PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

Algoritmo bioinspirado Lobo Gris (Grey Wolf) para la optimización de índices en bases de datos relacionales

Tesis para obtener el título profesional de Ingeniero Informático

AUTOR:

Fernando Guillermo Verástegui Sánchez

ASESOR:

Rony Cueva Moscoso

Lima, Marzo, 2024

Informe de Similitud

Informe de Similitud

Yo, ...Rony Cueva Moscoso.....,

docente de la Facultad de .....Ciencias e Ingeniería..... de la Pontificia Universidad Católica del Perú, asesor(a) de la tesis/el trabajo de investigación titulado

.....Algoritmo bioinspirado Lobo Gris (Grey Wolf) para la optimización de índices en bases de datos relacionales.....,

del/de la autor(a)/ de los(as) autores(as)

.....Fernando Guillermo Verástegui Sánchez.....,

.....,

.....,

dejo constancia de lo siguiente:

- El mencionado documento tiene un índice de puntuación de similitud de 20%. Así lo consigna el reporte de similitud emitido por el software *Turnitin* el 22/03/2024.
- He revisado con detalle dicho reporte y la Tesis o Trabajo de Suficiencia Profesional, y no se advierte indicios de plagio.
- Las citas a otros autores y sus respectivas referencias cumplen con las pautas académicas.

Lugar y fecha: ...Lima, 22 de marzo del 2024.....

Apellidos y nombres del asesor / de la asesora: Cueva Moscoso Rony	
DNI: 09942265	Firma
ORCID: 0000-0003-4861-571X	

Resumen

Dentro del mundo empresarial actual, los datos cobran una importancia crucial para el desarrollo de una compañía, ya sea para análisis, seguimiento comercial, estrategias de negocios, entre otros. Por lo que, su almacenamiento y extracción son una parte importante para su uso. Las bases de datos sirven para almacenar esta información, para que luego puedan ser consultadas. Estas pueden ser tanto relacionales como no relacionales, siendo la primera en donde los datos que se almacenan están relacionados entre ellos. La información se organiza mediante tablas, que incluyen columnas y filas, pudiendo obtener acceso a estos datos de diferentes maneras (AWS, n.d.).

Para los administradores de base de datos (DBA) e investigadores la mejora del rendimiento de las bases de datos ha sido un reto persistente a lo largo de los años. Este desafío está fuertemente relacionado con la forma de organización lógica de los registros y, especialmente, a la rapidez con la que se accede y procesa esos registros. En ese sentido, los índices ejercen como un modo de acceso más rápido a la información. Siendo un caso de ejemplo, cuando los datos son solicitados, el sistema de gestión de base de datos primero verifica si existe un índice y su estructura, la cual está ordenada y contiene la dirección física del dato, permitiendo así recuperar la información directamente en el disco duro, simplificando el trabajo de búsqueda (Pedrozo & Vaz, 2014).

Si bien los índices sirven para acelerar la recuperación de los datos, uno mal diseñado deteriora el rendimiento general. Esto se debe a los diferentes factores que influyen en su creación, como lo son: El tipo de dato, la estructura de la tabla, el número de veces que se buscó un campo específico, la frecuencia de ciertas consultas, la frecuencia de valores distintos en una columna, la carga de trabajo, el número de operaciones de lectura y escritura, entre otros (Naik, 2018).

En consecuencia, el propósito de este proyecto consiste en mejorar la eficiencia del tiempo de respuesta en las consultas mediante el uso de índices, brindando la mejor opción para la creación de estos, sobre las tablas dentro un sistema de gestión de bases de datos relacionales (RDBMS). Para este proyecto se plantea utilizar un algoritmo metaheurístico aplicado al problema de selección de índices (ISP), el cual consiste en, dado una base de datos y un conjunto de consultas, seleccionar automáticamente un conjunto apropiado de índices (Chaudhuri, Datar, and Narasayya 2004). Este problema es considerado un “NP-Hard Problem”, la elección de utilizar un algoritmo metaheurístico, en combinación con la amplia gama de variables que pueden influir, se revela como una opción óptima en comparación con otros tipos de algoritmos

Dedicatoria

A mi abuelita Pochita, quien me ha guiado toda mi vida.

Tema FCI

FACULTAD DE CIENCIAS E INGENIERÍA

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

TEMA DE TESIS

TEMA	: Algoritmo bioinspirado Lobo Gris (Grey Wolf) para la optimización de índices en bases de datos relacionales
ÁREA	: Ciencia de la Computación
ASESOR	: Msc. Rony Cueva Moscoso
ALUMNO(S)	: Fernando Guillermo Verástegui Sánchez - 20170824
FECHA	: 28/01/2023

DESCRIPCIÓN Y OBJETIVOS:

En el competitivo mundo actual, las empresas ponen más énfasis en la información considerándose como el activo más valioso de la organización. El beneficio de los datos yace en el hecho de que tan pronto se pueden transformar en información útil y en conocimiento estratégico (Wijesiriwardana & Firdhous, 2019). Un sistema de gestión de bases de datos es el componente principal de cualquier sistema de información, de manera que su almacenamiento y extracción son una parte importante para su uso (Wijesiriwardana & Firdhous, 2019). Para los administradores de base de datos e investigadores, la mejora del rendimiento de las bases de datos ha sido un reto persistente a lo largo de los años (Pedrozo & Vaz, 2014). En ese sentido, los índices de las tablas son una estructura física que se crea en la misma tabla para potenciar la recuperación de datos durante la ejecución de la transacción (Naik, 2018). Si bien los índices sirven para acelerar la recuperación de los datos, uno mal diseñado deteriora el rendimiento general (Naik, 2018). Encontrar una configuración de índice óptima para obtener el costo de ejecución mínimo, equilibrando tanto la búsqueda como las consultas de mantenimiento sobre los datos de la carga de trabajo de la aplicación, se conoce formalmente como el problema de selección de índice (Kovačević & Filipič, 2006).

A lo largo de los años numerosas investigaciones han intentado dar respuesta a dicho problema, y si bien cada una propone una solución novedosa, no han logrado satisfacer las necesidades que tiene el usuario. Por lo tanto, en el presente proyecto de fin de carrera, se busca optimizar el consumo de recursos, considerando tanto el tiempo de respuesta como el espacio en disco, de las consultas mediante el uso de índices, brindando la mejor opción para la creación de estos, sobre las tablas dentro un sistema de gestión de bases de datos relacionales.

Con el fin de resolver este problema, se plantea utilizar un algoritmo metaheurístico adaptado a la selección de índices en el entorno de una base de datos relacional, se revisará la literatura para identificar las principales variables y restricciones a tener en cuenta. Asimismo, se buscará acerca del desempeño de los algoritmos más empleados

y de diversas herramientas propietarias. Utilizando las variables y restricciones identificadas en la revisión de la literatura, se planteará el diseño de un algoritmo.

En ese mismo sentido, se implementará dicho algoritmo, el cual permitirá conectarse a una base de datos relacional y recibir un conjunto de consultas que se deseen mejorar su eficiencia. De esta manera, la herramienta brindará las alternativas encontradas para una óptima configuración del índice.

Finalmente, se revisará la literatura para poder identificar la solución más utilizada con la finalidad de ratificar los resultados. Se implementará dicha solución para la posterior comparación del rendimiento de ambos algoritmos mediante la experimentación numérica.

Objetivo General:

Implementar un algoritmo lobo gris que brinde solución al problema de selección de índices lógicos en bases de datos relacionales.

Objetivos específicos:

1. Definir las variables y restricciones necesarias del problema para el establecimiento de la función objetivo.
2. Adaptar un algoritmo lobo gris que brinde una óptima selección de índices.
3. Comparar el desempeño del algoritmo lobo gris propuesto contra la solución más utilizada, encontrada en el estado del arte para el problema de selección de índices con el fin de validar los resultados.

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
Departamento de Ingeniería

Dr. LUIS FLORES GARCÍA
Director de Carrera
Ingeniería Informática

Índice general

Informe de Similitud 2
Resumen 3
Dedicatoria 4
Tema FCI 5
Índice de figuras 12
Índice de tablas 14
Capítulo 1. Aspectos Generales 15
1.1 Problemática 15
1.1.1 Árbol de problemas 15
1.1.2 Descripción 15
1.1.3 Elección del problema 20
1.2 Objetivos 20
1.2.1 Objetivo general 20
1.2.2 Objetivos específicos 20
1.2.3 Resultados esperados 20
1.2.4 Asignación de objetivos, logros y verificación 21
1.3 Métodos y procedimientos 22
1.3.1 Pseudocódigo 23
1.3.2 PseInt 23
1.3.3 Algoritmo Lobo Gris 24
1.3.4 IntelliJ IDEA 24
1.3.5 Java 24
1.3.6 Git 25
1.3.7 Kanban 25
1.3.8 RStudio 25
1.3.9 Pruebas estadísticas no paramétricas 25
Capítulo 2. Marco Legal y Conceptual 27
2.1 Introducción 27
2.2 Desarrollo del marco conceptual 27
2.2.1 Base de datos relacional 27
2.2.2 Sistema de administración de base de datos relacional 27
2.2.3 Optimización de base de datos 28
2.2.4 Consulta 28
2.2.5 Índice 28

2.2.6 Tiempo de respuesta 29
2.2.7 Problema de selección de índices 29
2.2.8 Algoritmo metaheurístico 30
2.2.9 Algoritmo genético 30
Capítulo 3. Estado del Arte 32
3.1 Introducción 32
3.2 Objetivos de la revisión 32
3.3 Preguntas de revisión 32
3.4 Métodos de indagación 33
3.4.1 Motores de búsqueda 33
3.4.2 Cadenas de búsqueda a emplear 33
3.4.3 Documentos encontrados 34
3.4.4 Criterios de inclusión y exclusión 35
3.5 Formulario de extracción 36
3.6 Resultados de la revisión 37
3.6.1 Respuesta a ¿Qué tipo y cómo funcionan los algoritmos metaheurísticos implementados para atacar el problema de selección de índices? 37
3.6.2 Respuesta a ¿Cuáles son las variables y/o restricciones más importantes en la selección de índices y de qué manera afectan a los algoritmos para la optimización? 38
3.6.3 Respuesta a ¿Cómo se calcula la eficacia y la eficiencia de los algoritmos implementados? 40
3.7 Conclusiones 40
Capítulo 4. Definición de Parámetros, Restricciones y Función Objetivo 42
4.1 Introducción 42
4.2 Resultados obtenidos 42
4.2.1 Establecimiento de variables y restricciones del problema 42
4.2.1.1 Variables identificadas 42
4.2.1.2 Restricciones identificadas 43
4.2.1.3 Enfoques, Medios de Confirmación y Marcadores 44
4.2.2 Elaboración de la Función Objetivo 44
4.2.2.1 Descripción 45
4.2.2.2 Enfoques, Medios de Confirmación y Marcadores 46
4.3 Análisis y reflexión de los resultados 46
Capítulo 5. Adecuación del algoritmo Lobo Gris 48
5.1 Introducción 48
5.2 Resultados obtenidos 48

5.2.1 Definición de las estructuras de datos 48
5.2.1.1 Representación de una tabla 48
5.2.1.2 Representación de una columna 48
5.2.1.3 Representación de un lobo 49
5.2.2 Diseño del algoritmo Lobo Gris 49
5.2.2.1 Algoritmo Lobo Gris 50
5.2.2.2 Generación de la población inicial 50
5.2.2.3 Método de actualizar posición 51
5.2.2.4 Control de aberraciones 52
5.2.2.5 Cálculo de la función fitness 54
5.2.2.6 Obtención de las tres mejores soluciones 54
5.2.2.7 Método de selección de columnas 54
5.2.3 Enfoques, Medios de Confirmación y Marcadores 55
5.3 Análisis y reflexión de los resultados 55
Capítulo 6. Codificación del algoritmo Lobo Gris 57
6.1 Introducción 57
6.2 Resultados obtenidos 57
6.2.1 Codificación del algoritmo lobo gris adaptado al problema de selección de índices 57
6.2.1.1 Archivos de entrada 57
6.2.1.2 Método principal del Algoritmo Lobo Gris 60
6.2.1.3 Generación de la población inicial 61
6.2.1.4 Actualizar posición de la población 62
6.2.1.5 Control de aberraciones 63
6.2.1.6 Cálculo de la función fitness 65
6.2.1.7 Obtención de las tres mejores soluciones 70
6.2.1.8 Selección de columnas 70
6.2.1.9 Enfoques, Medios de Confirmación y Marcadores 71
6.2.2 Calibración de variables 71
6.2.2.1 Descripción 71
6.2.2.2 Máximo número de iteraciones 72
6.2.2.3 Máximo número de generaciones sin mejora 72
6.2.2.4 Porcentaje de aceptación para elección de columnas 73
6.2.2.5 Enfoques, Medios de Confirmación y Marcadores 73
6.3 Análisis y reflexión de los resultados 73
Capítulo 7. Adaptación del algoritmo Genético 75

7.1 Introducción 75
7.2 Resultados obtenidos 75
7.2.1 Definición de las estructuras de datos 75
7.2.1.1 Representación de una tabla 75
7.2.1.2 Representación de una columna o gen 75
7.2.1.3 Representación de un cromosoma 76
7.2.2 Diseño del algoritmo Genético 77
7.2.2.1 Algoritmo Genético 77
7.2.2.2 Cálculo de espacio de tablas en querys 78
7.2.2.3 Generación de la población inicial 78
7.2.2.4 Obtención de la mejor solución 79
7.2.2.5 Control de aberraciones 79
7.2.2.6 Generación de una población 80
7.2.2.7 Cálculo de la función fitness 81
7.3 Análisis y reflexión de los resultados 81
Capítulo 8. Codificación del algoritmo Genético 82
8.1 Introducción 82
8.2 Resultados obtenidos 82
8.2.1 Codificación del algoritmo genético seleccionado del estado del arte 82
8.2.1.1 Archivos de entrada 82
8.2.1.2 Método principal del Algoritmo Genético 85
8.2.1.3 Generación de la población inicial 86
8.2.1.4 Crear una nueva población 87
8.2.1.5 Control de aberraciones 88
8.2.1.6 Obtención de la mejor solución 90
8.2.1.7 Cálculo de la función fitness 90
8.2.2 Enfoques, Medios de Confirmación y Marcadores 93
8.3 Análisis y reflexión de los resultados 93
Capítulo 9. Experimentación numérica 94
9.1 Introducción 94
9.2 Resultados obtenidos 94
9.2.1 Informe de experimentación numérica 94
9.2.1.1 Recolección de datos 94
9.2.1.2 Prueba de Shapiro-Wilk 95
9.2.1.3 Prueba de Bartlett 95
9.2.1.4 Prueba Z 96

9.2.2 Enfoques, Medios de Confirmación y Marcadores 96
9.3 Análisis y reflexión de los resultados 96
Capítulo 10. Conclusiones y trabajos futuros 97
10.1 Conclusiones 97
10.2 Trabajos futuros 97
Referencias 98
Anexos 102
Anexo A: Formulario de extracción 102
Anexo B: Plan de proyecto 103
Anexo C: Documento de validación de las variables y restricciones por un especialista en bases de datos 115
Anexo D: Registro de las pruebas ejecutadas en relación a la función objetivo 116
Anexo E: Certificación de validación de la función objetivo, autenticado por un especialista en bases de datos y algoritmia metaheurística 117
Anexo F: Informe de pruebas del flujo de datos del diseño del algoritmo 118
Anexo G: Certificación de validación del pseudocódigo del algoritmo lobo gris, autenticado por un experto en algoritmia metaheurística 119
Anexo H: Código fuente del algoritmo lobo gris 120
Anexo I: Documento de pruebas unitarias sobre el algoritmo lobo gris 121
Anexo J: Certificación de validación de la implementación del algoritmo lobo gris, respaldada por la firma de un especialista en algoritmia metaheurística 122
Anexo K: Archivos CSV de la calibración de variables 123
Anexo L: Certificado de validación de la calibración de variables del algoritmo lobo gris, autenticado por un experto en algoritmia 124
Anexo M: Documento de validación del pseudocódigo del algoritmo genético de la literatura firmado por un especialista en algoritmia metaheurística 125
Anexo N: Código fuente del algoritmo genético 126
Anexo O: Documento de pruebas unitarias sobre el algoritmo genético 127
Anexo P: Certificado de validación de la implementación del algoritmo genético, autenticado por un especialista en algoritmia metaheurística 128
Anexo Q: Valores fitness utilizados para la experimentación numérica 129
Anexo R: Certificado de validación de la experimentación numérica, con la firma de un especialista en algoritmia 131

Índice de figuras

Figura 1. Árbol de problemas. Zona central: problema principal. Zona superior: problemas efectos. Zona inferior: problemas causas 15
Figura 2. Representación de algoritmo genético. A la izquierda representación del gen, cromosoma y población. A la derecha representación de las operaciones de cruzamiento y mutación 31
Figura 3. Estructura de datos - Tabla 48
Figura 4. Estructura de datos - Columna 49
Figura 5. Estructura de datos - Índice 49
Figura 6. Pseudocódigo del algoritmo lobo gris. Fuente: Creado por el autor 50
Figura 7. Pseudocódigo de la generación de la población inicial. Fuente: Creado por el autor 50
Figura 8. Pseudocódigo del método de actualizar posición. Fuente: Creado por el autor 51
Figura 9. Pseudocódigo del control de aberraciones. Fuente: Creado por el autor 52
Figura 10. Pseudocódigo del cálculo de la función fitness. Fuente: Creado por el autor 54
Figura 11. Pseudocódigo de la obtención de las tres mejores soluciones. Fuente: Creado por el autor 54
Figura 12. Pseudocódigo del método de selección de columnas. Fuente: Creado por el autor 54
Figura 13. Modelo de archivo csv sobre las tablas de base de datos. Fuente: Creado por el autor 58
Figura 14. Modelo de archivo csv sobre las columnas de base de datos. Fuente: Creado por el autor 58
Figura 15. Modelo de archivo sql. Fuente: Creado por el autor 59
Figura 16. Codificación del algoritmo lobo gris. Fuente: Creado por el autor 60
Figura 17. Codificación de la generación de la población inicial. Fuente: Creado por el autor 61
Figura 18. Codificación de la construcción de un nuevo lobo. Fuente: Creado por el autor 61
Figura 19. Codificación de la actualización de posición de la población. Fuente: Creado por el autor 62
Figura 20. Codificación del control de aberraciones, parte 1. Fuente: Creado por el autor 63
Figura 21. Codificación del control de aberraciones, parte 2. Fuente: Creado por el autor 64
Figura 22. Codificación del control de aberraciones, parte 3. Fuente: Creado por el autor 65
Figura 23. Codificación del cálculo de la función fitness. Fuente: Creado por el autor 65
Figura 24. Codificación del cálculo del espacio de índice. Fuente: Creado por el autor 66
Figura 25. Codificación del cálculo de la frecuencia total de índice. Fuente: Creado por el autor 67
Figura 26. Codificación del cálculo de la penalidad total de índice. Fuente: Creado por el autor 67
Figura 27. Codificación del cálculo del tiempo de ejecución del índice, parte 1. Fuente: Creado por el autor 68
Figura 28. Ejemplo de sugerencia de índice en MySQL. Fuente: MySQL :: MySQL 8.0 Reference Manual :: 8.9.4 Index Hints, s.f. 68
Figura 29. Codificación del cálculo del tiempo de ejecución del índice, parte 2. Fuente: Creado por el autor 69
Figura 30. Codificación de obtención de las tres mejores soluciones. Fuente: Creado por el autor 70

Figura 31. Codificación de la selección de columnas. Fuente: Creado por el autor 70
Figura 32. Estructura de datos - Tabla 75
Figura 33. Estructura de datos - Columna 76
Figura 34. Estructura de datos - Índice 76
Figura 35. Pseudocódigo del algoritmo genético. Fuente: Creado por el autor 77
Figura 36. Pseudocódigo del cálculo de espacio de tablas en querys. Fuente: Creado por el autor 78
Figura 37. Pseudocódigo de la generación de la población inicial. Fuente: Creado por el autor 78
Figura 38. Pseudocódigo de la obtención de la mejor solución. Fuente: Creado por el autor 79
Figura 39. Pseudocódigo del control de aberraciones. Fuente: Creado por el autor 79
Figura 40. Pseudocódigo de la generación de una población. Fuente: Creado por el autor 80
Figura 41. Pseudocódigo del cálculo de la función fitness. Fuente: Creado por el autor 81
Figura 42. Modelo de archivo csv sobre las tablas de base de datos. Fuente: Creado por el autor 83
Figura 43. Modelo de archivo csv sobre las columnas de base de datos. Fuente: Creado por el autor 83
Figura 44. Modelo de archivo sql. Fuente: Creado por el autor 84
Figura 45. Codificación del algoritmo lobo gris. Fuente: Creado por el autor 85
Figura 46. Codificación de la generación de la población inicial. Fuente: Creado por el autor 86
Figura 47. Codificación de la actualización de posición de la población. Fuente: Creado por el autor 87
Figura 48. Codificación del control de aberraciones, parte 1. Fuente: Creado por el autor 88
Figura 49. Codificación del control de aberraciones, parte 2. Fuente: Creado por el autor 89
Figura 50. Codificación de obtención de la mejor solución. Fuente: Creado por el autor 90
Figura 51. Codificación del cálculo de la función fitness. Fuente: Creado por el autor 90
Figura 52. Codificación del cálculo de la penalidad total de índice. Fuente: Creado por el autor 91
Figura 53. Codificación del cálculo del tiempo de ejecución del índice, parte 1. Fuente: Creado por el autor 91
Figura 54. Ejemplo de sugerencia de índice en MySQL. Fuente: MySQL :: MySQL 8.0 Reference Manual :: 8.9.4 Index Hints, s.f. 92
Figura 55. Codificación del cálculo del tiempo de ejecución del índice, parte 2. Fuente: Creado por el autor 92
Figura 56. Menor valor fitness en la recolección de datos. Fuente: Creado por el autor 95
Figura 57. Estructura de descomposición de trabajo 107

Índice de tablas

Tabla 1. Asignación de objetivos, logros y verificación 21
Tabla 2. Herramientas, métodos y procedimientos 22
Tabla 3. Resultados del Criterio PICOC para la Revisión Sistemática 33
Tabla 4. Ampliación de los términos claves 34
Tabla 5. Hallazgos de la Investigación Sistemática 35
Tabla 6. Artículos considerados como adecuados para la Revisión Sistemática 36
Tabla 7. Planilla de extracción de datos 36
Tabla 8. Tipo de algoritmos 37
Tabla 9. Resultados obtenidos al calibrar el máximo número de iteraciones 72
Tabla 10. Resultados obtenidos al calibrar el máximo número de generaciones sin mejora 72
Tabla 11. Resultados obtenidos al calibrar el porcentaje de aceptación para elección de columnas 73
Tabla 12. Riesgos del proyecto 105
Tabla 13. Lista de tareas 107
Tabla 14. Cronograma 111
Tabla 15. Costeo 114
Tabla 16. Fitness utilizados para la experimentación numérica 129

Capítulo 1. Aspectos Generales

1.1 Problemática

Enseguida, enfatizando la necesidad identificada y respaldando la pertinencia del proyecto de fin de carrera, se describen tanto el contexto de la problemática como las causas y consecuencias fundamentales.

1.1.1 Árbol de problemas

La Figura 1 exhibe el árbol de problemas, se analiza tanto el marco contextual de la problemática como las causas y consecuencias esenciales, con el propósito de resaltar la necesidad identificada y respaldar la pertinencia del proyecto de fin de carrera.

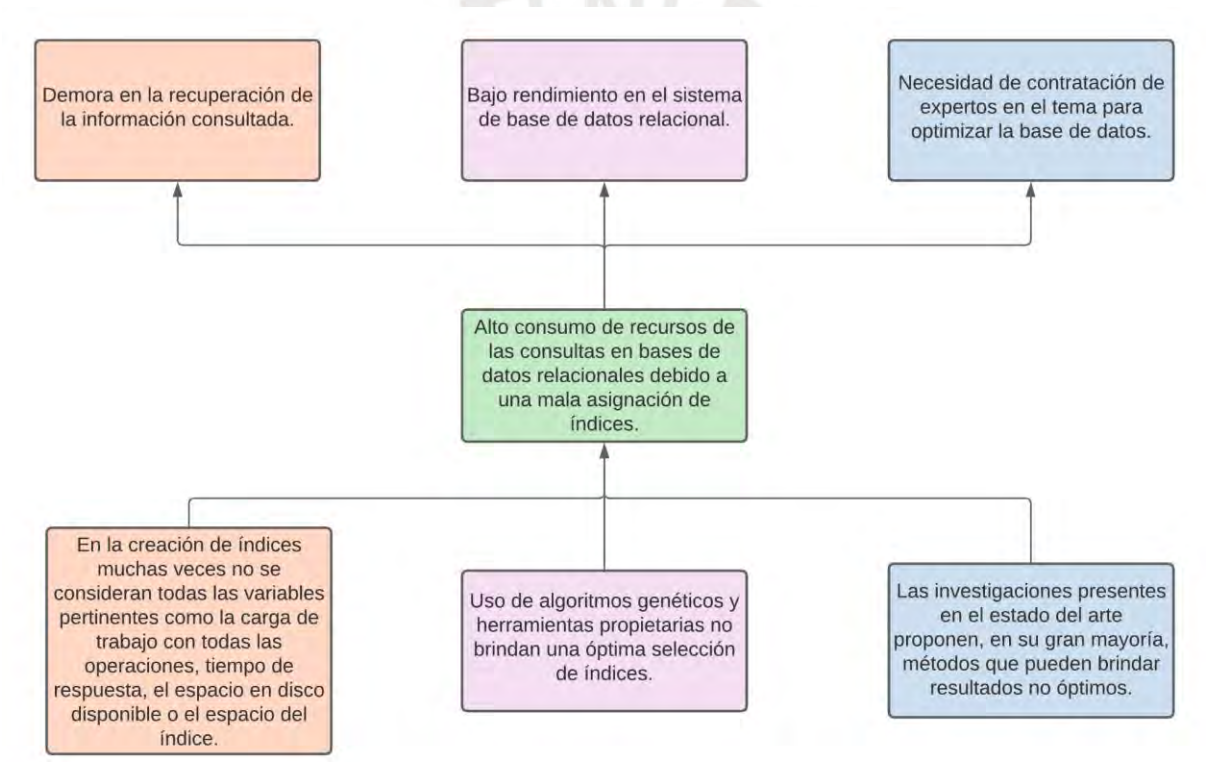


Figura 1. Representación visual del árbol de problemas. En el centro se ubica el problema central, en la sección superior se identifican los problemas resultantes, mientras que en la zona inferior se describen los problemas subyacentes.

1.1.2 Descripción

En el mundo altamente competitivo de hoy en día, las empresas focalizan su atención de manera significativa en la información, considerándola como el activo más preciado de la organización, por encima de otros activos importantes como el hardware y software, e inclusive, sobre recurso humano. El beneficio de los datos yace en el hecho de que tan pronto se pueden transformar en información útil y en conocimiento estratégico (Wijesiriwardana & Firdhous, 2019). En consecuencia, un número creciente de organizaciones está optando por

implementar sus propios sistemas de información con el objetivo de respaldar una toma de decisiones más eficiente (Wijesiriwardana & Firdhous, 2019). Un sistema de gestión de bases de datos es el componente principal de cualquier sistema de información, independientemente de su tamaño y naturaleza de operación. De manera que, su almacenamiento y extracción son una parte importante para su uso (Wijesiriwardana & Firdhous, 2019).

Las bases de datos pueden clasificarse en bases de datos relacionales y no relacionales. En el caso de las bases de datos relacionales, los datos almacenados están interconectados entre sí. La información se organiza mediante tablas, que incluyen columnas y filas, pudiendo obtener acceso a estos datos de diferentes maneras (AWS, s.f.).

De acuerdo con la encuesta realizada por la empresa DB-Engines en marzo del 2020, en la cual se incluyen un total de 354 sistemas, las bases de datos relacionales se posicionan en primer lugar de popularidad con 74.8% (Lieponienė, 2020). Por lo cual, se puede afirmar que en la actualidad este tipo de base de datos sigue manteniéndose relevante y resulta necesario encontrar maneras de mejorar su rendimiento.

La tendencia dominante del desarrollo moderno de bases de datos es la producción de sistemas de gestión de bases de datos relacionales con procesos y herramientas basadas en el conocimiento humano, necesarios para la explotación eficiente de estos sistemas. Para mantener los datos, los usuarios se comunican con las bases de datos mediante consultas. Por lo mismo, estos sistemas deben ofrecer un uso óptimo de los recursos del sistema y brindar la información solicitada en un tiempo de respuesta mínimo (Kovačević & Filipič, 2006).

Para los administradores de base de datos (DBA, por sus siglas en inglés) e investigadores, la mejora del rendimiento de las bases de datos ha sido un reto persistente a lo largo de los años (Pedrozo & Vaz, 2014). Este desafío tiene diversos factores, entre los que se encuentra la configuración de hardware, la cual, si se realiza incorrectamente, no se llegan a obtener mayores beneficios. Dentro de los principales recursos de hardware para tener en cuenta están el procesador, la memoria y el disco (Fritchey, 2018). Otro factor fuertemente relacionado es la forma de organización lógica de los registros y, especialmente, a la rapidez con la que se accede y procesa esos registros (Pedrozo & Vaz, 2014).

Por un lado, a través de los años, la memoria y el procesador se han vuelto cada vez más rápidos, de manera que ya no resultan un problema. Por otro lado, en cuanto a el disco, a excepción de casos particulares con mejoras radicales empleando discos de estado sólido como tecnología principal, no han existido muchos cambios. Además, los discos siguen siendo una de las partes más lentas de la mayoría de los sistemas (Fritchey, 2018).

En ese sentido, los índices de las tablas son una estructura física que se crea en la misma tabla para potenciar la recuperación de datos durante la ejecución de la transacción (Naik, 2018). Siendo un caso de ejemplo, cuando los datos son solicitados, el sistema de gestión de base de datos primero verifica si existe un índice y su estructura, la cual está ordenada y contiene la dirección física del dato, permitiendo así recuperar la información directamente en el disco duro, simplificando el trabajo de búsqueda (Pedrozo & Vaz, 2014).

Si bien los índices sirven para acelerar la recuperación de los datos, uno mal diseñado deteriora el rendimiento general. Esto se debe a los diferentes factores que influyen en su creación, como lo son: El tipo de dato, la estructura de la tabla, el número de veces que se buscó un campo específico, la frecuencia de ciertas consultas, la frecuencia de valores distintos en una columna, la carga de trabajo, el número de operaciones de lectura y escritura, entre otros (Naik, 2018).

