17**	54.20	54.07	55.10	54.10	54.00	55.13	54.13	53.93
	16	50.83	46.47	47.03	50.87	47.17	47.23	50.83	46.63	47.83	**51.03**	47.37	48.40
	32	55.37	46.40	47.20	54.50	47.17	45.70	55.13	46.23	47.20	**55.87**	47.77	47.37
300	2	**165.47**	165.27	165.43	**165.47**	165.43	165.33	**165.47**	165.27	165.37	**165.47**	165.43	**165.47**
	4	**102.73**	102.43	102.50	**102.73**	102.50	102.40	**102.73**	102.53	102.53	**102.73**	102.53	102.47
	8	75.07	73.30	73.57	**75.10**	73.67	73.47	75.07	73.33	73.70	**75.10**	73.77	73.60
	16	66.30	61.40	61.87	**66.77**	61.93	62.00	66.27	61.90	62.87	66.63	62.47	62.90
	32	66.03	57.00	56.90	66.50	58.07	56.90	66.47	57.77	58.13	**67.63**	59.23	58.93
500	2	**271.67**	271.50	**271.67**	**271.67**	271.50	271.60	**271.67**	271.50	271.63	**271.67**	271.50	**271.67**
	4	162.73	162.33	162.33	**162.77**	162.33	162.47	**162.77**	162.30	162.30	**162.77**	162.43	162.67
	8	109.93	108.40	108.27	**110.20**	108.60	108.40	110.13	108.57	108.37	**110.20**	108.70	108.67
	16	90.60	85.30	85.40	90.80	85.07	85.13	90.33	86.17	86.40	**91.17**	86.53	87.33
	32	85.50	74.43	74.70	85.77	75.37	74.50	86.07	74.97	75.47	**87.17**	77.27	75.70
1000	2	**530.23**	530.03	530.07	**530.23**	530.03	530.17	**530.23**	530.03	530.10	**530.23**	530.03	530.10
	4	**301.10**	300.53	300.27	**301.10**	300.50	300.60	301.00	300.37	300.80	301.00	300.60	301.03
	8	**191.87**	189.57	189.33	191.23	189.73	188.97	191.23	189.60	189.53	191.80	189.90	189.87
	16	142.90	137.27	137.63	143.33	138.17	136.83	142.77	137.53	137.80	**143.37**	138.13	138.43
	32	123.97	111.43	110.60	124.00	112.33	108.47	123.97	110.47	111.73	**126.23**	113.20	110.90
2000	2	**1041.73**	1041.70	1041.57	**1041.73**	1041.70	1041.67	**1041.73**	1041.63	1041.70	**1041.73**	1041.70	1041.70
	4	**572.67**	572.13	572.03	572.50	572.07	572.07	572.50	572.03	572.27	572.50	572.33	572.30
	8	343.17	340.63	340.90	343.10	340.87	339.87	343.70	340.83	340.73	**343.90**	340.50	342.23
	16	234.63	227.13	224.63	235.00	227.13	226.03	233.73	226.97	227.23	**235.10**	231.37	229.07
	32	186.80	167.77	168.67	188.13	167.53	166.57	186.23	167.97	168.73	**189.90**	171.17	170.80
5000	2	**2567.47**	**2567.47**	**2567.47**	**2567.47**	**2567.47**	2567.33	**2567.47**	**2567.47**	2567.30	**2567.47**	**2567.47**	2567.37
	4	**1361.30**	1360.70	1360.07	1361.17	1360.50	1360.80	**1361.30**	1360.47	1360.57	1361.17	1361.17	1360.67
	8	766.87	762.83	763.40	**768.50**	762.53	760.27	767.13	762.43	765.53	768.07	762.77	765.00
	16	483.57	470.90	473.83	481.33	471.67	471.53	483.90	468.60	473.97	**486.23**	474.97	477.23
	32	335.73	315.80	312.67	336.93	313.40	318.87	336.50	315.03	312.73	**338.40**	321.80	316.87
Average		312.52	308.30	308.26	312.61	308.46	308.13	312.53	308.27	308.69	**313.07**	309.29	309.15

Es evidente el gran desempeño del algoritmo híbrido con las probabilidades obtenidas mediante LLMs con respecto a las probabilidades aleatorias tanto estáticas como dinámicas, lo que valida la propuesta de Chacón et al. [2] sobre integrar conocimiento aprendido por modelos de lenguaje como guía heurística y respalda el enfoque explorado en esta investigación.

