

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
SANTIAGO - CHILE



“DISEÑAR UNA LIBRERÍA DE MÉTRICAS DE
DESEMPEÑO PARA METAHEURÍSTICAS
POBLACIONALES”

JUAN PABLO CASSIS OYANADEL

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN INFORMÁTICA

Profesor Guía: Nicolas Rojas
Profesor Correferente: Elizabeth Montero

Noviembre - 2024

DEDICATORIA

Dedico esta memoria, el conocimiento y las horas de esfuerzo que tomó, a mis padres, quienes fueron fundamentales con su apoyo incondicional y su cariño. A mi pareja, quien lleva muchos años apoyándome en las buenas y en las malas, y espero que lo siga haciendo. A mis amigos, quienes fueron esa compañía constante, ese apoyo académico y social, y un importante espacio de distinción. Finalmente, a mis queridas mascotas, cuya compañía fue un pilar importante durante el desarrollo de mi carrera y esta investigación.

AGRADECIMIENTOS

Quiero agradecer, primero que nada, a mi madre, María Loreto, de quien aprendí todo lo que me representa como persona, quien ha sido mi mayor pilar de apoyo desde el primer día, y no ha fallado hasta el día de hoy. A mis padres, Félix y David, que, a sus tiempos y maneras, siempre han estado para mí cuando lo he necesitado y espero sientan lo mismo de mí. A mi hermano Antonio, que espero pueda rescatar algo de lo que le he intentado transmitir a lo largo de los años. A mis abuelos, Gladys y José Luis, Félix y Anamari, y Rocío y Jalil, quienes espero puedan sentir orgullo de lo que he logrado.

A mis amigos, algunos que traigo de etapas anteriores, como Nicolás Bravo, Ignacio Solís y Matías Morales. O aquellos que conocí en el camino, como Patricio Pérez, Juan Pablo Labbé, Matías González, Óscar Acevedo, Bastian Kretschmer y Nicolas Roman. E incluso, quienes me han integrado en sus vidas estos últimos meses, Roberto Artigues, Ana Vargas, Nicolás Soto, Alejandro Neira, y muchos más. Todos ellos, han representado mis momentos de relajo, entretenimiento y apoyo durante esta importante etapa y espero poder mantener sus amistades por un largo tiempo.

A mi pareja, Catalina, con quien he compartido incontables experiencias, quien me ha alentado a seguir adelante y forma una parte importante de por qué he llegado hasta acá.

Finalmente, quiero plasmar mi cariño y apreciación por Matilde, mi pequeña mascota que, solo con dormir a mis espaldas, ha sido mi mayor compañía en este proceso.

RESUMEN

Resumen— Las metaheurísticas son algoritmos que resuelven problemas complejos en tiempos limitados, ofreciendo soluciones de buena calidad, aunque sin garantizar el óptimo. Para medir su desempeño, se suelen usar métricas simples como la aptitud de las soluciones y el tiempo de ejecución. Usualmente, estas métricas simples no requieren de mayor desarrollo ni procesamiento, pero que carecen de análisis profundo sobre el algoritmo. Este trabajo presenta una librería que implementa métricas de desempeño para metaheurísticas poblacionales, analizando su comportamiento en sus etapas de exploración y explotación de la solución. Para evaluar la propuesta, se utilizaron tres algoritmos poblacionales: un algoritmo genético estándar (vGA) y dos variantes de un algoritmo genético rápido (fGA), las cuales fueron usadas en un análisis comparativo. Los experimentos validan estas métricas, la información que entregan y su interpretación para evaluar o comparar algoritmos a través de gráficos con datos históricos de la ejecución del algoritmo.

Palabras Clave— Métrica de rendimiento, Optimización mono-objetivo, Metaheurística poblacional, Librería

ABSTRACT

Abstract— Metaheuristics are algorithms that solve complex problems within limited timeframes, offering high-quality solutions, although without guaranteeing optimality. To measure their performance, simple metrics such as solution quality and execution time are often used. Usually, these metrics require little development or processing but lack in-depth analysis of the algorithm. This work presents a library that implements performance metrics for population-based metaheuristics, analyzing their behavior in the exploration and exploitation stages of the solution. To evaluate the proposal, three population-based algorithms were used: a standard genetic algorithm (vGA) and two variants of a fast genetic algorithm (fGA), which were used in a comparative analysis. The experiments validate these metrics, the information they provide, and their interpretation for evaluating or comparing algorithms through graphs with historical data from the algorithm's execution.

Keywords— Performance metric, Single-Objective optimization, Population-based metaheuristics, Library

GLOSARIO

CSOP: *Constraint Satisfaction Optimization Problems*

COP: *Constraint Optimization Problems*

GA: *Genetic Algorithm*

EA: *Evolutionary Algorithm* PSO: *Particle Swarm Optimization*

ACO: *Ant Colony Optimization*

ABC: *Artificial Bee Colony*

vGA: *Vanilla Genetic Algorithm*

FGA: *Fast Genetic Algorithm*

ÍNDICE DE CONTENIDOS

RESUMEN	IV
ABSTRACT	IV
GLOSARIO	V
ÍNDICE DE FIGURAS	VIII
ÍNDICE DE TABLAS	XI
INTRODUCCIÓN	1
CAPÍTULO 1: DEFINICIÓN DEL PROBLEMA	3
1.1 Contexto	3
1.2 Situación Actual	4
1.3 Objetivos	5
1.3.1 Objetivo principal	5
1.3.2 Objetivos específicos	5
CAPÍTULO 2: MARCO CONCEPTUAL	6
2.1 Conceptos	6
2.2 Clasificación de metaheurísticas	9
2.3 Metaheurísticas poblacionales	9
2.3.1 <i>Genetic Algorithm</i> (GA)	10
2.3.2 <i>Particle Swarm Optimization</i> (PSO)	13
2.3.3 <i>Ant Colony Optimization</i> (ACO)	14
2.3.4 <i>Artificial Bee Colony</i> (ABC)	17
2.4 Métricas de desempeño para metaheurísticas	17
2.5 Métricas de eficiencia	19
2.5.1 Convergencia	19
2.5.2 Diversidad	25
2.5.3 Costo computacional	30
2.6 Métricas de efectividad	31
2.6.1 Perfiles de desempeño	32
2.7 Resumen	33
CAPÍTULO 3: PROPUESTA DE SOLUCIÓN	35
3.1 Estrategia de solución	35
3.2 Ejecución de algoritmos y recolección de datos	35
3.2.1 Ejecución de metaheurísticas	37
3.2.2 Problemas disponibles	51
3.2.3 Recolección de datos	59
3.3 Librería de métricas para metaheurísticas poblacionales (LibMetMH-P)	60

3.3.1	Métricas de eficiencia	60
3.3.2	Métricas de operadores	74
3.4	Resumen	77
CAPÍTULO 4: VALIDACIÓN DE LA SOLUCIÓN		79
4.1	Configuración del ambiente	79
4.2	Elección de algoritmo y problemas	79
4.3	Resultados	80
4.3.1	Aptitud en el tiempo	80
4.3.2	<i>Rate of fitness change</i>	85
4.3.3	Convergencia	88
4.3.4	E-value	103
4.3.5	<i>Entropy Diversity</i>	105
4.3.6	Métricas de operadores	112
CAPÍTULO 5: CONCLUSIONES		120
REFERENCIAS BIBLIOGRÁFICAS		124

ÍNDICE DE FIGURAS

1	Movimiento <i>Bit-Flip</i> . [Abdel-Basset <i>et al.</i> , 2022]	8
2	Movimiento <i>Crossover</i> o cruzamiento en un punto. Elaboración propia	8
3	Taxonomía de metaheurísticas. [Dreo, 2007]	10
4	Taxonomía de metaheurísticas. [Singh <i>et al.</i> , 2022]	11
5	Taxonomía de metaheurísticas. [Elshaer y Awad, 2020]	11
6	Esquema de operadores y representaciones para GA. [Katoch <i>et al.</i> , 2021] . .	13
7	Movimiento de partículas en PSO. [Thevenot, 2022]	14
8	Comportamiento de colonias de hormigas. [Katona <i>et al.</i> , 2019]	15
9	Tasa promedio de convergencia para <i>synergistic fibroblast optimization</i> . [Dhivyaprabha <i>et al.</i> , 2018]	20
10	Estudio de tasa de convergencia basada en error consecutivo. [Halim <i>et al.</i> , 2021]	21
11	Estudio de tasa promedio geométrico de convergencia. [He y Lin, 2016] . . .	23
12	Estudio de eficiencia en redes de distribución de agua. [Mora-Meliá <i>et al.</i> , 2015]	24
13	Estudio de <i>Entropy Diversity</i> en PSO. [Pires <i>et al.</i> , 2013]	28
14	Estudio de diversidad en GAs basados en <i>Entropy Diversity</i> , su versión modificada y distancia entre pares de soluciones para problema TSP. [Nikfarjam <i>et al.</i> , 2021]	29
15	<i>Accuracy Profile</i> sobre 4 algoritmos y un mismo set de problemas. [Yan <i>et al.</i> , 2023]	33
16	Diagrama de interacción de componentes IOHprofiler. Elaboración propia . .	36
17	Instancia para <i>Maximum Independent Vertex Set</i> con $n = 10$ [Doerr <i>et al.</i> , 2020]	55
18	Función trampa con óptimo local en $u = 0$ pero óptimo global en $u = l$ [Deb y Goldberg, 1993]	56
19	Mejor aptitud encontrada para cada generación. Elaboración propia	61
20	Aptitud evaluada en cada evaluación. Elaboración propia	61
21	Cambio de aptitud entre generaciones. Elaboración propia	63
22	Cambio de aptitud para cada individuo por generación. Elaboración propia .	64
23	Tasa de convergencia con óptimo conocido. Elaboración propia	65
24	Error relativo con óptimo conocido. Elaboración propia	66
25	Convergencia sin uso de óptimo, por individuo y por generación. Elaboración propia	67
26	Convergencia sin óptimo adaptada a error relativo. Elaboración propia	68
27	<i>Geometric average convergence rate</i> . Elaboración propia	69
28	<i>Entropy diversity</i> . Elaboración propia	74
29	Tasas de éxito para operadores en GA. Elaboración propia	75
30	Tasas de mejora para operadores en GA. Elaboración propia	77
31	Tasas de deterioro para operadores en GA. Elaboración propia	77
32	Cambio de aptitud por evaluación para <i>OneMax</i> con $n = 100$ y $\lambda = 50$. Elaboración propia	81

33	Cambio de aptitud por evaluación para <i>ConcatenatedTrap</i> con $n = 100$ y $\lambda = 50$. Elaboración propia	81
34	Cambio de aptitud con mejor solución encontrada por generación para <i>OneMax</i> con $n = 100$ y $\lambda = 50$. Elaboración propia	82
35	Cambio de aptitud con mejor solución encontrada por generación para <i>ConcatenatedTrap</i> con $n = 100$ y $\lambda = 50$. Elaboración propia	82
36	Cambio de aptitud de FGA $F = 1,5$ por evaluación para N-reinas con $n = 64$ y $\lambda = 5$. Elaboración propia	83
37	Cambio de aptitud de FGA $F = 2$ por evaluación para N-reinas con $n = 64$ y $\lambda = 5$. Elaboración propia	83
38	Cambio de aptitud de FGA $F = 1,5$ por mejor solución encontrada por generación para N-reinas con $n = 64$ y $\lambda = 5$. Elaboración propia	84
39	Cambio de aptitud de FGA $F = 2$ por mejor solución encontrada por generación para N-reinas con $n = 64$ y $\lambda = 5$. Elaboración propia	84
40	Tasa de cambio de aptitud con mejor solución encontrada por generación para <i>N-queens</i> con $n = 625$ y $\lambda = 5$. Elaboración propia	85
41	Tasa de cambio de aptitud respecto del óptimo por individuo para <i>N-queens</i> con $n = 625$ y $\lambda = 5$. Elaboración propia	86
42	Tasa de cambio de aptitud respecto del óptimo por individuo para <i>N-queens</i> con $n = 625$ y $\lambda = 15$. Elaboración propia	86
43	Tasa de cambio de aptitud respecto del óptimo por individuo para <i>N-queens</i> con $n = 625$ y $\lambda = 50$. Elaboración propia	87
44	Tasa de cambio de aptitud respecto del óptimo por individuo para <i>Ising Triangular</i> con FGA $F = 1,5$, $n = 625$ y $\lambda = 15$. Elaboración propia	87
45	Tasa de cambio de aptitud respecto del óptimo por individuo para <i>Ising Triangular</i> con con FGA $F = 2$, $n = 625$ y $\lambda = 15$. Elaboración propia	88
46	Convergencia basada en óptimo para <i>OneMax</i> con $n = 625$ y $\lambda = 5$. Elaboración propia	89
47	Convergencia basada en óptimo para <i>OneMax</i> con $n = 625$ y $\lambda = 15$. Elaboración propia	90
48	Convergencia basada en óptimo para <i>OneMax</i> con $n = 625$ y $\lambda = 50$. Elaboración propia	90
49	Convergencia basada en óptimo para <i>OneMax + W</i> con $n = 625$ y $\lambda = 5$. Elaboración propia	91
50	Convergencia basada en óptimo para <i>OneMax + W</i> con $n = 625$ y $\lambda = 15$. Elaboración propia	91
51	Convergencia basada en óptimo para <i>OneMax + W</i> con $n = 625$ y $\lambda = 50$. Elaboración propia	91
52	Error relativo basado en óptimo para <i>OneMax + W</i> con $n = 625$ y $\lambda = 5$. Elaboración propia	92
53	Error relativo basado en óptimo para <i>OneMax + W</i> con $n = 625$ y $\lambda = 15$. Elaboración propia	92
54	Error relativo basado en óptimo para <i>OneMax + W</i> con $n = 625$ y $\lambda = 50$. Elaboración propia	93

55	Comparativa convergencia basada en óptimo para FGA en <i>OneMax</i> con $n = 625$ y $\lambda = 15$	94
56	Comparativa convergencia basada en óptimo para FGA en <i>Ising Torus</i> con $n = 625$ y $\lambda = 5$	94
57	Comparativo error relativo basado en óptimo para FGA en <i>OneMax</i> con $n = 625$ y $\lambda = 15$	95
58	Comparativo error relativo basado en óptimo para FGA en <i>Ising torus</i> con $n = 625$ y $\lambda = 5$	95
59	Convergencia sin óptimo por individuo para <i>OneMax</i> con $n = 64$ y $\lambda = 5$. Elaboración propia	96
60	Convergencia sin óptimo por individuo <i>OneMax</i> con $n = 64$ y $\lambda = 15$. Elaboración propia	96
61	Convergencia sin óptimo por individuo <i>OneMax</i> con $n = 64$ y $\lambda = 50$. Elaboración propia	96
62	Error relativo sin óptimo para <i>OneMax</i> con $n = 64$ y $\lambda = 5$. Elaboración propia	97
63	Error relativo sin óptimo para <i>OneMax</i> con $n = 64$ y $\lambda = 15$. Elaboración propia	97
64	Error relativo sin óptimo para <i>OneMax</i> con $n = 64$ y $\lambda = 50$. Elaboración propia	98
65	Convergencia con uso de óptimo para <i>Ising Triangle</i> con $n = 625$ y $\lambda = 5$. Elaboración propia	98
66	Convergencia sin óptimo para <i>Ising Triangle</i> con $n = 625$ y $\lambda = 5$. Elaboración propia	98
67	Comparativa convergencia sin óptimo para <i>concatenated trap</i> con $n = 625$ y $\lambda = 50$	99
68	Comparativa convergencia sin óptimo por individuo para <i>concatenated trap</i> con $n = 625$ y $\lambda = 50$	100
69	Comparativo error relativo con óptimo para <i>concatenated trap</i> con $n = 625$ y $\lambda = 50$	100
70	<i>Geometric rate of fitness change</i> para N-reinas con $n = 64$ y $\lambda = 5$. Elaboración propia	101
71	<i>Geometric rate of fitness change</i> para N-reinas con $n = 625$ y $\lambda = 5$. Elaboración propia	101
72	Comparativo <i>Geometric rate of fitness change</i> para <i>linear</i> con $n = 64$ y $\lambda = 50$.	102
73	Comparativo <i>Geometric rate of fitness change</i> para <i>ising torus</i> con $n = 64$ y $\lambda = 15$	103
74	<i>Entropy Diversity</i> para N-reinas con $n = 625$ y $\lambda = 5$. Elaboración propia . . .	106
75	<i>Entropy Diversity</i> para <i>Ising torus</i> con $n = 625$ y $\lambda = 5$. Elaboración propia . .	106
76	<i>Entropy Diversity</i> para N-reinas con $n = 625$ y $\lambda = 15$. Elaboración propia . .	107
77	<i>Entropy Diversity</i> para <i>Ising torus</i> con $n = 625$ y $\lambda = 15$. Elaboración propia .	107
78	<i>Entropy Diversity</i> para N-reinas con $n = 625$ y $\lambda = 50$. Elaboración propia . .	108
79	<i>Entropy Diversity</i> para <i>Ising torus</i> con $n = 625$ y $\lambda = 50$. Elaboración propia .	108
80	<i>Entropy Diversity</i> para <i>OneMax + W</i> con $n = 625$ y $\lambda = 15$. Elaboración propia	109
81	Comparativo <i>Entropy diversity</i> para N-reinas con $n = 64$ y $\lambda = 15$	110

82	Comparativo <i>Entropy diversity</i> para N-reinas con $n = 64$ y $\lambda = 50$	110
83	Comparativo <i>Entropy diversity</i> para <i>OneMax</i> + <i>W</i> con $n = 625$ y $\lambda = 15$	111
84	Comparativo <i>Entropy diversity</i> para <i>OneMax</i> + <i>W</i> con $n = 625$ y $\lambda = 50$	111
85	Tasa de éxito por operador para <i>Ising torus</i> con $n = 64$ y $\lambda = 5$. Elaboración propia	112
86	Tasa de éxito por operador para <i>Ising torus</i> con $n = 64$ y $\lambda = 50$. Elaboración propia	113
87	Tasa de éxito por operador para <i>Ising torus</i> con $n = 5625$ y $\lambda = 5$. Elaboración propia	113
88	Tasa de éxito por operador para <i>Ising torus</i> con $n = 5625$ y $\lambda = 50$. Elaboración propia	114
89	Comparativo de tasa de éxito por operador para N-reinas con $n = 64$ y $\lambda = 50$.	114
90	Comparativo de tasa de éxito por operador para <i>OneMax</i> con $n = 625$ y $\lambda = 15$	115
91	Comparativo de tasa de éxito por operador para <i>Ising triangular</i> con $n = 625$ y $\lambda = 15$	115
92	Tasa de mejora por operador para LABS con $n = 64$ y $\lambda = 15$. Elaboración propia	116
93	Tasa de deterioro por operador para LABS con $n = 64$ y $\lambda = 15$. Elaboración propia	116
94	Tasa de mejora por operador para <i>OneMax</i> + <i>W</i> con $n = 64$ y $\lambda = 15$. Elaboración propia	117
95	Tasa de deterioro por operador para <i>OneMax</i> + <i>W</i> con $n = 64$ y $\lambda = 15$. Elaboración propia	117
96	Comparativo de tasa de mejora por operador para <i>OneMax</i> con $n = 625$ y $\lambda = 15$	118
97	Comparativo de tasa de deterioro por operador para <i>concatenated trap</i> con $n = 625$ y $\lambda = 15$	118
98	Comparativo de tasa de mejora por operador para <i>concatenated trap</i> con $n = 625$ y $\lambda = 15$	119
99	Comparativo de tasa de deterioro por operador para <i>concatenated trap</i> con $n = 625$ y $\lambda = 15$	119

ÍNDICE DE TABLAS

1	Evaluaciones y valores evaluados para ejecución de GA genérico. Elaboración propia.	22
2	Valores convergencia y convergencia promedio geométrico	22
3	Parámetros de ejecución de los algoritmos. Elaboración propia.	40
4	Comparación de Algoritmos. Elaboración propia.	50
5	Problemas de optimización y sus características principales	58
6	Métricas de rendimiento propuestas	78
7	Comparación de E-value para <i>Ising ring</i>	103

8	Comparación de E-value para <i>OneMax</i> utilizando FGA	104
9	Comparación de E-value para <i>OneMax + W</i> utilizando FGA	105
10	Comparación de E-value para <i>Ising triangular</i> utilizando FGA	105

INTRODUCCIÓN

En el área de la optimización combinatorial, el estudio de nuevos algoritmos y su comportamiento en distintas situaciones ha sufrido una falta de información con la cual realizar estudios completos e informativos sobre estos algoritmos. Esta problemática nace a partir de la dificultad que implica la implementación de distintas métricas de desempeño para estos algoritmos, teniendo en cuenta que estos pueden variar en múltiples aspectos clave a evaluar. Por esto es que, en general, los estudios en esta área se basan en utilizar métricas como el tiempo de ejecución de un algoritmo o la aptitud de las soluciones que encuentra. Para ampliar esta gama de métricas es que se propone el desarrollo de una librería estandarizada de métricas de desempeño para metaheurísticas. Particularmente, en esta memoria, el desarrollo se enfoca en metaheurísticas poblacionales, es decir, aquellos algoritmos que hacen uso de metaheurísticas sobre un conjunto de múltiples soluciones a través de distintas generaciones.

Para comenzar, el capítulo 1 describe y plantea el problema a resolver, donde se describe el contexto en el cual se encuentra la optimización combinatorial, incluyendo aspectos clave de esta área de estudio, tales como, la definición de los problemas a resolver y sus características, las distintas categorías de algoritmos estudiados y los mecanismos con los que se evalúa su aptitud. Además, se explora brevemente el sujeto de esta investigación, las métricas de desempeño para metaheurísticas, sus categorías y definiciones de estas.

Luego, en el capítulo 2 se presenta la investigación realizada para el consiguiente desarrollo de esta memoria. En esta investigación, a modo de marco conceptual, se presentan conceptos clave como las definiciones matemáticas de un problema combinatorial y los mecanismos que utilizan los algoritmos que los resuelven, conocidos como movimientos. Además, se presentan distintos tipos de metaheurísticas poblacionales, tales como *Genetic Algorithm* (GA) y *Ant Colony Optimization* (ACO). Finalmente, se realiza una investigación sobre distintas métricas de desempeño encontradas en la literatura, incluyendo aquellas en las categorías de eficiencia, efectividad y de costo computacional.

En el tercer capítulo de este escrito se encuentra la propuesta de solución, donde se describe la implementación de una librería de métricas de desempeño para metaheurísticas poblacionales, con las definiciones matemáticas y pseudocódigo asociado para cada métrica, en conjunto a casos que ejemplifiquen y demuestren su funcionamiento. Adicionalmente, se describe el *framework* utilizado como base para la obtención de datos de distintas ejecuciones de metaheurísticas por sobre un gran conjunto de problemas, *IOHprofiler*.

En el cuarto capítulo se presentan los resultados obtenidos de la ejecución de la librería de métricas, demostrando los distintos casos que se pueden obtener a partir del análisis de distintos problemas sobre un algoritmo del tipo GA. Además, se presentan resultados obtenidos para un caso comparativo sobre un mismo algoritmo con variaciones en sus parámetros, con lo que se realiza un análisis sobre como estos afectan su ejecución y las soluciones que este

obtiene, simulando así, un caso de estudio real con las métricas proporcionadas.

Finalmente, en el quinto capítulo se presentan las conclusiones obtenidas de este estudio. Donde se analiza lo logrado con este estudio, el valor que este puede entregar a futuras investigaciones y consideraciones para el trabajo futuro que se pueda realizar por sobre lo desarrollado en esta memoria.

CAPÍTULO 1

DEFINICIÓN DEL PROBLEMA

1.1. Contexto

La optimización de problemas combinatoriales es un área del conocimiento que, formalmente, se estudia desde hace más de 70 años. En su definición inicial, un problema combinatorial es aquel donde se tiene un grupo de variables a los cuales se debe asignar un valor dentro de un dominio definido, tal de que los conjuntos resultantes cumplan con un conjunto de restricciones [Chowla y Ryser, 1950]. Este tipo de problemas es denominado *Constraint satisfaction problems* (CSP) y tienen por objetivo encontrar un subconjunto de asignaciones para estas variables de tal forma que satisfagan todo el conjunto de restricciones impuesto [Kumar, 1992].

Por otra parte, existen los *Constraint Satisfaction and Optimization Problems* (CSOP), donde se agrega una función objetivo a optimizar. Las restricciones definidas se dividen en dos (aquellas que definen si una solución es factible), restricciones duras, y (aquellas que, al no cumplirse, afectan el que tan buena sea la solución encontrada) restricciones blandas [Kearfott, 1996]. Así, se requiere satisfacer las restricciones duras y minimizar la cantidad de restricciones blandas no cumplidas.

Para la resolución de estos problemas se han desarrollado una variedad de algoritmos, existen dos categorías principales de algoritmos: los algoritmos de búsqueda local y los algoritmos de búsqueda global. Los algoritmos de búsqueda local se enfocan en explorar soluciones cercanas a una solución inicial, moviéndose iterativamente entre soluciones vecinas con el objetivo de mejorar progresivamente el resultado [Johnson *et al.*, 1989], estos movimientos se realizan de forma probabilística o aleatoria, lo que les otorga un carácter estocástico. Por otro lado, los algoritmos de búsqueda exhaustiva exploran el espacio de soluciones de manera que se encuentren todas las soluciones o bien, se determine si existe solución que satisfaga todas las restricciones [Papadimitriou y Steiglitz, 2013].

Así, los algoritmos de optimización combinatorial han sido un gran campo de investigación a lo largo de los años, el cual se ha aplicado extensivamente en el contexto de los CSP y CSOP. Posteriormente, desde la década de los años 70, el desarrollo e investigación de algoritmos de búsqueda local ha crecido y tomado un rol importante, de forma que hoy en día son altamente utilizados constantemente en áreas como la ingeniería eléctrica, las cadenas de abastecimiento, el ruteo de vehículos y muchas más [Adetunji *et al.*, 2021, Golden *et al.*, 1998, Griffis *et al.*, 2012].

Previo al uso de estos algoritmos en CSOP, se debe diseñar un mecanismo para evaluar las soluciones que se obtienen de estos. Para esto existe la función de evalua-

ción (en algunos casos definida de igual forma que la función objetivo del problema de interés), componente responsable de definir la aptitud de la solución encontrada [Aguilar-Fuster y Rubio-Loyola, 2021]. Es este factor el que comúnmente se ha utilizado como métrica para medir la calidad de un algoritmo, no entregando una mirada completa al desempeño de estos, sobretodo teniendo en cuenta que, dada su naturaleza estocástica, introduciendo cierta aleatoriedad a las decisiones tomadas, es necesario entender y medir el comportamiento del algoritmo para llegar a las soluciones.

1.2. Situación Actual

Dada la necesidad de comparar métodos de resolución para estos problemas, es que surgen distintas métricas de desempeño a partir de dos conceptos principales, la eficiencia y la efectividad [Bartz-Beielstein, 2005]. El primer concepto está asociado a los tiempos de cómputo de solución, la velocidad de convergencia y los recursos utilizados para la ejecución del algoritmo, mientras que el concepto de efectividad es asociado a la capacidad de encontrar soluciones adecuadas en términos de su aptitud, cuantas de estas son encontradas y el análisis estadístico capaz de juzgar lo significativo de los resultados [Halim *et al.*, 2021].

Incluso con una gran variedad de métricas de desempeño introducidas a lo largo de los años, estas siguen siendo implementadas manualmente por los desarrolladores que producen y utilizan estos algoritmos, lo cual lleva a que los trabajos realizados por distintos investigadores sean, en gran parte, difíciles de comparar y evaluar [Silberholz y Golden, 2010]. Además, esto introduce un esfuerzo no menor por parte de los investigadores al adaptar e implementar estas métricas.

Por lo tanto, el problema a resolver es la falta de homogeneidad sobre la evaluación de algoritmos en esta área. El objetivo es facilitar la comparación de algoritmos utilizando métricas de desempeño que consideren más información que solo la calidad de sus soluciones.

En este trabajo se busca analizar las métricas existentes implementadas y utilizadas en el Estado del Arte, para así desarrollar un sistema y metodología que permita aplicarlo a algoritmos de optimización combinatorial. Particularmente, en esta memoria se trabajarán únicamente en metaheurísticas poblacionales para problemas mono-objetivo, es decir, metaheurísticas que trabajan con una población de soluciones por cada iteración para resolver problemas con una única función a optimizar.

1.3. Objetivos

1.3.1. Objetivo principal

Desarrollar una librería de métricas de desempeño para metaheurísticas poblacionales en problemas combinatoriales mono-objetivo.

1.3.2. Objetivos específicos

- Estudiar el Estado del Arte respecto de métricas de desempeño existentes para metaheurísticas
- Diseñar e implementar métricas de desempeño para metaheurísticas
- Evaluar y comparar las métricas implementadas en algoritmos poblacionales (*Genetic algorithm*, *Particle Swarm Optimization*, etc) en instancias benchmark.

CAPÍTULO 2

MARCO CONCEPTUAL

El área de los problemas combinatoriales y los algoritmos asociados a la resolución de estos ha sido estudiada por decenas de años, incluso, se puede rastrear los orígenes de estos problemas hasta 1784 con el problema de asignación de [Monge, 1781], por parte de los algoritmos, podemos ir hasta la definición del algoritmo símplex de [Dantzig, 1990].

Las investigaciones posteriores en esta área se han aplicado en diversos problemas del mundo real, tales como el ruteo de vehículos [Jozefowicz *et al.*, 2008], la asignación de turnos de trabajo [Morris y Showalter, 1983], la asignación de espacios de trabajo [Pereira *et al.*, 2010], y mucho más. Esto ha llevado a una reducción de costos económicos, de tiempo, mayor satisfacción laboral, entre otros.

Para tener un entendimiento completo de la tarea a resolver en esta investigación, se presenta en este capítulo un estudio de los algoritmos, problemas y métricas de desempeño encontradas en la literatura.

El estudio de *Constraint Programming* (CP) es aquel enfocado en la resolución de problemas complejos delimitados por restricciones, siendo una restricción una relación entre una o más variables en un definido dominio [Leite *et al.*, 2014].

El desarrollo de buenos algoritmos para la resolución de estos problemas es de suma importancia, ya que son aplicados en diversas áreas de producción e investigación. Así, entregar herramientas capaces de medir las capacidades de un algoritmo es una tarea sumamente relevante, por lo cual se han desarrollado distintas métricas de desempeño con este objetivo en mente, lo que conduce naturalmente a la tarea de disponibilizarlas y adecuarlas para su uso directo en la evaluación de estos algoritmos.

2.1. Conceptos

Los problemas de optimización combinatorial suelen nacer de una necesidad natural, como la organización del espacio de un edificio, o más cercano al mundo animal, la búsqueda de una ruta ideal hacia fuentes alimenticias. El primero de estos se asocia a un problema de asignación de espacio [Yang *et al.*, 2017], el cual tiene por objetivo principal reducir el costo asociado a la entrega de espacios a distintos grupos de personas en un espacio limitado. Mientras, el segundo problema propuesto es generalizable a un problema de vendedor viajero (TSP por sus siglas en inglés), el cual se enfoca en la búsqueda de una ruta ideal a través de un conjunto de nodos, de tal forma que todos estos sean visitados al final de la ruta [Hoffman *et al.*, 2013].

Estos problemas pueden ser incluso generalizables hacia el problema de asignación simple,

el cual es definido como: Dada una matriz de costo C de tamaño n^2 , se debe encontrar una permutación π de $1, \dots, n$ para la cual

$$\sum_{i=1}^n c_{i,\pi(i)} \quad (1)$$

sea mínimo [Schrijver, 2005]. Una instanciación completa de estos problemas consiste en asignar un conjunto de valores específicos a las variables del problema. A este problema se le puede agregar un conjunto de restricciones que obligatoriamente deben ser satisfechas, definiendo lo que es una instanciación completa factible de las variables (Restricciones duras), u otras que sean deseables de satisfacer y podrían modificar el valor calculado de la función a optimizar (Restricciones blandas), las cuales no deciden sobre la factibilidad de la solución.

Siguiendo la exploración de los conjuntos de soluciones, dentro de las soluciones candidatas se encuentra un conjunto de soluciones que se consideran óptimos locales y un conjunto aún más selecto de óptimos globales, donde el primer concepto se asocia a aquellas soluciones donde el vecindario inmediato (a la distancia de un movimiento) a su alrededor empeora la solución, más no es la mejor solución posible. Mientras que un óptimo global sí cumple con ser la mejor solución en el espacio de búsqueda en términos de su función objetivo.

Teniendo una definición básica del problema de asignación, este se puede transformar en una gran variedad de problemas estandarizados, de forma más directa, se puede encontrar la ya mencionada asignación de turnos [Morris y Showalter, 1983]. Extendiendo lo que se entiende por asignación y costo hacia la selección y distancia de una ruta, se define el TSP, necesitando de un grafo no dirigido con V su conjunto de vértices y E su conjunto de arcos y x_{ij} si es que el arco encontrado en ij seleccionado y c_{ij} la función de costo (distancia) asociada tenemos su función objetivo, la cual le dará valor a la solución:

$$\min \sum_{e \in E} c_e X_e \quad (2)$$

con $e = \{i, j\} \in E, i \in V, j \in V$. Esta ecuación está sujeta a la restricción que todo vértice debe ser visitado una única vez en algún momento, demarcando la separación entre soluciones candidatas y aquellas que no lo son.

Para la resolución de estos problemas, se presenta una forma de guiar ordenadamente la búsqueda de soluciones, esta guía se aplica en forma de un movimiento definido. Dicho movimiento es una operación que modifica la solución de forma predefinida, las soluciones generadas a partir de aplicar este movimiento sobre una solución forman el conjunto llamado vecindario. Por ejemplo, en ciertas implementaciones se puede apreciar el movimiento "Bit-flip". Este movimiento involucra modificar una variable binaria de la solución propuesta, generando un vecindario de soluciones basado en el intercambio del valor de cada variable

por su contraparte [Rodríguez Rueda *et al.*, 2009]. Un ejemplo de este movimiento se puede encontrar en la figura 1, donde se observa la cuarta variable siendo modificada, cambiando su valor desde 1 a 0. Otra forma de modificar el conjunto de variables es el combinar secciones de dos o más soluciones, este procedimiento, conocido como *crossover*, se considera una forma de perturbar la solución, la cual no utiliza el vecindario de la solución como tal, si no que selecciona un punto aleatorio sobre el cual ejecutar el movimiento. Por ejemplo, si se tiene un conjunto de variables de longitud i , se elige un número aleatorio x entre 0 y $i - 1$ para dividir el conjunto. A partir de esta división, se selecciona, para una solución, una sección desde la variable en la posición 1 hasta la variable en la posición x , y para una solución distinta, se selecciona desde la variable x hasta la variable i . Con esto, se unen los sectores de distintas soluciones para formar una nueva. La figura 2 muestra este proceso de manera gráfica.

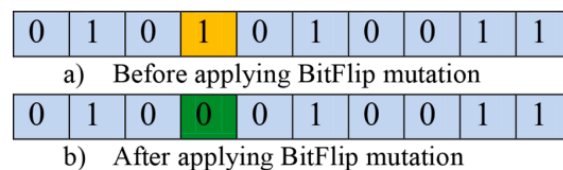


Figura 1: Movimiento *Bit-Flip*. [Abdel-Basset *et al.*, 2022]

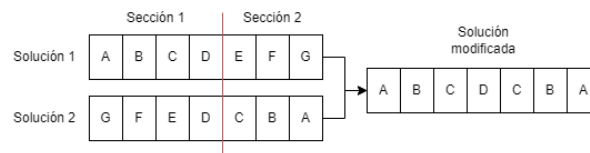


Figura 2: Movimiento *Crossover* o cruzamiento en un punto. Elaboración propia

Los algoritmos de búsqueda local se formulan a partir de los conceptos previamente expuestos. Estos algoritmos buscan la resolución de problemas de optimización a través del uso de movimientos sobre una solución inicial encontrada de manera independiente, donde dicha solución será modificada por una del vecindario generado por el movimiento. La idea es, iterativamente, mejorar el valor de la función objetivo que define la aptitud del problema [Johnson *et al.*, 1988]. Esta búsqueda puede ser guiada por las heurísticas y metaheurísticas.

El término de heurística se introduce en el área de la investigación operacional, y en este contexto se define como 'un algoritmo iterativo que no converge hacia la solución del problema' [Müller-Merbach, 1981], en otras palabras, un algoritmo el cual no produce una solución óptima o incluso factible para resolver el problema. Sin embargo, estos algoritmos proveen de una forma rápida para llegar a algún resultado, independiente de la calidad percibida de este. Estos métodos son desarrollados para problemas específicos o similares entre sí, tales como el *Quadratic assignment problem* [Burkard, 1975] y *Flow shop scheduling* [Liesegang y Schirmer, 1975].

Los métodos metaheurísticos, tales como *Tabu Search* [Glover, 1986], *Genetic algorithm* [Holland, 1975] y *Ant colony optimization* [Dorigo et al., 2006] son aquellos que permiten encontrar soluciones cercanas al óptimo en tiempos acotados. Por su definición en griego, “meta” es algo “más allá”, por lo tanto, una metaheurística es aquel método que realiza las cualidades de una heurística [Beheshti y Shamsuddin, 2013]. Así, [Voß et al., 2012] define una metaheurística como un proceso que modifica y guía las operaciones de heurísticas subordinadas para producir soluciones de alto nivel de aptitud. A diferencia de las heurísticas más prevalentes en los inicios del estudio de estos algoritmos, las metaheurísticas son pensadas como métodos aplicables agnósticos al problema en sí.

2.2. Clasificación de metaheurísticas

Dentro del mundo de las metaheurísticas, existen distintas categorías para clasificarlas, dichas categorías se pueden resumir en dos principales, las metaheurísticas poblacionales y trayectoriales.

Las metaheurísticas de trayectoria son aquellas que trabajan una única solución por iteración, explorando el vecindario de soluciones para encontrar nuevas soluciones y así obtener una solución final apropiada. Dentro de esta categoría se encuentran algoritmos como *Tabu Search* o GRASP [Barbarosoglu y Ozgur, 1999][Ferrer et al., 2016].

Por otra parte, existen las metaheurísticas poblacionales. Estos algoritmos trabajan sobre un conjunto de soluciones en cada iteración, las cuales tienen su enfoque en la modificación de este conjunto de soluciones al utilizar elementos y conocimiento obtenidos del mismo, así, obteniendo una o más soluciones con buen potencial. Usualmente, este enfoque permite mayor exploración del espacio de búsqueda en comparación a algoritmos trayectoriales.

La figura 3 presenta una taxonomía de metaheurísticas, donde podemos observar distintos subconjuntos, tales como aquellos inspirados por fenómenos naturales, algoritmos que no utilizan conocimiento previo para sus soluciones y más.

2.3. Metaheurísticas poblacionales

Una característica sumamente relevante de este tipo de algoritmos es que sacrifican una fracción del tiempo de ejecución y recursos computacionales, gracias a esta población de soluciones. Sin embargo, esto permite que el algoritmo use dos enfoques, la exploración global y la explotación global [Beheshti y Shamsuddin, 2013], o en otras palabras, al tener un conjunto de soluciones, se explora el espacio de búsqueda de forma mucho más completa y efectiva que si se manejara solo una de ellas. Y además, para cada solución candidata, se aumenta iterativamente la idoneidad hacia los máximos locales, obteniendo un conjunto de soluciones de alta aptitud y con gran diversidad en asignaciones.

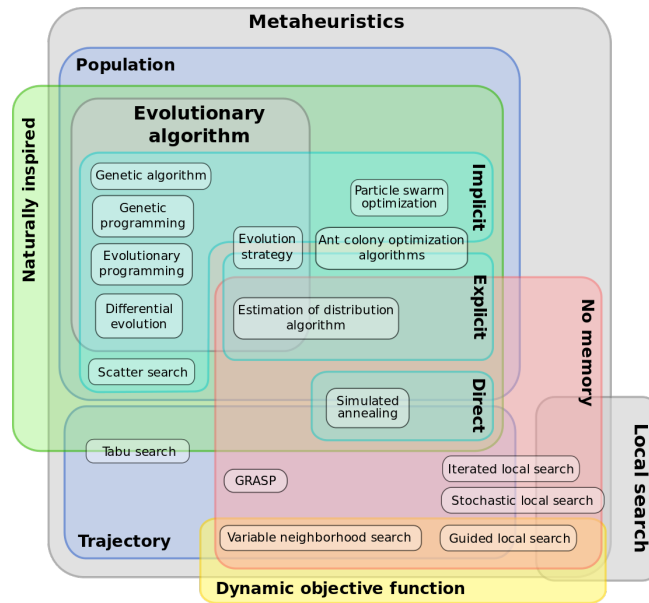


Figura 3: Taxonomía de metaheurísticas. [Dreo, 2007]

A su vez, las metaheurísticas poblacionales pueden ser clasificadas en distintas subcategorías. En particular, a partir de las figuras 3 y 4, se puede ver como las metaheurísticas poblacionales son inspiradas en su totalidad por fenómenos naturales, subdividiéndose en el origen del fenómeno (Químico, físico o biológico) y en el manejo de la población. Aquí, se puede encontrar la subcategoría de algoritmos evolutivos, donde ciertos conjuntos de la población se ven favorecidos por sobre otros para pasar a siguientes generaciones. También se tiene el subconjunto de algoritmos basados en enjambres, donde el conjunto de soluciones suele compartir información para su convergencia.

Al analizar distintas taxonomías (Figuras 3, 4 y 5) se puede concluir que una gran mayoría de las metaheurísticas poblacionales desarrolladas son basadas en fenómenos naturales, en particular destacando las subcategorías de algoritmos evolutivos y aquellos de inteligencia de enjambre. A continuación, se exponen distintas metaheurísticas poblacionales y sus mecanismos de búsqueda.

2.3.1. Genetic Algorithm (GA)

Este algoritmo es parte de la subcategoría de algoritmos evolutivos, ya que emplea mecanismos encontrados en teoría evolutiva de la biología, en particular, la "supervivencia del más fuerte" (o el más apto).

Para este algoritmo, la representación de la solución suele ser un *string* con carácter de este representando los genes (variables) del individuo (solución). Estas representaciones pueden ir desde una representación binaria (cada gen es un 0 o 1) a incluso una representación de

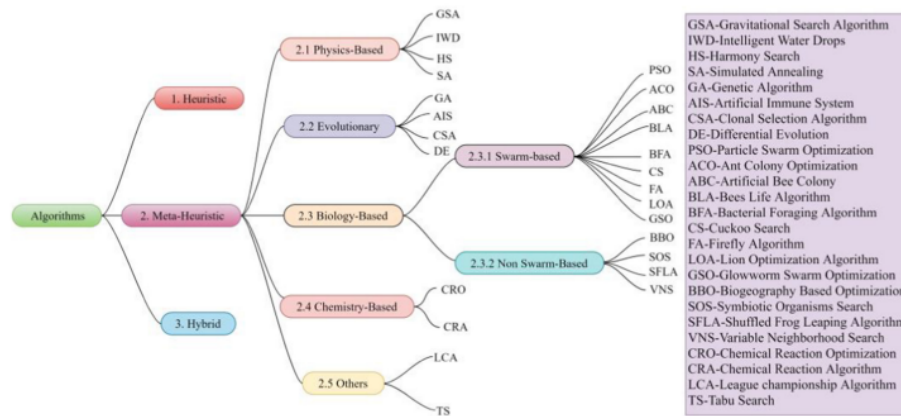


Figura 4: Taxonomía de metaheurísticas. [Singh et al., 2022]

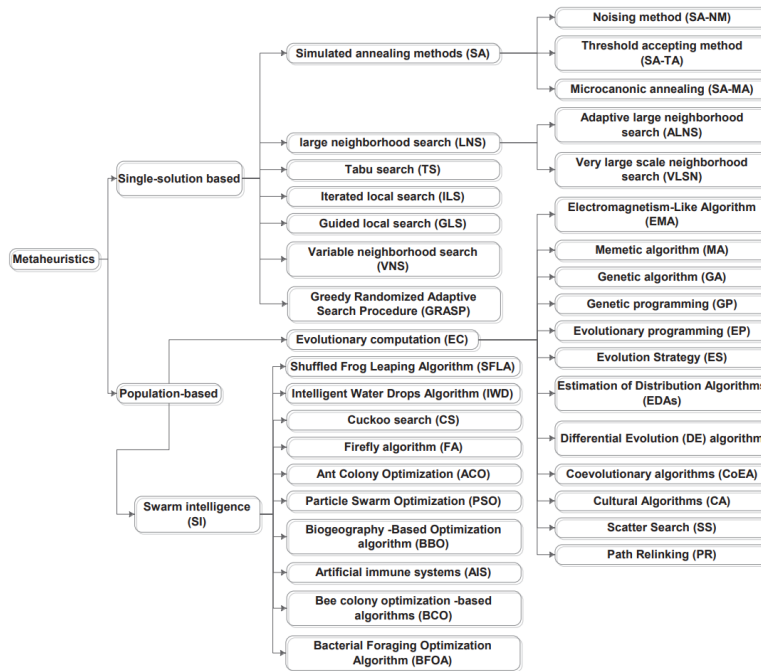


Figura 5: Taxonomía de metaheurísticas. [Elshaer y Awad, 2020]

árbol donde cada gen desencadena procesos y funciones para computar su valor.

En particular, un GA genérico funciona de la forma observada en el algoritmo 1, donde se inicializa una población inicial de soluciones y sus aptitudes (obtenidas de la función objetivo, o de evaluación). Luego, una vez que se comienza a iterar, se realiza una selección de un par de individuos (soluciones) S_1 y S_2 , esta selección se puede hacer a través de distintos métodos, tales como la selección elitista por ranking, o un método semi-elitista como el torneo o incluso, una selección estocástica utilizando el método de ruleta.