La configuración de índices se puede entender como la creación de un índice dado un conjunto de columnas pertenecientes a una o más tablas de tal manera que proporcione el mejor rendimiento para recuperar los datos (Neuhaus et al., 2019).

Encontrar una configuración de índice óptima para obtener el costo de ejecución mínimo, equilibrando tanto la búsqueda como las consultas de mantenimiento sobre los datos de la carga de trabajo de la aplicación, se conoce formalmente como el problema de selección de índice (Kovačević & Filipič, 2006).

Entre los principales problemas de la selección de índices se puede encontrar que la mayoría de enfoques no usan las mismas variables, mientras que unos se centran en variables de entrada (Neuhaus et al., 2019) (Boronski & Bocewicz, 2014) (Kołaczkowski & Rybiński, 2011b) (Kołaczkowski & Rybiński, 2009) (Kovačević & Filipič, 2006) (Korytkowski et al., 2004) (Chaudhuri et al., 2004), otros se centran en las variables propias del algoritmo (Kołaczkowski & Rybiński, 2011a) (Calle et al., 2011) (Kratica et al., 2003). Cabe resaltar que dentro de cada grupo la cantidad de variables es distinta, por ejemplo, para la mayoría de estudios es suficiente contar de 4 a 6 variables principales (Neuhaus et al., 2019) (Boronski & Bocewicz, 2014) (Calle et al., 2011) (Kołaczkowski & Rybiński, 2009) (Korytkowski et al., 2004) (Chaudhuri et al., 2004), por el contrario, ciertos autores consideran de 7 hasta 15 variables (Kołaczkowski & Rybiński, 2011a) (Kołaczkowski & Rybiński, 2011b) (Kovačević & Filipič, 2006) (Kratica et al., 2003). Entre las variables de entradas más usadas se puede listar la carga de trabajo de consultas, estando presente en todos los estudios, el tiempo de respuesta (Neuhaus et al., 2019) (Boronski & Bocewicz, 2014) (Korytkowski et al., 2004) o el espacio en disco de la base de datos (Boronski & Bocewicz, 2014) (Kołaczkowski & Rybiński, 2011b) (Chaudhuri et al., 2004) (Kratica et al., 2003) y el índice a crear (Kołaczkowski & Rybiński,

2011b) (Kołaczkowski & Rybiński, 2011a) (Kratica et al., 2003). Dichas variables son consideradas básicas y pertinentes para atacar este problema. Sin embargo, existe una discrepancia sobre qué variables deben ser utilizadas dado que tienen un impacto en el rendimiento de la solución, causando demora en la recuperación de la información consultada, siendo este uno de los objetivos primordiales a solucionar del problema de selección de índices (Neuhaus et al., 2019).

Otro problema causa muy importante recae en las investigaciones, porque casi en su totalidad se basan en algoritmos genéticos, los cuales tienen más de 20 años de uso en el mercado (Katoch et al., 2021), y otras investigaciones se basan en herramientas propietarias de cada motor de base de datos (Naik, 2018), por lo que no pueden ser aplicadas a cualquier motor. Cabe resaltar que las herramientas comerciales disponibles como Oracle Access Advisor no son capaces de emplear todas las capacidades de indexación de tablas (Boronski & Bocewicz, 2014). En este sentido, estas soluciones alternativas no ofrecen una selección completa de índices, lo que resulta en un impacto negativo en la eficiencia en el sistema de base de datos relacional. Dado que este rendimiento está directamente relacionado con el diseño físico, se puede mejorar mediante la selección adecuada de índices (Calle et al., 2011).

Finalmente, estas alternativas de soluciones no han dado resultados satisfactorios y se necesita hacer comparaciones, ya que, si bien cada una presenta intentos interesantes para resolver el problema y poseen muchos beneficios, no son suficientes para implementarlos en la práctica para cualquier motor (Boronski & Bocewicz, 2014). Esto se debe a que en estudios anteriores se prestó poca atención a casos como el tamaño y tiempo de creación del índice, o la indexación de tablas de consulta única. También se omitieron los casos de consultas relacionadas a la misma tabla, resultando demasiados índices similares (Boronski & Bocewicz, 2014). Esto puede generar la necesidad de contratación de expertos en el tema para optimizar la base de datos, siendo este costo en algunos casos más alto que una licencia de base de datos (Kołaczkowski & Rybiński, 2009).

Desde hace muchos años se han adaptado diferentes formas de solucionar este problema, por ejemplo, Kratica en el año 2003 desarrolló un algoritmo genético obteniendo resultados de buen rendimiento, confiabilidad y eficiencia bajo el enfoque propuesto. Korytkowski en el año 2004 propone un nuevo método para mejorar el rendimiento de la base de datos usando algoritmos genéticos. En ese mismo año, Chaudhuri desarrolla un nuevo algoritmo tratándolo como el problema de la mochila, siendo la novedad de su acercamiento el uso del método de programación lineal en la asignación de beneficios para cada índice. En el 2006, Kovačević presenta una nueva versión de su algoritmo GALIO (Genetic ALgorithm for Index Optimization), contando ahora con un método de adaptación, basado en un algoritmo voraz,

y probándose sobre bases de datos en escenarios reales mostrando mejoras frente anteriores resultados. Kołaczkowski desarrolló 3 algoritmos durante 3 años. El primero, en el 2009, enfocándose en un método que busque en el espacio de los posibles planes de ejecución de las consultas, en lugar que en el espacio de configuraciones de índice, usando algoritmos genéticos. El segundo, en el 2011, una herramienta para la selección automática de índices secundarios, también usando algoritmos genéticos afirmando conseguir resultados cercanos al óptimo en corto tiempo. El tercero, igualmente en el mismo año, utiliza un enfoque evolutivo (algoritmo genético) de estado estacionario para proporcionar recomendaciones de índices de manera continua, de modo que el sistema de administración de bases de datos pueda adaptarse a una carga de trabajo cambiante. Adicionalmente, se tiene en cuenta la reutilización de índices en diferentes consultas. En 2011, Calle propuso el uso de algoritmos evolutivos, especialmente genéticos, como una solución óptima para abordar esta problemática. El objetivo principal era desarrollar un sistema de base de datos autoajustable que necesitara una intervención mínima o nula de expertos para el diseño físico. Por su lado, Boronski en el 2014 propone un método propio usando algoritmos genéticos para la indexación de tablas para un grupo de consultas, logrando reducir el tiempo de ejecución y la memoria necesaria para la creación. Finalmente, Neuhaus en el 2019 desarrolla GADIS (Genetic Algorithm for Database Index Selection), Este algoritmo elige automáticamente la configuración óptima de índices y es adaptable a cualquier estructura de base de datos.

Del mismo modo, existen diversas herramientas de recomendación de índices desarrolladas para un solo motor de base de datos. MS SQL Server posee SQL Profiler y Database Engine Tuning Advisor, mientras que Oracle posee SQL Tuning Advisor, para mejorar el diseño físico y el rendimiento de la base de datos (Naik, 2018).

Como hemos podido apreciar en estas investigaciones, si bien cada una propone una solución novedosa, no han logrado satisfacer las necesidades que tiene el usuario. Por lo tanto, el objetivo de este proyecto de fin de carrera es optimizar el consumo de recursos (tiempo de respuesta y espacio en disco) de las consultas mediante el uso de índices, brindando la mejor opción para la creación de estos, sobre las tablas dentro un sistema de gestión de bases de datos relacionales.

Para el presente proyecto se plantea utilizar un algoritmo metaheurístico aplicado al problema de selección de índices, el cual se resume en, dada una base de datos y un conjunto de consultas, proponer un conjunto de índices (Chaudhuri et al., 2004). Dado que este problema es considerado un “NP-Hard Problem”, no es adecuado emplear algoritmos exactos ya que demandarían demasiados recursos (memoria RAM y tiempo).

1.1.3 Elección del problema

El desafío identificado en este proyecto de fin de carrera es el alto consumo de recursos (tiempo de respuesta y espacio en disco) de las consultas en bases de datos relacionales debido a una ineficiente asignación de índices.

1.2 Objetivos

En esta parte del trabajo, se detalla el objetivo general, los objetivos específicos y las expectativas de resultados para el desarrollo del proyecto de fin de carrera.

1.2.1 Objetivo general

Implementar un algoritmo lobo gris que brinde solución al problema de selección de índices lógicos en bases de datos relacionales.

1.2.2 Objetivos específicos

O1. Definir las variables y restricciones necesarias del problema para el establecimiento de la función objetivo.
O2. Adaptar un algoritmo lobo gris que brinde una óptima selección de índices.
O3. Evaluar y comparar el rendimiento del algoritmo lobo gris propuesto con la solución más prevalente identificada en el estado del arte, con el propósito de validar los resultados obtenidos.

1.2.3 Resultados esperados

R1. Identificación y especificación de las variables críticas y restricciones esenciales para ser consideradas en la selección de índices (O1).
R2. Definición de la función objetivo que ayude con la optimización (O1).
R3. Pseudocódigo del algoritmo lobo gris propuesto (O2).
R4. Codificación del algoritmo lobo gris adaptado al problema de selección de índices (O2).
R5. Calibración de variables y constantes del algoritmo lobo gris (O2).
R6. Pseudocódigo del algoritmo genético obtenido del estado del arte (O3).
R7. Codificación del algoritmo genético seleccionado (O3).
R8. Experimentación numérica para la comparación entre el algoritmo propuesto y el algoritmo genético (O3).

1.2.4 Asignación de objetivos, logros y verificación

Tabla 1.Asignación de objetivos, logros y verificación

Objetivo 1: Definir las variables y restricciones necesarias del problema para el establecimiento de la función objetivo.		
Resultado	**Medio para la comprobación**	**Indicador que puede ser objetivamente corroborado**
R1. Identificación y especificación de las variables críticas y restricciones esenciales para ser consideradas en la selección de índices.	Informe que presenta la especificación de las variables y restricciones	Evaluación integral y validación de las variables y restricciones por parte de un experto en administración de bases de datos.
R2. Definición de la función objetivo que ayude con la optimización.	Informe que detalla la formulación matemática de la función objetivo.	Evaluación completa y validación de la función objetivo por parte de un especialista en administración de bases de datos. Evaluación completa y validación de la función objetivo por un experto en algoritmia metaheurística.
Objetivo 2: Adaptar un algoritmo lobo gris que brinde una óptima selección de índices.		
Resultado	**Medio para la comprobación**	**Indicador que puede ser objetivamente corroborado**
R3. Pseudocódigo del algoritmo lobo gris propuesto.	Informe que expone el pseudocódigo del algoritmo lobo gris. Informe exhaustivo sobre la ejecución de las pruebas de flujo de datos del diseño del algoritmo	Análisis exhaustivo y comprobación del pseudocódigo llevado a cabo por un especialista en algoritmia metaheurística. Evaluación completa y verificación del flujo de datos realizadas por un experto en algoritmos.
R4. Codificación del algoritmo lobo gris adaptado al problema de selección de índices.	Código fuente completo del algoritmo lobo gris, que abarca tanto el núcleo del algoritmo como sus funciones auxiliares. Informe que engloba el conjunto de pruebas realizadas para evaluar la implementación del algoritmo lobo gris	Evaluación minuciosa y confirmación del código por un experto en algoritmia metaheurística. Realización exitosa de las pruebas unitarias del algoritmo lobo gris y sus funciones secundarias.
R5. Calibración de variables	nforme detallado sobre la	Verificación exhaustiva y

y constantes del algoritmo lobo gris.	calibración de parámetros y constantes del algoritmo lobo gris.	validación completa de la calibración de variables por parte del especialista en algoritmia metaheurística.
Objetivo 3: Evaluar y comparar el rendimiento del algoritmo lobo gris propuesto con la solución más prevalente identificada en el estado del arte, con el propósito de validar los resultados obtenidos		
Resultado	**Medio para la comprobación**	**Indicador que puede ser objetivamente corroborado**
R6. Pseudocódigo del algoritmo genético obtenido del estado del arte.	Informe que expone el pseudocódigo del algoritmo genético.	Examen riguroso y confirmación completa del pseudocódigo por parte del especialista en algoritmia genética
R7. Codificación del algoritmo genético seleccionado.	Código fuente completo del algoritmo genético, que abarca tanto el núcleo del algoritmo como sus funciones auxiliares. Informe que engloba el conjunto de pruebas realizadas para evaluar la implementación del algoritmo genético.	Validación exhaustiva del código por parte del especialista en algoritmia genética. Realización satisfactoria del conjunto completo de pruebas unitarias del algoritmo genético y sus funciones auxiliares.
R8. Experimentación numérica para la comparación entre el algoritmo propuesto y el algoritmo genético.	Informe de experimentación numérica que evidencie el algoritmo que arroje los mejores resultados	Evaluación completa y validación exhaustiva de la experimentación numérica por parte del especialista en algoritmia.

1.3 Métodos y procedimientos

Tabla 2. Herramientas, métodos y procedimientos

Expectativas de resultados	**Instrumentos, enfoques, prácticas y técnicas**
Definir las variables y restricciones necesarias del problema para la definición de la función objetivo (O1).	- No aplica.
Definición de la función objetivo a optimizar (O1).	- No aplica.
Pseudocódigo del algoritmo lobo gris desarrollado (O2).	- Pseudocódigo. - PseInt. - Algoritmo Lobo Gris
Codificación del algoritmo lobo gris adaptado al problema de selección de	- IntelliJ IDEA. - Java.

índices (O2).	- Git. - Buenas prácticas de la metodología Kanban.
Calibración de variables y constantes del algoritmo lobo gris (O2).	- No aplica.
Pseudocódigo del algoritmo genético obtenido del estado del arte (O3).	- Pseudocódigo. - PseInt.
Codificación del algoritmo genético seleccionado (O3).	- IntelliJ IDEA. - Java. - Git. - Buenas prácticas de la metodología Kanban.
Experimentación numérica que demuestre el algoritmo que brinde mejores resultados (O3).	- RStudio. - Pruebas estadísticas no paramétricas.

1.3.1 Pseudocódigo

Esta palabra es derivada de otras dos: Pseudo y código. Pseudo significa imitación y código se puede interpretar como una serie de instrucciones escritas en lenguaje computadora o de programación. El propósito del pseudocódigo es describir la secuencia lógica de un programa sin importar cual sea el lenguaje de programación (Nita & Kartikawati, 2020).

Se empleará en el desarrollo del proyecto de tesis con el propósito de elaborar el esquema detallado de ejecución de los algoritmos propuestos.

1.3.2 PseInt

PseInt es una herramienta utilizada para la construcción de diagramas de flujo, ofreciendo la ventaja de la autogeneración del código y su ejecución en lugar de realizar estos diagramas manualmente. Presenta dos modalidades: la escritura de pseudocódigo y la construcción de diagramas de flujos. Además, incluye características propias de un Entorno de Desarrollo Integrado (IDE), como autocompletado, ayuda emergente, resaltado de sintaxis, indentado, y un listado completo de funciones, operadores y variables. También permite compilar, depurar y ejecutar el código (Sánchez et al., 2020).

En el contexto de este proyecto de tesis, se utilizará PseInt con el propósito de diseñar y visualizar de manera gráfica el flujo de ejecución de los algoritmos propuestos.

1.3.3 Algoritmo Lobo Gris

El algoritmo lobo gris es una metaheurística que emula la jerarquía de liderazgo y el comportamiento de caza de los lobos grises. Inspirándose en la naturaleza, utiliza cuatro tipos de lobos grises: Alfa, beta, delta y omega, para simular la estructura jerárquica de liderazgo. Además, incorpora tres etapas o métodos principales de caza: búsqueda de presas, cercado de presas y ataque de presas, con el objetivo de llevar a cabo la optimización (*GWO*, s.f.). La elección de este algoritmo metaheurístico se basa en su historial de buenos resultados en problemas de asignación de complejidad similar. Además, se ha evidenciado su competitividad en comparación con otras metaheurísticas reconocidas, tales como la optimización de enjambre de partículas, el algoritmo de búsqueda gravitacional, la evolución diferencial, la programación evolutiva y estrategias de evolución (Mirjalili et al., 2014).

Se empleará y adaptará este algoritmo como propuesta de solución para el problema de selección de índices.

1.3.4 IntelliJ IDEA

IntelliJ IDEA es una herramienta avanzada de desarrollo integrado (IDE) que está especialmente orientada a los lenguajes que se ejecutan en la Máquina Virtual de Java (JVM), como Java, Kotlin, Scala y Groovy. Su diseño se centra en maximizar la productividad del desarrollador al facilitar tareas rutinarias y repetitivas, ofreciendo características como finalización inteligente de código, análisis de código estático y refactorizaciones. Gracias a estas funcionalidades, el programador puede concentrarse en el desarrollo sin distracciones innecesarias (*IntelliJ IDEA Overview | IntelliJ IDEA*, s.f.).

Para llevar a cabo la implementación de los algoritmos en el proyecto, se empleará el entorno de trabajo IntelliJ IDEA.

1.3.5 Java

Ampliamente utilizado en una variedad de aplicaciones y sitios web, Java es tanto un lenguaje de programación como una plataforma informática, estando en constante uso actualmente y en un futuro. El uso de Java se puede ver en distintas plataformas como portátiles, centros de datos, consolas para juegos, súper computadoras, teléfonos móviles y teléfonos móviles, siendo rápido, seguro y fiable (*¿Qué Es Java y Para Qué Es Necesario?*, n.d.).

Como lenguaje de programación principal para la implementación de los algoritmos, se utilizará Java.

1.3.6 Git

Git es un sistema de control de versiones distribuido de código abierto, diseñado para gestionar proyectos de diversos tamaños, desde pequeños hasta grandes, con rapidez y eficiencia (Git, s.f.).

En este proyecto, se empleará como repositorio para el código fuente de los algoritmos.

1.3.7 Kanban

Kanban es una metodología que persigue lograr un proceso productivo organizado y eficiente. Originada en Toyota, Japón, fue inicialmente empleada para controlar el avance del trabajo en una cadena de producción. Pertenece a la metodología "Lean Manufacturing", la cual se basa en técnicas just-in-time (JIT). Su objetivo principal es asegurar una tasa de producción sostenible para prevenir problemas como el exceso de productos terminados, cuellos de botella y retrasos en la entrega de pedidos. La metodología implica organizar el trabajo en curso de acuerdo con la capacidad del centro de trabajo y los equipos, así como mantener una comunicación en tiempo real sobre la capacidad y una transparencia total en el trabajo. (Castellano, 2019).

Para el desarrollo del proyecto de fin de carrera se utilizarán algunas buenas prácticas de esta metodología.

1.3.8 RStudio

RStudio es un entorno de desarrollo integrado (IDE) diseñado específicamente para el lenguaje de programación R, distribuido bajo una licencia de código abierto pero también disponible comercialmente. Incluye una consola, un editor con resaltado de sintaxis que permite la ejecución directa de código, así como herramientas para trazado, historial, depuración y gestión del espacio de trabajo (RStudio - RStudio, s.f.).

En el proyecto de tesis, se empleará RStudio para llevar a cabo diversos análisis estadísticos sobre los resultados obtenidos a través de los algoritmos desarrollados..

1.3.9 Pruebas estadísticas no paramétricas

Cuando los procedimientos estadísticos no requieren realizar inferencias sobre los parámetros de la población, como la media y la dispersión, se les denomina no paramétricos o de distribución libre. Esto se debe a que no se hacen suposiciones acerca de la distribución de la población proveniente de la muestra. En este tipo de pruebas, se pueden utilizar

muestras pequeñas de datos ordinales o categóricos, independientemente de la distribución de las muestras a contrastar (Gómez et al., 2003).

En el proyecto de tesis, se emplearán estas pruebas no paramétricas para llevar a cabo la experimentación numérica en casos donde no se cumplan las condiciones para realizar una prueba paramétrica, permitiendo así la comparación de los resultados de los algoritmos.

Capítulo 2. Marco Legal y Conceptual

2.1 Introducción

El propósito de este marco es introducir los conceptos pertinentes al área de investigación y abordar la problemática asociada, que se centra en el excesivo consumo de recursos durante las consultas en bases de datos relacionales, originado por una asignación inadecuada de índices.

2.2 Desarrollo del marco conceptual

2.2.1 Base de datos relacional

En términos generales, una base de datos se define como una forma de almacenar información para que pueda ser recuperada posteriormente (A Relational Database Overview, s.f.). En particular, una base de datos relacional se basa en el modelo de datos propuesto por Edgar Frank Codd en 1970 (F., 1970). Este modelo se describe mediante dos conceptos clave: "instancia", que se refiere a una tabla con filas y columnas; y "esquema", que especifica la estructura, el nombre de la relación y los nombres y tipos de las columnas. Estas ideas transformaron la manera en que las personas concebían las bases de datos (Berg et al., 2013).

En el modelo relacional, cada fila de una tabla representa un registro con un identificador único llamado clave. Las columnas de la tabla contienen atributos de los datos, y cada registro generalmente tiene un valor para cada atributo, facilitando el establecimiento de relaciones entre los datos (What Is a Relational Database | Oracle, s.f.).

2.2.2 Sistema de administración de base de datos relacional

Un Sistema de Administración de Bases de Datos (DBMS) tiene la responsabilidad de manejar cómo se almacenan, mantienen y recuperan los datos. En el caso de una base de datos relacional, un Sistema de Gestión de Bases de Datos Relacionales (RDBMS) realiza estas funciones (A Relational Database Overview, s.f.). Mientras que una base de datos relacional describe el tipo de base de datos que gestiona un RDBMS, el RDBMS se refiere al programa de base de datos en sí. En otras palabras, es el software que ejecuta las operaciones sobre los datos y proporciona una representación visual de los mismos (RDBMS Definition, s.f.).

Aunque el modelo relacional es simple, es muy poderoso y se utiliza en organizaciones de todos los tamaños para diversas necesidades de información. Entre los beneficios del RDBMS se encuentra su aplicabilidad a cualquier requerimiento de información en el cual los

datos estén relacionados entre sí y deban ser gestionados de manera segura, basada en reglas y consistente (What Is a Relational Database | Oracle, s.f.).

2.2.3 Optimización de base de datos

La optimización de la base de datos, definida como la tarea de analizar y mejorar el rendimiento del sistema de administración de bases de datos, requiere un conocimiento extenso de los aspectos internos de la base de datos por parte del administrador (Kołaczkowski & Rybiński, 2011a; Almeida et al., 2019). Con el fin de mejorar la velocidad del sistema, el proceso de optimización generalmente implica ajustes en las estructuras de datos, los parámetros del sistema, la configuración del sistema operativo o el hardware (Almeida et al., 2019). Adicionalmente, el número de todas las configuraciones posibles crece exponencialmente con el tamaño del esquema de la base de datos (Kołaczkowski & Rybiński, 2011a).

2.2.4 Consulta

Una consulta, o query por su nombre en inglés, son los datos requeridos que se recuperan de una base de datos mediante instrucciones al sistema. Estas consultas se realizan utilizando un lenguaje de alto nivel llamado Structured Query Language (SQL). Cuando una consulta es emitida, esta es sometida a una serie de operaciones denominadas “procesamiento de consultas” para recuperar la información necesaria de la base de datos. Estas operaciones incluyen la transformación de consultas en lenguajes de alto nivel a instrucciones de máquina de bajo nivel, un grupo de optimizaciones de consultas y la evaluación real de la consulta. Estos pasos se pueden agrupar en tres partes: análisis y traducción, optimización y evaluación (Wijesiriwardana & Firdhous, 2019).

2.2.5 Índice

Un índice de base de datos es una estructura de acceso físico para una tabla de base de datos e indica dónde se almacenan físicamente los registros en el disco. El índice tiene como fin mejorar la velocidad de las operaciones de recuperación de datos a costa de escrituras más lentas y mayor espacio de almacenamiento. Su creación involucra una o más columnas de una tabla y el espacio que requiere es comparativamente menor que el almacenamiento de la tabla original (Gupta & Badal, 2012).

Los índices se pueden clasificar en dos categorías: Índice agrupado, clustered, e índice no agrupado, non-clustered (Gupta & Badal, 2012). El primero, también llamado índice primario (Silberschatz et al., 2019), altera la base de datos de manera física para que el orden coincida con el índice, haciendo que los datos de la fila se almacenen en orden (Gupta & Badal, 2012). Este tipo de índice se construye normalmente en base a la llave primaria (Silberschatz et al.,

2019). Por lo tanto, solo se puede crear un índice agrupado en una tabla de base de datos determinada (Gupta & Badal, 2012). Por otro lado, el segundo, también llamado índice secundario (Silberschatz et al., 2019), los datos están de manera aleatoria en una tabla, pero el orden lógico se detalla en el índice. Este contiene un puntero hacia el número de la fila que contiene los datos (Gupta & Badal, 2012).

En caso no se use un índice, el sistema de base de datos deberá buscar en todos los registros de una tabla para obtener la respuesta correcta, lo que, en el peor de los casos, puede llevar mucho tiempo. Un óptimo índice puede reducir el tiempo de búsqueda de varias horas a unos pocos segundos. Ya que es un objeto de base de datos y está físicamente separado no tiene ningún impacto directo en los datos. Eso permite colocarse en el mismo archivo que los datos y en caso de modificación o eliminación, estos cambios no provocan pérdida ni daño de datos (Korytkowski et al., 2004).

2.2.6 Tiempo de respuesta

El tiempo de respuesta se define como el rendimiento de una transacción o consulta, detalladamente, es el tiempo transcurrido desde que el usuario ingresa un comando hasta que la aplicación indica que se ha completado la acción. Este tiempo está compuesto por una secuencia de acciones, cada una con una cantidad de tiempo diferente. Cabe recalcar que el tiempo de respuesta no incluye el tiempo que le toma al usuario pensar e ingresar una consulta (*Response Time - IBM Documentation*, s.f.).

2.2.7 Problema de selección de índices

Según Calle, “El problema de selección de índices se puede definir como la búsqueda de una combinación particular de índices de modo que se minimice el costo de una determinada carga de trabajo en la base de datos”.

Esta tarea resulta un desafío por varias razones. La primera, en vista que los esquemas de bases de datos de aplicaciones reales pueden ser grandes, en cuestión de tablas y columnas, y se pueden crear indefinidos índices para una secuencia de una o más columnas, el espacio que ocupa para la carga de trabajo puede ser muy grande. La segunda, los optimizadores de consultas actuales están en la capacidad de explotar los índices disponibles utilizando técnicas sofisticadas. Por lo tanto, es importante tener en cuenta las interacciones entre índices. La tercera, la elección de los índices se encuentran bajo ciertas restricciones, como la limitación de la cantidad de espacio de almacenamiento asignado a los índices. En síntesis, la selección de índices se plantea como un problema de optimización cuyo objetivo es identificar el subconjunto más efectivo de índices relevantes para la carga de trabajo, dentro de las restricciones establecidas (Chaudhuri et al., 2004).

2.2.8 Algoritmo metaheurístico

Los algoritmos metaheurísticos se pueden definir como paradigmas de inteligencia computacional que se utilizan especialmente para resolver problemas de optimización sofisticados. Estos se pueden clasificar de dos maneras, los que están basados en metáforas y los que no. Dentro del primer grupo se puede encontrar algoritmo genético, optimización de enjambre de partículas, optimización de ondas de agua, entre otros. Mientras que en el segundo está la búsqueda tabú o la búsqueda de vecindario variable (Abdel-Basset et al., 2018).

Este tipo de algoritmos, frecuentemente inspirados en la naturaleza, están diseñados para abordar problemas complejos de optimización. Estas fungen como alternativas exitosas frente a enfoques más clásicos para resolver problemas de optimización que incluyen información incierta, estocástica y dinámica (Ae et al., 2008).

2.2.9 Algoritmo genético

Este concepto fue desarrollado por Holland inspirado en la teoría evolucionista. El algoritmo genético simula los principios de la evolución biológica, el cual se basa en modificar repetidamente una población de individuos usando reglas modeladas en reproducción y combinaciones de genes. En esta simulación, se aprovecha los mejores genes de los individuos más aptos para mantenerse en las siguientes generaciones. Este algoritmo modela a los individuos en términos de genoma, tradicionalmente representado como cadenas de bits. Cada generación reemplaza a la población anterior por su descendencia más apta, en donde la aptitud se mide por medio de una función objetivo que asigna una puntuación a cada individuo. Esta función generalmente mide qué tan bien se resuelve el problema. El algoritmo funciona inicializando una población aleatoria y trabaja mediante operaciones de selección, cruzamiento y mutación. Ese proceso no termina hasta que se satisface el criterio de optimización o se alcanza un número determinado de generaciones (Neuhaus et al., 2019).

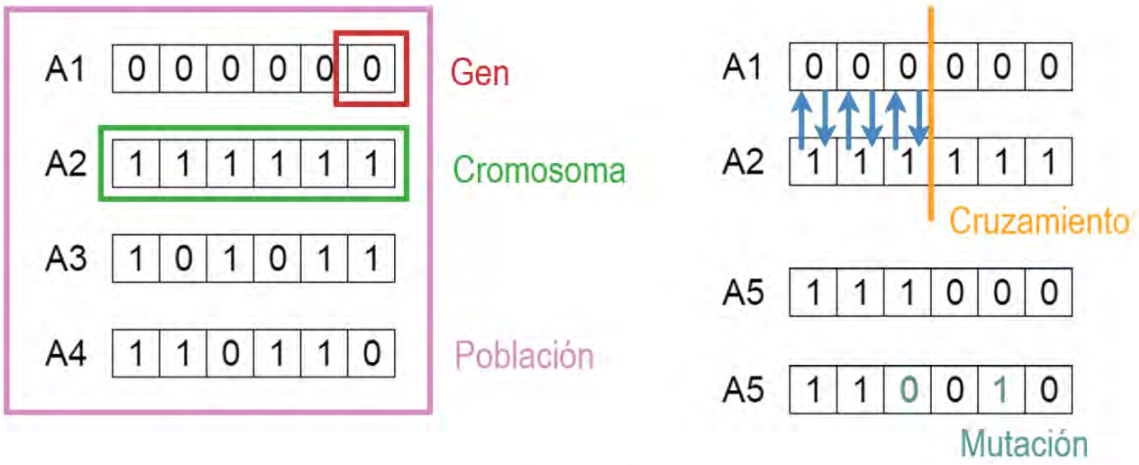


Figura 2. Representación de algoritmo genético. A la izquierda representación del gen, cromosoma y población. A la derecha representación de las operaciones de cruzamiento y mutación

Capítulo 3. Estado del Arte

3.1 Introducción

Para llevar a cabo este trabajo, resulta necesario examinar el contexto actual a través de publicaciones académicas, para lo cual se emplea la metodología de revisión sistemática. Esta metodología comprende los siguientes pasos: (1) Identificar las preguntas de investigación, (2) definir los criterios de inclusión y exclusión, (3) búsqueda y selección de estudios, (4) extracción de datos, y (5) sintetizar y presentar los resultados (Covidence, s.f.).