6.4.2. Comparación de convergencia

Finalmente se presenta el desempeño de los modelos híbridos con respecto al BRKGA clásico en algunas instancias específicas como comparación de convergencia a las soluciones.



(a) $|\Sigma| = 32$, $length = 100$, $instance = 1$

(b) $|\Sigma| = 32$, $length = 200$, $instance = 1$

(c) $|\Sigma| = 32$, $length = 500$, $instance = 1$

(d) $|\Sigma| = 32$, $length = 5000$, $instance = 2$

Figura 6.3: Comparación de puntaje vs tiempo en instancias específicas del dataset: BRKGA estándar vs hibridaciones con LLMs

Como se observa en la Figura 6.3, se analizó el comportamiento de las soluciones en instan-

cias con $|\Sigma| = 32$ para diferentes valores de length. Los resultados muestran que la hibridación BRKGA+Meta-Llama4Maverick logra consistentemente soluciones de mayor calidad que el BRKGA estándar (representado por el área sombreada bajo la curva) y otras hibridaciones basadas en probabilidades aleatorias, demostrando su superioridad en términos de puntaje final. Particularmente en instancias pequeñas (length=100 y length=200), como se aprecia en las Figuras 6.3a y 6.3b, el algoritmo propuesto alcanza la solución en tiempos significativamente menores, lo que evidencia su eficiencia en problemas de menor escala. Esta ventaja se mantiene en instancias más grandes, donde continúa superando a los demás enfoques evaluados.

6.5. Discusión de resultados

La ventaja del BRKGA resulta particularmente interesante al compararlo con algoritmos como CMSA o LCMSA, que adaptan la subinstancia a lo largo de toda la ejecución mediante un mecanismo de envejecimiento de componentes. En contraste, BARRAKUDA crea una subinstancia nueva en cada iteración, generando soluciones que permiten una exploración más dinámica del espacio de soluciones.

Este último enfoque parece ser especialmente beneficioso para el problema LRS, donde la capacidad de generar y mejorar soluciones de forma continua y diversa supera a las estrategias que se basan en la adaptación progresiva de una única subinstancia. Además, BARRAKUDA, siendo un método híbrido, combina esta perspectiva evolutiva con la resolución exacta de subproblemas, optimizando el tiempo al evitar almacenar y gestionar información redundante, lo que podría ralentizar la búsqueda.

La integración con modelos de lenguaje emergió como la línea más prometedora. Como se observa en la Figura 6.1, todas las variantes de BRKGA+LLM superaron al algoritmo base, con mejoras notables en instancias de mayor tamaño. El modelo Meta-Llama4Maverick demostró ser estadísticamente superior, alcanzando mejoras significativas en casos con$|\Sigma| \geq 16$.

Los tiempos de respuesta, analizados en la Figura 6.2, revelan que el tiempo computacional extra introducido por los LLMs resulta razonable considerando las mejoras obtenidas. Para instancias con length=5000, el tiempo adicional promedio fue de 4 segundos en modelos comerciales, contra 22 segundos en la versión gratuita.

Las pruebas con aleatorización confirmaron que las mejoras son consecuencia directa de la integración con LLMs. Los gráficos de convergencia en la Figura 6.3 muestran que las versiones de BRKGA con Llama4Maverick alcanzan soluciones de mayor calidad en menos tiempo, particularmente en instancias pequeñas.

Capítulo 7. Conclusiones y trabajo futuro

Esta exploración de distintos enfoques se enmarca en el desafío de la optimización combinatoria aplicada a problemas bioinformáticos, particularmente en el proceso de ensamblaje genómico para especies sin referencia. Como se estableció en la Sección 1, el problema LRS emerge como una herramienta crucial para el scaffolding genómico mediante homología, pero su naturaleza NP-hard demanda soluciones aproximadas eficientes. Las contribuciones en la mejora del algoritmo BRKGA mediante técnicas de hibridación con LLMs responden directamente a esta necesidad, ofreciendo un nuevo enfoque para abordar este desafío computacional en genómica.