Con las soluciones a trabajar seleccionadas, se procede con el primer operador, *crossover*, esta operación involucra la selección parcial de elementos de S_1 y S_2 para formar una nueva solución, la figura 2 ilustra un *single point crossover*, dado que se utiliza una división clara de la cual tomar secciones de cada solución. Luego, se aplica un segundo procedimiento, *mutation*, como lo dice su nombre, este operador busca la modificación de la solución, ya sea a través de la permutación de una variable, modificando el orden en el que los valores de las variables están definidos, o invirtiendo los valores de variables. Ambos operadores de modificación están sujetos a una probabilidad de ser ejecutados para cada par de soluciones seleccionados [Katoch et al., 2021].

Finalmente, se ejecuta el proceso de reemplazo de la población, el cual puede realizarse de distintas maneras, ya sea a través de una selección que solo permita reemplazar soluciones que sean de mayor aptitud, realizar un reemplazo completo de la población, o aplicar distintos criterios propios del algoritmo. El proceso de selección, *crossover*, *mutation* y reemplazo se repite hasta formar una población descendiente completa [Lambora et al., 2019].

Algorithm 1 Genetic Algorithm

```
1: begin
2: Generar población inicial con  $n$  soluciones  $S_i$  ( $i = 1, 2, \dots, n$ )
3: Calcular aptitud para cada solución inicial
4: Contador de iteraciones  $t = 0$ 
5: Máximo de iteraciones  $MAX$ 
6: while  $t < MAX$  do
7:   Seleccionar un par de soluciones de la población inicial basado en la aptitud
8:   Aplicar la operación de crossover en el par seleccionado con probabilidad de crossover
9:   Aplicar operador mutation en la descendencia con probabilidad de mutation
10:  Reemplazar la población antigua con la nueva población generada
11:   $t = t + 1$ 
12: end while
13: Retornar la mejor solución encontrada.
14: end
```

En [Katoch et al., 2021] se presenta un árbol detallado de las variantes existentes para los operadores presentes en este algoritmo y para las representaciones que se pueden utilizar para la solución, este se encuentra en la figura 6.

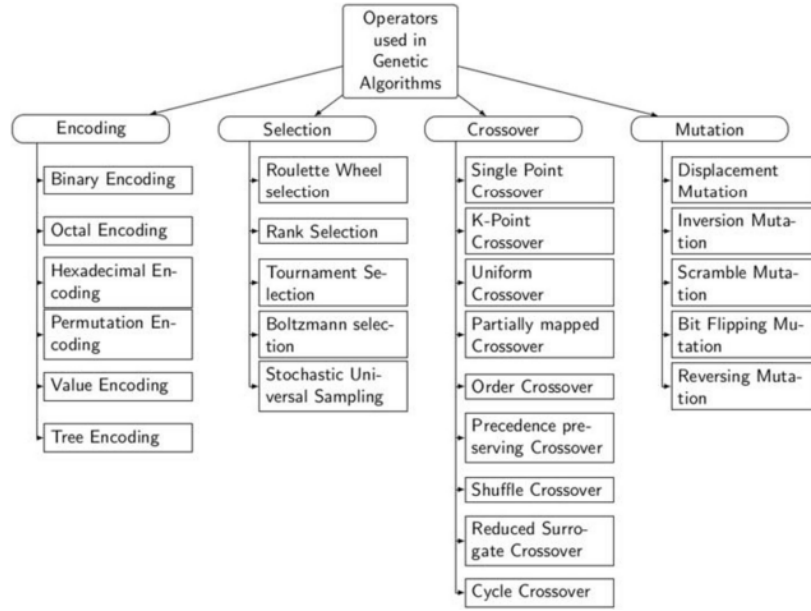


Figura 6: Esquema de operadores y representaciones para GA. [Katoch et al., 2021]

Algoritmos de este tipo han sido implementados para realizar tareas como procesamiento de imágenes [Gong y Yang, 2001] y resolver problemas del tipo *Job Shop* [Madureira et al., 2002].

2.3.2. Particle Swarm Optimization (PSO)

Propuesto originalmente por [Kennedy y Eberhart, 1995], PSO es un algoritmo del subconjunto de algoritmos de inteligencia de enjambre, específicamente basado en el modelo social de una bandada de aves (partículas). Cada partícula mantiene una posición y velocidad, mientras se mueve por el espacio de búsqueda. Esto en términos de las implementaciones más recientes se ve reflejado en algoritmos donde se genera una población inicial de soluciones en un espacio de coordenadas, asociados a una velocidad inicial, la cual se actualizará iterativamente utilizando la velocidad y posición actual [Clerc y Kennedy, 2002].

Investigaciones posteriores agregan al cálculo de la nueva posición, la información sobre las soluciones obtenidas por cada partícula individualmente, tanto como la información obtenida por el conjunto completo de estas.

La ecuación 3 muestra una implementación matemática de este cálculo [Shi y Eberhart, 1998]

$$v_{k+1}^i = v_k^i + c_1 r_1 (p_i - x_k^i) + c_2 r_2 (p_k^g - x_k^i) \quad (3)$$

donde v_k^i representa al vector velocidad de la partícula i en la iteración k , c_1 y c_2 constantes

y x_k^i la posición de la partícula i en la iteración k .

La figura 7 muestra una representación gráfica de como las partículas se comportan en el espacio de búsqueda luego de su inicialización y cálculo de su primer movimiento.

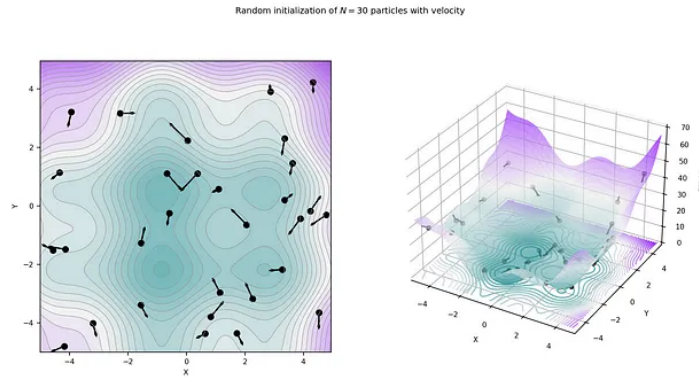


Figura 7: Movimiento de partículas en PSO. [Thevenot, 2022]

En este caso, la comunicación entre partículas se representa en $p_i - x_k^i$ y p_g , donde p_i es la mejor posición encontrada por la partícula i y así, g representa a la partícula que maneja la mejor posición (solución candidata) entre toda la población. Se tiene a r_1 y r_2 como números aleatorios entre $[0, 1]$ que permiten al algoritmo mantener una exploración y explotación adecuada del espacio de búsqueda.

2.3.3. Ant Colony Optimization (ACO)

Formulado a comienzos de la década de 1990 [Dorigo, 1991][Dorigo, 1992], ACO es una metaheurística que se inspira en colonias reales de hormigas, principalmente en el comportamiento que estas tienen para la obtención de alimentos y el cómo direccionan su búsqueda hacia dichos alimentos.

Las hormigas presentes en una colonia buscan su alimento basándose en la concentración de feromonas liberadas por otras hormigas, es decir, buscan el camino con mayor concentración de feromonas, pues siguiendo este es que deberían encontrar el alimento. Una vez encontrada la fuente de alimento, cada hormiga evalúa la calidad y cantidad de este. El alimento encontrado se lleva de vuelta hacia la colonia, dejando un camino marcado por feromonas en su viaje, guiando al resto de hormigas cercanas por el mismo camino, sobre todo una vez que las feromonas de caminos menos utilizados, se evaporan.

Este comportamiento se aprecia gráficamente en la figura 8, donde en primera instancia se muestra un recorrido inicial de exploración realizado por una hormiga, el cual deja un camino de feromonas marcado. Luego, distintas hormigas emprenden búsqueda hacia el alimento, eligiendo seguir o no el camino y a su vez dejando su propio rastro y reforzando los caminos que conducen más rápidamente al alimento, optimizando la búsqueda. Una vez realizados

los recorridos individuales, el camino más recorrido, es decir, el camino con más feromonas a causa de numerosos viajes exitosos y a pesar de la evaporación de las feromonas en el tiempo, es el escogido para la extracción de recursos.

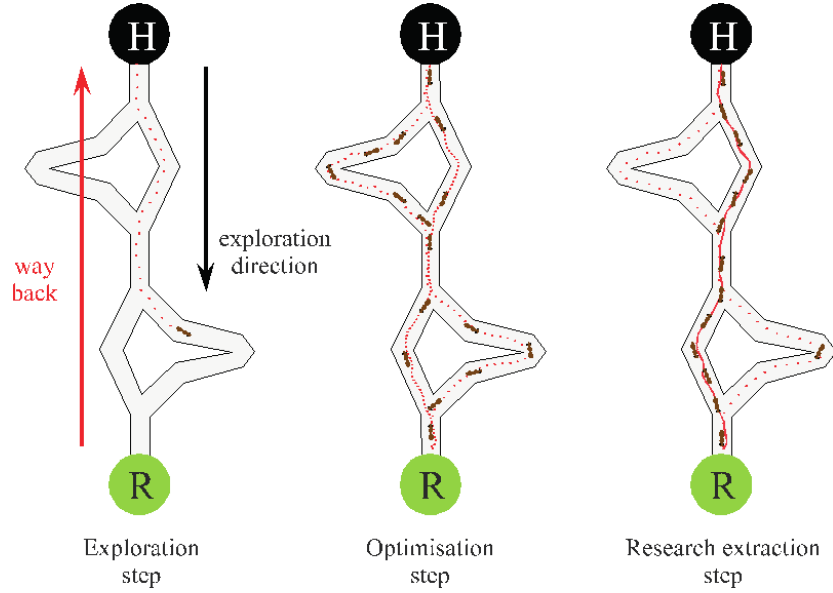


Figura 8: Comportamiento de colonias de hormigas. [Katona *et al.*, 2019]

En forma de algoritmo, este comportamiento se formula de forma que cada hormiga pasa a ser un agente de búsqueda, el cual iterativamente construye una solución. Mientras que las fuentes de alimento pasan a ser los distintos componentes de la solución, como por ejemplo, los nodos de un grafo en un problema del tipo *Traveling salesman problem* (TSP). El pseudocódigo asociado a esta metaheurística se muestra en el algoritmo 2.

El ciclo de este algoritmo comienza con una población de agentes de búsqueda en una posición inicial (solución candidata vacía) para que luego cada uno de estos agentes agregue probabilísticamente un componente a su solución. Luego, en cada iteración, se actualizan los rastros de feromonas, lo cual se hace a partir de uno o más componentes de una o más soluciones encontradas por los agentes de búsqueda, priorizando aquellas con mejor aptitud. Aquí, se aumenta el factor de feromonas hacia dichos componentes, y también se ejecuta el proceso de evaporación de las feromonas (el cual reduce el nivel de éstas para así fomentar la exploración de distintos caminos y no permitir que el algoritmo tenga una convergencia prematura hacia una sección subóptima). De esta forma, la selección probabilística de componentes de solución se ve afectada por la cantidad de feromonas hacia cada componente. La formulación matemática de este fenómeno se puede encontrar en la ecuación 4, la cual entrega la probabilidad de escoger un componente para una solución particular [Blum, 2005].

$$p(c_i | s) = \frac{[\tau_i] \cdot [\eta(c_i)]}{\sum_{c_j \in \mathcal{N}(s)} [\tau_j] \cdot [\eta(c_j)]}, \quad \forall c_i \in \mathcal{N}(s) \quad (4)$$

Algorithm 2 Optimización por Colonia de Hormigas (ACO)

```
1: begin
2: Inicializar los niveles de feromonas en todos los caminos
3: Inicializar la mejor solución global  $S_{best}$  y su longitud  $L_{best}$ 
4: Inicializar el contador de iteraciones  $t = 0$ 
5: Definir el número máximo de iteraciones  $MAX$ 
6: while  $t < MAX$  do
7:   Inicializar todas las hormigas en los nodos de partida
8:   for cada hormiga  $k$  do
9:     while la hormiga  $k$  no haya completado su recorrido do
10:      Elegir el siguiente nodo basado en la probabilidad influenciada por feromonas e información heurística
11:      Mover la hormiga  $k$  al nodo elegido
12:      Actualizar el camino recorrido por la hormiga  $k$ 
13:    end while
14:    Calcular la longitud  $L_k$  del recorrido de la hormiga  $k$ 
15:    if  $L_k < L_{best}$  then
16:      Actualizar  $S_{best}$  y  $L_{best}$ 
17:    end if
18:  end for
19:  Evaporar y actualizar feromonas en todos los caminos
20:  for cada hormiga  $k$  do
21:    Depositar feromonas en los caminos utilizados por la hormiga  $k$ 
22:  end for
23:   $t = t + 1$ 
24: end while
25: Retornar la mejor solución encontrada  $S_{best}$ 
26: end
```

Con τ_i el factor de feromonas hacia el componente en la posición i , $\eta(c_i)$ el factor con el que el componente c_i influencia la solución y $\mathcal{N}(s)$ el conjunto de componentes candidatos para la solución s .

2.3.4. Artificial Bee Colony (ABC)

Siendo uno de los tres algoritmos más conocidos basados en el comportamiento de abejas [Yang, 2005][Pham *et al.*, 2006][Karaboga *et al.*, 2005], este se basa en tres conceptos principales, fuentes de alimento, recolectores empleados y no empleados.

Desde el punto de vista biológico, una abeja considera distintos aspectos respecto de una fuente alimenticia, tales como la calidad del alimento, la cercanía a la colonia o la facilidad de extracción.

Esto va directamente relacionado con las abejas recolectoras empleadas, las cuales se encuentran realizando la tarea de extracción de una fuente ya encontrada. Aquellas no empleadas se dividen en abejas encargadas de buscar fuentes de alimento nuevas, de forma aleatoria, o aquellas que lo hacen con la información entregada por abejas *scout*, abejas espectadoras y el resto de la población. Las abejas *scout* y abejas espectadoras representan alrededor de un 5 % y 10 % de la población [Karaboga y Akay, 2009].

Esta analogía es llevada a un algoritmo metaheurístico, para así pasar a funcionar de forma que la posición de una fuente de alimento representa una solución candidata y los parámetros que describían a esta fuente se reducen a la aptitud de la solución.

De esta forma, inicialmente se asignan abejas recolectoras a soluciones iniciales, de forma que realicen un proceso de explotación de la solución entregada a través de movimientos definidos para modificarla. Luego de que cada uno de estos procesos finalice, la información de las soluciones obtenidas se comparte con las abejas espectadoras, las cuales decidirán sobre a qué región se dirigirán basándose en una probabilidad basada en la aptitud de la fuente de alimento estudiada en dicha región.

Finalmente, las abejas *scout* se envían a una sección aleatoria a buscar fuentes de comida y explotarlas. De esta forma, iterativamente, se explora el espacio de búsqueda y se explotan soluciones hasta conseguir una solución adecuada.

2.4. Métricas de desempeño para metaheurísticas

El desempeño de una metaheurística se puede estudiar desde distintos enfoques, tradicionalmente, se ha asociado a la aptitud de la solución con mejor aptitud obtenida durante el proceso de búsqueda, es decir, que tanto se ajusta a su función objetivo [Fonseca y Fleming, 1996]. Sin embargo, existe un área de estudios dedicada a desarrollar y aplicar distintas formas de medir el desempeño de un algoritmo desde otros puntos de vista y así, tener un mayor entendimiento de como se comportan distintas metaheurísticas

Algorithm 3 Algoritmo de Colonia Artificial de Abejas (ABC)

```
1: begin
2: Inicializar población con  $N$  soluciones  $x_i$  ( $i = 1, 2, \dots, SN$ )
3: Evaluar aptitud para cada solución inicial
4: while no se cumpla criterio de parada do
5:   // Fase de Abejas Empleadas
6:   for cada abeja empleada do
7:     Generar nueva solución  $v_i$  en la vecindad de  $x_i$ 
8:     Evaluar aptitud de  $v_i$ 
9:     if  $aptitud(v_i)$  mejor que  $aptitud(x_i)$  then
10:      Reemplazar  $x_i$  con  $v_i$ 
11:    end if
12:  end for
13:  // Fase de Abejas Observadoras
14:  for cada abeja observadora do
15:    Seleccionar solución  $x_i$  basada en probabilidad de aptitud
16:    Generar nueva solución  $v_i$  en la vecindad de  $x_i$ 
17:    Evaluar aptitud de  $v_i$ 
18:    if  $aptitud(v_i)$  mejor que  $aptitud(x_i)$  then
19:      Reemplazar  $x_i$  con  $v_i$ 
20:    end if
21:  end for
22:  // Fase de Abejas Exploradoras
23:  if alguna solución no mejorada por  $limite$  intentos then
24:    Reemplazar con nueva solución aleatoria
25:  end if
26:  Memorizar la mejor solución encontrada hasta ahora
27: end while
28: Retornar la mejor solución encontrada
29: end
```

y las distintas implementaciones de estas.

En el estudio desarrollado por [Ivković *et al.*, 2016], se define el desempeño de un algoritmo estocástico de optimización en dos conceptos, la aptitud de la solución candidata retornada por el algoritmo, y los recursos computacionales que implica llegar a dicha solución. Los recursos computacionales se relacionan al tiempo de ejecución, cuantas iteraciones, evaluaciones o soluciones generadas se deben realizar antes de llegar a una solución final.

En el trabajo de [Hoos, 1999] se realiza un análisis exhaustivo sobre el comportamiento de distintos algoritmos de búsqueda local, teniendo en cuenta aspectos como el comportamiento en tiempo de ejecución, la fase de búsqueda inicial, entre otros.

Particularmente, se enfocan los análisis sobre la utilización de tiempo de ejecución o conteos de operaciones, la relación de hiperparámetros con la aptitud de las soluciones. Aplicando el análisis a distintos algoritmos, el autor concluye sobre distintos aspectos a partir de los cuales se puede mejorar el desempeño del algoritmo, proporcionando mecanismos, como el reinicio aleatorio del algoritmo, y métodos específicos, como el paralelismo, para aplicar dichas mejoras.

En los últimos años, estudios enfocados en el análisis del estado del desarrollo de métricas de desempeño [Ivković *et al.*, 2016, Memari *et al.*, 2017], categorizan dichas métricas en dos categorías principales, las métricas de eficiencia y de efectividad. Las métricas de eficiencia son enfocadas en la rapidez para la búsqueda de soluciones, mientras que las de efectividad estudian la capacidad de encontrar soluciones de buena aptitud [Halim *et al.*, 2021]. A continuación se presentan métricas para cada una de estas categorías, con sus definiciones y objetivos.

2.5. Métricas de eficiencia

Las métricas de eficiencia contienen elementos relacionados con la capacidad de explorar el espacio de búsqueda y explotar la aptitud del conjunto de soluciones candidatas. Además, consideran criterios como el costo computacional, la complejidad algorítmica y el análisis estadístico. En general, es el área que concentra las métricas enfocadas en la ejecución del algoritmo y los elementos que lo componen, sin realizar un análisis extenso sobre las soluciones entregadas por este.

2.5.1. Convergencia

La convergencia de una metaheurística informa sobre la capacidad de ésta para guiar al algoritmo hacia una solución de mejor aptitud, intentando alcanzar siempre una solución óptima o cercana a esta. El cálculo de la convergencia se realiza a partir de las iteraciones necesarias para alcanzar dicha solución y la tasa de cambio observada en la aptitud de la solución. La

tasa de convergencia es definida por [Nebro *et al.*, 2008] como la cantidad de evaluaciones necesarias para alcanzar el nivel de aptitud esperado para la solución.

En la práctica, esta métrica es implementada de la forma:

$$\frac{|f_{opt} - f_i|}{|f_{opt} - f_{i-1}|} \quad (5)$$

Donde f_{opt} representa al valor de aptitud de la solución óptima dada una instancia particular.

Esta métrica tiene una gran similitud al error relativo [Li y Zhao, 2005], el cual, en comparación a la convergencia propuesta, elimina el uso de la aptitud encontrada en la evaluación previa, resultando en una ecuación de la forma:

$$\frac{|f_{opt} - f_i|}{|f_{opt}|} \quad (6)$$

[Dhivyaprabha *et al.*, 2018] utiliza esta formulación de la convergencia para calcular una tasa promedio que informa sobre la naturaleza sobre la tasa de convergencia. Esta tasa toma valor 1 para representar una tasa de convergencia lineal, y 2 para una convergencia cuadrática. En la figura 9 se presenta este parámetro graficado para distintos algoritmos, donde los ejes x e y representan los algoritmos y sus valores de convergencia promedio, respectivamente. Se puede observar que el algoritmo *cosine mixture* obtiene una alta convergencia para las primeras 1000 iteraciones y también para su promedio sobre las primeras 10000 iteraciones, con un valor máximo de 1,12.

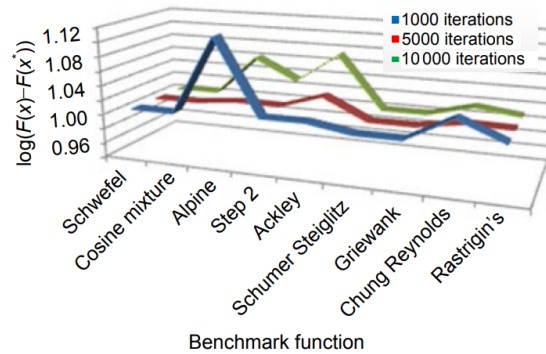


Figura 9: Tasa promedio de convergencia para *synergistic fibroblast optimization*. [Dhivyaprabha *et al.*, 2018]

Ahora, en el caso de que no se conozca un óptimo para la instancia del problema a resolver, la convergencia se puede definir como se expresa a continuación:

$$\frac{|f_{i+1} - f_i|}{|f_i - f_{i-1}|} \quad (7)$$

donde se utilizan los valores evaluados de la solución obtenida en la iteración i , $i+1$ e $i-1$. De la misma forma que en el caso donde se utiliza el óptimo, se puede obtener el error relativo y eliminar la necesidad de utilizar la evaluación previa.

En la figura 10 se muestra una gráfica de comparación de la tasa de convergencia postulada en la ecuación 7 para tres distintos algoritmos. El eje x de este gráfico representa el valor de la tasa de convergencia obtenida, mientras que el eje y, la cantidad de evaluaciones consumidas al momento de medir la convergencia, mientras que cada una de las líneas representa un algoritmo. Podemos ver que la convergencia de estos algoritmos rápidamente decae debido a que la formulación matemática requiere del valor de aptitud obtenido en la iteración anterior, lo que lleva a que una vez que los valores se repitan o sean muy cercanos, la tasa sea cercana o igual a 0, demostrando un estancamiento en la búsqueda de mejores soluciones.

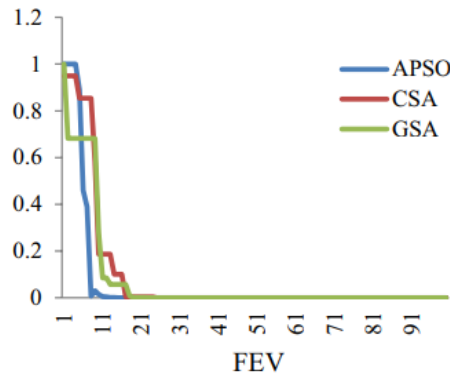


Figura 10: Estudio de tasa de convergencia basada en error consecutivo. [Halim *et al.*, 2021]

A partir del concepto de tasa de convergencia y su implementación, se desarrollan métricas de desempeño más refinadas y complejas.

2.5.1.1. Geometric average convergence rate

Esta métrica extiende el concepto de convergencia presentado en la ecuación 5, calculando la convergencia para cada iteración y posteriormente obtiene el promedio geométrico para estas.

Su formulación matemática se encuentra en la ecuación 8 [He y Lin, 2016] y se desprende directamente de la definición de convergencia encontrada en la ecuación 5:

$$R(t) := 1 - \left(\left| \frac{f_{\text{opt}} - f_1}{f_{\text{opt}} - f_0} \right| \cdot \dots \cdot \left| \frac{f_{\text{opt}} - f_t}{f_{\text{opt}} - f_{t-1}} \right| \right)^{1/t} \equiv 1 - \left(\left| \frac{f_{\text{opt}} - f_l}{f_{\text{opt}} - f_0} \right| \right)^{1/l} \quad (8)$$

con $R(t)$ la tasa de convergencia promedio (Geométrico) f_{opt} el valor de aptitud óptima para el problema, y f_t la función de evaluación en la iteración t .

Este cálculo presenta una versión más estable para el cálculo de la convergencia, ya que no se ve afectado por la situación recurrente donde $f_{opt} - f_i \approx f_{opt} - f_{i-1}$ debido a que toma información sobre toda la ejecución y no solo en un instante determinado. Para ejemplificar esto, la tabla 1 contiene datos para una ejecución de un algoritmo genético sobre el problema OneMax.

Evaluación	Mejor aptitud encontrada
1	11.00000
2	11.00000
3	13.00000
4	13.00000
5	13.00000
6	13.00000
7	13.00000
8	13.00000
9	14.00000
10	14.00000

Tabla 1: Evaluaciones y valores evaluados para ejecución de GA genérico. Elaboración propia.

Utilizando los datos presentados para calcular la convergencia tradicional, su promedio geométrico normalizado y su promedio aritmético como punto de referencia, asumiendo como óptimo $f_{opt} = 20$, se obtiene la tabla 2.

Convergencia	Convergencia promedio geométrico
0.00000	0.00000
0.22000	0.08036
0.00000	0.06090
0.00000	0.04902
0.00000	0.04102
0.00000	0.03527
0.00000	0.03093
0.14000	0.04405
0.00000	0.03974

Tabla 2: Valores convergencia y convergencia promedio geométrico

Se puede observar que la tasa de convergencia convencional no entrega información utilizable, extrapolando esto a un caso con gran cantidad de evaluaciones, solo ocasionalmente se modifica su valor, haciendo difícil la visualización del cambio. En cambio, al aplicar el promedio, se tiene información sobre la ejecución completa en todo momento, proporcionando información sobre el comportamiento de la tasa de convergencia en sí y no solo en que momentos mejora la solución.

[He y Lin, 2016] utiliza dos algoritmos, *Multigrid EA* y *Fast Evolutionary Programming*, para

probar esta métrica, dicha evaluación se puede observar en la figura 11, donde ambos algoritmos son graficados como líneas sólidas y punteadas, respectivamente. En la figura, se representa el valor de la métrica en el eje y , mientras que se utiliza la cantidad de iteraciones como métrica de tiempo en el eje x . Es evidente, en los valores obtenidos, que esta métrica reduce fuertemente la volatilidad del resto de métricas de convergencia.

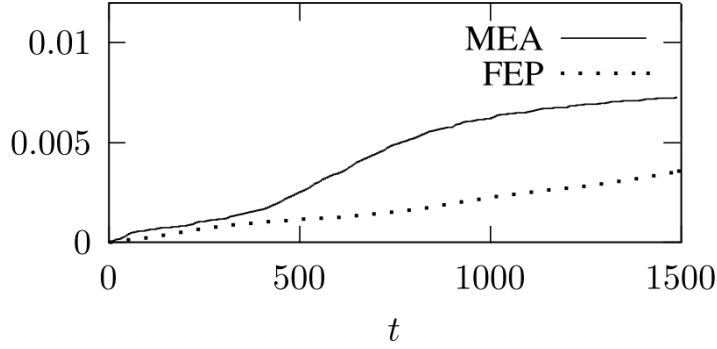


Figura 11: Estudio de tasa promedio geométrica de convergencia. [He y Lin, 2016]

2.5.1.2. *E-value*

En [Mora-Meliá *et al.*, 2015] se propone una métrica a la cual solo le llama *Efficiency* (E). En este trabajo, para poder diferenciar a este concepto, se le representará como *E-value*.

La definición de esta métrica utiliza dos proporciones para su cálculo y tiene por objetivo representar el esfuerzo asociado a encontrar la solución final. Su formulación matemática se encuentra representada en la ecuación 9

$$E = \frac{\eta_{quality}}{\eta_{convergence}} \quad (9)$$

donde $\eta_{quality}$ corresponde a una representación particular de calidad de las soluciones encontradas y se define de la siguiente forma:

$$\eta_{quality} = \frac{N_{successful}}{N_{sim}} \quad (10)$$

donde $N_{successful}$ se define como la cantidad de soluciones finales “suficientemente buenas” (controlando un parámetro de umbral de aptitud arbitrariamente definido) y N_{sim} es el total de soluciones encontradas.

Luego, el término $\eta_{convergence}$ representa, en la propuesta original, el tiempo que toma el algoritmo para converger a la solución resultante, finalizando la ejecución del algoritmo. Sin

embargo, [El-Ghandour y Elbeltagi, 2018] implementa esta métrica de forma que se reemplaza el tiempo para converger, el cual depende de una cantidad muy grande de variables difíciles de controlar, por una proporción entre la cantidad de evaluaciones realizadas para llegar a la solución final y el espacio de búsqueda total. Así, se tiene un análisis generalizado de la velocidad de convergencia hacia soluciones aceptables.

Los autores de esta métrica de desempeño la evalúan para el problema de redes de distribución de agua utilizando cuatro algoritmos distintos, *Harmony Search* (HS), *Pseudogenetic Algorithm* (PGA), PSO y *Shuffled Frog Leaping* (SFLA), todos algoritmos poblacionales evolutivos. A partir de la experimentación en distintos *datasets*, se obtiene una visualización donde la tendencia a mayor eficiencia se hace evidente, dado que, un algoritmo que se dirige de forma eficiente a una solución, mantendrá un valor relativamente constante hasta el final de su ejecución. La visualización de estos resultados se encuentra en la figura 12.

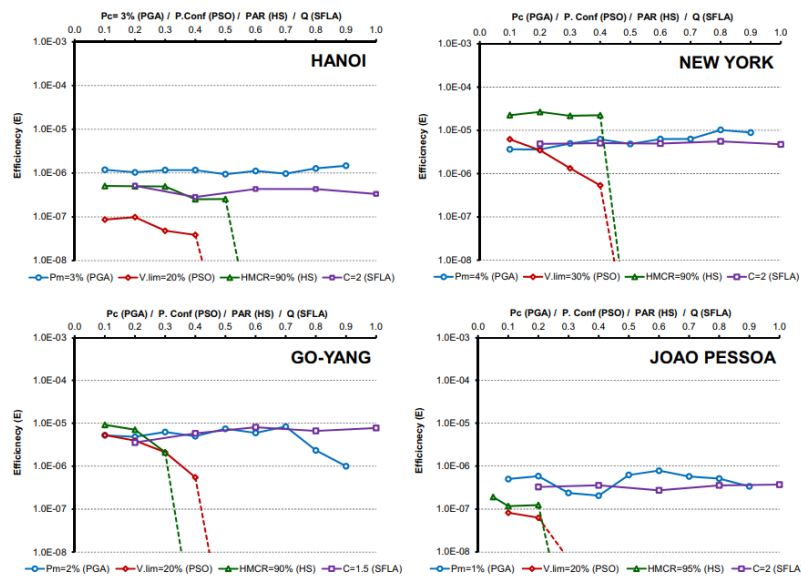


Figura 12: Estudio de eficiencia en redes de distribución de agua. [Mora-Meliá *et al.*, 2015]

En la figura se pueden observar 4 gráficos distintos, las cuales representan cada uno de los *datasets* sobre los cuales se ejecutan los algoritmos, mientras que cada línea representa un algoritmo. En el eje x se representa la proporción respecto del óptimo que tiene que tener la aptitud de la solución para ser considerado en el cálculo de la métrica, mientras que en el eje y, el valor evaluado de la métrica. Tomando el caso de la red Hanói, el valor de PGA es aproximadamente 2.9 veces mayor que el de HS y SFLA, aunque HS decae rápidamente en eficiencia alrededor de la mitad de su ejecución.

2.5.2. Diversidad

Las métricas de diversidad son aquellas que miden la capacidad de explorar el espacio de búsqueda, observable en la distribución de la población de soluciones de un algoritmo en el espacio de búsqueda para cada iteración. También, permiten realizar análisis sobre las diferencias estructurales de las soluciones. Esta subcategoría de métricas de desempeño es sumamente relevante dado su enfoque en métricas para metaheurísticas poblacionales. Esto se analiza por cada ejecución del algoritmo y reporta sobre el comportamiento de la población misma.

2.5.2.1. Segmentación del espacio de búsqueda

Para el análisis de diversidad es importante considerar el manejo del espacio de búsqueda en sí, donde muchos autores requieren discretizar el espacio para comprender como se comportan los agentes de búsqueda [Yu *et al.*, 2005][Chacón Sartori *et al.*, 2023].

Por lo general, los problemas resueltos con metaheurísticas manejan soluciones de gran dimensionalidad, es decir, contienen una cantidad considerable de variables. Para hacer un análisis correcto del espacio de búsqueda, sin generarlo explícitamente, se consideran métodos que permitan diferenciar soluciones entre sí (más allá de la aptitud que generan o su relación con el problema).

Para conseguir este objetivo existe *Multidimensional Scaling* (MDS). Este método se basa en la información obtenida de las distancias entre pares de elementos de un conjunto para generar una configuración de puntos en espacio euclidiano [Mead, 1992]. Es decir, a través de la información obtenida de las diferencias entre variables de cada solución, se puede calcular una distancia entre el conjunto completo de elementos para poder representarse en un espacio de menor dimensionalidad que el original.

Para utilizar los métodos de *Multidimensional Scaling*, se necesita medir la disimilitud entre soluciones. En el contexto de los problemas de optimización, se debe tener en cuenta, además, la existencia de problemas continuos y discretos.

En el caso discreto existen diversas formas de obtener la disimilitud. [Cicirello y Cernera, 2013] utiliza dos métodos distintos, *Reinsertion Distance* e *Interchange Distance*, ambos basados en el concepto de *Edit Distance* [Wagner y Fischer, 1974], donde el primero involucra buscar la cantidad de veces, entre un par de soluciones, que se debe remover elementos y reinsertarlos en otra posición de la solución para poder obtener dos soluciones idénticas. El segundo método involucra directamente un intercambio de elementos en una solución para lograr dos soluciones iguales. Casos con una mayor complejidad se pueden ver en *Deviation Distance* [Ronald, 1998]. Aquí, se utiliza la suma normalizada entre las distancias de cada variable. La formulación matemática de esto se expone en la ecuación 11.

$$d_{i,j}^d = \sum_{v=1}^l \Delta'_v = \sum_{v=1}^l \frac{\Delta_v}{l-1} \quad (11)$$

donde Δ_v es la distancia del elemento v entre las dos soluciones, por ejemplo, si se tienen las soluciones:

$$\begin{aligned} x &= [a, f, c, d, b, e] \\ y &= [b, e, f, c, a, d] \end{aligned}$$

La distancia Δ_a es 4, ya que hay que desplazar la variable que contiene el valor a en 4 posiciones para que tenga la misma posición en ambas soluciones.

Luego, se realiza la suma normalizada de estas distancias basándose en lo definido en la ecuación 11, obteniendo un valor para $d_{i,j}^d = \frac{4+4+1+2+4+1}{5} = 3,2$.

Para situaciones donde se debe abordar ambos casos (Discreto y continuo), se puede utilizar un método simple como *Exact Match* [Ronald, 1998], donde se suma la cantidad de veces que una variable no coincide entre las soluciones.

Ejecutando alguno de estos métodos es posible obtener la matriz de distancias D . Esta matriz de $n \times n$, con n la cantidad de vectores a reducir, contiene las distancias d_{ij} calculadas previamente para las soluciones candidatas i y j .

Utilizando los vectores x e y previamente definidos, en conjunto a $z=[d, c, a, b, f, e]$, es posible utilizar *Deviation Distance* para calcular D y obtener:

	X	Y	Z
X	0	3.2	2
Y	3.2	0	3.6
Z	2	3.6	0

Esta matriz permite el uso de MDS. El algoritmo de MDS clásico, propuesto por [Kruskal y Wish, 1978] y expuesto en 4, consta de 4 pasos posteriores al cálculo de D . Primero, se calcula D^2 y luego se forma una matriz J para 'centrar' D , la definición de J es $J = I - \frac{1}{n}11^T$. En J el concepto de centrar significa que es una matriz cuya media es 0 tanto para filas como columnas, de forma que al ser aplicada a cualquier matriz o vector, por ejemplo $J \times a$, con a un vector cualquiera, se obtiene la versión centrada de a . Entonces, con J , ejecutamos el proceso de centrado doble para obtener una versión centrada de las columnas y filas de D^2 . El proceso de centrado se realiza de la siguiente forma:

$$B = -\frac{1}{2}JD^2J \quad (12)$$

Con la matriz B resultante, una matriz simétrica donde cada elemento representa el producto interno entre las coordenadas de los puntos originales i y j en el nuevo espacio euclidiano. Luego, se hace una descomposición de valores propios de B , matemáticamente, $B = V\Lambda V^T$ con V la matriz de vectores propios y Λ matriz diagonal de valores propios. Finalmente, se calcula la matriz X_k de coordenadas en el espacio reducido de dimensión k . La expresión utilizada para esto está definida como:

$$X_k = V_k \Lambda_k^{\frac{1}{2}} \quad (13)$$

con Λ_k la matriz diagonal de los k mayores valores propios, y V_k los vectores propios correspondientes.

Algorithm 4 Classical Multidimensional Scaling (MDS)

- 1: **Input:** Matriz de distancias $D = [d_{ij}]$ de tamaño $n \times n$, dimensión objetivo k
- 2: **Output:** Coordenadas en menor dimensión X_k de tamaño $n \times k$
- 3: **Paso 1: Cálculo de matriz de distancias al cuadrado**
- 4: Calcular la matriz de distancias al cuadrado D^2 donde $D_{ij}^2 = d_{ij}^2$ para todos i, j
- 5: **Paso 2: Construir matriz para centrar**
- 6: Definir la matriz de centrado $J = I - \frac{1}{n}\mathbf{1}\mathbf{1}^T$, donde I es la matriz identidad y $\mathbf{1}$ es un vector de unos
- 7: **Paso 3: Centrar la matriz de distancias al cuadrado**
- 8: Calcular la matriz B de la siguiente manera:

$$B = -\frac{1}{2}JD^2J$$

- 9: **Paso 4: Descomponer matriz B por valores y vectores propios**
- 10: Calcular la descomposición en valores propios de B :

$$B = V\Lambda V^T$$

donde V es la matriz de vectores propios y Λ es la matriz diagonal de valores propios

- 11: **Paso 5: Construir matriz de coordenadas en dimensión reducida**
- 12: Seleccionar los k valores propios más grandes Λ_k y los vectores propios correspondientes V_k
- 13: Calcular las coordenadas en menor dimensión X_k como:

$$X_k = V_k \Lambda_k^{\frac{1}{2}}$$

- 14: **Return** X_k
-

Gracias a la representación que entregan estos métodos, es que podemos diseñar formas de dividir el espacio de búsqueda de soluciones de forma arbitraria y acorde al problema a analizar.

2.5.2.2. Entropy Diversity

Propuesta por [Yu *et al.*, 2005], esta métrica se basa en dividir el espacio de búsqueda en una cantidad arbitraria de áreas de igual tamaño, las cuales se forman basándose en los dominios definidos para las variables. El objetivo de esta métrica es analizar y evaluar sobre la cantidad de agentes de búsqueda que han visitado ciertas áreas del espacio de búsqueda, para medir la distribución de las soluciones candidatas visitadas. Matemáticamente, se define en la ecuación 14.

$$E(t) = - \sum_{j=1}^Q \frac{Z_i}{N} \log_e \left(\frac{Z_i}{N} \right) \quad (14)$$

donde Q es la cantidad de áreas, Z_i la cantidad de agentes en el área i y N la cantidad de agentes de búsqueda en la población.

Este índice se ve aplicado en algoritmos del tipo PSO [Pires *et al.*, 2013], obteniendo resultados que informan sobre como el algoritmo hace más uniforme a la población de soluciones candidatas con el tiempo, tal como se puede apreciar en la figura 13, donde se grafica un algoritmo ejecutado con distintos tamaños de población (representados por las distintas líneas coloreadas) y como su diversidad, cuyo valor se grafica en el eje y , varía a través de las distintas generaciones, las cuales definen el eje x del gráfico.

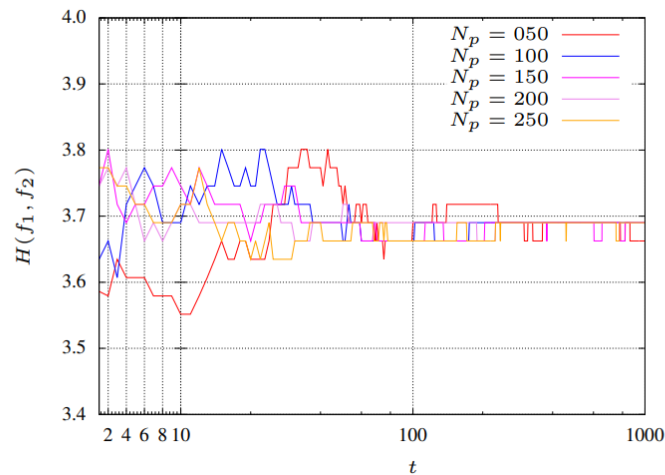


Figura 13: Estudio de *Entropy Diversity* en PSO. [Pires *et al.*, 2013]

Esta métrica también se puede apreciar en versiones modificadas específicamente para diseñar algoritmos que resuelvan problemas tipo TSP, tal como se demuestra en [Nikfarjam *et al.*, 2021], donde se utiliza la repetición de los distintos segmentos posibles en las soluciones candidatas a diferencia de una sección de cierto tamaño en el espacio de búsqueda. En este caso, la figura 14 representa distintos algoritmos ejecutando un mismo

problema con distintos parámetros, los colores representando algoritmos distintos y los tipos de línea los distintos parámetros para cada uno. Mientras que los ejes x e y representan la entropía y la cantidad de evaluaciones dividida por 100,000, respectivamente. En esta figura se observa que el algoritmo propuesto por los autores [Nikfarjam *et al.*, 2021] logra mayor convergencia que algoritmos basados en *Entropy Diversity* sin modificar. Esto es apreciable en los algoritmos catalogados como ENT, los cuales mantienen una mayor diversidad consistentemente frente al resto de algoritmos expuestos.

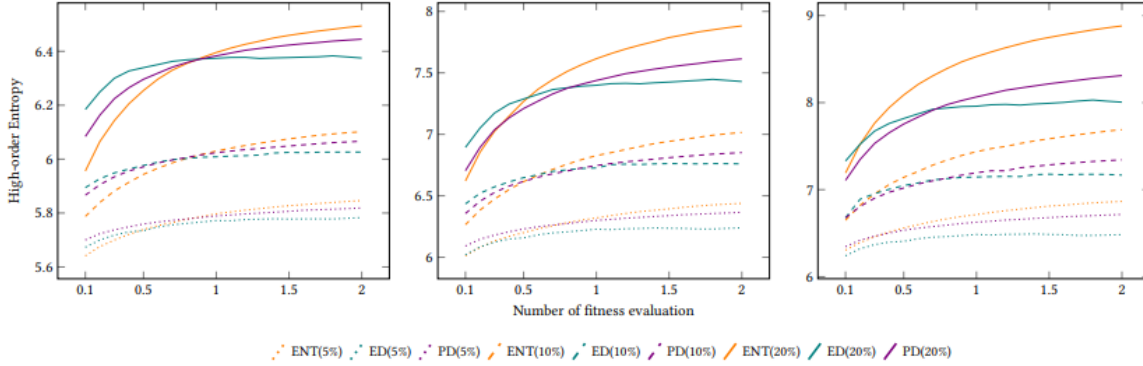


Figura 14: Estudio de diversidad en GAs basados en *Entropy Diversity*, su versión modificada y distancia entre pares de soluciones para problema TSP. [Nikfarjam *et al.*, 2021]

2.5.2.3. Métricas de desempeño basadas en PSO

A partir de las investigaciones para el algoritmo PSO, [Krink *et al.*, 2002] plantea una forma de medir la diversidad en métodos de este tipo, donde se trabaja directamente con una topología para localizar y guiar a los agentes de búsqueda de soluciones.

De esta forma, propone una medición de diversidad, la cual se basa en la distancia de las partículas hacia el centro del enjambre, o bien, la distancia de cada partícula hacia la posición promedio.

La ecuación 15 expone su formulación matemática.

$$diversity(S) = \frac{1}{|S|} \cdot \sum_{i=1}^{|S|} \sqrt{\sum_{j=1}^N (p_{ij} - \bar{p}_j)^2} \quad (15)$$

Aquí, S es la población, N la dimensionalidad del problema, p_{ij} la aptitud de la partícula i en la iteración j , mientras que $\bar{p}_j$ corresponde al valor de aptitud promedio en la iteración j .

Este concepto puede ser expandido para tener una métrica de desempeño que evalúe el comportamiento de cada partícula respecto del resto, tomándose a sí misma como centro, indicando su distancia promedio hacia el resto del enjambre. Esto provee información del

comportamiento del algoritmo incluso en situaciones borde donde se pueden generar soluciones con valores de aptitud que, a partir de un valor de aptitud lejano a la media, puedan sesgar el resto de métricas.

Para esto se define x_{ij} como la dimensión j del agente de búsqueda i . Así, la formulación matemática de esta métrica se define como:

$$D_{all} = \frac{1}{|N|} \sum_{i=1}^N \left(\frac{1}{|N|} \sum_{j=1}^{|N|} \sqrt{\sum_{k=1}^{|Dim|} (x_{ik} - x_{jk})^2} \right) \quad (16)$$

2.5.3. Costo computacional

El costo computacional de un algoritmo se refiere tanto al tiempo utilizado en una ejecución de un algoritmo como a los recursos computacionales que este requiere (memoria, espacio en disco, etc). Tener este concepto en mente es clave para formular métodos eficientes, ya que es una forma de medir empíricamente los recursos utilizados para ejecutar un algoritmo, independiente de su naturaleza y formulación.

Para generalizar este concepto se pueden utilizar métricas tales como el tiempo de computación de la solución [Osuna-Enciso *et al.*, 2022] o una proporción entre el tiempo que toma encontrar soluciones de aptitud aceptable y el tiempo que toma encontrar un óptimo global [Barr *et al.*, 1995] o la solución final.