3.2 Objetivos de la revisión

El tipo de revisión que llevaremos a cabo será tanto empírica como conceptual, con la expectativa de obtener conceptos que sirvan como base tanto para el estado del arte como para los experimentos a realizar. El objetivo es hallar una solución óptima a este problema. A continuación, se enumeran un conjunto de objetivos que se espera lograr al finalizar la revisión sistemática.

- Comprender conceptos generales sobre el problema de selección de índices.
- Identificar los estudios que permitan desarrollar una solución al problema de selección de índices, reconociendo variables y restricciones que se aplican para atacar el problema.
- Conocer la eficiencia y resultados de los algoritmos metaheurísticos que se han adaptado al problema de selección de índices en base de datos relacionales para luego, seleccionar los que muestran una mejor performance y posteriormente, realizar la comparativa.
- Revisar métodos para la evaluación de algoritmos metaheurísticos que resuelven el problema de selección de índices.

3.3 Preguntas de revisión

El objetivo de esta revisión sistemática es identificar estudios previos sobre el problema de selección de índices en bases de datos relacionales aplicando algoritmos metaheurísticos, con el propósito de obtener información crucial para abordar el proyecto. En este contexto, se busca responder a las siguientes preguntas de investigación:

P1: ¿Qué tipo y cómo funcionan los algoritmos metaheurísticos implementados para atacar el problema de selección de índices?

P2: ¿Cuáles son las variables y/o restricciones más importantes en la selección de índices y de qué manera afectan a los algoritmos para la optimización?

P3: ¿Cómo se calcula la eficacia y la eficiencia de los algoritmos implementados?

Se empleó el criterio PICOC (*Población, Intervención, Comparación, Resultado, Contexto*) para estructurar las preguntas de investigación. En la Tabla 3 se presentan los resultados derivados de la aplicación de este criterio en el marco de la revisión sistemática.

Tabla 3.

Resultados del Criterio PICOC para la Revisión Sistemática

Criterio	**Descripción**
Población	Soluciones al problema de selección de índices en base de datos relacionales.
Intervención	Algoritmos metaheurísticos.
Comparación	Algoritmos metaheurísticos adaptados al problema de selección de índices en base de datos relacionales más utilizados en la actualidad.
Resultados	Publicaciones, artículos o casos de estudio en los cuales se apliquen algoritmos metaheurísticos adaptados al problema de selección de índices en base de datos relacionales.
Contexto	Académico y comercial.

3.4 Métodos de indagación

3.4.1 Motores de búsqueda

Se seleccionaron las bases de datos para la revisión de literatura considerando su diversidad en documentos relacionados al área del proyecto, que abarca Ciencias de la Computación y temas afines. Además, estas bases de datos están disponibles para toda la comunidad de la Pontificia Universidad Católica del Perú. Las bases de datos elegidas fueron las siguientes:

- Scopus
- IEEE Xplore Digital Library
- ScienceDirect

3.4.2 Cadenas de búsqueda a emplear

Para construir la cadena de búsqueda, se establecieron los términos clave y sus sinónimos, siguiendo el criterio PICOC establecido. Dado que la mayoría de la literatura está disponible en inglés en las bases de datos, se optó por trabajar con los términos en este idioma. En la Tabla 4 se enumeran los términos principales junto con sus sinónimos correspondientes:

Tabla 4. Ampliación de los términos claves

Concepto	**Términos afines**
C1: Metaheuristic algorithm	genetic algorithm, bioinspired algorithm, memetic algorithm, Knapsack algorithm, evolutionary algorithm, taboo algorithm.
C2: Index selection problem	Index selection, index optimization, index tuning, indexing.
C3: Relational database	RDBMS, DBMS, Database, Database Systems, Database management systems.

La cadena de búsqueda resultante es la combinación lógica "AND" de los términos clave identificados. Enseguida, se presenta la cadena de búsqueda con el formato adecuado para cada una de las bases de datos seleccionadas:

- **Scopus**

(TITLE-ABS-KEY("relational database") OR TITLE-ABS-KEY(RDBMS) OR TITLE-ABS-KEY(DBMS) OR TITLE-ABS-KEY("database systems") OR TITLE-ABS-KEY("database management systems") OR TITLE-ABS-KEY(database)) AND (TITLE-ABS-KEY("Index selection problem") OR TITLE-ABS-KEY("Index selection") OR TITLE-ABS-KEY("index optimization") OR TITLE-ABS-KEY("index tuning") OR TITLE-ABS-KEY(indexing)) AND ((TITLE-ABS-KEY(metaheuristic) OR TITLE-ABS-KEY(genetic) OR TITLE-ABS-KEY(memetic) OR TITLE-ABS-KEY(bioinspired) OR TITLE-ABS-KEY(Knapsack) OR TITLE-ABS-KEY(evolutionary) OR TITLE-ABS-KEY(taboo)) AND TITLE-ABS-KEY(algorithm)).

- **IEEE Xplore Digital Library**

("relational database" OR RDBMS OR DBMS OR "database systems" OR database OR "database management systems") AND ("index selection problem" OR "index selection" OR "index optimization" OR "index tuning" OR indexing) AND ((metaheuritic OR genetic OR memetic OR bioinspired OR Knapsack OR evolutionary OR taboo) AND algorithm).

- **Science Direct (Dado la restricción de hasta 8 conectores lógicos dentro de la búsqueda avanzada, se utilizaron una cantidad de sinónimos)**

("relational database" OR database) AND ("index selection problem" OR "index selection" OR "index optimization") AND ("metaheuritic algorithm" OR "evolutionary algorithm" OR "bioinspired algorithm" or "genetic algorithm").

3.4.3 Documentos encontrados

Los resultados obtenidos en cada una de las bases de datos tras aplicar la cadena de búsqueda se presentan en la Tabla 5.

Tabla 5. Hallazgos de la Investigación Sistemática

Base de datos	Resultados	Repetidos
Scopus	202	1
IEEE Xplore Digital Library	199	0
Science Direct	49	0
Total	450	1

3.4.4 Criterios de inclusión y exclusión

Se establecen los siguientes criterios de inclusión y exclusión para discernir cuáles de las publicaciones obtenidas en los motores de búsqueda seleccionados son relevantes para la presente investigación.

- **Criterios de inclusión**
  - CI1. Considerar los resultados que involucren el desarrollo de algún algoritmo metaheurístico para abordar el problema de selección de índices.
  - CI2. Incluir los resultados que abarquen el uso de base de datos relacionales, sin importar el motor que se aplique.
  - CI3. Incluir los resultados que impliquen evaluación de factores claves para el problema de selección de índices.

- **Criterios de exclusión**
  - CE1. Descartar los resultados que involucren el desarrollo de algún algoritmo que no pertenezca al tipo metaheurístico para abordar el problema de selección de índices.
  - CE2. Descartar los resultados que estén en un idioma distinto al inglés, español o francés.
  - CE3. Descartar los resultados que involucren otro tipo de técnicas, como la minería de datos o el aprendizaje automático.
  - CE4. Excluir los resultados que abarquen otro tipo de base de datos que no sea relacional como NoSQL, ANYDB o en entornos de big data.
  - CE5. Excluir los resultados que se han publicado antes del año 2000.
  - CE6. Excluir aquellas publicaciones que parten de los índices ya generados.

La Tabla 6 muestra las publicaciones seleccionadas como relevantes, aplicando los criterios de exclusión e inclusión descritos.

Tabla 6. Artículos considerados como adecuados para la Revisión Sistemática

ID	Título del Artículo	Autores	Año
E01	GADIS: A genetic algorithm for database index selection	Neuhaus P., Couto J., Wehrmann J., Ruiz D.D., Meneguzzi F.	2019
E02	Relational Database Index Selection Algorithm	Boronski R., Bocewicz G.	2014
E03	Online index selection in RDBMS by evolutionary approach	Kołaczkowski, P., Rybiński, H.	2011
E04	An interactive tool for automatic index selection in relational database management systems	Kołaczkowski, P., Rybiński, H.	2011
E05	Genetic algorithm for database indexing	Korytkowski, M., Gabryel, M., Nowicki, R., Scherer, R.	2004
E06	Index selection for databases: A hardness study and a principled heuristic solution	Chaudhuri, S., Datar, M., Narasayya, V.	2004
E07	A genetic algorithm for the index selection problem	Kratica, J., Ljubić, I., Tošić, D.	2003
E08	An evolutionary approach to the index selection problem	Calle, J., Sáez, Y., Cuadra, D.	2011
E09	A genetic algorithm with an adaptation mechanism for database index optimization	Kovačević, V., Filipič, B.	2006
E10	Automatic index selection in RDBMS by exploring query execution plan space	Kołaczkowski P., Rybiński H.	2009

3.5 Formulario de extracción

A continuación, se describen los criterios tenidos en cuenta para la configuración del formulario de extracción de datos, con el propósito de incluir los elementos esenciales para abordar las preguntas planteadas en la revisión.

Tabla 7. Planilla de extracción de datos

Información general	Descripción	Pregunta
Identificador del estudio	E[Número], por ejemplo, E01	General
Título	Título de la publicación	General
Autor(es)	Autor(es) de la publicación	General

Tipo de fuente	Informes, tesis, casos de estudio, Libros, entre otros.	General
Año de publicación		General
Base de datos de extracción	Scopus, IEEE, Science Direct, Springer	General
Abstract		General
Algoritmo propuesto	¿Qué tipo es y cómo funciona el algoritmo para atacar el problema?	P1
Variables utilizadas	¿Cuáles son las variables utilizadas en la solución del problema implementado?	P2
Restricciones utilizadas	¿Cuáles son las restricciones utilizadas en la solución del problema implementado?	P2
Criterios para evaluar eficacia y eficiencia	¿Cuáles son los criterios para evaluar la eficacia y eficiencia de los algoritmos?	P3

En el anexo A se encontrará el formulario detallado para la extracción de datos, el cual responde exhaustivamente a todas las interrogantes planteadas.

3.6 Resultados de la revisión

3.6.1 Respuesta a ¿Qué tipo y cómo funcionan los algoritmos metaheurísticos implementados para atacar el problema de selección de índices?

De la bibliografía seleccionada, se pudo observar que tipo de algoritmos se han implementado para resolver el problema de selección de índices desde el año 2000, siendo en su mayoría algoritmos genéticos. Dentro de los estudios encontrados, se pretende mejorar el rendimiento de la base de datos, centrándose en la selección óptima de índices, frente a soluciones comerciales o a otros algoritmos encontrados en la revisión de la literatura.

A continuación, la tabla 8 presenta el tipo de algoritmos encontrados en los artículos relevantes de la revisión sistemática.

Tabla 8. Tipo de algoritmos

Algoritmo	**Cantidad**
Genético	9
Genético y voraz	1

Total	10

Como se puede evidenciar de la tabla, para las soluciones implementadas con algoritmos metaheurísticos, los algoritmos genéticos son mayoría con 9 (Kratica et al., 2003) (Kovačević & Filipič, 2006) (Calle et al., 2011) (Kołaczkowski & Rybiński, 2011a) (Kołaczkowski & Rybiński, 2009) (Neuhaus et al., 2019) (Korytkowski et al., 2004) (Kołaczkowski & Rybiński, 2011b) (Boronski & Bocewicz, 2014), mientras que solo una de las soluciones emplea una combinación de algoritmo genético y voraz (Chaudhuri et al., 2004). Esta predominancia de algoritmos genéticos se puede explicar en su definición, ya que en este tipo de algoritmos regularmente se obtienen soluciones rápidas, cercanas al óptimo, para problemas complejos y puede adaptar su solución a diferentes condiciones externas (Kołaczkowski & Rybiński, 2011b). Por otro lado, la estrategia que utilizan los algoritmo genéticos tiene ventajas sobre otras heurísticas, como lo pueden ser la búsqueda tabú, el ascenso de colinas o la cristalización simulada, ya que tanto la función de adecuación como los espacios requeridos por el problema y la solución pueden cambiar dinámicamente durante la resolución del problema, y con la correcta intensidad en la mutación, la población seguirá los cambios y se llegará a una solución sin necesidad de repetir el proceso. Esto hace que este enfoque resulte ideal para bases de datos autónomas y autoajustables (Kołaczkowski & Rybiński, 2009).

Dado que la mayor parte de los algoritmos son genéticos, se repiten varios conceptos relacionados a estos, como lo son la generación de población inicial, selección, operadores genéticos de mutación o crossover, y la función de adecuación.

3.6.2 Respuesta a ¿Cuáles son las variables y/o restricciones más importantes en la selección de índices y de qué manera afectan a los algoritmos para la optimización?

A partir de la literatura seleccionada, se detallan distintas variables utilizadas en cada solución para el problema de selección de índices. De los parámetros descritos, se pueden clasificar en dos grandes grupos, el primero, variables de entrada e independientes al tipo de algoritmo seleccionado, y el segundo, variables propias del algoritmo implementado.

Para el primer grupo, hay variables que todos emplean ya que se pueden considerar básicas para solucionar este problema, como el workload de querys y las columnas con posibilidad de indexación. Si bien el workload es común en todos los casos, al menos con querys de tipo SELECT, los demás tipos de operaciones no son incluidos o no son detallados (Boronski & Bocewicz, 2014) (Kołaczkowski & Rybiński, 2011b) (Kołaczkowski & Rybiński, 2011a) (Kratica et al., 2003). Del resto de operaciones, como INSERT, DELETE y UPDATE, algunos

estudios solo incluyen una combinación de estas (Neuhaus et al., 2019) (Calle et al., 2011) (Korytkowski et al., 2004), mientras que otros abarcan todas (Kołaczkowski & Rybiński, 2009) (Kovačević & Filipič, 2006) (Chaudhuri et al., 2004). De igual manera, existen variables repetidas, como el tiempo de respuesta (Neuhaus et al., 2019) (Boronski & Bocewicz, 2014) (Korytkowski et al., 2004), espacio de la base de datos (Boronski & Bocewicz, 2014) (Kołaczkowski & Rybiński, 2011b) (Chaudhuri et al., 2004) (Kratica et al., 2003) o del índices (Kołaczkowski & Rybiński, 2011b) (Kołaczkowski & Rybiński, 2011a) (Kratica et al., 2003). Otros autores emplean variables del sistema de administración de base de datos, como el query execution plan (Kołaczkowski & Rybiński, 2011b) (Kołaczkowski & Rybiński, 2009), o las estadísticas relacionadas los querys (Kovačević & Filipič, 2006) (Korytkowski et al., 2004). Un concepto también utilizado es el costo de ejecución del índice (Kołaczkowski & Rybiński, 2011b) (Kołaczkowski & Rybiński, 2009) (Kovačević & Filipič, 2006) (Chaudhuri et al., 2004). Un enfoque particular usado es la clasificación de los índices, en clustered y nonclustered, diferenciándolos para resolver el problema aplicando condiciones particulares para cada caso (Kołaczkowski & Rybiński, 2009) (Chaudhuri et al., 2004). En casos determinados, se emplean variables propias del motor de base de datos seleccionado, como el “Query-per-hour metric” de PostgreSQL, para simplificar la tarea de estimación de costos (Neuhaus et al., 2019), o se toman condiciones de la base de datos, como el tiempo de mantenimiento, índices creados previamente y la usabilidad de estos (Kovačević & Filipič, 2006).

En el segundo grupo, como se precisó en la pregunta anterior, la mayoría de algoritmos seleccionados fueron del tipo genético, por lo que las variables relacionadas a estas también son comunes, como el tamaño de la población inicial (Kołaczkowski & Rybiński, 2011a) (Calle et al., 2011) (Kratica et al., 2003), inclusión de una población elitista (Kratica et al., 2003), métodos y porcentajes de selección y reemplazo respectivamente (Kołaczkowski & Rybiński, 2011a) (Calle et al., 2011), tipo y factores asociados al crossover y mutación (Kołaczkowski & Rybiński, 2011a) (Calle et al., 2011) (Calle et al., 2011), y condiciones de parada (Calle et al., 2011).

Por otro lado, las restricciones en muchos casos no están descritas explícitamente, pero se pueden extraer a partir del método utilizado. Una restricción simple es la exclusión de operaciones de tipo DELETE (Korytkowski et al., 2004) e INSERT (Boronski & Bocewicz, 2014) del workload, evitando todos los casos posibles. Otras restricciones están enfocadas al motor de base de datos, como ser exclusivo para PostgreSQL (Kołaczkowski & Rybiński, 2011a) o que la base de datos sea de código abierto (Kołaczkowski & Rybiński, 2009). Del mismo modo, algunos estudios toman en consideración aspectos generales de la base de datos, como mantenerse en condiciones estáticas (Calle et al., 2011) o la asunción de contar con un espacio ilimitado (Korytkowski et al., 2004). Por último, otra perspectiva es hacia las

condiciones necesarias para ejecutar el algoritmo, como la exclusión de columnas que sean llaves primarias y foráneas, la distribución uniforme para la población inicial (Neuhaus et al., 2019) o que la generación de índices candidatos ya ha sido producida (Chaudhuri et al., 2004).

3.6.3 Respuesta a ¿Cómo se calcula la eficacia y la eficiencia de los algoritmos implementados?

La meta general para la evaluación de los algoritmos es mejorar el rendimiento de la base de datos a través de la selección óptima de índices. Para cuantificar una medida que sirva de apoyo, los autores plantean diferentes enfoques. Uno bastante común es el costo mediante una función propia (Kołaczkowski & Rybiński, 2011b) (Chaudhuri et al., 2004) (Kratica et al., 2003) o empleando métricas proporcionadas por sistema de administración de base de datos (Neuhaus et al., 2019) (Kołaczkowski & Rybiński, 2011a) (Kołaczkowski & Rybiński, 2009) (Kovačević & Filipič, 2006). Otra estadística utilizada es el tiempo de respuesta del workload con los índices seleccionados (Boronski & Bocewicz, 2014) (Calle et al., 2011) y el espacio necesario para crear los índices sugeridos (Neuhaus et al., 2019) (Boronski & Bocewicz, 2014) (Kołaczkowski & Rybiński, 2011b), ambas relacionadas directamente con las variables empleadas dentro de los algoritmos. Ciertas soluciones particulares prefieren analizar directamente los resultados obtenidos, como la cantidad de resultados consistentes (Calle et al., 2011) o similares a lo esperado (Korytkowski et al., 2004), el número de índices creados y el tiempo de ejecución (Boronski & Bocewicz, 2014).

Todas las medidas descritas son comparadas de diversas maneras. Una de ellas, con el fin de garantizar que los resultados esperados tengan validez, se compara contra soluciones comerciales con un gran grado de aceptación, otros índices planteados por expertos en el tema, u otros algoritmos encontrados dentro de la revisión de la literatura. Mediante las métricas descritas anteriormente, se efectúa una comparación estadística usando tablas (Boronski & Bocewicz, 2014) (Kołaczkowski & Rybiński, 2011b) (Kołaczkowski & Rybiński, 2009) y gráficos como de líneas, de barras, histogramas, diagrama de cajas (Neuhaus et al., 2019) (Kołaczkowski & Rybiński, 2011b) (Calle et al., 2011) (Kołaczkowski & Rybiński, 2009) (Chaudhuri et al., 2004), inclusive se realizan pruebas de hipótesis para probar la significancia de los resultados (Calle et al., 2011). Del mismo modo, para evitar posibles errores o sobreestimaciones de los resultados, cada implementación se ejecuta una cantidad mínima de veces.

3.7 Conclusiones

Basándonos en la revisión realizada, llegamos a la conclusión de que existe un dominio de los algoritmos genéticos en la implementación de soluciones para el problema de selección

de índices a lo largo de los años. Esto se debe a su adaptabilidad y ajuste adecuado para este tipo de problema. Por lo mismo, presentan un funcionamiento, a grandes rasgos, similar, teniendo las mismas operaciones genéticas.

De la misma manera, se observa que las variables tienen dos funciones, la primera siendo de entrada agnóstica a cualquier tipo de solución, y la segunda siendo particulares del tipo de algoritmo seleccionado por cada autor. En estos dos grupos se puede notar un factor común entre las variables, como el workload de consultas, así como consideraciones únicas, como métricas propias de un motor particular. En cuanto a las restricciones, si bien no son evidentes en muchos casos, estas sirven para reducir el tamaño del problema y evitar cambios dinámicos difíciles de parametrizar.

Por último, hay varias maneras de medir la eficacia y eficiencia de los algoritmos, entre los que destaca el costo y los tiempos de respuesta. Por otro lado, para que estos resultados sean válidos, se hicieron comparaciones contra otras soluciones, mostrando en la mayoría gráficos estadísticos para apoyarse.

A través de lo revisado se pudo contestar correctamente las preguntas planteadas y se alcanzaron los objetivos de esta revisión, siendo estos comprender los conceptos sobre el problema de selección de índices, identificar las variables y restricciones aplicadas, conocer la eficacia y eficiencia de los algoritmos y los métodos de evaluación aplicados a estos.

Capítulo 4. Definición de Parámetros, Restricciones y Función Objetivo

4.1 Introducción

En este capítulo se persigue el objetivo específico 1, que se centra en "Definir las variables y restricciones requeridas del problema para la elaboración de la función objetivo". Este objetivo se divide en dos resultados alcanzados: en primer lugar, la determinación de las variables y restricciones necesarias para abordar la formulación de la función objetivo; y, en segundo lugar, la formulación de la función objetivo con el fin de facilitar su optimización. El primer resultado implica detallar cada variable y restricción identificada en el problema, mientras que el segundo consiste en la exposición del planteamiento de la función objetivo.

4.2 Resultados obtenidos

4.2.1 Establecimiento de variables y restricciones del problema

En este apartado se especificarán las variables y restricciones encontradas para el problema de selección de índices, relacionado con el resultado esperado 1.

4.2.1.1 Variables identificadas

- Workload de querys

Esta variable representa a la consulta o consultas de entrada a las cuales se desea crear un índice.

$$Q_m$$

- Tablas de base de datos

Esta variable representa a las tablas presentes en la base de datos.

$$T = (T_1, \ldots, T_i, \ldots, T_t)$$

Donde:

- T_i es la i-ésima tabla.
- t es el número de tablas dentro de la base de datos.

- Columnas de base de datos

Esta variable representa a las columnas pertenecientes a una tabla de la base de datos.

$$K = (K_1 \cup \ldots \cup K_i \cup \ldots \cup K_n)$$

Donde:

- $K_i = \{k_{i,1}, \ldots, k_{i,len(i)}\}$ son las columnas pertenecientes a la tabla T_i, por ende, $k_{i,j}$ es la j-ésima columna de la i-ésima tabla y $len(i)$ es la cantidad total de columnas de la tabla T_i.

- Estadísticas de tablas

Se refieren a la información interna de la base de datos relacionada con las tablas y columnas empleadas. En cuanto a las estadísticas, se hará principal hincapié en la frecuencia de uso.

$$\forall k_{i,j}, \exists frec_{i,j}$$

$$0 \leq frec_{i,j} \leq 1$$

4.2.1.2 Restricciones identificadas

- Cantidad de índices por tabla

Esta restricción asegura evitar la sobrecarga de índices a una tabla, dado que una gran cantidad de estos tiene efectos negativos sobre las operaciones de INSERT, UPDATE y DELETE, esto debido a que se debe recalcular el orden del índice al momento de realizar una de estas operaciones. De igual manera, durante la ejecución de una consulta, si se poseen varios índices creados a una misma tabla, el motor de base de datos puede escoger un índice no óptimo en el plan de ejecución.

$$\sum_{l=1}^{m} J_{i,l} \leq \delta, \forall\, J_i \in T_i$$

Donde:

- $J_{i,l}$ es el l-ésimo índice recomendado para la i-ésima tabla.
- m es la cantidad total de índices recomendados.
- δ es una constante no negativa.

De acuerdo con Kratica, Ljubić y Tošić (2003), el número máximo recomendado de índices por cada tabla es de 5.

- Cantidad de columnas por índice

Esta restricción garantiza minimizar el coste de reordenamiento en caso de las operaciones del tipo INSERT, UPDATE y DELETE.

$$\sum_{x=1}^{len(J_i)} k_{i,j} \leq \gamma, \forall\, k_{i,j} \in J_i$$

Donde:

- $len(J_i)$ es la cantidad de columnas pertenecientes al índice J_i.
- γ es una constante no negativa.

De acuerdo con Kratica, Ljubić y Tošić (2003), la cantidad de columnas para índices es regularmente entre 3 y 4.

- Columnas no repetidas

Esta restricción asegura que en caso existan dos o más índices recomendados que posean las mismas columnas, solo se considere uno de ellos.

$$J_o \equiv J_p \rightarrow J_p$$

- Espacio total de índices

Esta restricción garantiza que el espacio total de los índices recomendados no sea mayor al espacio disponible dentro de la base de datos.

$$E(J) < E_{disp}$$

Donde:

- $E(J)$ es el espacio total de los índices recomendados expresado en gb.
- E_{disp} es el espacio máximo disponible de la base de datos expresado en gb.

- Columnas de llave primaria

Esta restricción evita que se formen posibles índices solo con las llaves primarias de la tabla, puesto que estas ya poseen un índice.

4.2.1.3 Enfoques, Medios de Confirmación y Marcadores

La descripción detallada de las variables y restricciones más significativas para la asignación de índices se presenta en el resultado anterior. Para delimitar esta información, se consideró la revisión realizada en el estado del arte, particularmente en respuesta a la segunda pregunta: "¿Cuáles son las variables y/o restricciones más importantes en la selección de índices y de qué manera afectan a los algoritmos para la optimización?". Además, se buscó la validación del resultado con un especialista en bases de datos, cuyo documento firmado de validación se adjunta en el Anexo C.

4.2.2 Elaboración de la Función Objetivo

En este apartado se detallará la relación entre las variables y restricciones que sirven como base para formular la función objetivo, relacionado con el resultado esperado 2.

4.2.2.1 Descripción

La principal meta del desafío relacionado con la selección de índices es encontrar la configuración de un índice de tal manera que minimice el costo, principalmente el tiempo, de una consulta dada. En otros términos, para cada consulta a base de datos, lo que se aspira obtener es un índice que tenga el mejor rendimiento al momento de retornar la información buscada. De este modo, el tiempo de respuesta de la consulta será menor.

Para ello, primero es necesario definir el conjunto de posibles índices en base a las variables identificadas:

$$J_m \subseteq P\left(\bigcup_{i \in \{j_1,\dots,j_b\}} \bigcup_{a=1}^{|KQ_m^i|} VQ_m^i\,(a)\right)$$

Donde:

- J es el índice.
- $P(X)$ es el conjunto potencia de X.
- $j_1, j_2, \dots, j_b$ son el número de tablas presentes en el query Q_m.
- KQ_m^i es el conjunto de columnas de la tabla T_i presentes en el query Q_m.
- $VQ_m^i(a)$ es un conjunto de tuplas de a-elementos formadas por a partir de un conjunto de elementos únicos de KQ_m^i de la siguiente manera:

$$VQ_m^i(a) = \{\, \{(x) \mid x \in KQ_m^i\}\ para\ a = 1, \{(x_1,\dots,x_a) \mid x_1,\dots,x_a \in KQ_m^i; x_1 \neq \dots \neq x_a\}\ para\ a > 1\}$$

$$a \in \{1,\dots,|KQ_m^i|\}$$

A partir de que J_m está compuesto por las columnas con las cuales se construye el índice, el tiempo de ejecución de una consulta Q_m está definido como $t(J_m)$ y que se puede considerar $Q = \{Q_1,\dots,Q_n\}$ un conjunto de consultas, la función objetivo se define como:

$$minimizar \sum_{m=1}^{n} t(J_m) \cdot \left(1 + \alpha \cdot \frac{1}{frec_J}\right) \cdot \left(1 + \beta \cdot \frac{E(J_m)}{E_{disp}}\right) \cdot penalidad_J$$

Donde:

- J es el índice sugerido.

$$J \subseteq P\left(\bigcup_{m=1}^{n} \bigcup_{i \in \{j_1,\dots,j_b\}} \bigcup_{a=1}^{|KQ_m^i|} VQ_m^i\,(a)\right)$$

- $frec_J$ es la frecuencia total del índice sugerido.

$$frec_J = frec_1 + \ldots + frec_x, \forall\, frec_x \in J$$

- α es una constante no negativa definida por el usuario que determina cuánto afecta la frecuencia del índice sugerido a la función objetivo.

$$0 \leq \alpha \leq 1$$

- β es una constante no negativa definida por el usuario que determina cuánto afecta el espacio del índice sugerido a la función objetivo.

$$0 \leq \beta \leq 1$$

- $penalidad_J$ es la penalidad total del índice sugerido. Donde para cada columna presente de la consulta, se le asigna una penalización θ dependiendo si forma parte de las uniones de tablas o filtros de la misma, y otra penalización κ dependiendo si forma parte de las columnas seleccionadas. El cálculo de penalidad total es la multiplicación de la penalización de cada columna.

4.2.2.2 Enfoques, Medios de Confirmación y Marcadores

El resultado previamente presentado aborda la formulación de la función objetivo, incluyendo detalles y las variables implicadas en la misma. Con el propósito de evaluar diversas soluciones utilizando la función objetivo propuesta y asegurarse de que los resultados obtenidos sean coherentes con las expectativas, se llevaron a cabo una serie de pruebas, detalladas en el Anexo D. Similar a la sección anterior, la función objetivo fue revisada y validada por un experto en bases de datos y algoritmia metaheurística. El documento de validación firmado se encuentra en el Anexo E.

4.3 Análisis y reflexión de los resultados

En este apartado, se han logrado los resultados esperados R1 y R2 del proyecto de tesis. El logro de estos resultados permite concretar la propuesta inicial sobre los parámetros y condiciones más relevantes que facilitarán el diseño de las estructuras de datos y el algoritmo en las etapas subsiguientes. La definición de la función objetivo también resulta crucial, ya que posibilita elección del mejor curso de acción para superar la dificultad.

En primer lugar, se abordó la definición de las variables y restricciones identificadas para el problema de selección de índices, teniendo en cuenta los resultados obtenidos de las preguntas durante la revisión del estado del arte. Diversas investigaciones consultadas exploran la aplicación de uno o más algoritmos metaheurísticos al problema de selección de índices. Estas investigaciones detallan los parámetros, restricciones y la formulación de la función objetivo que se consideran en cada caso. El objetivo es encontrar la mejor solución

para el problema de selección de índices, y las referencias primarias proporcionan información valiosa sobre la eficacia de los algoritmos metaheurísticos para este tipo de problemas. De manera similar, se formuló la función objetivo a partir de la información recopilada de las fuentes primarias.