La exploración realizada ha demostrado que la hibridación de BRKGA con modelos de lenguaje constituye una estrategia viable que puede ser aplicada para superar las limitaciones del estado del arte en optimización combinatoria para problemas bioinformáticos. Dos resultados destacan particularmente: primero, la confirmación experimental de que las variantes híbridas BRKGA+LLMs superan consistentemente al BRKGA tradicional en el contexto del problema LRS; y segundo, la implementación del modelo *Meta:Llama-4-Maverick* como nuevo estado del arte para el LRS, que mostró mejoras estadísticamente significativas con respecto a los demás algoritmos e hibridaciones. Estos resultados son consistentes con un artículo de revista científica actualmente en preparación, donde se documenta en detalle la metodología empleada y los hallazgos de esta investigación, con el objetivo de compartir estas contribuciones con la comunidad académica y promover nuevas líneas de trabajo en la intersección entre modelos de lenguaje y optimización combinatoria.

Los resultados obtenidos tienen importantes repercusiones. Proporcionan herramientas más eficientes para el ensamblaje de genomas no modelados, particularmente en especies menos estudiadas. Además, establecen un precedente para la aplicación de LLMs en problemas de optimización en biología computacional. Estas mejoras algorítmicas pueden acelerar significativamente investigaciones en genética evolutiva.

Como trabajo futuro se identifican principalmente la adaptación de este enfoque a otros problemas de ensamblaje genómico más allá del LRS, incluso fuera del área de bioinformática. Por otro lado, se propone profundizar en el estudio de metaheurísticas que dependen de una

heurística, como es el caso de ACO. En este trabajo, se diseñó una heurística mejorada que superó a la actual, pero su rendimiento aún presenta oportunidades de optimización. Dado el éxito observado en el uso de modelos de lenguaje para la ponderación de importancia de cada métrica, una dirección prometedora sería utilizarlos para el ajuste dinámico de heurísticas en cada instancia. Esto podría permitir una adaptación a las características específicas del problema, mejorando la calidad de las soluciones.

Bibliografía

[1] Riccardo Dondi and Florian Sikora. The Longest Run Subsequence Problem: Further Complexity Results. In Paweł Gawrychowski and Tatiana Starikovskaya, editors, *32nd Annual Symposium on Combinatorial Pattern Matching (CPM 2021)*, volume 191 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 14:1–14:15, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[2] Camilo Chacón Sartori, Christian Blum, Filippo Bistaffa, and Guillem Rodríguez Corominas. Metaheuristics and large language models join forces: Toward an integrated optimization approach. *IEEE Access*, 13:2058–2079, 2025.

[3] Christian Blum and Pedro Pinacho. A biased random key genetic algorithm for solving the longest run subsequence problem. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '25)*, GECCO '25, pages 1–9. Association for Computing Machinery, 2025.

[4] Chien-Feng Huang. Combinatorial optimization in biology using Probability Collectives Multi-agent Systems. *Expert Systems with Applications*, 39(2):1763–1771, 2012.

[5] Sven Schrinner, Manish Goel, Moritz Wulfert, et al. Using the longest run subsequence problem within homology-based scaffolding. *Algorithms for Molecular Biology*, 16(1):11, 2021. Open access article.

[6] Junwei Luo, Yawei Wei, Mengna Lyu, Zhengjiang Wu, Xiaoyan Liu, Huimin Luo, and Chaokun Yan. A comprehensive review of scaffolding methods in genome assembly. *Briefings in Bioinformatics*, 22(5):bbab033, 02 2021.

[7] Sven Schrinner, Manish Goel, Moritz Wulfert, Philipp Spohr, Korbinian Schneeberger, and Gunnar W. Klau. The longest run subsequence problem. In Carl Kingsford and Nadia Pisanti, editors, *Proceedings of the 20th International Workshop on Algorithms in Bioinformatics (WABI 2020)*, volume 172 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:13. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020.

[8] Zbigniew Michalewicz and David B. Fogel. *How to Solve It: Modern Heuristics*. Springer, 2000. ISBN 3540224947.

[9] Dominik Putz, Daniel Schwabeneder, Hans Auer, and Bernadette Fina. A comparison between mixed-integer linear programming and dynamic programming with state prediction as novelty for solving unit commitment. *International Journal of Electrical Power Energy Systems*, 125:106426, 2021.