2.5.3.1. Normalización de métricas

Es importante tener en cuenta que el ambiente, ya sea en términos de sistema operativo, hardware u otros, afecta la capacidad de analizar correctamente las pruebas ejecutadas. Incluso teniendo en cuenta este factor, no siempre se puede asegurar una igualdad de condiciones. Sin embargo, se puede aliviar este problema al considerar e implementar en las métricas de desempeño una normalización a través de dos valores numéricos que permiten realizar un análisis sobre la ejecución del algoritmo en hardware específico.

A continuación se presentan métricas universales, las cuales permiten realizar este proceso.

2.5.3.2. *Million instructions per second (MIPS)*

Este factor numérico mide la cantidad de instrucciones realizadas por el procesador en un segundo, permitiendo así, comparar el desempeño entre sistemas y así, normalizando el comportamiento de las distintas métricas de desempeño.

Implementado por [Attiya *et al.*, 2020] de la forma:

$$MIPS = \frac{\text{Cantidad de instrucciones} * \text{Frecuencia de procesador}}{\text{Ciclos de procesador} * 10^6} \quad (17)$$

donde la cantidad de instrucciones ejecutadas en cada ciclo se multiplica por la frecuencia del procesador para obtener la cantidad de instrucciones totales, mientras que se divide por los ciclos efectivos del procesador para finalizar la ejecución, esto multiplicado por 10^6 , ya que la métrica se maneja en el orden de millones de instrucciones por segundo.

2.5.3.3. *Million floating point operations per second (MFLOPS)*

Una expresión más específica para medir el desempeño de un procesador son los MFLOPS, los cuales miden la cantidad de operaciones de punto flotante por segundo, dichas operaciones se refieren a toda operación básica del procesador, ya sea adición, sustracción, multiplicación o división.

Su definición matemática descrita en [Lo *et al.*, 2010] lo define como:

$$MFLOPS = \frac{\text{Numero de operaciones de punto flotante}}{\text{Tiempo de ejecucion} * 10^6} \quad (18)$$

donde, en este caso, se realiza el cálculo sobre las operaciones de punto flotante y se usa el tiempo de ejecución como el factor que delimita el análisis. Nuevamente, se utiliza la medida en millones de operaciones de punto flotante, por lo que se usa el factor de 10^6 .

2.6. Métricas de efectividad

Las métricas de efectividad se encargan de informar sobre la idoneidad del algoritmo en encontrar soluciones de alta aptitud dentro del espacio de búsqueda.

Esta categoría se centra en informar sobre conclusiones a partir de aspectos como la aptitud de las soluciones candidatas de la población, si es que estas son factibles, el rango de valores de aptitud o que tan cercanas están las soluciones generadas de algún óptimo global conocido, entre otras. En general, se busca llegar a conclusiones sobre los resultados del algoritmo y como estos son útiles al problema que resuelve más que sobre el comportamiento del algoritmo en tiempo de ejecución.

2.6.1. Perfiles de desempeño

En general, los perfiles de desempeño de una metaheurística son definiciones que permiten observar como se comporta uno o más algoritmos a través de distintas instancias de prueba.

Matemáticamente, gran parte del tiempo, se representan en forma de proporciones sobre un conjunto de pruebas realizadas.

2.6.1.1. Accuracy Profile

Desarrollada por [Hare y Sagastizábal, 2006], esta métrica se basa en las soluciones encontradas y la precisión decimal de sus valores de aptitud. Lógicamente, en la resolución de estos problemas se busca que las soluciones encontradas por el algoritmo tengan un valor de aptitud muy cercano al óptimo. Para medir esta cercanía se utilizan los dígitos de precisión del valor evaluado de la solución candidata y aquellos de la solución óptima.

Para esto, el primer paso es calcular la cantidad de decimales que comparte la solución candidata con el óptimo (Ver ecuación 19). Matemáticamente, se utilizan los parámetros $f(\bar{x}_{p,s})$, el cual representa el valor obtenido para la instancia p utilizando el algoritmo s , mientras que $f(x_{popt})$ representa la solución óptima para el problema p y, finalmente, $f(x_p^0)$ representa al valor evaluado para la solución inicial con la cual trabajan los algoritmos, al resolver el problema p

Este resultado es evaluado por la ecuación por tramos $\gamma_{p,s}$, la cual define un máximo valor de cambio en la precisión, M . Este parámetro se define como la mayor cantidad de decimales iguales al óptimo que una solución puede tener, con el objetivo de evitar considerar las soluciones exactas (que pueden considerarse casos excepcionales no útiles para el análisis).

Finalmente, la ecuación 21 muestra la proporción obtenida, donde P es el conjunto total de pruebas realizadas, τ es la precisión requerida y $\gamma_{p,s}$ es el resultado obtenido para la prueba p usando el algoritmo s

$$f_{acc}^{p,s} = \log_{10} (f(\bar{x}_{p,s}) - f(x_{popt})) - \log_{10} (f(x_p^0) - f(x_{popt})) \quad (19)$$

$$\gamma_{p,s} = \begin{cases} -f_{acc}^{p,s}, & \text{if } -f_{acc}^{p,s} \leq M \\ M, & -f_{acc}^{p,s} > M \text{ o } f_{acc}^{p,s} \text{ es indefinido} \end{cases} \quad (20)$$

$$R_s(\tau) = \frac{1}{|P|} \text{size } \{ \gamma_{p,s} \mid \gamma_{p,s} \geq \tau, p \in P \} \quad (21)$$

En [Yan *et al.*, 2023] se puede apreciar el uso de *Accuracy Profile* entre métodos para evitar las "paradojas" de la comparación de algoritmos de optimización, es decir, evitar las situaciones donde distintas estrategias para comparar los algoritmos entregan respuestas distintas sobre cuál es mejor para el caso estudiado. Para esto, se realizan pruebas sobre 4 algoritmos distintos. Usando un umbral de precisión $M = 16$, se utiliza la métrica para obtener la proporción de problemas que cada algoritmo es capaz de resolver para distintos umbrales de precisión entre 0 y 16. Los resultados obtenidos se muestran en la figura 15. Aquí, en el eje x se expresa la cantidad de dígitos de precisión que se debía obtener en la ejecución representada, mientras que en el eje y , la proporción de ejecuciones que cumple el requisito para cada uno de los algoritmos, representados por cada línea coloreada.

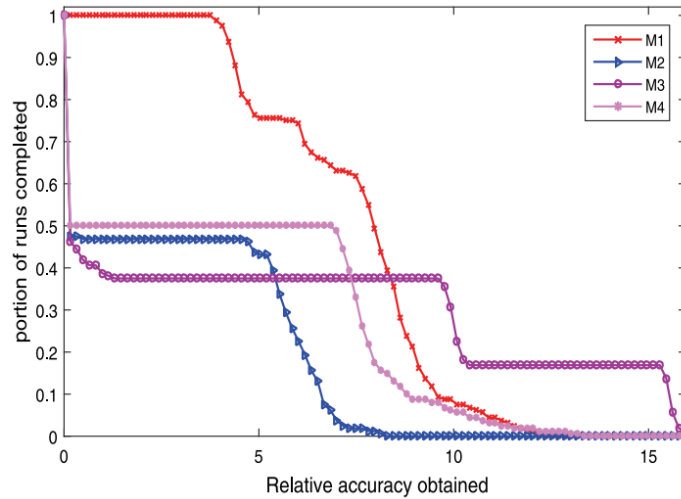


Figura 15: *Accuracy Profile* sobre 4 algoritmos y un mismo set de problemas. [Yan *et al.*, 2023]

Dependiendo de la prioridad que tenga el encontrar soluciones más precisas o resolver la mayor cantidad de problemas, se puede decidir sobre si M1 o M3 es el mejor algoritmo. Es fácil ver que M1 resuelve una gran proporción de problemas hasta una precisión de 8 dígitos, mientras que M3 es más consistente con obtener resultados precisos, sufriendo por no obtener una gran proporción de problemas resueltos con esta precisión.

2.7. Resumen

Durante este capítulo se exponen distintos aspectos del mundo de la optimización combinatorial. Comenzando por los conceptos y contexto necesarios para la comprensión de lo que es un problema de optimización y las metaheurísticas a través de las cuales estos se resuelven.

En términos de los problemas, se entrega la definición formal de uno, donde se formula como una permutación de variables tal que una función de evaluación (como la vista en la

ecuación 1), obtenga un costo o aptitud para el problema. Luego se realiza una investigación sobre metaheurísticas, algoritmos que buscan soluciones de forma guiada, no siempre garantizando encontrar la mejor solución, pero sí garantizando ser más eficiente que un algoritmo que explore todas las soluciones para encontrar dicha solución. Esta investigación lleva a explorar distintos tipos de metaheurísticas, sin embargo, el enfoque que se le da es el de las metaheurísticas poblacionales, las cuales se caracterizan por manejar un conjunto de soluciones las cuales pueden informar la ejecución del algoritmo.

Finalmente, se realiza un estudio sobre métricas para evaluar el desempeño de los distintos algoritmos desde distintas aristas, tales como su eficiencia en encontrar soluciones de alta aptitud, su capacidad de explorar el espacio de búsqueda, el costo computacional que estas incurren, entre otras. Estas métricas informan la creación de una librería dedicada a la evaluación de metaheurísticas a través de distintas métricas, la cual es presentada y desarrollada en el siguiente capítulo.

CAPÍTULO 3

PROPUESTA DE SOLUCIÓN

En este capítulo se presenta la solución diseñada para el problema propuesto, la cual, para su correcta ejecución, utiliza un *framework* para la ejecución de metaheurísticas y la recaudación de datos a partir de estas ejecuciones. Este *framework* cuenta con 8 algoritmos genéticos y evolutivos predefinidos, además de un conjunto de 10 problemas disponibles y 14 variaciones sobre estos.

La recopilación de datos también forma parte del *framework*, aunque para poder realizar un correcto análisis, se definen también nuevos elementos a monitorear. Estos son específicamente asociados al caso de las metaheurísticas poblacionales, entre ellos se encuentran variantes como la generación a la que pertenece una solución, la cantidad de veces que se utilizan los operadores con los que cuenta el algoritmo y cuantas veces se ha aceptado una solución proveniente de cada operador.

Posterior a la ejecución de metaheurísticas y su recopilación de datos, se presenta la herramienta que constituye la solución al problema, la cual realiza un análisis exhaustivo de los datos recaudados. El análisis se presenta a través de distintas métricas definidas desde tres distintas miradas. Primero, se introducen las métricas de eficiencia, luego de efectividad y, finalmente, las métricas de operadores, un conjunto nuevo de métricas, las cuales serán dedicadas al análisis del comportamiento de los operadores utilizados en las ejecuciones de algoritmos poblacionales, tales como *mutation* y *crossover*.

3.1. Estrategia de solución

El trabajo a realizar tiene por objetivo el desarrollo de una librería capaz de medir distintos aspectos de rendimiento de una metaheurística. Para esto se definen 3 procesos clave, la ejecución y recolección de datos del algoritmo, la ejecución de métricas sobre dichos datos y, finalmente, la presentación de los resultados obtenidos.

3.2. Ejecución de algoritmos y recolección de datos

La primera fase requiere, primero que nada, de una forma estandarizada para ejecutar algoritmos sobre problemas conocidos, para este fin, se propone el uso de IOHprofiler [Doerr *et al.*, 2018]. El diagrama de la figura 16 muestra los componentes de esta suite, la cual provee los elementos IOHexperimenter e IOHalgorithm, siendo el primero de estos el que provee la suite de problemas a resolver y los métodos a través de los cuales recopilar datos de la resolución de estos en tiempo de ejecución. Mientras que, IOHalgorithm se

encarga de la ejecución de distintos algoritmos, y aunque estos incluyen algoritmos trayectoriales, para el objetivo de esta investigación, se utilizarán solo sus capacidades de algoritmos poblacionales.

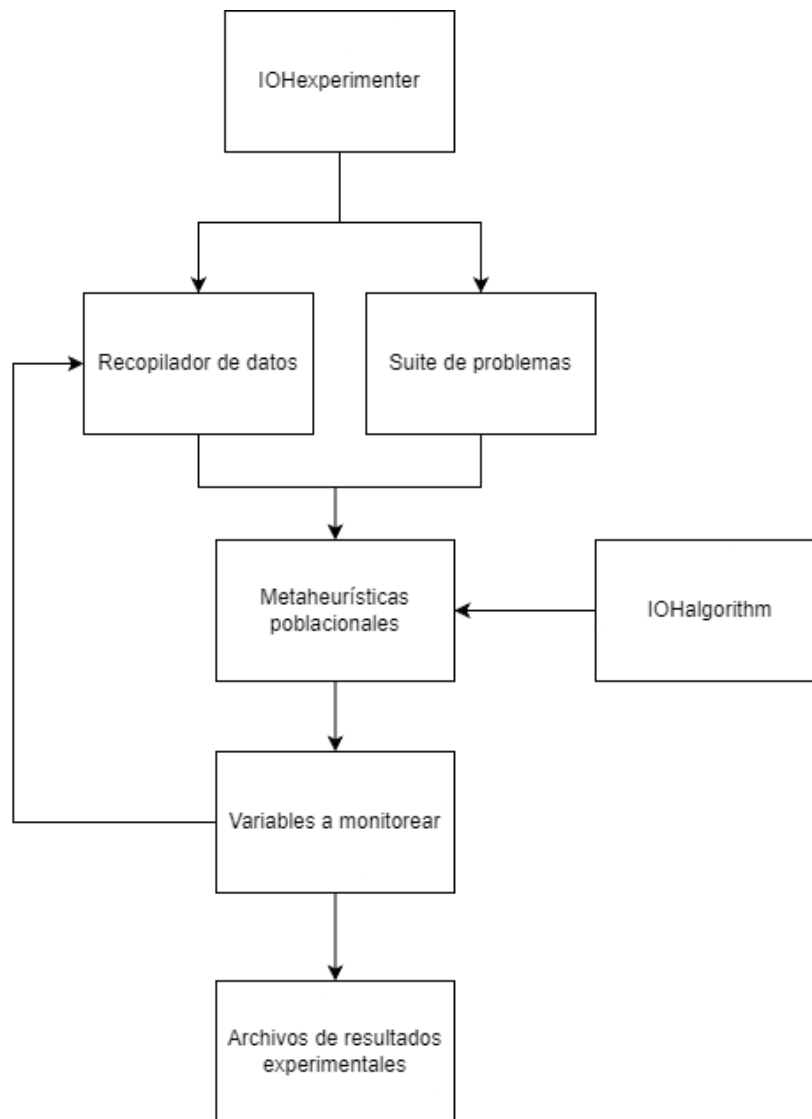


Figura 16: Diagrama de interacción de componentes IOHprofiler. Elaboración propia

Es sumamente importante notar que es el algoritmo el que deberá informar a la herramienta recopiladora sobre qué variables monitorear, ya que dependerá de la naturaleza del algoritmo que métricas serán computadas, y por ende, que se necesita para el correcto análisis.

3.2.1. Ejecución de metaheurísticas

Al momento de la ejecución del algoritmo, IOHalgorithm cuenta con ciertos algoritmos predefinidos, los cuales utilizan variables binarias para representar soluciones. Poniendo atención en las metaheurísticas poblacionales que presenta, IOHalgorithm entrega 8 distintos algoritmos, los cuales consisten de algoritmos evolutivos (EA) y genéticos (GA).

En más de una ocasión, los algoritmos disponibles en esta herramienta utilizan una distribución binomial condicional para definir distintos aspectos clave de la ejecución. Los elementos a definir incluyen las l posiciones de variables modificables y las probabilidades con las cuales aplicar operadores. La distribución se presenta en la ecuación 22:

$$Bin(n, p)_{>0} = Bin(n, p)(k) / (1 - (1 - p)^n) \quad (22)$$

donde n es la cantidad de variables y p , la tasa de mutación, calculada como $p = 1/n$. La tasa de mutación indica la probabilidad de mutación que se le entrega a todo $k \in n$. Ahora, esta distribución tiene la particularidad de ser condicional, lo cual, en este caso, aplica la condición de que se trabaja únicamente con k positivos, cualquier k no positivo recibe una probabilidad de mutación igual a 0 automáticamente. Esta condición se aplica de tal forma que no se generen soluciones en la descendencia que sean idénticas a los padres.

Los algoritmos evolutivos se enfocan en la utilización de un operador tradicional, como *bit-flip* sobre una población, mientras que los algoritmos genéticos, una subcategoría de los algoritmos evolutivos, introducen la fase de *crossover* en su funcionamiento. Los procedimientos de estos dos operadores se encuentran en los algoritmos 5, 6, 7 y 8.

El movimiento $flip_\ell$ (algoritmo 5) es el encargado de modificar los valores de las variables de una solución en particular. Esto lo hace a través de la selección de ℓ variables a modificar, la cual se hace sobre una distribución uniforme a menos que se especifique lo contrario. Es sobre estas variables donde se ejecuta la expresión $y_{ij}(1 - x_{ij}) \% 2$, la cual, aumenta el valor de la variable, y dado que se manejan variables binarias, utiliza el operador módulo para asegurar que se cumpla esta condición, efectivamente, ejecutando un *bit-flip* sobre estas variables.

Algorithm 5 $flip_\ell$ elige ℓ posiciones diferentes y cambia los valores en estas posiciones.

- 1: **Entrada:** $x \in \{0, 1\}^n$, $\ell \in \mathbb{N}$;
 - 2: Selecciona ℓ posiciones diferentes por pares $i_1, \dots, i_\ell \in [n]$ uniformemente al azar;
 - 3: $y \leftarrow x$;
 - 4: **for** $j = 1, \dots, \ell$ **do**
 - 5: $y_{ij} \leftarrow (1 - x_{ij}) \% 2$;
 - 6: **end for**
-

Luego se tienen distintas variantes de *crossover*. Primero, *One-point crossover* (algoritmo 6), el cual toma como entrada dos soluciones padres, para las cuales se decide un punto al azar a través de una distribución uniforme (a menos que se especifique lo contrario), luego, se crean copias de ambas soluciones. Finalmente, usando ambas copias, se crean dos nuevas soluciones reemplazando los valores previos por aquellos encontrados desde el punto seleccionado ℓ para cada una de las soluciones padre.

Algorithm 6 *One-point crossover* de dos padres $x^{(1)}$ y $x^{(2)}$

```
1: Entrada:  $x^{(1)}, x^{(2)} \in \{0, 1\}^n$ 
2: Muestra  $\ell \in [n]$  uniformemente al azar;
3: for  $i = 1, 2, \dots, \ell$  do
4:   Establece  $y_i^{(1)} \leftarrow x_i^{(1)}$  y  $y_i^{(2)} \leftarrow x_i^{(2)}$ ;
5: end for
6: for  $i = \ell + 1, \dots, n$  do
7:   Establece  $y_i^{(1)} \leftarrow x_i^{(2)}$  y  $y_i^{(2)} \leftarrow x_i^{(1)}$ ;
8: end for
```

En segundo lugar, existe *Crossover* $cross_c$ (algoritmo 7), el cual utiliza el parámetro c para ejecutar la distribución binomial antes planteada, de tal forma que, el conjunto de variables a intercambiar entre cada solución, es controlado por este parámetro.

Algorithm 7 *Crossover* $cross(x, x^*)$ con sesgo c

```
1:  $y \leftarrow x$ 
2: Muestra  $\ell \sim \text{Bin}_{>0}(n, c)$ 
3: Selecciona  $\ell$  posiciones diferentes  $\{i_1, \dots, i_\ell\} \in [n]$ 
4: for  $j = 1, 2, \dots, \ell$  do
5:   Establece  $y_{i_j} \leftarrow x_{i_j}^*$ 
6: end for
```

Finalmente, se tiene *uniform crossover* (algoritmo 8), el cual evalúa si hacer un cruce entre ambos padres para cada una de las variables. Para esto, el algoritmo utiliza un parámetro propio que define un umbral que permite decidir que elemento será utilizado en el cruce. El procedimiento para el cruce es el siguiente: para cada variable, a través de una distribución uniforme, obtiene un número aleatorio, y en caso de que este sea menor que el umbral definido, se toma un elemento del primer padre, en caso contrario, se toma un elemento del segundo padre. Así, se obtiene una única solución a partir de ambos padres.

Algorithm 8 UniformCrossover

Require: x_1, x_2 : vectores de entrada del mismo tamaño**Require:** y : vector de salida

```
1:  $n \leftarrow |x_1|$ 
2: for  $i \leftarrow 0$  hasta  $n - 1$  do
3:   if aleatorio_uniforme()  $< p_u$  then
4:     Agregar  $x_2[i]$  a  $y$ 
5:   else
6:     Agregar  $x_1[i]$  a  $y$ 
7:   end if
8: end for
```

3.2.1.1. Parámetros de ejecución

Para una ejecución controlada de los algoritmos, se definen parámetros, los cuales son compartidos por todos los algoritmos a ejecutar, sin embargo, previo a la ejecución de los algoritmos, la suite de problemas define la cantidad de variables n , es decir, el tamaño del problema a resolver.

Luego, todos los algoritmos poblacionales definidos utilizan los parámetros μ y λ para representar el tamaño de la población inicial y la población descendiente, correspondientemente, estos parámetros deben ser números naturales. Para la ejecución de operadores en estos algoritmos, se definen las probabilidades de ejecución para ambos tipos de operadores definidos (*mutation* y *crossover*) como parámetros arbitrarios con valores entre 0 y 1.

La variable a utilizar en la ecuación 22, l , también se define como un parámetro de ejecución para cada algoritmo, el cual tiene sus valores en los naturales entre 1 y n . Este puede ser modificado durante la ejecución de la metaheurística y se utiliza principalmente para la selección del conjunto de variables a ser modificadas por algún operador. En esta misma ecuación se utiliza p , otro parámetro, el cual define la tasa de mutación del algoritmo, el cual se encuentra entre 0 y 1.

En resumen, los parámetros de los algoritmos son:

Parámetro	Significado
n	Cantidad de variables / Tamaño del problema a resolver
μ	Tamaño de la población inicial en algoritmos poblacionales
λ	Tamaño de la población descendiente en algoritmos poblacionales
$p_{mutation}$	Probabilidad de ejecución del operador de mutación (valor entre 0 y 1)
$p_{crossover}$	Probabilidad de ejecución del operador de cruce (valor entre 0 y 1)
l	Variable que representa conjunto de variables a modificar en <i>mutation</i> y punto escogido en <i>one-point crossover</i> , valor natural entre 1 y n
p	Tasa de mutación del algoritmo (valor entre 0 y 1)

Tabla 3: Parámetros de ejecución de los algoritmos. Elaboración propia.

3.2.1.2. EA with Static Mutation Rate

Este algoritmo comienza por la inicialización de la población de μ soluciones iniciales y λ soluciones descendientes, con n variables aleatorias, parámetro entregado por la suite de problemas, el cual se define de forma previa a la ejecución.

Algorithm 9 EA with Static Mutation Rate

```

1: Inicialización: Muestrea  $x \in \{0, 1\}^n$  uniformemente al azar y evalúa  $f(x)$ ;
2: Optimización:
3: for  $t = 1, 2, 3, \dots$  do
4:   for  $i = 1, \dots, \lambda$  do
5:     Elige variables a modificar  $\ell^{(i)} \sim \text{Bin}_{>0}(n, 1/n)$ ;
6:     crea  $y^{(i)} \leftarrow \text{flip}_{\ell^{(i)}}(x)$ , y evalúa  $f(y^{(i)})$ ;
7:   end for
8: end for
9:  $x^* \leftarrow \arg \max\{f(y^{(1)}), \dots, f(y^{(\lambda)})\}$  (desempate seleccionado por el primer  $f(y^{(i)})$  máximo);
10: if  $f(x^*) \geq f(x)$  then
11:    $x \leftarrow x^*$ ;
12: end if

```

El ciclo de optimización involucra el uso de una distribución binomial condicional para definir cuáles son las l variables a modificar (esta distribución se encuentra en la ecuación 21) para cada una de las soluciones en la población, esta modificación se realiza a través de un *flip* (definido en el algoritmo 5). Al tener solo un movimiento, la exploración del algoritmo se ve definida por el parámetro l , ya que al manejar más variables a modificar, existe la posibilidad de generar soluciones muy distintas a su solución padre, y en caso de tener un l muy bajo,

se mantiene una gran parte de las variables sin modificación alguna.

Luego, como se aprecia en la línea 9 del algoritmo 9 se selecciona una solución para reemplazar a la actual. Esto se hace con selección elitista entre la descendencia generada, es decir, el individuo que sobrevive y reemplaza al actual, dada la condición de que la nueva solución sea mejor, es él con mayor aptitud entre los generados. Esta selección hace que el algoritmo se enfoque en la explotación de las mejores soluciones, ya que al no almacenar ninguna solución peor que la actual, no se permiten grandes movimientos en el espacio de búsqueda, a menos que estos sean una mejora en la aptitud de la solución.

Finalmente, se decide si reemplazar la solución actual. En caso de que la mejor solución encontrada en la población descendiente sea mejor que la solución actual, esta es reemplazada, en caso contrario, se mantiene la solución actual hasta, al menos, la próxima generación.

3.2.1.3. *Two-Rate EA*

Esta variante, desarrollada por [Doerr *et al.*, 2021] tiene la característica de manejar 2 tasas de mutación al mismo tiempo, las cuales son definidas por un parámetro particular para este algoritmo, r , definido arbitrariamente (con valor por defecto $r = 2$). La forma en la que se utilizan es bastante simple, una mitad de la población descendiente, es decir, $\frac{\lambda}{2}$, se genera con una tasa de mutación $r/2n$, mientras que la otra mitad lo hace con una tasa de mutación $2r/n$.

El parámetro r se actualiza en cada iteración, dependiendo de dos parámetros aleatorios q y s . s depende de la región en la que se encuentra la mejor solución generada, donde si esta fue generada con una tasa de mutación $r/2$ toma el valor de $3/4$, en caso contrario $1/4$. Luego se obtiene un valor aleatorio para $q \in [0, 1]$ y se comparan estos valores, en caso de que $q \geq s$, r toma el valor mayor entre $r/2$ y 2 , caso contrario, r toma el valor mínimo entre $2r$ y $n/4$.

El generar la población descendiente con distintas tasas de mutación, genera una mitad de la población con mayor probabilidad de cambiar los valores de sus variables, por lo tanto, con mayor potencial de exploración. Mientras que existe otra mitad con menos probabilidad de modificar sus variables, manteniendo más valores de sus soluciones padre, explorando menos. Además, al tener un r dinámico, se prioriza la exploración y explotación en distintos casos. Asumiendo que la solución seleccionada fue generada por la región con mayor tasa de mutación, al s tomar el valor $1/4$, existe mayor probabilidad de que $q \geq s$, y así, r tome un valor más bajo, bajando las tasas de mutación y priorizando la explotación para la siguiente generación. El caso contrario pasa en caso de que la solución se genere en la región con tasa de mutación más alta.

Algorithm 10 *Two-Rate EA*

```
1: Inicialización: Muestrea  $x \in \{0, 1\}^n$  uniformemente al azar y evalúa  $f(x)$ ;
2: Inicializa  $r \leftarrow r_{\text{init}}$ 
3: Optimización: for  $t = 1, 2, 3, \dots$  do
4:   for  $i = 1$  to  $\lambda/2$  do
5:     Muestrea  $\ell^{(i)} \sim \text{Bin}_{>0}(n, \frac{r}{2n})$ 
6:     Crea  $y^{(i)} \leftarrow \text{flip}_{\ell^{(i)}}(x)$  y evalúa  $f(y^{(i)})$ 
7:   end for
8:   for  $i = \lambda/2 + 1$  to  $\lambda$  do
9:     Muestrea  $\ell^{(i)} \sim \text{Bin}_{>0}(n, \frac{2r}{n})$ 
10:    Crea  $y^{(i)} \leftarrow \text{flip}_{\ell^{(i)}}(x)$  y evalúa  $f(y^{(i)})$ 
11:  end for
12:  $x^* \leftarrow \arg \max\{f(y^{(1)}), \dots, f(y^{(\lambda)})\} \triangleright$  desempate seleccionado uniformemente al azar
13: if  $f(x^*) \geq f(x)$  then
14:    $x \leftarrow x^*$ 
15: end if
16: if  $x^*$  ha sido creado con la tasa de mutación  $r/2$  then
17:    $s \leftarrow \frac{3}{4}$ 
18: else
19:    $s \leftarrow \frac{1}{4}$ 
20: end if
21: Muestrea  $q \in [0, 1]$  uniformemente al azar
22: if  $q \leq s$  then
23:    $r \leftarrow \max(\frac{r}{2}, 2)$ 
24: else
25:    $r \leftarrow \min(2r, \frac{n}{4})$ 
26: end if
```

3.2.1.4. EA with normalized standard bit mutation

En este caso, la variante de EA es definida en [Ye *et al.*, 2019] y se destaca por el uso de una distribución normal para calcular la probabilidad de mutación, y por ende l . La distribución es definida por el uso de r , parámetro que es utilizado en la media y varianza utilizadas en la distribución, estas están definidas como $r = pn$ y $pn(1-p) = r(1-r/n)$, respectivamente.

Además, este algoritmo actualiza el parámetro r en cada ciclo de optimización, al final de cada ciclo, a r se le asigna el valor de l , este, al ser muestreado desde una distribución normal condicional, permite al algoritmo explorar el espacio de búsqueda o explotar la aptitud de las soluciones dependiendo del valor obtenido.

Algorithm 11 *EA with normalized standard bit mutation*

```
1: Inicialización: Muestrea  $x \in \{0, 1\}^n$  uniformemente al azar y evalúa  $f(x)$ ;
2: Inicializa  $r \leftarrow r^{\text{init}}$ ;
3: Optimización:
4: for  $t = 1, 2, 3, \dots$  do
5:   for  $i = 1, \dots, \lambda$  do
6:     Muestrea  $\ell^{(i)} \sim \text{mín}\{N_{>0}(r, r(1 - r/n)), n\}$ 
7:      $y^{(i)} \leftarrow \text{flip}_{\ell^{(i)}}(x)$ 
8:     Evalúa  $f(y^{(i)})$ ;
9:   end for
10:   $i \leftarrow \text{mín}\{j \mid f(y^{(j)}) = \text{máx}\{f(y^{(k)}) \mid k \in [\lambda]\}\}$ ;
11:   $r \leftarrow \ell^{(i)}$ ;
12:  if  $f(y^{(i)}) \geq f(x)$  then
13:     $x \leftarrow y^{(i)}$ ;
14:  end if
15: end for
```

3.2.1.5. EA with normalized standard bit mutation and controlled variance

Este algoritmo se construye a partir del anterior, donde además de la variación en la tasa de mutación, se agrega un control a la varianza de dicha tasa para así evitar que la tasa varíe ampliamente en momentos de la ejecución donde es conveniente mantenerla constante en algún valor particular. En el caso de los experimentos realizados por los autores, este fenómeno se aprecia hacia el final de la ejecución, donde es óptimo mantener la tasa de mutación en su máximo, 1.

Este control de la tasa se realiza a través de los parámetros F y c , donde F es un parámetro estático el cual los autores definen como 0,98 y recomiendan mantener en valores mayores a 0,95, de tal forma que la tasa de mutación nunca pase a valores subóptimos. Mientras que c se inicializa como 0 y va evolucionando dependiendo del parámetro r , el cual, si al final de la iteración, se mantiene igual, se ejecuta $c = c + 1$, es decir, es un contador de la cantidad de iteraciones donde r no ha sido modificado.

De tal forma, se puede reemplazar la distribución normal condicional del algoritmo previo por $N_{>0}(r, F^c r(1 - r/n))$, la cual hace uso de estos cambios para controlar el cambio en la tasa de mutación.

Esta modificación también afecta a la capacidad del algoritmo para explotar la aptitud de las soluciones, debido a que se utiliza la varianza controlada para mantener la tasa de mutación alta, modificando constantemente las soluciones, de tal forma que se espera una mayor exploración.

Algorithm 12 *EA with normalized standard bit mutation and controlled variance*

```
1: Inicialización: Muestrea  $x \in \{0, 1\}^n$  uniformemente al azar y evalúa  $f(x)$ ;  
2: Inicializa  $r \leftarrow r^{\text{init}}$ ; ▷ usamos  $r^{\text{init}} = 2$ ;  
3: Inicializa  $c \leftarrow 0$ ;  
4: Optimización:  
5: for  $t = 1, 2, 3, \dots$  do  
6:   for  $i = 1, \dots, \lambda$  do  
7:     Muestrea  $\ell^{(i)} \sim \text{mín}\{N_{>0}(r, F^c r(1 - r/n)), n\}$   
8:      $y^{(i)} \leftarrow \text{flip}_{\ell^{(i)}}(x)$   
9:     Evalúa  $f(y^{(i)})$ ;  
10:   end for  
11:    $i \leftarrow \text{mín}\{j \mid f(y^{(j)}) = \text{máx}\{f(y^{(k)}) \mid k \in [\lambda]\}\}$ ;  
12:   if  $r = \ell^{(i)}$  then  
13:      $c \leftarrow c + 1$ ;  
14:   else  
15:      $c \leftarrow 0$ ;  
16:   end if  
17:    $r \leftarrow \ell^{(i)}$ ;  
18:   if  $f(y^{(i)}) \geq f(x)$  then  
19:      $x \leftarrow y^{(i)}$ ;  
20:   end if  
21: end for
```

3.2.1.6. EA with log-Normal self-adaptation on mutation rate

Muy similar al previamente presentado EA with normalized standard bit mutation en su objetivo de estabilizar la tasa de mutación, propuesto por [Kruisselbrink et al., 2011], tiene como su factor característico la expresión utilizada para calcular dicha tasa, ésta siendo definida en la ecuación 22.

$$p^{(i)} = \left(1 + \frac{1-p}{p} \cdot \exp(\gamma \cdot \mathcal{N}(0, 1))\right)^{-1} \quad (23)$$

Dicha ecuación se utiliza para cada individuo de la población descendiente. Esta tiene como parámetros p y γ . Además, cuentan con valores por defecto $p = 0,2$ y $\gamma = 0,22$, esto debido a una serie de experimentos realizados por los autores, donde se generan 10^6 valores de p bajo las condiciones $n = 1000$ y $\gamma = 0,22$. A partir de estos experimentos se observa que la varianza y el sesgo hacia ciertos valores de la distribución de p , presenta un comportamiento similar a una función seno, obteniendo valores altos en $p = 0,5$ y valores cercanos a 0 en $p = 0$ y $p = 1$, escogiendo un valor intermedio para p y manteniendo γ . Con esta información, es importante definir p de forma que se adapte a la necesidad de mantener una varianza más alta en sus valores (mayor exploración) o menor (mayor explotación). Como parte de la implementación, se actualiza el parámetro p con el cálculo de $p^{(i)}$ para la mejor solución encontrada en cada iteración.

Algorithm 13 EA with log-Normal self-adaptation on mutation rate

```

1: Inicialización: Muestrea  $x \in \{0, 1\}^n$  uniformemente al azar y evalúa  $f(x)$ ;
2:  $p = 0,2$ ;
3: Optimización:
4: for  $t = 1, 2, 3, \dots$  do
5:   for  $i = 1, \dots, \lambda$  do
6:      $p^{(i)} = \left(1 + \frac{1-p}{p} \cdot \exp(0,22 \cdot \mathcal{N}(0, 1))\right)^{-1}$ ;
7:     Muestrea  $\ell^{(i)} \sim \text{Bin}_{>0}(n, p^{(i)})$ ;
8:     crea  $y^{(i)} \leftarrow \text{flip}_{\ell^{(i)}}(x)$ 
9:     Evalúa  $f(y^{(i)})$ ;
10:   end for
11:    $i \leftarrow \min\{j \mid f(y^{(j)}) = \max\{f(y^{(k)}) \mid k \in [\lambda]\}\}$ ;
12:    $p \leftarrow p^{(i)}$ ;
13:    $x^* \leftarrow \arg \max\{f(y^{(1)}), \dots, f(y^{(\lambda)})\}$  (desempata favoreciendo el índice más pequeño);
14:   if  $f(x^*) \geq f(x)$  then
15:      $x \leftarrow x^*$ ;
16:   end if
17: end for

```

3.2.1.7. "Vanilla" Genetic Algorithm (GA)

Este algoritmo genético intenta ser lo más tradicional en su implementación, esto quiere decir que tiene 5 etapas principales, las cuales se aprecian en el algoritmo 13.

Algorithm 14 "Vanilla" Genetic Algorithm

```
1: Inicialización:
2: for  $i = 1, \dots, \mu$  do
3:   muestrea  $x^{(i)} \in \{0, 1\}^n$  uniformemente al azar y evalúa  $f(x^{(i)})$ ;
4: end for
5: Optimización:
6: for  $t = 1, 2, 3, \dots$  do
7:   Fase de selección de padres: Aplica selección por ruleta a  $\{x^{(1)}, \dots, x^{(\mu)}\}$  para seleccionar  $\mu$  individuos padres  $y^{(1)}, \dots, y^{(\mu)}$ ;
8:   Fase de cruce:
9:   for  $i = 1, \dots, \mu/2$  do
10:    con probabilidad  $p_c$  reemplaza  $y^{(i)}$  y  $y^{(2i)}$  por los dos descendientes que resultan de un one-point crossover, para un punto  $j \in [n]$  elegido al azar;
11:  end for
12:  Fase de mutación:
13:  for  $i = 1, \dots, \mu$  do
14:    Muestrea  $\ell^{(i)} \sim \text{Bin}(n, p_m)$ , establece  $y^{(i)} \leftarrow \text{flip}_{\ell^{(i)}}(y^{(i)})$ , y evalúa  $f(y^{(i)})$ ;
15:  end for
16:  Reemplazo:
17:  for  $i = 1, \dots, \mu$  do
18:    Reemplaza  $x^{(i)}$  por  $y^{(i)}$ ;
19:  end for
20: end for
```

Primero, al igual que en el caso evolutivo, se inicializa la población de soluciones con n variables aleatorias, para posteriormente comenzar el ciclo de optimización, donde se seleccionan padres con un método de ruleta basada en aptitud. Este es un proceso estocástico con la particularidad de que una solución con mayor aptitud tiene más oportunidades de ser seleccionada. Para este proceso se obtienen una cantidad de padres tal que la población mantenga su tamaño, siendo μ el tamaño de la población se seleccionan $\mu/2$ pares de individuos.

Luego, se aplican los operadores definidos, los cuales son *crossover* y *mutation*. El primero de estos se aplica en la forma expuesta en 6, escogiendo una variable aleatoria en la solución a la cual aplicar el operador, y utilizando la posición de esta variable en la solución como punto a partir del cual ejecutar *crossover*. *Mutation* en este caso se refiere a una modificación de la solución seleccionada, donde, por cada variable, se hace un flip_{ℓ} con cierta probabilidad. Finalmente, se reemplazan las soluciones nuevas en la población. A partir de este GA, se definen distintas variaciones a este proceso, generando nuevos algoritmos.

3.2.1.8. Fast Genetic Algorithm (fGA)

Basado en el trabajo de [Doerr *et al.*, 2017], utiliza el largo de mutación l , con el cual la solución a manejar solo ejecuta su operador de mutación de forma parcial, obteniendo una mayor exploración del espacio de búsqueda.

Esta implementación tiene por objetivo mantener una probabilidad baja de que el operador *flip* se ejecute en solo una variable, y mantener una probabilidad mayor de ejecutar el operador mientras más sean las variables a modificar. Similar a algoritmos previamente propuestos, esto se hace para aumentar la tasa de mutación en sectores cercanos al óptimo y facilitar la exploración de forma de encontrar el óptimo más rápidamente.

Algorithm 15 Fast Genetic Algorithm

```

1: Inicialización: Muestrea  $x \in \{0, 1\}^n$  uniformemente al azar y evalúa  $f(x)$ ;
2: Optimización:
3: for  $t = 1, 2, 3, \dots$  do
4:   for  $i = 1, \dots, \lambda$  do
5:     Muestrea  $\ell^{(i)} \sim D_{n/2}^\beta$ ;
6:     crea  $y^{(i)} \leftarrow \text{flip}_{\ell^{(i)}}(x)$ , y evalúa  $f(y^{(i)})$ ;
7:   end for
8:    $x^* \leftarrow \arg \max\{f(y^{(1)}), \dots, f(y^{(\lambda)})\}$  (desempata favoreciendo el índice más grande);
9:   if  $f(x^*) \geq f(x)$  then
10:      $x \leftarrow x^*$ ;
11:   end if
12: end for

```

El largo de mutación l , en este caso, se define a través de una ley potencial, lo que significa que los parámetros entregados al cálculo de este largo tienen una relación de la forma $A = cB^p$, siendo A y B las magnitudes relacionadas. En este caso, la relación está definida en las ecuaciones 23 y 24.

$$D_{n/2}^\beta = (C_{n/2}^\beta)^{-1} k^{-\beta} \quad (24)$$

$$C_{n/2}^\beta = \sum_{i=1}^{n/2} i^{-\beta} \quad (25)$$

Aquí, n es la cantidad de variables y β un parámetro arbitrario, que en este caso se utiliza como valor por defecto $\beta = 1,5$, decisión tomada por los autores originales para controlar el comportamiento de este cálculo y mantener el valor de la tasa de mutación dentro del óptimo (teniendo en mente el objetivo de mantener una tasa alta y controlada). Esta decisión

está dada por la relación de $C_{n/2}^\beta$ con la función ζ de Riemann, donde uno de los valores conocidos es $\zeta(1,5) = 2,612$. Otros valores conocidos de esta función, que podrían informar más fácilmente sobre el comportamiento de la ley potencial utilizada, incluyen $\zeta(2) = 1,645$ y $\zeta(3) = 1,202$. Esta expresión asigna a cada entero $k \in [n/2]$ una probabilidad de $D_{n/2}^\beta$ de mutar.

Finalmente, esta implementación de GA utiliza selección elitista entre la descendencia generada.

3.2.1.9. *The Self-Adjusting (1+(\lambda+\lambda)) GA*

Este algoritmo, introducido por [Doerr *et al.*, 2015] basa su comportamiento en dos elementos principales, el ajuste automático del parámetro λ y el cálculo de l a través de la distribución binomial encontrada en la ecuación 21. Además, este utiliza un parámetro constante F , denominado *Update Strength*, para controlar el algoritmo de ajuste de λ . Este parámetro es fijado en 1,5 por los autores originales. Esta decisión se hace a partir de la investigación realizada por [Auger, 2009]. En esta, se utiliza dicho valor para la ejecución del algoritmo en el que este se basa, sin embargo, no se realiza un análisis extensivo ni una sintonización de los parámetros más allá de que este cumpla con obtener una mejor solución cada 5 iteraciones, lo cual es considerado una buena proporción entre exploración y explotación. De todas formas, el parámetro F es modificable.

Para la fase de mutación, se aplica la distribución binomial para obtener l posiciones en las cuales realizar la operación. Luego de la mutación, se aplica una selección elitista entre las soluciones generadas, y en caso de que existan dos o más soluciones con la misma aptitud, en el algoritmo original se realiza una selección aleatoria sobre este conjunto, por concepto de ahorro de recursos, en esta implementación se selecciona la solución de mayor índice.

En la etapa de *crossover*, se utiliza la solución seleccionada en la etapa anterior x^* y la solución inicial x para generar un nuevo conjunto de soluciones descendientes. Luego, se vuelve a seleccionar sobre el conjunto generado.

Finalmente, se tiene una etapa de selección final, donde se aplican condiciones para la aceptación de una nueva solución y así incorporarla o no a la población, además de modificar el valor de λ dependiendo del caso.

En el primer caso, si la aptitud de la nueva solución es mejor que la solución actual, se reemplaza y λ pasa a ser el mayor número entre λ/F y 1. El segundo caso, donde la aptitud entre la nueva solución y la actual son iguales, la solución nueva reemplaza a la actual y λ se reemplaza por el mínimo entre $\lambda F^{1/4}$ y n . Finalmente, el caso donde la aptitud empeora, no se reemplaza la solución y se ejecuta el mismo cálculo para λ que en el caso anterior.

Algorithm 16 *The Self-Adjusting $(1+(\lambda+\lambda))$ GA*

```
1: Inicialización: Muestrea  $x \in \{0, 1\}^n$  uniformemente al azar y evalúa  $f(x)$ ;  
2: Optimización:  
3: for  $t = 1, 2, 3, \dots$  do  
4:   Fase de mutación:  
5:   Muestrea  $\ell \sim \text{Bin}_{>0}(n, \lambda/n)$ ;  
6:   for  $i = 1, \dots, \lambda$  do  
7:     crea  $y^{(i)} \leftarrow \text{flip}_\ell(x)$ , y evalúa  $f(y^{(i)})$ ;  
8:   end for  
9:    $x^* \leftarrow \arg \max\{f(y^{(1)}), \dots, f(y^{(\lambda)})\}$ ;  
10:  Fase de cruce:  
11:  for  $i = 1, \dots, \lambda$  do  
12:     $y^{(i)} \leftarrow \text{cross}_c(x, x^*)$   
13:    Evalúa  $f(y^{(i)})$ ;  
14:  end for  
15:   $y^* \leftarrow \arg \max\{f(y^{(1)}), \dots, f(y^{(\lambda)})\}$ ;  
16:  Fase de selección:  
17:  if  $f(y^*) > f(x)$  then  
18:     $x \leftarrow y^*$ ;  $\lambda \leftarrow \max\{\lambda/F, 1\}$ ;  
19:  else if  $f(y^*) = f(x)$  then  
20:     $x \leftarrow y^*$ ;  $\lambda \leftarrow \min\{\lambda F^{1/4}, n\}$ ;  
21:  else if  $f(y^*) < f(x)$  then  
22:     $\lambda \leftarrow \min\{\lambda F^{1/4}, n\}$ ;  
23:  end if  
24: end for
```

Finalmente, a continuación, se presenta una tabla que resume los algoritmos disponibles, sus operadores, parámetros y un breve análisis sobre qué elemento de estos algoritmos le permite explorar el espacio de búsqueda y explotar la aptitud de las soluciones.