Los resultados presentados se han planteado de manera general, lo que implica que se pueden emplear diversas unidades de medición para las variables descritas, como el tiempo de ejecución, la frecuencia de uso de las columnas o el espacio en disco. Además, tanto las variables como las restricciones pueden aplicarse a cualquier motor de base de datos, siendo aplicables de manera transversal a toda base de datos. No obstante, es importante tener en cuenta que los resultados, es decir, los índices sugeridos, pueden variar según la distribución de la base de datos.

Capítulo 5. Adecuación del algoritmo Lobo Gris

5.1 Introducción

En este capítulo, se procede con la primera etapa del objetivo específico 2, el cual implica: “Adaptar un algoritmo lobo gris que brinde una óptima selección de índices”. Dicho objetivo se compone de tres resultados alcanzados, de los cuales solo se desarrollará el primero de estos, el cual es el pseudocódigo del algoritmo lobo gris propuesto. Para la implementación de dicho resultado, primero se debe definir las estructuras de datos empleadas por el algoritmo lobo gris para luego proceder con el planteamiento del algoritmo en sí.

5.2 Resultados obtenidos

5.2.1 Definición de las estructuras de datos

Estructuras de datos diseñadas para almacenar y gestionar eficientemente la información requerida por el algoritmo lobo gris.

5.2.1.1 Representación de una tabla

La estructura denominada "Tabla" se emplea en el diseño del algoritmo lobo gris para representar una tabla de base de datos. Esta estructura almacena información como el nombre de la tabla, el número que le corresponde dentro de la base de datos y la cantidad de filas. Se propone codificar esta estructura mediante una clase llamada "Tabla", que contendría los atributos mencionados anteriormente. Una representación visual se muestra en la Figura 3.

Nombre de la tabla	Número de tabla	Cantidad de filas
Empleado	2	4500

Figura 3. Estructura de datos - Tabla

5.2.1.2 Representación de una columna

La estructura llamada "Columna" se utiliza en el diseño del algoritmo lobo gris para representar una columna de una base de datos. Esta estructura almacena información como el nombre de la columna, una tupla que indica el número de tabla y columna a la que pertenece, la frecuencia de uso (en porcentaje), la cantidad de bytes del tipo de dato, la penalidad de la columna, si es una llave primaria de su tabla y la probabilidad de elección (valor entre 0 y 1). Este último dato se empleará en el algoritmo y no es un parámetro de entrada. Se planea codificar esta estructura mediante una clase llamada "Columna", que

contendría los siete atributos mencionados anteriormente. Una representación visual se muestra en la Figura 4.

Nombre de la columna	Tupla	Frecuencia de Uso	Cantidad de bytes	Probabilidad de elección	Penalidad	esPk
Apellido	(5, 2)	0.8	1	0.895	1.5	true

Figura 4. Estructura de datos - Columna

5.2.1.3 Representación de un lobo

La estructura llamada "lobo" se emplea en el diseño del algoritmo lobo gris para representar un agente o "lobo" de la población, del cual surge la sugerencia de un índice. Esta estructura almacena información como el query a optimizar, el tiempo de ejecución en segundos, el espacio en GB, la frecuencia y penalidad total de las columnas a indexar, un arreglo de las posibles columnas a indexar y el valor de ajuste (fitness). Se planea codificar esta estructura mediante una clase llamada "Lobo", que contendría los atributos mencionados anteriormente. Una representación visual se muestra en la Figura 5.

Query	Tiempo de ejecución (seg)	Espacio (GB)	Frecuencia total	Penalidad total	Arreglo de columnas	Fitness
Select * from empleado	7	0.16	0.7	2.25	Estructura Columnas	13.463

Figura 5. Estructura de datos - Índice

5.2.2 Diseño del algoritmo Lobo Gris

Se organiza la descripción del diseño del algoritmo lobo gris en secciones correspondientes a los principales métodos que lo componen. Esta estructuración tiene como propósito proporcionar una explicación detallada de cada uno de los procedimientos de manera individual.

5.2.2.1 Algoritmo Lobo Gris

```
1  Funcion algoritmoLoboGris(querys, tablas, columnas, tamanoPoblacion, alpha, beta, eDisp)
2      poblacion ← generarPoblacionInicial(querys, tablas, columnas, tamanoPoblacion, alpha, beta, eDisp)
3      obtenerTresMejoresSoluciones ( poblacion, tablas, alpha, beta, eDisp, alphaWolf, betaWolf, gammaWolf )
4      t ← 1
5      MIENTRAS t ≤ maxIter HACER
6          a ← 2 * (1 - t/maxIter)
7          PARA cada agente EN poblacion HACER
8              agente ← actualizarPosicion(agente, a, alphaWolf, betaWolf, gammaWolf, alpha, beta, eDisp, tablas, poblacion)
9          FIN PARA
10         obtenerTresMejoresSoluciones ( poblacion, tablas, alpha, beta, eDisp, alphaWolf, betaWolf, gammaWolf )
11         t ← t + 1
12     FIN MIENTRAS
13 Fin Funcion
```

Figura 6. Pseudocódigo del algoritmo lobo gris. Fuente: Creado por el autor

Se presenta en la Figura 6 el pseudocódigo principal del algoritmo lobo gris, que recibe ocho parámetros esenciales: los queries o consultas a optimizar, los datos de las tablas y columnas de la base de datos, el tamaño de la población, la cantidad de iteraciones, las constantes alpha y beta, y el espacio en disco disponible.

En el proceso presentado, se genera una población aleatoria, asegurándose de que cada agente producido cumpla con las restricciones del problema (línea 2). Posteriormente, se seleccionan las tres mejores soluciones o lobos (línea 3). Luego, mientras no se cumpla la condición de parada (línea 5), se calcula el valor de "a" (línea 6), conforme al pseudocódigo original del algoritmo lobo gris (Mirjalili et al., 2014), y se actualiza la posición de cada agente en la población (línea 7-9). Finalmente, se actualizan las tres mejores soluciones (línea 10), y el proceso descrito se repite.

5.2.2.2 Generación de la población inicial

```
1  Funcion poblacion ← generarPoblacionInicial(querys, tablas, columnas, tamanoPoblacion, alpha, beta, eDisp)
2      columnas ← obtenerColumnas(querys, tablas, columnas)
3      cont ← 1
4      poblacion ← []
5      MIENTRAS cont ≤ tamanoPoblacion HACER
6          agente ← generarAgente(columnas, querys)
7          SI esValido(agente, poblacion, alpha, beta, eDisp, tablas) ENTONCES
8              agregar(agente, poblacion)
9              cont ← cont + 1
10         FIN SI
11     FIN MIENTRAS
12 Fin Funcion
```

Figura 7. Pseudocódigo de la generación de la población inicial. Fuente: Creado por el autor

La Figura 7 ilustra el proceso de generación de la población inicial, el cual se repite las veces necesarias hasta alcanzar el tamaño deseado.

En primer lugar, se obtienen todas las *columnas* presentes del grupo de consultas (línea 2). Luego, se inicializa el contador y se declara un arreglo vacío (líneas 3 y 4). A continuación, mientras que el contador no llegue al tamaño requerido (línea 5), se genera un nuevo agente

o *lobo* asignando de manera aleatoria una probabilidad de elección a cada *columna* (línea 6) y se verifica que este sea válido en base a las restricciones establecidas (línea 7). En caso si cumpla, se agrega a la población (línea 8) y aumenta el contador (línea 9). Finalmente, se retorna la población generada (línea 1).

5.2.2.3 Método de actualizar posición

```
1  Funcion agente ← actualizarPosicion(agente, a, alphaWolf, betaWolf, gammaWolf, alpha, beta, eDisp, tablas, poblacion)
2      Repetir
3          A1 ← a * (2 * aleatorio(0,1) - 1)
4          A2 ← a * (2 * aleatorio(0,1) - 1)
5          A3 ← a * (2 * aleatorio(0,1) - 1)
6
7          C1 ← 2 * aleatorio(0,1)
8          C2 ← 2 * aleatorio(0,1)
9          C3 ← 2 * aleatorio(0,1)
10
11         X1 ← obtenerProbabilidad(alphaWolf) - A1 * ABS(C1 * obtenerProbabilidad(alphaWolf) - obtenerProbabilidad(agente))
12         X2 ← obtenerProbabilidad(betaWolf) - A2 * ABS(C2 * obtenerProbabilidad(betaWolf) - obtenerProbabilidad(agente))
13         X3 ← obtenerProbabilidad(gammaWolf) - A3 * ABS(C3 * obtenerProbabilidad(gammaWolf) - obtenerProbabilidad(agente))
14
15         agenteNuevo ← (X1 + X2 + X3)/3
16     Hasta Que esValido(agenteNuevo, poblacion, alpha, beta, eDisp, tablas)
17
18     SI obtenerFitness(agenteNuevo) < obtenerFitness(agente) ENTONCES
19         agente ← agenteNuevo
20     FIN SI
21 Fin Funcion
```

Figura 8. Pseudocódigo del método de actualizar posición. Fuente: Creado por el autor

En la Figura 8, se muestra el método de actualizar la posición de los agentes o *lobos*, simulando la caza en manada de lobos grises (Mirjalili et al., 2014).

Al principio, se generan los valores de los vectores A1, A2, A3 (líneas 3-4) y C1, C2, C3 (líneas 7-9), propios del algoritmo, para luego actualizar la posición del agente o *lobo* en función de cada uno de los *lobos alpha*, *beta* y *gamma* correspondientemente, y según la probabilidad de elección de las *columnas* (líneas 11-14), basado en el comportamiento de los lobos grises cuando rodean a sus presas (Mirjalili et al., 2014). A continuación, se obtiene un nuevo agente promediando las posiciones anteriormente computadas (línea 15) y se constata que este nuevo *lobo* sea válido (línea 16), si no se repite todo el proceso anteriormente descrito hasta generar un agente válido. En última instancia, se comprueba si el fitness de este nuevo agente es mejor que el anterior (línea 18), que en caso si cumpla, se actualiza el agente (línea 19) y retorna el mejor agente (línea 1).

5.2.2.4 Control de aberraciones

```
Funcion flagError ← esValido ( agente, poblacion, alpha, beta, eDisp, tablas )
    flagError ← 0

    //Cantidad de índices por tabla
    indiceTablas ← []
    obtenerCantidadIndicesPorTabla(agente, indiceTablas)
    PARA cada cantTabla EN indiceTablas HACER
        SI cantTabla > 5 ENTONCES
            flagError ← 1
            RETORNAR flagError
        FIN SI
    FIN PARA

    //Cantidad de columnas por índice
    indiceColumnas ← []
    obtenerCantidadColumnasPorIndice(agente, indiceColumnas)
    PARA cada cantColumnas EN indiceColumnas HACER
        SI cantColumnas > 4 ENTONCES
            flagError ← 1
            RETORNAR flagError
        FIN SI
    FIN PARA

    calcularFitness(agente, tablas, alpha, beta, eDisp)

    //Columnas no repetidas
    columnasAgente ← seleccionarColumnas(agente)
    PARA cada agente EN poblacion HACER
        columnasAgente ← seleccionarColumnas(agente)
        SI verificarIgualdad(columnasAgente, columnasAgente) ENTONCES
            flagError ← 1
            RETORNAR flagError
        FIN SI
    FIN PARA

    //Espacio total
    SI obtenerEspacio(agente) > eDisp ENTONCES
        flagError ← 1
        RETORNAR flagError
    FIN SI

    //EsPk
    indiceColumnasPk ← []
    obtenerCantidadColumnasPkPorIndice(agente, indiceColumnasPk)
    Para i←1 Hasta longitud(indiceColumnasPk) Con Paso i<-i+1 Hacer
        SI indiceColumnasPk[i] == indiceColumnas[i] Y indiceColumnas[i] ≠ 0 ENTONCES
            flagError ← 1
            RETORNAR flagError
        FIN SI
    Fin Para

Fin Funcion
```

Figura 9. Pseudocódigo del control de aberraciones. Fuente: Creado por el autor

En la Figura 9, se muestra el método del control de aberraciones, en base a las restricciones planteadas en el capítulo 4.

Al principio, se crea una bandera de control para identificar si existe un error (línea 2). Primero, se valida la restricción sobre la cantidad de índices por tabla (líneas 4-12). Se declara un arreglo "*tablas*" donde se almacenan en cada posición la cantidad de índices por cada tabla de base de datos (líneas 5-6). Luego, por cada cantidad dentro de "*tablas*" (línea 8) se verifica que esta no sea mayor a 5 (línea 7), en caso se cumpla la condición, se cambia el estado de la bandera (línea 9) y se retorna el valor (línea 10). Segundo, se comprueba la restricción sobre la cantidad de *columnas* por índice (líneas 14-22). Se declara un arreglo "*columnas*" donde se guarda en cada posición la cantidad de columnas por cada índice posible a crearse (líneas 15-16). Posteriormente, por cada cantidad dentro de "*columnas*" (línea 17) se cerciora que esta no sea mayor a 4, si se cumple la condición (línea 18), se cambia el estado de la bandera (línea 19) y se retorna el valor (línea 20). Tercero, se calcula el fitness del agente (línea 24) que simplificará el cálculo de las próximas restricciones. Cuarto, se constata la restricción sobre la cantidad de *columnas* (líneas 26-34). Se seleccionan las posibles columnas para crear los índices (línea 27). Posteriori, por cada agente o *lobo* dentro de la población (línea 28), se obtienen igualmente sus columnas a indexar (línea 29) y se verifica que una no esté incluida dentro de otra, o en otras palabras, que ya no esté dentro de la población, en caso se cumpla la condición (línea 30), se cambia el estado de la bandera (línea 31) y se retorna el valor (línea 32). Quinto, se asegura la restricción sobre el espacio total (líneas 36-40). Se examina que el espacio del posible índice a crear no sea mayor al disponible, en el supuesto que cumpla la condición (línea 37), se cambia el estado de la bandera (línea 38) y se retorna el valor (línea 39). Sexto, se valida la restricción sobre las columnas de llave primaria (líneas 43-50). Se declara un arreglo "indiceColumnasPk" donde se almacenan en cada posición la cantidad de columnas que son llave primaria por cada índice posible a crearse (líneas 43-44). Luego, se itera el arreglo creado (línea 45) se verifica que la cantidad de columnas que son llave primaria no sea igual a la cantidad de columnas total del índice y este último no sea igual 0 (línea 46), en caso se cumpla la condición, se cambia el estado de la bandera (línea 47) y se retorna el valor (línea 48). Al final, en caso el agente cumpla todas las condiciones, se retorna la bandera de manera correcta (línea 1).

5.2.2.5 Cálculo de la función fitness

```
1  Funcion calcularFitness ( agente, tablas, alpha, beta, eDisp )
2      columnasAgente ← seleccionarColumnas(agente)
3
4      calcularFrecuencia(agente, columnasAgente)
5      calcularEspacio(agente, columnasAgente, tablas)
6      calcularTiempo(agente, columnasAgente, tablas)
7      calcularPenalidad(agente, columnasAgente)
8
9      agente ← obtenerTiempo(agente) * (1 + alpha * 1 / obtenerFrecuencia(agente)) * (1 + beta * obtenerEspacio(agente) / eDisp) * obtenerPenalidad(agente)
10 Fin Funcion
```

Figura 10. Pseudocódigo del cálculo de la función fitness. Fuente: Creado por el autor

En la Figura 10, se muestra el método del cálculo de la función fitness.

Para empezar, se seleccionan las columnas que se usarán para construir el índice o índices (línea 2). Luego, en base a dichas columnas, se calcula la frecuencia (línea 4), el espacio (línea 5), el tiempo (línea 6) y la penalidad (línea 7). En la fase final, se aplica la fórmula de la función objetivo planteada en el capítulo 4 (línea 9).

5.2.2.6 Obtención de las tres mejores soluciones

```
1  Funcion obtenerTresMejoresSoluciones ( poblacion, tablas, alpha, beta, eDisp, alphaWolf, betaWolf, gammaWolf )
2      PARA cada agente EN poblacion HACER
3          calcularFitness(agente, tablas, alpha, beta, eDisp)
4      FIN PARA
5
6      orderPorFitness(poblacion)
7      alphaWolf ← poblacion[1]
8      betaWolf ← poblacion[2]
9      gammaWolf ← poblacion[3]
10 Fin Funcion
```

Figura 11. Pseudocódigo de la obtención de las tres mejores soluciones. Fuente: Creado por el autor

En la Figura 11, se muestra el método de obtención de las tres mejores soluciones.

Como primeros pasos, para cada individuo en la población (línea 2) se calcula el fitness del agente o *lobo* (línea 3). Después, se ordena la población en base al fitness calculado (línea 6). Luego, se seleccionan las tres mejores soluciones, en otras palabras, los *lobos alpha*, *beta* y *gamma* (líneas 7-9).

5.2.2.7 Método de selección de columnas

```
1  Funcion columnasSeleccionadas ← seleccionarColumnas ( agente )
2      columnasSeleccionadas ← []
3      PARA cada columna EN agente HACER
4          SI obtenerFrecuencia(columna) ≥ 0.75 ENTONCES
5              agregar(columna, columnasSeleccionadas)
6          FIN SI
7      FIN PARA
8  Fin Funcion
```

Figura 12. Pseudocódigo del método de selección de columnas. Fuente: Creado por el autor

En la Figura 12, se muestra el método de selección de columnas de un agente.

En primer lugar, se declara un arreglo para almacenar las *columnas* (línea 2). Seguidamente, para cada *columna* dentro del agente (línea 3), se verifica que la frecuencia sea mayor o igual 0.75 (línea 4), valor mínimo para que se considere en la construcción de un índice y el cual se calibrará en el Capítulo 6. En caso cumpla la condición, se añade al arreglo (línea 5). Finalmente, se retorna el arreglo con las columnas escogidas (línea 1).

5.2.3 Enfoques, Medios de Confirmación y Marcadores

Los resultados obtenidos se detallan en la sección de diseño del algoritmo lobo gris en este capítulo. En este apartado se ofrece una descripción detallada del diseño del algoritmo lobo gris y sus principales métodos, los cuales servirán como base para la implementación de la solución propuesta en el proyecto de tesis.

Adicionalmente, en el Anexo F se detallan las pruebas de flujo de datos efectuadas. Estas pruebas comprenden la representación de los estados del algoritmo diseñado y se llevan a cabo con el fin de identificar posibles errores en su diseño mediante simulaciones.

Al igual que los resultados anteriores, este ha sido sometido a revisión y validación por un especialista en algoritmia metaheurística, quien ha determinado que las estructuras de datos, como el pseudocódigo del algoritmo propuesto para el algoritmo lobo, son apropiadas. El documento de validación firmado se encuentra disponible en el Anexo G.

5.3 Análisis y reflexión de los resultados

Se ha logrado alcanzar el resultado esperado R3 del proyecto de tesis, que consiste en elaborar el pseudocódigo del algoritmo lobo gris para abordar el problema de selección de índices. La consecución de este resultado proporciona una comprensión detallada del funcionamiento del algoritmo lobo gris y establece una sólida base de diseño que servirá como fundamento durante la fase de codificación del algoritmo.

En un primer paso, se desarrolla la definición de las estructuras de datos requeridas para la implementación del algoritmo lobo gris. Estas estructuras se diseñan con el propósito de organizar eficientemente los diversos datos, facilitando su uso en la implementación del algoritmo. La sección de definición de variables y restricciones del capítulo 4 sirve como referencia principal para la elaboración de este resultado. Asimismo, se tiene en cuenta la naturaleza específica del algoritmo lobo gris, que introduce la noción de una estructura "agente" o "lobo". Este componente es fundamental para el algoritmo y contiene la solución

que se busca optimizar, imitando la jerarquía y el proceso de caza de los lobos grises (Mirjalili et al., 2014).

En un segundo plano, se aborda el diseño del algoritmo lobo gris y sus métodos principales, que constituyen las funciones clave a implementar. Este enfoque facilita la comprensión de las características individuales de cada método y contribuye a reducir posibles errores durante la fase de implementación. Además, se llevó a cabo un conjunto de pruebas de flujo de datos para simular el recorrido de ejecución del algoritmo, proporcionando así una evaluación anticipada de su funcionalidad y detección de posibles fallos.

En contraste, no se encontraron investigaciones previas que hayan utilizado el algoritmo lobo gris, según lo revelado por la revisión de la literatura, para abordar el problema de selección de índices en bases de datos relacionales. La mayoría de las investigaciones se centraron principalmente en algoritmos genéticos. Por lo tanto, se anticipa que esta adaptación del algoritmo lobo gris representará una contribución al estado del arte en la selección de índices.

Capítulo 6. Codificación del algoritmo Lobo Gris

6.1 Introducción

En el presente capítulo se desarrolla la parte restante el objetivo específico 2, el cual es: “Adaptar un algoritmo lobo gris que brinde una óptima selección de índices”. Dicho objetivo se compone de tres resultados alcanzados, de los cuales se desarrollará los dos restantes, que son la “Codificación del algoritmo lobo gris adaptado al problema de selección de índices” y la “Calibración de variables y constantes del algoritmo lobo gris”. Para la implementación de dichos resultados, se tomó como base los diseños descritos en el capítulo anterior, como las estructuras de datos y el pseudocódigo.

Se presentarán los métodos principales del mismo similarmente que en el diseño del algoritmo, considerando sobre todo aspectos técnicos y consideraciones sobre situaciones presentadas en el proceso de desarrollo.

6.2 Resultados obtenidos

6.2.1 Codificación del algoritmo lobo gris adaptado al problema de selección de índices

En esta sección se detalla el proceso llevado a cabo para cumplir con el resultado esperado R4. Para alcanzar este resultado, se requiere disponer de una base de datos sobre la cual realizar las pruebas. Por lo tanto, se tomaron como referencia tres diseños previamente implementados. El primero es de "northwindextended" (Google Code Archive - Long-Term Storage for Google Code Project Hosting., s.f.) diseñado para MySQL. El segundo es de "employees" (MySQL :: Employees Sample Database, s.f.) proporcionado por MySQL como muestra, con un total de 4 millones de registros. Y el tercero corresponde a "chinook" (Lerocha/Chinook-Database: Sample Database for SQL Server, Oracle, MySQL, PostgreSQL, SQLite, DB2, s.f.), también diseñado para MySQL. Se seleccionaron estos tres ejemplos de bases de datos debido a sus tamaños variados, lo que permitirá evaluar el rendimiento del algoritmo en entornos diversos.

6.2.1.1 Archivos de entrada

El algoritmo lobo gris espera tres archivos de entrada, dos en formato CSV y otro SQL. Los dos primeros son sobre la información de las tablas y columnas de la base de datos. Estas corresponden a las Estructuras de datos planteadas para el algoritmo lobo gris en el capítulo anterior.

```
nombreTabla,numeroTabla,cantidadFilas
categories,1,8
customercustomerdemo,2,0
customerdemographics,3,0
customers,4,93
employees,5,9
employeeterritories,6,49
`order details`,7,2155
orders,8,830
products,9,77
region,10,4
shippers,11,3
suppliers,12,29
territories,13,53
```

Figura 13. Modelo de archivo csv sobre las tablas de base de datos. Fuente: Creado por el autor

La estructura del archivo que contiene información sobre las tablas se muestra en la Figura 13. Dicho archivo presenta la siguiente disposición: nombre de la tabla, número de la tabla (correlativo) y la cantidad de filas de dicha tabla.

```
nombreColumna,numeroTabla,numeroColumna,frecuenciaUso,cantidadBytes,esPK
CategoryID,1,1,0.015267176,4,1
CategoryName,1,2,0.038167939,17,0
Description,1,3,0.001908397,16777215,0
Picture,1,4,0.005725191,4294967295,0
CustomerID,2,1,0,7,1
CustomerTypeID,2,2,0,12,1
CustomerTypeID,3,1,0,12,1
CustomerDesc,3,2,0,16777215,0
CustomerID,4,1,0.015267176,7,1
CompanyName,4,2,0.030534351,42,0
ContactName,4,3,0.003816794,32,0
ContactTitle,4,4,0.001908397,32,0
Address,4,5,0.007633588,62,0
City,4,6,0.028625954,17,0
Region,4,7,0.022900763,17,0
PostalCode,4,8,0.022900763,12,0
Country,4,9,0.007633588,17,0
Phone,4,10,0.005725191,26,0
Fax,4,11,0,26,0
```

Figura 14. Modelo de archivo csv sobre las columnas de base de datos. Fuente: Creado por el autor

Como se aprecia en la Figura 14, el archivo sobre las columnas tiene la siguiente estructura: nombre la columna, número de la tabla que corresponde, número de la columna (correlativo), la frecuencia de uso (valor entre 0 y 1), cantidad de bytes del tipo de dato y si es la llave primaria de su tabla (valor binario de 0 o 1).

```
SELECT Categories.CategoryName,
       Products.ProductName,
       Products.QuantityPerUnit,
       Products.UnitsInStock,
       Products.Discontinued
FROM Categories
     INNER JOIN Products ON Categories.CategoryID = Products.CategoryID
WHERE Products.Discontinued <> 1;
```

Figura 15. Modelo de archivo sql. Fuente: Creado por el autor

Por último, el archivo SQL posee dos consideraciones. En primer lugar, no usar alias para las tablas y redactar las columnas de la siguiente manera: “nombreTabla.nombreColumna” como se aprecia en la Figura 15. Esto con el fin de que el algoritmo pueda detectar las columnas presentes en las consultas. En segundo lugar, las operaciones SQL diferentes a los “JOINS” y “WHERE” como pueden ser “HAVING”, “GROUP BY”, “ORDER BY” no son admisibles, debido a que dichas operaciones se realizan posterior a la búsqueda en disco de la base de datos y el enfoque de este proyecto de tesis es que la optimización mediante la selección de índices sea a nivel físico de la base de datos, es decir, a nivel de disco. Si bien las operaciones excluidas influyen en el rendimiento, escapan al Alcance planteado.

6.2.1.2 Método principal del Algoritmo Lobo Gris

```java
public class AlgoritmoLoboGris {
    public static void main(String[] args) {
        String filenameTablas = "tablas_"+Constante.DATABASE_SELECTED+"_mysql.csv",
                filenameColumnas = "columnas_"+Constante.DATABASE_SELECTED+"_mysql.csv",
                filenameQuery = "query_"+Constante.DATABASE_SELECTED+".sql";

        Lector lector = new Lector( rutaArchivoTablas: Constante.PATH_INPUT_CSV + filenameTablas,
                 rutaArchivoColumnas: Constante.PATH_INPUT_CSV + filenameColumnas,
                Path.of( first: Constante.PATH_INPUT_CSV + filenameQuery));
        lector.leerArchivos();

        int tamanoPoblacion = lector.getColumnasQuery().size() / 2, t = 1, maxIter = 500, sinMejora = 0;
        double alpha = 0.5, beta = 0.5, eDisp = 1000000, a, startTime, endTime;
        Lobo alphaWolf;
        startTime = System.nanoTime();
        Poblacion poblacion = new Poblacion(tamanoPoblacion);
        poblacion.crearPoblacion(lector, eDisp);
        poblacion.seleccionarTresMejoresSoluciones(lector, alpha, beta, eDisp);
        while (t <= maxIter){
            System.out.println("Iteración n°"+t);
            if (sinMejora == Math.max(30, (int) (maxIter*0.3)))
                break;
            a = 2 * (1 - t/maxIter);
            alphaWolf = new Lobo(poblacion.getAlphaWolf());
            poblacion.actualizarPosicion(a, lector, alpha, beta, eDisp);
            poblacion.seleccionarTresMejoresSoluciones(lector, alpha, beta, eDisp);
            if (alphaWolf.mismasColumnasSeleccionadas(poblacion.getAlphaWolf()))
                sinMejora++;
            else
                sinMejora = 0;
            t++;
        }
        poblacion.printMejorSolucion();
        endTime = (System.nanoTime() - startTime)/1000000000;
        System.out.println("Cantidad de iteraciones: " + t);
        System.out.println("Tiempo total: " + endTime + " segundos.");
    }
}
```

Figura 16. Codificación del algoritmo lobo gris. Fuente: Creado por el autor

En la Figura 16, se puede observar el código del método principal del algoritmo lobo gris. En este método, se establecen inicialmente las rutas de los archivos de entrada que contienen la información sobre las tablas, columnas y consultas de la base de datos. Estos archivos son leídos por el objeto "lector". Luego, se definen cinco parámetros esenciales: el tamaño de la población, la cantidad de iteraciones, las constantes alpha y beta, y el espacio en disco disponible. Se ha introducido una nueva variable que registra el número de iteraciones consecutivas sin mejoras, lo que sugiere la selección repetida de las mismas columnas. Esta adición busca evitar que el algoritmo continúe su ejecución cuando ha transcurrido un número considerable de iteraciones sin que se observe progreso en las soluciones.

6.2.1.3 Generación de la población inicial

```
public void crearPoblacion(Lector lector, double eDisp){
    if (lector == null)
        throw new InvalidParameterException(Constante.INVALID_PARAMETER_MSG);
    if (eDisp < 0)
        throw new InvalidParameterException(Constante.NEGATIVE_PARAMETER_MSG);
    List<Columna> columnasSel = lector.getColumnasQuery();
    int cont = 0;
    List<Lobo> p = new ArrayList<>();
    while (cont < this.tamanoPoblacion){
        Lobo nuevoLobo = new Lobo(lector.getQuerys(), columnasSel);
        if (nuevoLobo.esValido(p, eDisp, lector.getTablas())){
            p.add(new Lobo(nuevoLobo));
            cont++;
        }
    }
    this.poblacion = p;
}
```

Figura 17. Codificación de la generación de la población inicial. Fuente: Creado por el autor

En la Figura 17 se observa el código de cómo se genera la población inicial. En primer lugar, se valida que no se reciba parámetros nulos o negativos, para luego obtener las columnas involucradas dentro del query a optimizar. Esta lista está conformada con los datos internos de la base de datos que sirven para el cálculo de la función objetivo. Luego se itera hasta que se alcance la población deseada. Cuando se genera un nuevo lobo en base a las columnas mencionadas anteriormente y el query, por cada columna se le asigna una probabilidad de elección de manera aleatoria.

```
public Lobo(String querys, List<Columna> columnasSeleccionadas) {
    if (querys == null || columnasSeleccionadas == null)
        throw new InvalidParameterException(Constante.INVALID_PARAMETER_MSG);
    if (columnasSeleccionadas.isEmpty())
        throw new InvalidParameterException(Constante.EMPTY_LIST_PARAMETER_MSG);

    this.querys = querys;
    for(int i=0; i<columnasSeleccionadas.size(); i++){
        columnasSeleccionadas.get(i).setProbabilidadEleccion();
    }
    this.columnas = columnasSeleccionadas;
    this.fitness = 1000000000;
}
```

Figura 18. Codificación de la construcción de un nuevo lobo. Fuente: Creado por el autor

Después se confirma que este nuevo lobo sea válido para que pueda ser agregado a la población.