[10] B. Pollack-Johnson. Hybrid structures and improving forecasting and scheduling in project management. *Journal of Operations Management*, 12(2):101–117, Feb 1995.

[11] Y. Wang. Review on greedy algorithm. *Theoretical and Natural Science*, 14:233–239, 2023.

[12] Mariana A. Londe, Luciana S. Pessoa, Carlos E. Andrade, and Mauricio G.C. Resende. Biased random-key genetic algorithms: A review. *European Journal of Operational Research*, 321(1):1–22, 2025.

[13] Mauricio G. C. Resende and Celso C. Ribeiro. *GRASP: The basic heuristic*, pages 95–112. Springer New York, New York, NY, 2016.

[14] Marco Dorigo, Mauro Birattari, and Thomas Stutzle. Ant colony optimization. *IEEE Computational Intelligence Magazine*, 1(4):28–39, 2006.

[15] Christian Blum, Pedro Pinacho, Manuel López-Ibáñez, and José A. Lozano. Construct, merge, solve and adapt a new general algorithm for combinatorial optimization. *Computers and Operations Research*, 68:75–88, 2016.

[16] Christian Blum. *Adding Learning to CMSA*, pages 71–93. Springer Nature Switzerland, Cham, 2024.

[17] Pedro Pinacho Davidson and Christian Blum. Barrakuda: A hybrid evolutionary algorithm for minimum capacitated dominating set problem. *Mathematics*, 8, 10 2020.

[18] Andreea Dréau, Manuel López-Ibáñez, and Thomas Stützle. Automatically improving the anytime behaviour of multiobjective evolutionary algorithms. pages 825–840, 03 2013.

[19] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

[20] Borja Calvo and Guzmán Santafé. scmamp: Statistical comparison of multiple algorithms in multiple problems. *The R Journal*, 8(1):248–256, 2016. R package.

Anexo A. Primer anexo

En este anexo se presenta la plantilla utilizada para obtener los parámetros α y β mediante la herramienta OptiPattern.

```
[BEGIN PROBLEM]
The Longest Run Subsequence (LRS) problem is defined as follows: Given an input string S over an
alphabet Σ, the goal is to extract a subsequence S* composed of entire runs from S, such that each
symbol from the alphabet appears in at most one run in S*, and the total length of S* is maximized. A
run is a maximal sequence of consecutive identical characters. The selected runs in S* must preserve
their original order in S, and cannot overlap.
[END PROBLEM]
[BEGIN EVALUATION GRAPH]
[BEGIN DATA]
[BEGIN DATA - FIELD DESCRIPTION]
- normalized_length: Length of the run divided by the total string length.
- opportunity: Estimated potential contribution of the run to the total LRS  1/(1+ gap). where gap=
next_run_start - start.
- distance_next: Normalized distance to the next occurrence of the same symbol.
- local_density: Frequency of the character in the entire string divided by its total length.
[END DATA - FIELD DESCRIPTION]
node,normalized-length,opportunity ,distance-next,local_density
{{graph_data_evalution}}
[END DATA]
[END EVALUATION GRAPH]
[BEGIN RULES ANSWERING]
Consider the following equation to assign a probability range to each node:
\text{Influence(N)} = \text{sigmoid}\left(\alpha_1 \cdot (1 - (\beta_1 - normalized-length)) + \alpha_2
\cdot (1 - (\beta_2 - opportunity)) + \alpha_3 \cdot (1 - (\beta_3 - distance-next)) + \alpha_4 \cdot
(1 - (\beta_4 - local_density))\right)
- Alpha: Represents the weighting coefficients assigned to each metric in the influence calculation.
The sum of all alpha values must equal 1 (\(\sum_{i=1}^{4} \alpha_i = 1\)), and each alpha value could
be different, inferring the proper distribution representing the importance among the metrics. Each
alpha coefficient (\(\alpha_{i}\)) is constrained to the range (0, 1).
- Beta: Represents a factor of desirable results for each metric, indicating their relative importance.
Each beta value (\(\beta_{i}\)) is independent and constrained to the range (0, 1). The beta value
indicates where the favorable values are expected relative to the metrics; it serves as a repair
parameter.
The task is to analyze the provided Evaluation Graph metrics and determine appropriate values for the α
(alpha) and β (beta) parameters that best reflect the influence of each node on the selection process
for the Longest Run Subsequence problem.
The response must be only in the following format:
alpha_1={{value_alpha_1}}
alpha_2={{value_alpha_2}}
alpha_3={{value_alpha_3}}
alpha_4={{value_alpha_4}}
beta_1={{value_beta_1}}
beta_2={{value_beta_2}}
beta_3={{value_beta_3}}
beta_4={{value_beta_4}}
[END RULES ANSWERING]
```

Figura A.1: Plantilla de la herramienta OptiPattern para cálculo de parámetros α y β en la función de influencia del LLM. La herramienta automatiza el proceso de optimización.

Anexo B. Segundo anexo

En este anexo se presenta el cálculo total de tokens (GPT) utilizado para obtener los valores alpha y beta en el dataset completo, es decir, se utilizaron 23 millones de tokens aproximadamente para obtener los valores alpha y beta de cada racha del dataset solo para un modelo LLM. Tabla fundamental para realizar el análisis de costos.

Tabla B.1: Cálculo de tokens utilizados para el dataset completo (150 instancias por length)

Longitud de instancia	**Tokens por instancia**	**Total tokens (150 instancias)**
100	1920	288 000
200	2980	447 000
300	4520	678 000
500	8360	1 254 000
1000	15 210	2 281 500
2000	33 721	5 058 150
5000	90 237	13 535 550
	Total tokens para el dataset	**23 542 200**
	Total tokens (en millones)	**23,54**

Anexo C. Tercer anexo

Prompt estructurado para obtener métricas relevantes sobre las rachas del LRS.

```
[Problem definition]
```

The Longest Run Subsequence problem (LRS) is defined as follows: Given an alphabet Σ and a string $S = s_1 s_2 \dots s_n$ where each $s_i \in \Sigma$, the goal is to find a longest run subsequence S^* of S such that every $\sigma \in \Sigma$ occurs in at most one run in S^* and the length $|S^*|$ is maximized over all run subsequences of S.

```
[End problem definition]


[Example instance]

    Input string S = abacacbbab

    Runs:
    start:letter:length
    0:a:1
    1:b:1
    2:a:1
    3:c:1
    4:a:1
    5:c:1
    6:b:2
    8:a:1
    9:b:1

    Optimal Runs selected:
```

```
    10110111
    10101111

    Optimal:
    s' = aaccbbb. score=7
    s' = aaacbbb. score=7

    Valid but not optimal:
    s' = aaabb. score=5
    s' = abbbb. score=5

[End example instance]

[Request]

    Provide five independent and non-overlapping metrics
    to evaluate or approximate solutions for the Longest
    Run Subsequence (LRS) problem. Each metric should be
    quantifiable, capturing a distinct aspect of the
    problem without relying on other proposed metrics.
    For example, existing metrics include Normalized
    length (run length divided by the total string length).

[End request]
```

Anexo D. Cuarto anexo

D.1. Declaración de Uso Ético de Herramientas de IA Generativa

Yo, Martín Isla Pino declaro que he utilizado herramientas de inteligencia artificial generativa, como ChatGPT, de manera ética y responsable para apoyar la realización de este trabajo. A continuación, se detalla específicamente el uso otorgado:

- ✓ **Redacción, Estructuración, Mejora del Texto y Corrección Ortográfica:** Uso de IA para reescribir ideas originales, organizar secciones, mejorar la coherencia, claridad, estilo, corregir errores gramaticales y ortográficos.
- ✓ **Traducción:** Uso de IA para traducir textos a distintos idiomas.
- ✓ **Generación de Ideas:** Uso de la IA como fuente de inspiración o para explorar enfoques novedosos en el desarrollo del trabajo. Siempre que se han utilizado ideas específicas provenientes de la IA, se ha citado adecuadamente su origen.
- ✓ **Asesoría Técnica o Conceptual:** Consulta sobre conceptos técnicos o metodológicos complejos. La información proporcionada por la IA ha sido revisada, contrastada y validada con fuentes académicas o científicas adecuadas para asegurar su precisión y pertinencia.

Declaro que todo contenido generado o asistido por IA ha sido revisado, adaptado y validado para asegurar su originalidad y pertinencia. Soy el único responsable del trabajo presentado y me comprometo a que las fuentes utilizadas sean debidamente citadas.