Algoritmo	Operadores	Parámetros	Exploración	Explotación
<i>EA with Static Mutation Rate</i>	$flip_\ell$, selección elitista	λ, n	$flip_\ell$	Selección elitista
<i>Two-Rate EA</i>	$flip_\ell$, selección elitista con desempate aleatorio	r, λ, n, s	$flip_\ell$	Selección elitista
<i>EA with normalized standard bit mutation</i>	$flip_\ell$	r, λ, n, c, F	$flip_\ell$ y control de	Modifica solución solo si mejora aptitud
<i>EA with normalized standard bit mutation and controlled variance</i>	$flip_\ell$	r, λ, n	$flip_\ell$	Modifica solución solo si mejora aptitud y controla la varianza de p
<i>EA with log-Normal self-adaptation on mutation rate</i>	$flip_\ell$	p (inicial), γ, n, λ	$flip_\ell$	Modifica solución solo si mejora aptitud y controla la varianza de p
<i>"Vanilla" Genetic Algorithm (GA)</i>	$flip_\ell$, one-point crossover	$\mu, \lambda, p_{mutation}, p_{crossover}$	one-point crossover y selección por ruleta	$flip_\ell$
<i>Fast Genetic Algorithm (fGA)</i>	$flip_\ell$, selección elitista	λ, β, n	Cálculo de l a través de ley potencial	$flip_\ell$ y selección elitista
<i>The Self-Adjusting $(1+(\lambda+\lambda))$ GA</i>	$flip_\ell$, crossover con sesgo c	F, λ, n	crossover con sesgo, modifica parámetro F para cambiar prioridad de exploración o explotación	$flip_\ell$

Tabla 4: Comparación de Algoritmos. Elaboración propia.

3.2.2. Problemas disponibles

Los problemas definidos y utilizables en el framework *IOHexperimenter* son parte de dos grandes grupos, *Pseudo-boolean Optimization* (PBO) y *Black box optimization benchmarking* (BBOB).

El conjunto de problemas PBO se caracteriza por manejar un dominio binario para sus variables, es decir, son problemas del tipo $f: \{0, 1\}^n \rightarrow \mathbb{R}$.

Los problemas PBO disponibles fueron implementados por los autores del *framework* y se definen en las siguientes secciones, todos los problemas de este conjunto son consistentes en la utilización de una solución en forma de *string* de tamaño n donde cada carácter es una variable.

3.2.2.1. OneMax

El problema consiste en mantener la mayor cantidad de 1 en la solución. El problema, a pesar de su simpleza, al ser resuelto por metaheurísticas tiene una mejora constante cuando los valores de su función objetivo son bajos (Es decir, existen pocos 1 en el string solución). A medida que aumenta el valor de la función objetivo, los movimientos que puede realizar el algoritmo para mejorar la solución, disminuyen significativamente. Su función objetivo se expone a continuación:

$$\{0, 1\}^n \rightarrow [0..n]. x \mapsto \sum_{i=1}^n x_i \quad (26)$$

3.2.2.2. LeadingOnes

Similar a *OneMax*, este problema trata con la cantidad de 1 que existen en la solución, con la particularidad de que solo se ven tomados en cuenta la cantidad de 1 seguidos desde la primera posición, es decir, los 1 iniciales, donde el óptimo, al igual que antes, se encuentra cuando la solución está poblada por completo de 1. Esta función se expone en la ecuación 27

$$\{0, 1\}^n \rightarrow [0..n], x \mapsto \max\{i \in [0..n] \mid \forall j \leq i : x_j = 1\} = \sum_{i=1}^n \prod_{j=1}^i x_j \quad (27)$$

3.2.2.3. Linear Function with Harmonic Weights

$$\{0, 1\}^n \rightarrow \mathbb{R}, x \mapsto \sum_i ix_i \quad (28)$$

Este es un problema lineal casi idéntico a *OneMax*, con su distinción siendo la inclusión de un "peso armónico", indicado por la posición de la variable en el string solución.

3.2.2.4. *W-model transformations*

Propuestas por [Weise *et al.*, 2008] estas transformaciones se aplican, en este caso, a los problemas *OneMax* y *LeadingOnes*, dichas transformaciones modifican el problema a partir de 4 ejes distintos, por lo que la transformación final se representa como $W(m, \mu, \nu, r)$. Estos 4 ejes de modificación son:

- Variables ficticias.

El primer punto desde el cual se puede modificar el problema es una reducción de variables, o variables ficticias. Este procedimiento reduce el string solución $(x_1, \dots, x_n)$ a un substring $(x_{i_1}, \dots, x_{i_m})$, con m siendo el parámetro que define el largo de este substring, donde cada elemento $i_1, \dots, i_m$ es distinto y se refiere a una posición en el string solución original.

Por ejemplo, si se tiene como string solución original $X = 0010011011$, $n = 10$, $m = 3$, $i_1 = 3$, $i_2 = 5$ e $i_3 = 10$, el substring solución queda definido como 101, haciendo así que solo las variables en las posiciones 3, 5 y 10 influyan en el valor objetivo obtenido.

- Neutralidad.

Otra forma de modificar la solución, la cual, similarmente a las variables ficticias, busca la reducción del string que contiene la solución, donde el parámetro μ representa la cantidad de secciones en la cual dividir la solución, es decir, el string solución se divide en $m = n/\mu$ substrings, donde, a partir de cada substring generado, se revisa si es que este contiene una cantidad predominante de 1 o 0, donde aquel elemento predominante es el que se representa como el valor del substring, concatenando el valor resultante de cada substring se obtiene la nueva reducción del string solución.

En el caso donde $n/\mu \notin \mathbb{N}$, los elementos sobrantes se eliminan y no influyen en la función de evaluación.

Un ejemplo de esto es si se tiene el string solución $X = 0010011011$, $n = 10$, $\mu = 5$, el string solución en su forma reducida es 01111, ya que los substrings de tamaño n/μ producidos son 00, 10, 01, 10, 11.

- Epistasia.

Esta modificación tiene por objetivo la perturbación de la solución. Con esa idea en mente, el string solución se divide en secciones de tamaño ν , con cada substring ge-

nerado se genera un nuevo string basándose en una permutación e_ν , esta se escoge de forma que la distancia de Hamming obtenida de esta sea $\nu - 1$.

En este caso, para cualquier elemento sobrante se ajusta la función e_ν para que funcione de la forma $e_{n-\nu[n/\nu]}$, es decir, se ajusta a la cantidad de elementos sobrantes.

Para ilustrar esta modificación se puede usar el ejemplo donde $X = 11100011101$ y $\nu = 4$, los substrings generados son $e_4(1110) = 0011$, $e_4(0011) = 0110$ y $e_3(101) = 110$ y así el string perturbado resulta en 00110110110.

- **Perturbación de aptitud.**

Finalmente, esta transformación no apunta a la solución, sino a su aptitud, esto a través de la introducción de una nueva forma de calcular el valor objetivo utilizando la función r entregada como parámetro. La forma en la que se calcula la aptitud entonces pasa a ser $r(f(x))$, es decir, se ejecuta la función r por sobre el valor de aptitud calculado por la función original, de forma que el problema a resolver se transforma en $r \circ f$.

En la investigación original se definen dos restricciones para la función r , la primera siendo que el valor máximo a entregar por esta debe ser idéntico al valor máximo entregado por la función original, por otra parte, se requiere que r sea una permutación de la función original.

Cada una de estas puede ser aplicada independiente de la otra, es decir, puede haber casos donde se usen las 4 transformaciones o solo una selección parcial de estas.

Las combinaciones definidas en IOHexperimenter son las siguientes:

- OneMax+W([n/2],1,1,id)
- OneMax+W([0.9n],1,1,id)
- OneMax+W([n], $\mu=3$,1,id)
- OneMax+W([n],1, $v=4$,id)
- OneMax+W([n],1,1,r1)
- OneMax+W([n],1,1,r2)
- OneMax+W([n],1,1,r3)
- LeadingOnes+W([n/2],1,1,id)
- LeadingOnes+W(0.9n,1,1,id)
- LeadingOnes+W([n], $\mu=3$,1,id)
- LeadingOnes+W([n],1, $v=4$,id)
- LeadingOnes+W([n],1,1,r1)
- LeadingOnes+W([n],1,1,r2)
- LeadingOnes+W([n],1,1,r3)

3.2.2.5. *Low Autocorrelation Binary Sequences (LABS)*

Es un problema combinatorial que busca encontrar secuencias binarias de tal forma que tengan mínima autocorrelación [Pakebusch y Mertens, 2016]. La autocorrelación para una secuencia $S = (s_1, \dots, s_N)$, con $s_i = \pm 1$ se define a continuación:

$$C_k(S) = \sum_{i=1}^{N-k} s_i s_{i+k} \quad (29)$$

Con $k = 0, 1, \dots, N - 1$. También se define el concepto de energía para el problema, cuya expresión matemática se encuentra en la ecuación 30.

$$E(S) = \sum_{k=1}^{N-1} C_k^2(S) \quad (30)$$

De esta forma, se define el problema LABS como el encontrar la secuencia S y largo N que maximice la expresión:

$$F(S) = \frac{N^2}{2E(S)} \quad (31)$$

3.2.2.6. Ising Model

Este modelo describe un sistema de N espines representados por $s_i = \pm 1$, estos espines se distribuyen en una malla, donde cada punto (i, j) tiene una fuerza de interacción asociada J_{ij} . El objetivo del problema es encontrar configuraciones de esta malla de forma que la energía sea mínima. Con ese objetivo en mente se define la función objetivo del problema en la ecuación 32.

$$- \sum_{i < j} J_{ij} S_i S_j - \sum_{i=1}^n h_i S_i \quad (32)$$

Con h_i siendo un campo magnético externo, el cual no se considera en el diseño de los problemas disponibles, por lo tanto, es igual a 0.

La aplicación de estos problemas en *IOHexperimenter* se basa en tratar la malla como un grafo no dirigido $G(V, E)$ con V el conjunto de espines, con la salvedad de que el conjunto de espines se trata como variables de decisión, es decir, pasan de $s_i = \pm 1$ a $s_i = \{0, 1\}^n$ [Doerr et al., 2020].

La forma de este grafo la entrega la variante del problema a trabajar, *IOHexperimenter* provee tres. Primero, un anillo unidimensional, donde la secuencia S define un grafo cíclico de

forma circular, luego un torus, donde la malla de dos dimensiones y de tamaño N^2 , con exactamente 4 vecinos por vértice. Finalmente, se define otra variante de dos dimensiones, esta vez triangular, esto entrega la particularidad de que al construir la malla, los vectores unitarios deben formar ángulos de $\frac{2\pi}{3}$, aplicando estas reglas, se consigue un grafo regular donde cada vértice tiene 6 vecinos.

3.2.2.7. Maximum Independent Vertex Set

Este problema utiliza un grafo de n vértices $G = ([n], E)$, donde un set de vértices independiente es aquel subconjunto de $[n]$ donde ninguno de los vértices contenidos en él es vecino directo de cualquier otro vértice en el conjunto. El objetivo es encontrar el conjunto de vértices independientes $V' \subset [n]$ más grande posible. Para la definición de este problema se utiliza $V' = \{i \in [n] \mid x_i = 1\}$, de tal forma, que la función objetivo se puede modelar como la maximización de la expresión:

$$\sum_i x_i - n \cdot \sum_{i,j} x_i x_j e_{i,j} \quad e_{ij} = \begin{cases} 1 & \text{si } \{i, j\} \in E, \\ 0 & \text{en caso contrario.} \end{cases} \quad (33)$$

La instancia utilizable en *IOHexperimenter* fue construida de forma escalable y simple estructura, la cual se puede visualizar en la figura 17.

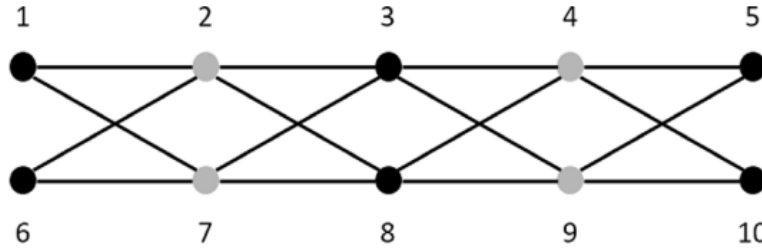


Figura 17: Instancia para *Maximum Independent Vertex Set* con $n = 10$ [Doerr et al., 2020]

3.2.2.8. N-reinas

Un clásico problema matemático, propuesto como el “problema de las 8 reinas” por [Bezzel, 1848], es un desafío donde se busca colocar n reinas en un tablero de ajedrez de $n \times n$ de manera que ninguna reina ataque a otra. Esto significa que no puede haber dos reinas en la misma fila, columna o diagonal.

La definición matemática utilizada corresponde a la encontrada en la ecuación 34, sujeta al conjunto de restricciones encontrado en 35

$$Max \sum_{ij} x_i \quad (34)$$

$$\begin{aligned} \sum_i x_j &\leq 1 \quad \forall j \in \{1, \dots, N\} \\ \sum_j \sum_{i=i} x_{ij} &\leq 1 \quad \forall i \in \{1, \dots, N\} \\ \sum_{i=1} x_{ij} &\leq 1 \quad \forall k \in \{3, 4, \dots, 2N-3, 2N-1\} \\ \sum_{i=k} x_{ij} &\leq 1 \quad \forall i \in \{3, 4, \dots, N\} \\ x_{ij} &\in \{0, 1\} \quad \forall i, j \in \{1, \dots, N\} \end{aligned} \quad (35)$$

3.2.2.9. Concatenated Trap

Este problema tiene por principal característica su espacio de búsqueda deceptivo, esto a partir de que la función matemática a resolver se transforma en una función por partes que genera una o más cuencas en el espacio de búsqueda [Deb y Goldberg, 1993], llevando a distintos algoritmos a caer en óptimos locales y haciendo más difícil la búsqueda de un óptimo global. En una función lineal se puede graficar fácilmente el fenómeno generado por las funciones "trampa", tal como se expone en la figura 18.

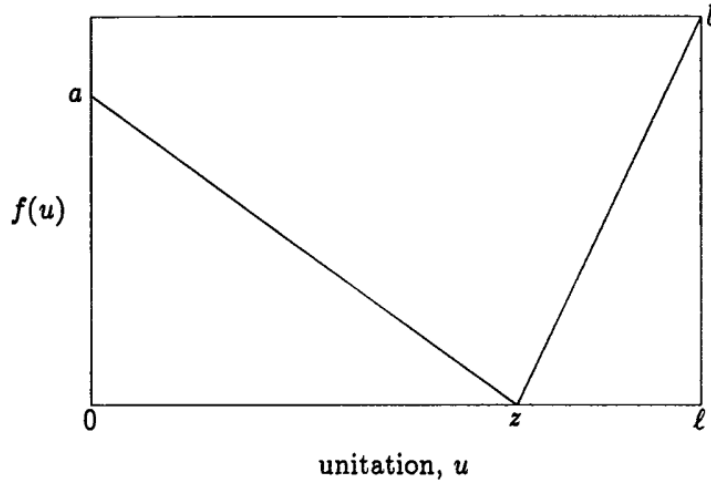


Figura 18: Función trampa con óptimo local en $u = 0$ pero óptimo global en $u = l$ [Deb y Goldberg, 1993]

La forma en la que se aplica este tipo de funciones en *IOHexperimenter* es en forma de un problema donde se participan un string de largo n en segmentos de largo k y concatenando

n/k funciones trampa, las cuales se encargan de evaluar cada segmento generado. Cada función trampa está definida:

$$f_k^{\text{tap}}(u) = \begin{cases} 1 & \text{si } u = k, \\ \frac{k-1-u}{k} & \text{en caso contrario.} \end{cases} \quad (36)$$

Con k un parámetro arbitrario con valor por defecto $k = 5$.

3.2.2.10. *NK-Landscapes*

Propuestos por [Kauffman y Levin, 1987], *NK-Landscapes* es un *framework* para modelar la interacción entre genes en sistemas biológicos a través del entendimiento de como las relaciones entre genes afectan el genotipo final y la proyección de las posibilidades de aptitud de este a un espacio cartesiano. Su definición depende de dos conceptos clave, sus componentes (N), los cuales representan genes binarios con valores 0 o 1, y las interacciones entre estos (K), donde K representa la cantidad de genes con los cuales cada gen individual interactúa.

Visto desde el punto de vista algorítmico, K indica la complejidad del espacio de búsqueda, esto es ilustrado tomando los casos límite, con $K = 0$ ninguna variable depende de otra y su contribución a la aptitud de la solución es individual haciendo que la aptitud se mueva de forma lineal con los cambios en la solución, de forma contraria, con $K = N - 1$ todas las variables interactúan y dependen entre sí, definiendo un espacio de búsqueda sumamente complejo con una gran cantidad de máximos locales.

Entonces, su aplicación como problema utiliza N como la cantidad de variables y K un parámetro arbitrario para la creación de una matriz aleatoria de relaciones entre variables, en el caso de *IOHexperimenter* se utiliza $K = 1$, es decir, por cada variable en $\{N\}$, se selecciona aleatoriamente una segunda variable para que mantengan una interacción. La aptitud de una configuración es definida por N subfunciones F_i que transforman un set de $k + 1$ variables a un real entre $(0, 1)$, es decir, $F_i : \{0, 1\}^{k+1} \rightarrow \mathbb{R}$, los valores de dichas funciones se promedian, entregando la aptitud de la secuencia de N genes [Altenberg, 1996]. Para el caso de *IOHexperimenter* los valores de las funciones son generados independientemente con una distribución uniforme entre $(0, 1)$.

A modo de resumen, se tiene la tabla 5, la cual lista los problemas anteriormente examinados, la cual incluye el título del problema y un resumen de sus características.

Problema	Características
<i>OneMax</i>	Problema de optimización más simple. Cuenta el número de unos en una cadena binaria.
<i>LeadingOnes</i>	Cuenta el número de unos consecutivos desde el inicio de la cadena. Dependencia posicional fuerte.
<i>Linear Function with Harmonic Weights</i>	Función lineal con pesos armónicos decrecientes.
<i>W-model</i> Variables ficticias	Introduce variables que no afectan al valor de aptitud. Evalúa robustez frente a dimensionalidad innecesaria.
<i>W-model</i> Neutralidad	Diferentes configuraciones de bits producen el mismo valor de aptitud. Evalúa comportamiento en presencia de un espacio de búsqueda con alta presencia de valles.
<i>W-model</i> Epistasis	La contribución de una variable al valor de aptitud depende de los valores de otras. Evalúa capacidad para manejar dependencias entre variables.
<i>W-model</i> Perturbación de aptitud	Introduce ruido en la función de aptitud. Evalúa robustez frente a un espacio de búsqueda sumamente rugoso.
<i>Low Autocorrelation Binary Sequences (LABS)</i>	Busca secuencias binarias que minimicen la suma de productos entre bits a diferentes distancias. La autocorrelación mide cuánto se parece una secuencia a versiones desplazadas de sí misma.
<i>Ising Model</i>	Modelo de sistemas de espín. Problema con múltiples óptimos locales. Estructura de vecindad compleja.
<i>Maximum Independent Vertex Set</i>	Búsqueda del conjunto independiente más grande en un grafo.
N-reinas	Problema de colocación de N reinas en tablero NxN sin amenazas. Múltiples soluciones, restricciones complejas.
<i>Concatenated Trap</i>	Función engañosa compuesta por subfunciones trampa concatenadas. Diseñada para engañar a la búsqueda gradual.

Tabla 5: Problemas de optimización y sus características principales

3.2.3. Recolección de datos

La recolección de datos es un aspecto clave de esta investigación, particularmente el cómo se hace y los resultados que entrega.

Parte de la suite IOHexperimenter es, también, recolectar datos de la resolución de problemas. Este proceso es realizado con un manejo de variables contenidas en instancias de un problema. El elemento encargado de esto es el *logger*, que por defecto, mantiene la recolección de la solución actual, la mejor solución encontrada hasta el momento, la cantidad de evaluaciones registradas, las restricciones incumplidas y la penalización asociada a estas. Sin embargo, este conjunto de datos no es suficiente para realizar un análisis completo, por lo tanto, se realiza una modificación, la cual consiste en agregar una estructura variable para el *logger*. Aquí, el algoritmo entrega sus parámetros por referencia al *logger* en tiempo de ejecución, con lo que se puede monitorear toda variable que el algoritmo mantenga en memoria.

Ahora bien, el *logger* es configurable desde varios aspectos, más allá de recolectar variables personalizadas a monitorear. Para esta investigación hay dos configuraciones de particular relevancia, la capacidad de almacenar la solución actual con sus variables en el archivo final y la frecuencia con la que se registran datos.

En el primer caso, es relevante el almacenar cada variable asociada a la solución registrada, aunque esto implica un aumento significativo en el tamaño del archivo y la cantidad de información almacenada en este. Esto es útil para métricas tales como *entropy diversity*, la cual depende totalmente de la estructura de los valores de variables encontrados en la solución.

Por otra parte, la frecuencia de monitoreo se basa en el uso de la función de evaluación y el comportamiento de la recolección de datos está dado por *triggers*, elementos que definen las condiciones para registrar una evaluación en el sistema de monitoreo. Estos componentes pueden seguir las condiciones predefinidas, tales como registrar datos en todas las evaluaciones, cada cierta cantidad de evaluaciones, o solo en caso de mejora. En casos de algoritmos poblacionales, se realiza también una distinción sobre la estructura de la solución para saber si dicha solución debe ser evaluada (y, por lo tanto, almacenada en el *logger*). Esta distinción consiste en revisar si es que la solución entregada es idéntica a la anteriormente evaluada. En tal caso, la solución no es evaluada y se continúa con la ejecución, con el objetivo de no aumentar el tamaño y complejidad de los datos entregados.

Finalmente, el *logger*, al finalizar la ejecución del algoritmo, genera un archivo en texto plano con una matriz de datos recolectados, con las columnas representando las propiedades observadas y las filas representando distintas evaluaciones.

Para esta investigación, en cada ejecución de algoritmos poblacionales, se monitorearon los siguientes elementos:

- Evaluación

- Valor evaluado de la solución actual
- Valor evaluado de la mejor solución hasta el momento
- Cantidad de restricciones incumplidas
- Penalización por restricciones incumplidas
- Generación actual
- Cantidad de *crossovers* realizados
- Cantidad de mutaciones realizadas
- Cantidad de *crossovers* que mejoraron la solución
- Cantidad de mutaciones que mejoraron la solución
- Variables del problema

donde la generación y la cantidad de ejecuciones (totales y exitosas) por operador son variables entregadas por el algoritmo en tiempo de ejecución para el análisis de métricas.

3.3. Librería de métricas para metaheurísticas poblacionales (LibMetMHP)

La metodología propuesta para la evaluación de los algoritmos anteriormente propuestos se enfoca en el cálculo de métricas de desempeño. A partir de los datos obtenidos desde la ejecución de metaheurísticas para resolución de problemas combinatoriales, se diseñan y proponen distintas métricas para analizar el comportamiento de estos algoritmos.

La ejecución de este conjunto de métricas contempla la ingesta de los datos, el cálculo de la métrica y su presentación gráfica o numérica.

Las métricas propuestas se dividen en 3 secciones, dos de ellas (efectividad y eficiencia) inspiradas en la literatura y una tercera (métricas de operadores) diseñada para esta investigación.

Las representaciones gráficas y valores presentados para las siguientes métricas se elaboran a partir de una ejecución de *Vanilla GA* (algoritmo 14) sobre el problema *OneMax* con 64 variables.

3.3.1. Métricas de eficiencia

Las métricas de eficiencia estudian la capacidad de converger hacia una solución candidata en un tiempo acotado, a través de la exploración y explotación del espacio de búsqueda. En el marco de esta investigación hubo un enfoque hacia la convergencia del algoritmo y la diversidad de soluciones en el espacio de búsqueda. Las métricas implementadas se exponen

a continuación.

3.3.1.1. Aptitud en el tiempo

Una forma simple de visualizar el comportamiento de un algoritmo, es el cambio en la aptitud de las soluciones encontradas en un tiempo determinado. Para esto, se grafica la aptitud de la mejor solución encontrada en el eje y , hasta una cierta generación, representada en el eje x , tal como se aprecia en la figura a continuación:

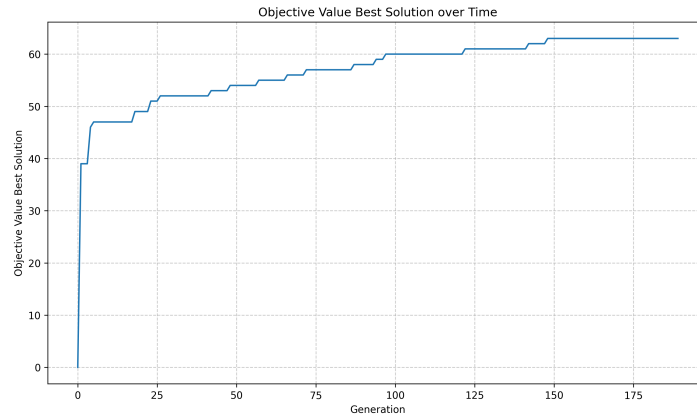


Figura 19: Mejor aptitud encontrada para cada generación. Elaboración propia

De forma más granular, también se puede analizar el valor de la función objetivo en el tiempo, ya sea por la mejor solución encontrada en cada generación (no necesariamente la mejor hasta dicha generación) o netamente la solución utilizada en cada evaluación. Graficar la solución para cada evaluación resulta en una visualización de la siguiente forma, donde el eje x representa la evaluación en la que se observa el valor de la función objetivo representado en el eje y :

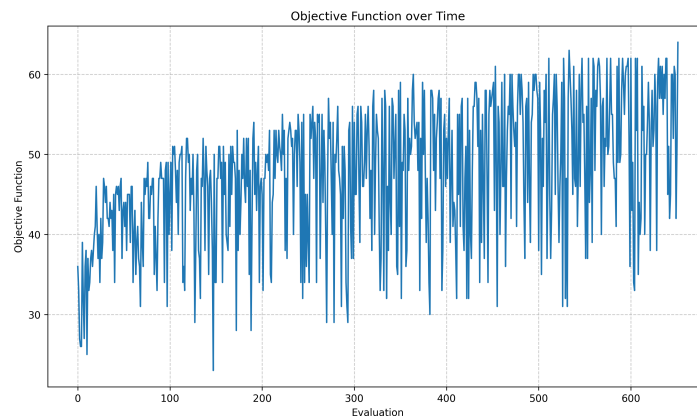


Figura 20: Aptitud evaluada en cada evaluación. Elaboración propia

3.3.1.2. *Rate of fitness change*

Esta métrica es la más básica observada en el estudio de la convergencia de un algoritmo hacia una solución candidata. Tal como expresa su nombre, esta métrica estudia la diferencia absoluta entre la aptitud de la solución actual y la de una solución óptima conocida, lo que se expresa en la ecuación 37.

$$|f_i - f_{opt}| \quad (37)$$

Además, en caso de no conocer el óptimo, se puede usar el cambio relativo de error, es decir, el cambio respecto de la iteración previa. Esto se implementa utilizando la mejor solución encontrada hasta la generación i y la generación $i - 1$, matemáticamente:

$$|f_i - f_{i-1}| \quad (38)$$

Dado que se maneja una población de soluciones, esta métrica se implementa de dos formas distintas. En el primer caso, se hace el análisis sobre el mejor individuo encontrado hasta la generación i , es decir, para cada generación, se toma f_i como la aptitud más adecuada encontrada hasta ese punto y se compara con la mejor aptitud encontrada en la generación anterior. Este primer algoritmo se observa en el pseudocódigo 17, y gráficamente, se presenta en la figura 21, donde el eje x representa la generación donde se observa el cambio en el valor de aptitud, el valor calculado se observa en el eje y de la figura.

Algorithm 17 cambioEnValorObjetivoPorGeneración

- 1: Transformar datos iniciales en valorObjetivoPorGeneración
 - 2: Inicializar vector vacío cambioEnValorObjetivo
 - 3: $maximoAnterior \leftarrow 0$
 - 4: $maximoActual \leftarrow 0$
 - 5: **for** cada par de generaciones consecutivas en valorObjetivoPorGeneración **do**
 - 6: $maximoAnterior \leftarrow$ como el elemento máximo en la primera generación del par
 - 7: $maximoActual \leftarrow$ como el elemento máximo en la segunda generación del par
 - 8: Añadir ($maximoActual - maximoAnterior$) a cambioEnValorObjetivo
 - 9: **end for**
 - 10: **return** cambioEnValorObjetivo
-

Aquí, se observa que la aptitud está en constante cambio, por lo que el algoritmo y las soluciones que este produce indican un buen balance de exploración y explotación. Para el segundo enfoque, se agrupan los individuos por generación en un arreglo ordenado, de esta forma, a cada elemento de este conjunto se le asigna una posición. A medida que se genera la población descendiente para cada generación, se asignan las soluciones a estas posiciones en orden, de forma que sea rastreable el comportamiento de las soluciones generadas por cada agente de búsqueda. Así, se compara la aptitud del individuo encontrado en la posición

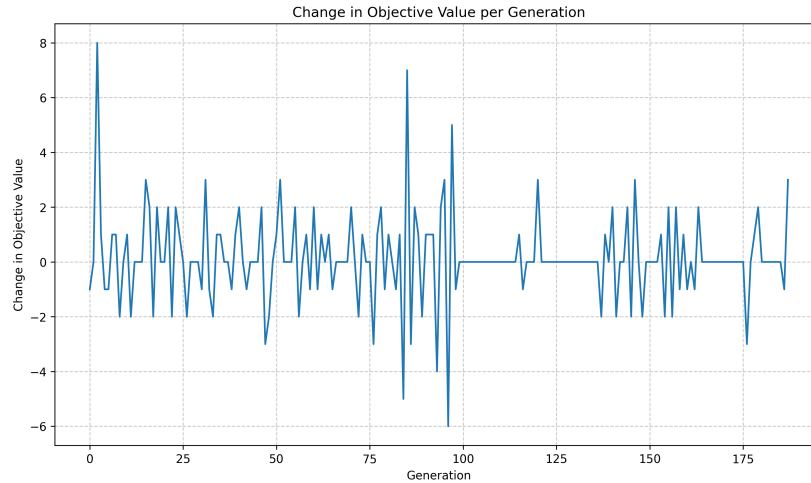


Figura 21: Cambio de aptitud entre generaciones. Elaboración propia

x , en la iteración i con el óptimo conocido. Visualmente, esta implementación genera una curva por cada posición x en la población. Esta implementación se observa en el algoritmo 18, con su gráfico correspondiente en 22, donde el eje x representa la generación y el eje y el valor de aptitud evaluado, con cada línea de color representando un agente de búsqueda distinto.

Algorithm 18 tasaDeCambioPorIndividuoPorGeneración

```

1: function tasaDeCambioPorIndividuoPorGeneración(óptimo)
2:   Transformar datos iniciales en valorObjetivoPorGeneracion
3:    $cantIndividuosGen \leftarrow$  cantidad de individuos en población descendiente
4:   Inicializar matriz valorObjetivoPorIndividuoPorGeneración
5:   for  $i = 0$  hasta  $cantIndividuosGen - 1$  do
6:     Añadir vector vacío a valorObjetivoPorIndividuoPorGeneración
7:   end for
8:   for  $i = 0$  hasta tamaño de  $valorObjetivoPorGeneracion - 1$  do
9:     for  $j = 0$  hasta tamaño de  $valorObjetivoPorGeneracion[i] - 1$  do
10:      Añadir ( $valorObjetivoPorGeneracion[i][j] - optimo$ ) a valorObjetivoPorIndividuoPorGeneración[j]
11:    end for
12:  end for
13:  return valorObjetivoPorIndividuoPorGeneración
14: end function

```

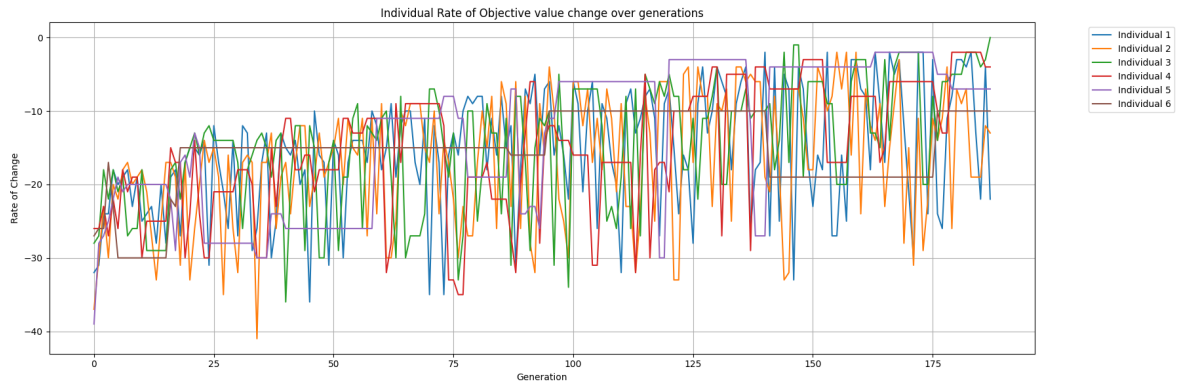


Figura 22: Cambio de aptitud para cada individuo por generación. Elaboración propia

3.3.1.3. Convergencia

La convergencia del algoritmo hacia una solución candidata es la métrica más comúnmente utilizada fuera de la aptitud de solución y el tiempo de ejecución. Su definición matemática e implementación dependen de tres factores distintos, tal como se expresa en la ecuación 5, estos siendo la aptitud de la solución en la iteración a evaluar, la aptitud óptima y la aptitud de la solución previa, con lo que se obtiene una métrica que informa sobre la velocidad con la que el algoritmo se mueve hacia soluciones de mayor aptitud. Además, en [Li y Zhao, 2005] se formula el cálculo de tal forma que se puede prescindir del valor óptimo de aptitud.

En la implementación, estos valores se utilizan en base a las mejores soluciones encontradas en cada generación, es decir f_i representa la aptitud de la mejor solución encontrada en la generación i , de igual forma, f_{i-1} representa la aptitud de la mejor solución encontrada en la generación $i - 1$, mientras que f_{opt} sigue representando la aptitud óptima. A diferencia de la ecuación encontrada en 5, se normaliza el cálculo restando el valor de la convergencia a 1, haciendo el cambio en convergencia 0 cuando la aptitud no cambie.

Gráficamente, permite analizar de forma rápida el avance del algoritmo hacia la solución óptima del problema, lo cual se evidencia en 23, donde el eje x representa la generación donde se realiza el análisis, mientras que el eje y representa la convergencia observada en ese instante. Aquí, es fácil ver que en la última generación se observa un valor de 1, indicando que la búsqueda concluye al encontrar una solución óptima al problema.

Y utilizando el error relativo, es decir, eliminando el uso del máximo previo, se tiene la figura 24.

De forma homóloga a la diferencia absoluta, en caso de no tener acceso al óptimo, se puede reemplazar el cálculo de la métrica por la ecuación 7, de tal forma que se utiliza la aptitud de la mejor solución en la siguiente generación.

El gráfico de esta métrica no es utilizable en su forma natural, ya que depende de que la solución y su aptitud estén en constante cambio, no dando una buena representación del

Algorithm 19 Calcular Convergencia con óptimo

```
function calcularConvergenciaConOptimo(óptimo)
  Inicializar una lista vacía llamada convergencia
  Agrupar valores de aptitud por generación. Llamar a función valorObjetivoPorGeneracion
  for cada generacion en valorObjetivoPorGeneracion do
    Establecer pasoConvergencia a 0
    Encontrar el valor maximo en la generacion actual
    Encontrar el valor maximo en la generacion previa
    Calcular pasoConvergencia como  $1 - \frac{\text{optimo} - \text{maximo}}{\text{optimo} - \text{maximo previo}}$ 
    Añadir pasoConvergencia a la lista convergencia
  end for
  return la lista convergencia
end function
```

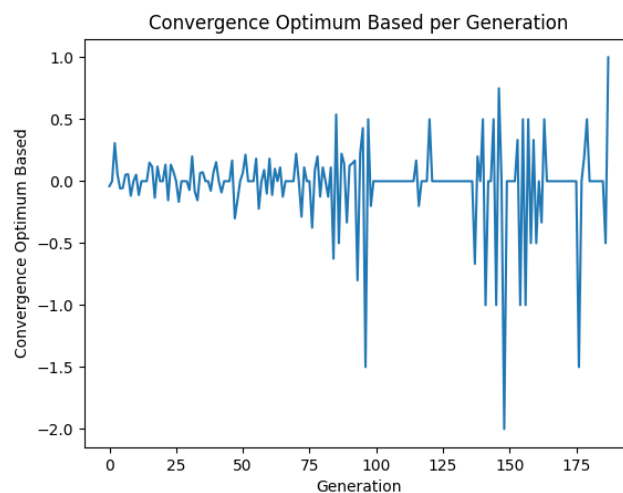


Figura 23: Tasa de convergencia con óptimo conocido. Elaboración propia

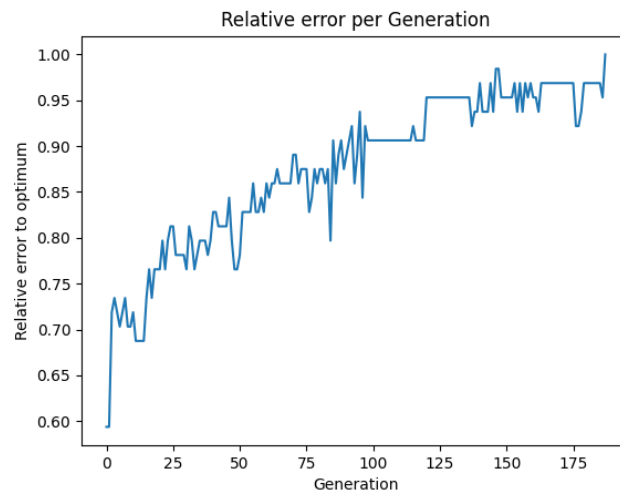


Figura 24: Error relativo con óptimo conocido. Elaboración propia

Algorithm 20 Calcular Convergencia sin óptimo

```
function calcularConvergenciaSinOptimo
  Inicializar una lista vacía llamada convergencia
  Declarar una variable pasoConvergencia
  for j desde 1 hasta el penúltimo elemento de data do
    if data[j] es igual a data[j - 1] then
      Asignar 0 a pasoConvergencia
    else
      Calcular pasoConvergencia como  $\frac{\text{data}[j + 1] - \text{data}[j]}{\text{data}[j] - \text{data}[j - 1]}$ 
    end if
    Añadir pasoConvergencia a la lista convergencia
  end for
  return la lista convergencia
end function
```

comportamiento del algoritmo. Para combatir esto, se proponen dos soluciones. Primero, realizando el análisis para las soluciones que maneja cada individuo de la población, en cada generación, (de la misma forma que en la figura 22) resultando en la siguiente visualización, donde el eje y representa el valor de la convergencia con error relativo observado en la generación indicada en el eje x , esto para cada agente de búsqueda, representados en las distintas líneas:

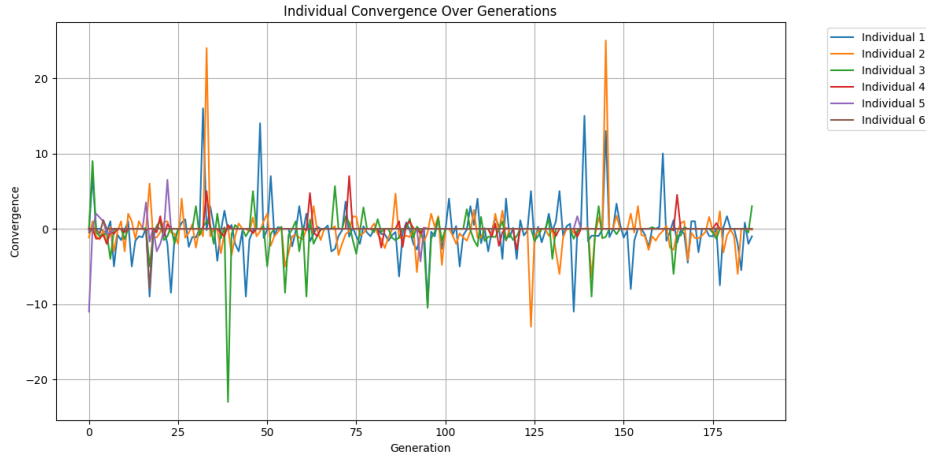


Figura 25: Convergencia sin uso de óptimo, por individuo y por generación. Elaboración propia

Segundo, utilizando el acercamiento al error relativo, se propone el uso de la mejor solución por generación para el cálculo reformulado de la forma:

$$\frac{|f_{i+1} - f_i|}{|f_i|} \quad (39)$$

de tal manera, no depende del cambio constante de la solución, ya que el cálculo se ve afectado inmediatamente luego de un cambio en la aptitud. Utilizando este cálculo se obtiene la figura 26, la cual muestra las diferencias inmediatas entre soluciones.

Sin embargo, esta forma de visualizar la convergencia puede verse exagerada en caso de que los valores cambien drásticamente o se manejen números de gran orden de magnitud. Para solventar esta situación, se puede utilizar *geometric average convergence rate*, matemáticamente expresado en la ecuación 8. A nivel de implementación, esta métrica utiliza la aptitud óptima, la aptitud de la solución inicial obtenida y la aptitud de la solución en la iteración i . Dada la necesidad de la aptitud de una solución inicial, esta métrica se implementa tomando los valores de la mejor solución encontrada por generación, es decir, f_0 sería la aptitud de la mejor solución encontrada hasta la primera generación y f_i la aptitud de la mejor solución encontrada hasta la iteración i .

El uso del factor $x^{1/j}$ hace que esta métrica priorice la convergencia temprana del algoritmo

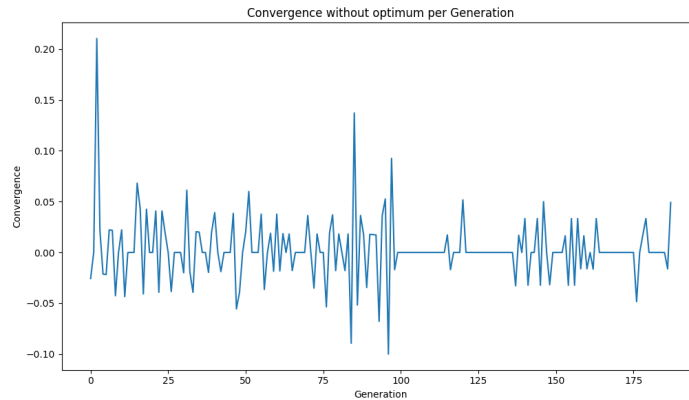


Figura 26: Convergencia sin óptimo adaptada a error relativo. Elaboración propia

Algorithm 21 tasaGeométricaDeCambioEnAptitud

```
1: function tasaGeométricaDeCambioEnAptitud(óptimo)
2:   Inicializar vector tasaGeometrica
3:   tasaGeneracion  $\leftarrow 0$ 
4:   for  $j = 0, \dots, \text{cantidadDeGeneraciones}$  do
5:      $x \leftarrow (\text{óptimo} - \text{mejor aptitud hasta generación } j) / (\text{óptimo} - \text{mejor aptitud en población inicial})$ 
6:     tasaGeneracion  $\leftarrow 1 - x^{1/j}$ 
7:     Añadir tasaGeneracion a tasaGeometrica
8:   end for
9:   return tasaGeometrica
10: end function
```

hacia una solución, y permite que el análisis sobre el resto de generaciones sea normalizado. Esta métrica se visualiza en la figura 27, donde se puede apreciar un fuerte decaimiento en la tasa geométrica (representada en el eje y) luego de la generación 25 (observable en el eje x). Solo en la última generación, donde se encuentra el óptimo, se vuelve a observar un cambio grande en esta representación.

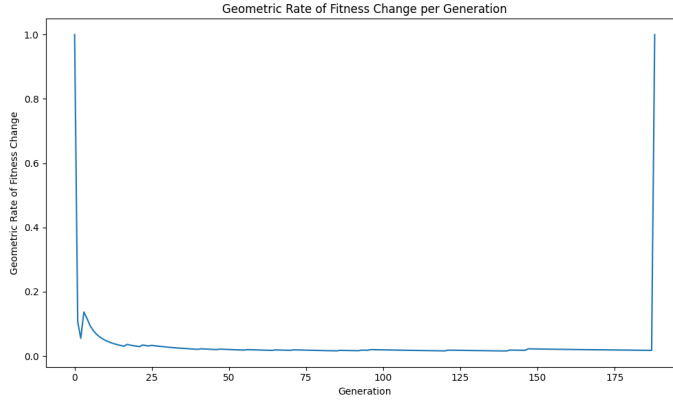


Figura 27: *Geometric average convergence rate*. Elaboración propia

3.3.1.4. *E-value*

Esta métrica define la eficiencia de un algoritmo como su capacidad de llegar a soluciones con aptitud suficientemente buena (definido por un parámetro arbitrario) en el menor tiempo posible. Para la implementación de este se utilizan las bases matemáticas ideadas por [Mora-Meliá *et al.*, 2015], por lo tanto, se utilizan directamente las ecuaciones 9 y 10. Gracias a las modificaciones realizadas por [El-Ghandour y Elbeltagi, 2018], se reemplaza la medición del tiempo de ejecución en la ecuación 9 por una expresión que utiliza las características del problema a resolver para mantener la constancia de los resultados, esta se presenta a continuación:

$$\eta_{convergence} = \frac{V}{\binom{n}{2}} \quad (40)$$

donde V indica la cantidad de evaluaciones para finalizar la ejecución del algoritmo y se utiliza una combinatoria sobre la cantidad de variables n , donde se define que las variables toman valores binarios por naturaleza de los problemas descritos previamente. Por lo tanto, el proceso de calcular esta métrica involucra, inicialmente, agrupar datos de valores de aptitud por generación. Luego, requiere chequear cuántas soluciones candidatas encontradas por el algoritmo cumplen con la condición de estar por sobre el parámetro de umbral de aptitud que define una solución “exitosa”. Con este dato y la cantidad de soluciones evaluadas, se obtiene $\eta_{quality}$. Por otra parte, utilizando la cantidad de evaluaciones y la combinatoria previamente definida, se calcula $\eta_{convergence}$. Este proceso se plasma en el algoritmo 22.