6.2.1.4 Actualizar posición de la población

```
public void actualizarPosicion(double a, Lector lector, double alpha, double beta, double eDisp){
    if (lector == null)
        throw new InvalidParameterException(Constante.INVALID_PARAMETER_MSG);
    if (a < 0 || alpha < 0 || beta < 0 || eDisp < 0)
        throw new InvalidParameterException(Constante.NEGATIVE_PARAMETER_MSG);
    int cantCols = this.alphaWolf.getColumnas().size();
    Boolean valido;
    Lobo nuevoLobo;
    for (int i = 0; i<this.poblacion.size(); i++){
        System.out.println("Lobo n°" + i);
        do {
            double A1 = a * (2 * Math.random() - 1), A2 = a * (2 * Math.random() - 1), A3 = a * (2 * Math.random() - 1);
            double C1 = 2 * Math.random(), C2 = 2 * Math.random(), C3 = 2 * Math.random();

            double[] X1 = new double[cantCols], X2 = new double[cantCols], X3 = new double[cantCols], XNuevo = new double[cantCols];
            for (int j = 0; j<cantCols; j++){
                X1[j] = this.alphaWolf.getColumnas().get(j).getProbabilidadEleccion() -
                        A1 * Math.abs(C1 * this.alphaWolf.getColumnas().get(j).getProbabilidadEleccion() -
                                this.poblacion.get(i).getColumnas().get(j).getProbabilidadEleccion());
                X2[j] = this.betaWolf.getColumnas().get(j).getProbabilidadEleccion() -
                        A2 * Math.abs(C2 * this.betaWolf.getColumnas().get(j).getProbabilidadEleccion() -
                                this.poblacion.get(i).getColumnas().get(j).getProbabilidadEleccion());
                X3[j] = this.gammaWolf.getColumnas().get(j).getProbabilidadEleccion() -
                        A3 * Math.abs(C3 * this.gammaWolf.getColumnas().get(j).getProbabilidadEleccion() -
                                this.poblacion.get(i).getColumnas().get(j).getProbabilidadEleccion());
                XNuevo[j] += X1[j] + X2[j] + X3[j];
            }
            for (int j = 0; j<cantCols; j++){
                XNuevo[j] = Math.min(Math.max(XNuevo[j]/3, 0), 1);
            }
            nuevoLobo = new Lobo(this.poblacion.get(i));
            nuevoLobo.updatePosicion(XNuevo);

            valido = nuevoLobo.esValido(this.poblacion, eDisp, lector.getTablas());
        } while (!valido);

        nuevoLobo.setFitness(lector.getTablas(), alpha, beta, eDisp);
        if (nuevoLobo.getFitness() < this.poblacion.get(i).getFitness() ){
            this.poblacion.set(i, nuevoLobo);
        }
    }
```

Figura 19. Codificación de la actualización de posición de la población. Fuente: Creado por el autor

En la Figura 19 se observa el código de cómo se actualiza la posición de la población. Primero, se valida que no se reciba parámetros nulos o negativos, para luego iterar toda la población. Por cada lobo, se generarán las variables A1, A2, A3, C1, C2, C3 aleatoriamente y se creará un arreglo de tamaño de las columnas involucradas, las cuales poseerán las nuevas posiciones en base a los lobos alpha, beta y gamma respectivamente. Seguidamente, se promedian las tres posiciones entre el rango de 0 y 1. Después, se creará un nuevo lobo con las nuevas posiciones. En caso que el nuevo lobo sea válido, se calculará su fitness y si es menor al lobo iterado, se reemplazará con este nuevo lobo. Si el nuevo lobo no es válido, se repetirá el proceso hasta encontrar una configuración correcta.

6.2.1.5 Control de aberraciones

```
public Boolean esValido (List<Lobo> poblacion, double eDisp, List<Tabla> tablas){
    Boolean flagValido = true;
    this.setColumnasSeleccionadas();

    if (this.columnasSeleccionadas.size() == 0){
        flagValido = false;
        return flagValido;
    }

    //Cantidad de indices por tabla
    int[] indiceTablas = new int[tablas.size()];
    int count;
    for (int i=0; i<tablas.size(); i++){
        count = 0;
        for(Columna col : this.columnasSeleccionadas){
            if(tablas.get(i).getNumeroTabla() == col.getTuplaTabla()){
                count ++;
                break;
            }
        }
        indiceTablas[i] = count;
    }
    for(int cantTabla : indiceTablas){
        if (cantTabla >= 5){
            flagValido = false;
            return flagValido;
        }
    }
```

Figura 20. Codificación del control de aberraciones, parte 1. Fuente: Creado por el autor

En la Figura 20 se observa la primera parte del código del control de aberraciones. Antes de empezar con las restricciones, se seleccionan las columnas con las que se va construir el índice en base a las probabilidades de elección. En caso no se seleccione ninguna, se considera como inválido. Empezando con las restricciones, primero se crea un arreglo con la cantidad de índices creado por cada tabla de la base datos y se verifica que no se creen más de 5 índices por tabla.

```
//Cantidad de columnas por indice
int[] indiceColumnas = new int[tablas.size()];
for (int i=0; i<tablas.size(); i++){
    count = 0;
    for(Columna col : this.columnasSeleccionadas){
        if(tablas.get(i).getNumeroTabla() == col.getTuplaTabla()){
            count ++;
        }
    }
    indiceColumnas[i] = count;
}
for(int cantColumnas : indiceColumnas){
    if (cantColumnas > 4){
        flagValido = false;
        return flagValido;
    }
}

//Columnas no repetidas
for(Lobo agente : poblacion){
    if (agente.mismasColumnasSeleccionadas( otro: this) || this.mismasColumnasSeleccionadas(agente)){
          System.out.println("Mismas columnas");
        flagValido = false;
        return flagValido;
    }
}

this.setEspacio(tablas);

//Espacio total
if (this.espacio >= eDisp){
    flagValido = false;
    return flagValido;
}
```

Figura 21. Codificación del control de aberraciones, parte 2. Fuente: Creado por el autor

En la Figura 21 se observa la segunda parte del código del control de aberraciones. Continuando con las restricciones, se crea otro arreglo con la cantidad de columnas por cada posible índice y se verifica que no sean más de 5 columnas por columna. Seguidamente, se comprueba que las columnas seleccionadas por este nuevo posible lobo no sean iguales a otro dentro de la población. En caso sean iguales, se considera no válido. A continuación, se calcula el espacio de los posibles índices para cerciorarse que no sea mayor al espacio total en disco.

```
//Solo PK
int total;
for (int i=0; i<tablas.size(); i++){
    total = 0;
    for(Columna c : this.columnasSeleccionadas){
        if (tablas.get(i).getNumeroTabla() == c.getTuplaTabla() && c.isEsPkFk())
            total++;
    }
    if (indiceColumnas[i] != 0 && total == indiceColumnas[i]){
          System.out.println("Solo pk");
        flagValido = false;
        return flagValido;
    }
}

return flagValido;
```

Figura 22. Codificación del control de aberraciones, parte 3. Fuente: Creado por el autor

En la Figura 22 se observa la tercera parte del código del control de aberraciones. Finalmente, se constata que las columnas seleccionadas solo están compuestas por las llaves primarias de la tabla, para evitar se genere un índice solo con la llave primaria.

6.2.1.6 Cálculo de la función fitness

```
public void setFitness(List<Tabla> tablas, double alpha, double beta, double eDisp) {
    this.setEspacio(tablas);
    this.setFrecuenciaTotal();
    this.setPenalidadTotal();
    this.setTiempoEjecucion(tablas);

    double factorFrecuencia, factorEspacio = (1 + beta * this.espacio / eDisp);
    if (this.frecuenciaTotal == 0)
        factorFrecuencia = 1;
    else
        factorFrecuencia = (1 + alpha * 1 / this.frecuenciaTotal);
    double fit = this.tiempoEjecucion * factorFrecuencia * factorEspacio * this.penalidadTotal;
    this.fitness = fit;
}
```

Figura 23. Codificación del cálculo de la función fitness. Fuente: Creado por el autor

En la Figura 23 se observa el código del cálculo de la función fitness. Primero, se calculan el espacio, frecuencia, penalidad y tiempo de ejecución del índice. Posteriormente, se calculan los factores respectivos y se aplica la función objetiva planteada en el capítulo 4.

```java
public void setEspacio(List<Tabla> tablas) {
    if (tablas == null)
        throw new InvalidParameterException(Constante.INVALID_PARAMETER_MSG);
    if (tablas.isEmpty())
        throw new InvalidParameterException(Constante.EMPTY_LIST_PARAMETER_MSG);
    if (this.columnasSeleccionadas == null)
        throw new InvalidParameterException(Constante.INCONSISTENT_PARAMETER_MSG);
    double sum = 0;
    double espTab;
    for (int i=0; i<tablas.size(); i++){
        espTab = 0;
        for(Columna col : this.columnasSeleccionadas){
            if(i + 1 == col.getTuplaTabla()){
                espTab += col.getCantidadBytes();
            }
        }
        if(espTab != 0){
            espTab += 11; //Factor de overhead
            espTab *= tablas.get(i).getCantidadFilas();
            sum += espTab;
        }
    }
    sum = sum * 2 / 1000000000;
    this.espacio = sum;
}
```

Figura 24. Codificación del cálculo del espacio de índice. Fuente: Creado por el autor

En la Figura 24 se observa la primera parte del código del espacio que ocuparía el índice. Para la fórmula aplicada se tomó como referencia lo establecido por IBM (Space Requirements for Indexes - IBM Documentation, s.f.) el cual es:

$$(tamaño\ medio\ del\ índice\ +\ sobrecarga\ de\ clave\ de\ índice)\ \times\ número\ de\ filas\ \times\ 2$$

El tamaño medio del índice se calcula a partir del espacio en bytes de las columnas que lo forman, y la sobrecarga de acuerdo con la documentación, para cualquier tipo de tabla es de 11 bytes. El número de filas depende de la tabla a la cual pertenecen las columnas a indexar. Por lo mismo, dentro del código primero se validan los parámetros de entrada, para luego por cada tabla obtener espacio total en bytes de las columnas del índice, se suma el factor de sobrecarga y se multiplica por la cantidad de filas. Cuando ya se posee el total por cada tabla, recién se multiplica por 2 y se divide entre 1,000,000,000 para convertirlo a gigabytes.

```java
public void setFrecuenciaTotal() {
    if (this.columnasSeleccionadas == null)
        throw new InvalidParameterException(Constante.INCONSISTENT_PARAMETER_MSG);
    double sum = 0;
    for(Columna col : this.columnasSeleccionadas){
        sum += col.getFrecuenciaUso();
    }
    this.frecuenciaTotal = sum;
}
```

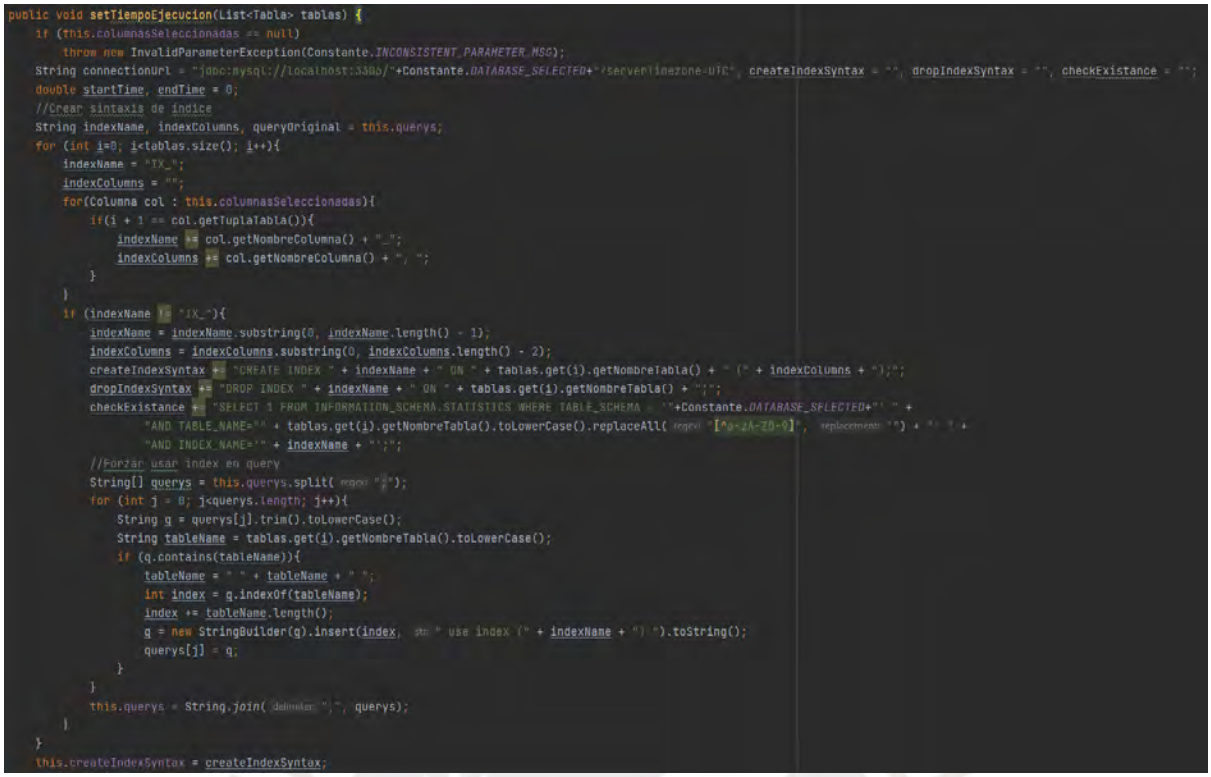
Figura 25. Codificación del cálculo de la frecuencia total de índice. Fuente: Creado por el autor

En la Figura 25 se muestra el código correspondiente al cálculo de la frecuencia total del índice. Esta es la sumatoria de la frecuencia de cada columna que conforma el índice.

```java
public void setPenalidadTotal() {
    if (this.columnasSeleccionadas == null)
        throw new InvalidParameterException(Constante.INCONSISTENT_PARAMETER_MSG);
    double sum = 1;
    for(Columna col : this.columnasSeleccionadas){
        sum *= col.getPenalidad();
    }
    this.penalidadTotal = sum;
}
```

Figura 26. Codificación del cálculo de la penalidad total de índice. Fuente: Creado por el autor

En la Figura 26 se presenta el código relacionado con el cálculo de la frecuencia total del índice. Esta es la multiplicación de la penalidad de cada columna que conforma el índice.



```
public void setTiempoEjecucion(List<Tabla> tablas) {
    if (this.columnasSeleccionadas == null)
        throw new InvalidParameterException(Constante.INCONSISTENT_PARAMETER_MSG);
    String connectionUrl = "jdbc:mysql://localhost:3306/"+Constante.DATABASE_SELECTED+"?serverTimezone=UTC", createIndexSyntax = "", dropIndexSyntax = "", checkExistance = "";
    double startTime, endTime = 0;
    //Crear sintaxis de indice
    String indexName, indexColumns, queryOriginal = this.querys;
    for (int i=0; i<tablas.size(); i++){
        indexName = "IX_";
        indexColumns = "";
        for(Columna col : this.columnasSeleccionadas){
            if(i + 1 == col.getTuplaTabla()){
                indexName += col.getNombreColumna() + "_";
                indexColumns += col.getNombreColumna() + ", ";
            }
        }
        if (indexName != "IX_"){
            indexName = indexName.substring(0, indexName.length() - 1);
            indexColumns = indexColumns.substring(0, indexColumns.length() - 2);
            createIndexSyntax += "CREATE INDEX " + indexName + " ON " + tablas.get(i).getNombreTabla() + " (" + indexColumns + ");";
            dropIndexSyntax += "DROP INDEX " + indexName + " ON " + tablas.get(i).getNombreTabla() + ";";
            checkExistance += "SELECT 1 FROM INFORMATION_SCHEMA.STATISTICS WHERE TABLE_SCHEMA = '"+Constante.DATABASE_SELECTED+"' " +
                    "AND TABLE_NAME='" + tablas.get(i).getNombreTabla().toLowerCase().replaceAll( regex: "[^a-zA-Z0-9]", replacement: "") + "' " +
                    "AND INDEX_NAME='" + indexName + "';";
            //Forzar usar index en query
            String[] querys = this.querys.split( regex: ";");
            for (int j = 0; j<querys.length; j++){
                String q = querys[j].trim().toLowerCase();
                String tableName = tablas.get(i).getNombreTabla().toLowerCase();
                if (q.contains(tableName)){
                    tableName = " " + tableName + " ";
                    int index = q.indexOf(tableName);
                    index += tableName.length();
                    q = new StringBuilder(q).insert(index, str: " use index (" + indexName + ") ").toString();
                    querys[j] = q;
                }
            }
            this.querys = String.join( delimiter: ";", querys);
        }
    }
    this.createIndexSyntax = createIndexSyntax;
```

Figura 27. Codificación del cálculo del tiempo de ejecución del índice, parte 1. Fuente: Creado por el autor

En la Figura 27, se puede observar el código correspondiente al tiempo de ejecución del índice. Inicialmente, se establece la cadena de conexión con la base de datos MySQL. Posteriormente, se crea la sintaxis necesaria para la creación y eliminación del índice en la base de datos. Es crucial destacar que se modifica la sintaxis del query que se va a optimizar para asegurar el uso del índice recién creado. En el caso de MySQL, la sintaxis modificada es la siguiente:

```
SELECT * FROM table1 USE INDEX (col1_index,col2_index)
  WHERE col1=1 AND col2=2 AND col3=3;
```

Figura 28. Ejemplo de sugerencia de índice en MySQL. Fuente: MySQL :: MySQL 8.0 Reference Manual :: 8.9.4 Index Hints, s.f.

En la Figura 28 se observa la sintaxis necesaria para emplear un índice en específico, siendo colocar “USE INDEX (nombreIndice1, …, nombreIndiceN)” después de la tabla a la cual pertenecen dichos índices.

También se crea un query de verificación de existencia del índice a crear, para no duplicar índices dentro la base de datos.

```
//Correr querys
try (Connection conn = DriverManager.getConnection(connectionUrl,  user: "root",  password: "Lobogris22-1")) {
    Statement stmt = conn.createStatement();
    String[] checkIndex = checkExistance.split( regex: ";");
    String[] createIndex = createIndexSyntax.split( regex: ";");
    for (int i = 0; i<createIndex.length; i++){
        ResultSet rs = stmt.executeQuery(checkIndex[i].trim());
        rs.last();
        if (rs.getRow()==0){
            stmt.execute(createIndex[i].trim());
        }
    }
    String[] query = this.querys.split( regex: ";");
    for (int i = 0; i<query.length; i++){
        //Solo se cuenta el tiempo de la tercera ejecución
        stmt.execute(query[i]);
        stmt.execute(query[i]);
        startTime = System.nanoTime();
        stmt.execute(query[i]);
        endTime += (System.nanoTime() - startTime)/1000000000;
    }
    this.tiempoEjecucion = endTime;
    this.querys = queryOriginal;
    String[] dropIndex = dropIndexSyntax.split( regex: ";");
    for (int i = 0; i<dropIndex.length; i++){
        stmt.execute(dropIndex[i]);
    }
} catch (SQLException ex) {
    if (!ex.getMessage().contains("needed in a foreign key constraint")){
        System.out.println(ex.getMessage());
    }
}
```

Figura 29. Codificación del cálculo del tiempo de ejecución del índice, parte 2. Fuente: Creado por el autor

En la Figura 29, se presenta la segunda parte del código referente al espacio ocupado por el índice. Se inicia estableciendo la conexión con la base de datos y ejecutando la consulta para verificar la existencia del índice. En caso de que no exista, se procede a crearlo en la base de datos. Luego, se ejecuta el query tres veces. Esta elección se basa en que, durante la primera y posiblemente la segunda ejecución, el motor realiza diversos cálculos internos que pueden afectar el tiempo de ejecución. Por lo tanto, se toma en cuenta solo la tercera ejecución para medir el tiempo. Una vez finalizado, se eliminan los índices para evitar sobrecargar la base de datos con ellos.

6.2.1.7 Obtención de las tres mejores soluciones

```
public void seleccionarTresMejoresSoluciones(Lector lector, double alpha, double beta, double eDisp){
    if (lector == null)
        throw new InvalidParameterException(Constante.INVALID_PARAMETER_MSG);
    if (alpha < 0 || beta < 0 || eDisp < 0)
        throw new InvalidParameterException(Constante.NEGATIVE_PARAMETER_MSG);
    System.out.println("Mejores lobos");
    for (int i = 0; i<this.poblacion.size(); i++){
        if (this.poblacion.get(i).getFitness() == 1000000000){
            System.out.println("ML Lobo n°" + i);
            this.poblacion.get(i).setFitness(lector.getTablas(), alpha, beta, eDisp);
        }
    }

    Collections.sort(this.poblacion);
    this.alphaWolf = this.poblacion.get(0);
    this.betaWolf = this.poblacion.get(1);
    this.gammaWolf = this.poblacion.get(2);
}
```

Figura 30. Codificación de obtención de las tres mejores soluciones. Fuente: Creado por el autor

En la Figura 30, se presenta el código relacionado con la obtención de las tres mejores soluciones de la población. Inicialmente, para cada lobo dentro de la población, se verifica que no tenga el valor de fitness por defecto, que es 1,000,000,000. Si no tiene ese valor, se calcula su fitness. Posteriormente, se ordena la población en función del fitness de manera ascendente, y se seleccionan los tres primeros lobos como el lobo alfa, beta y gamma, respectivamente.

6.2.1.8 Selección de columnas

```
public void setColumnasSeleccionadas() {
    if (this.columnas == null)
        throw new InvalidParameterException(Constante.INCONSISTENT_PARAMETER_MSG);
    List<Columna> colSelect = new ArrayList<>();
    for(Columna col : this.columnas){
        if(col.getProbabilidadEleccion() >= 0.75){
            colSelect.add(col);
        }
    }
    this.columnasSeleccionadas = colSelect;
}
```

Figura 31. Codificación de la selección de columnas. Fuente: Creado por el autor

En la Figura 31 se observa la selección de las columnas que constituyen el índice. En base a la probabilidad de elección de la columna, se escogen aquellas que sean mayor o igual a 0.75, valor el cual se calibrará en la siguiente sección.

6.2.1.9 Enfoques, Medios de Confirmación y Marcadores

Se ha logrado el resultado esperado R5, que consiste en la codificación del algoritmo lobo gris para el problema de selección de índices en bases de datos relacionales. Para llevar a cabo este resultado, se utilizó el lenguaje de programación Java, aprovechando su orientación a objetos para organizar de manera ordenada las estructuras de datos y los métodos del algoritmo.

El código fuente del algoritmo lobo gris se encuentra detallado en el Anexo H. Además, se realizaron pruebas unitarias para cada método implementado, lo que contribuyó a obtener resultados confiables. El informe detallado sobre las pruebas unitarias se puede consultar en el Anexo I.

Se optó por utilizar un tablero Kanban en línea a través de la plataforma ClickUp durante el proceso de implementación, lo que facilitó la planificación y gestión de las tareas pendientes, así como su seguimiento y nivel de avance.

El documento de validación, firmado por un especialista en algoritmia metaheurística, confirma que la codificación del algoritmo lobo gris es adecuada. Este resultado ha sido revisado exhaustivamente y se encuentra disponible en el Anexo J.

6.2.2 Calibración de variables

En este apartado, se detalla el proceso llevado a cabo para lograr el resultado esperado R5.

6.2.2.1 Descripción

justar las variables es un paso empírico esencial para determinar la combinación óptima de parámetros que maximice el rendimiento de los algoritmos de optimización. Para el algoritmo lobo gris, se tomaron en cuenta los siguientes parámetros: el límite máximo de iteraciones, el número máximo de generaciones sin mejoras y el porcentaje de aceptación para la selección de columnas.

Las pruebas de calibración se llevaron a cabo utilizando la base de datos "Northwind extended" (Google Code Archive - Long-Term Storage for Google Code Project Hosting., s.f.). En cada prueba, el algoritmo se ejecutó 50 veces, registrando el mejor resultado, el resultado promedio y la desviación estándar. Este enfoque permite evaluar el rendimiento del algoritmo bajo diferentes configuraciones de parámetros y seleccionar aquellos que produzcan los resultados más consistentes y efectivos.

6.2.2.2 Máximo número de iteraciones

Esta variable corresponde al máximo número de iteraciones permitidas para el algoritmo lobo gris, cuando se alcanza esta cantidad máxima, se detiene y retorna el resultado óptimo hasta ese momento. Para calibrar esta variable se tomaron en cuenta valores entre 100 y 600.

Tabla 9. Resultados obtenidos al calibrar el límite máximo de iteraciones.

Máximo número de iteraciones	Óptimo desempeño	Desempeño Promedio	Desviación estándar
100	0.0072	0.0317	0.0318
200	0.0060	0.0284	0.0297
300	0.0059	0.0243	0.0244
400	0.0063	0.0246	0.0194
500	0.0059	0.0241	0.0157
600	0.0076	0.0263	0.0277

Como se aprecia, existe un empate de mejores resultados con 300 y 500 iteraciones, pero la media de 500 iteraciones incluyendo una desviación estándar de menor, indicando con los resultados no se encuentran tan alejados entre sí, por lo que se establece el número máximo de iteraciones en 500 para el algoritmo lobo gris.

6.2.2.3 Máximo número de generaciones sin mejora

Esta variable corresponde al máximo número de generaciones sin mejora permitidos en el algoritmo lobo gris antes de que este sea finalizado. Para calibrar esta variable se tomaron en cuenta valores entre 50 y 250.

Tabla 10. Resultados obtenidos al calibrar el máximo número de generaciones sin mejora

Máximo número de generaciones sin mejora	Mejor Resultado	Resultado Promedio	Desviación estándar
50	0.0090	0.0334	0.0274
100	0.0067	0.0181	0.0103
150	0.0061	0.0197	0.0122
200	0.0060	0.0225	0.0202
250	0.0075	0.0223	0.0153

Al revisar los resultados de la calibración, se destaca que el máximo rendimiento se logra con un límite de 200 iteraciones. Sin embargo, al considerar la media, se evidencia que un límite de 100 iteraciones resulta en una media más alta y una desviación estándar más baja.

6.2.2.4 Porcentaje de aceptación para elección de columnas

Esta variable corresponde al porcentaje de aceptación en base a la probabilidad de cada columna con las cuales se va a construir en índice. Para calibrar esta variable se tomaron en cuenta valores entre 0.65 y 0.85.

Tabla 11. Resultados obtenidos al calibrar el porcentaje de aceptación para elección de columnas

Porcentaje de aceptación para elección de columnas	Mejor Resultado	Resultado Promedio	Desviación estándar
0.65	0.0060	0.0242	0.0244
0.7	0.0060	0.0174	0.0403
0.75	0.0060	0.0241	0.0206
0.8	0.0062	0.0263	0.0194
0.85	0.0067	0.0157	0.0096

Como se aprecia, se obtienen mejores resultados con 0.65, 0.7 y 0.75, pero estos valores poseen un promedio y desviación estándar más alto, por lo que, a pesar de tener el peor resultado de los 5 porcentajes, se establece el porcentaje de aceptación para elección de columnas en 0.85 para el algoritmo lobo gris por una menor media y datos más uniformes.

6.2.2.5 Enfoques, Medios de Confirmación y Marcadores

Se utilizaron archivos CSV para registrar los resultados de manera iterativa de cada una de las pruebas de calibración de variables, empleando el lenguaje de programación Java, los cuales pueden visualizarse en el Anexo K.

Este resultado ha sido sometido a revisión y validación por parte de un especialista en algoritmia metaheurística, quien ha confirmado que la calibración de variables del algoritmo lobo gris es apropiada. El documento de validación firmado está disponible en el Anexo L.

6.3 Análisis y reflexión de los resultados

Se lograron los resultados esperados R4 y R5, referentes a la codificación del algoritmo lobo gris, que incluye las pruebas unitarias, y la calibración de variables para el algoritmo. La

consecución del resultado R4 implica contar con el algoritmo lobo gris completo y funcionando de manera eficiente. La implementación del código y la evaluación de diversas situaciones durante su ejecución proporcionaron una comprensión más profunda del funcionamiento de este algoritmo metaheurístico y cómo los lobos grises interactúan para rodear a sus presas.

Por otro lado, el resultado R5 facilita la definición de valores óptimos para minimizar la función objetivo y, al mismo tiempo, Optimizar la calidad de las soluciones producidas por el algoritmo en el contexto del problema de selección de índices. La calibración de variables contribuye a mejorar el rendimiento y la eficacia del algoritmo, permitiendo encontrar una combinación adecuada de parámetros para obtener soluciones de alta calidad.

Capítulo 7. Adaptación del algoritmo Genético

7.1 Introducción

En esta sección, se aborda la primera parte del objetivo específico 3, que consiste en “Comparar el desempeño del algoritmo metaheurístico propuesto contra la solución más relevante encontrada en el estado del arte para el problema de selección de índices con el fin de validar los resultados”. Este objetivo se desglosa en tres resultados, y se abordará el primero de ellos, que consiste en presentar el pseudocódigo del algoritmo genético obtenido del estado del arte. Para llevar a cabo esta implementación, es necesario primero definir las estructuras de datos mencionadas en el documento antes de proceder con la descripción del algoritmo.

7.2 Resultados obtenidos

7.2.1 Definición de las estructuras de datos

En esta sección, se presentan las estructuras de datos obtenidas del documento del estado del arte.

7.2.1.1 Representación de una tabla

En el diseño del algoritmo genético, se utiliza la estructura "Tabla" para representar una tabla de base de datos. Esta estructura almacena información como el nombre de la tabla, el número de la tabla dentro de la base de datos, la cantidad de filas y el espacio en GB ocupado por la tabla. Se ha propuesto la implementación de una clase llamada "Tabla", que contendrá los atributos mencionados previamente. Se proporciona una representación visual en la Figura 32.

Nombre de la tabla	Número de tabla	Cantidad de filas	Espacio
Empleado	2	500	5

Figura 32. Estructura de datos - Tabla

7.2.1.2 Representación de una columna o gen

En el contexto del algoritmo genético, la estructura "Columna" o "Gen" se emplea para representar una columna de una tabla de base de datos o un gen dentro del algoritmo. Esta estructura almacena información como el nombre de la columna, una tupla que indica el número de tabla y columna, la cantidad de bytes del tipo de dato, la probabilidad de elección (un valor binario de 0 o 1) y la penalidad de la columna. Se propone la implementación de

una clase llamada "Gen", que contendrá estos cinco atributos. Se proporciona una representación visual en la Figura 33.

Nombre de la columna	Tupla	Cantidad de bytes	Probabilidad de elección	Penalidad
Apellido	(5, 2)	1	0	1

Figura 33. Estructura de datos - Columna

7.2.1.3 Representación de un cromosoma

En el contexto del algoritmo genético, la estructura "Cromosoma" se emplea para representar un individuo de la población, el cual representa una posible solución al problema de selección de índices. Esta estructura almacena información como el query a optimizar, el tiempo de ejecución en segundos, el espacio en gigabytes, la penalidad total de las columnas, un arreglo de las posibles columnas a indexar y el valor de fitness del cromosoma. El fitness se calcula como el producto del tiempo de ejecución y la penalidad total. Se propone la implementación de una clase llamada "Cromosoma", que contendrá estos seis atributos. Se proporciona una representación visual en la Figura 34.

Query	Tiempo de ejecución (seg)	Espacio (GB)	Penalidad total	Arreglo de columnas	Fitness
Select * from empleado	7	0.16	5	Estructura Columnas	35

Figura 34. Estructura de datos - Índice

De acuerdo con los autores (Boronski & Bocewicz, 2014), propone 6 tipos de cromosomas:

- No indexado: No se crea un índice en cualquiera de las columnas presentes en el grupo de consultas.
- Ponderado: Se construye el índice en base a la columna más común de cada tabla presente en el grupo de consultas.
- Mitad: Se construye el índice con la primera mitad de las columnas de cada tabla presente en el grupo de consultas.
- No indexado mutado: Se construye de manera similar al cromosoma no indexado, con la diferencia de que cada gen perteneciente a un cromosoma tiene una probabilidad Pk de mutar.
- Ponderado mutado: Se construye de manera similar al cromosoma ponderado, con la diferencia de que cada gen perteneciente a un cromosoma tiene una probabilidad Pk de mutar.

- Mitad mutado: Se construye de manera similar al cromosoma mitad, con la diferencia de que cada gen perteneciente a un cromosoma tiene una probabilidad Pk de mutar.

7.2.2 Diseño del algoritmo Genético

La explicación del diseño del algoritmo genético se estructura en secciones dedicadas a sus métodos principales, lo que facilita la comprensión detallada de cada uno de ellos por separado. Esta organización permite abordar el procedimiento de manera sistemática, proporcionando una visión clara y completa de su funcionamiento.

7.2.2.1 Algoritmo Genético

```
1  Funcion algoritmoGenetico(querys, tablas, columnas, tamanoPoblacion, tamanoPoblacionElitista, p, rh, Ht, probMutacion, probCruzamiento)
2      espacioTablasQuerys ← calcularEspacioTablasQuerys(querys, tablas)
3      poblacion ← generarPoblacionInicial(querys, tablas, columnas, rh)
4      mejorAnterior ← obtenerMejorSolucion(poblacion, tablas)
5      t ← 1
6      MIENTRAS t ≤ p O obtenerTiempo(mejorAnterior) > Ht HACER
7          poblacion ← generarNuevaPoblacion(poblacion, tamanoPoblacion, tamanoPoblacionElitista, rh, espacioTablasQuerys, probMutacion, probCruzamiento)
8          mejorActual ← obtenerMejorSolucion(poblacion, tablas)
9          SI obtenerFitness(mejorActual) < obtenerFitness(mejorAnterior) ENTONCES
10             mejorAnterior ← mejorActual
11         FIN SI
12         t ← t + 1
13     FIN MIENTRAS
14 Fin Funcion
```

Figura 35. Pseudocódigo del algoritmo genético. Fuente: Creado por el autor

La Figura 35 exhibe el pseudocódigo principal del algoritmo genético, el cual recibe diez parámetros fundamentales para su ejecución: los queries o consultas a optimizar, los datos de las tablas y columnas de la base de datos, el tamaño de la población y de la población elitista, la cantidad de iteraciones por la letra “p”, el coeficiente de utilización de espacio “rh”, el tiempo de ejecución de las consultas después del cual el algoritmo debe finalizar “Ht”, y las probabilidades de mutación y cruzamiento (línea 1).

Para el procedimiento descrito, se realiza el cálculo del espacio total de las tablas involucradas en las consultas (línea 2) y se genera la población inicial (línea 3). Luego, se selecciona la mejor solución (línea 4). Posteriormente, mientras no se cumpla la condición de parada (línea 6), se genera una nueva población (línea 7) y se obtiene la mejor solución de esta nueva población (línea 8). En caso de que la nueva mejor solución tenga un fitness superior (línea 9), se actualiza la mejor solución (línea 10), repitiendo este procedimiento hasta que se cumpla la condición establecida.

7.2.2.2 Cálculo de espacio de tablas en querys

```
1  Funcion sum ← calcularEspacioTablasQuerys ( querys, tablas )
2      tablasQuerys ← obtenerTablasPresentesQuery(querys, tablas)
3      sum ← 0
4      PARA cada tabla EN tablasQuerys HACER
5          sum ← sum + obtenerEspacio(tabla)
6      FIN PARA
7  Fin Funcion
```

Figura 36. Pseudocódigo del cálculo de espacio de tablas en querys. Fuente: Creado por el autor

La Figura 36 presenta el pseudocódigo que ilustra cómo se calcula el espacio de las tablas en los queries.

En primer lugar, se obtienen las tablas presentes en las consultas y se guardan en el arreglo tablasQuerys (línea 2). Seguidamente, inicializa la variable sum en 0 (línea 3). Luego, por cada tabla dentro del arreglo tablasQuerys (línea 4), se va sumando a la variable sum el espacio en gb de la tabla (línea 5). Finalmente, se retorna la sumatoria (línea 1).

7.2.2.3 Generación de la población inicial

```
1  Funcion poblacion ← generarPoblacionInicial(querys, tablas, columnas, rh)
2      columnas ← obtenerColumnas(querys, tablas, columnas)
3      poblacion ← []
4      agente ← generarAgenteNoIndexado(columnasSel, querys)
5      agregar(agente, poblacion)
6      agente ← generarAgentePonderado(columnasSel, querys)
7      agregar(agente, poblacion)
8      agente ← generarAgenteMitad(columnasSel, querys)
9      agregar(agente, poblacion)
10     agente ← generarAgenteNoIndexadoMutado(columnasSel, querys)
11     agregar(agente, poblacion)
12     agente ← generarAgentePonderadoMutado(columnasSel, querys)
13     agregar(agente, poblacion)
14     agente ← generarAgenteMitadMutado(columnasSel, querys)
15     agregar(agente, poblacion)
16 Fin Funcion
```

Figura 37. Pseudocódigo de la generación de la población inicial. Fuente: Creado por el autor

La Figura 37 ilustra el proceso de generación de la población inicial, tal como se describe en el estudio seleccionado.

En primer lugar, se obtienen todas las columnas presentes del grupo de consultas (línea 2). Luego, se declara un arreglo vacío (línea 3). A continuación, se crean 6 agentes o

cromosomas tomados de la literatura y se van agregando a la población (líneas 4-15), los cuales se puede ver su explicación detallada en la sección Representación de un cromosoma. Finalmente, se retorna la población generada (línea 1).

7.2.2.4 Obtención de la mejor solución

```
1  Funcion mejorSolucion ← obtenerMejorSolucion ( poblacion, tablas )
2      PARA cada agente EN poblacion HACER
3          calcularFitness(agente, tablas)
4      FIN PARA
5
6      orderPorFitness(poblacion)
7      mejorSolucion ← poblacion[0]
8  Fin Funcion
```

Figura 38. Pseudocódigo de la obtención de la mejor solución. Fuente: Creado por el autor

En la Figura 38 se representa el método utilizado para obtener la mejor solución.

Como primeros pasos, para cada individuo en la población (línea 2) se calcula el fitness del cromosoma (línea 3). Después, se ordena la población en base al fitness calculado (línea 6). Luego, se seleccionan la primera posición de la población, en otras palabras, la mejor solución (líneas 7-9), y se retoma dicho valor (línea 1).

7.2.2.5 Control de aberraciones

```
1  Funcion flagValido ← esValido ( agente, espacioTablasQuerys, rh )
2      flagValido ← true
3      calcularEspacio(agente)
4
5      //Ratio espacio
6      SI obtenerEspacio(agente)/espacioTablasQuerys ≥ rh ENTONCES
7          flagValido ← false
8      FIN SI
9  Fin Funcion
```

Figura 39. Pseudocódigo del control de aberraciones. Fuente: Creado por el autor

La Figura 39 ilustra el método empleado para controlar las aberraciones en el algoritmo genético.

Al principio, se crea una bandera de control para identificar si es válido (línea 2). Primero, se calcula el espacio del agente o cromosoma (línea 3). Luego, se verifica que la división del espacio del agente sobre el espacio de las tablas de las consultas no sea mayor o igual al coeficiente de utilización de espacio (línea 6), en caso se cumpla la condición, se cambia el estado de la bandera (línea 7). Al final se retorna la bandera (línea 1).

7.2.2.6 Generación de una población

```
1  Funcion nuevaPoblacion ← generarNuevaPoblacion(poblacion, tamanoPoblacion, tamanoPoblacionElitista, rh, espacioTablasQuerys, probMutacion, probCruzamiento)
2      poblacionMutada ← []
3      poblacionHijos ← []
4      nuevaPoblacion ← []
5
6      //Mutación
7      PARA cada individuo EN poblacion HACER
8          SI aleatorio(0,1) ≤ probMutacion ENTONCES
9              nuevoIndividuo ← mutacion(individuo)
10             SI esValido(nuevoIndividuo, espacioTablasQuerys, rh) ENTONCES
11                 agregar(nuevoIndividuo, poblacionHijos)
12             FIN SI
13         FIN SI
14     FIN PARA
15
16     //Cruzamiento
17     PARA cada individuo EN poblacion HACER
18         SI aleatorio(0,1) ≤ probCruzamiento ENTONCES
19             padres ← seleccionar2cromosomas(poblacion)
20             hijo ← cruzamiento(padres, tasaCruzamiento)
21             SI esValido(hijo, espacioTablasQuerys, rh) ENTONCES
22                 agregar(hijo, poblacionHijos)
23             FIN SI
24         FIN SI
25     FIN PARA
26
27     totalPoblacion ← juntar(poblacion, poblacionMutada, poblacionHijos)
28     orderPorFitness(poblacion)
29
30     //Selección por torneo
31     PARA i←0 HASTA tamañoPoblacion CON PASO i<-i+1
32         SI i < tamanoPoblacionElitista ENTONCES
33             agregar(totalPoblacion[i], nuevaPoblacion)
34         SINO
35             competidores ← seleccionar2cromosomas(totalPoblacion)
36             SI obtenerFitness(competidores[0]) ≤ obtenerFitness(competidores[1]) ENTONCES
37                 agregar(competidores[0], nuevaPoblacion)
38             SINO
39                 agregar(competidores[1], nuevaPoblacion)
40             FIN SI
41         FIN SI
42     FIN PARA
43 Fin Funcion
```

Figura 40. Pseudocódigo de la generación de una población. Fuente: Creado por el autor

La Figura 40 presenta el pseudocódigo que describe el proceso de generación de una nueva población.

En primer lugar, se declaran arreglos vacíos para almacenar los nuevos individuos a generar (líneas 2-4). Se empieza con la mutación (líneas 7-14), donde para cada individuo dentro de la población (línea 7), se genera un aleatorio entre 0 y 1, y si este valor es menor o igual que la probabilidad de mutación (línea 8), se muta el individuo (línea 9). En caso el nuevo individuo mutado sea valido (línea 10), se agrega al arreglo de poblacionHijos (línea 11). Continuando, se procede con el cruzamiento (líneas 17-25), donde para cada individuo dentro de la población (línea 17), se genera un aleatorio entre 0 y 1, y si este valor es menor o igual que la probabilidad de cruzamiento (línea 18), se seleccionan dos padres de manera aleatoria (línea 19). Luego se cruzan los padres con una tasa de cruzamiento (línea 20), y dicho hijo (línea 20) se confirma que sea válido (línea 21) para añadirlo al arreglo poblacionHijos (línea 22). Prosiguiendo, se junta la población anterior, la población mutada y los hijos en el arreglo totalPoblacion (línea 27) y se ordena en base al fitness (línea 28). Finalmente, se realiza una selección por torneo (líneas 31-42). Donde, se genera una población (línea 31) seleccionando primero a los mejores cromosomas, es decir, la población elitista (líneas 32-34), para luego

pasar con la selección por torneo (líneas 30-41). Aquí, se escogen dos individuos o competidores del arreglo totalPoblacion (línea 35), y se escoge aquel que posea un menor fitness (líneas 36-40).

7.2.2.7 Cálculo de la función fitness

```
1  Funcion calcularFitness ( agente, tablas )
2      calcularTiempo(agente, tablas)
3      calcularPenalidadTotal(agente)
4
5      agente ← obtenerTiempo(agente) * obtenerPenalidadTotal(agente)
6  Fin Funcion
```

Figura 41. Pseudocódigo del cálculo de la función fitness. Fuente: Creado por el autor

En la Figura 41 se presenta el método que describe el cálculo de la función de ajuste (fitness).

Para empezar, se calcula el tiempo (línea 2) y la penalidad total (línea 3). Para concluir, se aplica la fórmula de la función objetivo la cual es la multiplicación de los valores calculados (línea 5).

7.3 Análisis y reflexión de los resultados

En este capítulo, se ha logrado el resultado esperado R6 del proyecto de tesis, que implica la formulación del pseudocódigo del algoritmo genético obtenido de la literatura. Este logro proporciona una comprensión detallada del funcionamiento del algoritmo genético y sirve como una base sólida de diseño para su posterior implementación.

Primordialmente, se explora la definición de las estructuras de datos fundamentales necesarias para el diseño del algoritmo genético. Estas estructuras están diseñadas para organizar los datos de manera que su utilización durante la implementación del algoritmo sea más sencilla y eficiente.

En segundo plano, se aborda el diseño del algoritmo genético y sus métodos principales, que constituyen las funciones a implementar según lo extraído del documento de referencia. Este enfoque facilita la comprensión de las características individuales de cada método, al tiempo que reduce la posibilidad de errores durante la implementación.

Capítulo 8. Codificación del algoritmo Genético

8.1 Introducción

En este capítulo, se aborda la segunda parte del objetivo específico 3, que consiste en “Comparar el desempeño del algoritmo metaheurístico propuesto contra la solución más relevante encontrada en el estado del arte para el problema de selección de índices con el fin de validar los resultados”. Dicho objetivo se compone de tres resultados alcanzados, de los cuales se desarrollará el segundo, que es “Codificación del algoritmo genético seleccionado”. Para la implementación de dicho resultado, se tomó como base los diseños descritos en el capítulo anterior, como las estructuras de datos y el pseudocódigo.

En este capítulo, al igual que en el capítulo 6, se presentarán los métodos principales del algoritmo genético. Se abordarán aspectos técnicos y consideraciones relacionadas con situaciones que surgieron durante el proceso de desarrollo.

8.2 Resultados obtenidos

8.2.1 Codificación del algoritmo genético seleccionado del estado del arte

En esta sección se detalla el proceso llevado a cabo para lograr el resultado esperado R7. Para alcanzar este objetivo, se emplearon tres diseños de bases de datos previamente utilizados: el primero, proveniente de "northwindextended" (Google Code Archive - Long-Term Storage for Google Code Project Hosting., s.f.) para una base de datos MySQL; el segundo, de "employees" (MySQL :: Employees Sample Database, s.f.) proporcionado por MySQL como muestras, con un total de 4 millones de registros; y el tercero, de "chinook" (Lerocha/Chinook-Database: Sample Database for SQL Server, Oracle, MySQL, PostgreSQL, SQLite, DB2, s.f.), también para una base de datos MySQL. Estos diseños de bases de datos se utilizaron para realizar pruebas del algoritmo genético.

8.2.1.1 Archivos de entrada

El algoritmo genético espera tres archivos de entrada, dos en formato CSV y otro SQL. Los dos primeros son sobre la información de las tablas y columnas de la base de datos. Estas corresponden a las Estructuras de datos planteadas para el algoritmo genético en el capítulo anterior.

```
nombreTabla,numeroTabla,cantidadFilas,espacio
categories,1,8,0.000137329102
customercustomerdemo,2,0,0.000030517578
customerdemographics,3,0,0.000015258789
customers,4,93,0.000015258789
employees,5,9,0.000167846680
employeeterritories,6,49,0.000030517578
`order details`,7,2155,0.000167846680
orders,8,830,0.000183105469
products,9,77,0.000045776367
region,10,4,0.000015258789
shippers,11,3,0.000015258789
suppliers,12,29,0.000015258789
territories,13,53,0.000030517578
```

Figura 42. Modelo de archivo csv sobre las tablas de base de datos. Fuente: Creado por el autor

Como se aprecia en la Figura 42, el archivo sobre las tablas tiene la siguiente estructura: nombre de la tabla, número de la tabla (correlativo), la cantidad de filas de dicha tabla y el espacio en gigabytes de la tabla.

```
nombreColumna,numeroTabla,numeroColumna,cantidadBytes
CategoryID,1,1,4
CategoryName,1,2,17
Description,1,3,16777215
Picture,1,4,4294967295
CustomerID,2,1,7
CustomerTypeID,2,2,12
CustomerTypeID,3,1,12
CustomerDesc,3,2,16777215
CustomerID,4,1,7
CompanyName,4,2,42
ContactName,4,3,32
ContactTitle,4,4,32
Address,4,5,62
City,4,6,17
Region,4,7,17
PostalCode,4,8,12
Country,4,9,17
Phone,4,10,26
Fax,4,11,26
```

Figura 43. Modelo de archivo csv sobre las columnas de base de datos. Fuente: Creado por el autor

Como se aprecia en la Figura 43, el archivo sobre las columnas tiene la siguiente estructura: nombre la columna, número de la tabla que corresponde, número de la columna (correlativo), y cantidad de bytes del tipo de dato.

```
SELECT Categories.CategoryName,
       Products.ProductName,
       Products.QuantityPerUnit,
       Products.UnitsInStock,
       Products.Discontinued
FROM Categories
     INNER JOIN Products ON Categories.CategoryID = Products.CategoryID
WHERE Products.Discontinued <> 1;
```

Figura 44. Modelo de archivo sql. Fuente: Creado por el autor

Por último, el archivo SQL posee dos consideraciones. En primer lugar, no usar alias para las tablas y redactar las columnas de la siguiente manera: “nombreTabla.nombreColumna” como se aprecia en la Figura 44. Esto con el fin de que el algoritmo pueda detectar las columnas presentes en las consultas. Dado que los autores no detallan alguna restricción en cuanto al workload de entrada, se tomará la consideración sobre las operaciones planteadas en el capítulo 6.

8.2.1.2 Método principal del Algoritmo Genético

```java
public class AlgoritmoGenetico {
    public static void main(String[] args) {
        String filenameTablas = "tablas_"+ Constante.DATABASE_SELECTED+"_mysql.csv",
                filenameColumnas = "columnas_"+Constante.DATABASE_SELECTED+"_mysql.csv",
                filenameQuery = "query_"+Constante.DATABASE_SELECTED+".sql";

        Lector lector = new Lector( rutaArchivoTablas: Constante.PATH_INPUT_CSV + filenameTablas,
                rutaArchivoColumnas: Constante.PATH_INPUT_CSV + filenameColumnas,
                Path.of( first: Constante.PATH_INPUT_CSV + filenameQuery));
        lector.leerArchivos();

        int tamanoPoblacion = 30, tamanoPoblacionElitista = 8, t = 1, maxIter = 500, sinMejora = 0;
        float probMutacion = 0.15f, probCruzamiento = 0.85f, rh = 0.9f, Ht = 0.000001f;
        long startTime, endTime;
        List<Cromosoma> mejorAnterior, mejorActual;
        startTime = System.nanoTime();
        Poblacion poblacion = new Poblacion(tamanoPoblacion, tamanoPoblacionElitista);
        poblacion.crearPoblacionInicial(lector, rh);
        poblacion.obtenerMejorSolucion(lector);
        mejorAnterior = poblacion.getMejorSolucion();
        while (t <= maxIter || mejorAnterior.get(0).getFitness() > Ht){
            System.out.println("Iteración nº "+t);
            if (sinMejora == 100)
                break;
            System.out.println("Sin mejora nº "+sinMejora);
            List<Cromosoma> cromosomasNuevos = poblacion.crearNuevaPoblacion(lector, rh, lector.getEspacioTablasQuerys(), probMutacion, probCruzamiento);
            poblacion = new Poblacion(cromosomasNuevos, tamanoPoblacion);
            poblacion.obtenerMejorSolucion(lector);
            mejorActual = poblacion.getMejorSolucion();
            if (mejorActual.get(0).getFitness() < mejorAnterior.get(0).getFitness()){
                mejorAnterior = mejorActual;
                sinMejora = 0;
            }
            else
                sinMejora++;
            t++;
        }
        System.out.println("Mejor solución");
        for (Cromosoma c : mejorAnterior){
            System.out.println("Cromosoma");
            c.printSolucion();
        }
        endTime = (System.nanoTime() - startTime);
        System.out.println("Tiempo total: " + TimeUnit.MINUTES.convert(endTime, TimeUnit.NANOSECONDS) + " minutos.");
    }
}
```

Figura 45. Codificación del algoritmo lobo gris. Fuente: Creado por el autor

En la Figura 45 se visualiza el código principal que define el funcionamiento del método del algoritmo genético. En este código, se establecen inicialmente las rutas a los archivos de entrada que contienen los datos de las tablas y columnas de la base de datos, así como las consultas a optimizar. Posteriormente, se utiliza un objeto llamado "lector" para leer los datos mencionados. A continuación, se definen siete parámetros esenciales: el tamaño de la población, el tamaño de la población elitista, la cantidad de iteraciones, las constantes "rh" y "Ht", y las probabilidades de mutación y cruzamiento. También, se incorpora una variable adicional que registra el número de iteraciones en las que no se ha registrado mejoría, es decir, cuando el fitness se mantiene constante. Esto se hace con el fin de evitar la ejecución continua del algoritmo sin observar mejoras significativas..

8.2.1.3 Generación de la población inicial

```
public void crearPoblacionInicial(Lector lector, float rh){
    if (lector == null)
        throw new InvalidParameterException(Constante.INVALID_PARAMETER_MSG);
    if (rh < 0f)
        throw new InvalidParameterException(Constante.NEGATIVE_PARAMETER_MSG);
    Cromosoma nuevoGen;
    List<Cromosoma> p = new ArrayList<>();
    do {
        nuevoGen = new Cromosoma(lector.getQuerys(), lector.getColumnasQuery());
        nuevoGen.crearCromosomaNoIndexado();
    } while (!nuevoGen.esValido(lector.getTablas(), lector.getEspacioTablasQuerys(), rh));
    p.add(new Cromosoma(nuevoGen));
    do {
        nuevoGen = new Cromosoma(lector.getQuerys(), lector.getColumnasQuery());
        nuevoGen.crearCromosomaPonderado(lector.getColumnaMasComunPorTabla());
    } while (!nuevoGen.esValido(lector.getTablas(), lector.getEspacioTablasQuerys(), rh));
    p.add(new Cromosoma(nuevoGen));
    do {
        nuevoGen = new Cromosoma(lector.getQuerys(), lector.getColumnasQuery());
        nuevoGen.crearCromosomaMitad(lector.getTablas());
    } while (!nuevoGen.esValido(lector.getTablas(), lector.getEspacioTablasQuerys(), rh));
    p.add(new Cromosoma(nuevoGen));
    do {
        nuevoGen = new Cromosoma(lector.getQuerys(), lector.getColumnasQuery());
        nuevoGen.crearCromosomaNoIndexadoMutado();
    } while (!nuevoGen.esValido(lector.getTablas(), lector.getEspacioTablasQuerys(), rh));
    p.add(new Cromosoma(nuevoGen));
    do {
        nuevoGen = new Cromosoma(lector.getQuerys(), lector.getColumnasQuery());
        nuevoGen.crearCromosomaPonderadoMutado(lector.getColumnaMasComunPorTabla());
    } while (!nuevoGen.esValido(lector.getTablas(), lector.getEspacioTablasQuerys(), rh));
    p.add(new Cromosoma(nuevoGen));
    do {
        nuevoGen = new Cromosoma(lector.getQuerys(), lector.getColumnasQuery());
        nuevoGen.crearCromosomaMitadMutado(lector.getTablas());
    } while (!nuevoGen.esValido(lector.getTablas(), lector.getEspacioTablasQuerys(), rh));
    p.add(new Cromosoma(nuevoGen));
    this.poblacion = p;
}
```

Figura 46. Codificación de la generación de la población inicial. Fuente: Creado por el autor

En la Figura 46 se presenta el código que se encarga de generar la población inicial del algoritmo. En primer lugar, se valida que no se reciba parámetros nulos o negativos. Luego se crea cada uno de los tipos de cromosomas planteado por los autores y se confirma que este sea válido para que pueda ser agregado a la población.

8.2.1.4 Crear una nueva población

```java
public List<Cromosoma> crearNuevaPoblacion (Lector lector, float rh, double espacioTablasQuerys, float probMutacion, float probCruzamiento) {
    if (lector == null)
        throw new InvalidParameterException(Constante.INVALID_PARAMETER_MSG);
    if (rh < 0 || espacioTablasQuerys < 0 || probMutacion < 0 || probCruzamiento < 0)
        throw new InvalidParameterException(Constante.NEGATIVE_PARAMETER_MSG);
    List<Cromosoma> poblacionMutada = new ArrayList<>(), poblacionHijos = new ArrayList<>(),
            nuevaPoblacion = new ArrayList<>(), totalPoblacion;

    //Mutación
    for (Cromosoma c : this.poblacion){
        if (Math.random() <= probMutacion){
            Cromosoma nuevoCromosoma = new Cromosoma(c);
            nuevoCromosoma.mutar();
            if (nuevoCromosoma.esValido(lector.getTablas(), lector.getEspacioTablasQuerys(), rh))
                poblacionMutada.add(new Cromosoma(nuevoCromosoma));
        }
    }

    //Cruzamiento
    for (Cromosoma c : this.poblacion){
        if (Math.random() <= probCruzamiento){
            Random rand = new Random();
            int padre1Index = rand.nextInt(this.poblacion.size()), padre2Index = rand.nextInt(this.poblacion.size());
            Cromosoma padre1 = new Cromosoma(this.poblacion.get(padre1Index)), padre2 = new Cromosoma(this.poblacion.get(padre2Index)), nuevoCromosoma;
            nuevoCromosoma = padre1.cruzar(padre2);
            if (nuevoCromosoma.esValido(lector.getTablas(), lector.getEspacioTablasQuerys(), rh))
                poblacionHijos.add(new Cromosoma(nuevoCromosoma));
        }
    }

    totalPoblacion = Stream.of(this.poblacion, poblacionMutada, poblacionHijos)
            .flatMap(Collection::stream)
            .collect(Collectors.toList());
    Collections.sort(totalPoblacion);

    for (int i=0; i<this.tamanoPoblacion; i++){
        if (i<this.tamanoPoblacionElitista){
            nuevaPoblacion.add(new Cromosoma(totalPoblacion.get(i)));
        }
        else {
            Random rand = new Random();
            int competidor1Index = rand.nextInt(totalPoblacion.size()), competidor2Index = rand.nextInt(totalPoblacion.size());
            Cromosoma competidor1 = new Cromosoma(totalPoblacion.get(competidor1Index)), competidor2 = new Cromosoma(totalPoblacion.get(competidor2Index));
            if (competidor1.getFitness() <= competidor2.getFitness())
                nuevaPoblacion.add(new Cromosoma(competidor1));
            else
                nuevaPoblacion.add(new Cromosoma(competidor2));
        }
    }

    return nuevaPoblacion;
}
```

Figura 47. Codificación de la actualización de posición de la población. Fuente: Creado por el autor

En la Figura 47 se observa el código de cómo se crea una nueva población. Primero, se valida que no se reciba parámetros nulos o negativos, para luego proceder con las operaciones. Primero en la mutación, para cada cromosoma se genera un aleatorio el cual, si es menor igual que la probabilidad de mutación se muta dicho cromosoma, el cual solo se añadirá a la población mutada si es válido. De manera análoga en el cruzamiento, para cada cromosoma se genera un aleatorio el cual, si es menor igual que la probabilidad de cruzamiento, se escogen dos cromosomas o padres aleatoriamente para cruzarlos y generar un hijo, el cual solo se añadirá a la población cruzada si es válido. Seguidamente, se juntan las dos poblaciones anteriores y la población inicial para ordenarlas en base al fitness. Después, se selecciona la población elitista para luego proceder con la selección por torneo donde se

seleccionan dos cromosomas aleatoriamente y se añade a la nueva población aquel con menor fitness.

8.2.1.5 Control de aberraciones

```
public Boolean esValido (List<Tabla> tablas, double espacioTablasQuerys, float rh) {
    Boolean flagValido = true;
    this.setEspacio(tablas);

    //Ratio espacio
    if (this.espacio/espacioTablasQuerys >= rh) {
        flagValido = false;
    }
    return flagValido;
}
```

Figura 48. Codificación del control de aberraciones, parte 1. Fuente: Creado por el autor

En la Figura 48 se muestra la primera parte del código correspondiente al control de aberraciones. Antes de empezar con las restricciones, se calcula el espacio del índice sugerido como se aprecia en la Figura 49. En cuanto a las restricciones, sólo se valida que la división del espacio del índice sobre el espacio de las tablas presentes dentro del query no sea mayor a la constante “rh”.

```java
public void setEspacio(List<Tabla> tablas) {
    if (tablas == null)
        throw new InvalidParameterException(Constante.INVALID_PARAMETER_MSG);
    if (tablas.isEmpty())
        throw new InvalidParameterException(Constante.EMPTY_LIST_PARAMETER_MSG);
    this.setColumnasSeleccionadas();
    if (this.columnasSeleccionadas == null)
        throw new InvalidParameterException(Constante.INCONSISTENT_PARAMETER_MSG);
    double sum = 0;
    double espTab;
    for (int i=0; i<tablas.size(); i++){
        espTab = 0;
        for(Gen col : this.columnasSeleccionadas){
            if(i + 1 == col.getTuplaTabla()){
                espTab += col.getCantidadBytes();
            }
        }
        if(espTab != 0){
            espTab += 11; //Factor de overhead
            espTab *= tablas.get(i).getCantidadFilas();
            sum += espTab;
        }
    }
    sum = sum * 2 / 1000000000;
    this.espacio = sum;
}
```

Figura 49. Codificación del control de aberraciones, parte 2. Fuente: Creado por el autor

En la Figura 49 se observa la primera parte del código del espacio que ocuparía el índice. De igual manera que en el algoritmo lobo gris, para dicha fórmula se tomó como referencia lo establecido por IBM (Space Requirements for Indexes - IBM Documentation, s.f.) el cual es:

$$(tamaño\ medio\ del\ índice\ +\ sobrecarga\ de\ clave\ de\ índice)\ \times\ número\ de\ filas\ \times\ 2$$

El tamaño medio del índice se calcula a partir del espacio en bytes de las columnas que lo forman, y la sobrecarga de acuerdo con la documentación, para cualquier tipo de tabla es de 11 bytes.