Algorithm 22 E-Value

```
function e-value(umbralDeAptitud)
  Agrupar valores de aptitud por generación. Llamar a función valorObjetivoPorGeneración
  Inicializar  $n\_successful \leftarrow 0$ 
  Inicializar  $n\_sim \leftarrow 0$ 
  Inicializar  $V \leftarrow$  Evaluaciones totales
  Inicializar  $n \leftarrow$  Variables del problema
  for  $i = 0, \dots, \text{valorObjetivoPorGeneración.size()}$  do
    for  $j = 0, \dots, \text{valorObjetivoPorGeneración}[i].\text{size()}$  do
      if  $\text{valorObjetivoPorGeneración}[i][j] \geq \text{umbralDeAptitud}$  then
        Incrementar  $n\_successful$  en 1
      end if
    Incrementar  $n\_sim$  en 1
  end for
  end for
  Calcular  $n\_quality$  como  $\frac{n\_successful}{n\_sim}$ 
  Calcular  $n\_convergence$  como  $\frac{V}{\binom{n}{2}}$ 
  Calcular  $e$  como  $\frac{n\_quality}{n\_convergence}$ 
  return  $e$ 
end function
```

Teniendo en cuenta que esta métrica entrega tan solo un valor numérico, es útil entender los posibles resultados de este valor. Primero, $n_{quality}$ depende de las evaluaciones que cumplen con el criterio arbitrario de su aptitud asociada, asumiendo que todas las evaluaciones cumplen con esto, $n_{successful} = n_{sim}$, por lo tanto, el valor máximo de $n_{quality}$ es 1. De igual forma, el cálculo propuesto para $n_{convergence}$ depende de la cantidad de evaluaciones sobre la cantidad posible de soluciones a generar, asumiendo que se explora el total de posibles soluciones, $V = \binom{n}{2}$, por lo tanto, también, su valor es como máximo 1. Teniendo en cuenta esto, uno de los casos borde es donde se obtiene un valor de 1, indicando que no hubo un proceso eficiente ni ineficiente, se obtuvieron soluciones suficientemente buenas en cada evaluación, pero se explora todo el espacio. El resto de casos, donde los valores de $n_{quality}$ y $n_{convergence}$ están entre 0 y 1. Aquí, el comportamiento es de la siguiente forma, en caso de que $n_{quality} > n_{convergence}$, se obtiene un número mayor a 1, indicando eficiencia en la búsqueda de soluciones, ya que se obtiene una cantidad de soluciones aceptables mayor en comparación al espacio explorado. En el caso contrario, se obtiene un número entre 0 y 1, indicando que la búsqueda es ineficiente debido a una gran exploración del espacio sin encontrar una cantidad de soluciones suficientemente buenas que justifiquen tal exploración.

Con esto en cuenta, para el caso propuesto para ejemplificar el uso de estas métricas, y utilizando un valor aceptable del 90 % del valor objetivo óptimo, se tiene $n_{successful} = 245$, $n_{sim} = 1134$, $V = 652$ y $\binom{64}{2} = 2016$, por lo tanto, $n_{quality} = 0,216$ y $n_{convergence} = 0,323$, resultando en $e = 0,668$, representando baja eficiencia.

3.3.1.5. *Entropy Diversity*

La diversidad es una parte fundamental para entender el comportamiento de un algoritmo. Sin embargo, es de difícil análisis debido a que se debe tener algún criterio para la división del espacio por el que se mueve la búsqueda de soluciones. Para superar esta dificultad, se utiliza la implementación de MDS realizada por [Wang y Boyer, 2013], con la cual se obtiene una reducción de la dimensionalidad de soluciones a un espacio cartesiano tridimensional.

La implementación de MDS depende, primero que nada, del cálculo de una matriz de distancias entre las soluciones candidatas, para esto se utiliza la distancia de Hamming, es decir, la distancia es representada por la cantidad de elementos que difieren en los dos conjuntos (soluciones) [Metcalf y Casey, 2016]. Para esto se transforma el conjunto de soluciones encontradas por el algoritmo, definido como toda solución que fuera evaluada por la función de evaluación, en una matriz X de tamaño $V \times n$, con V la cantidad de evaluaciones realizadas. Teniendo la matriz X , se calcula la matriz D de distancias entre cada solución utilizando la distancia de Hamming. Esta matriz D se entrega a la función de MDS definida en el algoritmo 23

El resultado de este algoritmo es una matriz Y de tamaño $V \times 3$, donde cada fila representa una reducción de la solución correspondiente en X . Dado que el espacio utilizado en este trabajo es tridimensional, se definen 8 cuadrantes en los cuales dividir el espacio de búsqueda. Para definir estos 8 cuadrantes, se utilizan los puntos medios entre los valores mínimos y máximos encontrados para cada una de las 3 coordenadas. Luego, con estos valores se puede definir a qué cuadrante pertenece cada solución, evaluando donde se encuentran las coordenadas que le corresponden respecto de cada punto medio. Finalmente, se crea el conjunto de datos que representa las áreas a las que pertenece cada solución, tal como se demuestra en 24.

Una vez obtenido el conjunto de datos sobre las áreas a las que corresponden las soluciones, éstas se agrupan por generación, permitiendo realizar el análisis por agente de búsqueda. En el caso poblacional general se puede asignar una posición x en el arreglo (población) para identificar individuos como agentes de búsqueda.

Luego de agrupar las asignaciones de áreas por generación, esta información se puede utilizar para calcular la métrica utilizando la formulación matemática encontrada en la ecuación 14. Para esto se identifica la cantidad de elementos en cada área del espacio de búsqueda en la generación, con lo cual se puede aplicar 14, con Z_i la cantidad de agentes de búsqueda en el área i , N la cantidad de agentes de búsqueda totales y $Q = 8$. De tal forma, se puede obtener un valor para la diversidad percibida en cada generación.

Nuevamente, es sumamente importante comprender los límites de este cálculo. Aquí, el caso propuesto como ejemplo considera poblaciones de tamaño 8. El caso de entropía mínima, donde todo se concentra en una zona, da valores para $\frac{8}{8} \times \log_e \left(\frac{8}{8} \right) = 0$, por lo tanto, el mínimo de entropía, sería en este caso, 0. En el caso del máximo, se debe tener en cuenta que dado que la población es menor a la cantidad de áreas, el valor máximo tampoco es exacto.

Algorithm 23 Algoritmo iterado de Levenberg-Marquardt (ILMA).

- 1: **Entrada:** Matriz de distancias $D = [D_{i,j}], 1 \leq i, j \leq N$;
- 2: Número máximo de iteraciones M ;
- 3: **Salida:** Códigos MDS $X = (x_1, \dots, x_N)^T$;
- 4: **Etapas de inicialización:**
- 5: Elegir aleatoriamente una entrada no diagonal D_{i_0, j_0} en D ;
- 6: Establecer $x_{i_0} = (0, 0, \dots, 0)^T$;
- 7: Establecer $x_{j_0} = (D_{i_0, j_0}, 0, \dots, 0)^T$;
- 8: Inicializar el conjunto de restricción $A = \{i_0, j_0\}$;
- 9: **while** $|A| < N$ **do**
- 10: Elegir aleatoriamente $j^* \notin A$;
- 11: Utilizar el algoritmo estándar de Levenberg-Marquardt para encontrar x_{j^*} que minimize:

$$\sum_{i \in A} (\|x_{j^*} - x_i\| - D_{j^*, i})^2;$$

- 12: Añadir j^* a A , establecer x_{j^*} en X ;
- 13: **end while**
- 14: **iniciar la etapa de ajuste:**
- 15: **for** $t \leftarrow 1$ to M **do**
- 16: Generar una permutación aleatoria $(p_1, p_2, \dots, p_N)$ de enteros de 1 a N ;
- 17: **for** $s \leftarrow 1$ to N **do**
- 18: Sacar p_s de A ;
- 19: Utilizar el algoritmo estándar de Levenberg-Marquardt para encontrar x_{p_s} que minimice:

$$\sum_{i \in A} (\|x_{p_s} - x_i\| - D_{p_s, i})^2;$$

- 20: Añadir p_s de nuevo a A , actualizar x_{p_s} en X ;
 - 21: **end for**
 - 22: **if** $\Delta \text{Stress}^* / \text{Stress}^* < \epsilon$ **then**
 - 23: **break**;
 - 24: **end if**
 - 25: **end for**
 - 26: **hecho**
-

Algorithm 24 División del espacio de búsqueda y asignación por solución

```
1: MatrizMDS  $\leftarrow$  MDS()
2: Inicializar puntos medios;  $midX, midY, midZ \leftarrow 0$ .
3: Inicializar extremos;  $maxX, minX, maxY, minY, maxZ, minZ \leftarrow 0$ .
4: Inicializar vector de 8 espacios de búsqueda searchSpaceAreas.
5: Calcular puntos medios;  $midX \leftarrow \frac{maxX+minX}{2}$ .
6:  $midY = \frac{maxY+minY}{2}$ .
7:  $midZ = \frac{maxZ+minZ}{2}$ .
8:
9: for 0, . . . , Tamaño matrizMDS do
10:   Chequear sector al que pertenece la solución.
11:   Agregar solución al espacio correspondiente en vector searchSpaceAreas.
12: end for
```

Algorithm 25 Entropy diversity

```
1: function entropyDiversity
2:   Llamar a la función SearchSpaceDivisionByGeneration para agrupar datos por generación.
3:    $diversity \leftarrow 0$ 
4:   Inicializar el arreglo areaScores de tamaño 8 con todos los elementos en 0
5:   Inicializar una lista vacía areaScoresVector
6:   for cada generación en searchSpaceAreasByGenerations do
7:      $diversity \leftarrow 0$ 
8:     Reiniciar todos los elementos de areaScores  $\leftarrow 0$ 
9:     for cada área en la generación actual do
10:      Incrementar el valor en areaScores correspondiente al área
11:    end for
12:    for cada puntuación en areaScores do
13:      if la puntuación  $\neq 0$  then
14:        Actualizar  $diversity \leftarrow diversity + \left( \frac{areaScores[j]}{N} \right) \cdot \log \left( \frac{areaScores[j]}{N} \right)$ 
15:      end if
16:    end for
17:    Añadir  $-diversity$  a areaScoresVector
18:  end for
19:  return areaScoresVector
20: end function
```

Asumiendo entonces un caso donde la población mantiene soluciones en zonas distintas, e ignorando las zonas donde no se encuentra ninguna solución (para evitar llegar a un valor indeterminado) $\frac{1}{8} \times \log_e \left(\frac{1}{8}\right) = -0,260$, luego $E(t) = -(-0,260 \times 8) = 2,08$, siendo este el valor máximo que puede tomar la entropía en este caso. Generalizando, el valor máximo que puede obtener esta métrica en condiciones ideales es cercano a 2, con un mínimo de 0, con un número mayor indicando mayor distribución de las soluciones.

Gráficamente se puede apreciar la entropía por generación para el caso propuesto en 28. En esta figura se observa que el valor de la diversidad, si bien no se mantiene estable, nunca llega a 0 ni tiende a ser menor a 1, lo cual es un buen indicador de la exploración del algoritmo. Aunque en las últimas generaciones observadas es fácil ver que se produce uno de los puntos más bajos de diversidad, para luego aumentar fuertemente en los momentos cercanos a la obtención del óptimo, indicando que, en periodos de baja diversidad, se intensifican y se llevan las soluciones a sectores con mejor aptitud, para luego diversificar nuevamente sus posiciones.

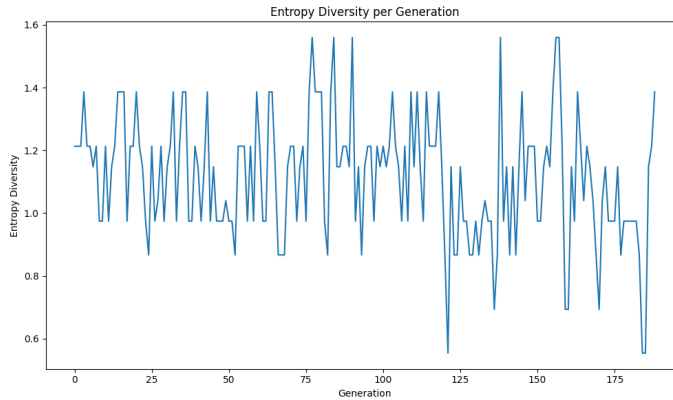


Figura 28: *Entropy diversity*. Elaboración propia

3.3.2. Métricas de operadores

Para entregar mayor granularidad al análisis de algoritmos, en esta investigación se diseñaron distintas métricas apuntadas a los operadores utilizados en la ejecución de metaheurísticas.

A partir del uso de algoritmos poblacionales, estas métricas se diseñan con los operadores de *crossover* y *mutation* en mente.

3.3.2.1. Tasa de éxito

Esta métrica mide la proporción encontrada entre la cantidad de veces que un operador cumple con mejorar la solución que se le entrega (Op_{mejora}) y la cantidad de veces que este se aplica (Op_{total}). Ésta métrica se calcula de la siguiente manera:

$$\frac{Op_{mejora}}{Op_{total}} \quad (41)$$

Con Op_{mejora} la cantidad de veces que se observa una mejora en la aptitud de la solución luego de la ejecución del operador y Op_{total} la cantidad total de veces ejecutado el operador.

Gráficamente, se muestra en forma de barras comparativas por operador, con valores máximos de 1, mientras que el mínimo es 0. Para el caso propuesto se tiene la figura 29

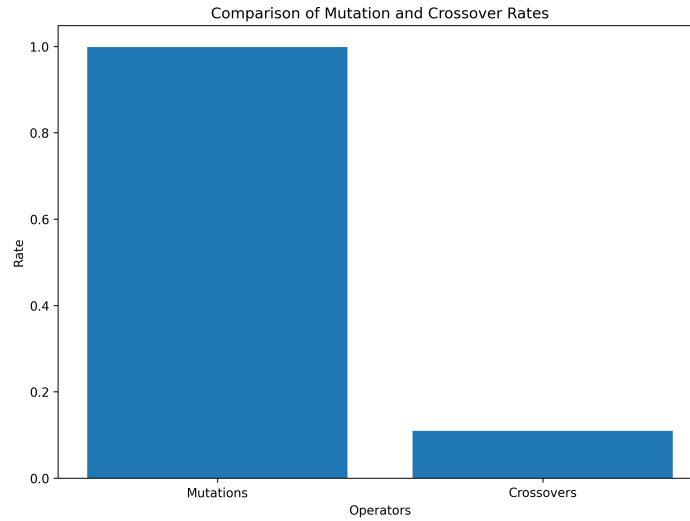


Figura 29: Tasas de éxito para operadores en GA. Elaboración propia

3.3.2.2. Tasa de mejora y deterioro

En este caso se filtran las evaluaciones donde el operador es capaz de modificar la aptitud de la solución, ya sea en forma de una mejora o un empeoramiento. El análisis de todos los casos donde la solución es modificada se utiliza para tener en cuenta situaciones en las que el algoritmo toma una ruta de ejecución que puede resultar en mayor exploración del espacio de búsqueda a partir de empeorar la aptitud.

Teniendo el conjunto de valores de aptitud en los cuales se observa la modificación, se puede comparar el cambio respecto de la solución antes de su modificación y así, obtener las tasas de cambio inmediatas. Con estos valores se calcula la media del cambio provocado por el operador. El algoritmo asociado a esta métrica se encuentra en el pseudocódigo 26.

Para el caso propuesto, se ejecuta el algoritmo y se extraen los valores para las medias de mejora y deterioro, además de los arreglos que contienen las diferencias encontradas entre cada ejecución del operador, con lo que se elabora un *boxplot* que permite observar y analizar como el operador modifica la aptitud de la solución a través de las distintas ejecuciones

de este. Para el caso de las mejoras, se puede observar en la figura 30, mientras que para los deterioros, se presenta en 31

Algorithm 26 Cálculo de la tasa de mejora y deterioro de operadores

```
1: function rateOfImprovementOperators ▷ Calcular mejora y deterioro de los operadores
2:   Inicializar arreglos mejoraOperador vacío
3:   for  $i = 0, \dots$ , cantidad de evaluaciones do
4:     if Evaluacion en primera generacion then
5:       mejoraOperador[0][ $i$ ]  $\leftarrow$  0
6:     else
7:       mejoraOperador[generacion][ $i$  % Cantidad individuos poblacion descendiente]  $\leftarrow$ 
Aptitud evaluacion actual – Aptitud evaluacion anterior
8:     end if
9:   end for

                                                                 ▷ Cálculo de medias

10:  mediaMejora  $\leftarrow$  0
11:  mediaDeterioro  $\leftarrow$  0
12:  contadorMejora  $\leftarrow$  0
13:  contadorDeterioro  $\leftarrow$  0
14:  for  $i = 0, \dots$ , Tamaño mejoraOperador do
15:    for  $j = 0, \dots$ , Tamaño generacion en mejoraOperador do
16:      if mejoraOperador[ $i$ ][ $j$ ]  $> 0$  then
17:        mediaMejora  $\leftarrow$  mediaMejora + mejoraOperador[ $i$ ][ $j$ ]
18:        contadorMejora + +
19:      else if mejoraOperador[ $i$ ][ $j$ ]  $< 0$  then
20:        mediaDeterioro  $\leftarrow$  mediaDeterioro + mejoraOperador[ $i$ ][ $j$ ]
21:        contadorDeterioro + +
22:      end if
23:    end for
24:  end for
25:  mediaMejora  $\leftarrow$  mediaMejora/contadorMejora
26:  mediaDeterioro  $\leftarrow$  mediaDeterioro/contadorDeterioro
27:  return (mediaMejora, mediaDeterioro)
28: end function
```



Figura 30: Tasas de mejora para operadores en GA. Elaboración propia

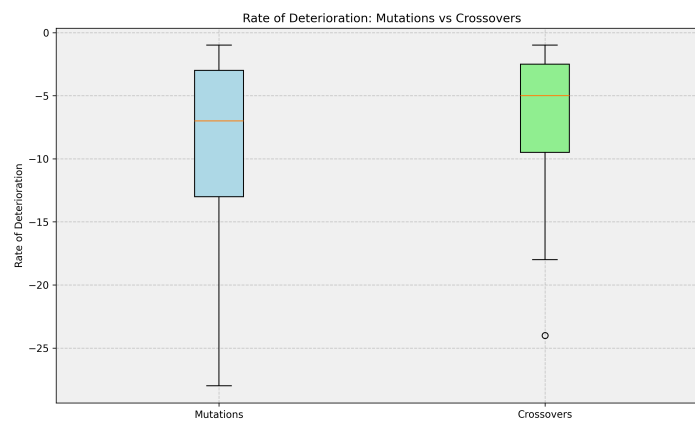


Figura 31: Tasas de deterioro para operadores en GA. Elaboración propia

3.4. Resumen

A modo de síntesis de las métricas presentadas, se elabora la tabla 6, la cual contiene, para cada métrica, su nombre, ecuación de cálculo (o referencia a su algoritmo) y su objetivo.

Métrica	Ecuación y/o algoritmo	Objetivo
Mejor aptitud en el tiempo	Función de evaluación del problema	Mostrar la calidad de la mejor solución encontrada hasta el momento
Aptitud evaluada en el tiempo	Función de evaluación del problema	Medir el rendimiento inmediato para cada evaluación.
Cambio de aptitud entre generaciones por individuo	$ f_i - f_{opt} $ (ecuación 37)	Evaluar la velocidad de mejora o deterioro, por individuo, entre generaciones consecutivas para la solución encontrada en comparación al óptimo.
Cambio de aptitud entre generaciones	$ \max f_i - \max f_{i-1} $ (ecuación 38)	Evaluar la velocidad de mejora o deterioro entre generaciones consecutivas para mejor solución encontrada.
Convergencia con óptimo	$\frac{ f_{opt} - f_i }{ f_{opt} - f_{i-1} }$ (ecuación 5)	Medir la capacidad del algoritmo para guiar la búsqueda hacia mejores soluciones en base a la aptitud de estas.
Error relativo con óptimo	$\frac{ f_i - f_{opt} }{ f_{opt} }$	Calcular el error porcentual respecto al óptimo conocido
Convergencia sin uso de óptimo	$\frac{ f_{i+1} - f_i }{ f_i - f_{i-1} }$	Medir el cambio en la mejor solución entre las tres últimas generaciones de cada agente de búsqueda.
Error relativo sin uso de óptimo	$\frac{ f_{i+1} - f_i }{ f_i }$ (ecuación 39)	Calcular el cambio porcentual en la mejor solución para cada generación
<i>geometric average convergence rate</i>	$1 - \left(\left \frac{f_{opt} - f_l}{f_{opt} - f_0} \right \right)^{1/l}$	Medir la tasa de convergencia promedio geométrico hacia el óptimo, evitando influencia de grandes cambios y priorizando temprana convergencia.
<i>E-value</i>	$\eta_{convergence} = \frac{V}{\binom{n}{2}} \cdot \text{Algoritmo 22}$	Evaluar la eficiencia de búsqueda del algoritmo en base a las soluciones encontradas, el espacio de búsqueda total y las soluciones que sobrepasan un umbral de aptitud definido.
<i>Entropy Diversity</i>	Algoritmos 24 y 25	Medir la distribución de la población en el espacio de búsqueda
Tasa de éxito de operadores	$\frac{Op_{mejora}}{Op_{total}}$	Evaluar la efectividad de cada operador con la cantidad de mejoras que realiza sobre la solución y las veces que se ejecuta
Tasa de mejora y deterioro de operadores	Algoritmo 26	Cuantificar el impacto positivo y negativo de cada operador

Tabla 6: Métricas de rendimiento propuestas

CAPÍTULO 4

VALIDACIÓN DE LA SOLUCIÓN

En este capítulo se busca exponer los resultados obtenidos para las métricas presentadas y desarrolladas previamente. Para realizar el análisis completo sobre un algoritmo, se definen valores para los parámetros que rigen la ejecución del algoritmo y la recopilación de los datos, estos se muestran en la sección 4.1. En términos de la ejecución, se utiliza “*Vanilla*” *Genetic Algorithm* (vGA) (algoritmo 14) y, a manera de demostrar las capacidades de las métricas de desempeño para comparar algoritmos, se utiliza *Fast GA* (FGA) (algoritmo 15), modificado para incluir *uniform crossover* (algoritmo 8) sobre un set completo de los problemas descritos en la sección 3.2.2. La naturaleza de la ejecución en términos del algoritmo utilizado y los problemas a resolverse especifican en la sección 4.2.

En la sección 4.3 se exponen los resultados obtenidos para un subconjunto de problemas, el cual representa el ideal análisis a realizar a través de la librería de métricas. Los resultados se exponen por cada métrica y con los problemas en instancias que mejor representen el objetivo de la métrica y permitan realizar un correcto análisis.

4.1. Configuración del ambiente

En términos de parámetros para la ejecución, se definen dos parámetros estáticos, primero, la probabilidad de *crossover*, esta se establece como 0,5, con el objetivo de obtener una cantidad similar de aplicaciones de *crossover* como de mutaciones. Segundo, se establece un máximo de 7500 evaluaciones en total, con el objetivo de mantener la utilización de recursos estable a la hora de ejecutar problemas y recaudar información. Ahora, para obtener una gran variedad de resultados por sobre los problemas previamente descritos en 3.2.2, se utiliza un conjunto de tamaños para estos problemas. Este conjunto se define como $n = \{64, 100, 625, 5625\}$ para el caso de vGA, mientras que para FGA solo se evalúa el subconjunto $n = \{64, 625\}$. Para analizar el comportamiento del algoritmo en distintas situaciones, también se utilizan distintos tamaños para la población. Aquí, se analizan los siguientes valores: $\{\mu, \lambda\} = \{5, 15, 50\}$. Finalmente, se realizan, para cada algoritmo, 5 ejecuciones independientes para asegurar que los resultados sean consistentes.

4.2. Elección de algoritmo y problemas

Para poder realizar un análisis completo de la resolución de problemas, y a su vez, demostrar el potencial de las métricas desarrolladas, se debe seleccionar un algoritmo que no sesgue las soluciones que éste obtiene y el proceso a través del cual se obtienen.

Para esto, los algoritmos a utilizar son, primero, una implementación de “*Vanilla*” *Genetic Algorithm* (vGA) (algoritmo 14) con parámetros modificados, de tal forma que este utiliza *flip* (algoritmo 5) y *one point crossover* (algoritmo 6), con un modificador que permite al algoritmo ejecutar tan solo uno de estos dos operadores a la vez, de forma que se obtengan cantidades similares de ejecuciones entre operadores. En conjunto a estos dos operadores, se hace uso de una distribución normal para muestrear el valor de l , de forma que los valores varíen solo en medida de la cantidad de variables y no exista sesgo. Finalmente, se utiliza una selección elitista por sobre todas las soluciones disponibles (soluciones padre y descendientes) para apoyar a la explotación de mejores soluciones y asegurar un análisis sobre soluciones incrementalmente mejores en términos de aptitud. El segundo algoritmo es *Fast GA* (FGA), el cual fue modificado para incluir una etapa de *crossover*, haciendo uso de *uniform crossover*. Este segundo algoritmo, además, utiliza un parámetro F para calcular las variables a modificar en la etapa de mutación, por lo que se utilizan dos configuraciones para este valor, siendo el valor por defecto $F = 1,5$, se utiliza además $F = 2$.

En términos de los problemas, se realizan pruebas por sobre todos los problemas descritos, salvo aquellos que modifiquen la cantidad de variables y/o la cantidad de soluciones por generación. Esto se hace de manera que no afecten el análisis de aquellas métricas que dependan de tener una cantidad estática de variables por solución. Aquellos problemas que inducen estas modificaciones son variantes de *W-model transformations*, ya que estas modifican estructuralmente como funciona el algoritmo. En particular, se omiten las pruebas del problema *Leading ones* en sus variantes con variables ficticias, neutralidad y epistasis.

4.3. Resultados

En esta sección se exponen resultados obtenidos para cada métrica. La selección de problemas a exponer dependerá, en cada caso, de problemas que permitan evidenciar claramente la utilidad de la métrica y como esta varía en distintos casos. Finalmente, para cada métrica, se hace una comparación de ambas configuraciones de FGA ($F = 1,5$ y $F = 2$)

4.3.1. Aptitud en el tiempo

Para esta métrica se analiza el cambio de la aptitud para cada solución encontrada en el tiempo. Esto se implementa de dos formas, primero, graficando cada solución encontrada en cada evaluación. Y segundo, graficando la mejor solución encontrada hasta el momento de manera global.

Se analizan 2 problemas en particular para demostrar estas métricas, estos problemas son *OneMax* y *ConcatenatedTrap*, el primero por su simplicidad y fácil análisis. En el caso del segundo, se utiliza por la naturaleza de su espacio de búsqueda, donde la función de evaluación genera soluciones peores pasado cierto umbral. El uso de un algoritmo elitista dificulta

salir de este sector del espacio de búsqueda, cosa que se busca evidenciar.

En las figuras 32 y 33 se evidencia el caso donde se grafica cada valor evaluado (eje y) para cada solución encontrada (eje x) para *OneMax* y *ConcatenatedTrap*, respectivamente. Estas se obtienen de las ejecuciones para estos problemas con un tamaño de población igual a 50 y un tamaño de problema igual a 100.

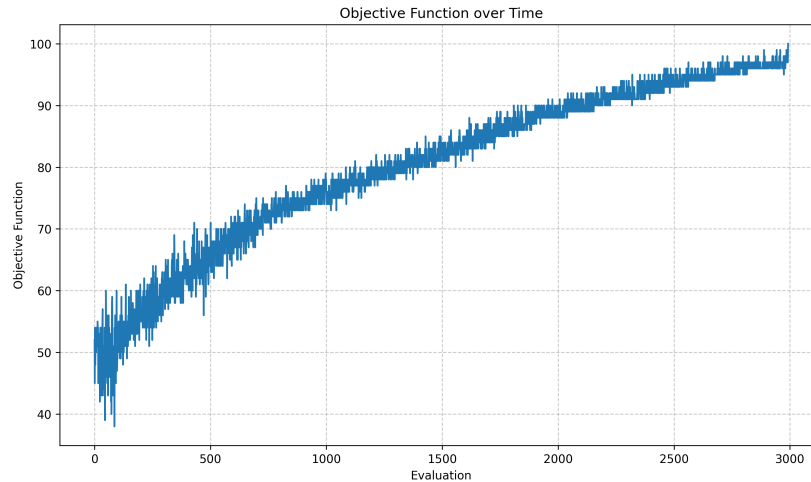


Figura 32: Cambio de aptitud por evaluación para *OneMax* con $n = 100$ y $\lambda = 50$. Elaboración propia

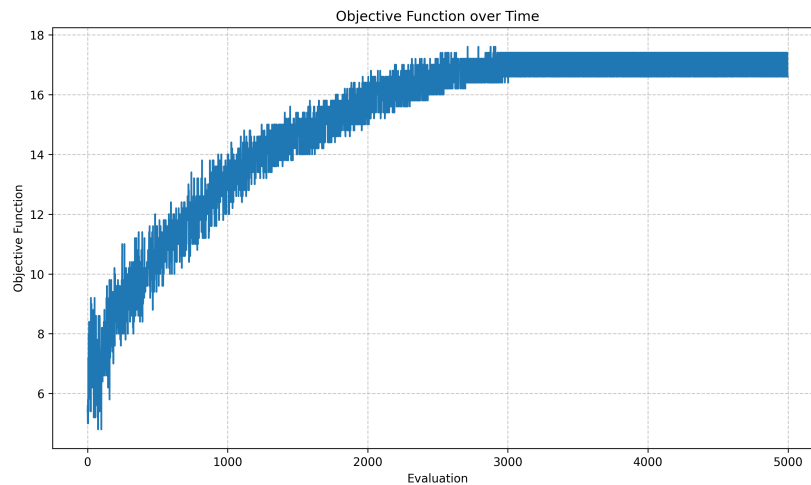


Figura 33: Cambio de aptitud por evaluación para *ConcatenatedTrap* con $n = 100$ y $\lambda = 50$. Elaboración propia

Se puede apreciar el comportamiento comúnmente encontrado con *OneMax*, donde el valor evaluado constantemente aumenta, mientras que *ConcatenatedTrap* se estanca alrededor de la evaluación número 3000. El mismo comportamiento se puede apreciar en las figuras 34 y 35, donde se utiliza la mejor solución encontrada hasta la generación evaluada.

Ahora, para comparar ambos casos de FGA se utiliza el caso del problema de N-reinas, en

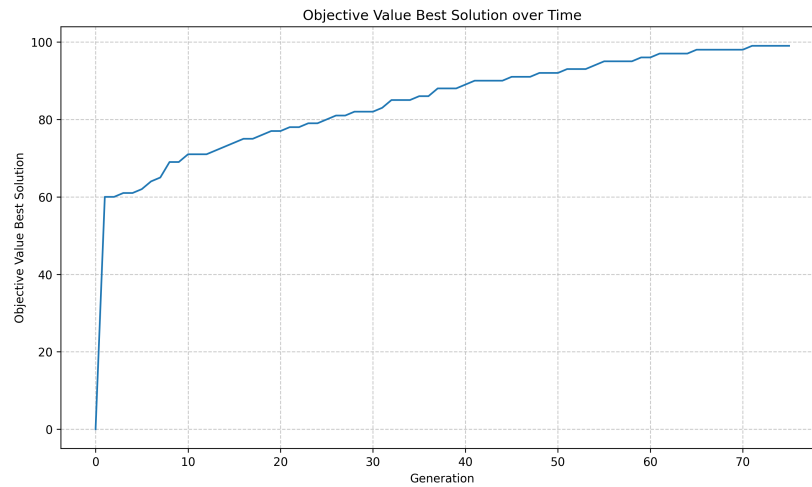


Figura 34: Cambio de aptitud con mejor solución encontrada por generación para *OneMax* con $n = 100$ y $\lambda = 50$. Elaboración propia

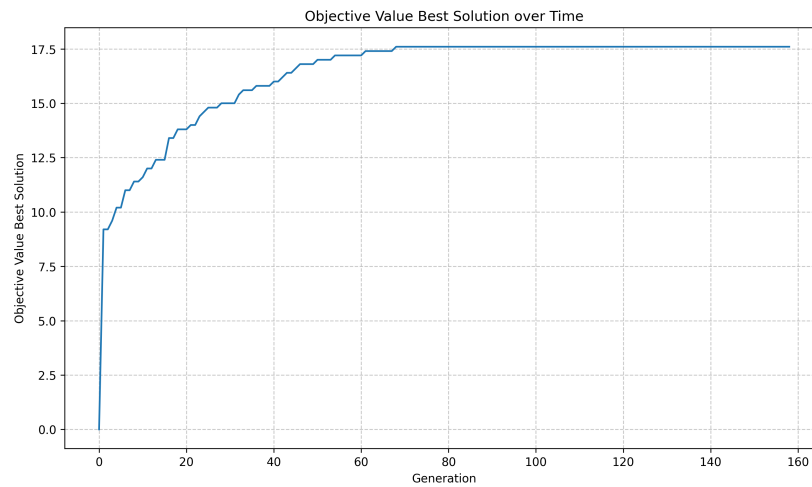


Figura 35: Cambio de aptitud con mejor solución encontrada por generación para *ConcatenatedTrap* con $n = 100$ y $\lambda = 50$. Elaboración propia

su tamaño de 64 variables y con tamaño de población $\lambda = 5$. Los gráficos a comparar se encuentran en las figuras 36 y 37, las cuales mantienen las estructuras antes vistas.

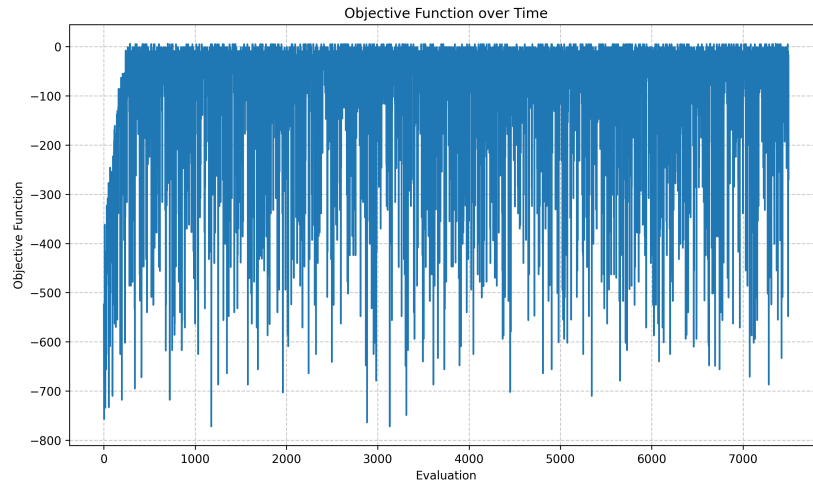


Figura 36: Cambio de aptitud de FGA $F = 1,5$ por evaluación para N-reinas con $n = 64$ y $\lambda = 5$. Elaboración propia

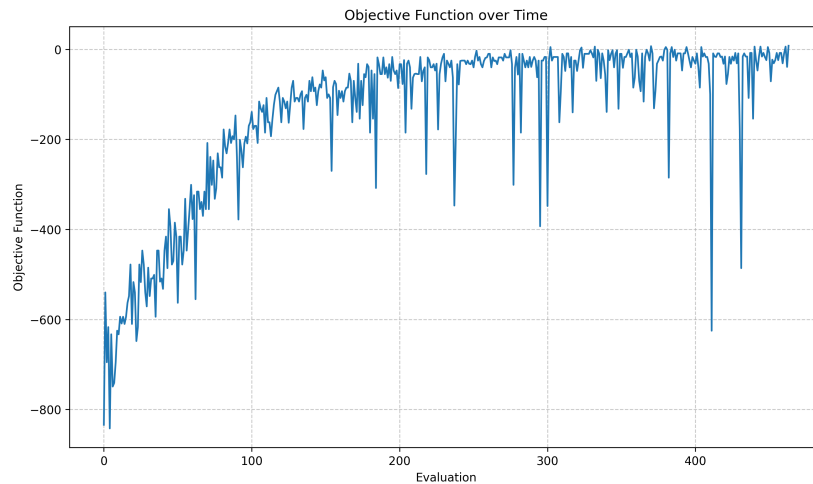


Figura 37: Cambio de aptitud de FGA $F = 2$ por evaluación para N-reinas con $n = 64$ y $\lambda = 5$. Elaboración propia

Si bien es un caso con una diferencia muy grande, el cual puede parecer exagerado, este demuestra que un cambio de parámetro puede ser fatal para este algoritmo, de hecho, el comportamiento observado se ve en cada uno de los tamaños de población ejecutados (encontrados en los anexos de este escrito). En el primer caso, se puede apreciar que el número de evaluaciones llega a su máximo de 7500 sin encontrar un óptimo, mientras que en el segundo caso se encuentra alrededor de las 475 evaluaciones. Esto evidencia un fuerte estancamiento para el primer caso, lo cual es reforzado con la información entregada por las figuras 38 y 39, donde se ve que la mejor solución encontrada por el algoritmo se estanca muy cerca del óptimo alrededor de la generación número 100, más nunca alcanzándolo.

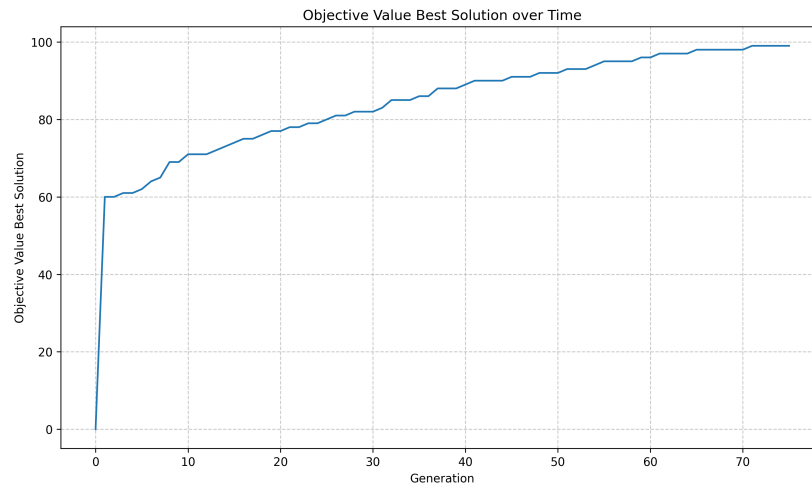


Figura 38: Cambio de aptitud de FGA $F = 1,5$ por mejor solución encontrada por generación para N-reinas con $n = 64$ y $\lambda = 5$. Elaboración propia

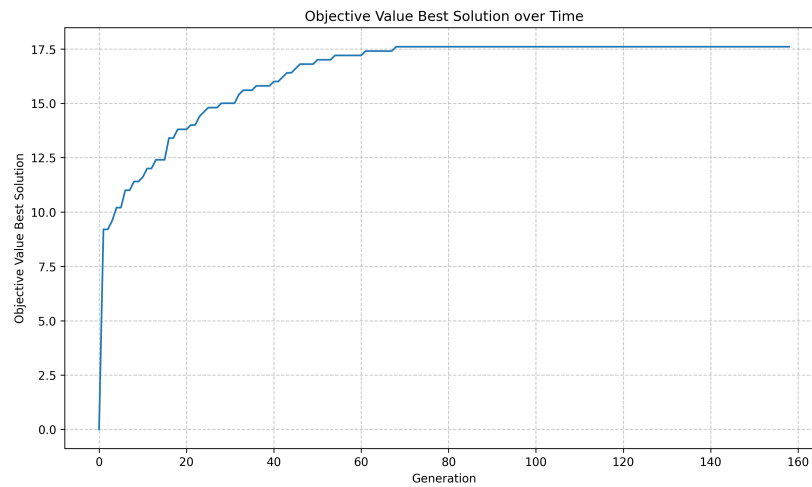


Figura 39: Cambio de aptitud de FGA $F = 2$ por mejor solución encontrada por generación para N-reinas con $n = 64$ y $\lambda = 5$. Elaboración propia

4.3.2. Rate of fitness change

En este caso se busca analizar la tasa de cambio en la aptitud respecto de un óptimo conocido, a través de este análisis se puede concluir sobre si la búsqueda de soluciones se estanca pasadas las generaciones. Para facilitar este análisis, se grafica esta métrica desde dos enfoques, primero, midiendo la diferencia de aptitud, o error relativo, entre las mejores soluciones encontradas para cada generación. Y segundo, graficando la tasa de cambio para cada solución encontrada por individuo, esto respecto de un óptimo conocido.

En este caso, se hace análisis del problema de las N-reinas con un tamaño de problema $n = 625$. La primera implementación se muestra en la figura 40, donde el algoritmo se ejecuta con un tamaño de población igual a 5. Esta configuración produce una ejecución más extensa y permite un análisis más claro.

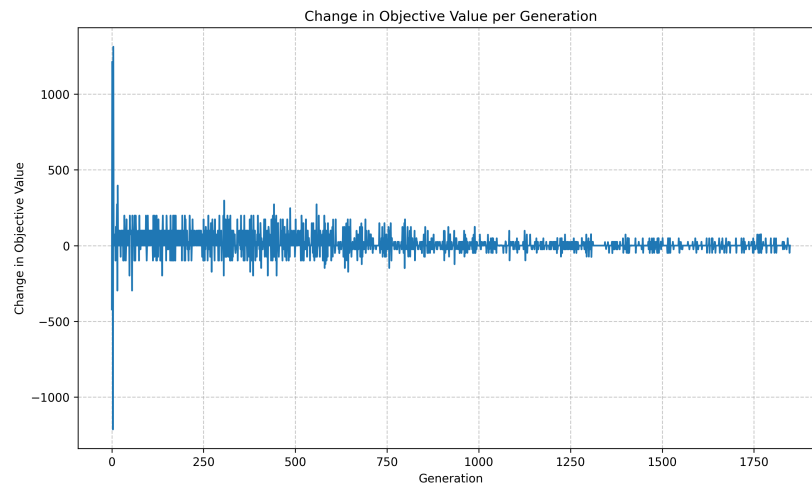


Figura 40: Tasa de cambio de aptitud con mejor solución encontrada por generación para *N-queens* con $n = 625$ y $\lambda = 5$. Elaboración propia

Aquí, se puede apreciar un comportamiento donde los incrementos de aptitud son constantemente menores a medida que se acerca al óptimo. Esto es un comportamiento esperado dado que, al comienzo de la búsqueda, es más probable que el algoritmo encuentre soluciones candidatas que resuelvan restricciones que lo acerquen al óptimo. A medida que avanza la ejecución, los movimientos ejecutados por los operadores, (como *flip*, que solo manejan un vecindario inmediato para la selección de soluciones) pueden llevar a óptimos locales o a soluciones que empeoren fuertemente la aptitud. Esta situación se acrecienta en un problema como *N-queens*, donde a medida que se resuelven restricciones, las variables se hacen interdependientes y un movimiento puede romper muchas de las restricciones. Particularmente, alrededor de la generación 1300 se reduce la tasa de cambio y se aprecian más generaciones donde no hay cambio aparente, indicando un leve estancamiento el cual se debe estudiar para mejorar la ejecución en las últimas etapas de búsqueda.

Luego, para el caso por individuo, se hace a través de 3 distintos tamaños de población, 5, 15 y

50, con el objetivo de demostrar la granularidad de análisis que permite esta representación. Estos gráficos se presentan en las figuras 41, 42 y 43, donde el eje x representa la generación donde se evalúa el valor de la tasa de cambio, el cual es encontrado en el eje y .

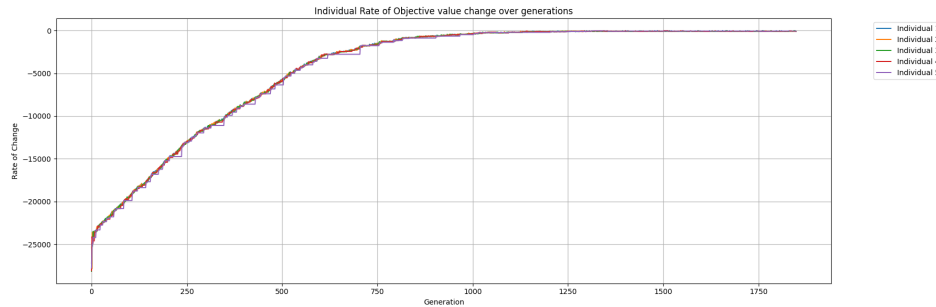


Figura 41: Tasa de cambio de aptitud respecto del óptimo por individuo para N -queens con $n = 625$ y $\lambda = 5$. Elaboración propia

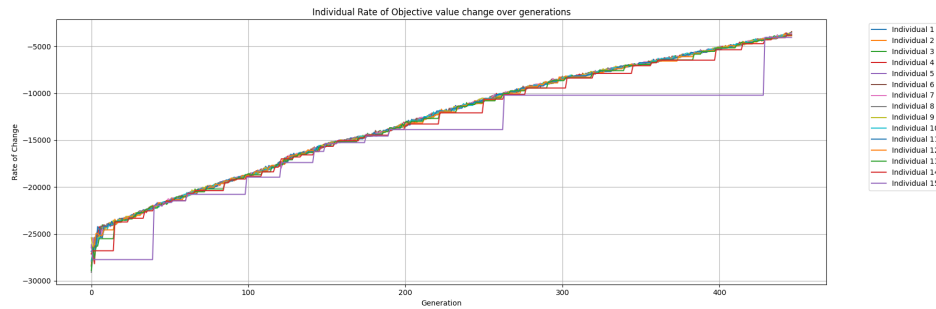


Figura 42: Tasa de cambio de aptitud respecto del óptimo por individuo para N -queens con $n = 625$ y $\lambda = 15$. Elaboración propia

Tal como se adelantaba, la granularidad entregada por esta métrica permite visualizar como la población se comporta a través de las generaciones. Este comportamiento se hace evidente una vez que comparamos los casos de tamaño de población 5 y 50, donde en el primero se observa una constante mejora de forma homogénea, es decir, todos los individuos mantienen la mejora de su aptitud de solución. En el caso donde la población es de tamaño 50, es fácil observar que alrededor de 6 individuos sufren de estancamientos por múltiples generaciones, indicando que idealmente se debe efectuar una reducción en el número de individuos por generación para este problema.