8.2.1.6 Obtención de la mejor solución

```
public void obtenerMejorSolucion(Lector lector) {
    if (lector == null)
        throw new InvalidParameterException(Constante.INVALID_PARAMETER_MSG);
    for (int i = 0; i<this.poblacion.size(); i++){
        if (this.poblacion.get(i).getFitness() == 1000000000){
            this.poblacion.get(i).setFitness(lector.getTablas());
        }
    }
    Collections.sort(this.poblacion);
    double fitnessMin = this.poblacion.get(0).getFitness();
    List<Cromosoma> poblacionTop = this.poblacion.stream()
            .filter(cromosoma -> cromosoma.getFitness() == fitnessMin)
            .collect(Collectors.toList());
    this.mejorSolucion = poblacionTop;
}
```

Figura 50. Codificación de obtención de la mejor solución. Fuente: Creado por el autor

En la Figura 50 se muestra el proceso de obtener la mejor solución dentro de la población. Primero, por cada cromosoma dentro de la población se constata que no tenga el valor del fitness por defecto, de 1.000,000,000, sino se calcula su fitness. Después, se ordena la población en base al fitness de manera ascendente, se obtiene el primer resultado y se guarda como mejor solución aquellos cromosomas que tengan el mismo fitness.

8.2.1.7 Cálculo de la función fitness

```
public void setFitness(List<Tabla> tablas) {
    this.setTiempoEjecucion(tablas);
    this.setPenalidadTotal();

    this.fitness = (double)Math.round(this.getTiempoEjecucion() * this.getPenalidadTotal() * 100000000d) / 100000000d;
}
```

Figura 51. Codificación del cálculo de la función fitness. Fuente: Creado por el autor

En la Figura 51 se presenta el código encargado del cálculo de la función de fitness. En primer lugar, se realiza el cálculo de la penalidad y el tiempo de ejecución del índice. Luego, el valor de fitness se obtiene multiplicando estos valores, redondeados al octavo decimal.

```java
public void setPenalidadTotal() {
    if (this.columnasSeleccionadas == null)
        throw new InvalidParameterException(Constante.INCONSISTENT_PARAMETER_MSG);
    double sum = 1;
    for(Gen col : this.columnasSeleccionadas){
        sum *= col.getPenalidad();
    }
    if (this.columnasSeleccionadas.size() == 0){
        sum = 5;
    }
    this.penalidadTotal = sum;
}
```

Figura 52. Codificación del cálculo de la penalidad total de índice. Fuente: Creado por el autor

En la Figura 52 se observa el código de la frecuencia total del índice. Esta es la multiplicación de la penalidad de cada columna que conforma el índice y en caso no se han seleccionado ninguna columna se coloca el valor de 5 por defecto.

```java
public void setTiempoEjecucion(List<Tabla> tablas) {
    if (this.columnasSeleccionadas == null)
        throw new InvalidParameterException(Constante.INCONSISTENT_PARAMETER_MSG);
    String connectionUrl = "jdbc:mysql://localhost:3306/"+Constante.DATABASE_SELECTED+"?serverTimezone=UTC", createIndexSyntax = "", dropIndexSyntax = "", checkExistance = "";
    double startTime, endTime = 0;
    //Crear sintaxis de indice
    String indexName, indexColumns, queryOriginal = this.querys;
    for (int i=0; i<tablas.size(); i++){
        indexName = "IX_";
        indexColumns = "";
        for(Gen col : this.columnasSeleccionadas){
            if(i + 1 == col.getTuplaTabla()){
                indexName += col.getNombreColumna() + "_";
                indexColumns += col.getNombreColumna() + ", ";
            }
        }
        if (indexName != "IX_"){
            indexName = indexName.substring(0, indexName.length() - 1);
            if (indexName.length() >= 64)
                indexName = indexName.substring(0, 63);
            indexColumns = indexColumns.substring(0, indexColumns.length() - 2);
            createIndexSyntax += "CREATE INDEX " + indexName + " ON " + tablas.get(i).getNombreTabla() + " (" + indexColumns + ");";
            dropIndexSyntax += "DROP INDEX " + indexName + " ON " + tablas.get(i).getNombreTabla() + ";";
            checkExistance += "SELECT 1 FROM INFORMATION_SCHEMA.STATISTICS WHERE TABLE_SCHEMA = '"+Constante.DATABASE_SELECTED+"' " +
                    "AND TABLE_NAME='" + tablas.get(i).getNombreTabla().toLowerCase().replaceAll( regex: "[^a-zA-Z0-9]", replacement: "") + "' " +
                    "AND INDEX_NAME='" + indexName + "';";
            //Forzar usar index en query
            String[] querys = this.querys.split( regex: ";");
            for (int j = 0; j<querys.length; j++){
                String q = querys[j].trim().toLowerCase();
                String tableName = tablas.get(i).getNombreTabla().toLowerCase();
                if (q.contains(tableName)){
                    tableName = " " + tableName + " ";
                    int index = q.indexOf(tableName);
                    index += tableName.length();
                    q = new StringBuilder(q).insert(index, str: " use index (" + indexName + ") ").toString();
                    querys[j] = q;
                }
            }
            this.querys = String.join( delimiter: ";", querys);
        }
    }
    this.createIndexSyntax = createIndexSyntax != "" ? createIndexSyntax : "No necesita indice";
```

Figura 53. Codificación del cálculo del tiempo de ejecución del índice, parte 1. Fuente: Creado por el autor

En la Figura 53 se presenta el código relacionado con el tiempo de ejecución del índice en el algoritmo genético. En primer lugar, se establece la conexión con la base de datos MySQL. Luego, se crea la sintaxis para la creación y eliminación del índice dentro de la base de datos.

Además, se ajusta la sintaxis del query a optimizar para asegurar que se utilice el índice creado. Para MySQL, la sintaxis específica es la siguiente:

```
SELECT * FROM table1 USE INDEX (col1_index,col2_index)
  WHERE col1=1 AND col2=2 AND col3=3;
```

Figura 54. Ejemplo de sugerencia de índice en MySQL. Fuente: MySQL :: MySQL 8.0 Reference Manual :: 8.9.4 Index Hints, s.f.

En la Figura 54 se observa la sintaxis necesaria para emplear un índice en específico, siendo colocar “USE INDEX (nombreIndice1, …, nombreIndiceN)” después de la tabla a la cual pertenecen dichos índices. También se crea un query de verificación de existencia del índice a crear, para no duplicar índices dentro la base de datos.

```
//Correr querys
try (Connection conn = DriverManager.getConnection(connectionUrl,  user: "root",  password: "Lobogris22-1")) {
    Statement stmt = conn.createStatement();
    String[] checkIndex = checkExistance.split( regex: ";");
    String[] createIndex = createIndexSyntax.split( regex: ";");
    if (checkExistance != ""){
        for (int i = 0; i<createIndex.length; i++){
            ResultSet rs = stmt.executeQuery(checkIndex[i].trim());
            rs.last();
            if (rs.getRow()==0){
                stmt.execute(createIndex[i].trim());
            }
        }
    }
    String[] query = this.querys.split( regex: ";");
    for (int i = 0; i<query.length; i++){
        //Solo se cuenta el tiempo de la tercera ejecución
        stmt.execute(query[i]);
        stmt.execute(query[i]);
        startTime = System.nanoTime();
        stmt.execute(query[i]);
        endTime += (System.nanoTime() - startTime)/1000000000;
    }
    this.tiempoEjecucion = (double)Math.round(endTime * 1000000d) / 1000000d;
    this.querys = queryOriginal;
    String[] dropIndex = dropIndexSyntax.split( regex: ";");
    if (dropIndexSyntax != "") {
        for (int i = 0; i<dropIndex.length; i++){
            stmt.execute(dropIndex[i]);
        }
    }
} catch (SQLException ex) {
    if (!ex.getMessage().contains("needed in a foreign key constraint")){
        System.out.println(ex.getMessage());
    }
}
```

Figura 55. Codificación del cálculo del tiempo de ejecución del índice, parte 2. Fuente: Creado por el autor

En la Figura 55, se presenta la segunda parte del código que aborda el espacio que ocuparía el índice en el contexto del algoritmo genético. Primero, se establece la conexión con la base de datos y se ejecuta la consulta para verificar la existencia del índice; en caso de no existir, se crea en la base de datos. Luego, se procede con la ejecución del query, el cual se ejecuta tres veces. Esto se debe a que, durante la primera y posiblemente la segunda ejecución, el motor realiza diversos cálculos internos que afectan el tiempo de ejecución. Por esta razón, solo se considera la tercera ejecución para la toma del tiempo. Una vez completado este proceso, se eliminan los índices para evitar sobrecargar de espacio a la base de datos.

8.2.2 Enfoques, Medios de Confirmación y Marcadores

El resultado obtenido está detallado en la sección de implementación del algoritmo genético de este capítulo. Para su implementación, se empleó el lenguaje de programación Java, aprovechando su paradigma orientado a objetos para organizar de manera estructurada las clases y métodos del algoritmo. El código fuente del algoritmo genético se encuentra disponible en el Anexo N. Además, se llevaron a cabo pruebas unitarias para evaluar individualmente cada método implementado, proporcionando una mayor confiabilidad en los resultados obtenidos. Para más detalles, se puede consultar el documento de pruebas unitarias en el Anexo O.

Además, con el fin de planificar y gestionar eficientemente el proceso de implementación del algoritmo genético, se utilizó un tablero Kanban en línea a través de la plataforma web ClickUp. Este enfoque permitió una visualización clara de las tareas pendientes y su progreso, facilitando así la coordinación y seguimiento del trabajo realizado durante la fase de implementación.

Finalmente, este resultado ha sido sometido a revisión y validación por parte de un especialista en algoritmia metaheurística. El experto ha concluido que la codificación del algoritmo genético es adecuada. El documento detallado de validación, debidamente firmado, se encuentra disponible en el Anexo P.

8.3 Análisis y reflexión de los resultados

En este capítulo se ha logrado el resultado esperado R7, que abarca la codificación del algoritmo genético junto con las pruebas unitarias. Esta implementación proporciona un algoritmo genético completo y funcional, permitiendo una comprensión detallada de su funcionamiento y brindando la posibilidad de comparar su rendimiento con el algoritmo lobo gris.

Capítulo 9. Experimentación numérica

9.1 Introducción

Se abordará la tercera y última fase del objetivo específico 3, que consiste en “Comparar el desempeño del algoritmo metaheurístico propuesto contra la solución más relevante encontrada en el estado del arte para el problema de selección de índices con el fin de validar los resultados”. Este objetivo se compone de tres logros, de los cuales se explorará el tercero: la “Experimentación numérica para la comparación entre el algoritmo propuesto y el algoritmo genético”. Esta etapa permitirá determinar cuál de los algoritmos se considera una solución superior para el problema de selección de índices en bases de datos relacionales, utilizando el valor de la función objetivo como métrica principal..

9.2 Resultados obtenidos

9.2.1 Informe de experimentación numérica

La sección se organiza en dos partes principales: la recolección de datos y la realización de las pruebas de Shapiro-Wilk, Bartlett y Prueba Z. Estas pruebas se llevan a cabo con el objetivo de examinar las instancias de ejecución adquiridas para los algoritmos elaborados.

9.2.1.1 Recolección de datos

Se utilizó la base de datos "northwindextended" para la recolección de datos, y se emplearon los mismos conjuntos de datos de entrada para ambos algoritmos. Es importante señalar que durante la ejecución del algoritmo lobo gris, las constantes alpha y beta se fijaron en "0" para asegurar la igualdad en la función objetivo.

Los resultados recopilados se exhiben en la Figura 56 y se describen en detalle en el Anexo Q.

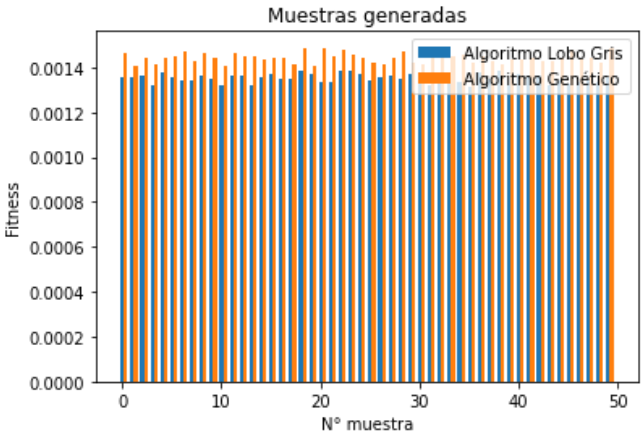


Figura 56. Valor de fitness más bajo registrado durante la recopilación de datos. Fuente: Creado por el autor

9.2.1.2 Prueba de Shapiro-Wilk

Se llevó a cabo la prueba de Shapiro-Wilk para verificar la normalidad de las muestras obtenidas. Aunque se realizó la validación de las muestras para cada algoritmo de forma independiente, se mantiene la hipótesis de normalidad, ya que ambos casos comparten el mismo objetivo:

H_0: La muestra sigue una distribución normal

H_1: La muestra no sigue una distribución normal

Para el algoritmo genético:

Con un nivel de significancia del 5%, se obtuvo un p-valor de 0.1715. Al ser este p-valor mayor que 0.05, se acepta la hipótesis nula y se concluye que los resultados del algoritmo genético siguen una distribución normal.

Para el algoritmo lobo gris:

Con un nivel de significancia del 5%, se obtuvo un p-valor de 0.5433. Al ser este p-valor mayor que 0.05, se acepta la hipótesis nula y se concluye que los resultados del algoritmo lobo gris siguen una distribución normal.

9.2.1.3 Prueba de Bartlett

Se empleó la prueba de Bartlett para examinar la homogeneidad de la varianza entre las muestras. Se plantearon las siguientes hipótesis:

H_0: Las varianzas entre las muestras son homogéneas

H_1: Las varianzas entre las muestras son diferentes

Con un nivel de significancia del 5%, se obtuvo un p-valor de 0.2234. Al ser este p-valor mayor que 0.05, se acepta la hipótesis nula, concluyendo que las varianzas entre las muestras son homogéneas.

9.2.1.4 Prueba Z

Después de confirmar la normalidad y homogeneidad de las varianzas entre las muestras, se procedió a realizar la Prueba Z para comparar las medias de los algoritmos lobo gris y genético. Las hipótesis planteadas fueron las siguientes:

H_0: La media del algoritmo lobo gris es igual a la del algoritmo genético

H_1: La media del algoritmo lobo gris es diferente a la del algoritmo genético

Con un nivel de significancia del 5%, el p-valor obtenido fue de 4.2047e-105. Este valor proporciona evidencia suficiente para rechazar la hipótesis nula, lo que sugiere que la media del algoritmo lobo gris es diferente de la del algoritmo genético.

Al analizar los resultados en detalle, se observa que la media del algoritmo lobo gris es de 0.001352, mientras que para el algoritmo genético es de 0.001443. Por lo tanto, se puede concluir que el algoritmo lobo gris produce soluciones de mayor calidad en comparación con el algoritmo genético.

9.2.2 Enfoques, Medios de Confirmación y Marcadores

El contenido de este capítulo ha sido revisado por un experto en algoritmia, quien ha validado la información presentada en él. Se puede verificar esta validación en el Anexo R.

9.3 Análisis y reflexión de los resultados

El informe de experimentación numérica, correspondiente al resultado esperado R8, ha sido completado satisfactoriamente. Este resultado permite realizar un análisis estadístico de la eficacia de los algoritmos en respuesta al problema de estudio, utilizando el valor de la función objetivo como métrica de evaluación.

La amplitud de este resultado reside en la capacidad para identificar el algoritmo que se presenta como una solución más efectiva, basándose en el análisis de las muestras de ejecución obtenidas. En este caso específico, la Prueba Z reveló que el algoritmo lobo gris supera al algoritmo genético. Además, el algoritmo lobo gris incluye la frecuencia y el espacio en su función objetivo, lo que lo hace más adaptable a las necesidades del usuario.

En conclusión, se puede afirmar que el algoritmo de lobo gris proporciona resultados superiores en comparación con el algoritmo genético para abordar el problema de selección de índices en bases de datos relacionales.

Capítulo 10. Conclusiones y trabajos futuros

10.1 Conclusiones

Se ofrecerá una alternativa de solución al problema de selección de índices en bases de datos relacionales, a través de la exposición de las conclusiones derivadas de la finalización de los objetivos específicos y sus respectivos resultados esperados.

En primer lugar, la función objetivo propuesta se integró sin dificultades en el desarrollo de los algoritmos, ya que las soluciones generadas por estos contienen todos los elementos necesarios para su evaluación. Se observó que la función objetivo desempeñó su función de distinguir entre soluciones, considerando diversos factores.

En segundo lugar, tanto el algoritmo genético como el algoritmo lobo gris se adaptaron correctamente al problema de selección de índices, proporcionando las soluciones esperadas.

En tercer lugar, los resultados de la experimentación numérica indican que el algoritmo de lobo gris supera al algoritmo genético. El enfoque del algoritmo lobo gris, centrado en buscar constantemente el óptimo global y evitar óptimos locales, contrasta con la estrategia del algoritmo genético, que se basa en heredar y mejorar las mejores soluciones iterativamente. A pesar de ello, los resultados del algoritmo lobo gris no difieren significativamente de los del algoritmo genético, como se refleja en el fitness obtenido en la experimentación numérica.

10.2 Trabajos futuros

En este apartado, se exploran potenciales direcciones para futuras investigaciones que podrían complementar o enriquecer el proyecto de fin de carrera actual.

Se sugiere la exploración del desarrollo de una interfaz gráfica que exhiba los resultados finales generados por el algoritmo. Esta interfaz proporcionaría a los usuarios una representación visual del progreso del algoritmo y mostraría qué columnas están aumentando su probabilidad de selección en la construcción del índice. Este enfoque podría mejorar la comprensión y la interacción de los usuarios con el algoritmo, permitiéndoles tener una visión más intuitiva y detallada del proceso de selección de índices.

Referencias

Abdel-Basset, M., Abdel-Fatah, L., & Sangaiah, A. K. (2018). Metaheuristic Algorithms: A Comprehensive Review. Computational Intelligence for Multimedia Big Data on the Cloud with Engineering Applications, 185–231. https://doi.org/10.1016/B978-0-12-813314-9.00010-4

Ae, L. B., Dorigo, M., Luca, A. E., Ae, M. G., & Gutjahr, W. J. (2008). A survey on metaheuristics for stochastic combinatorial optimization. https://doi.org/10.1007/s11047-008-9098-4

Almeida, F., Silva, P., & Araújo, F. (2019). Performance analysis and optimization techniques for Oracle Relational Databases. Cybernetics and Information Technologies, 19(2), 117–132. https://doi.org/10.2478/CAIT-2019-0019

AWS. (s.f.). What is a Relational Database?. Recuperado el 19 de agosto de 2021, a partir de https://aws.amazon.com/relational-database/?nc1=h_ls

A Relational Database Overview. (s.f.). Recuperado el 26 de setiembre de 2021, a partir de https://docs.oracle.com/javase/tutorial/jdbc/overview/database.html

Berg, K. L., Seymour, T., & Goel, R. (2013). History Of Databases. International Journal of Management & Information Systems (IJMIS), 17(1), 29–36. https://doi.org/10.19030/IJMIS.V17I1.7587

Boronski, R., & Bocewicz, G. (2014). Relational Database Index Selection Algorithm. In Communications in Computer and Information Science (Vol. 431). https://doi.org/10.1007/978-3-319-07941-7_34

Calle, J., Sáez, Y., & Cuadra, D. (2011). An evolutionary approach to the index selection problem. Proceedings of the 2011 3rd World Congress on Nature and Biologically Inspired Computing, NaBIC 2011, 485–490. https://doi.org/10.1109/NABIC.2011.6089637

Castellano Lendínez, L. (2019). Kanban. Metodología para aumentar la eficiencia de los procesos. 3C Tecnología. Glosas de innovación aplicadas a la pyme. 8(1), 30–41. https://doi.org/10.17993/3ctecno/2019

Chaudhuri, S., Datar, M., & Narasayya, V. (2004). Index selection for databases: A hardness study and a principled heuristic solution. IEEE Transactions on Knowledge and Data Engineering, 16(11), 1313–1323. https://doi.org/10.1109/TKDE.2004.75

F., CoddE. (1970). A relational model of data for large shared data banks. Communications of the ACM, 13(6), 377–387. https://doi.org/10.1145/362384.362685

Git. (n.d.). Recuperado el 25 de octubre de 2021, a partir de https://git-scm.com/

Gómez, M. G.-, Danglot-Banck, C., & Vega-Franco, L. (2003). Sinopsis de pruebas estadísticas no paramétricas. Cuándo usarlas. Revista Mexicana de Pediatría, 70(2), 91–99.

Google Code Archive - Long-term storage for Google Code Project Hosting. (s.f.). Recuperado el 4 de abril del 2022, de https://code.google.com/archive/p/northwindextended/

Gupta, M., & Badal, D. (2012). COMPARATIVE STUDY OF INDEXING TECHNIQUES IN DBMS.

GWO. (s.f.). Recuperado el 18 de mayo del 2022, de https://seyedalimirjalili.com/gwo

How to conduct a systematic review from beginning to end - Covidence. (s.f.). Recuperado el 24 de agosto del 2021, de https://www.covidence.org/blog/how-to-conduct-a-systematic-review-from-beginning-to-end/?campaignid=13271466385&adgroupid=123024099579&adid=524233276792

IntelliJ IDEA overview | IntelliJ IDEA. (n.d.). Recuperado el 24 de octubre de 2021, a partir de https://www.jetbrains.com/help/idea/discover-intellij-idea.html#IntelliJ-IDEA-supported-languages

Katoch, S., Sumit, &, Chauhan, S., & Kumar, V. (2021). A review on genetic algorithm: past, present, and future. Multimedia Tools and Applications, 80, 8091–8126. https://doi.org/10.1007/s11042-020-10139-6

Kołaczkowski, P., & Rybiński, H. (2009). Automatic index selection in RDBMS by exploring query execution plan space. In Studies in Computational Intelligence (Vol. 223). https://doi.org/10.1007/978-3-642-02190-9_1

Kołaczkowski, P., & Rybiński, H. (2011a). An interactive tool for automatic index selection in relational database management systems. In Studies in Computational Intelligence (Vol. 369). https://doi.org/10.1007/978-3-642-22732-5_6

Kołaczkowski, P., & Rybiński, H. (2011b). Online index selection in RDBMS by evolutionary approach. In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics): Vol. 6861 LNCS (Issue PART 2). https://doi.org/10.1007/978-3-642-23091-2_41

Korytkowski, M., Gabryel, M., Nowicki, R., & Scherer, R. (2004). Genetic algorithm for database indexing. Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science), 3070, 1142–1147. https://doi.org/10.1007/978-3-540-24844-6_179

Kovačević, V., & Filipič, B. (2006). A genetic algorithm with an adaptation mechanism for database index optimization. Bioinspired Optimization Methods and Their Applications - Proceedings of the 2nd International Conference on Bioinspired Optimization Methods and Their Applications, BIOMA 2006, 111–120.

Kratica, J., Ljubić, I., & Tošić, D. (2003). A Genetic Algorithm for the Index Selection Problem. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2611, 280–290. https://doi.org/10.1007/3-540-36605-9_26

lerochka/chinook-database: Sample database for SQL Server, Oracle, MySQL, PostgreSQL, SQLite, DB2. (n.d.). Recuperado el 11 de abril del 2022, de https://github.com/lerocha/chinook-database

Lieponienė, J. (2020). Recent Trends in Database Technology. Baltic J. Modern Computing, 8(4), 551–559. https://doi.org/10.22364/bjmc.2020.8.4.06

Mirjalili, S., Mirjalili, S. M., & Lewis, A. (2014). Grey Wolf Optimizer. Advances in Engineering Software, 69, 46–61. https://doi.org/10.1016/J.ADVENGSOFT.2013.12.007

MySQL :: Employees Sample Database. (n.d.). Recuperado el 11 de abril del 2022, de https://dev.mysql.com/doc/employee/en/

MySQL :: MySQL 8.0 Reference Manual :: 8.9.4 Index Hints. (s.f.). Recuperado el 11 de abril del 2022, de https://dev.mysql.com/doc/refman/8.0/en/index-hints.html

Naik, S. (2018). TIER: Table index evaluator and recommender - A proposed model to improve transaction performance in distributed heterogeneous database. 2017 International Conference on Soft Computing and Its Engineering Applications: Harnessing Soft Computing Techniques for Smart and Better World, IcSoftComp 2017, 2018-January, 1–8. https://doi.org/10.1109/ICSOFTCOMP.2017.8280085

Neuhaus, P., Couto, J., Wehrmann, J., Ruiz, D. D., & Meneguzzi, F. (2019). GADIS: A genetic algorithm for database index selection. Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE, 2019-July, 39–42. https://doi.org/10.18293/SEKE2019-135

Nita, S., & Kartikawati, S. (2020). Analysis Of The Impact Narrative Algorithm Method, Pseudocode And Flowchart Towards Students Understanding Of The Programming Algorithm Courses. IOP Conference Series: Materials Science and Engineering, 835(1), 012044. https://doi.org/10.1088/1757-899X/835/1/012044

Pedrozo, W. G., & Vaz, M. S. M. G. (2014). A Tool for Automatic Index Selection in Database Management Systems. 2014 International Symposium on Computer, Consumer and Control, 1061–1064. https://doi.org/10.1109/IS3C.2014.277

¿Qué es Java y para qué es necesario? (n.d.). Recuperado el 24 de octubre de 2021, a partir de https://www.java.com/es/download/help/whatis_java.html

RDBMS Definition. (s.f.). Recuperado el 26 de setiembre de 2021, a partir de https://techterms.com/definition/rdbms

Response time - IBM Documentation. (s.f.). Recuperado el 29 de setiembre de 2021, a partir de https://www.ibm.com/docs/en/informix-servers/12.10?topic=performance-response-time

RStudio - RStudio. (n.d.). Recuperado el 27 de octubre de 2021, a partir de https://www.rstudio.com/products/rstudio/

Sánchez, M., Bahamondez, E. V., & de Clunie, G. T. (2020). Use of PSeInt in teaching programming: A case study. ACM International Conference Proceeding Series, Part F166737. https://doi.org/10.1145/3401895.3402083

Silberschatz, A., Korth, H. F., & Sudarshan, S. (2019). DatabaseSystem Concepts.

Space requirements for indexes - IBM Documentation. (s.f.). Recuperado el 11 de abril de 2022, a partir de https://www.ibm.com/docs/en/db2/11.5?topic=indexes-space-requirements

What Is a Relational Database | Oracle. (s.f.). Recuperado el 26 de setiembre de 2021, a partir de https://www.oracle.com/database/what-is-a-relational-database/

Wijesiriwardana, C., & Firdhous, M. F. M. (2019). An Innovative Query Tuning Scheme for Large Databases. 2019 International Conference on Data Science and Engineering, ICDSE 2019, 154–159. https://doi.org/10.1109/ICDSE47409.2019.8971483

Anexos

Anexo A: Formulario de extracción

El formulario se podrá apreciar en el siguiente enlace:

https://docs.google.com/spreadsheets/d/1oHLtXj-M07n1ImUPMrq6c0_G9uLeyj8Z63Knu-8k2rk/edit?usp=sharing

Anexo B: Plan de proyecto

- **Justificación**

Las bases de datos procesan y almacenan cada vez más datos. Por lo mismo, con el fin de consultar de manera eficiente estas grandes cantidades de datos, los sistemas de administración de bases de datos ofrecen estructuras de datos, como lo son los índices, para mejorar los tiempos de ejecución de dichas consultas (Kołaczkowski & Rybiński, 2011a). Sin embargo, resulta importante seleccionar correctamente las columnas a indexar, debido a que el índice afecta el tiempo de inserción y actualización de datos, así como aumenta el consumo de disco, siendo una tarea compleja (Neuhaus et al., 2019). En el caso ideal, todas las columnas consultadas con mayor frecuencia se deben indexar. No obstante, es bastante complejo encontrar un equilibrio entre el rendimiento y el almacenamiento requerido, por lo que en muchos casos es el administrador de base de datos quien toma la decisión final sobre la estrategia de indexación (Neuhaus et al., 2019). Este problema se conoce formalmente en la literatura como el problema de selección de índices, y es clasificado como “NP-hard” dado su complejidad (Kołaczkowski & Rybiński, 2009). De igual manera, existen varios factores que influyen en la selección como el tipo de dato, la estructura de tabla, el espacio en disco, frecuencia de consultas, entre otros (Naik, 2018).

Se puede afirmar que, mientras más variables intervengan, más complejo resulta el problema. Por lo mismo, el enfoque aplicado a lo largo de los años por varios autores ha sido el uso de algoritmos, en especial el uso de algoritmos metaheurísticos (Kratica et al., 2003; Chaudhuri et al., 2004; Korytkowski et al., 2004; Kovačević & Filipič, 2006; Kołaczkowski & Rybiński, 2009, 2011b, 2011a; Calle et al., 2011; Boronski & Bocewicz, 2014; Neuhaus et al., 2019). Dentro de este grupo de soluciones, se destaca la concurrente aparición de los algoritmos genéticos, los que, si bien se han obtenido buenos resultados, estos han sido bastante utilizados comercialmente a lo largo de los últimos 20 años (Katoch et al., 2021). Así mismo, los sistemas de bases de datos, como Oracle, DB2 y SQL Server, incluyen los denominados “Asesores de índices”, capaces de analizar la carga de trabajo y generar recomendaciones para la creación de índices (Kovačević & Filipič, 2006).

El presente proyecto de fin de carrera lo que se busca es hallar una manera eficiente de conseguir resultados cercanos al óptimo para la selección de índices, tomando en consideración diferentes aspectos como el tiempo de respuesta y el espacio en disco.

En ese sentido, se pretende implementar una nueva alternativa de solución que permita optimizar la selección de índices por medio del uso de un algoritmo de lobo gris, el cual ha demostrado tener buenos resultados en tareas de asignación de similar complejidad y siendo competitivo frente a metaheurísticas conocidas como la optimización de enjambre de

partículas, el algoritmo de búsqueda gravitacional, la evolución diferencial, la programación evolutiva y estrategias de evolución (Mirjalili et al., 2014). Adicionalmente, este algoritmo no ha sido aplicado directamente en la tarea de selección de índices.