Ahora, para el caso comparativo de FGA, se utiliza el problema *Ising Triangular*, en su tamaño $n = 625$, en particular, el análisis se hace a partir de los tamaños de población $\lambda = 15$, aunque el comportamiento del algoritmo se ve replicado para el resto de tamaños.

En el primer caso, con $F = 1,5$, presentado en la figura 44 se puede apreciar que la tasa de cambio no mantiene una pendiente constante, ya que se observa una constante baja en la aptitud y su diferencia con el óptimo. En este caso, se puede entender que para mejorar la aptitud de la solución, el algoritmo debe estar en constante exploración del espacio de búsqueda en sectores de baja aptitud.

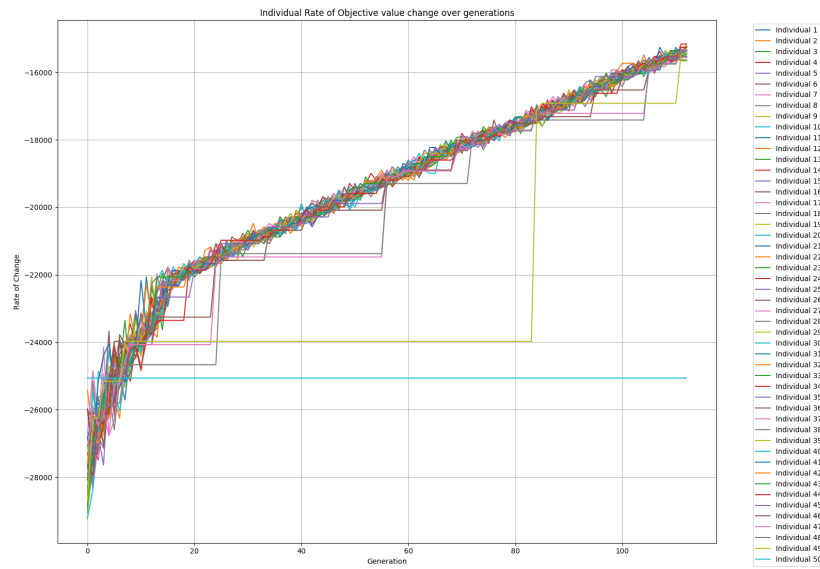


Figura 43: Tasa de cambio de aptitud respecto del óptimo por individuo para *N-queens* con $n = 625$ y $\lambda = 50$. Elaboración propia

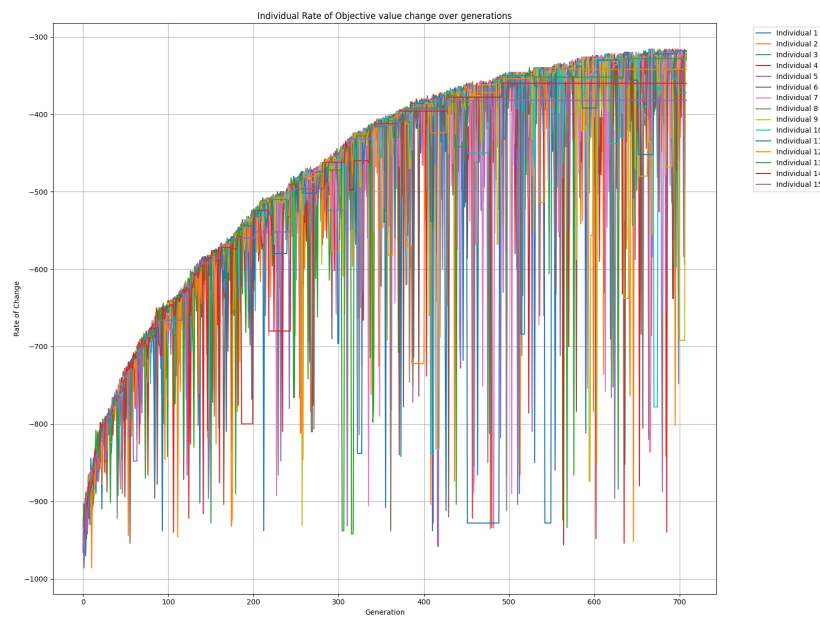


Figura 44: Tasa de cambio de aptitud respecto del óptimo por individuo para *Ising Triangular* con FGA $F = 1,5$, $n = 625$ y $\lambda = 15$. Elaboración propia

Mientras que, en el segundo caso, con $F = 2$, presentado en la figura 45, las curvas para cada individuo no se ven afectadas por este fenómeno, sino que aumentan de forma constante, indicando que la intensificación del algoritmo es eficiente. Sin embargo, a pesar de esto, es el primer algoritmo el que obtiene una mejor solución al finalizar la ejecución, notable por su diferencia menor con el óptimo.

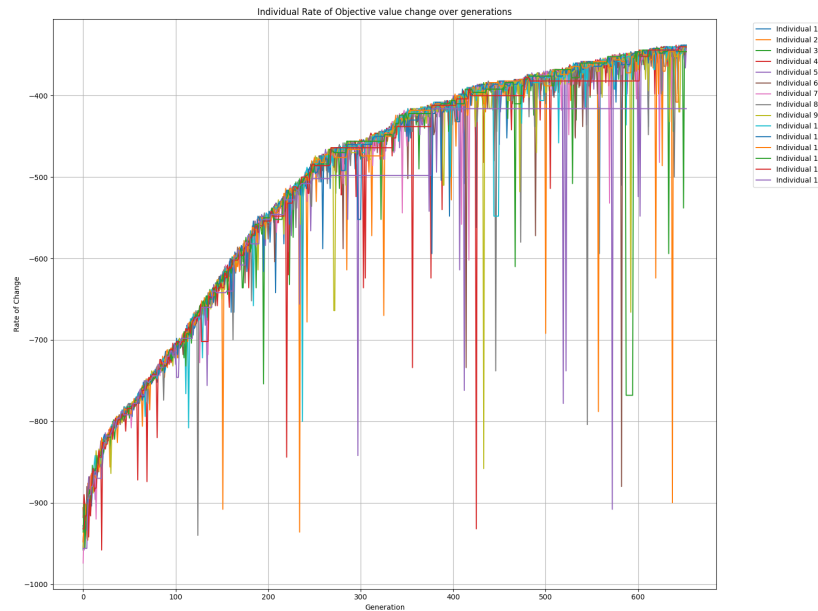


Figura 45: Tasa de cambio de aptitud respecto del óptimo por individuo para *Ising Triangular* con con FGA $F = 2$, $n = 625$ y $\lambda = 15$. Elaboración propia

4.3.3. Convergencia

En términos de convergencia, se desarrollan 5 implementaciones. Estas serán analizadas utilizando los distintos tamaños de problema y tamaños de población para el problema *OneMax* dado que este presenta un espacio de búsqueda el cual aumenta su sensibilidad a medida que se acerca al óptimo, es decir, las soluciones generadas difieren mucho en su valor evaluado a medida que se acerca al óptimo. Esto permite al algoritmo no quedarse estancado en soluciones de baja aptitud, ya que estas serán descartadas rápidamente. De esta forma, el algoritmo es capaz de generar visualizaciones de un caso óptimo.

Además, para un caso con un espacio de búsqueda complejo, donde la convergencia no es ideal, se analizara *OneMax* con *W-model* en perturbación de aptitud (desde ahora *OneMax + W*). En particular, la transformación escogida modifica la función de evaluación de tal forma que, en la práctica, los óptimos locales están a distancias de 5 en términos del valor de aptitud evaluado. En la práctica, este cambio implica que el espacio de búsqueda es sumamente rugoso, con cambios que no son fácilmente distinguibles en los valores evaluados, sobre

todo en tamaños de problema más altos. Provocando que un algoritmo elitista no pueda recuperarse y encontrar un óptimo fácilmente.

Primero, se analiza la convergencia (con uso de óptimo) desde dos puntos de vista. El primero con uso de los mejores valores de aptitud por generación. Esta implementación utiliza el mejor valor de la generación actual y la generación previa, con el objetivo de normalizar los valores y permitiendo analizar la convergencia inmediata. El segundo caso hace omisión de la mejor solución previa, acercando el concepto al del error relativo, métrica más comúnmente usada en la literatura y que permite una vista más global de la convergencia hacia el óptimo.

OneMax presenta una curva de convergencia la cual, por lo general, aumenta su varianza con el pasar de las generaciones, lo que se condice con su espacio de búsqueda sensible. Esto se evidencia en las figuras 46, 47 y 48, las cuales muestran la convergencia con óptimo propuesta para los casos con tamaño de problema $n = 625$ y tamaños de población $\lambda = \{5, 15, 50\}$.

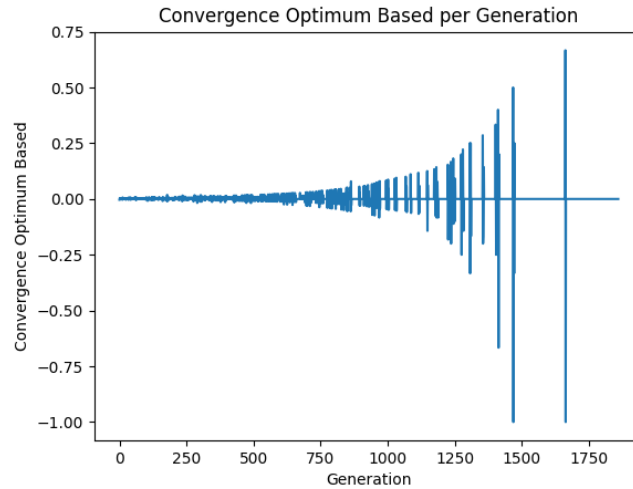


Figura 46: Convergencia basada en óptimo para *OneMax* con $n = 625$ y $\lambda = 5$. Elaboración propia

En el caso de tamaño de población $\lambda = 50$ se puede apreciar una tasa de convergencia mayor que en el caso de $\lambda = 15$ para las primeras 100 generaciones, esto indica que la mayor exploración que provee una población más numerosa es beneficiosa en este algoritmo. Aunque mantener una población de gran tamaño impacta en el costo de ejecutar el algoritmo, evidenciado por el hecho de que este tan solo logra ejecutarse por alrededor de 115 generaciones antes de llegar a su límite de 7500 evaluaciones.

Este caso es importante analizarlo también para el propuesto *OneMax + W*, ya que en cada uno de los casos estudiados, nuevamente con la excepción del caso con $\lambda = 50$, el algoritmo rápidamente se estanca, como se evidencia en las figuras 49 y 50.

Es fácil ver que el algoritmo no es capaz de realizar avances sobre la aptitud de las soluciones

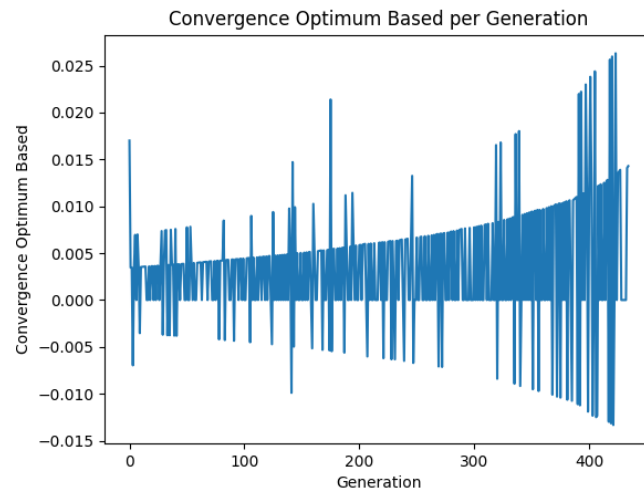


Figura 47: Convergencia basada en óptimo para *OneMax* con $n = 625$ y $\lambda = 15$. Elaboración propia

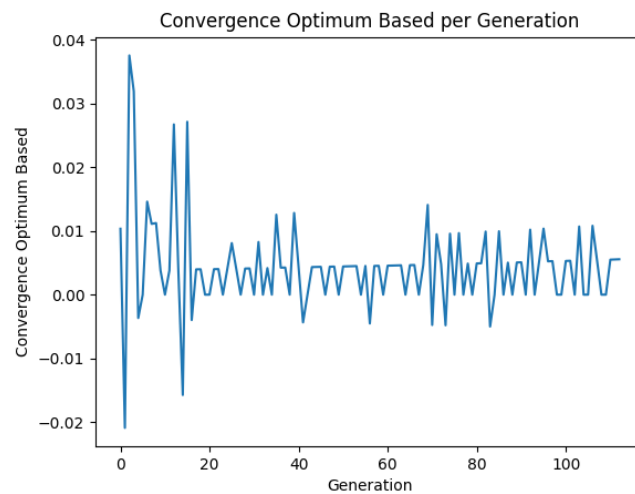


Figura 48: Convergencia basada en optimo para *OneMax* con $n = 625$ y $\lambda = 50$. Elaboración propia

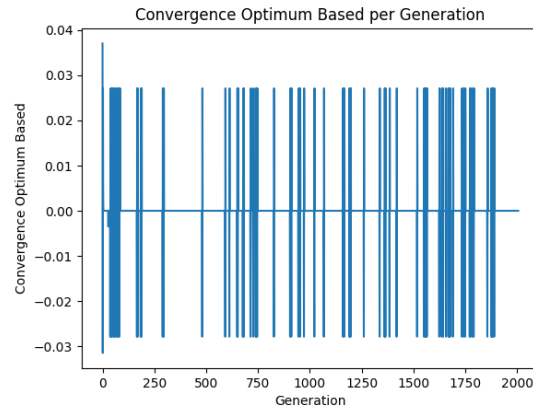


Figura 49: Convergencia basada en óptimo para *OneMax + W* con $n = 625$ y $\lambda = 5$. Elaboración propia

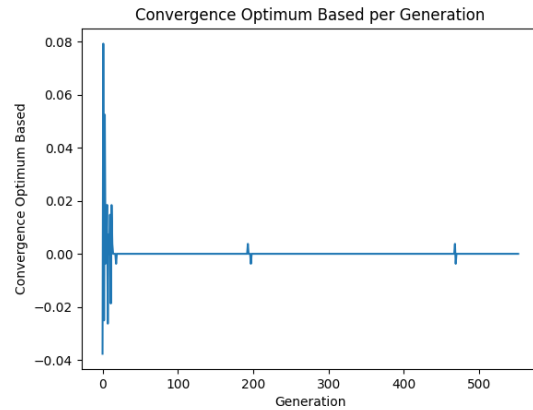


Figura 50: Convergencia basada en óptimo para *OneMax + W* con $n = 625$ y $\lambda = 15$. Elaboración propia

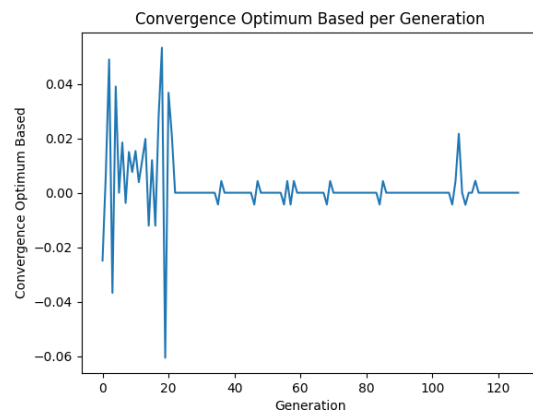


Figura 51: Convergencia basada en óptimo para *OneMax + W* con $n = 625$ y $\lambda = 50$. Elaboración propia

que encuentra, muy a diferencia del caso con $\lambda = 50$ expuesto en la figura 51, lo que sugiere que mantener una población de tamaño más cercano al de la cantidad de variables es útil, ya que explora más soluciones y evita el extremadamente rápido estancamiento.

Ahora, este análisis se puede fácilmente complementar con la versión de esta métrica, la cual utiliza el error relativo respecto del óptimo. Utilizando esta vista general sobre el avance del algoritmo hacia una solución óptima, podemos ver más fácilmente el fenómeno encontrado en la ejecución con $\lambda = 50$. Las figuras que corresponden a esta métrica se encuentran en 52, 53 y 54, para cada tamaño de población respectivamente.

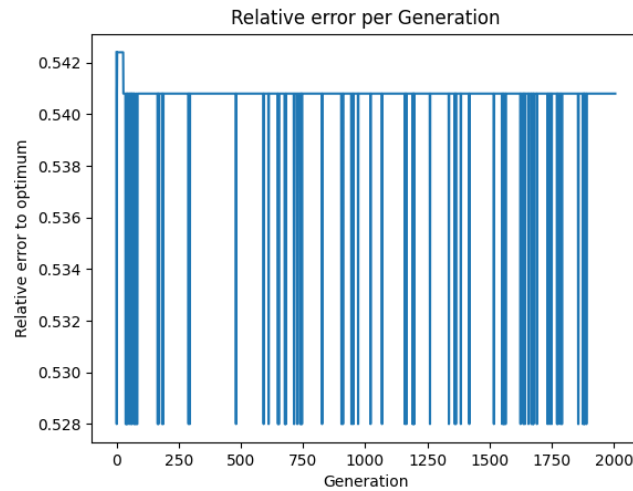


Figura 52: Error relativo basado en óptimo para *OneMax + W* con $n = 625$ y $\lambda = 5$. Elaboración propia

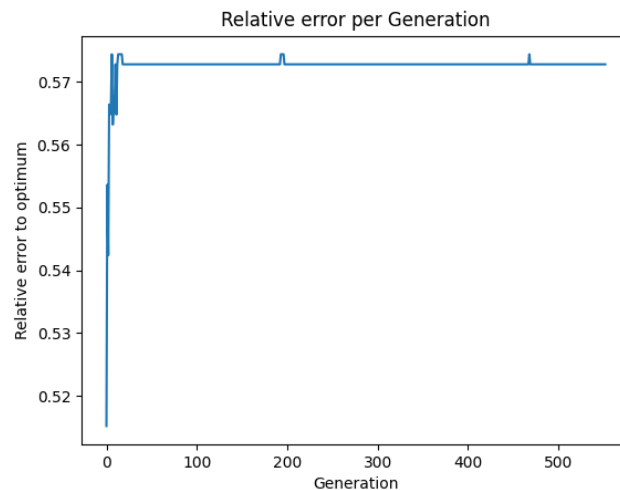


Figura 53: Error relativo basado en óptimo para *OneMax + W* con $n = 625$ y $\lambda = 15$. Elaboración propia

Se puede observar que el error relativo respecto del óptimo se mantiene con un máximo constante, es decir, el algoritmo no es capaz de acercarse al óptimo en ninguno de los dos casos. Aunque incluso si en ambos casos, se mantiene el valor del mejor error relativo constante, en el caso con $\lambda = 15$ se observa que este se mantiene totalmente constante, indicando una mejora por sobre el caso donde $\lambda = 5$, donde este constantemente empeora y luego vuelve al valor previamente encontrado.

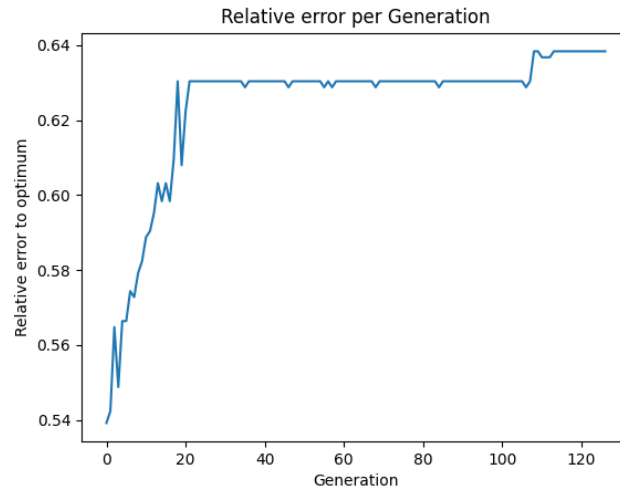
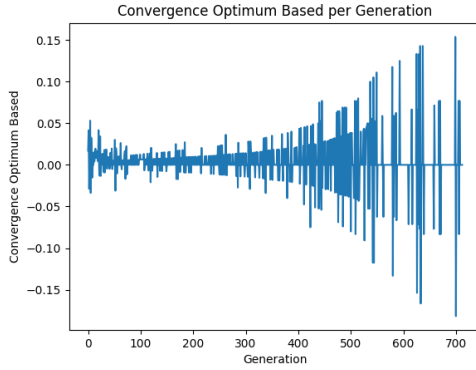


Figura 54: Error relativo basado en óptimo para *OneMax* + *W* con $n = 625$ y $\lambda = 50$. Elaboración propia

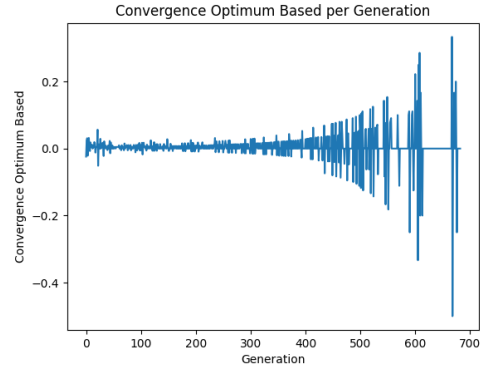
En el caso donde el valor de λ es mayor, se puede observar que, si bien el valor del error relativo parece estar constante a través de una gran cantidad de generaciones, de todas formas mejora hacia las últimas generaciones observadas. Esto sustenta el análisis realizado sobre que, para casos con alta rugosidad, se hace imperativa la exploración en gran escala del espacio de búsqueda.

Para ambas variantes de FGA se puede observar que la convergencia basada en óptimo presenta un comportamiento muy similar en cada uno de los problemas. Para ejemplificar este comportamiento se presentan los casos de *OneMax* y *Ising Torus*, ambos en sus tamaños $n = 625$, aunque con tamaños de población $\lambda = 15$ y $\lambda = 5$, respectivamente, esto con el objetivo de demostrar los factores de los cuales depende la convergencia de un algoritmo.

Los gráficos que representan los casos anteriormente descritos se agrupan por métrica y por problema, es decir, se tiene para la convergencia con óptimo, las figuras 55a y 55b para *OneMax* con cada versión del algoritmo, y 56a y 56b para *Ising Torus* con cada versión del algoritmo.

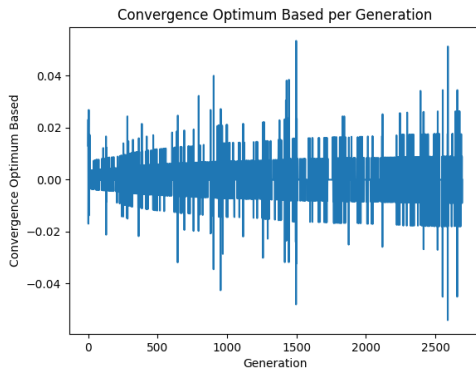


(a) Convergencia basada en óptimo para *OneMax* con $n = 625$ y $\lambda = 15$. Caso FGA $F = 1,5$. Elaboración propia

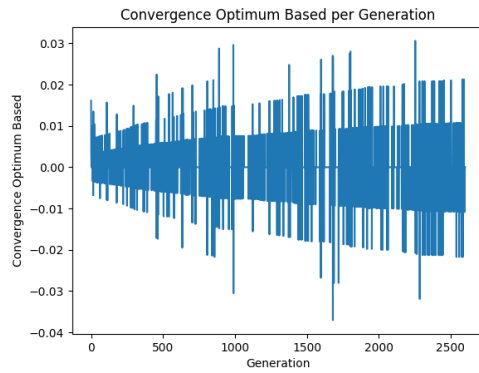


(b) Convergencia basada en óptimo para *OneMax* con $n = 625$ y $\lambda = 15$. Caso FGA $F = 2$. Elaboración propia

Figura 55: Comparativa convergencia basada en óptimo para FGA en *OneMax* con $n = 625$ y $\lambda = 15$.



(a) Convergencia basada en óptimo para *Ising Torus* con $n = 625$ y $\lambda = 5$. Caso FGA $F = 1,5$. Elaboración propia

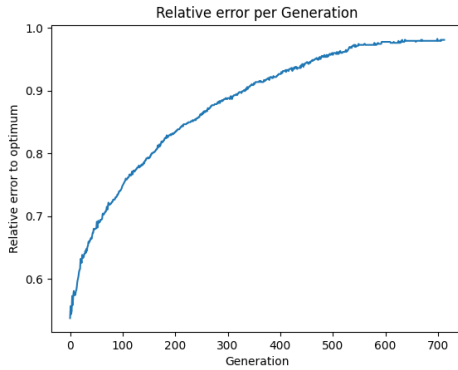


(b) Convergencia basada en óptimo para *Ising Torus* con $n = 625$ y $\lambda = 5$. Caso FGA $F = 2$. Elaboración propia

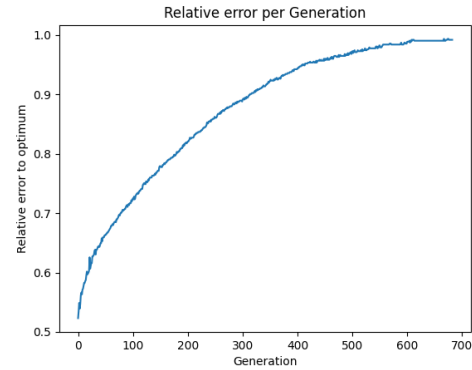
Figura 56: Comparativa convergencia basada en óptimo para FGA en *Ising Torus* con $n = 625$ y $\lambda = 5$.

El error relativo prueba nuevamente que se demuestra un comportamiento sumamente similar para ambos problemas y ambos algoritmos, los graficos correspondientes se visualizan en las figuras 57a y 57b para *OneMax* con $F = 1,5$ y $F = 2$, respectivamente. Estructura que se repite para *Ising Torus*, con las figuras 58a y 58b.

La fuerte similitud entre estos graficos, incluso en el hecho de que para ninguno de los casos se llega al óptimo, demuestra que la convergencia es fuertemente dependiente del algoritmo utilizado, sus movimientos y como este maneja las soluciones. Mas alla de sus parametros, problema escogido (notando la diferencia en dificultad de ambos problemas) y parametros

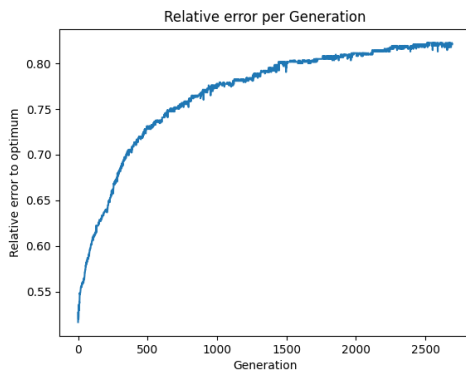


(a) Error relativo basado en óptimo para *OneMax* con $n = 625$ y $\lambda = 15$. Caso FGA $F = 1,5$. Elaboración propia

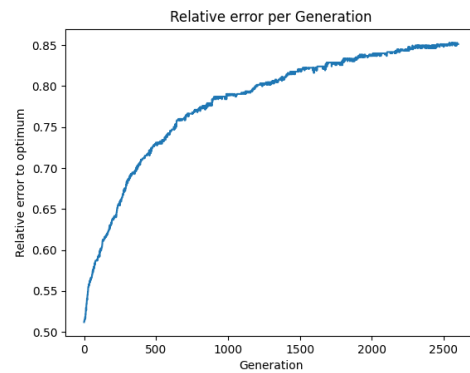


(b) Error relativo basado en óptimo para *OneMax* con $n = 625$ y $\lambda = 15$. Caso FGA $F = 2$. Elaboración propia

Figura 57: Comparativo error relativo basado en óptimo para FGA en *OneMax* con $n = 625$ y $\lambda = 15$.



(a) Error relativo basado en óptimo para *Ising Torus* con $n = 625$ y $\lambda = 5$. Caso FGA $F = 1,5$. Elaboración propia



(b) Error relativo basado en óptimo para *Ising Torus* con $n = 625$ y $\lambda = 5$. Caso FGA $F = 2$. Elaboración propia

Figura 58: Comparativo error relativo basado en óptimo para FGA en *Ising torus* con $n = 625$ y $\lambda = 5$.

de ambiente.

Luego, en caso de no conocerse un óptimo para el problema, se puede realizar el procedimiento descrito en el algoritmo 20, donde se analiza la convergencia inmediata entre las mejores soluciones entre cada generación para cada individuo, analizando granularmente el avance de cada individuo de la población. Además, se calcula el error relativo a la solución actual utilizando la ecuación 39.

La primera de estas métricas se muestra para el caso de *OneMax* en su tamaño de problema

$n = 64$ para los tamaños de población $\lambda = \{5, 15, 50\}$.

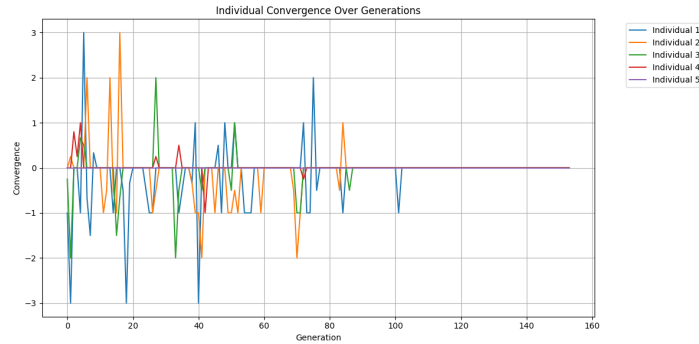


Figura 59: Convergencia sin óptimo por individuo para *OneMax* con $n = 64$ y $\lambda = 5$. Elaboración propia

La figura 59, correspondiente a la ejecución con menor tamaño de población, muestra estancamiento hacia el final de la ejecución, a pesar de que esta encuentra el óptimo, lo hace de forma súbita luego de un periodo de casi 50 generaciones estancado.

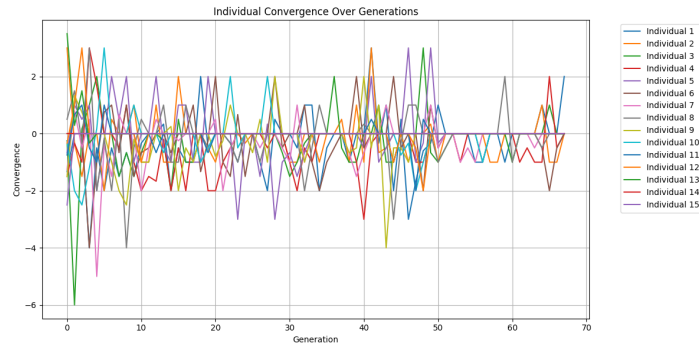


Figura 60: Convergencia sin óptimo por individuo *OneMax* con $n = 64$ y $\lambda = 15$. Elaboración propia

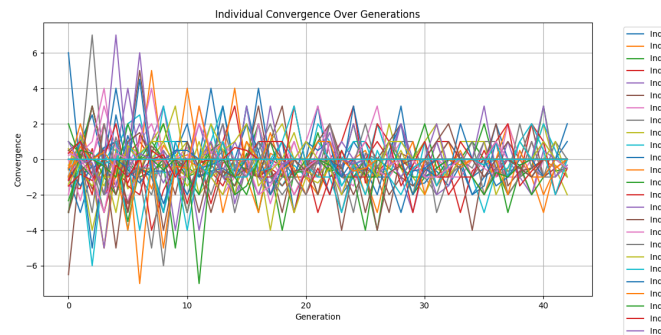


Figura 61: Convergencia sin óptimo por individuo *OneMax* con $n = 64$ y $\lambda = 50$. Elaboración propia

Mientras, las figuras 60 y 61, correspondientes a los tamaños 15 y 50, muestran una alta variación en los valores evaluados para cada individuo, indicando una buena explotación y exploración del espacio de búsqueda.

Ahora, este análisis se hace más útil en conjunto con la segunda métrica propuesta para el caso sin óptimo. Esta métrica, el error relativo a la solución actual, exhibe un comportamiento muy similar a la convergencia con uso de óptimo, por lo que se obtiene un conjunto de métricas que, aunque no exactamente idénticas, permite realizar un análisis muy similar al previamente visto con uso del óptimo.

Los gráficos generados por esta métrica para *OneMax* son presentados en 62, 63 y 64

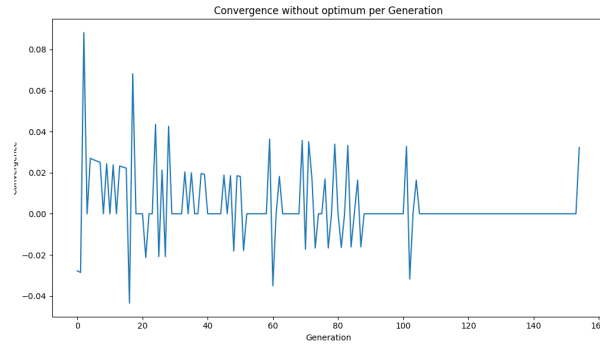


Figura 62: Error relativo sin óptimo para *OneMax* con $n = 64$ y $\lambda = 5$. Elaboración propia

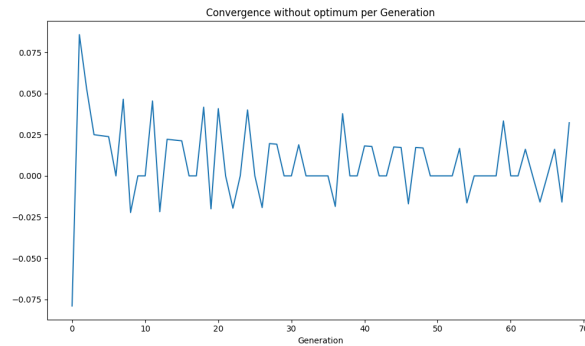


Figura 63: Error relativo sin óptimo para *OneMax* con $n = 64$ y $\lambda = 15$. Elaboración propia

Incluso, es posible afirmar que en casos donde la convergencia con óptimo presenta patrones, estos mismos patrones se ven reflejados en el caso de error relativo sin óptimo. Este fenómeno es observable en el caso del problema *Ising Triangle*, donde la convergencia con óptimo, evidencia una mejora constante. De igual forma, el error relativo sin óptimo permite observar el mismo patrón, solo que con menor amplitud debido a la constante comparación con iteraciones previas. Las figuras correspondientes a *Ising Triangle* con $n = 625$ y población $\lambda = 5$ se encuentran en 65 y 66, para convergencia con óptimo y sin óptimo, respectivamente.

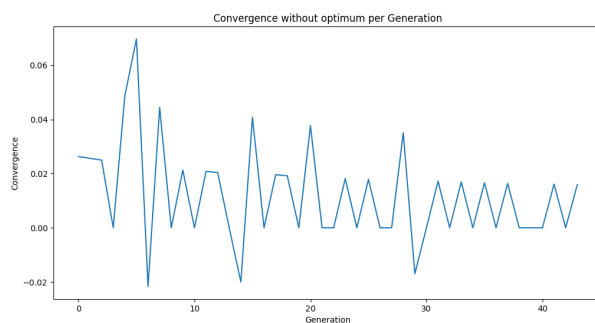


Figura 64: Error relativo sin óptimo para *OneMax* con $n = 64$ y $\lambda = 50$. Elaboración propia

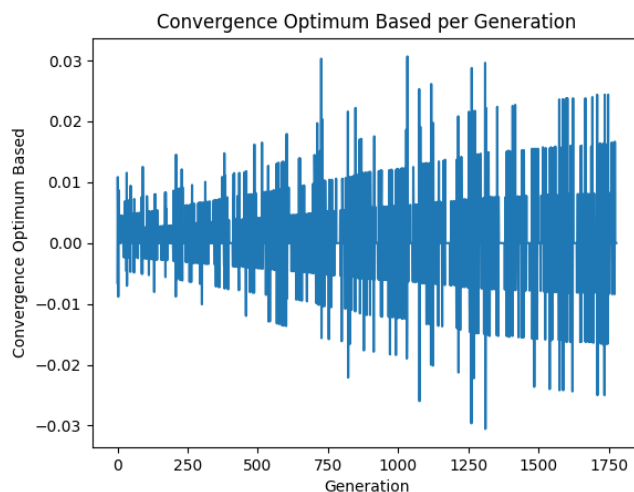


Figura 65: Convergencia con uso de óptimo para *Ising Triangle* con $n = 625$ y $\lambda = 5$. Elaboración propia

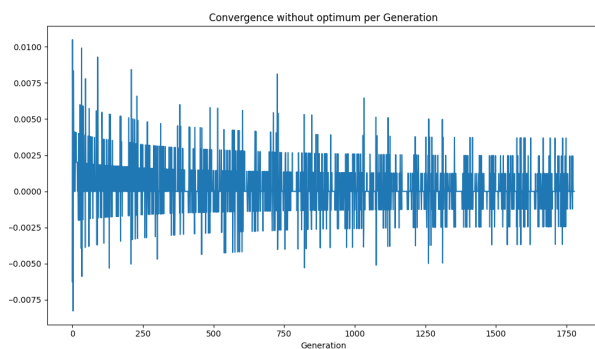
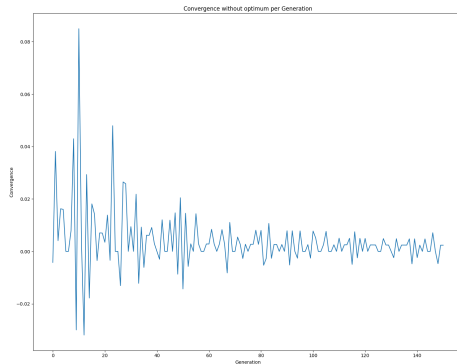
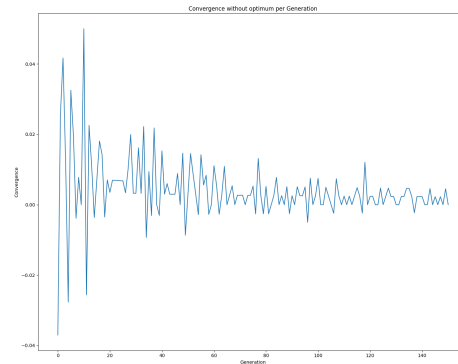


Figura 66: Convergencia sin óptimo para *Ising Triangle* con $n = 625$ y $\lambda = 5$. Elaboración propia

Para estas métricas, se hace un estudio sobre el caso de *concatenated trap* en su tamaño $n = 625$ para ambos casos de FGA, donde se utiliza, además, un tamaño de población $\lambda = 50$. Utilizando la convergencia sin óptimo es posible observar la tendencia con la que mejoran las soluciones respecto de sus iteraciones pasadas, y así tener una mirada sobre el rendimiento del algoritmo en su intensificación hacia mejores soluciones. Para el caso propuesto, es posible observar en las figuras 67a y 67b el resultado de la convergencia sin óptimo para ambos valores de F .



(a) Convergencia sin óptimo para *concatenated trap* con $n = 625$ y $\lambda = 50$. Caso FGA $F = 1,5$. Elaboración propia



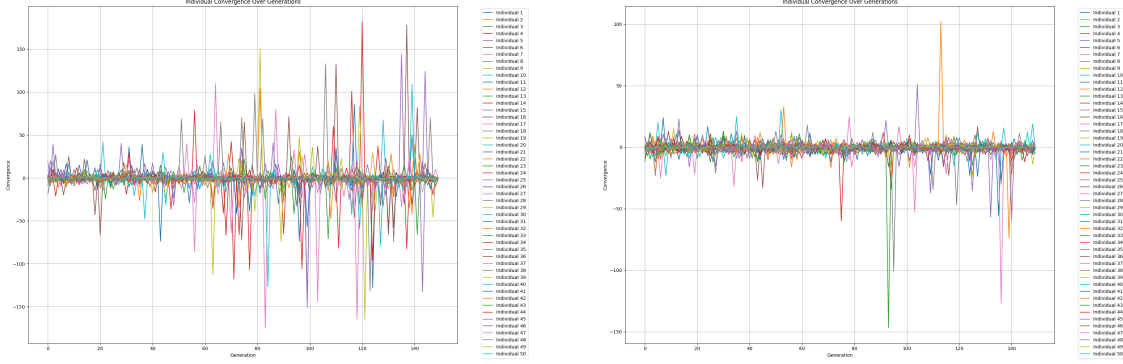
(b) Convergencia sin óptimo para *concatenated trap* con $n = 625$ y $\lambda = 50$. Caso FGA $F = 2$. Elaboración propia

Figura 67: Comparativa convergencia sin óptimo para *concatenated trap* con $n = 625$ y $\lambda = 50$.

De estos gráficos es posible desprender que el rendimiento observado es similar entre ambos gráficos, al menos en el estudio generación a generación. Aquí, no se observan valores sumamente distantes entre generaciones en ningún momento. Sin embargo, es de notar que en el caso de $F = 2$ se puede apreciar que los valores, aunque bajos, están más consistentemente por sobre el 0, indicando leves mejoras entre generaciones de forma constante.

El análisis previo se puede complementar a través del análisis por individuo, el cual se realiza a partir de las figuras 68a y 68b. En estas figuras se puede apreciar una fuerte diferencia en el comportamiento del algoritmo. Es fácil ver que, al mantener un F menor, se aprecian distancias altas entre las soluciones, lo cual se puede deber a una prevalencia mayor del uso de *crossover* en comparación a un caso con el parámetro F (*mutation strength*) más alto, donde al dar prioridad a la intensificación de las soluciones, las variaciones entre generaciones son menores y las mejoras en la solución son más bajas pero constantes.

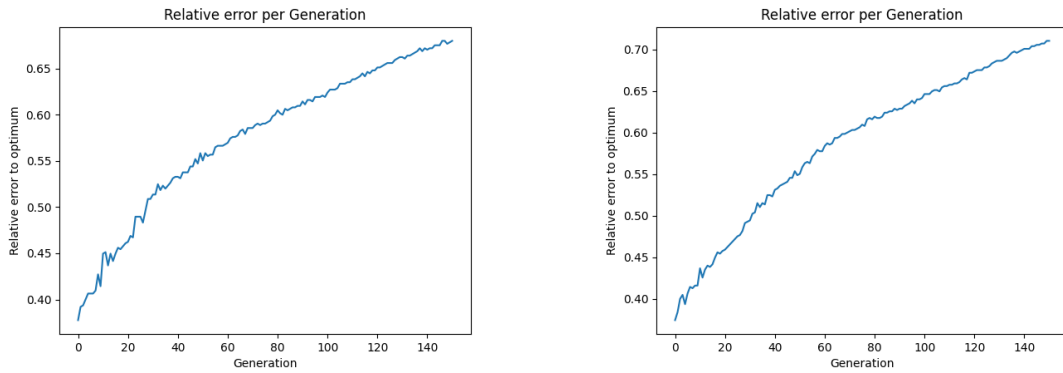
Todo esto se puede confirmar si se observa el error relativo con óptimo encontrado en las figuras 69a y 69b, donde se puede observar que el caso con mayor *mutation strength* logra una solución más cercana al óptimo por alrededor de un 3 %.



(a) Convergencia sin óptimo por individuo para *concatenated trap* con $n = 625$ y $\lambda = 50$. Caso FGA $F = 1,5$. Elaboración propia

(b) Convergencia sin óptimo por individuo para *concatenated trap* con $n = 625$ y $\lambda = 50$. Caso FGA $F = 2$. Elaboración propia

Figura 68: Comparativa convergencia sin óptimo por individuo para *concatenated trap* con $n = 625$ y $\lambda = 50$.



(a) Error relativo con óptimo para *concatenated trap* con $n = 625$ y $\lambda = 50$. Caso FGA $F = 1,5$. Elaboración propia

(b) Error relativo con óptimo para *concatenated trap* con $n = 625$ y $\lambda = 50$. Caso FGA $F = 2$. Elaboración propia

Figura 69: Comparativo error relativo con óptimo para *concatenated trap* con $n = 625$ y $\lambda = 50$.

La última métrica en términos de convergencia es *geometric rate of fitness change*. Esta métrica permite analizar cómo el algoritmo se comporta en relación a la cantidad de generaciones que han pasado. Es decir, esta métrica muestra en mayor medida la mejora de la solución de forma temprana y normaliza el cambio que ocurra en generaciones muy lejanas al inicio de la ejecución.

Para el análisis de esta métrica se utiliza el problema de N-reinas con tamaño de problema 64 y tamaño de población 5, esto debido a que este problema presenta una rápida convergencia al inicio de su ejecución. Además, se utiliza el mismo problema, pero con tamaño de

problema 625 y población 5 a modo de comparación.

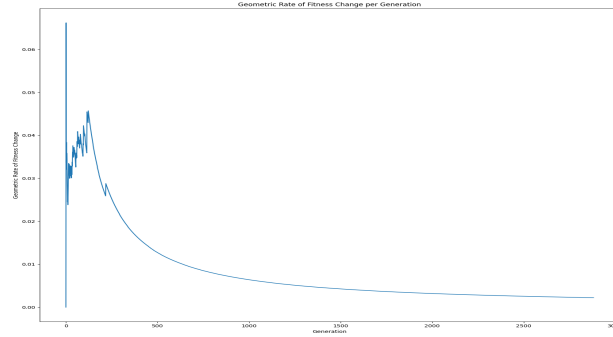


Figura 70: *Geometric rate of fitness change* para N-reinas con $n = 64$ y $\lambda = 5$. Elaboración propia

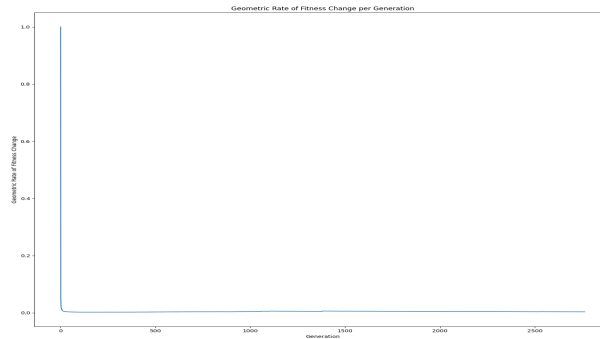


Figura 71: *Geometric rate of fitness change* para N-reinas con $n = 625$ y $\lambda = 5$. Elaboración propia

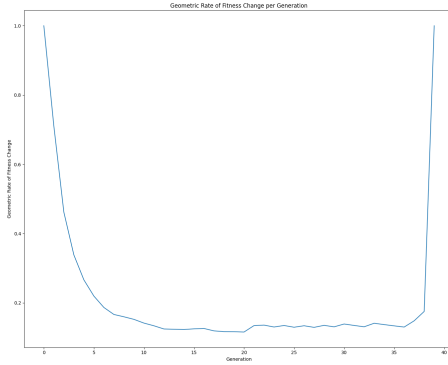
Primero, el caso de bajo tamaño, encontrado en la figura 70 presenta una tasa cercana a 0,03 para las primeras 400 generaciones, lo cual puede parecer bajo; sin embargo, lo que es relevante para el análisis de esta métrica es la consistencia con la que se mantiene este valor. Luego, al cruzar el umbral de las 400 generaciones, su comportamiento es de una curva suave, lo que indica que el algoritmo sigue convergiendo hacia una mejor solución de forma consistente, siendo este un caso ideal para esta métrica.

En el segundo caso, al enfrentarse a un problema de gran envergadura, con una población muy pequeña, el algoritmo no logra mantener una mejora constante o considerable, por lo que la curva presenta una caída inmediata hacia 0 para esta métrica. Este fenómeno se observa en la figura 71

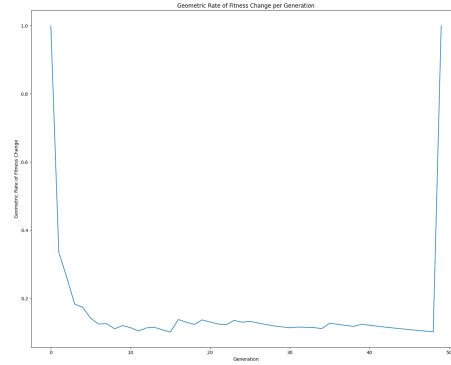
Para analizar el caso de FGA se toman casos de los problemas *linear function with harmonic weights* (desde ahora *linear* e *ising torus*). Para ambos problemas se utiliza el tamaño de problema $n = 64$, mientras que para los tamaños de población se utiliza $\lambda = 50$ para el primer problema, y $\lambda = 15$ para el segundo.

En el primer caso, para el problema *linear*, se puede observar la comparación en la figura 72.

Aquí, se puede deducir que, gracias a la menor *mutation strength* en el caso $F = 1,5$, hay mayor presencia de cambios a través de *crossover* y dada la mayor exploración que permite esto, mantiene una mayor tasa promedio en las primeras generaciones. Esto le permite incluso reducir la cantidad de generaciones que toma para llegar al óptimo en 10 ($F = 1,5$ toma 39 generaciones y $F = 2$ toma 49).



(a) *Geometric rate of fitness change* para *linear* con $n = 64$ y $\lambda = 50$. Caso FGA $F = 1,5$.
Elaboración propia

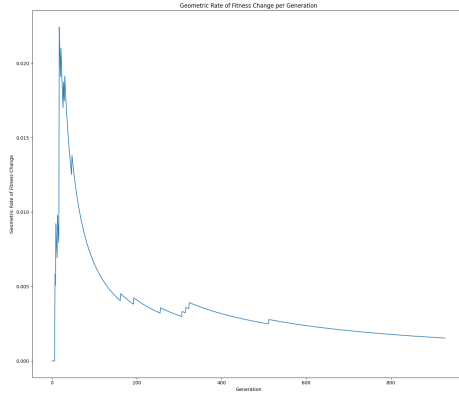


(b) *Geometric rate of fitness change* para *linear* con $n = 64$ y $\lambda = 50$. Caso FGA $F = 2$.
Elaboración propia

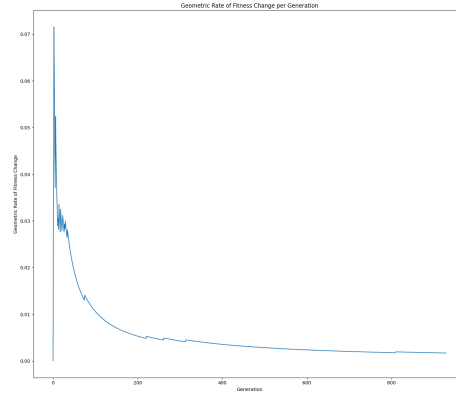
Figura 72: Comparativo *Geometric rate of fitness change* para *linear* con $n = 64$ y $\lambda = 50$.

Luego, en el caso del problema *ising torus*, se ve una comparación donde el efecto de los operadores no afecta al resultado final, ya que ninguna de las dos variantes del algoritmo encuentra el óptimo en su límite de evaluaciones. Sin embargo, el promedio geométrico de la tasa de cambio demuestra una búsqueda muy distinta. En las figuras 73a y 73b se pueden observar los gráficos correspondientes para cada variante del algoritmo.

En el caso de $F = 2$, al tener una prevalencia del operador de mutaciones, se observa una tasa muy alta para las primeras 100 generaciones. En comparación al caso de $F = 1,5$, donde se puede observar que, si bien la tasa es menor, es más constante y se mantiene hasta la generación 200, donde alcanza el mismo nivel que la variante con la que se le compara y ambas mantienen dicho nivel hasta el final de la ejecución. De esto se concluye que, dependiendo del problema y el comportamiento de los operadores, la velocidad con la que se encuentran mejores soluciones puede variar en gran medida. Incluso, en casos donde se encuentran soluciones similares en su aptitud, la forma en la que se encuentran estas demuestra grandes variaciones, lo cual permite, por ejemplo, en el caso de $F = 2$, concluir que se debe dar mayor prioridad a los distintos operadores luego de que la tasa baja de cierto punto, para poder explorar en mayor medida el espacio de búsqueda y llegar más rápidamente a una mejor solución.



(a) Geometric rate of fitness change para ising torus con $n = 64$ y $\lambda = 15$. Caso FGA $F = 1,5$. Elaboración propia



(b) Geometric rate of fitness change para ising torus con $n = 64$ y $\lambda = 15$. Caso FGA $F = 2$. Elaboración propia

Figura 73: Comparativo Geometric rate of fitness change para ising torus con $n = 64$ y $\lambda = 15$.

4.3.4. E-value

Como ultima métrica directamente relacionada a la aptitud del algoritmo, se tiene *E-value*, la cual permite realizar el análisis sobre cuantas soluciones suficientemente cercanas al óptimo se obtienen en relación al espacio de búsqueda total recorrido. Para definir que soluciones están suficientemente cerca del óptimo se definen 3 casos de estudio asociados a una relación porcentual con el óptimo de cada problema, siendo estos los casos donde la solución es mayor o igual a un 50 %, 75 % y 90 % del valor de aptitud del óptimo. Es decir, esta métrica, numéricamente será mayor en la medida que se obtengan soluciones con un valor de aptitud mayor a este cálculo sobre el valor óptimo, siempre y cuando se haya explorado una sección menor del espacio de búsqueda. El tamaño del espacio de búsqueda se calcula de forma teórica con la ecuación 40.

Para exponer los distintos valores que puede tomar esta métrica, se selecciona el problema de *Ising ring* con $n = \{64, 100, 625, 5625\}$ y $\lambda = \{5, 15, 50\}$.

E-value									
n	50 %			75 %			90 %		
	$\lambda = 5$	$\lambda = 15$	$\lambda = 50$	$\lambda = 5$	$\lambda = 15$	$\lambda = 50$	$\lambda = 5$	$\lambda = 15$	$\lambda = 50$
64	0.269	0.269	0.267	0.267	0.247	0.220	0.255	0.049	0.180
100	0.660	0.660	0.658	0.650	0.634	0.548	0.487	0.487	0.394
625	25.984	25.951	25.836	22.641	19.039	7.592	0.000	0.000	0.000
5625	2106.680	2106.590	2097.160	0.000	0.000	0.000	0.000	0.000	0.000

Tabla 7: Comparación de E-value para *Ising ring*

De esta tabla se puede extraer como afectan los distintos parámetros a la eficiencia en la

búsqueda de soluciones.

A partir de los resultados obtenidos se concluye, primero, que el tamaño de la población no afecta en sí a la cantidad de soluciones que se extraen del problema (que estén por sobre su umbral correspondiente), ya que el valor se mantiene consistente a través de los distintos tamaños de población. La excepción es el caso donde se evalúa sobre un 75 % del óptimo, donde para el tamaño de problema 625 se ve una reducción en la eficiencia, esto se puede deber a que al aumentar la cantidad de agentes de búsqueda, aumenta la cantidad de soluciones que se evalúan por generación e implica una mayor exploración del espacio de búsqueda, la cual puede no ser fructífera.

Luego, otra excepción al comportamiento normal del problema se da en la evaluación al 90 % del caso con tamaño $n = 64$ se observa un aumento con la población de tamaño 50, en este caso se explica con el haber obtenido más soluciones suficientemente buenas y posiblemente, dado que la población aumenta considerablemente, en un espacio de búsqueda pequeño, puede llevar a una exploración más completa en menos generaciones.

Ahora, esta métrica es sumamente valiosa a la hora de hacer una comparación directa, tal como en el caso comparativo de FGA con $F = 1,5$ y $F = 2$, esto debido a que permite comprender tanto el conjunto de soluciones encontradas y su aptitud, como la capacidad exploratoria del algoritmo y como este llega a sectores del espacio de búsqueda de alto potencial de aptitud.

Para este análisis se utilizan tres problemas distintos, *OneMax*, *OneMax* con perturbación de aptitud (*OneMax* + W) e *ising triangular*. Estos últimos dos problemas presentan un espacio de búsqueda complejo con altas posibilidades de estancamiento. Dichos problemas se presentan con su tamaño $n = 625$, sus 3 tamaños de población $\lambda = \{5, 15, 50\}$ y para definir aquella solución que sea suficientemente buena, se utilizan 3 valores porcentuales respecto del óptimo del problema $\{50\%, 75\%, 90\%\}$. Las tablas utilizadas en este análisis corresponden a la tabla 8 para el problema *OneMax*, 9 para el problema *OneMax* + W y 10 para el problema *ising triangular*.

E-value <i>OneMax</i> en FGA									
	50 %			75 %			90 %		
F	$\lambda = 5$	$\lambda = 15$	$\lambda = 50$	$\lambda = 5$	$\lambda = 15$	$\lambda = 50$	$\lambda = 5$	$\lambda = 15$	$\lambda = 50$
1.5	25.983	25.947	25.818	23.394	21.439	16.248	18.803	11.548	0.000
2	39.893	25.960	25.850	33.714	21.097	18.347	25.223	13.623	3.982

Tabla 8: Comparación de E-value para *OneMax* utilizando FGA

En ambos casos de *OneMax*, se puede apreciar que los algoritmos son muy similares en su rendimiento para cada caso porcentual, con dos diferencias relevantes. En *OneMax* se puede observar una clara ventaja del algoritmo con $F = 2$ y $\lambda = 5$ para encontrar una mayor cantidad de soluciones por sobre el 50 % de la aptitud del óptimo. Sin embargo, es de mayor relevancia el hecho de que en ambos problemas es posible apreciar que, en el caso donde se evalúa respecto del 90 % del óptimo, se ve una importante ventaja para el

E-value <i>OneMax</i> + <i>W</i> en FGA									
	50 %			75 %			90 %		
<i>F</i>	$\lambda=5$	$\lambda=15$	$\lambda=50$	$\lambda=5$	$\lambda=15$	$\lambda=50$	$\lambda=5$	$\lambda=15$	$\lambda=50$
1.5	25.973	25.952	25.815	22.613	20.397	17.793	15.586	10.598	4.393
2	25.996	25.944	25.830	23.135	19.281	17.983	17.398	9.137	5.577

Tabla 9: Comparación de E-value para *OneMax* + *W* utilizando FGA

E-value <i>Ising triangular</i> en FGA									
	50 %			75 %			90 %		
<i>F</i>	$\lambda=5$	$\lambda=15$	$\lambda=50$	$\lambda=5$	$\lambda=15$	$\lambda=50$	$\lambda=5$	$\lambda=15$	$\lambda=50$
1.5	25.949	25.873	25.638	19.651	14.218	0.000	0.000	0.000	0.000
2	25.988	25.921	25.740	19.280	14.349	0.051	0.000	0.000	0.000

Tabla 10: Comparación de E-value para *Ising triangular* utilizando FGA

algoritmo que prioriza el uso de mutaciones. Esto indicaría que, utilizando un enfoque con menor exploración del espacio a través de *crossover* y un mayor enfoque en la explotación de la solución, FGA se hace más eficiente para *OneMax*.

Sin embargo, para *Ising triangular*, se aprecia una diferencia menor en el desempeño de ambas variantes del algoritmo, lo que indicaría que para esta categoría de problemas, el algoritmo en sí es el que se debe cambiar para alcanzar los niveles de eficiencia vistos anteriormente.

4.3.5. Entropy Diversity

Como única métrica relacionada directamente a estudiar el movimiento por el espacio de búsqueda (independientemente de la aptitud presentada por la solución), se tiene *Entropy Diversity*. Esta métrica ejecuta una transformación de la solución candidata en cada evaluación la cual analiza la relación entre variables de cada par de soluciones y reduce esta relación a coordenadas tridimensionales, de forma que, con esta relación se pueden aproximar 8 cuadrantes para utilizar como espacio de búsqueda conocido. Teniendo las coordenadas para cada solución, se puede utilizar la ecuación 14 para obtener un valor numérico que represente la diversidad de la población.

Al ejecutar esta operación para cada generación, se puede graficar y obtener una representación de como las soluciones se distribuyen por el espacio propuesto. Para la demostración y análisis de esta métrica se seleccionan 3 problemas, *N-reinas*, *Ising torus* y *OneMax* + *W*, donde para los dos primeros se exponen los gráficos obtenidos para un tamaño $n = 625$ con sus 3 tamaños de población (5,15,50). El caso de *OneMax* + *W*, siendo ilustrativo de un caso extremo, solo se presenta en su formato $n = 625$ y $\lambda = 15$.

En las figuras 74 y 75 se encuentran los casos donde $\lambda = 5$. En estos casos, es evidente que

una población pequeña, en la práctica, solo tiene 2 comportamientos, aquel donde la gran mayoría de la población se encuentra en un sector del espacio de búsqueda, como es el de *Ising torus*, donde la diversidad se estanca y reactiva en dos ocasiones, indicando que en los valles, la población explota la aptitud de una solución en un sector del espacio, para luego diversificar y volver a repetir el proceso.

El segundo caso, visto en el comportamiento de N-reinas, es aquel donde se puede realizar el análisis desde dos puntos de vista. El primero permite llegar a la conclusión de que el espacio de búsqueda es sumamente rugoso, por lo que nunca se encuentra en un sector consistente donde poder explotar la solución, y por ende, la diversidad de la población no baja a 0, o bien, en el segundo análisis, se puede decir que las soluciones de alta aptitud se concentran en una cantidad finita, mayor a 1, de sectores, por lo que la información de las soluciones se comparte entre los individuos encontrados en dichos sectores, más no se concentran en uno.

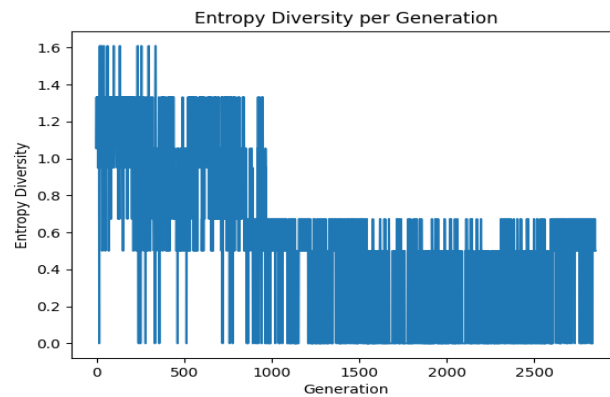


Figura 74: *Entropy Diversity* para N-reinas con $n = 625$ y $\lambda = 5$. Elaboración propia

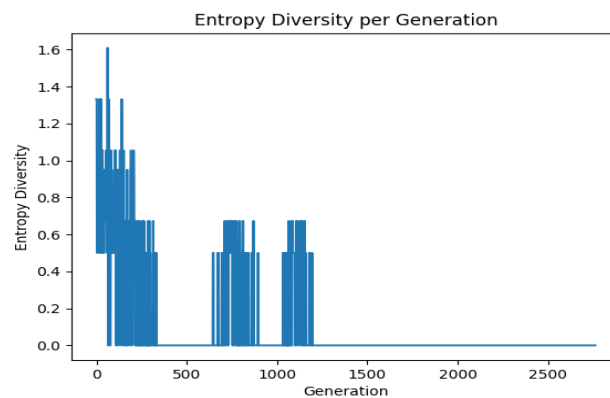


Figura 75: *Entropy Diversity* para *Ising torus* con $n = 625$ y $\lambda = 5$. Elaboración propia

Ahora, el análisis de la situación con una población pequeña permite observar patrones que luego se repiten, por ejemplo, en las figuras 76 y 77 se pueden observar los casos con $\lambda = 15$ para N-reinas e *Ising torus*.

Para *Ising torus* se ve un comportamiento similar a su anterior ejemplo, donde la población tiende a concentrarse en un sector, explorar una vez se estanca y volver a concentrarse, pero en el caso de N-reinas se observa este comportamiento en menor medida hasta alrededor de la generación número 375, para luego exhibir un comportamiento como el de su anterior ejemplo, por lo que se puede ahondar en como se comporta el algoritmo en este set de problemas a medida que se acerca al óptimo y sugerir que su comportamiento es de explotación seguida de exploración hasta encontrar un conjunto de sectores del espacio de búsqueda donde recolectar información de soluciones cercanas al óptimo.

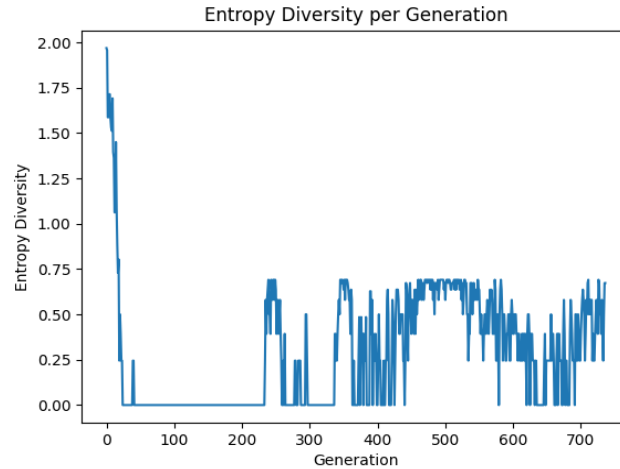


Figura 76: *Entropy Diversity* para N-reinas con $n = 625$ y $\lambda = 15$. Elaboración propia

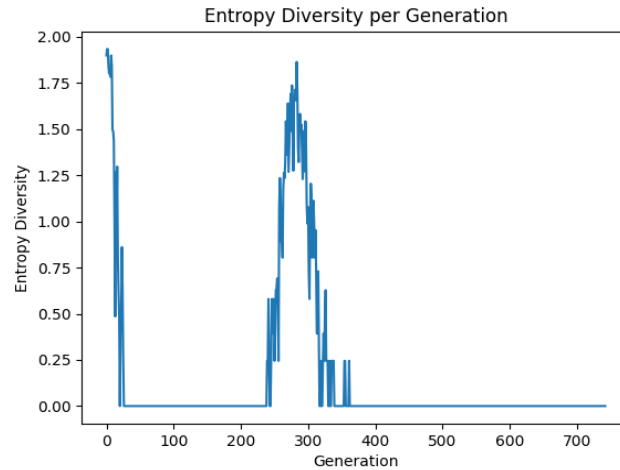


Figura 77: *Entropy Diversity* para *Ising torus* con $n = 625$ y $\lambda = 15$. Elaboración propia

Los casos finales para estos problemas, con su tamaño de población $\lambda = 50$ se hacen interesantes de analizar debido a este gran tamaño, donde se puede asumir preliminarmente que la diversidad debe ser alta o, al menos, mantenerse en el tiempo solo por el nivel de exploración que permite una población de gran tamaño. Sin embargo, el caso de N-reinas (figura

78) prueba lo contrario a esta suposición inicial, en este caso es posible ver que el patrón de explotación y exploración separadas se mantiene, y si bien, no se observa el patrón visto antes en sectores cercanos al óptimo, es importante notar como en un caso donde el algoritmo debería explorar en mayor medida, el algoritmo se mueve hacia un sector del espacio de búsqueda con mayor facilidad. Para *Ising torus* (figura 79) la explicación es totalmente distinta, y esta se condice con la hipótesis inicial, donde se observa una curva donde, incluso, nunca se llega a un valor de 0 en la diversidad, indicando que constantemente existen agentes de búsqueda en distintos sectores del espacio.

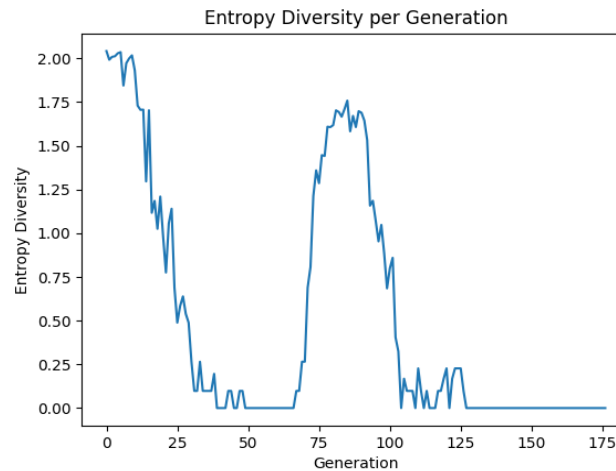


Figura 78: *Entropy Diversity* para N-reinas con $n = 625$ y $\lambda = 50$. Elaboración propia

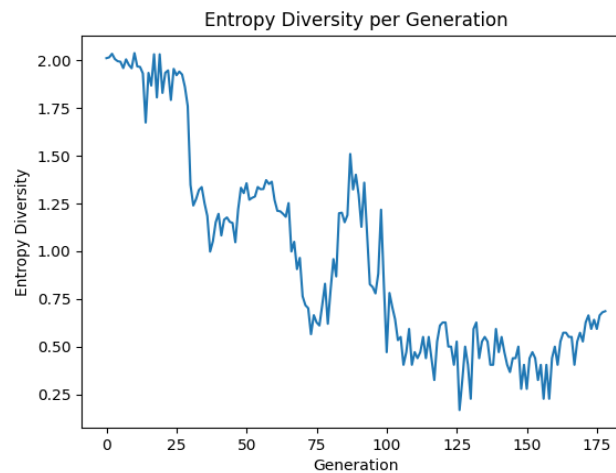


Figura 79: *Entropy Diversity* para *Ising torus* con $n = 625$ y $\lambda = 50$. Elaboración propia

A modo de contraste, como un caso muy lejano de ser óptimo, ya se ha hablado de como *OneMax + W* dificulta la búsqueda de soluciones, estancando la exploración y explotación de la solución con su modificación del espacio de búsqueda. Extendiendo el análisis realizado

en 4.3.3, es fácil ver como en la figura 80, la población muy rápidamente se ve estancada en un óptimo local, y al no poder salir del vecindario de este, se mantiene en un solo sector del espacio de búsqueda, llevando la diversidad a 0, complementando el análisis presente para la convergencia, donde el estancamiento se visualiza de igual forma.

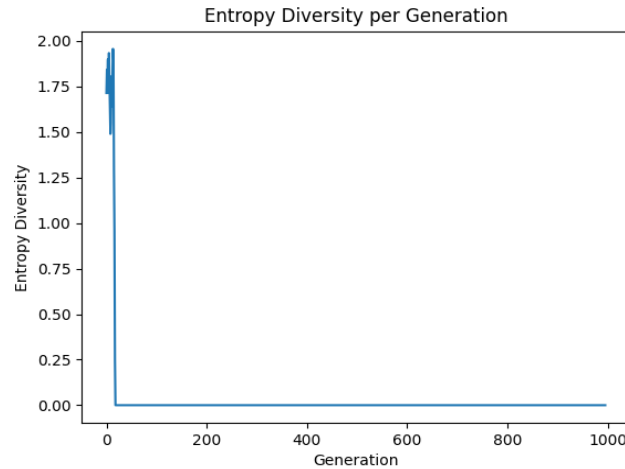


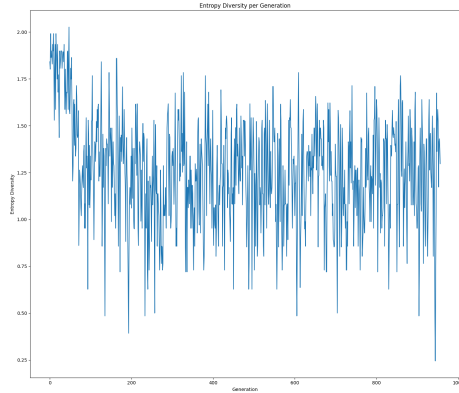
Figura 80: *Entropy Diversity* para *OneMax + W* con $n = 625$ y $\lambda = 15$. Elaboración propia

Finalmente, a modo de asemejar un estudio con uso de esta herramienta, el caso de FGA es analizado utilizando los problemas N-reinas y *OneMax + W* (*OneMax* con perturbación de aptitud). El primero de estos problemas se utiliza como un buen ejemplo de un espacio de búsqueda con una estructura simétrica dadas las fuertes relaciones que se establecen entre variables a medida que se resuelven las distintas restricciones del problema. El segundo de estos problemas ya ha sido utilizado por su complejo espacio de búsqueda y es relevante estudiar el estancamiento de las soluciones en este tipo de situaciones.

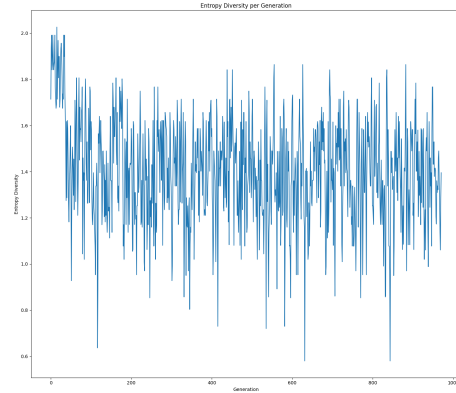
Los problemas antes mencionados se estudian bajo tamaños de población $\lambda = \{15, 50\}$, esta elección se hace para tener una buena visibilidad de los resultados gracias al gran conjunto de soluciones por generación y evitando fuertes cambios en la métrica a partir del cambio en tan solo un individuo. Además, para N-reinas se toma el tamaño $n = 64$, mientras que para *OneMax + W* se utiliza $n = 625$, esto con el objetivo de analizar por sobre problemas con distinta complejidad.

En las figuras 81a y 81b se encuentran los gráficos asociados al problema de N-reinas con $\lambda = 15$, mientras que en 82a y 82b se aprecian los resultados utilizando $\lambda = 50$

Es fácil ver que en la instancia con $\lambda = 15$, ambos algoritmos tienen una diversidad muy similar, aunque en el caso de $F = 2$ se puede notar una diversidad constantemente más alta, lo cual se evidencia en que su valor mínimo llega a ser cercano a 0,6, mientras que en el caso de $F = 1,5$ se observa una gran caída a 0,25 hacia el final de su ejecución. Este análisis se revierte con el caso donde $\lambda = 50$. En este caso se puede notar que, si bien la diversidad baja a niveles similares para ambos algoritmos luego de la generación 50, el mayor potencial

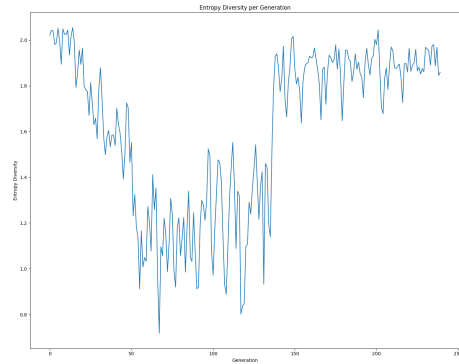


(a) *Entropy diversity* para N-reinas con $n = 64$ y $\lambda = 15$. Caso FGA $F = 1,5$. Elaboración propia

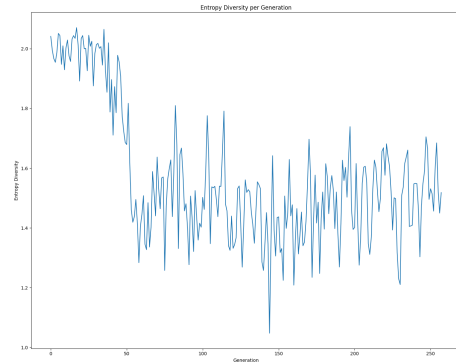


(b) *Entropy diversity* para N-reinas con $n = 64$ y $\lambda = 15$. Caso FGA $F = 2$. Elaboración propia

Figura 81: Comparativo *Entropy diversity* para N-reinas con $n = 64$ y $\lambda = 15$.



(a) *Entropy diversity* para N-reinas con $n = 64$ y $\lambda = 50$. Caso FGA $F = 1,5$. Elaboración propia



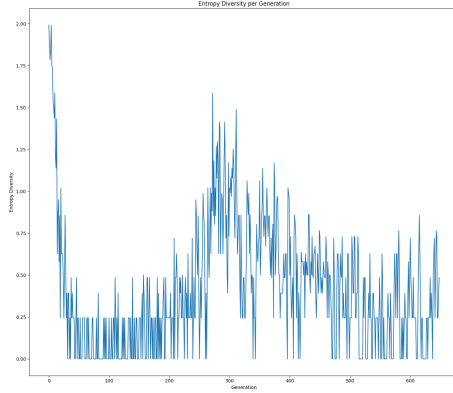
(b) *Entropy diversity* para N-reinas con $n = 64$ y $\lambda = 50$. Caso FGA $F = 2$. Elaboración propia

Figura 82: Comparativo *Entropy diversity* para N-reinas con $n = 64$ y $\lambda = 50$.

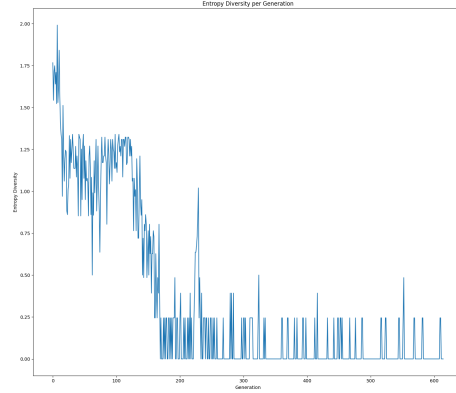
de exploración del algoritmo con $F = 1,5$ permite que en las ejecuciones cercanas a la generación 150 se pueda apreciar una alza importante en la diversidad, indicando que el algoritmo explora el espacio de búsqueda de forma mucho más completa que el caso de $F = 2$, donde se prioriza la explotación.

Luego, en el caso de *OneMax + W* las diferencias entre algoritmos se hacen evidentes a través de las figuras 83a y 83b para el caso donde $\lambda = 15$ y 84a y 84b para el tamaño de población $\lambda = 50$.

En las figuras asociadas a $\lambda = 15$ se puede apreciar que, si bien ambos algoritmos tienden al estancamiento en un número acotado de regiones del espacio de búsqueda, es fácil ver que

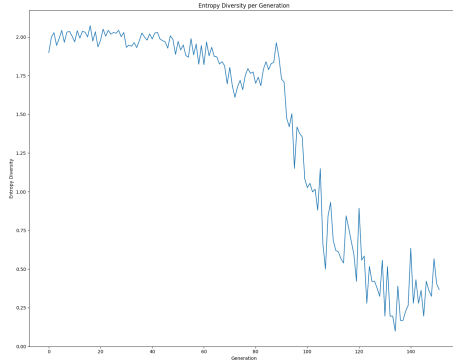


(a) *Entropy diversity* para *OneMax + W* con $n = 625$ y $\lambda = 15$. Caso FGA $F = 1,5$. Elaboración propia

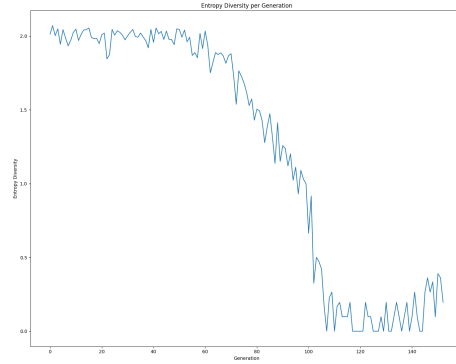


(b) *Entropy diversity* para *OneMax + W* con $n = 625$ y $\lambda = 15$. Caso FGA $F = 2$. Elaboración propia

Figura 83: Comparativo *Entropy diversity* para *OneMax + W* con $n = 625$ y $\lambda = 15$.



(a) *Entropy diversity* para *OneMax + W* con $n = 625$ y $\lambda = 50$. Caso FGA $F = 1,5$. Elaboración propia



(b) *Entropy diversity* para *OneMax + W* con $n = 625$ y $\lambda = 50$. Caso FGA $F = 2$. Elaboración propia

Figura 84: Comparativo *Entropy diversity* para *OneMax + W* con $n = 625$ y $\lambda = 50$.

el algoritmo que favorece la explotación de la solución ($F = 2$), tiene a una diversidad igual a 0 luego de la generación número 170. Mientras, el algoritmo con mayor diversidad en su utilización de los operadores ($F = 1,5$), incluso si tiende a mantener una baja diversidad, cerca de la iteración 300 presenta una importante exploración del espacio de búsqueda, manteniendo niveles mayores a su contraparte en la mayoría de casos. Aún más evidente, en el caso con $\lambda = 50$, se puede apreciar una fuerte caída en la diversidad para ambos algoritmos. Sin embargo, esta se produce en la generación 92 para el algoritmo con $F = 1,5$, mientras que para el caso con $F = 2$ esta caída en diversidad se produce en la generación 77. Es de notar también, que el primer algoritmo nunca cae a una diversidad de 0 en toda su ejecución, mientras que el segundo algoritmo, alrededor de la generación número 110, cae a su diversidad 0 para posteriormente tener dificultad en su recuperación.

4.3.6. Métricas de operadores

Las métricas de operadores diseñadas en 3.3.2 están dirigidas hacia descubrir como interactúan los distintos operadores que puede tener un problema y como estos afectan sus soluciones.

Es importante notar que en el contexto del algoritmo escogido existen dos operadores, *crossover* y *mutation*, por lo tanto, el análisis será por sobre estos dos operadores.

4.3.6.1. Tasa de éxito

Esta métrica es simple pero informativa para la ejecución del problema. El análisis de ésta indica cuan bueno es un operador respecto de modificar la solución, ya sea en caso de empeorar o mejorar su valor de aptitud, ya que en cualquier caso implica exploración del espacio de búsqueda, o en el caso de mejora de aptitud, también implica explotación de este.

Esta métrica se analizará para el problema *Ising torus*. Se seleccionan instancias con gran diferencia en tamaño de problema y tamaño de población, estos toman los valores $n = \{64, 5625\}$ y $\lambda = \{5, 50\}$ para realizar una completa demostración de la información extraída. El análisis se realiza a partir de las figuras 85, 86, 87 y 88.

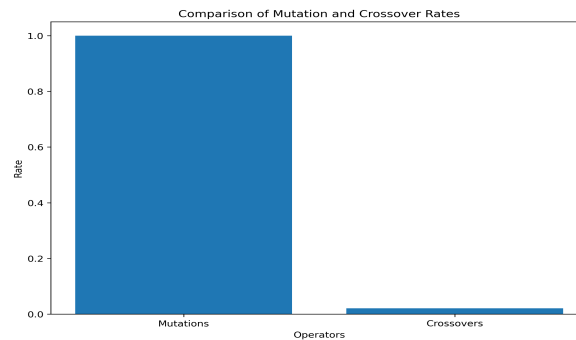


Figura 85: Tasa de éxito por operador para *Ising torus* con $n = 64$ y $\lambda = 5$. Elaboración propia

Es fácil notar, primero, que las mutaciones siempre mantienen una tasa de 1, es decir, siempre inducen un cambio, por lo que se concluye que incluso en espacios de búsqueda pequeños, este operador consistentemente modifica la aptitud de la solución.

No así el operador de *crossover*, el cual muestra una diferencia gigantesca entre instancias, indicando que ambos factores de tamaño son relevantes para mantener este operador en correcto funcionamiento. En el caso del tamaño de población, este le permite generar soluciones a partir de una selección más grande de soluciones y, por lo tanto, evitar obtener aptitudes o soluciones idénticas. Además, en el caso del tamaño del problema, este aumenta el tamaño del espacio de búsqueda, evitando caer en las mismas soluciones una y otra vez.

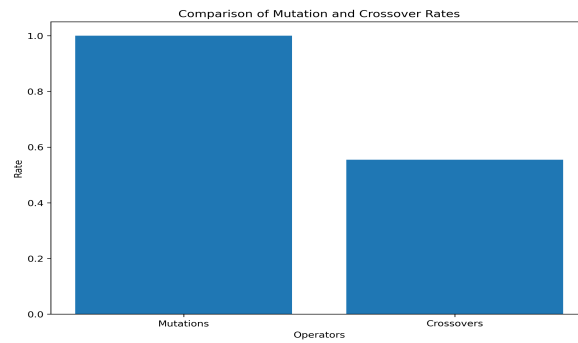


Figura 86: Tasa de éxito por operador para *Ising torus* con $n = 64$ y $\lambda = 50$. Elaboración propia

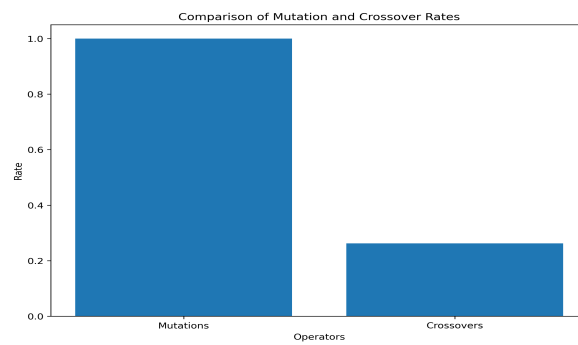


Figura 87: Tasa de éxito por operador para *Ising torus* con $n = 5625$ y $\lambda = 5$. Elaboración propia

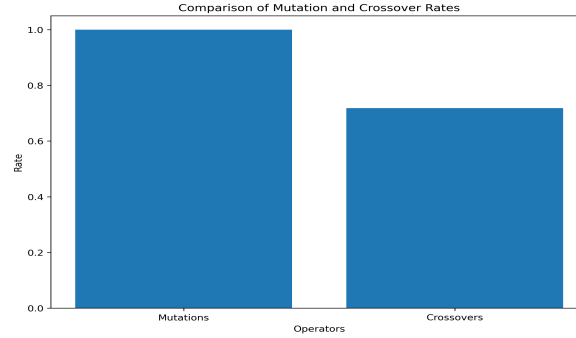
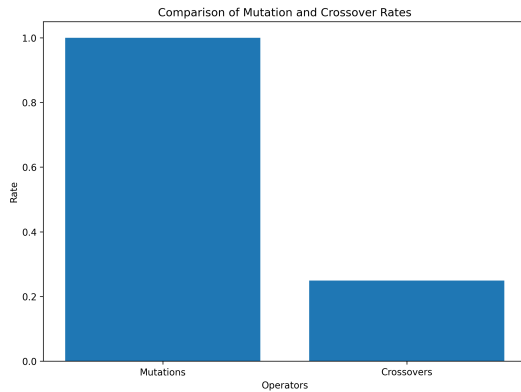


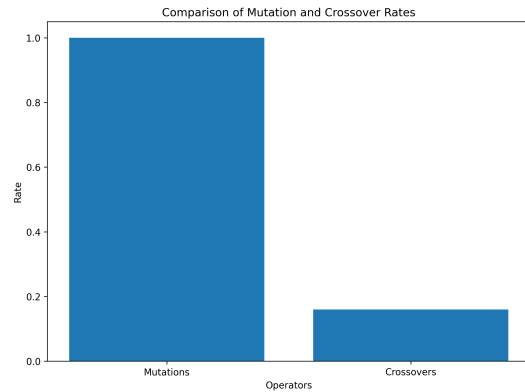
Figura 88: Tasa de éxito por operador para *Ising torus* con $n = 5625$ y $\lambda = 50$. Elaboración propia

En el caso de estudio de FGA, se busca entender como ambas variantes se comportan, teniendo en cuenta que el algoritmo con $F = 2$ mantiene una menor cantidad de *crossovers*. Sin embargo, a pesar de ejecutarse en una menor cantidad de iteraciones, pueden ser proporcionalmente más efectivos que en el caso comparado.

Para evaluar esta teoría se utilizan 3 casos distintos. Primero, se utiliza N-reinas con tamaño $n = 64$ y población de tamaño $\lambda = 50$, este problema se escoge por su espacio de búsqueda complejo, el cual se beneficia de explorar a través de una gran población. Luego, se evalúa *OneMax* con tamaño $n = 625$ y población de tamaño $\lambda = 15$ para comparar un problema con una búsqueda más simple y un tamaño de población moderado. Finalmente, también con un espacio de búsqueda complejo, pero que no presenta las simetrías de N-reinas, se utiliza *ising triangular* con tamaño $n = 625$ y población de tamaño $\lambda = 15$.



(a) Tasa de éxito por operador para N-reinas con $n = 64$ y $\lambda = 50$. Caso FGA $F = 1,5$. Elaboración propia



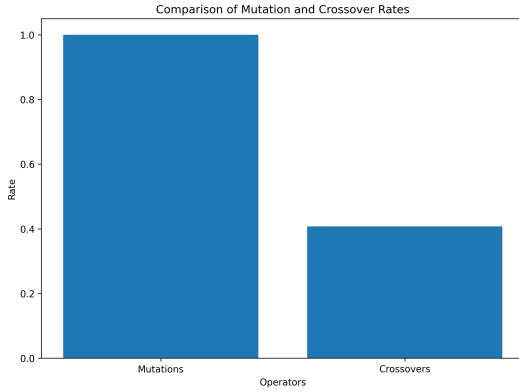
(b) Tasa de éxito por operador para N-reinas con $n = 64$ y $\lambda = 50$. Caso FGA $F = 2$. Elaboración propia

Figura 89: Comparativo de tasa de éxito por operador para N-reinas con $n = 64$ y $\lambda = 50$.

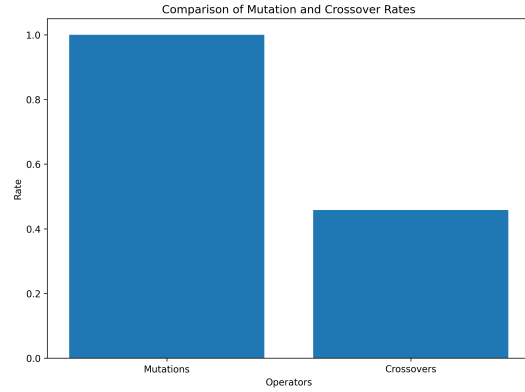
Para el primer caso, la figura 89 muestra una comparativa entre ambas variantes del algoritmo, donde se puede apreciar que la tasa de éxito se reduce para el caso con mayor *mutation*

strength, lo que indica que, a pesar de realizar menos *crossovers*, estos son también menos efectivos. Este fenómeno se puede deber a que la fuerte explotación de una solución puede ubicarla en un área del espacio de búsqueda muy lejana de soluciones que hagan efectiva la ejecución de *crossover* y, por ende, se rechaza la solución que este operador encuentre.

Luego, en el caso de *OneMax*, observado en las figuras 90a y 90b, se comprueba la teoría anteriormente planteada, donde el caso con $F = 2$ mantiene una mayor tasa de éxito para *crossover*, debido a que se tiene una menor cantidad de ejecuciones, pero una mayor proporción de estas ejecuciones producen una solución que modifique la aptitud de la solución.

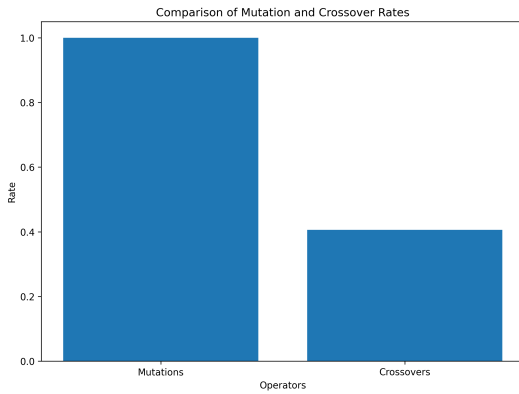


(a) Tasa de éxito por operador para *OneMax* con $n = 625$ y $\lambda = 15$. Caso FGA $F = 1,5$. Elaboración propia

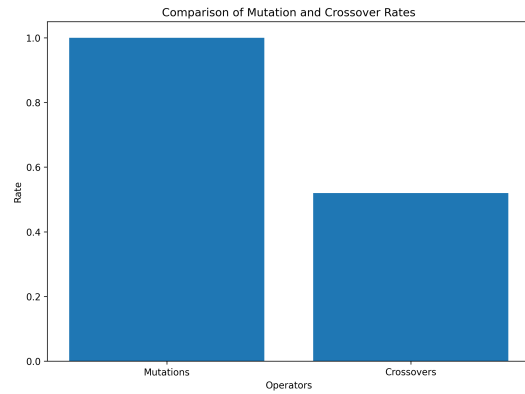


(b) Tasa de éxito por operador para *OneMax* con $n = 625$ y $\lambda = 15$. Caso FGA $F = 2$. Elaboración propia

Figura 90: Comparativo de tasa de éxito por operador para *OneMax* con $n = 625$ y $\lambda = 15$.



(a) Tasa de éxito por operador para *Ising triangular* con $n = 625$ y $\lambda = 15$. Caso FGA $F = 1,5$. Elaboración propia



(b) Tasa de éxito por operador para *OneMax + W* con $n = 625$ y $\lambda = 15$. Caso FGA $F = 2$. Elaboración propia

Figura 91: Comparativo de tasa de éxito por operador para *Ising triangular* con $n = 625$ y $\lambda = 15$.

Finalmente, en las figuras 91a y 91b se muestran los gráficos asociados a *ising triangular*, donde, a pesar de tener un espacio de búsqueda complejo, el algoritmo con $F = 2$ mantiene una mayor tasa de éxito en su movimiento de *crossover*, esto indica que el espacio de búsqueda, incluso con sus soluciones ubicadas en sectores de mayor aptitud, permite una exploración efectiva, de la misma forma que *OneMax*.

4.3.6.2. Tasa de mejora y deterioro

Finalmente, en este caso se busca analizar más granularmente el comportamiento de los operadores por sobre la aptitud de las soluciones que modifican. Esto se realiza a partir de analizar los cambios inmediatos en aptitud luego de una ejecución del operador, es decir, en caso de que exista mejora, se calcula la diferencia a partir del estado original de la solución con la nueva aptitud encontrada.

Los casos a analizar son para los problemas LABS y *OneMax* con *W-model* en perturbación de aptitud, la cual modifica la distancia entre óptimos locales a estar a distancias de 5 en términos del valor de aptitud evaluado. Ambos problemas se analizan en sus tamaños $n = 64$ y $\lambda = 15$.

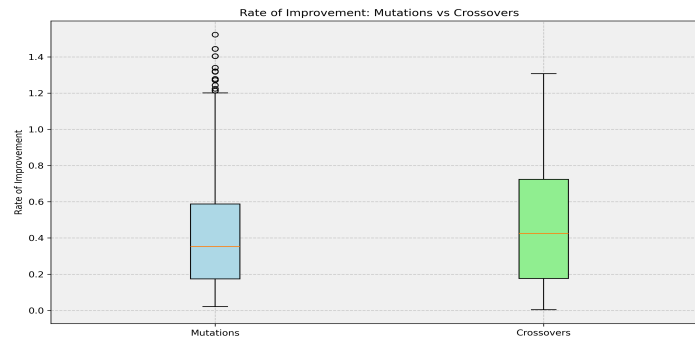


Figura 92: Tasa de mejora por operador para LABS con $n = 64$ y $\lambda = 15$. Elaboración propia

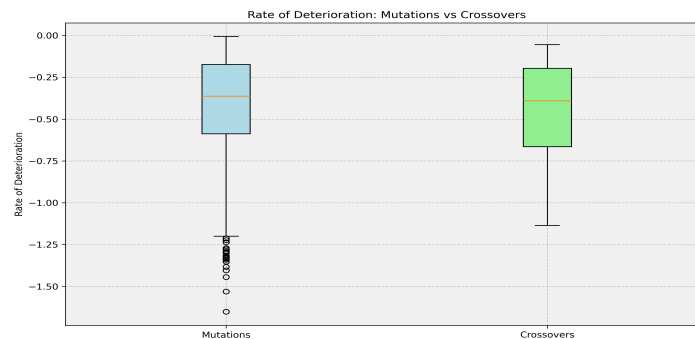


Figura 93: Tasa de deterioro por operador para LABS con $n = 64$ y $\lambda = 15$. Elaboración propia

Las figuras 92 y 93 exponen ambas tasas para el problema LABS. En estos gráficos se puede notar que, en el caso del primer problema, ambos operadores presentan alta consistencia en los cambios que realizan sobre la aptitud de las soluciones, aunque las mutaciones presentan muchos más *outliers*. Esto puede ser fácilmente explicado con la anterior métrica, la cual demuestra que las mutaciones son más consistentemente exitosas, y por ende, tomadas en cuenta para esta métrica.

Muy distinto del caso de LABS, *OneMax + W* (presentado en 94 y 95) demuestra un fuerte contraste entre mutaciones y *crossover*, donde las mutaciones registran un valor sumamente consistente (8 para mejoras, -7.6 para deterioros) y una gran serie de *outliers*. Por otra parte, los valores para *crossover* no indican gran presencia de *outliers*, pero sí presentan menos consistencia en los cuartiles alrededor de la mediana.

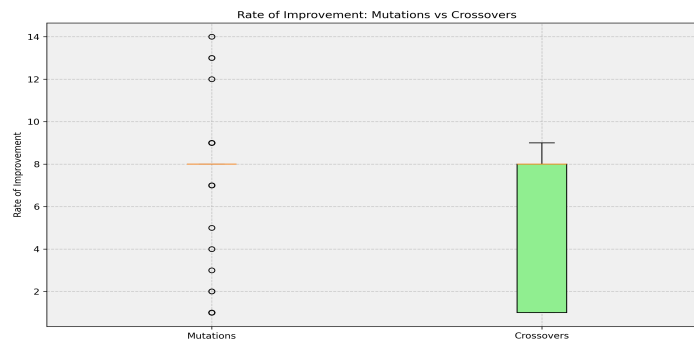


Figura 94: Tasa de mejora por operador para *OneMax + W* con $n = 64$ y $\lambda = 15$. Elaboración propia

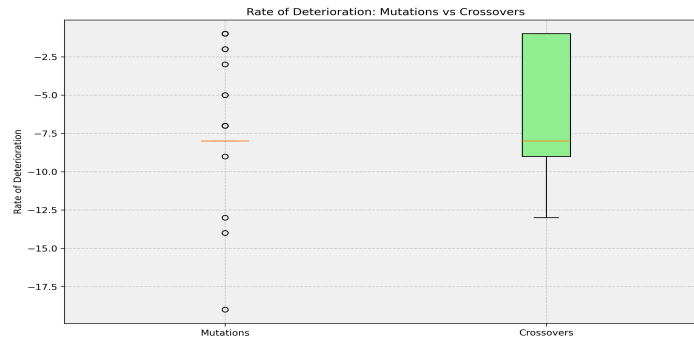
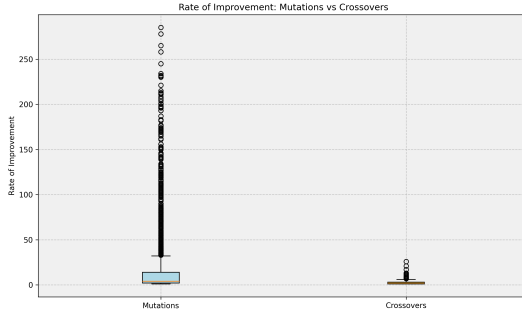
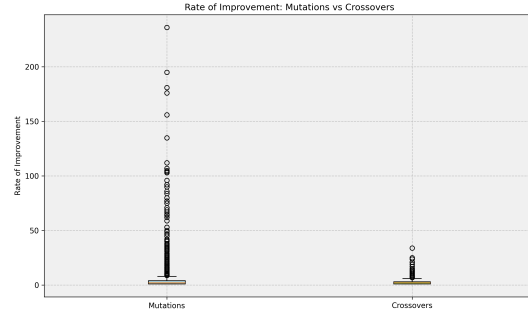


Figura 95: Tasa de deterioro por operador para *OneMax + W* con $n = 64$ y $\lambda = 15$. Elaboración propia

Para concluir este estudio de métricas de desempeño, en el caso comparativo de FGA se utilizan los problemas *OneMax* y *concatenated trap*, ambos en su tamaño $n = 625$ y tamaño de población $\lambda = 15$. Las comparativas del primer caso se presentan en las figuras 96 y 97 para la tasa de mejora y deterioro, respectivamente. Mientras, las comparativas del segundo caso se presentan en las figuras 98 y 99 siguiendo la misma estructura anterior.

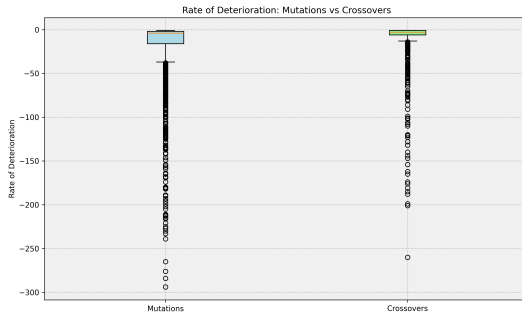


(a) Tasa de mejora por operador para *OneMax* con $n = 625$ y $\lambda = 15$. Caso FGA $F = 1,5$. Elaboración propia

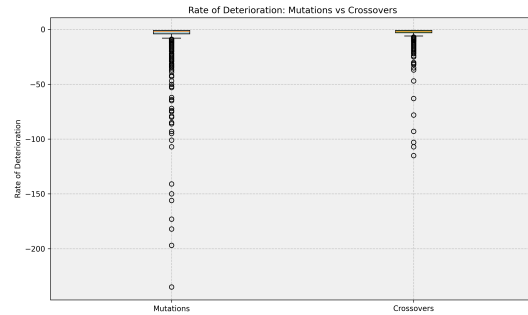


(b) Tasa de mejora por operador para *OneMax* con $n = 625$ y $\lambda = 15$. Caso FGA $F = 2$. Elaboración propia

Figura 96: Comparativo de tasa de mejora por operador para *OneMax* con $n = 625$ y $\lambda = 15$.



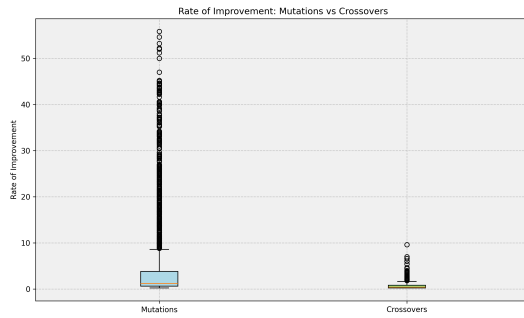
(a) Tasa de deterioro por operador para *OneMax* con $n = 625$ y $\lambda = 15$. Caso FGA $F = 1,5$. Elaboración propia



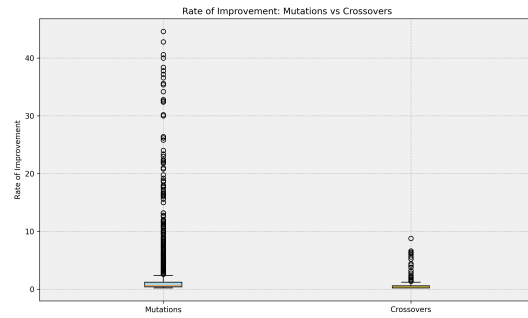
(b) Tasa de deterioro por operador para *OneMax* con $n = 625$ y $\lambda = 15$. Caso FGA $F = 2$. Elaboración propia

Figura 97: Comparativo de tasa de deterioro por operador para *concatenated trap* con $n = 625$ y $\lambda = 15$.

En ambos problemas se presenta el mismo fenómeno. El caso con $F = 1,5$ presenta una gran variación en la aptitud del conjunto de soluciones gracias a una mayor cantidad de ejecuciones de *crossover*. Aunque esta variación está más presente en el caso de los deterioros, esto indica que el algoritmo explora más que su contraparte, aunque por sectores del espacio de búsqueda con menor aptitud, ubicando las soluciones en sectores del espacio de búsqueda con mayor potencial de explotación. Esto se ve reflejado en que la mayor parte de las mutaciones generan una mejora entre 2 y 20 para el caso con $F = 1,5$, mientras que el caso con $F = 2$ mantiene la mayor cantidad de mejoras por mutación cercanas al 0. A pesar de ello, estas mejoras son constantes dado que existe menor perturbación del conjunto de soluciones a través de *crossover*, lo cual puede llevar a obtener una mayor cantidad de soluciones de alta aptitud.

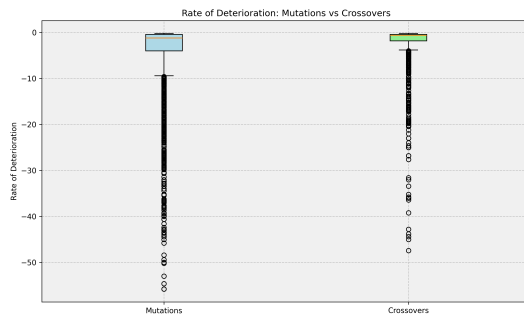


(a) Tasa de mejora por operador para *concatenated trap* con $n = 625$ y $\lambda = 15$. Caso FGA $F = 1,5$. Elaboración propia

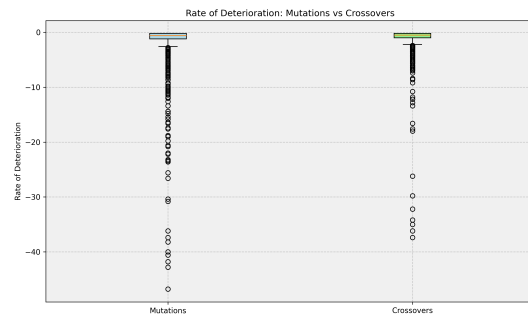


(b) Tasa de mejora por operador para *concatenated trap* con $n = 625$ y $\lambda = 15$. Caso FGA $F = 2$. Elaboración propia

Figura 98: Comparativo de tasa de mejora por operador para *concatenated trap* con $n = 625$ y $\lambda = 15$.



(a) Tasa de deterioro por operador para *concatenated trap* con $n = 625$ y $\lambda = 15$. Caso FGA $F = 1,5$. Elaboración propia



(b) Tasa de deterioro por operador para *concatenated trap* con $n = 625$ y $\lambda = 15$. Caso FGA $F = 2$. Elaboración propia

Figura 99: Comparativo de tasa de deterioro por operador para *concatenated trap* con $n = 625$ y $\lambda = 15$.

CAPÍTULO 5

CONCLUSIONES

En este escrito se presenta un estudio en el área de la optimización combinatorial enfocado en las métricas de desempeño para metaheurísticas poblacionales. Esta memoria propone el diseño y desarrollo de una librería de métricas de desempeño para estas metaheurísticas.

El desarrollo de una librería de métricas de desempeño es de suma importancia para el desarrollo de investigaciones en esta área. Principalmente, tiene por objetivo brindar un método generalizado con el cual obtener un conocimiento completo sobre la ejecución del algoritmo, su proceso de búsqueda y las soluciones que este obtiene.

En este trabajo, se aborda la evaluación de esta librería a través de distintos escenarios. Estos son provistos por el *framework IOHprofiler*. En particular, se desarrolla un caso para un algoritmo genético con parámetros adaptables. Para esto se utiliza *vanilla genetic algorithm* (vGA) como base. Además, se aborda un escenario donde se simula un caso real de estudio utilizando el algoritmo *fast genetic algorithm* (FGA) modificado para el uso del operador *crossover*. Para lograr este objetivo, se ejecuta FGA con su parámetro F en los valores 1,5 y 2, con lo que se puede realizar una comparación directa y evaluar como este cambio afecta al algoritmo y los casos donde cada valor de F presenta mejor rendimiento.

El ambiente en el que se ejecutan estos algoritmos mantiene distintos parámetros que controlan la ejecución, estos son:

- Cantidad de variables (tamaño del problema) $n = \{64, 100, 625, 5625\}$
- Tamaño de población $\lambda = \{5, 15, 50\}$
- Cantidad máxima de evaluaciones: 7500

El caso de vGA se utiliza para demostrar las capacidades de la librería desarrollada, demostrando como un algoritmo puede presentar comportamientos distintos para cada set de problemas y lo que se puede concluir a partir de los resultados generados. Mientras que, el caso de FGA se utiliza a modo de estudio sobre el comportamiento del algoritmo con distintas configuraciones y como la librería de métricas permite realizar un análisis profundo sobre su ejecución.

En cuanto a las métricas desarrolladas, la librería contiene 14 métricas, las cuales pertenecen a las categorías de eficiencia y efectividad. Además, contiene una nueva subcategoría de métricas, las métricas de operadores, las cuales informan sobre el comportamiento de los distintos operadores utilizados por el algoritmo.

Las primeras métricas introducidas en el escrito corresponden a las métricas de cambio de aptitud, las cuales, a través de graficar la aptitud de las soluciones encontradas por el algoritmo, permiten observar el avance del algoritmo en su búsqueda hacia mejores soluciones.

Usando cada solución encontrada por evaluación, se logra visualizar procesos de estancamiento y exploración en la búsqueda. Por otra parte, utilizando la mejor solución encontrada por generación se puede observar la rapidez con la que el algoritmo avanza en su búsqueda de mejores soluciones.

Por otra parte, las métricas de *Rate of fitness change* entregan información sobre el cambio en la aptitud observada para las soluciones encontradas en cada generación. Esto entrega información valiosa sobre la magnitud del cambio en la aptitud, ya sea a la hora de explotar la aptitud de la solución, o en procesos exploratorios. Además, se implementa una variante encargada de calcular la métrica para cada individuo en cada generación de la ejecución del algoritmo, lo que informa sobre individuos que puedan estancarse, sobre todo en problemas con espacios de búsqueda complejos es fácil observar que en grandes poblaciones, existe entre un 5 % a 10 % de los individuos con problemas de estancamiento. Esta implementación permite estudiar a fondo el tamaño de población ideal para un algoritmo en distintos casos.

El conjunto de las métricas de convergencia es de sumo interés en esta área de investigación, por lo que comprende 5 de las métricas implementadas. La tasa de convergencia permite el análisis sobre la rapidez con la que el algoritmo se acerca hacia el óptimo. Este enfoque prioriza el análisis de la explotación del algoritmo, el cual se puede complementar con el error relativo respecto del óptimo, el cual entrega datos sobre el avance acumulado hacia la solución óptima. Estas dos métricas son sumamente relevantes en situaciones propensas al estancamiento, ya que permite observar si es que existe algún avance hacia mejores soluciones, o incluso, si existe algún retroceso en la aptitud de las soluciones encontradas. Es de notar que estas métricas se implementan de forma que puedan utilizarse con un óptimo conocido o bien, en un problema sin información al respecto, de forma que las conclusiones que se pueden obtener de estas dos implementaciones se asemejen en gran medida.

Como complemento a las métricas de convergencia previas y a partir del comportamiento comúnmente observado en los algoritmos ejecutados, donde la aptitud mejora fuertemente en el comienzo de la ejecución para luego estancarse lentamente, se introduce *Geometric rate of fitness change*, métrica que le da mayor prioridad a las mejoras tempranas en la ejecución del algoritmo, reduciendo su valor exponencialmente a medida que las generaciones avanzan. A partir de esto, se puede observar que tan eficiente es la explotación y exploración de un algoritmo. Al obtener una curva con valores mayores a 0, se puede afirmar que el algoritmo está en una búsqueda eficiente, nunca estancándose por completo, mientras que en el caso contrario, se infiere que el algoritmo necesita de una gran exploración del espacio de búsqueda para posicionar las soluciones en sectores donde se pueda explotar la aptitud.

Las últimas dos métricas asociadas a la categoría de eficiencia son *E-value* y *Entropy diversity*. La primera de estas métricas apunta a calcular una proporción de soluciones, encontradas en todo el proceso de búsqueda, que cumplan con una aptitud mínima, la cual se generaliza utilizando un porcentaje arbitrario del óptimo conocido para el problema a resolver. La segunda parte de la proporción que calcula *E-value* es la porción del espacio de búsqueda explorado por el algoritmo. Con esto, la métrica permite comprender la capacidad del algoritmo de encontrar soluciones de una aptitud arbitraria (o mayor a esta) en un tiempo acotado, este

siendo representado por la porción explorada del espacio de búsqueda. Además, al comparar esta métrica en distintas configuraciones de ambiente, tales como el tamaño de población, es posible evaluar que tan eficientemente se explora el espacio de búsqueda, apoyando la elección de parámetros y comprensión del espacio de búsqueda del problema.

Luego, *entropy diversity* es una métrica centrada en estudiar la distribución de múltiples agentes de búsqueda en el espacio de búsqueda. La implementación de esta es sumamente compleja, requiriendo la ejecución de múltiples algoritmos que permiten dilucidar relaciones matemáticas de distancia entre las soluciones encontradas, con lo que se puede formular una aproximación del espacio de búsqueda y dividirlo de forma acorde. La información que entrega esta métrica es de suma importancia, debido a que con esto se puede comprender que tan eficiente es la exploración del espacio de búsqueda y, en algunos casos, como se comporta el algoritmo en momentos donde se explota la aptitud del conjunto de soluciones. Cabe destacar que, para esta métrica, se menciona la complejidad de su implementación. Esto impacta fuertemente en los tiempos de ejecución, lo cual está fuertemente vinculado al tamaño del problema, ya que se utilizan los valores de cada variable para su cálculo. Con esto, es la única métrica que destaca por requerir una cantidad de tiempo significativa para su ejecución, la cual, en un problema de tamaño 5625, llega alrededor de 1616,263 segundos. Por otra parte, el tiempo de ejecución también depende de la complejidad de las relaciones de distancia entre soluciones, las cuales pueden aumentar o disminuir este tiempo.

Las últimas métricas desarrolladas son aquellas asociadas a los operadores del algoritmo. La primera de estas es la tasa de éxito del operador, la cual distingue entre ejecuciones exitosas de un operador basándose en si este logra modificar la aptitud de la solución (ya sea si mejora o empeora). La métrica mide la proporción de ejecuciones del operador que logran ser exitosas. Esta información, en conjunto con otras métricas, puede informar sobre la falta o exceso de ejecuciones de un operador, lo cual puede apoyar la elección de aquellos parámetros que controlen la elección de operadores a ejecutar en cada iteración.

Finalmente, se tienen dos métricas complementarias, la tasa de deterioro y la tasa de mejora. Estas miden el efecto que tiene cada ejecución del operador en términos de la diferencia de aptitud generada. Si esta diferencia es negativa, es decir, empeora la aptitud de la solución, se agrega a los datos de la tasa de deterioro, mientras que en caso contrario, se agrega a la tasa de mejora. Esta métrica permite una visión de los efectos de cada operador en la aptitud de la solución y la exploración del espacio de búsqueda. Tal como se estudia, existen casos donde grandes valores, independientemente del operador donde se encuentren, indican una fuerte exploración del espacio de búsqueda, la cual puede ser poco eficiente en la búsqueda de soluciones de mayor aptitud, pero también puede salvar al algoritmo en casos de estancamiento. Lo contrario se puede concluir de casos donde los valores de cambio observados son menores, estos pueden significar un fuerte estancamiento, o bien, una mejora constante de la aptitud.

En el trabajo de [Doerr et al., 2018], además de entregar el *framework* utilizado para la validación de la solución propuesta, también se tiene el módulo *IOAnalyzer* [Wang et al., 2022], el cual también provee de métricas de desempeño del algoritmo ejecu-

tado de forma generalizada. Sin embargo, este mantiene su enfoque en métricas de tiempo de ejecución. Es posible concluir, a través de la herramienta desarrollada y los resultados que esta provee, que se logra obtener una solución que explora distintas aristas de la ejecución y resultados de un algoritmo metaheurístico, particularmente con un enfoque en las metaheurísticas poblacionales.

A modo de trabajo futuro, se pueden definir distintas avenidas de progreso para esta herramienta. Primero, es importante comprender que la librería aún presenta limitaciones en su enfoque, tal como su dificultad en ejecutarse sobre algoritmos o problemas que modifiquen parámetros, tales como el tamaño de población o el tamaño de problema, en tiempo de ejecución. Esta limitante no permite la implementación de métricas específicas a estos ambientes dinámicos, lo cual puede ser considerado para extender las funcionalidades de este trabajo.

Luego, por parte de las métricas introducidas en este trabajo, las métricas de operadores, es importante destacarlas como un primer acercamiento a estudiar estos componentes del algoritmo de forma específica y granular. Es por esto que es importante extender el conocimiento que se puede obtener de los operadores a través de nuevos aspectos a monitorear, además de adaptar el uso de estas métricas a operadores que puedan entregar estos nuevos aspectos.

Otro aspecto a considerar en el futuro es el de la métrica *entropy diversity*, la cual, actualmente utiliza una división de 8 sectores para el espacio de búsqueda aproximado. Sin embargo, esto podría ser expandido para tener una mayor comprensión del espacio de búsqueda y desarrollar criterios para definir la cantidad de espacios que se utilizan. Por otra parte, esto debe ir acompañado de una modificación en el algoritmo con el que se reduce la dimensionalidad de las distancias entre soluciones, para así ser representable en el espacio de búsqueda definido.

Para finalizar, se propone también, extender la librería para abordar una mayor gama de metaheurísticas y las métricas que estas puedan presentar para sus casos específicos. Además, la extensión de la librería puede también abordar mecanismos para ejecutar métricas de efectividad (las cuales requieren, como mínimo, múltiples ejecuciones del algoritmo en sí) y normalización de estas métricas a partir de los factores externos al ambiente de ejecución, tales como el *hardware* en el que se ejecuta y las capacidades de este.

REFERENCIAS BIBLIOGRÁFICAS

- [Abdel-Basset *et al.*, 2022] Abdel-Basset, M., Mohamed, R., y Elkomy, O. M. (2022). Knapsack cipher-based metaheuristic optimization algorithms for cryptanalysis in blockchain-enabled internet of things systems. *Ad Hoc Networks*, 128:102798.
- [Adetunji *et al.*, 2021] Adetunji, K. E., Hofsajer, I. W., Abu-Mahfouz, A. M., y Cheng, L. (2021). A review of metaheuristic techniques for optimal integration of electrical units in distribution networks. *IEEE Access*, 9:5046–5068.
- [Aguilar-Fuster y Rubio-Loyola, 2021] Aguilar-Fuster, C. y Rubio-Loyola, J. (2021). A novel evaluation function for higher acceptance rates and more profitable metaheuristic-based online virtual network embedding. *Computer Networks*, 195:108191.
- [Altenberg, 1996] Altenberg, L. (1996). B2. 7.2 nk fitness landscapes. *Evolution*, 2:1–11.
- [Attiya *et al.*, 2020] Attiya, Ibrahim and Abd Elaziz, Mohamed and Xiong, Shengwu and others (2020). Job scheduling in cloud computing using a modified harris hawks optimization and simulated annealing algorithm. *Computational intelligence and neuroscience*, 2020.
- [Auger, 2009] Auger, A. (2009). Benchmarking the (1+1) evolution strategy with one-fifth success rule on the bbob-2009 function testbed. En *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, GECCO '09, p. 2447–2452, New York, NY, USA. Association for Computing Machinery.
- [Barbarosoglu y Ozgur, 1999] Barbarosoglu, G. y Ozgur, D. (1999). A tabu search algorithm for the vehicle routing problem. *Computers Operations Research*, 26(3):255–270.
- [Barr *et al.*, 1995] Barr, R. S., Golden, B. L., Kelly, J. P., Resende, M. G., y Stewart, W. R. (1995). Designing and reporting on computational experiments with heuristic methods. *Journal of heuristics*, 1:9–32.
- [Bartz-Beielstein, 2005] Bartz-Beielstein, T. (2005). New experimentalism applied to evolutionary computation.
- [Beheshti y Shamsuddin, 2013] Beheshti, Z. y Shamsuddin, S. M. H. (2013). A review of population-based meta-heuristic algorithms. *Int. j. adv. soft comput. appl*, 5(1):1–35.
- [Bezzel, 1848] Bezzel, M. (1848). Proposal of 8-queens problem. *Berliner Schachzeitung*, 3(363):1848.
- [Blum, 2005] Blum, C. (2005). Ant colony optimization: Introduction and recent trends. *Physics of Life Reviews*, 2(4):353–373.

- [Burkard, 1975] Burkard, R. E. (1975). Heuristische verfahren zur lösung quadratischer zuordnungsprobleme. *Zeitschrift für Operations Research*, 19:183–193.
- [Chacón Sartori et al., 2023] Chacón Sartori, C., Blum, C., y Ochoa, G. (2023). Stnweb: A new visualization tool for analyzing optimization algorithms. *Software Impacts*, 17:100558.
- [Chowla y Ryser, 1950] Chowla, S. y Ryser, H. J. (1950). Combinatorial problems. *Canadian Journal of Mathematics*, 2:93–99.
- [Cicirello y Cernera, 2013] Cicirello, V. A. y Cernera, R. (2013). Profiling the distance characteristics of mutation operators for permutation-based genetic algorithms. En *Proceedings of the twenty-sixth international florida artificial intelligence research society conference*, pp. 46–51.
- [Clerc y Kennedy, 2002] Clerc, M. y Kennedy, J. (2002). The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, 6(1):58–73.
- [Dantzig, 1990] Dantzig, G. B. (1990). Origins of the simplex method. En *A history of scientific computing*, pp. 141–151.
- [Deb y Goldberg, 1993] Deb, K. y Goldberg, D. E. (1993). Analyzing deception in trap functions. En WHITLEY, L. D., editor, *Foundations of Genetic Algorithms*, volumen 2 de *Foundations of Genetic Algorithms*, pp. 93–108. Elsevier.
- [Dhivyaprabha et al., 2018] Dhivyaprabha, T., Subashini, P., y Krishnaveni, M. (2018). Synergistic fibroblast optimization: a novel nature-inspired computing algorithm. *Frontiers of Information Technology & Electronic Engineering*, 19:815–833.
- [Doerr et al., 2015] Doerr, B., Doerr, C., y Ebel, F. (2015). From black-box complexity to designing new genetic algorithms. *Theoretical Computer Science*, 567:87–104.
- [Doerr et al., 2017] Doerr, B., Le, H. P., Makhmara, R., y Nguyen, T. D. (2017). Fast genetic algorithms. En *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '17*, p. 777–784, New York, NY, USA. Association for Computing Machinery.
- [Doerr et al., 2021] Doerr, B., Witt, C., y Yang, J. (2021). Runtime analysis for self-adaptive mutation rates. *Algorithmica*, 83(4):1012–1053.
- [Doerr et al., 2018] Doerr, C., Wang, H., Ye, F., van Rijn, S., y Bäck, T. (2018). Iohprofiler: A benchmarking and profiling tool for iterative optimization heuristics. *CoRR*, abs/1810.05281.
- [Doerr et al., 2020] Doerr, C., Ye, F., Horesh, N., Wang, H., Shir, O. M., y Bäck, T. (2020). Benchmarking discrete optimization heuristics with iohprofiler. *Applied Soft Computing*, 88:106027.
- [Dorigo, 1991] Dorigo, M. (1991). Positive feedback as a search strategy. *Technical report*, pp. 91–16.

- [Dorigo, 1992] Dorigo, M. (1992). Optimization, learning and natural algorithms. *Ph. D. Thesis, Politecnico di Milano*.
- [Dorigo et al., 2006] Dorigo, M., Birattari, M., y Stutzle, T. (2006). Ant colony optimization. *IEEE Computational Intelligence Magazine*, 1(4):28–39.
- [Dreo, 2007] Dreo, J. (2007). Dreaming of metaheuristics.
- [El-Ghandour y Elbeltagi, 2018] El-Ghandour, H. A. y Elbeltagi, E. (2018). Comparison of five evolutionary algorithms for optimization of water distribution networks. *Journal of Computing in Civil Engineering*, 32(1):04017066.
- [Elshaer y Awad, 2020] Elshaer, R. y Awad, H. (2020). A taxonomic review of metaheuristic algorithms for solving the vehicle routing problem and its variants. *Computers Industrial Engineering*, 140:106242.
- [Ferrer et al., 2016] Ferrer, J. M., Ortuño, M. T., y Tirado, G. (2016). A grasp metaheuristic for humanitarian aid distribution. *Journal of Heuristics*, 22:55–87.
- [Fonseca y Fleming, 1996] Fonseca, C. M. y Fleming, P. J. (1996). On the performance assessment and comparison of stochastic multiobjective optimizers. En Voigt, H.-M., Ebeling, W., Rechenberg, I., y Schwefel, H.-P., editores, *Parallel Problem Solving from Nature — PPSN IV*, pp. 584–593, Berlin, Heidelberg. Springer Berlin Heidelberg.
- [Glover, 1986] Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers Operations Research*, 13(5):533–549. Applications of Integer Programming.
- [Golden et al., 1998] Golden, B. L., Wasil, E. A., y Kelly, J. P. (1998). 2 the impact of metaheuristics on solving the vehicle routing problem: Algorithms, problem sets, and computational results 2.1 introduction.
- [Gong y Yang, 2001] Gong, M. y Yang, Y.-H. (2001). Multi-resolution stereo matching using genetic algorithm. En *Proceedings IEEE Workshop on Stereo and Multi-Baseline Vision (SMBV 2001)*, pp. 21–29.
- [Griffis et al., 2012] Griffis, S. E., Bell, J. E., y Closs, D. J. (2012). Metaheuristics in logistics and supply chain management. *Journal of Business Logistics*, 33(2):90–106.
- [Halim et al., 2021] Halim, A. H., Ismail, I., y Das, S. (2021). Performance assessment of the metaheuristic optimization algorithms: an exhaustive review. *Artificial Intelligence Review*, 54:2323–2409.
- [Hare y Sagastizábal, 2006] Hare, W. y Sagastizábal, C. (2006). Benchmark of some nonsmooth optimization solvers for computing nonconvex proximal points.
- [He y Lin, 2016] He, J. y Lin, G. (2016). Average convergence rate of evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 20(2):316–321.

- [Hoffman *et al.*, 2013] Hoffman, Karla L and Padberg, Manfred and Rinaldi, Giovanni and others (2013). Traveling salesman problem. *Encyclopedia of operations research and management science*, 1:1573–1578.
- [Holland, 1975] Holland, J. (1975). Adaptation in neural and artificial system. *Ann Arbor, Univeristy of Michigan Press*.
- [Hoos, 1999] Hoos, H. (1999). *Stochastic local search-methods, models, applications*. los Press.
- [Ivković *et al.*, 2016] Ivković, N., Jakobović, D., y Golub, M. (2016). Measuring performance of optimization algorithms in evolutionary computation. *International Journal of Machine Learning and Computing*, 6(3):167–171.
- [Johnson *et al.*, 1989] Johnson, D. S., Aragon, C. R., McGeoch, L. A., y Schevon, C. (1989). Optimization by simulated annealing: An experimental evaluation; part i, graph partitioning. *Operations research*, 37(6):865–892.
- [Johnson *et al.*, 1988] Johnson, D. S., Papadimitriou, C. H., y Yannakakis, M. (1988). How easy is local search? *Journal of Computer and System Sciences*, 37(1):79–100.
- [Jozefowicz *et al.*, 2008] Jozefowicz, N., Semet, F., y Talbi, E.-G. (2008). Multi-objective vehicle routing problems. *European journal of operational research*, 189(2):293–309.
- [Karaboga *et al.*, 2005] Karaboga and Dervis and others (2005). An idea based on honey bee swarm for numerical optimization. Technical report, Technical report-tr06, Erciyes university, engineering faculty, computer
- [Karaboga y Akay, 2009] Karaboga, D. y Akay, B. (2009). A comparative study of artificial bee colony algorithm. *Applied Mathematics and Computation*, 214(1):108–132.
- [Katoch *et al.*, 2021] Katoch, S., Chauhan, S. S., y Kumar, V. (2021). A review on genetic algorithm: past, present, and future. *Multimedia Tools and Applications*, 80(5):8091–8126.
- [Katona *et al.*, 2019] Katona, G., Lénárt, B., y Juhasz, J. (2019). Parallel ant colony algorithm for shortest path problem. *Periodica Polytechnica Civil Engineering*, 63.
- [Kauffman y Levin, 1987] Kauffman, S. y Levin, S. (1987). Towards a general theory of adaptive walks on rugged landscapes. *Journal of theoretical Biology*, 128(1):11–45.
- [Kearfott, 1996] Kearfott, R. B. (1996). *A Review of Techniques in the Verified Solution of Constrained Global Optimization Problems*, pp. 23–59. Springer US, Boston, MA.
- [Kennedy y Eberhart, 1995] Kennedy, J. y Eberhart, R. (1995). Particle swarm optimization. En *Proceedings of ICNN'95 - International Conference on Neural Networks*, volumen 4, pp. 1942–1948 vol.4.

- [Krink et al., 2002] Krink, T., Vesterstrom, J., y Riget, J. (2002). Particle swarm optimisation with spatial particle extension. En *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*, volumen 2, pp. 1474–1479 vol.2.
- [Kruisselbrink et al., 2011] Kruisselbrink, J. W., Li, R., Reehuis, E., Eggermont, J., y Bäck, T. (2011). On the log-normal self-adaptation of the mutation rate in binary search spaces. En *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, GECCO '11*, p. 893–900, New York, NY, USA. Association for Computing Machinery.
- [Kruskal y Wish, 1978] Kruskal, J. B. y Wish, M. (1978). *Multidimensional scaling*. Número 11. Sage.
- [Kumar, 1992] Kumar, V. (1992). Algorithms for constraint-satisfaction problems: A survey. *AI Magazine*, 13(1):32.
- [Lambora et al., 2019] Lambora, A., Gupta, K., y Chopra, K. (2019). Genetic algorithm- a literature review. En *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, pp. 380–384.
- [Leite et al., 2014] Leite, A. R., Enembreck, F., y Barthès, J.-P. A. (2014). Distributed constraint optimization problems: Review and perspectives. *Expert Systems with Applications*, 41(11):5139–5157.
- [Li y Zhao, 2005] Li, X. y Zhao, Z. (2005). Relative error measures for evaluation of estimation algorithms. En *2005 7th International Conference on Information Fusion*, volumen 1, pp. 8 pp.–.
- [Liesegang y Schirmer, 1975] Liesegang, G. y Schirmer, A. (1975). Heuristische verfahren zur maschinenbelegungsplanung bei reihenfertigung. *Zeitschrift für Operations Research*, 19:195–211.
- [Lo et al., 2010] Lo, M.-H., Liang, Y.-C., y Hsieh, J.-C. (2010). A modified variable neighborhood search algorithm for orienteering problems. En *The 40th International Conference on Computers Industrial Engineering*, pp. 1–6.
- [Madureira et al., 2002] Madureira, A., Ramos, C., y do Carmo Silva, S. (2002). A coordination mechanism for real world scheduling problems using genetic algorithms. En *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*, volumen 1, pp. 175–180 vol.1.
- [Mead, 1992] Mead, A. (1992). Review of the development of multidimensional scaling methods. *Journal of the Royal Statistical Society: Series D (The Statistician)*, 41(1):27–39.
- [Memari et al., 2017] Memari, Ashkan and Ahmad, Robiah and others (2017). Metaheuristic algorithms: guidelines for implementation. *Journal of Soft Computing and Decision Support Systems*, 4(6):1–6.

- [Metcalf y Casey, 2016] Metcalf, L. y Casey, W. (2016). Chapter 2 - metrics, similarity, and sets. En Metcalf, L. y Casey, W., editores, *Cybersecurity and Applied Mathematics*, pp. 3–22. Syngress, Boston.
- [Monge, 1781] Monge, G. (1781). Mémoire sur la théorie des déblais et des remblais. *Mem. Math. Phys. Acad. Royale Sci.*, pp. 666–704.
- [Mora-Meliá et al., 2015] Mora-Meliá, D., Iglesias-Rey, P. L., Martínez-Solano, F. J., y Ballesteros-Perez, P. (2015). Efficiency of evolutionary algorithms in water network pipe sizing. *Water resources management*, 29:4817–4831.
- [Morris y Showalter, 1983] Morris, J. G. y Showalter, M. J. (1983). Simple approaches to shift, days-off and tour scheduling problems. *Management Science*, 29(8):942–950.
- [Müller-Merbach, 1981] Müller-Merbach, H. (1981). Heuristics and their design: a survey. *European Journal of Operational Research*, 8(1):1–23.
- [Nebro et al., 2008] Nebro, A. J., Durillo, J. J., Coello Coello, C. A., Luna, F., y Alba, E. (2008). A study of convergence speed in multi-objective metaheuristics. En Rudolph, G., Jansen, T., Beume, N., Lucas, S., y Poloni, C., editores, *Parallel Problem Solving from Nature – PPSN X*, pp. 763–772, Berlin, Heidelberg. Springer Berlin Heidelberg.
- [Nikfarjam et al., 2021] Nikfarjam, A., Bossek, J., Neumann, A., y Neumann, F. (2021). Entropy-based evolutionary diversity optimisation for the traveling salesperson problem. En *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 600–608.
- [Osuna-Enciso et al., 2022] Osuna-Enciso, V., Cuevas, E., y Morales Castañeda, B. (2022). A diversity metric for population-based metaheuristic algorithms. *Information Sciences*, 586:192–208.
- [Packebusch y Mertens, 2016] Packebusch, T. y Mertens, S. (2016). Low autocorrelation binary sequences. *Journal of Physics A: Mathematical and Theoretical*, 49(16):165001.
- [Papadimitriou y Steiglitz, 2013] Papadimitriou, C. H. y Steiglitz, K. (2013). *Combinatorial optimization: algorithms and complexity*. Courier Corporation.
- [Pereira et al., 2010] Pereira, R., Cummiskey, K., y Kincaid, R. (2010). Office space allocation optimization. En *2010 IEEE Systems and Information Engineering Design Symposium*, pp. 112–117.
- [Pham et al., 2006] Pham, D., Ghanbarzadeh, A., Koç, E., Otri, S., Rahim, S., y Zaidi, M. (2006). - the bees algorithm — a novel tool for complex optimisation problems. En Pham, D., Eldukhri, E., y Soroka, A., editores, *Intelligent Production Machines and Systems*, pp. 454–459. Elsevier Science Ltd, Oxford.
- [Pires et al., 2013] Pires, E. J. S., Machado, J. A. T., y de Moura Oliveira, P. B. (2013). Entropy diversity in multi-objective particle swarm optimization. *Entropy*, 15(12):5475–5491.

- [Rodríguez Rueda *et al.*, 2009] Rodríguez Rueda, D., Cotta, C., y Fernández, A. J. (2009). Finding balanced incomplete block designs with metaheuristics. En Cotta, C. y Cowling, P., editores, *Evolutionary Computation in Combinatorial Optimization*, pp. 156–167, Berlin, Heidelberg. Springer Berlin Heidelberg.
- [Ronald, 1998] Ronald, S. (1998). More distance functions for order-based encodings. En *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360)*, pp. 558–563.
- [Schrijver, 2005] Schrijver, A. (2005). On the history of combinatorial optimization (till 1960). En Aardal, K., Nemhauser, G., y Weismantel, R., editores, *Discrete Optimization*, volumen 12 de *Handbooks in Operations Research and Management Science*, pp. 1–68. Elsevier.
- [Shi y Eberhart, 1998] Shi, Y. y Eberhart, R. (1998). A modified particle swarm optimizer. En *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360)*, pp. 69–73.
- [Silberholz y Golden, 2010] Silberholz, J. y Golden, B. (2010). *Comparison of Metaheuristics*, pp. 625–640. Springer US, Boston, MA.
- [Singh *et al.*, 2022] Singh, R. M., Awasthi, L. K., y Sikka, G. (2022). Towards metaheuristic scheduling techniques in cloud and fog: An extensive taxonomic review. *ACM Comput. Surv.*, 55(3).
- [Thevenot, 2022] Thevenot, A. (2022). Particle swarm optimization visually explained.
- [Voß *et al.*, 2012] Voß, S., Martello, S., Osman, I. H., y Roucairol, C. (2012). Meta-heuristics: Advances and trends in local search paradigms for optimization.
- [Wagner y Fischer, 1974] Wagner, R. A. y Fischer, M. J. (1974). The string-to-string correction problem. *J. ACM*, 21(1):168–173.
- [Wang *et al.*, 2022] Wang, H., Vermetten, D., Ye, F., Doerr, C., y Bäck, T. (2022). Iohalyzer: Detailed performance analyses for iterative optimization heuristics. *ACM Transactions on Evolutionary Learning and Optimization*, 2(1).
- [Wang y Boyer, 2013] Wang, Q. y Boyer, K. L. (2013). Feature learning by multidimensional scaling and its applications in object recognition. En *SIBGRAPI 2013 - Technical Papers*, Arequipa, Peru.
- [Weise *et al.*, 2008] Weise, T., Niemczyk, S., Skubch, H., Reichle, R., y Geihs, K. (2008). A tunable model for multi-objective, epistatic, rugged, and neutral fitness landscapes. En *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pp. 795–802.
- [Yan *et al.*, 2023] Yan, Y., Liu, Q., y Li, Y. (2023). Paradox-free analysis for comparing the performance of optimization algorithms. *IEEE Transactions on Evolutionary Computation*, 27(5):1275–1287.

- [Yang *et al.*, 2017] Yang, L., Sun, X., Zhu, A., y Chi, T. (2017). A multiple ant colony optimization algorithm for indoor room optimal spatial allocation. *ISPRS International Journal of Geo-Information*, 6(6).
- [Yang, 2005] Yang, X.-S. (2005). Engineering optimizations via nature-inspired virtual bee algorithms. En Mira, J. y Álvarez, J. R., editores, *Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach*, pp. 317–323, Berlin, Heidelberg. Springer Berlin Heidelberg.
- [Ye *et al.*, 2019] Ye, F., Doerr, C., y Bäck, T. (2019). Interpolating local and global search by controlling the variance of standard bit mutation. En *2019 IEEE Congress on Evolutionary Computation (CEC)*, pp. 2292–2299.
- [Yu *et al.*, 2005] Yu, H.-J., Zhang, L.-Q., Chen, D.-Z., y Hu, S.-X. (2005). Adaptive particle swarm optimization algorithm based on feedback mechanism. *Zhejiang Daxue Xuebao(Gongxue Ban)/Journal of Zhejiang University(Engineering Science)*, 39(9):1286–1291.