- **Viabilidad**
  - **Técnica**

    La viabilidad técnica de este proyecto de fin de carrera se respalda en la familiaridad y experiencia del tesista con las herramientas a utilizar, como el lenguaje de programación Java, Kanban, entre otros. Además, la existencia de documentación extensa en Internet sobre estas herramientas proporciona recursos adicionales para abordar posibles desafíos técnicos. La experiencia previa del tesista en el uso de estas herramientas durante su formación en Ingeniería Informática brinda una base sólida para el desarrollo del proyecto. Además, la asesoría continua del mentor estará disponible para abordar cualquier pregunta o inquietud relacionada con los algoritmos metaheurísticos bioinspirados.
  - **Temporal**

    El presente trabajo de fin de carrera tendrá una duración aproximada de 11 meses, comenzando en agosto de 2021 y finalizando en junio de 2022. Esto se puede verificar en la sección "Cronograma del proyecto", que se encuentra más adelante.
  - **Económica**

    Este proyecto es viable económicamente debido a que no requiere inversión para el desarrollo, ya que todas las herramientas de software utilizadas son gratuitas o cuentan con licencias gratuitas para estudiantes.

- **Alcance**

El presente proyecto de tesis se ubica en el área de ciencias de la computación, específicamente en la rama de algoritmos de optimización. Su objetivo principal es implementar un algoritmo lobo gris que ofrezca una solución al problema de selección de índices lógicos en bases de datos relacionales. Este algoritmo se centrará en proporcionar el índice óptimo para una tabla de base de datos relacional, optimizando el consumo de recursos, especialmente en términos de tiempo de respuesta y espacio en disco. Este enfoque se dirigirá específicamente a consultas que involucran operaciones directamente relacionadas con la información de almacenamiento de la base de datos, como las búsquedas. Cabe destacar que las operaciones posteriores, como el ordenamiento, aunque afectan el rendimiento general, emplean otro tipo de recursos, como la memoria.

En tal sentido, el desarrollo del proyecto se realizará usando la metodología Kanban. En primer lugar, se identificarán las variables y restricciones más relevantes para este problema. A continuación, con el reconocimiento completo, se procederá a definir la función objetivo.

Seguidamente, se implementará tanto el algoritmo lobo gris seleccionado como el algoritmo genético encontrado de la literatura. Se desarrollarán ambos algoritmos con el motivo de compararlos y validar la eficiencia del algoritmo lobo gris propuesto, utilizando experimentación numérica.

Por último, se calibrarán las variables y constantes del algoritmo propuesto, y se presentarán los resultados obtenidos.

Para este proyecto de fin de carrera se deben considerar los siguientes puntos:

- Se realizará una adaptación del algoritmo del lobo gris para el problema de selección de índices.
- Los algoritmos tendrán los mismos parámetros de entrada y salida.
- Dada la naturaleza de los algoritmos metaheurísticos, los resultados serán cercanos al óptimo, más no la mejor solución.
- El algoritmo genético se implementará en base a las investigaciones realizadas por los autores en el tema, extraído de la revisión bibliográfica.
- La interacción del usuario será mediante una interfaz gráfica para el ingreso de datos de entrada y la visualización de los resultados obtenidos de los algoritmos.
- No se desarrollará un sistema de información.
- No se implementará una base de datos.
- No se usará data real de una empresa para ninguna ejecución.
- No se realizará análisis de datos.

- **Limitaciones**

El proyecto de fin de carrera presenta algunas limitaciones, entre las cuales se destaca que el tiempo de ejecución de los algoritmos está directamente influenciado por las características del equipo en el que se ejecuten y por la complejidad inherente de dichos algoritmos.

- **Riesgos**

Tabla 12. Riesgos del proyecto

Descripción	Síntomas	P	I	S	Mitigación	Contingencia
Ausencia de energía eléctrica al momento de realizar	Mantenimientos de circuitos eléctricos por parte del	1	4	4	Revisión con anticipación de los cortes programados	Reprogramar actividades afectadas.

actividades.	proveedor.				por el proveedor de energía eléctrica.	
Los especialistas no realizan la validación de los entregables en los tiempos pactados.	Especialistas no responden a mensajes y correos previos a la fecha de entrega establecida.	2	4	8	Programar reuniones de revisión de manera oportuna, validando fechas con los especialistas.	Solicitar ayuda de un especialista externo para la revisión de entregables.
La estimación del proyecto no se cumple debido a demoras no consideradas.	Los entregables y fechas no se cumplen de acuerdo al cronograma.	2	3	6	Tratar de avanzar con anticipación los entregables en un plazo estimado de una semana.	Replantear el cumplimiento semanal bajo las nuevas consideraciones.

Leyenda:

- P: Nivel de posibilidad (1: Muy bajo, 2: Bajo, 3: Medio, 4: Alto, 5: Muy alto).
- I: Impacto (1: Muy bajo, 2: Bajo, 3: Medio, 4: Alto, 5: Muy alto).
- S: Severidad = Probabilidad * Impacto.

- **Estructura de descomposición de trabajo**

En la Figura 57 se muestra la descomposición del trabajo, detallando su estructura y organización.

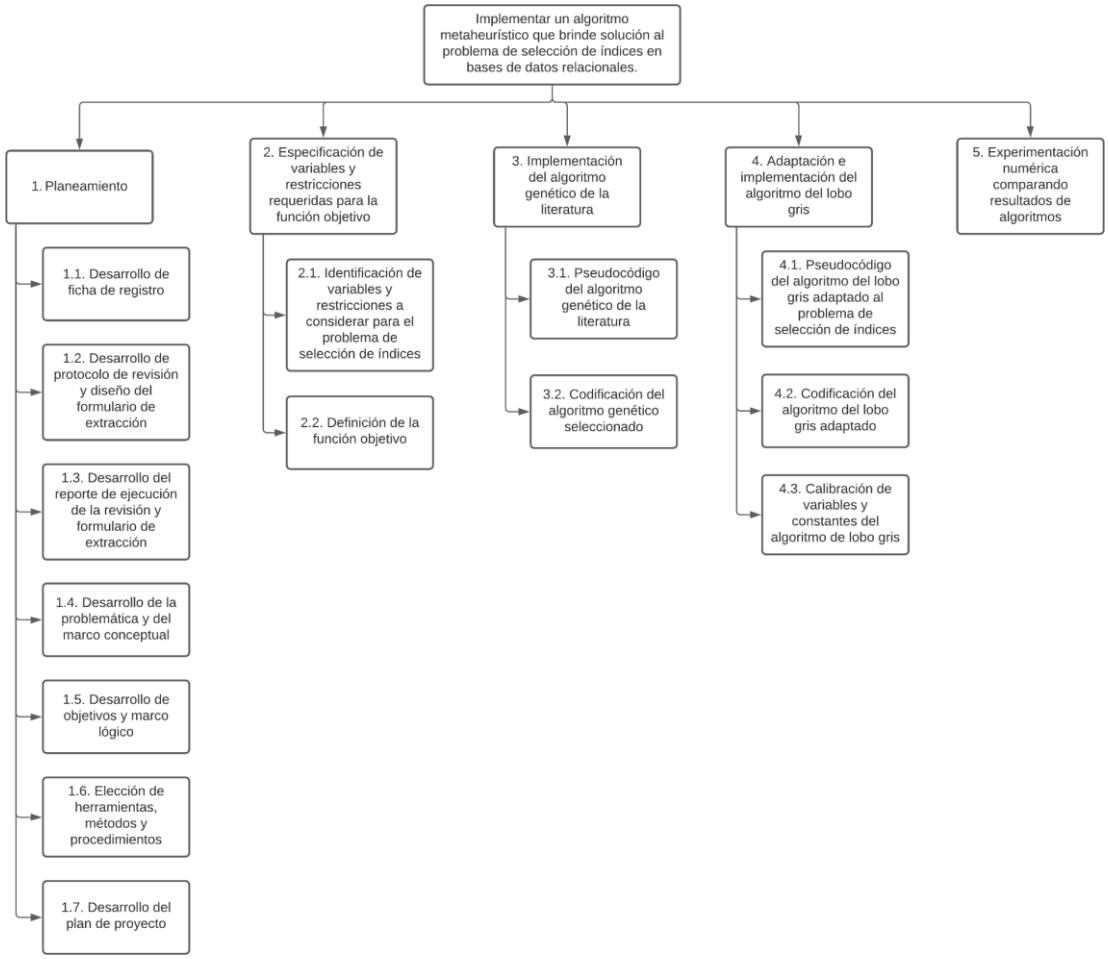


Figura 57. Estructura de descomposición de trabajo

- **Lista de tareas**

En la Tabla 13, se proporciona un desglose detallado de las tareas incluidas en la planificación del proyecto de fin de carrera.

Tabla 13. Lista de tareas

Tarea	**Duración estimada (días)**	**Esfuerzo asociado (horas persona)**	**Costo estimado**
Entregable 1: Planificación.			
Entregable 1.1: Creación de ficha de registro.			
Desarrollo de ficha de registro.	2	7	210
Encuentro con el asesor.	1	1	150
Evaluación por parte del asesor.	1	2	120

Entregable 1.2: Desarrollo de protocolo de revisión y diseño del formulario de extracción.			
Desarrollo de protocolo de revisión y diseño del formulario de extracción.	3	15	450
Encuentro con el asesor.	1	1	150
Implementación de ajustes en el entregable anterior	1	2	60
Evaluación por parte del asesor.	1	2	120
Entregable 1.3: Desarrollo del reporte de ejecución de la revisión y formulario de extracción.			
Creación del informe de ejecución de la revisión y del formulario de extracción.	3	15	450
Encuentro con el asesor.	1	1	150
Implementación de ajustes en el entregable anterior	1	2	60
Evaluación por parte del asesor.	1	2	120
Entregable 1.4: Desarrollo de la problemática y del marco conceptual.			
Desarrollo de la problemática y del marco conceptual.	6	12	360
Encuentro con el asesor.	1	1	150
Implementación de ajustes en el entregable anterior	1	2	60
Evaluación por parte del asesor.	1	2	120
Entregable 1.5: Desarrollo de objetivos y marco lógico.			
Desarrollo de objetivos y marco lógico.	4	8	240
Encuentro con el asesor.	1	1	150
Implementación de ajustes en el entregable anterior	1	2	60
Evaluación por parte del asesor.	1	2	120
Entregable 1.6: Elección de herramientas, métodos y procedimientos.			
Elección de herramientas, métodos y procedimientos.	3	8	240
Encuentro con el asesor.	1	1	150

Implementación de ajustes en el entregable anterior	1	2	60
Evaluación por parte del asesor.	1	2	120
Entregable 1.7: Desarrollo del plan de proyecto.			
Desarrollo del plan de proyecto.	4	10	300
Encuentro con el asesor.	1	1	150
Implementación de ajustes en el entregable anterior	1	2	60
Evaluación por parte del asesor.	1	2	120
Entregable 2: Especificación de variables y restricciones requeridas para la función objetivo.			
Entregable 2.1: Identificación de variables y restricciones a considerar para el problema de selección de índices.			
Elaboración del documento que contiene la lista de variables y restricciones.	5	10	300
Encuentro con el asesor.	1	1	150
Validación del documento por parte del especialista en base de datos.	2	2	240
Entregable 2.2: Definición de la función objetivo.			
Desarrollo del documento con la definición de la función objetivo.	5	10	300
Encuentro con el asesor.	1	1	150
Validación del documento por parte del especialista en base de datos y de algoritmia metaheurística.	1	4	480
Entregable 3: Implementación del algoritmo genético de la literatura.			
Entregable 3.1: Pseudocódigo del algoritmo genético de la literatura.			
Desarrollo del documento que contiene el pseudocódigo del algoritmo genético de la literatura.	10	15	450
Encuentro con el asesor.	1	1	150
Validación del documento por parte del especialista en algoritmia genética.	1	2	240
Entregable 3.2: Codificación del algoritmo genético seleccionado.			

Codificación del algoritmo genético.	15	35	1050
Encuentro con el asesor.	1	1	150
Realización de pruebas unitarias.	4	8	240
Validación del documento del código por parte del especialista en algoritmia genética.	1	2	240
Entregable 4: Adaptación e implementación del algoritmo del lobo gris.			
Entregable 4.1: Pseudocódigo del algoritmo del lobo gris adaptado al problema de selección de índices.			
Desarrollo del documento que contiene el pseudocódigo del algoritmo de lobo gris a adaptar.	10	25	750
Desarrollo del documento que contiene la ejecución de las pruebas de flujo de datos del diseño del algoritmo.	5	8	240
Encuentro con el asesor.	1	1	150
Validación del documento por parte del especialista en algoritmia metaheurística.	1	2	240
Entregable 4.2: Codificación del algoritmo del lobo gris adaptado.			
Codificación del algoritmo de lobo gris adaptado al problema de selección de índices.	15	45	1350
Encuentro con el asesor.	1	1	150
Realización de pruebas unitarias.	4	8	240
Validación del documento del código por parte del especialista en algoritmia metaheurística.	1	2	240
Entregable 4.3: Calibración de variables y constantes del algoritmo de lobo gris.			
Desarrollo del documento de variables y constantes del algoritmo de lobo gris.	9	30	900
Encuentro con el asesor.	1	1	150
Validación del documento por parte del especialista en algoritmia.	1	2	240
Entregable 5: Experimentación numérica comparando resultados de algoritmos.			
Desarrollo del informe de experimentación numérica.	10	25	750
Encuentro con el asesor.	1	1	150

Validación del documento por parte del especialista en algoritmia.	1	2	240

- **Cronograma**

En la tabla 14, se detalla el cronograma del presente proyecto de fin de carrera, detallando las fechas de inicio y fin de cada actividad.

Tabla 14. Cronograma

Actividad	**Inicio**	**Fin**
Objetivo específico 1: Definir las variables y restricciones necesarias del problema para el establecimiento de la función objetivo.		
Resultado esperado 1: Definición de las variables y restricciones más relevantes a tomar en cuenta en la selección de índices.	10/01/2022	20/01/2022
Desarrollo del documento con la lista de variables y restricciones.	10/01/2022	17/01/2022
Encuentro con el asesor.	17/01/2022	18/01/2022
Validación del documento por parte del especialista en base de datos.	18/01/2022	20/01/2022
Resultado esperado 2: Definición de la función objetivo que ayude con la optimización.	21/01/2022	30/01/2022
Desarrollo del documento con la definición de la función objetivo.	21/01/2022	28/01/2022
Encuentro con el asesor.	28/01/2022	29/01/2022
Validación del documento por parte del especialista en base de datos y de algoritmia metaheurística.	29/01/2022	31/01/2022
Objetivo específico 2: Adaptar un algoritmo lobo gris que brinde una buena selección de índices.		
Resultado esperado 3: Pseudocódigo del algoritmo lobo gris propuesto.	01/02/2022	21/02/2022
Desarrollo del documento que contiene el pseudocódigo del algoritmo de lobo gris a adaptar.	01/02/2022	12/02/2022
Desarrollo del documento que contiene la ejecución de las pruebas de flujo de datos del diseño del algoritmo.	12/02/2022	18/02/2022
Encuentro con el asesor.	18/02/2022	19/02/2022
Validación del documento por parte del especialista en algoritmia metaheurística.	19/02/2022	21/02/2022

Resultado esperado 4: Codificación del algoritmo lobo gris adaptado al problema de selección de índices.	21/02/2022	17/03/2022
Codificación del algoritmo de lobo gris adaptado al problema de selección de índices.	21/02/2022	09/02/2022
Encuentro con el asesor.	09/02/2022	10/02/2022
Realización de pruebas unitarias.	10/02/2022	15/03/2022
Validación del documento del código por parte del especialista en algoritmia metaheurística.	15/03/2022	17/03/2022
Resultado esperado 5: Calibración de variables y constantes del algoritmo lobo gris.	17/03/2022	31/03/2022
Desarrollo del documento de variables y constantes del algoritmo de lobo gris.	17/03/2022	28/03/2022
Encuentro con el asesor.	28/03/2022	29/03/2022
Validación del documento por parte del especialista en algoritmia.	29/03/2022	31/03/2022
Objetivo específico 3: Comparar el desempeño del algoritmo metaheurístico propuesto contra la solución más relevante encontrada en el estado del arte para el problema de selección de índices con el fin de validar los resultados.		
Resultado esperado 6: Pseudocódigo del algoritmo genético obtenido del estado del arte.	01/04/2022	21/04/2022
Desarrollo del documento que contiene el pseudocódigo del algoritmo genético de la literatura.	01/04/2022	12/04/2022
Encuentro con el asesor.	12/04/2022	18/04/2022
Validación del documento por parte del especialista en algoritmia genética.	19/04/2022	21/05/2022
Resultado esperado 7: Codificación del algoritmo genético seleccionado.	21/04/2022	14/05/2022
Codificación del algoritmo genético.	21/04/2022	06/05/2022
Encuentro con el asesor.	06/05/2022	07/05/2022
Realización de pruebas unitarias.	07/05/2022	12/05/2022
Validación del documento del código por parte del especialista en algoritmia genética.	12/05/2022	14/05/2022
Resultado esperado 8: Experimentación numérica para la comparación entre el algoritmo propuesto y el algoritmo genético.	14/05/2022	29/05/2022
Desarrollo del informe de experimentación numérica.	14/05/2022	26/05/2022

Encuentro con el asesor.	26/05/2022	27/05/2022
Validación del documento por parte del especialista en algoritmia.	27/05/2022	29/05/2022

- **Recursos**

En esta sección, se presentan los recursos necesarios para el desarrollo del presente proyecto de fin de carrera.

- **Persona involucrada y necesidades de capacitación**
  - **Tesista: Fernando Guillermo Verástegui Sánchez**
    Alumno que elabora el proyecto de tesis con el objetivo de obtener el título universitario.
  - **Asesor: Msc. Rony Cueva Moscoso**
    Especialista encargado de guiar al tesista en el desarrollo de proyectos de tesis.
- **Materiales requeridos para el proyecto**
  No aplica.
- **Estándares utilizados en el proyecto**
  - **Kanban**

  Se utilizarán algunas buenas prácticas de este estándar para el desarrollo organizado y eficaz de los entregables.
- **Equipamiento requerido**
  - **Laptop personal**

  Esencial para la elaboración del proyecto, la redacción de la documentación del mismo, la codificación y la experimentación numérica.
- **Herramientas requeridas**
  - **PseInt**

  Se usará con el fin de diagramar el flujo de ejecución de los algoritmos.
  - **Java**

  Se utilizará como lenguaje de programación para la implementación de los algoritmos.
  - **IntelliJ IDEA**

  Se empleará este entorno de trabajo para la implementación de los algoritmos.
  - **Git**

  Se utilizará como repositorio del código fuente de los algoritmos.
  - **RStudio**

Se utilizará para realizar diversos análisis estadísticos sobre los resultados de los algoritmos desarrollados.

- **Licencia en herramientas de ofimática**

Se empleará para redactar la documentación del proyecto.

- **Licencia pro de Zoom**

Se usará para tener y grabar las reuniones síncronas con el asesor y con los especialistas en algoritmia y base de datos.

- **Costeo**

Tabla 15. Costeo

Ítem	Descripción	Unidad	Cantidad	Valor Unidad (S/.)	Monto Parcial (S/.)	Monto Total (S/.)
0.	**Costo total del proyecto**	**---**	**---**	**---**	**---**	**18,514**
1.	**Estudiante o tesistas**	**---**	**---**	**---**	**---**	**8,100**
1.1	Fernando Guillermo Verástegui Sánchez	Horas	270	30	8,100	
2.	**Otros participantes**	**---**	**---**	**---**	**---**	**6,030**
2.1	Msc. Rony Cueva Moscoso	Horas	29	150	4,350	
2.2	Especialista en base de datos	Horas	4	120	480	
2.3	Especialista en algoritmia	Horas	10	120	1200	
3.	**Bienes y equipos**	**---**	**---**	**---**	**---**	**4,384**
3.1	Laptop	Equipo	1	3,300	3,300	
3.2	Licencia en herramientas de ofimática	Meses	11	22[1]	242	
4.3	Licencia pro en Zoom	Meses	11	60[2]	660	

[1] Basado en el costo del plan mensual de Microsoft Office 365 Personal

[2] Basado en el costo del plan mensual pro de Zoom solarizado con el tipo de cambio al 3 de noviembre de 2021

Anexo C: Documento de validación de las variables y restricciones por un especialista en bases de datos

Documento de validación de resultados alcanzados

Yo, Rony Cueva Moscoso, con DNI: 09942265, por medio del presente hago constar que he leído y evaluado el resultado alcanzado 1 referente a la "Definición de las variables y restricciones más relevantes a tomar en cuenta en la selección de índices", correspondiente al objetivo 1 "Definir las variables y restricciones necesarias del problema para el establecimiento de la función objetivo", presentado por el estudiante Fernando Guillermo Verástegui Sánchez como parte de la realización del proyecto de tesis titulado "Algoritmo lobo gris para la selección de índices en bases de datos relacionales". Por medio de este documento certifico que el resultado alcanzado es válido desde el punto de vista de la administración de bases de datos.

Firma o Firma electrónica:

Anexo D: Registro de las pruebas ejecutadas en relación a la función objetivo

Las pruebas de la función objetivo se podrán apreciar en el siguiente enlace:

https://docs.google.com/spreadsheets/d/1WYgypVTLO13_BO7pE3pPUUkC-oUTxGnzr47sX3Eynj0/edit?usp=sharing

Anexo E: Certificación de validación de la función objetivo, autenticado por un especialista en bases de datos y algoritmia metaheurística

Documento de validación de resultados alcanzados

Yo, Rony Cueva Moscoso, con DNI: 09942265, por medio del presente hago constar que he leído y evaluado el resultado alcanzado 2 referente a la "Definición de la función objetivo que ayude con la optimización", correspondiente al objetivo 1 "Definir las variables y restricciones necesarias del problema para el establecimiento de la función objetivo", presentado por el estudiante Fernando Guillermo Verástegui Sánchez como parte de la realización del proyecto de tesis titulado "Algoritmo lobo gris para la selección de índices en bases de datos relacionales". Por medio de este documento certifico que el resultado alcanzado es válido desde el punto de vista de la administración de bases de datos y de la algoritmia metaheurística.

Firma o Firma electrónica:

Anexo F: Informe de pruebas del flujo de datos del diseño del algoritmo

Las pruebas de flujo de datos se podrán apreciar en el siguiente enlace:

https://docs.google.com/spreadsheets/d/1MxU8vw8k3ZTMecHq299C14OGdmTnqYYWZ8OmD_Ks-Do/edit?usp=sharing

Anexo G: Certificación de validación del pseudocódigo del algoritmo lobo gris, autenticado por un experto en algoritmia metaheurística

Documento de validación de resultados alcanzados

Yo, Rony Cueva Moscoso, con DNI: 09942265, por medio del presente hago constar que he leído y evaluado el resultado alcanzado 3 referente a la "Pseudocódigo del algoritmo lobo gris propuesto", correspondiente al objetivo 2 "Adaptar un algoritmo lobo gris que brinde una buena selección de índices", presentado por el estudiante Fernando Guillermo Verástegui Sánchez como parte de la realización del proyecto de tesis titulado "Algoritmo lobo gris para la selección de índices en bases de datos relacionales". Por medio de este documento certifico que el resultado alcanzado es válido desde el punto de vista de la algoritmia metaheurística.

Firma o Firma electrónica:

Anexo H: Código fuente del algoritmo lobo gris

El código fuente se podrá apreciar en el siguiente enlace:
https://github.com/PeterThonks/AlgoritmoLoboGris

Anexo I: Documento de pruebas unitarias sobre el algoritmo lobo gris

Las pruebas unitarias se podrán apreciar en el siguiente enlace:

https://docs.google.com/document/d/1Pzu_tEgQ49-fD9xLtWAYaYeudcfVT_osQ9gSZnA7hZ8/edit?usp=sharing

Anexo J: Certificación de validación de la implementación del algoritmo lobo gris, respaldada por la firma de un especialista en algoritmia metaheurística

Documento de validación de resultados alcanzados

Yo, Rony Cueva Moscoso, con DNI: 09942265, por medio del presente hago constar que he leído y evaluado el resultado alcanzado 4 referente a la "Codificación del algoritmo lobo gris adaptado al problema de selección de índices", correspondiente al objetivo 2 "Adaptar un algoritmo lobo gris que brinde una buena selección de índices", presentado por el estudiante Fernando Guillermo Verástegui Sánchez como parte de la realización del proyecto de tesis titulado "Algoritmo lobo gris para la selección de índices en bases de datos relacionales". Por medio de este documento certifico que el resultado alcanzado es válido desde el punto de vista de la algoritmia metaheurística.

Firma o Firma electrónica:

Anexo K: Archivos CSV de la calibración de variables

Los archivos csv se podrá apreciar en el siguiente enlace:
https://drive.google.com/file/d/14SII5BuaKNbVYqUwpTus0gpRun6b6cqh/view?usp=sharing

Anexo L: Certificado de validación de la calibración de variables del algoritmo lobo gris, autenticado por un experto en algoritmia

Documento de validación de resultados alcanzados

Yo, Rony Cueva Moscoso, con DNI: 09942265, por medio del presente hago constar que he leído y evaluado el resultado alcanzado 5 referente a la "Calibración de variables y constantes del algoritmo lobo gris", correspondiente al objetivo 2 "Adaptar un algoritmo lobo gris que brinde una buena selección de índices", presentado por el estudiante Fernando Guillermo Verástegui Sánchez como parte de la realización del proyecto de tesis titulado "Algoritmo lobo gris para la selección de índices en bases de datos relacionales". Por medio de este documento certifico que el resultado alcanzado es válido desde el punto de vista de la algoritmia.

Firma o Firma electrónica:

Anexo M: Documento de validación del pseudocódigo del algoritmo genético de la literatura firmado por un especialista en algoritmia metaheurística

Documento de validación de resultados alcanzados

Yo, Rony Cueva Moscoso, con DNI: 09942265, por medio del presente hago constar que he leído y evaluado el resultado alcanzado 5 referente a la "Pseudocódigo del algoritmo genético obtenido del estado del arte", correspondiente al objetivo 3 "Comparar el desempeño del algoritmo metaheurístico propuesto contra la solución más relevante encontrada en el estado del arte para el problema de selección de índices con el fin de validar los resultados", presentado por el estudiante Fernando Guillermo Verástegui Sánchez como parte de la realización del proyecto de tesis titulado "Algoritmo lobo gris para la selección de índices en bases de datos relacionales". Por medio de este documento certifico que el resultado alcanzado es válido desde el punto de vista de la algoritmia.

Firma o Firma electrónica:

Anexo N: Código fuente del algoritmo genético

El código fuente se podrá apreciar en el siguiente enlace:
https://github.com/PeterThonks/AlgoritmoGenetico

Anexo O: Documento de pruebas unitarias sobre el algoritmo genético

Las pruebas unitarias se podrán apreciar en el siguiente enlace:

https://docs.google.com/document/d/1W71SEkdn011ESa2BYrV63HPmNqaxRevXFtljSK2_VVY/edit?usp=sharing

Anexo P: Certificado de validación de la implementación del algoritmo genético, autenticado por un especialista en algoritmia metaheurística

Documento de validación de resultados alcanzados

Yo, Rony Cueva Moscoso, con DNI: 09942265, por medio del presente hago constar que he leído y evaluado el resultado alcanzado 7 referente a la "Codificación del algoritmo genético seleccionado", correspondiente al objetivo 3 "Comparar el desempeño del algoritmo metaheurístico propuesto contra la solución más relevante encontrada en el estado del arte para el problema de selección de índices con el fin de validar los resultados", presentado por el estudiante Fernando Guillermo Verástegui Sánchez como parte de la realización del proyecto de tesis titulado "Algoritmo lobo gris para la selección de índices en bases de datos relacionales". Por medio de este documento certifico que el resultado alcanzado es válido desde el punto de vista de la algoritmia.

Firma o Firma electrónica:

Anexo Q: Valores fitness utilizados para la experimentación numérica

Tabla 16. Fitness utilizados para la experimentación numérica

Muestra	Fitness algoritmo Lobo Gris	Fitness algoritmo Genético
1	0.001345	0.001442
2	0.001324	0.001407
3	0.001336	0.001433
4	0.001387	0.001413
5	0.001349	0.001443
6	0.001349	0.001448
7	0.001379	0.001442
8	0.001345	0.001424
9	0.001371	0.001445
10	0.001335	0.001458
11	0.001369	0.001407
12	0.001355	0.001464
13	0.001372	0.001433
14	0.001339	0.001412
15	0.001328	0.001478
16	0.001353	0.001449
17	0.001358	0.001436
18	0.001367	0.001445
19	0.001339	0.001425
20	0.001349	0.001413
21	0.001323	0.001427
22	0.001319	0.001488
23	0.001356	0.001487
24	0.001347	0.001469
25	0.001382	0.001460
26	0.001344	0.001473
27	0.001365	0.001428
28	0.001321	0.001437
29	0.001353	0.001443
30	0.001371	0.001422
31	0.001324	0.001412
32	0.001312	0.001424
33	0.001339	0.001484
34	0.001364	0.001463
35	0.001369	0.001446
36	0.001364	0.001449
37	0.001337	0.001449
38	0.001383	0.001484

39	0.001354	0.001406
40	0.001335	0.001421
41	0.001362	0.001440
42	0.001361	0.001443
43	0.001386	0.001476
44	0.001359	0.001416
45	0.001355	0.001470
46	0.001324	0.001454
47	0.001379	0.001451
48	0.001345	0.001442
49	0.001366	0.001466
50	0.001347	0.001431

Anexo R: Certificado de validación de la experimentación numérica, con la firma de un especialista en algoritmia

Documento de validación de resultados alcanzados

Yo, Rony Cueva Moscoso, con DNI: 09942265, por medio del presente hago constar que he leído y evaluado el resultado alcanzado 8 referente a la "Experimentación numérica para la comparación entre el algoritmo propuesto y el algoritmo genético", correspondiente al objetivo 3 "Comparar el desempeño del algoritmo metaheurístico propuesto contra la solución más relevante encontrada en el estado del arte para el problema de selección de índices con el fin de validar los resultados", presentado por el estudiante Fernando Guillermo Verástegui Sánchez como parte de la realización del proyecto de tesis titulado "Algoritmo lobo gris para la selección de índices en bases de datos relacionales". Por medio de este documento certifico que el resultado alcanzado es válido desde el punto de vista de la algoritmia.

Firma o Firma electrónica: