PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

IMPLEMENTACIÓN DE UN ALGORITMO MEMÉTICO PARA OPTIMIZAR LA ASIGNACIÓN DE TABLAS A UNIDADES DE ALMACENAMIENTO DE BASES DE DATOS RELACIONALES

TESIS PARA OBTENER EL TÍTULO PROFESIONAL DE INGENIERO INFORMÁTICO

AUTOR

Jhamir Arturo Lucas Ramos

ASESOR:

Mg. Rony Cueva Moscoso

Lima, abril, 2022

Resumen

En la actualidad, los sistemas de bases de datos son considerados como un componente fundamental para casi cualquier organización, ya que estos sistemas permiten acceder a información puntual de forma segura y rápida, lo cual es clave para una correcta toma de decisiones y una adecuada atención a los usuarios. Sin embargo, debido al rápido desarrollo de las tecnologías de información, cada vez más sistemas de información generan enormes cantidades de datos y necesitan lidiar con estos de forma eficiente y, dado que las bases de datos relacionales juegan un rol vital en muchos sistemas de información, el rendimiento de estos mismos sistemas depende directamente del rendimiento del sistema de base de datos. En ese sentido, se considera crítico aplicar diversos métodos para optimizar el rendimiento del sistema de base de datos. Uno de estos métodos es la asignación de tablas, el cual consiste en distribuir de manera adecuada a las tablas de una base de datos en los dispositivos de almacenamiento disponibles. Dicho método es útil porque permite mejorar el rendimiento del sistema de base de datos y aprovechar de mejor manera los recursos de hardware disponibles. Sin embargo, muchas veces esta tarea se realiza considerando sólo algunas variables o factores al momento de tomar una decisión. Asimismo, existe una ausencia en el empleo de esta técnica por parte de muchos sistemas modernos. Esto, sumado al hecho de que la asignación suele realizarse de manera manual y también a que los estudios presentes en el estado del arte utilizan, en su gran mayoría, soluciones basadas en heurísticas o cálculos simples, las cuales pueden no brindar buenos resultados, conducen a que se realice una deficiente asignación de tablas a unidades de almacenamiento. Esta situación provoca un bajo rendimiento del sistema de base de datos, un deficiente funcionamiento de la entrada y salida de disco y que las tareas de administración sean más propensas a errores.

Ante esta situación, se torna necesario el uso de métodos que automaticen y optimicen esta tarea, en ese sentido, el presente trabajo de tesis propone el diseño y la implementación de un algoritmo memético que permita optimizar la asignación de tablas a unidades de almacenamiento de bases de datos relacionales.

Tabla de Contenidos

Índice de Figuras

Índice de Tablas

Capítulo 1. Generalidades

1.1 Problemática

1.1.1 Árbol de problemas

En la **Tabla 1** se presenta el árbol de problemas con el detalle de las causas (problemas causas), el problema central (problema central) y los efectos (problemas efectos).

Tabla 1. *Árbol de problemas*

Problemas efectos	Bajo rendimiento en el sistema de base de datos relacional	Bajo rendimiento de entrada/salida de las unidades de almacenamiento de bases de datos relacionales	La tarea de administración se vuelve propensa a errores	
Problema central	Deficiente asignación de tablas en las unidades de almacenamiento de bases de datos relacionales			
Problemas causas	La asignación de tablas a unidades de almacenamiento se realiza frecuentemente considerando solo algunas de las variables o factores más relevantes para esta tarea	Ausencia en el empleo de técnicas de optimización en la tarea de asignación de tablas a unidades de almacenamiento de bases de datos relacionales por parte de herramientas modernas de optimización de sistemas de bases de datos	Los administradores de bases de datos usualmente realizan la asignación de tablas a unidades de almacenamiento de manera manual	Las investigaciones presentes en el estado del arte proponen, en su gran mayoría, métodos que pueden brindar resultados no óptimos

1.1.2 Descripción

La ciencia, los negocios, la educación, la economía, todas las áreas del desarrollo humano "trabajan" con la ayuda constante de los datos. Estos datos, representan hechos, conceptos o instrucciones que deben ser recolectados, procesados e interpretados con el objetivo de obtener información (Sumathi & Esakkirajan, 2007). La recolección de grandes cantidades de datos conllevó a que científicos informáticos busquen la forma de organizarlos, almacenarlos y mantenerlos disponibles cuando sean necesarios. De esta manera, se acuñó el término base de datos para referirse a una colección organizada de datos (Berg et al., 2012). El poder de las bases de datos proviene de un conjunto de conocimiento y tecnología que se desarrolló durante las últimas décadas y se materializó en un software especializado llamado sistema de administración de base de datos, o más coloquialmente, sistema de base de datos (Garcia-Molina et al., 2014). Este sistema permite, entre otras cosas, acceder, manipular y compartir los datos de una base de datos con el objetivo de obtener información (Sumathi & Esakkirajan, 2007).

Hoy en día, los sistemas de base de datos son considerados como la fuente principal para administrar una empresa (Kamatkar et al., 2018), esto sucede debido a que, en el competitivo mundo empresarial, se toma más importancia a la información, considerándola como el activo más valioso de la organización (Wijesiriwardana & Firdhous, 2019), ya que, la información puntual y relevante es la clave para la correcta toma de decisiones y, a su vez, una buena toma de decisiones es la clave para la supervivencia de cualquier organización (Sumathi & Esakkirajan, 2007). Por esta razón, los datos reciben más atención que el hardware, el software e incluso, a veces, más que los recursos humanos (Wijesiriwardana & Firdhous, 2019).

Sin embargo, para ser útiles, los datos en una base de datos deben estar organizados de tal manera que puedan recuperarse cuando sean necesarios (Harrington, 2016), las maneras en cómo se puede organizar los datos en una base de datos son llamadas "modelos". Las bases

de datos pueden seguir distintos “modelos”, tales como relacional, jerárquico, no relacional, entre otros. Muchos de los documentos que las empresas ejecutan para rastrear el inventario, las ventas, las finanzas o incluso realizar proyecciones financieras, provienen de una base de datos relacional que opera detrás de escena (IBM Cloud Education, 2019). Las bases de datos relacionales son las fuentes de datos más usadas en la actualidad y continuarán siendo populares en el futuro (Burbank & Knight, 2019).

Algunas de las posibles razones por las cuales se prefiere este modelo de base de datos son: (i) permite almacenar los datos de manera eficiente, eliminando la redundancia en estos a través de la normalización (Connolly & Begg, 2016) y (ii) permite poder alcanzar eficiencia al momento de acceder y modificar información de una base de datos (Garcia-Molina et al., 2014).

Sin embargo, debido al rápido desarrollo de las tecnologías de información, cada vez más sistemas de información tales como sistemas de telecomunicaciones, de bancos, de educación, de cuidado de la salud, entre otros, generan enormes cantidades de datos y necesitan lidiar con estos de manera eficiente, haciendo que la tarea de almacenar, organizar y manipular los datos se vuelva altamente desafiante (Matalqa & Mustafa, 2016) y, dado que las bases de datos relacionales juegan un rol vital en muchos sistemas de información, el rendimiento de estos mismos sistemas depende directamente del rendimiento del sistema que administra la base de datos relacional (Wijesiriwardana & Firdhous, 2019).

Por lo tanto, mejorar y mantener el rendimiento del sistema de base de datos es considerado como el aspecto más crítico dentro de la administración de un sistema de información (Maabreh, 2018). Ante esta situación, los profesionales de tecnologías de la información encargados de gestionar el sistema de base de datos, o más comúnmente llamados, administradores de base de datos, se enfrentan a demandantes tareas tales como optimizar y monitorear constantemente el sistema de base de datos, diagnosticar la causa raíz de los problemas de rendimiento y tomar medidas rápidas para restaurar el rendimiento de la base de datos (Yoon et al., 2015).

La optimización de base de datos es un proceso complejo, el cual necesita un profundo

conocimiento acerca de la estructura de la base de datos (Myalapalli et al., 2016). Este proceso consiste en identificar la causa del problema de rendimiento y brindar cambios apropiados para reducir o eliminar los efectos que este pudiese ocasionar (Oracle, 2019). Existen diversos factores que pueden afectar el rendimiento de un sistema de base de datos, típicamente, el estrés ocasionado en los recursos de hardware tales como los dispositivos de almacenamiento secundario (unidad de disco duro, unidad de estado sólido, entre otros), el procesador (CPU) y la memoria principal (memoria RAM), puede terminar por afectar significativamente el rendimiento del sistema de base de datos (Fritchey, 2018).

A través de los años, la memoria principal ha llegado a ser cada vez más rápida. Lo mismo puede decirse de los procesadores. Sin embargo, esto no sucedió con los dispositivos de almacenamiento secundario pues, a excepción de algunas mejoras radicales con tecnologías como las unidades de estado sólido (SSD), estos no han cambiado mucho y actualmente son considerados como una de las partes más lentas de cualquier sistema informático (Fritchey, 2018).

Todo sistema de base de datos relacional tiene la función de “leer” o “escribir” datos de o en las unidades de almacenamiento secundario, esta operación es llamada entrada/salida de disco (Oracle, 2019) y debido a las características inherentes de estos dispositivos, comparado con otros recursos, la entrada/salida de disco suele ser el recurso más lento del sistema de base de datos. Es por esta razón que es esencial investigar la manera en cómo se puede optimizar el diseño y configuración del sistema de almacenamiento con el objetivo de lograr un alto rendimiento del sistema de base de datos (Kamatkar et al., 2018). Precisamente en este contexto, se propone el método de asignación de datos, el cual busca almacenar los objetos de bases de datos relacionales (tablas, índices, entre otros) más “apropiados” en las unidades de almacenamiento secundario más “apropiadas” bajo un determinado criterio, con el objetivo de mejorar el rendimiento del sistema base de datos mediante la optimización de la entrada/salida de disco (Wu et al., 2019).

Sin embargo, este método no ha sido muy aplicado por los proveedores de sistemas de administración de bases de datos relacionales modernos, ya que muchas de las herramientas

de optimización de bases de datos que estos ofrecen no contemplan a la asignación de tablas a unidades de almacenamiento disponibles como una técnica de optimización, algunos ejemplos de estas herramientas son: (i) el “asesor de acceso SQL” de Oracle (Oracle, 2019) (ii) el “asesor de mejora de motor de base de datos” de SQL Server (Fritchey, 2018) y (iii) el “asesor de diseño” de IBM DB2 (Allen, 2012). Esto ocasiona que la tarea de asignar las tablas a las unidades de almacenamiento se tenga que hacer de manera manual por los administradores de base de datos, quienes no cuentan con la presencia de una herramienta de software que permita realizar dicha tarea de manera automática, asimismo, en su mayoría éstos administradores suelen confiar en su experiencia pasada o solo en algunas variables o factores estadísticos para poder tomar una decisión (Yoon et al., 2015).

En contraposición a lo descrito anteriormente, el estado del arte de la asignación de datos muestra una serie de diversos métodos propuestos para la optimización de la asignación de objetos a unidades de almacenamiento tales como, por ejemplo, el uso de distintas variaciones de algoritmos de programación dinámica (Canim et al., 2009; Shi et al., 2013), estos métodos son llamados heurísticas y tienen la limitante de que muchas de ellas han sido diseñadas para un problema específico sin posibilidad de generalización o aplicación a problemas similares (Martí, 2003). Por otro lado, se tiene al uso de métodos empíricos de priorización de datos y algoritmos iterativos (Lin et al., 2011; Ou et al., 2014; Wu et al., 2019), los cuales, si bien llegan a tener resultados razonables (Lin et al., 2011) se basan en cálculos simples creados por los propios autores específicamente para los casos que plantean en sus estudios.

Finalmente, se tiene al uso de un algoritmo GRASP (Cueva & Tupia, 2008), este método pertenece a la categoría de algoritmos metaheurísticos, los cuales son mecanismos de alto nivel que pueden proporcionar soluciones apropiadas a problemas de optimización usando menos esfuerzo computacional que los métodos basados en cálculos simples o en heurísticas (Du & Swamy, 2016), sin embargo, el algoritmo GRASP es un método ya consolidado, que ha probado su eficacia sobre una gran colección de problemas (Martí, 2003), incluyendo al problema de asignación de tablas. Esto no sucede con otros algoritmos metaheurísticos

como, por ejemplo, el algoritmo memético, que si bien ha sido empleado con éxito en problemas de similar complejidad (F. Neri et al., 2012), no fue aplicado directamente al problema de asignación de tablas a unidades de almacenamiento. En conclusión, a pesar de que la asignación de datos ha sido bastante estudiada en la literatura, en la mayoría de los casos se han empleado métodos que pueden brindar resultados que no son óptimos para la asignación de tablas a unidades de almacenamiento, pues estos están basados en heurísticas y en cálculos simples.

Los factores descritos previamente ocasionan que la asignación de tablas a unidades de almacenamiento se realice de manera deficiente y no cumpla su labor principal: optimizar el rendimiento de la base de datos, o aún peor, si la distribución de objetos de base de datos se realiza de manera irresponsable, simplemente asignando muchos objetos en distintas unidades de almacenamiento, el resultado puede ser un bajo rendimiento de entrada/salida de disco (Fritchey, 2018). Lo cual, a su vez, significa un deficiente rendimiento del sistema de base de datos en general y, debido a que los administradores de base de datos no son provistos de ayuda suficiente en esta difícil tarea, su único recurso es la prueba y error, lo cual es una solución tediosa, muchas veces sub óptima y propensa a errores (Yoon et al., 2015). En ese sentido, es relevante aplicar nuevas técnicas que puedan automatizar y optimizar la tarea de asignación de tablas a unidades de almacenamiento, como es el caso del algoritmo memético, el cual, como se mencionó anteriormente, es un algoritmo metaheurístico aplicado con éxito en tareas similares (F. Neri et al., 2012), además de que su aplicación en el problema de asignación de tablas a unidades de almacenamiento no ha sido estudiada en la literatura, es por ello que evaluar su rendimiento aplicado a esta tarea es planteado como materia de estudio en el presente proyecto.

1.1.3 Problema seleccionado

Los sistemas de base de datos relacionales son la herramienta principal de muchas organizaciones en la actualidad, es por ello que mejorar y mantener su rendimiento es una tarea muy importante y a la vez desafiante. Una de las principales limitantes de los sistemas

de bases de datos es la entrada/salida de disco, en ese sentido, se planteó una propuesta de optimización llamada “asignación de datos”, sin embargo, muchas veces esta tarea no es considerada en los sistemas de administración de bases de datos, por lo que debe realizarse de manera manual por los administradores de bases de datos, quienes suelen tomar en cuenta sólo algunas variables al momento de realizar el proceso de asignación, por otro lado, en el estado del arte no se proponen métodos adecuados para realizar la optimización de la asignación de tablas a unidades de almacenamiento. Todo esto puede conllevar a una deficiente asignación de las tablas en las unidades de almacenamiento, y, por ende, a un deficiente rendimiento del sistema de base de datos. Por esta razón, es importante aplicar técnicas que puedan optimizar esta tarea, como es el caso del algoritmo memético, el cual es planteado como objeto de estudio aplicado al problema de asignación de tablas a unidades de almacenamiento de base de datos relacionales.

1.2 Objetivos

Como se mencionó anteriormente, el presente proyecto de tesis tiene como materia de estudio a la utilización de una técnica algorítmica (en este caso al algoritmo memético) en la optimización de la asignación de tablas a unidades de almacenamiento de bases de datos relacionales, en ese sentido, a continuación, se presentan los objetivos del proyecto.

1.2.1 Objetivo general

Implementar un algoritmo memético para optimizar la asignación de tablas a unidades de almacenamiento de bases de datos relacionales.

1.2.2 Objetivos específicos

- **O1.** Determinar una relación adecuada entre las variables o factores más relevantes relacionados a la asignación de tablas a unidades de almacenamiento.
- **O2.** Adaptar el algoritmo memético de tal manera que pueda ser aplicado como una técnica de optimización en la asignación de tablas a unidades de almacenamiento.
- **O3.** Desarrollar una herramienta de software que permita la ejecución del algoritmo

memético aplicado a la optimización de la asignación de tablas a unidades de almacenamiento de manera automática.

O4. Comparar el desempeño del algoritmo memético propuesto contra la solución más relevante encontrada en el estado del arte con respecto a la optimización de la asignación de tablas a unidades de almacenamiento con la finalidad de verificar que este brinda buenos resultados y puede ser aplicado con éxito en dicha tarea de optimización.

1.2.3 Resultados esperados

R1. Definición de las variables y restricciones más relevantes a tomar en cuenta para la tarea de asignación de tablas (O1).

R2. Diseño de la función objetivo a ser usada en la optimización de la tarea asignación de tablas a unidades de almacenamiento (O1).

R3. Definición de las estructuras de datos que utilizará el algoritmo memético (O2).

R4. Diseño de un algoritmo memético adaptado al problema de asignación de tablas a unidades de almacenamiento (O2).

R5. Codificación del algoritmo memético adaptado al problema de asignación de tablas a unidades de almacenamiento (O3).

R6. Interfaz gráfica que permita ejecutar el algoritmo memético aplicado al problema de asignación de tablas a unidades de almacenamiento (O3).

R7. Diseño del algoritmo GRASP adaptado al problema de asignación de tablas a unidades de almacenamiento (O4).

R8. Codificación del algoritmo GRASP encontrado en el estado del arte (O4).

R9. Calibración de las variables que se usarán en los algoritmos memético y GRASP (O4).

R10. Experimentación numérica a modo de evaluación del desempeño de los algoritmos memético y GRASP (O4).

1.2.4 Mapeo de objetivos, resultados y verificación

En la **Tabla 2** se muestra de manera resumida al objetivo específico, los resultados esperados asociados a este, los medios de verificación y el indicador objetivamente verificables correspondientes a cada resultado esperado.

Tabla 2. *Mapeo de objetivos, resultados y verificación*

Objetivo (O1): Determinar una relación adecuada entre las variables o factores más relevantes relacionados a la asignación de tablas a unidades de almacenamiento		
Resultado	**Medio de verificación**	**Indicador objetivamente verificable**
(R1) Definición de variables y restricciones más relevantes a tomar en cuenta para la tarea de asignación de tablas	- Documento que contiene la definición de las variables y restricciones	- Revisión y validación al 100% por un especialista en administración de bases de datos
(R2) Diseño de la función objetivo a ser usada en la optimización de la tarea asignación de tablas a unidades de almacenamiento	- Documento que contiene la definición de la función objetivo	- Revisión y validación al 100% por un especialista en administración de base de datos
Objetivo (O2): Adaptar el algoritmo memético de tal manera que pueda ser aplicado como una técnica de optimización en la asignación de tablas a unidades de almacenamiento		
Resultado	**Medio de verificación**	**Indicador objetivamente verificable**
(R3) Definición de la estructura de datos que utilizará el algoritmo memético	- Documento que contiene la definición de las estructuras de datos necesarias para implementar el algoritmo memético	- Revisión y validación al 100% por un especialista en el diseño de algoritmos
(R4) Diseño de un algoritmo memético adaptado al problema de asignación de tablas a unidades de almacenamiento	- Documento con el pseudocódigo del algoritmo memético y sus funciones auxiliares - Documento que contiene la ejecución de las pruebas	- Revisión y validación al 100% por un especialista en el diseño de algoritmos

	de flujo de datos del diseño del algoritmo	
Objetivo (O3): Desarrollar una herramienta de software que permita la ejecución del algoritmo memético aplicado a la optimización de la asignación de tablas a unidades de almacenamiento de manera automática		
Resultado	**Medio de verificación**	**Indicador objetivamente verificable**
(R5) Codificación del algoritmo memético adaptado al problema de asignación de tablas a unidades de almacenamiento	- Código fuente del algoritmo memético y sus funciones auxiliares - Documento que contiene el conjunto de pruebas unitarias realizadas a la codificación del algoritmo memético	- 100% de las pruebas unitarias del algoritmo memético y sus funciones auxiliares ejecutadas con éxito - Revisión y validación al 100% por un especialista en el diseño de algoritmos
(R6) Interfaz gráfica que permita ejecutar el algoritmo memético aplicado al problema de asignación de tablas a unidades de almacenamiento	- Código fuente de la interfaz de ejecución del algoritmo - Documento que contiene el conjunto de pruebas unitarias realizadas a la interfaz gráfica	- 100% de las pruebas unitarias de la interfaz de ejecución del algoritmo ejecutadas con éxito - Revisión y validación al 100% por un especialista en el diseño de algoritmos
Objetivo(O4): Comparar el desempeño del algoritmo memético propuesto contra la solución más relevante encontrada en el estado del arte con respecto a la optimización de la asignación de tablas a unidades de almacenamiento con la finalidad de verificar que este brinda buenos resultados y puede ser aplicado con éxito en dicha tarea de optimización		
Resultado	**Medio de verificación**	**Indicador objetivamente verificable**
(R7) Diseño del algoritmo GRASP adaptado al problema de asignación de tablas a unidades de almacenamiento	- Documento con el pseudocódigo del algoritmo GRASP y sus funciones auxiliares - Documento que contiene la ejecución de las pruebas de flujo de datos del diseño del algoritmo GRASP	- Revisión y validación al 100% por un especialista en el diseño de algoritmos
(R8) Codificación del algoritmo GRASP	- Código fuente del algoritmo GRASP y sus funciones	- 100% de las pruebas unitarias del algoritmo GRASP y sus

encontrado en el estado del arte	auxiliares - Documento que contiene el conjunto de pruebas unitarias realizadas a la codificación del algoritmo GRASP	funciones auxiliares ejecutadas con éxito - Revisión y validación al 100% por un especialista en el diseño de algoritmos
(R9) Calibración de variables que se usarán en los algoritmos memético y GRASP	- Documento que contiene los resultados de la calibración de variables para los algoritmos memético y GRASP	- Revisión y validación al 100% por un especialista en el diseño de algoritmos
(R10) Documento de experimentación numérica de evaluación del desempeño de los algoritmos memético y GRASP	- Documento que contiene un informe sobre la experimentación numérica realizada sobre los algoritmos, incluyendo las conclusiones del experimento	- Revisión y validación al 100% por un especialista en el diseño de algoritmos, quien valida que la experimentación fue adecuada y los resultados correctos.

1.3 Métodos y procedimientos

En esta sección se describen las herramientas, métodos y procedimientos que se emplearán durante el presente proyecto de tesis, esta información se encuentra en la **Tabla 3**.

Tabla 3. *Herramientas, métodos y procedimientos*

Resultados esperados	**Herramientas, metodologías, métodos y procedimientos**
Definición de variables y restricciones a tomar en cuenta para la tarea de asignación de tablas	- Entrevistas semiestructuradas
Diseño de la función objetivo a tomar en cuenta para la tarea asignación de tablas	- No aplica
Definición de la estructura de datos que utilizará el algoritmo memético	- No aplica
Diseño de un algoritmo memético adaptado al problema de asignación de tablas a unidades de almacenamiento	- No aplica

Codificación del algoritmo memético adaptado al problema de asignación	- Java - Kanban - Extreme Programming - Pruebas unitarias
Interfaz para ejecutar el algoritmo memético adaptado al problema de asignación	- Java - Kanban - Extreme Programming - Pruebas unitarias
Diseño del algoritmo GRASP adaptado al problema de asignación de tablas a unidades de almacenamiento	- No aplica
Codificación del algoritmo GRASP encontrado en el estado del arte	- Java - Kanban - Extreme Programming - Pruebas unitarias
Calibración de variables que se usarán en los algoritmos memético y GRASP	- Java - Kanban
Documento de experimentación numérica de evaluación del desempeño de los algoritmos memético y GRASP	- Prueba de Shapiro-Wilk - Prueba F - Prueba Z - R Studio

1.3.1 Herramientas

1.3.1.1 Java

Java es un lenguaje de programación de propósito general, concurrente, basado en clases y orientado a objetos. Este lenguaje está diseñado para ser lo suficientemente simple como para que muchos programadores puedan lograr fluidez en él, además de que es fuertemente tipado lo cual ayuda a la detección de errores en tiempo de compilación (Oracle, 2020). Debido a estas características se ha considerado este lenguaje como el ideal para la construcción de los programas necesarios para llevar a cabo el presente proyecto de tesis.

1.3.1.2 R Studio

R es un lenguaje y un entorno de software libre usado para la computación estadística y gráfica. Es multiplataforma y proporciona una amplia variedad de técnicas estadísticas como pruebas estadísticas, análisis de series temporales, gráficas, entre otras (R Project, n.d.). Por otro lado, R Studio es un entorno de desarrollo integrado para R, este incluye una consola, un editor de resaltado de sintaxis y distintas herramientas para la administración del espacio

de trabajo (RStudio, 2020). Esta herramienta permite explotar el potencial de R y facilitar el desarrollo en este lenguaje. Debido a estas características, se ha considerado esta herramienta para la realización de la experimentación numérica en el presente proyecto de tesis.

1.3.2 Metodologías, métodos y procedimientos

1.3.2.1 Entrevistas semiestructuradas

Las entrevistas semiestructuradas son una técnica muy utilizada en las investigaciones sobre desarrollo. A diferencia de las entrevistas formales, que siguen un formato rígido con una serie de preguntas establecidas, las entrevistas semiestructuradas se centran en temas específicos y los abordan de tal manera que sea similar a una conversación (Troncoso & Daniele, 2004). Se eligió esta técnica debido a que, a menudo, proveen de información valiosa que muchas veces el investigador no contemplaba y existen consejos y técnicas en la literatura que permiten sacar un mejor provecho a la conversación (Troncoso & Daniele, 2004).

1.3.2.2 Kanban

Kanban es un método de control de flujo de trabajo para gestionar las operaciones de producción introducida a finales de los años 40. La aplicación de este método en el desarrollo de software surgió en el año 2004, y en la actualidad existe un creciente interés por la aplicación de este método en la ingeniería de software. Esto debido a que es un método que proporciona visibilidad al proceso de software, pues comunica claramente las prioridades y resalta los cuellos de botella. Una de las principales prácticas de este método es la visualización del flujo del trabajo (Ahmad et al., 2013), lo cual se considera muy útil pues permite tener claro el avance del proyecto en cada momento. Debido a estas características, se optó por utilizar este método para la gestión de las tareas en el presente proyecto de tesis.

1.3.2.3 Prueba de Shapiro-Wilk

La prueba de Shapiro-Wilk es una prueba estadística que permite evaluar la normalidad en

un determinado conjunto de datos, esta prueba de evaluación de la normalidad es conocido como un método numérico o formal. En la literatura existen diversos métodos numéricos tales como la prueba de Kolmogorov Smirnov, la prueba de Anderson Darling, entre otros. Sin embargo, la prueba de Shapiro Wilk se ha mostrado superior en potencia a las pruebas anteriormente mencionadas para distintos tipos de distribución y tamaños de muestra (Razali & Wah, 2011). Por esta razón se elige este método para ser utilizado en la etapa de experimentación numérica del presente proyecto de tesis.

1.3.2.4 Prueba F

La prueba F es una prueba estadística que permite probar la igualdad de varianzas obtenidas en dos muestras independientes (Fallas, 2012), esta prueba es considerada en el presente proyecto de tesis debido a que es necesario que para realizar la prueba Z se conozca la naturaleza de las varianzas en las muestras a comparar.

1.3.2.5 Prueba Z

La prueba Z es una prueba estadística que a diferencia de la prueba Shapiro-Wilk, puede ser usada para comparar las medias de dos muestras de datos, esto siempre y cuando se cumplan las condiciones de que ambos conjuntos de datos posean distribuciones normales, sus varianzas sean conocidas y tengan un tamaño de muestra significativo (Massey & Miller, 2016). El uso de esta prueba es relevante en el presente proyecto de tesis debido, principalmente, a que permite evaluar el comportamiento de los resultados que brindan los algoritmos planteados.

1.3.2.6 Pruebas unitarias

La verificación y validación consiste en un conjunto de procesos y técnicas que permiten analizar y determinar si un producto de software cumple con sus especificaciones y satisface las necesidades de los usuarios. Una técnica para la verificación y validación de software es la prueba de software, la cual consiste en la ejecución y simulación de un software con el propósito de encontrar fallas en el programa. Existen diversos tipos de pruebas de software

tales como pruebas de integración, pruebas de aceptación, pruebas unitarias, entre otros. Estas últimas, las pruebas unitarias, consisten en realizar pruebas sobre una sola unidad del programa, esta unidad puede ser, por ejemplo, una función, un procedimiento, una clase o una colección de datos, las pruebas unitarias tienen las ventajas de que pueden ser realizadas por la misma persona que desarrolla el programa y sirve para localizar errores y fallas en las áreas de más bajo nivel del software (Lindberg & Strandberg, 2006), esto es muy conveniente debido a que para el presente proyecto se requiere que cada funcionalidad incorporada funcione de manera correcta, por estas razones se ha optado por usar esta técnica en la implementación del presente proyecto de tesis.

1.3.2.7 Extreme Programming

Es una de marcos de referencia ágiles más populares en la actualidad y ha sido probada en muchas compañías exitosas alrededor del mundo. El marco Extreme Programming es exitosa en el desarrollo de software debido que mejora los proyectos de software de distintas maneras como, por ejemplo, mejorando la comunicación y el trabajo en equipo, manteniendo el diseño simple y claro y brindando retroalimentación mediante la incorporación de pruebas unitarias desde el primer día de desarrollo. Otra de las ventajas de este marco de referencia es que tiene un conjunto de reglas simples (Extreme Programming, 2013), las cuales son fáciles de seguir y son ideales para proyectos de corta duración, sin embargo, para el presente proyecto de tesis se está optando por incorporar solo las prácticas de Extreme Programming que se adapten a un proyecto de esta naturaleza, en ese sentido no se incluirán prácticas como la programación en pares, propiedad colectiva, entre otras.

Capítulo 2. Marco conceptual

2.1 Introducción

Este apartado tiene como objetivo describir algunos conceptos necesarios para comprender de mejor manera la problemática referida a la deficiente distribución de tablas de bases de datos relacionales en las unidades de almacenamiento disponibles. Por consiguiente, primero se desarrollarán conceptos relacionados a las bases de datos tales relacionales, luego conceptos acerca de la optimización de bases de datos, y finalmente, se desarrollarán conceptos acerca de las unidades de almacenamiento.

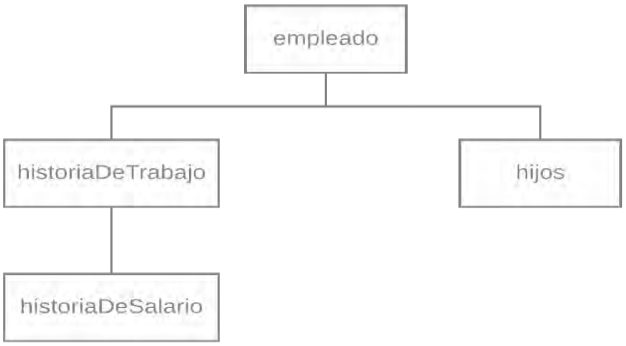
2.2 Desarrollo del marco

2.2.1 Base de datos relacional

Una base de datos relacional es una colección de datos organizada bajo una estructura de tablas relacionadas entre sí (IBM Cloud Education, 2019). Dicha estructura está basada en el modelo relacional, el cual fue concebido por Edgar Frank Codd en 1970 y cambió la manera en cómo se pensaba acerca de las bases de datos (Berg et al., 2012). Codd desarrolló una estructura que permitía a las personas de las organizaciones concentrarse en ciertos tipos de relaciones. Esas relaciones determinaron cómo pensaban las empresas acerca de sus clientes, sus facturas y sus empleados (Grier, 2012).

Un ejemplo claro de esta estructura se muestra en la **Figura 1.**, cada recuadro hace referencia a una relación o tabla, y esta a su vez se relaciona con otras tablas formando una estructura relacional.

Figura 1. *Ejemplo de una estructura relacional*



Nota. *Adaptado de "A relational model of data for large shared data banks", por E. F. Codd, 1970.*

Esto permite que se pueda acceder, por ejemplo, a la tabla "historiaDeTrabajo" sin necesidad de pasar por la tabla "empleado". Por otro lado, uno de los principios del modelo relacional asegura que los programas permanecen lógicamente intactos cuando se realizan cambios en las representaciones o los métodos de acceso a la base de datos, esto quiere decir que los cambios en cómo los datos son almacenados o recuperados no afectará cómo los usuarios acceden a estos (Sumathi & Esakkirajan, 2007). Por ejemplo, esta independencia física de los datos puede ocurrir cuando se almacena una cierta cantidad de tablas en una unidad de almacenamiento, mientras que otra cantidad en otra unidad, sin que el usuario tenga que cambiar la manera en cómo accede a una tabla en específico.

Estas son algunas de las características por las cuales el presente proyecto se enfoca en la optimización de las bases de datos relacionales.

2.2.2 Sistema de administración de base de datos relacional

Un sistema administración de base de datos es una colección de programas que son ejecutados en una computadora y ayudan al usuario a obtener, actualizar y proteger la información. De manera general, sirven como una herramienta para administrar la información (Paredaens et al., 1989).

Otra definición de los sistemas de administración de base de datos es que son sistemas de software que permiten a sus usuarios definir, crear, mantener y controlar el acceso a la base de datos (Connolly & Begg, 2016).

Para el presente proyecto de tesis, ambas definiciones acerca de los sistemas de administración de base de datos son válidas. Por otro lado, el término sistema de administración de base de datos relacional es usado, frecuentemente, para referirse a que la base de datos está basada en un modelo relacional.

Una de las principales cualidades de los sistemas de administración de base de datos es la independencia de los datos, esto quiere decir que la estructura de la base de datos no se ve afectada por cambios en aspectos físicos del almacenamiento. Debido a esto, un usuario podría cambiar las estructuras de almacenamiento de la base de datos sin afectar a otros usuarios (Sumathi & Esakkirajan, 2007). Por ejemplo, si un usuario aumenta la cantidad de unidades de disco duro y distribuye las tablas de base de datos en las nuevas unidades, esto no significa tener que cambiar la estructura de los programas, ya que el sistema de administración de base de datos soporta la independencia entre los programas y los datos.

2.2.3 Administrador de base de datos

El administrador de la base de datos es la persona que tiene el control central sobre los datos y los programas que acceden a estos. Algunas de las responsabilidades del administrador de base de datos son (Sumathi & Esakkirajan, 2007):

- ➢ Autorizar el acceso a la base de datos.
- ➢ Coordinar y monitorear el uso de la base de datos.
- ➢ Adquirir los recursos de hardware y software que sean necesarios.

Los administradores de bases de datos actualmente también son responsables de una lista de otras tareas demandantes, ellos necesitan 1) optimizar la base de datos según los requerimientos de las aplicaciones, 2) monitorear constantemente la base de datos, 3) diagnosticar la causa raíz de los problemas de rendimiento de una base de datos y 4) tomar medidas rápidas que puedan restaurar el rendimiento de una base de datos. Todas estas características han hecho que administrar la base de datos sea uno de los trabajos más estresantes en el mundo de las tecnologías de la información (Yoon et al., 2015).

Un ejemplo claro acerca de las responsabilidades de decisión que deben tomar los

administradores de base de datos, es cuando se tiene que buscar la mejor distribución de tablas de bases de datos en los recursos de almacenamiento disponibles, con el objetivo alcanzar un mejor rendimiento del sistema. La mayoría de los administradores de bases de datos simplemente confían en su experiencia pasada o algunas variables estadísticas para tomar una decisión (Yoon et al., 2015). El presente proyecto de tesis busca simplificar esta tediosa tarea.

2.2.4 Optimización de base de datos

La actividad de analizar y mejorar el rendimiento de un sistema de administración de base de datos es conocida como mejora u optimización de base de datos o “database tuning” (Almeida et al., 2019). Por otro lado, se dice que la optimización de base de datos es la actividad de hacer que una aplicación de base de datos tenga un mejor rendimiento y un menor tiempo de respuesta (Kamatkar et al., 2018). Por lo tanto, se entiende que la optimización de base de datos es el proceso a través del cual se busca mejorar el rendimiento de un sistema de base de datos.

La optimización de base de datos es un proceso complejo, el cual necesita un profundo conocimiento acerca de la estructura de la base de datos (Myalapalli et al., 2016). Este proceso normalmente involucra un cambio en la estructura de los datos, en los parámetros del sistema de base de datos, en la configuración del sistema operativo, o inclusive en el hardware (Almeida et al., 2019).

Algunos ejemplos de procesos de optimización de base de datos o “database tuning” son los siguientes:

- Optimización de índices

  Proceso que consiste en seleccionar los índices apropiados para una carga de trabajo dada, esta es una tarea crucial en la administración de base de datos (Schnaitter & Polyzotis, 2012).

- Particionamiento horizontal de base de datos

  Proceso que consiste en dividir una tabla de base de datos en partes más pequeñas,

llamadas particiones, de acuerdo con el rango de valores de un atributo dado o una combinación de atributos, llamado llave de partición (Alsultanny, 2010).

- ➢ Distribución de tablas de bases de datos en unidades de almacenamiento

  Proceso en el cual se busca mejorar el tiempo de respuesta de un sistema de base de datos mediante la reubicación de los datos frecuentemente usados en unidades de almacenamiento más rápidas (Wu et al., 2019).

El presente proyecto de investigación toma como propuesta de optimización de base de datos a la distribución de tablas de bases de datos relacionales en unidades de almacenamiento.

2.2.5 Unidades de almacenamiento

Existen varios tipos de almacenamiento de datos en la mayoría de los sistemas informáticos. Estos medios de almacenamiento pueden ser clasificados por la velocidad con la cual se puede acceder a los datos, por el costo del medio y por la confiabilidad del medio. Entre los medios típicamente disponibles se encuentran: el almacenamiento primario (caché y memoria principal), el almacenamiento secundario (discos magnéticos) y el almacenamiento terciario (discos ópticos) (Silberschatz, Korth, et al., 2011).

El presente estudio se enfoca en el almacenamiento secundario, refiriéndose a éste de manera general como “unidades de almacenamiento”, ya que dicha clasificación es usada para referirse, por ejemplo, a las unidades de disco duro o hard disk drive (HDD por sus siglas en inglés) y a las unidades estado sólido o solid state drive (SSD por sus siglas en inglés).

Se hace especial énfasis en las unidades de almacenamiento secundario debido a que, muchas veces, son la causa de los problemas de rendimiento de una base de datos relacional, un ejemplo claro de esto son los cuellos de botella de entrada/salida de las unidades de almacenamiento, esto ocurre cuando las unidades de almacenamiento no pueden soportar la carga de entrada/salida asignada a estas.

2.2.6 Entrada y Salida (Entrada/Salida)

El proceso de entrada y salida (Entrada/Salida) es un tema ampliamente estudiado en el mundo de la informática. Se dice que este proceso es la interfaz de todo sistema informático

con el mundo exterior (Stallings, 2015). Otra definición del proceso de entrada y salida asegura que es una de las principales tareas de todo computador, pues, por ejemplo, en muchas de las tareas cotidianas que realizamos (navegar por una página web o editar un archivo), nuestra principal intención es escribir (entrada) u obtener (salida) información (Silberschatz, Galvin, et al., 2011).

Por otro lado, un dispositivo de entrada y salida es entendido como aquel que permite "escribir" datos en él y "leer" datos en él (Null & Lobur, 2015). En el contexto de la presente investigación, el proceso de entrada y salida hace referencia al proceso de "escritura" o "lectura" de un dispositivo de entrada y salida, que en este caso son las unidades de almacenamiento secundario (Por ejemplo: unidades de disco duro o unidades de estado sólido).

2.2.7 Sistemas de información

Los sistemas de información son sistemas organizacionales diseñados para recolectar, procesar, almacenar y distribuir la información. En pocas palabras, el objetivo de una empresa al introducir un sistema de información es satisfacer sus necesidades de procesamiento de información. Muchas veces, los requisitos externos, como la información financiera, la seguridad o las regulaciones fiscales, exigen la introducción de un nuevo sistema de información. Más típicamente, una organización introduce un sistema de información en un esfuerzo de mejorar su eficiencia y eficacia (Piccoli & Pigni, 2016).

Los sistemas de información están a nuestro alrededor aunque no las podamos ver, por ejemplo: compañías como FedEx y UPS usan los sistemas de información para estructurar cargas y rastrear los paquetes que envían, empresas de retail como Walmart usan los sistemas de información para todo, desde optimizar sus compras hasta registrar las ventas o analizar los gustos y preferencias de sus clientes, inclusive muchas ciudades utilizan sistemas de información para el control de tráfico o de velocidad (Valacich & Schneider, 2017), y así se pueden mencionar muchísimos más ejemplos.

Con el objetivo de satisfacer sus necesidades de procesamiento de información, las

organizaciones deben capturar los datos relevantes, que después serán manipulados o procesados, para producir una “salida” que será útil para los usuarios internos (empleados) y externos (consumidores) del sistema (Piccoli & Pigni, 2016). Estos datos están típicamente acumulados o almacenarlos para su uso futuro en un sistema de administración de base de datos relacional, es por ello que mejorar y mantener el rendimiento del sistema de base de datos es considerado como el aspecto más crítico dentro de la administración de un sistema de información (Maabreh, 2018).

2.2.8 Algoritmo metaheurístico

El término metaheurística fue acuñado por Glover en 1986 para referirse a un conjunto de metodologías posicionadas por encima de las heurísticas (técnicas de resolución de problemas basadas en la experiencia). Los algoritmos metaheurísticos son procedimientos de alto nivel que pueden proporcionar soluciones válidas a problemas de optimización. Estos pueden, con frecuencia, encontrar apropiadas soluciones con menos esfuerzo computacional que los métodos basados en cálculos o simples heurísticas (Du & Swamy, 2016).

Los algoritmos metaheurísticos se pueden clasificar en:

- Basadas en una única solución

  La solución es única todo el tiempo y esta es reemplazada iterativamente por una de mejor calidad. Son consideradas como orientadas a la explotación, lo cual significa que toma ventaja de la información ya obtenida (Du & Swamy, 2016). Ejemplo: Algoritmo GRASP.

- Basadas en una población

  El número de soluciones es actualizado iterativamente hasta satisfacer una condición de terminación. Son consideradas como orientadas a la exploración, lo cual significa que busca la solución en distintas “regiones” del espacio de búsqueda (Du & Swamy, 2016). Ejemplo: Algoritmo Memético.

La presente investigación tiene como materia de estudio un algoritmo metaheurístico basado en una población.

2.2.9 Algoritmo memético

Los algoritmos meméticos (MA por sus siglas en inglés) son metaheurísticas basadas en poblaciones y compuestas por un algoritmo evolutivo y un conjunto de algoritmos de búsqueda local.

Este algoritmo está inspirado en el concepto de evolución cultural, el cual tiene como unidad de transmisión al “meme”. En otras palabras, las ideas complejas pueden ser descompuestas en “memes”, los cuales se propagan y mutan al interior de una población. De esta manera, la cultura evoluciona constantemente y tiende a experimentar mejoras progresivas. Las ideas fuertes resisten, mientras que las ideas débiles no sobreviven y desaparecen.

Los algoritmos meméticos son conocidos por brindar buenos resultados para una gama de problemas de optimización combinatoria (F. Neri et al., 2012).

Algunos ejemplos de casos de estudio con buenos resultados son:

- Algoritmos meméticos aplicados al problema del vendedor ambulante (F. Neri et al., 2012).
- Algoritmos meméticos aplicados al problema de programación cuadrática sin restricciones (F. Neri et al., 2012).

El caso de estudio propuesto en el presente proyecto está referido a la asignación de tablas a unidades de almacenamiento.

Capítulo 3. Estado del arte

3.1 Introducción

El objetivo del presente documento es identificar y dar a conocer el estado del arte de los distintos métodos de optimización de bases de datos relacionales. La metodología empleada para alcanzar dicho objetivo es la llamada “revisión sistemática”, la cual es un tipo especial de revisión de literatura que confiere ventajas adicionales a la revisión de literatura tradicional. Esta metodología permite identificar, seleccionar y evaluar críticamente la investigación relevante, y recopilar y analizar datos de los estudios que se incluyen en la revisión (Booth, 2012).

3.2 Objetivos de revisión

A continuación, se indican los objetivos por los cuales se realizó la presente revisión de literatura:

- Localizar toda la investigación relevante que se está realizando actualmente sobre la optimización de los sistemas de bases de datos relacionales.
- Entender las causas de los problemas de rendimiento en los sistemas de bases de datos relacionales.
- Conocer qué investigaciones se han realizado con respecto a la optimización de bases de datos relacionales mediante la asignación de tablas a unidades de almacenamiento.

3.3 Preguntas de revisión

A continuación, se indican las preguntas de investigación formuladas:

- **P1.** ¿Cuáles son las causas los problemas de rendimiento de los sistemas de bases de datos relacionales que están siendo investigados?
- **P2.** ¿Qué métodos se están empleando en la optimización de rendimiento de los sistemas de bases de datos?
- **P3.** ¿Qué investigaciones existen relacionadas al uso de técnicas algorítmicas,

algoritmos metaheurísticos o algoritmo meméticos aplicados a la optimización de los sistemas de bases de datos relacionales mediante la asignación de tablas a dispositivos de almacenamiento?

3.4 Estrategia de búsqueda

A continuación, se describen los factores más importantes de la estrategia de búsqueda utilizada.

3.4.1 Motores de búsqueda a usar

Las bases de datos usadas en la investigación son las siguientes:

- Scopus
- IEEE Xplore Digital Library
- ACM Digital Library

Estas fueron escogidas bajo los siguientes criterios:

- Son bases de datos que están disponibles para alumnos, docentes y tesistas pertenecientes a la comunidad de la Pontificia Universidad Católica del Perú.
- Son bases de datos que contienen una amplia variedad de documentos referidos a áreas como la computación, informática, ingeniería, entre otras.

3.4.2 Cadenas de búsqueda a usar

A continuación, se presenta la cadena final de búsqueda que se utilizará para realizar la revisión de literatura.

C1 = (("database performance" OR "database administra") OR ("data placement" OR "object placement"))*

C2 = ((disk OR database))

C3 = ((tuning OR optim) OR ("metaheuristic algorithm" OR algorithm) OR ("I/O dispatch" OR "I/O operation" OR "disk I/O"))*

C4 = ("nosql"OR "cloud"OR "iot"OR "3D" OR "social network" OR "file system" OR "facial")

Cadena final = C1 AND C2 AND C3 AND NOT C4

En la **Tabla 4** se muestran las cadenas de búsqueda en la sintaxis de las bases de datos utilizadas.

Tabla 4. *Cadenas de búsqueda según la base de datos utilizada*

Base de datos	Cadena de búsqueda
Scopus	TITLE-ABS-KEY ((("database performance" OR "database administra*") OR ("data placement" OR "object placement")) AND ((disk OR database)) AND ((tuning OR optim*) OR ("metaheuristic algorithm" OR algorithm) OR ("I/O dispatch" OR "I/O operation" OR "disk I/O")) AND NOT ("nosql"OR "cloud"OR "iot"OR "3D" OR "social network" OR "file system" OR "facial"))
IEEE Xplore Digital Library	(("All Metadata": "database performance" OR "database administra*") OR ("All Metadata": "data placement" OR "object placement")) AND (("All Metadata": disk OR database)) AND (("All Metadata": tuning OR optim*) OR ("All Metadata": "metaheuristic algorithm" OR algorithm) OR ("All Metadata": "I/O dispatch" OR "I/O operation" OR "disk I/O")) AND NOT ("All Metadata": "nosql"OR "cloud"OR "iot"OR "3D" OR "social network" OR "file system" OR "facial")
ACM Digital Library	("database performance" OR "database administra*" OR "data placement" OR "object placement") AND (disk OR database) AND (tuning OR optim* OR "metaheuristic algorithm" OR algorithm OR "I/O dispatch" OR "I/O operation" OR "disk I/O")

3.4.3 Criterios de inclusión/exclusión

Los criterios de inclusión y exclusión utilizados se describen a continuación en la **Tabla 5**.

Tabla 5. *Criterios de inclusión y exclusión utilizados*

Criterios de inclusión	Criterios de exclusión
CI1. Estudios que aborda como tema relacionado a la optimización del rendimiento de los sistemas de	CE1. Estudios escritos en idiomas distintos del inglés o español.

bases de datos relacionales o a la optimización de dispositivos de almacenamiento. CI2. Estudios que se encuentran en el ámbito de la Informática y/o Computer Science.	CE2. Estudios publicados con una antigüedad mayor a seis años.

Se eligieron los últimos seis años debido a que se desea realizar la investigación del estado del arte usando los estudios más recientes. Cabe mencionar que en caso de que no se encuentren suficientes investigaciones que permitan responder a las preguntas de investigación, se optará por incluir estudios con seis años más de antigüedad.

3.4.4 Documentos encontrados

En esta sección se muestra el proceso de selección de estudios primarios. En primer lugar, se procedió a realizar las búsquedas de documentación en las bases de datos mencionadas en el punto 4.1, mediante las cadenas de búsqueda mencionadas en el punto 4.2, obteniéndose los resultados mostrados en la **Tabla 6**.

Tabla 6. *Cantidad de documentos encontrados en cada motor de base de datos*

Base de datos	Cantidad
Scopus	707
IEEE Xplore Digital Library	317
ACM Digital Library	160
Total	1184

A continuación, se procedió a importar la información de los 1184 documentos resultantes hacia la herramienta Mendeley, usando esta misma herramienta, se pudo detectar y excluir 253 documentos duplicados, quedándose con 931 documentos.

Luego, se procedió a excluir aquellos documentos que cumplieron con los criterios de exclusión mencionados en el punto 4.3, quedándose con 234 documentos. Teniendo estos

documentos, se procedió a la revisión de los títulos y sumarios de las publicaciones y, considerando los criterios de inclusión descritos en el punto 4.3, se seleccionó un total de 15 documentos.

Una vez que se recolectaron los 15 documentos, se pudo notar que no había suficientes documentos que puedan responder a la pregunta número tres. Es por ello, que se procedió a extraer a las investigaciones con una antigüedad mayor a cinco años, pero menor a diez años, quedándose con 223 documentos. Luego de realizar un proceso idéntico al del grupo de documentos anterior, se seleccionó un total de 6 documentos adicionales.

Nuevamente, se pudo notar que los documentos seleccionados no aportaron suficiente información para poder responder a la pregunta 3, por ello, se procedió a extraer las investigaciones con una antigüedad mayor a diez años, pero menor a quince años, quedándose con 175. Finalmente, se realizó el mismo proceso que en los grupos de documentos anteriores y se seleccionó un total de 3 documentos.

Al tener suficientes estudios para poder responder a todas las preguntas planteadas, se procedió a extraer un total de 24 documentos seleccionados, el detalle de estos se muestra en el Anexo 1.

3.5 Formulario de extracción de datos

En esta sección se presenta el formulario de extracción, el cual permite identificar los aspectos más importantes de las investigaciones seleccionadas. El detalle de este formulario puede verse en el Anexo 1, el cual contiene la información que se busca extraer, un ejemplo de cómo debe llenarse y también un enlace al documento oficial completado en su totalidad.

3.6 Resultados de revisión

3.6.1 Respuesta a la pregunta 1

¿Cuáles son los problemas de rendimiento de los sistemas de bases de datos relacionales que están siendo investigados?

Se ha encontrado que los problemas de rendimiento de los sistemas de bases de datos son temas ampliamente investigados. Así, se identificaron problemas procedentes del diseño de las bases de datos (Cincy & Jeba, 2018; Colley & Stanier, 2017; Idhaim, 2019; Kamatkar et al., 2018; Maabreh, 2018; Wajszczyk & Gruszka, 2020; Yoon et al., 2015b), esto debido a que un deficiente diseño de la estructura de tablas de la base de datos conduce a un inadecuado rendimiento de la misma (Kamatkar et al., 2018). De igual manera, problemas relacionados a aspectos del hardware tales como los cuellos de botella de la CPU, la capacidad de E/S de las unidades de almacenamiento y el rendimiento de la estructura de memorias (Kamatkar et al., 2018; Myalapalli et al., 2015).

Por otro lado, se identificaron problemas derivados de la indexación en bases de datos (Colley & Stanier, 2017; Kamatkar et al., 2018; Neuhaus et al., 2019), esto sucede debido a que el servidor de base de datos realiza trabajo extra para mantener a los índices actualizados (Kamatkar et al., 2018). Finalmente, los problemas relacionados al diseño de las consultas SQL o “SQL queries” (Colley & Stanier, 2017; Idhaim, 2019; Myalapalli et al., 2015; Yoon et al., 2015b).

A manera de resumen, se presenta en la **Tabla 7** a los problemas de rendimiento en los sistemas de bases de datos relacionales más relevantes identificados en los estudios seleccionados.

Tabla 7. *Problemas de rendimiento en los sistemas de bases de datos relacionales*

Problemas	**Estudios**
Diseño de bases de datos	Kamatkar, S. J. et al., 2018 Saleh Maabreh, K., 2018 Colley, D., & Stanier, C., 2017 Wajszczyk, B., & Gruszka, I. M., 2020 Cincy, W. C., & Jeba, J. R., 2018 Idhaim, H. A., 2019 Yoon, D. Y. et al, 2015
Hardware de la base de datos	Kamatkar, S. J. et al., 2018 Myalapalli, V. K. et al., 2015
Indexación de bases de datos	Colley, D., & Stanier, C., 2017 Neuhaus, P., et al., 2019 Kamatkar, S. J. et al., 2018

Diseño de consultas SQL	Myalapalli, V. K. et al., 2015 Colley, D., & Stanier, C., 2017 Idhaim, H. A., 2019 Yoon, D. Y. et al, 2015

3.6.2 Respuesta a la pregunta 2

¿Qué métodos se están empleando en la optimización de rendimiento de los sistemas de bases de datos?

Respecto a los métodos empleados en la optimización del rendimiento de las bases de datos, se encontró que existen propuestas para la optimización del diseño de las consultas SQL, estas van dirigidas, por un lado, al uso de múltiples técnicas para el rediseño de las consultas o “queries” y la mejora del procesamiento interno de la base de datos (Kamatkar et al., 2018; Myalapalli et al., 2015) y, por otro lado, al uso de algoritmos de optimización para obtener un plan de ejecución de las consultas óptimo (Cincy & Jeba, 2018).

Asimismo, se identificaron propuestas para la optimización de la ejecución de consultas o “queries” tales como la utilización de métodos de particionamiento de tablas de base de datos (Alsultanny, 2010; Kechar & Nait-Bahloul, 2019; Maabreh, 2018; Matalqa & Mustafa, 2016; Zhong-zhuang et al., 2012), con el objetivo de poder controlar cómo los datos son distribuidos y ubicados en particiones individuales (Maabreh, 2018), el uso de métodos de procesamiento en paralelo de consultas o “queries” (Wajszczyk & Gruszka, 2020), el uso de un algoritmo genético diseñado para seleccionar automáticamente la mejor configuración de índices adaptables para cualquier esquema de base de datos (Neuhaus et al., 2019), la propuesta de una nueva técnica de recomendación de índices llamado “optimización semiautomática” (Schnaitter & Polyzotis, 2012), el uso de un algoritmo genético que permite determinar un conjunto de índices secundarios óptimo para una aplicación multi relacional de base de datos (Raja Kumar Reddy & Naidu, 2015), y por último, la propuesta de un algoritmo genético que permita seleccionar de manera eficiente el subconjunto de “agregados” utilizados en el

método de materialización de resultados parciales, el cual es quizás el método más importante para reducir el consumo de recursos de consultas SQL (Szulc et al., 2017).

Por otro lado, se identificó una propuesta para mejorar el rendimiento de entrada y salida de los sistemas de bases de datos de aplicaciones de alta prioridad mediante la distribución de objetos o tablas en unidades de almacenamiento (Wu et al., 2019).

Finalmente, se identificaron propuestas relacionadas a herramientas que contribuyen a la gestión de los sistemas de bases de datos, la primera de ellas es DBSeer, una herramienta de soporte para administradores de bases de datos (Yoon et al., 2015b), la segunda herramienta es HUTuneSQL, una herramienta usada para analizar el desempeño de comandos SQL (Idhaim, 2019).

3.6.3 Respuesta a la pregunta 3

¿Qué investigaciones existen relacionadas al uso de técnicas algorítmicas, algoritmos metaheurísticos o algoritmo meméticos aplicados a la optimización de los sistemas de bases de datos relacionales mediante la asignación de tablas a dispositivos de almacenamiento?

Respecto a las investigaciones que involucran a la asignación de objetos de bases de datos (tablas, índices, vistas materializadas, entre otros) a unidades de almacenamiento como una alternativa de optimización, se identificó el uso de heurísticas como la técnica de la mochila codiciosa y un algoritmo de programación dinámica, en la creación de un “asesor de ubicación de objetos”, el cual proporciona una estrategia para la ubicación de objetos de bases de datos en unidades de almacenamiento, ya sean discos SSD o discos duros HDD (Canim et al., 2009).

Asimismo, se identificó el uso del algoritmo Sliding-Window aplicado en un contexto de almacenamiento multimedia, esta investigación se centra en la optimización de la ubicación de “objetos” en múltiples unidades de almacenamiento (Golubchik et al., 2009) .

Por otro lado, se identificó un método de administración de datos, el cual tiene como objetivo mejorar el rendimiento de las bases de datos mediante el uso de la migración de los datos de alta prioridad hacia las unidades de almacenamiento de rápido acceso como los discos de estado sólido o SSD, y los datos de baja prioridad hacia los discos duros o HDD (Wu et al., 2019).

De igual manera, se identificó un estudio que propone un esquema de migración de datos, con una cuidadosa asignación de datos para así minimizar el volumen de datos que se trasladan de un dispositivo de almacenamiento a otro, este esquema utiliza un algoritmo que calcula la cantidad de datos que será necesario migrar hacia cada uno de los "dispositivos de destino", dicho algoritmo busca logar equilibrar el "uso" de los dispositivos (Ou et al., 2014).

Asimismo, se pudo identificar una propuesta basada en la asignación de datos en un sistema híbrido de almacenamiento "multi torre" para así alcanzar el máximo beneficio sin exceder la capacidad de almacenamiento disponible de cada "torre", esto se pretende lograr con el uso de un algoritmo de programación dinámica multi estado (Shi et al., 2013).

Además, se identificó la propuesta de un sistema híbrido de almacenamiento auto optimizado, con el objetivo de servir a la mayoría de los accesos aleatorios de entrada y salida desde el dispositivo de almacenamiento más rápido, en este caso el disco de estado sólido o SSD. Este sistema utiliza un algoritmo de "asignación de datos" basado en el problema de la mochila o "knapsack problem", con el objetivo de maximizar la utilización del disco de estado sólido en el sistema propuesto (Lin et al., 2011).

Finalmente, se pudo identificar la utilización de un algoritmo metaheurístico GRASP (Greedy Randomized Adaptive Search Procedure), en la optimización de un proceso conocido como "balance de carga", el cual consiste en asignar los mejores recursos (unidades de almacenamiento) a tablas con ciertas características tales como tablas de mayor tamaño o tablas con un mayor nivel de acceso(Cueva & Tupia, 2008).

3.7 Conclusiones

La presente revisión de literatura permitió tener una visión más clara de la investigación y los

métodos usados en la optimización de bases de datos relacionales. Asimismo, se pudo notar que los métodos de optimización más investigados en los últimos años no involucran, en su mayoría, a la asignación de objetos (tablas, índices, vistas materializadas, entre otros) a unidades de almacenamiento, sin embargo, los resultados muestran que una correcta distribución de dichos objetos en las unidades de almacenamiento disponibles mejora significativamente el rendimiento de entrada/salida de las bases de datos (Canim et al., 2009; Wu et al., 2019).

Asimismo, gracias a la revisión del estado del arte se encontró que los estudios en los cuales se plantea a la asignación de objetos a unidades de almacenamiento como un método de optimización de sistemas de bases de datos relacionales utilizan, en su mayoría, algoritmos heurísticos o métodos empíricos de priorización de datos y algoritmos iterativos para realizar esta tarea (Canim et al., 2009; Golubchik et al., 2009; Lin et al., 2011; Ou et al., 2014; Shi et al., 2013; Wu et al., 2019), mientras que existe solo un estudio en el cual se emplea un algoritmo metaheurístico en la optimización de la asignación de tablas a unidades de almacenamiento (Cueva & Tupia, 2008), por lo tanto, se espera que el desarrollo de un algoritmo memético que optimice la asignación de tablas a unidades de almacenamiento signifique una contribución al estado del arte de la optimización de bases de datos relacionales y, además, brinde una propuesta de aplicación del algoritmo memético en el “problema de asignación de datos”.

Capítulo 4. Definición de la función objetivo

4.1 Introducción

En el presente capítulo se desarrolla el primer objetivo específico: *"Determinar una relación adecuada entre las variables y factores más relevantes relacionados a la asignación de tablas a unidades de almacenamiento"*. Para evidenciar que este objetivo ha sido logrado se presentan dos resultados alcanzados, los cuales son, por un lado, la definición las variables y restricciones más relevantes a considerar para la tarea de asignación de tablas a unidades de almacenamiento, el cual consiste en describir de manera detallada a cada una de las variables y restricciones identificadas. Por otro lado, el segundo resultado alcanzado es la definición de la relación más adecuada entre las variables anteriormente identificadas a fin de ser usada en los algoritmos de optimización que se desea estudiar, el cual consiste en mostrar el planteamiento de la función objetivo y detallar las características de sus componentes principales. La descripción de cada uno de estos resultados se presenta en la sección de "Resultados alcanzados" y, adicionalmente, se tiene una última sección en la cual se discuten los resultados en relación a los problemas causa abordados por estos.

4.2 Resultados alcanzados

4.2.1 Definición de las variables y restricciones del problema

4.2.1.1 Descripción

En la presente sección se definen las variables y restricciones más relevantes a tomar en cuenta al momento de realizar un proceso de asignación de tablas.

4.2.1.1.1 Variables identificadas

- **Capacidad disponible de los dispositivos de almacenamiento**

La capacidad de almacenamiento disponible es la cantidad máxima de información que puede ser asignada a una unidad de almacenamiento. Por esta razón, esta variable puede ser usada para limitar la cantidad de tablas asignadas a un determinado dispositivo, asegurando que este no sobrepase su capacidad disponible, tal como se haría en un escenario real.

Los dispositivos de almacenamiento pueden tener distintas capacidades disponibles, pero este valor suele ser fijo para cada dispositivo de manera individual, es decir que no va a cambiar durante el proceso de asignación a menos que se reemplace el dispositivo. Esta variable se establece como $\boldsymbol{CapDisc_j}$ e indica la capacidad de almacenamiento disponible del dispositivo j.

- **Rendimiento de los dispositivos de almacenamiento**

Esta variable representa una medida del rendimiento teórico que posee una unidad de almacenamiento en términos del número neto de operaciones de entrada y salida que puede procesar en un segundo (IOPS por sus siglas en inglés). Esta medida es comúnmente brindada por el fabricante del dispositivo y, en la presente investigación, se utiliza para clasificar a las unidades de almacenamiento según su rendimiento. Esta variable se establece como $\boldsymbol{RendDisc_j}$ e indica el rendimiento del dispositivo de almacenamiento j.

- **Tamaño de las tablas de base de datos**

El tamaño de una tabla de base de datos representa qué tanta información está almacenada en una tabla y puede ser fácilmente obtenida utilizando herramientas de administración de bases de datos. Esta variable puede agregarle importancia a una determinada tabla, ya que aquellas que poseen un mayor tamaño, en ciertas ocasiones, pueden ser vistas como tablas más relevantes para un sistema. Además, esta variable también sirve para determinar la validez de una asignación, pues si un dispositivo de almacenamiento no tiene suficiente capacidad para almacenar a una tabla debido a su tamaño, la solución debe ser descartada. Esta variable se establece como $\boldsymbol{TamTab_i}$, e indica el tamaño de la tabla i.

- **Frecuencia de uso de las tablas de base de datos**

La frecuencia de uso de las tablas de base de datos describe qué tanto es utilizada una tabla en términos de operaciones de base de datos tales como guardar, leer, actualizar y borrar, la cual puede ser obtenida, de igual manera que el tamaño, usando herramientas de administración de bases de datos. Esta variable también puede sumarle importancia a una tabla pues aquellas que son más frecuentemente usadas pueden ser consideradas como más

relevantes y por ello requieren de un mejor tiempo de respuesta para sus operaciones de base de datos. De esta manera, esta variable se establece como ***FrecUsoTab$_i$***, e indica la frecuencia de uso de la tabla i.

- **Coeficientes de importancia**

Los coeficientes de importancia son valores numéricos que representan el nivel de relevancia que se le da al tamaño de las tablas (***TamTab$_i$***) y a la frecuencia de uso de las mismas (***FrecUsoTab$_i$***), al momento de decidir qué tablas se deben priorizar sobre las demás. Estas variables surgen debido a que, en contextos reales, usualmente cuando un administrador de bases de datos tiene que decidir qué tablas se debe priorizar en la asignación, resulta que existen tablas en las cuales su tamaño es más importante que su frecuencia de uso o, viceversa, su frecuencia de uso es más importante que su tamaño. Esta característica provoca que valores como la frecuencia de uso y el tamaño de una tabla no sean suficientes para determinar el dispositivo adecuado al cual se debe asignar dicha tabla, sino que se necesita un coeficiente que permita a los administradores configurar a cuál de estas dos variables se debe tomar más en cuenta al momento de decidir qué tablas son las más prioritarias. En ese sentido, estas variables se definen como ***CoefTam$_i$*** o coeficiente de importancia del tamaño de la tabla i y ***CoefFrecUso$_i$*** o coeficiente de importancia de la frecuencia de uso de la tabla i.

4.2.1.1.2 Restricciones identificadas

- **Restricción de la capacidad de almacenamiento**

Esta restricción asegura que la capacidad de cada una de las unidades de almacenamiento disponibles no sea sobrepasada al realizarse la distribución de tablas. También se encarga de asegurar la validez de una solución en un contexto real, pues si se alcanza el límite en la capacidad de las unidades de almacenamiento, será imposible asignar más información en estas.

$$\sum_{i=1 \mid S_i = j}^{n} TamTab_i \leq CapDisc_j, \forall\, j \in J$$

Donde:

- ***J*** conjunto de dispositivos de almacenamiento.
- ***S*** es una lista de n elementos en donde cada índice identifica a una tabla y cada elemento representa a la unidad de almacenamiento a la cual se asignó dicha tabla.
- S_i representa el identificador de la unidad de almacenamiento en la cual la tabla i ha sido asignada.
- **n**: número total de tablas a ser asignadas.

- **Restricción de los coeficientes de importancia**

Esta restricción asegura que los coeficientes de importancia suman un total de 1 en todo momento. Esta característica complementaria entre valores es necesaria, pues asegura que aquella variable que tenga la máxima prioridad (valor igual a 1) sea considerada en su totalidad para representar la importancia de la tabla, sin peligro de ser interferida por otra variable pues se su prioridad será mínima (valor igual a 0). De esta manera, esta restricción asegura que los coeficientes tengan un verdadero impacto en la búsqueda de una correcta configuración de tablas y unidades de almacenamiento.

$$\sum_{i=1}^{n} CoefTam_i + CoefFrecUso_i = 1, \forall\, i \in I$$

Donde:

- **n**: número total de tablas a ser asignadas.

- **Restricción de existencia**

Este conjunto de restricciones muestra formalmente los valores que pueden tomar las variables descritas anteriormente. Cabe mencionar que debido a la cantidad de variables y a los distintos rangos que estas poseen, se realiza un proceso de normalización en el cual se estandarizan los rangos de valores y se evita que las grandes diferencias entre estos puedan afectar negativamente los cálculos a realizar en la función objetivo. Esta estandarización de valores se hace usando la siguiente expresión:

$$X' = \left(\frac{\mathrm{X}}{\mathrm{X}_{\max}}\right) * (U - L) + L$$

En donde X es el valor que se quiere normalizar, X_{max} es el valor máximo del conjunto a normalizar, U es el límite superior del rango al cual se quiere normalizar y L el límite inferior de este rango. La ventaja que brinda esta técnica es que permite mantener la relación entre los datos originales, lo cual permite hacer un análisis sin distorsionar la información inicial. El rango de normalización elegido es desde 0 hasta 1 para todas las variables del problema. A continuación, en la **Tablas 8** se muestra un ejemplo del comportamiento de este método de normalización, puede notarse que la relación entre estos valores no se ve afectada.

Tabla 8. *Método de normalización empleado en valores cercanos*

Frecuencia de uso	%	Valor normalizado	%
100.0	0.3300	0.98	0.3300
101.0	0.3334	0.99	0.3334
102.0	0.3366	1.00	0.3366
303.0	**1**	**2.97**	**1**

A continuación, se describen los rangos de existencia de las variables bajo este criterio de normalización.

$$0 \leq CapDisc_j \leq 1$$

$$0 \leq RendDisc_j \leq 1$$

$$0 \leq TamTab_j \leq 1$$

$$0 \leq FrecUsoTab_j \leq 1$$

$$0 \leq CoefTam_j \leq 1$$

$$0 \leq CoefFrecUso_j \leq 1$$

Asimismo, cabe mencionar que los coeficientes de importancia tienen un rango de valores más limitado, el cual consta de tres valores:

- 1, si la variable correspondiente (***CoefTam$_i$*** o ***CoefFrecUso$_i$***) es la que se desea priorizar.
- 0, si la variable correspondiente (***CoefTam$_i$*** o ***CoefFrecUso$_i$***) es la que se desea no tomar

en cuenta.

- 0.5, si ambas variables (***CoefTam***$_i$ y ***CoefFrecUso***$_i$) tienen el mismo nivel de importancia.

4.2.1.2 Métodos, medios de verificación e indicadores

El resultado mostrado anteriormente contiene una descripción detallada de las variables y las restricciones más relevantes a tomar en cuenta para la tarea de asignación de tablas. El método utilizado para obtener esta información fue por medio de entrevistas semiestructuradas con un especialista en administración de bases de datos, con quién se establecieron reuniones para tratar temas relacionados a la tarea asignación de tablas y a los factores más importantes que pueden influir al momento de realizar una correcta distribución de tablas en contextos reales. Un diseño de la estructura general de estas entrevistas se puede visualizar en el Anexo 3, en donde se presenta de manera general las preguntas que se pudo realizar. Asimismo, en el Anexo 4 se puede visualizar las conclusiones obtenidas en cada tópico tratado durante las entrevistas, con ayuda del cual se puede verificar que se ha tomado en cuenta la opinión del especialista al momento de desarrollar el presente entregable.

Por otro lado, la forma en la que se puede verificar la existencia de este resultado alcanzado es por medio de la sección de definición de variables y restricciones mostrada en el presente capítulo. Dicha sección contiene, como ya se mencionó anteriormente, una descripción detallada de las variables y restricciones más relevantes a tomar en cuenta para la tarea de asignación de tablas y ha sido revisada y validada en su totalidad por un especialista en administración de bases de datos, con quien se ha certificado que las variables y restricciones propuestas son adecuadas para realizar una correcta distribución de tablas. El documento de validación firmado puede visualizarse en el Anexo 6.

4.2.2 Definición de una relación adecuada entre las variables y restricciones identificadas

4.2.2.1 Descripción

La tarea de asignación de tablas puede ser vista como un problema de asignación generalizado (GAP por sus siglas en inglés), este problema consiste en minimizar el costo de asignar n > 0 trabajos a m > 0 agentes, tal que cada trabajo sea asignado a exactamente un agente, esto sujeto a una capacidad disponible de cada agente (Kundakcioglu, 2009).

En el caso de la asignación de tablas a unidades de almacenamiento, los "trabajos" vienen a ser las tablas de bases de datos, las cuales tienen un determinado tamaño y frecuencia de uso, y los "agentes" son las unidades de almacenamiento, los cuales poseen una determinada capacidad para albergar tablas y una velocidad o rendimiento definido por el administrador de bases de datos.

Asimismo, debido a que el proceso de asignación de tablas se realiza con el objetivo de distribuir adecuadamente las tablas de una base de datos en los dispositivos de almacenamiento disponibles, se debe procurar que: (i) las tablas más prioritarias se depositen en los discos de mayor rendimiento y (ii) no se sobrecargue a los discos, haciendo que su rendimiento disminuya. La función objetivo planteada debe buscar que se cumplan estos dos requisitos para tener una configuración de tablas y discos óptima.

Por lo tanto, lo que se busca es maximizar el beneficio total ***"B"*** de asignar todas las tablas a los dispositivos de almacenamiento disponibles, entendiéndose como "beneficio total" al puntaje obtenido gracias a la distribución realizada. Un mayor beneficio total se traduce en un mejor rendimiento del sistema de base de datos y un mayor aprovechamiento del hardware disponible en la empresa. En ese sentido, formalmente se expresa la función objetivo de la siguiente manera:

$$Funcion\ Objetivo = maximizar(\sum_{i=1}^{n} B_{i,S_i})$$

Donde:

- ***n*** es el número total de tablas.
- $B_{i,j}$ es el beneficio esperado de asignar la tabla i al dispositivo de almacenamiento j.
- ***S*** es una lista de n elementos en donde cada índice identifica a una tabla y cada elemento representa a la unidad de almacenamiento a la cual se asignó dicha tabla.
- S_i representa el identificador de la unidad de almacenamiento en la cual la tabla i ha sido asignada.

El cálculo del beneficio esperado ***"$B_{i,j}$"*** se compone de dos partes igualmente importantes: por un lado, se encuentra el cálculo del beneficio de la asignación propiamente dicha, el cual tiene la función de procurar que las tablas más importantes se depositen en los discos de alto rendimiento, y por otro lado, se encuentra el cálculo de la penalización que se impone a la asignación realizada, el cual tiene la función de evitar problemas relacionados a la "sobrecarga" en los discos, ya que esto provoca que su rendimiento disminuya considerablemente. La combinación de estos dos componentes logra que se cumplan los requisitos (i) y (ii) que se plantearon para la función objetivo en los párrafos anteriores. En ese sentido, formalmente se expresa el cálculo del beneficio esperado de asignar una tabla i a un disco j de la siguiente manera:

$$B_{i,j} = PuntPriorTabla^2{}_i * (RendDisc_j * Penaliz_j)$$

Donde:

- ***PuntPriorTabla$_i$*** representa el puntaje total de priorización de la tabla, es decir, el puntaje que se le asigna a una tabla para determinar su nivel de importancia con respecto a las demás tablas que se desea distribuir en el sistema de almacenamiento. Este puntaje será determinado en función de las características principales de las tablas tal como se muestra a continuación:

  $$PuntPriorTabla_i = (CoefTam_i * TamTab_i) + (CoefFrecUso_i * FrecUsoTab_i)$$

  Donde:

- ***CoefTam$_i$*** es el coeficiente de importancia del tamaño de la tabla i
- ***CoefFrecUso$_i$*** es el coeficiente de importancia de la frecuencia de uso de la tabla i

Cabe mencionar que debido a que los rangos de existencia de las variables usadas para calcular el puntaje de priorización de una tabla o ***PuntPriorTabla$_i$*** se encuentran entre 0 y 1, este se encuentra, de igual manera, en dicho rango. Asimismo, en el cálculo del beneficio esperado ***"B$_{i,\ j}$"*** esta variable (***PuntPriorTabla$_i$***) tiene un crecimiento exponencial para asegurar, como ya se mencionó antes, que se coloquen las tablas más importantes en los discos de mejor rendimiento. Esto se logra debido a que cuando se transforma la función típica lineal, en una función cuadrática, se agranda las diferencias entre los valores, logrando que las tablas con alta prioridad destaquen mucho más que las tablas de baja prioridad y que se penalice de manera más drástica a las asignaciones incorrectas.

- ***RendDisc$_j$*** representa el rendimiento del disco al cual se asigna una determinada tabla.
- ***Penaliz$_j$*** representa la penalización impuesta a un determinado dispositivo de almacenamiento, esta se obtiene en función de qué tanta carga posee un disco con respecto al total de carga que se quiere distribuir en el sistema de almacenamiento, es decir que, a mayor carga tenga un disco con respecto al total, mayor será la penalización que reciba. El cálculo de la penalización se propone de esta manera debido a que la carga asignada a un disco es un factor que afecta directamente su rendimiento, es decir que, un disco que posee una gran cantidad de carga tiene que gestionar una mayor cantidad de operaciones de entrada y salida, haciendo que sea más probable la aparición de contenciones o bloqueos que afecten su rendimiento. En ese sentido, la penalización que se impone a una asignación realizada en un disco j se calcula de la siguiente manera:

$$Penaliz_j = 1 - \left(\frac{CargaDisco_j}{CargaTotal}\right)$$

Donde:

- ***CargaDisco**$_j$* representa la carga actual que "gestiona" el disco en términos de frecuencia de uso, este se calcula como la suma de las frecuencias de uso de las tablas asignadas al disco.
- ***CargaTotal*** representa la carga total que se deberá distribuir entre los distintos discos del sistema de almacenamiento, este se calcula como la suma de las frecuencias de uso de todas las tablas que se desea asignar.

Adicionalmente a la definición de la función objetivo se realizó un conjunto de pruebas del cálculo de la misma, las cuales pueden visualizarse de manera detallada en el Anexo 5. De dichas pruebas se puede concluir lo siguiente:

i. La función objetivo planteada logra cumplir los dos requerimientos que se busca: primero, las tablas más prioritarias son depositadas en los discos de mayor rendimiento y, segundo, no se sobrecargan a los discos, haciendo que su rendimiento disminuya.
ii. La función objetivo planteada mantiene un correcto comportamiento en distintos escenarios, ya que logra calificar de manera adecuada a diversas combinaciones de elementos, desde pruebas con solo 2 tablas y 2 discos hasta el uso de 10 tablas con 4 discos.
iii. La función objetivo mantiene un correcto comportamiento cuando se usan distintas combinaciones de valores para los coeficientes de importancia tanto del tamaño de las tablas como de la frecuencia de uso de las mismas.

Por estos motivos, la función objetivo se considera adecuada para ser usada en el proceso de optimización de la asignación de tablas a unidades de almacenamiento.

4.2.2.2 Métodos, medios de verificación e indicadores

La forma de verificar la existencia de este resultado alcanzado es por medio de la sección de definición de la función objetivo mostrada en el presente capítulo. Dicha sección contiene una descripción detallada del planteamiento de la función objetivo y las variables incluidas en este. Además, esta sección ha sido validada en su totalidad por un especialista en administración

de bases de datos, con quien se ha certificado que la función objetivo propuesta es adecuada para realizar una correcta asignación de tablas. El documento de validación firmado puede visualizarse en el Anexo 6.

4.3 Discusión de los resultados

En el presente capítulo se describe a los dos primeros resultados alcanzados del proyecto de tesis. En primer lugar, se tiene a la definición de las variables y restricciones más relevantes a tomar en cuenta para realizar una adecuada asignación de tablas a unidades de almacenamiento. Para obtener este resultado se realizaron entrevistas con un especialista en administración de bases de datos, de las cuales se pudo obtener resultados que ayudaron a profundizar en el concepto de un proceso de asignación de tablas en contextos reales. De estas entrevistas se pudo concluir lo siguiente:

i. La dificultad para aplicar técnicas de optimización adecuadas es uno de los factores que más afecta negativamente a la optimización del rendimiento de un sistema de bases de datos.
ii. Debido a la complejidad que significa aplicar un método de optimización adecuado en un conjunto de tablas y discos alto, no se suelen utilizar todos los factores más importantes (frecuencia de uso, tamaño de una tabla, rendimiento de los discos y capacidad de los discos) para realizar una asignación, sino que se emplean solo algunas de ellas o se utiliza la experiencia pasada.

En ese sentido, este resultado muestra claramente en un documento a los factores y restricciones más relevantes a tomar en cuenta en un proceso de asignación. Además, en el resultado se emplea una técnica de normalización de datos que permite utilizar todos estos factores en conjunto sin verse afectados entre sí por los distintos rangos de valores que suelen tener.

En segundo lugar, se tiene al diseño de la función objetivo a ser usada en la optimización de la tarea asignación de tablas a unidades de almacenamiento, el cual se desarrolla tomando en cuenta las variables y restricciones identificadas en el primer resultado. Esta función

objetivo representa la relación que se establece entre las variables y restricciones antes mencionadas y permite que estas puedan ser usadas de manera conjunta para obtener una asignación con el máximo beneficio posible. Adicionalmente a su definición, se presenta una serie de pruebas que fueron elaboradas con el fin de verificar la exactitud de esta función y que, debido a sus resultados positivos, tanto la definición de las variables como el diseño de la función objetivo fueron validadas en su totalidad por un especialista en administración de bases de datos (véase Anexo 5).

Estos resultados alcanzados son la evidencia de que se logró encontrar una manera de relacionar a las variables y factoras más relevantes en la tarea de asignación de tablas a unidades de almacenamiento, haciendo que estos puedan ser usados de manera conjunta y no individualmente como suelen hacer muchos administradores de bases de datos. Asimismo, son la base para estudiar el comportamiento de los algoritmos propuestos en el presente proyecto de tesis, ya que con ellos se puede identificar cuándo se está realizando una asignación adecuada y cuándo no.

Los resultados mostrados son generalizables en el sentido de que puede usarse distintas unidades de medición para variables como el rendimiento de las unidades de almacenamiento, frecuencia de uso de las tablas, capacidad de los discos y tamaño de las tablas. Además, estos resultados pueden servir como base para el desarrollo de un algoritmo de asignación en el contexto de servidores de archivos o servidores de correos.

Una limitación de los resultados mostrados es que no se ha tomado en consideración la situación actual del sistema de almacenamiento, es decir, aspectos como la antigüedad de los dispositivos o el tiempo de vida estimado. Sin embargo, se está procurando obtener la mejor asignación posible con los datos que el administrador disponga, los cuales se espera que sean lo suficientemente acertados y confiables para realizar una óptima distribución de tablas.

Capítulo 5. Adaptación del Algoritmo Memético

5.1 Introducción

En el presente capítulo se desarrolla el segundo objetivo específico: *"Adaptar el algoritmo memético de tal manera que pueda ser aplicado en la optimización de la asignación de tablas a unidades de almacenamiento"*. Para evidenciar que este objetivo ha sido logrado se presentan dos resultados alcanzados, los cuales son, por un lado, la definición de las estructuras de datos que utiliza el algoritmo memético, la cual consiste en realizar una descripción detallada de los componentes más importantes usados para representar datos reales en el algoritmo que se desea estudiar. Por otro lado, el segundo resultado alcanzado es el diseño del algoritmo memético adaptado al problema de asignación de tablas a unidades de almacenamiento, el cual consiste en mostrar el planteamiento del algoritmo memético y detallar las características principales de sus funciones auxiliares, mostrando que el algoritmo puede ser aplicado en la optimización de la asignación de tablas a unidades de almacenamiento. Cada uno de estos resultados se presenta de manera detallada en la sección de "Descripción" y, adicionalmente, se tiene una última sección en la cual se discuten los resultados en relación a los problemas causa abordados por estos.

5.2 Resultados alcanzados

5.2.1 Definición de las estructuras de datos

5.2.1.1 Descripción

En la presente sección se describen las estructuras de datos que sirven para agrupar y organizar mejor la información que será utilizada en el algoritmo memético. Asimismo, se presenta la estructura del cromosoma, uno de los elementos más importantes para la construcción de dicho algoritmo.

5.2.1.1.1 Representación de una tabla de base de datos

La estructura "Tabla" se usa para representar las tablas de base de datos en el diseño del algoritmo memético. Esta estructura almacena el tamaño y la frecuencia de uso de una tabla,

además de los coeficientes de importancia de estas dos variables. Para la codificación de esta estructura se planea utilizar una clase llamada “Tabla” con los cuatro atributos descritos anteriormente. Una representación visual de esta estructura se presenta en la **Figura 2** a manera de ejemplo.

Figura 2. *Estructura de datos Tabla*

Tamaño (KB)	Coeficiente de importancia del Tamaño	Frecuencia de Uso (accesos a la tabla)	Coeficiente de importancia de la Frecuencia de uso
54.34	0	185.00	1

Fuente: Elaboración propia

5.2.1.1.2 Representación de una unidad de almacenamiento

La estructura “Disco” representa a las unidades de almacenamiento en el diseño del algoritmo memético. Esta estructura almacena el puntaje de rendimiento y la capacidad de una unidad de almacenamiento, datos necesarios para realizar una correcta asignación de tablas. Para la codificación de esta estructura se planea utilizar una clase llamada “Disco” con los dos atributos descritos anteriormente. Una representación visual de esta estructura se presenta en la **Figura 3** a manera de ejemplo.

Figura 3. *Estructura de datos Disco*

Rendimiento (IOPS)	Capacidad (GB)
164.30	2048

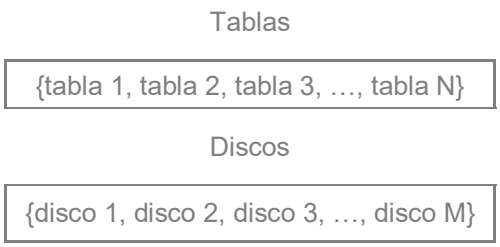
Fuente: Elaboración propia

5.2.1.1.3 Representación del problema

La estructura “Problema” representa al conjunto de tablas y discos a ser utilizada por el algoritmo para lograr una asignación óptima. Esta estructura contiene una lista de las tablas de base de datos a asignar y una lista de unidades de almacenamiento disponibles. Al igual

que en las estructuras anteriores, para su codificación se planea emplear una clase llamada “Problema”. Una representación de esta estructura se presenta en la **Figura 4**.

Figura 4. *Estructura de datos Problema*

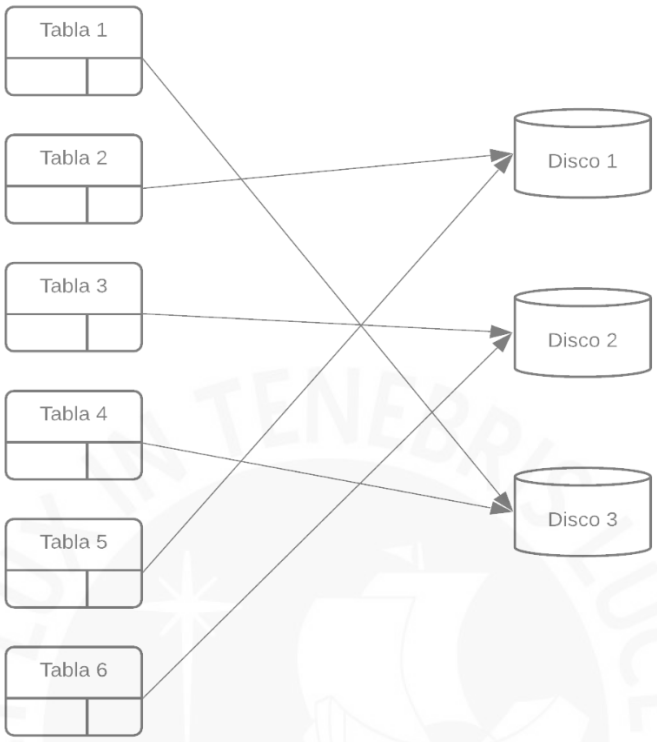


Fuente: Elaboración propia

5.2.1.1.4 Representación del cromosoma

La solución al problema de asignación consiste en una serie de tablas asignadas a un conjunto de unidades de almacenamiento. Esta solución se representa en el diseño del algoritmo memético mediante una estructura denominada “Cromosoma”. En este algoritmo se realiza una serie de “operaciones” sobre el “Cromosoma”, tales como la selección, cruzamiento, mutación, entre otras, con el objetivo de que este pueda “evolucionar” o mejorar constantemente. Un ejemplo de esta estructura se muestra en la **Figura 5**, en la cual, dado un sistema de base de datos compuesto por 6 tablas y 3 discos, se tiene una posible asignación representada por flechas unidireccionales.

Figura 5. *Posible asignación de tablas a unidades de almacenamiento*



Fuente: Elaboración propia

En esta solución podemos notar que las tablas 1 y 4 fueron asignadas al disco 3, las tablas 2 y 5 al disco 1 y las tablas 3 y 6 al disco 2. Para representar esta solución como un cromosoma se opta por utilizar una lista en la cual cada índice “i” corresponde a una tabla de base datos numerada desde 0 hasta n-1, siendo n el número de tablas a analizar, por ejemplo, cuando “i” es igual a 0 hace referencia a la primera tabla, cuando “i” es igual a 1 a la segunda y así sucesivamente. Por otro lado, cada valor en la lista es un gen del cromosoma y representa a la unidad de almacenamiento “j” a la cual la tabla “i” fue asignada, donde j tiene valores desde 0 hasta m-1, siendo m el número de unidades de almacenamiento, por ejemplo, si en la posición i=2 se tiene el valor j=1, significa que la tercera tabla de la base de datos fue asignada al segundo disco del sistema de almacenamiento.

Esta representación es similar a la usada en los algoritmos genéticos continuos, en los cuales, si se tiene N variables, el cromosoma es representado como una lista de N elementos en la que cada gen corresponde al valor de una de las variables del problema (Haupt & Haupt, 2004). Asimismo, esta asegura que cada tabla sea asignada exactamente a una unidad de almacenamiento y facilita las operaciones propias del algoritmo memético tales como el cruzamiento y la mutación, esto debido a que se tiene una única dimensión sobre la cual realizar los cálculos. Además, esta forma de representar el cromosoma requiere menos espacio que la representación tradicional propuesta en el problema de asignación generalizada, ya que es una lista de tamaño n y no una matriz de n x m. La representación de la solución mostrada en la **Figura 5** puede verse a continuación en la **Figura 6**.

Figura 6. *Representación del cromosoma en una estructura lineal*

Tabla i:	0	1	2	3	4	5
Disco j:	2	0	1	2	0	1

Fuente: Elaboración propia

La estructura de datos llamada Cromosoma contiene la lista de genes descrita anteriormente y un valor de fitness (el cual representa el puntaje de aptitud que se asigna al cromosoma y permite saber qué tan buena es la solución propuesta). Para la codificación de esta estructura se planea utilizar una clase llamada “Cromosoma” con los dos atributos descritos anteriormente. Un ejemplo de esta representación se muestra en la **Figura 7**.

Figura 7. *Estructura de datos Cromosoma*

Genes	Fitness
{2, 0, 1, 2, 0, 1}	24.55

Fuente: Elaboración propia

5.2.1.1.5 Representación de una población

Finalmente, se tiene a la representación de una población, un componente esencial de los algoritmos evolutivos tales como el algoritmo memético. En esta estructura se intenta representar a los individuos de una población o también llamados “agentes”, los cuales “evolucionan” o mejoran durante la ejecución del algoritmo. Este componente contiene el tamaño de la población y la lista de “agentes” o “individuos” pertenecientes a la misma. Una representación visual de esta estructura se presenta en la **Figura 8** a manera de ejemplo.

Figura 8. *Estructura de datos Población*

Agentes	Tamaño
{Cromosoma 1, Cromosoma 2, Cromosoma 3, …, Cromosoma N}	N

Fuente: Elaboración propia

5.2.1.2 Métodos, medios de verificación e indicadores

La forma de verificar la existencia de este resultado alcanzado es por medio de la sección de definición de las estructuras de datos mostrada en el presente capítulo. Dicha sección contiene una descripción detallada de las principales estructuras de datos propuestas para los algoritmos que se desea estudiar. Asimismo, esta sección ha sido revisada y validada en su totalidad por un especialista en el diseño algoritmos, con quien se ha comprobado que las estructuras propuestas son adecuadas para almacenar la información para su posterior uso. El documento de validación firmado puede visualizarse en el Anexo 8.

5.2.2 Diseño del algoritmo memético

5.2.2.1 Descripción

En la presente sección se utilizan las estructuras de datos planteadas anteriormente para diseñar el algoritmo memético de tal manera que pueda ser utilizado en la optimización de la asignación de tablas a unidades de almacenamiento.

El algoritmo memético se compone de diversas funciones auxiliares, cada una de las cuales

cumple con una responsabilidad definida. El diseño detallado de cada una de estas se encuentra en el Anexo 9.

5.2.2.1.1 Algoritmo Memético

A continuación, se presenta el diseño de la función más importante: el Algoritmo Memético. Este algoritmo pertenece a una familia de metaheurísticas que intentan integrar diversas ideas, conceptos y técnicas de resolución de problemas (Cotta, 2016), en ese sentido, el presente proyecto de tesis desarrolla la integración entre dos componentes: el primero es llamado “evolutivo”, y se caracteriza por mantener una población de agentes con un tamaño constante, la cual “evoluciona” o mejora al aplicarle operaciones como el casamiento, mutación y selección descritas anteriormente en este capítulo. El otro componente es el de “búsqueda local” y es la característica más distintiva del algoritmo memético, ya que es un método que permite explotar aún más las soluciones halladas en la fase evolutiva (C. Neri & Cotta, 2012). Este método consiste en mantener una solución “actual” y explorar sus “vecinos” en búsqueda de soluciones de mejor calidad. Estos vecinos son candidatos levemente distintos a la solución “actual”, es por ello que se le llama búsqueda local. Eventualmente alguno de ellos llega a ser mejor que la solución “actual” y se convierte en la nueva solución “actual”, con el cual se repite el proceso hasta cumplir algún criterio de terminación (Onwubolu & Babu, 2004). En el presente proyecto de tesis este componente es empleado de dos formas distintas: primero se aplica una búsqueda local sobre la población inicial y luego se aplica una búsqueda local sobre una cantidad “N” de agentes seleccionados por el método de la Ruleta (tal como se hace en la función de selección de agentes). Este proceso se realiza cada “M” generaciones, ya que permite tener un mejor control sobre cuándo y sobre qué agentes aplicar el proceso de búsqueda local, lo cual se considera útil para poder encontrar un balance entre el comportamiento del algoritmo evolutivo y el algoritmo de búsqueda local. El detalle del diseño de esta función se puede visualizar en la **Figura 9**.

Figura 9. *Diseño del algoritmo memético*

```
Inicio ejecutarAlgoritmoMemetico (P, param)
1:  tPob, tCasam, tMuta, pMuta, mIter,
    nGenNoMejora, cadaNGen, sobreNAg <= param
2:  pob = generarPoblacionInicial (tPob, P)
3:  evaluar (pob, P)
4:  aplicarBusquedaLocal (pob, P)
5:  mejorAgente = obtenerMejor(pob)
6:  cIter, cGenNoMejora, cCadaNGen = 1
7:  Mientras cIter <= mIter Y cGenNoMejora <= nGenNoMejora Hacer:
8:     descPob = aplicarCasamiento (pob, tCasam)
9:     mutarPoblacion (descPob, P, tMuta, pMuta)
10:    evaluar (descPob, P)
11:    Si cCadaNGen = cadaNGen:
12:       s = seleccionarAgentes (descPob, sobreNAg)
13:       Para cada agente en s:
14:          agenteMejorado = aplicarBusquedaLocal (P, agente, param)
15:          reemplazarEn (descPob, agenteMejorado)
16:       cCadaNGen = 0
17:    pob = aplicarSeleccion (pob, descPob)
18:    nuevoMejorAgente = obtenerMejor(pob)
19:    Si nuevoMejorAgente > mejorAgente:
20:       mejorAgente = nuevoMejorAgente
21:       cGenNoMejora = 0
22:    Si no:
23:       cGenNoMejora += 1
24:    cIter += 1
25:    cCadaNGen += 1
26: Retornar mejorAgente
Fin ejecutarAlgoritmoMemetico
```

Fuente: Elaboración propia

Donde:

- Los parámetros son los siguientes
  - P: estructura **Problema**, la cual contiene toda la información relevante acerca de las tablas y los discos
  - Param: una estructura que contiene todos los parámetros que el algoritmo necesitará para su ejecución
- Línea 1: se obtienen los valores de los parámetros necesarios
- Línea 2: se genera una población inicial diversa usando el tamaño de la población y el Problema

- Línea 3: se evalúa la aptitud de todos los agentes generados
- Línea 4: se aplica una búsqueda local sobre toda la población para mejorar cada elemento de la misma
- Línea 5: se obtiene el mejor agente de la población actual
- Línea 6: se inicializan los contadores necesarios para detener el algoritmo cuando sea necesario
- Línea 7: se inicia el bucle principal, el cual solo se detiene dependiendo de las condiciones de parada descritas anteriormente
    - Línea 8: se aplica el proceso de casamiento y se obtiene una nueva población llamada descendencia
    - Línea 9: se muta la oblación usando la función de mutación explicada anteriormente
    - Línea 10: se obtiene la aptitud de cada elemento de esta nueva población
    - Línea 11: se valida que el contador de generaciones “cCadaNGen” sea igual al parámetro “cadaNGen”, esto para poder aplicar la búsqueda local
        - Línea 12-15: se seleccionan por el método de la ruleta a una cantidad de agentes igual al parámetro “sobreNAg”, luego se aplica el algoritmo de búsqueda local sobre cada uno de ellos, y finalmente estos son devueltos a la población con sus nuevos valores.
            - Línea 16: se actualiza el contador “cCadaNGen” a cero para aplicar este mismo procedimiento cada N generaciones
    - Línea 17: se aplica el proceso de selección de agentes tal y como se describe anteriormente, usando Elitismo
    - Línea 18: se obtiene el mejor agente de esta nueva población
    - Línea 19 - 21: en caso de que el nuevo mejor agente tenga un mejor puntaje de aptitud que el mejor agente de la población anterior, se procede se procede a colocar como mejor agente al nuevo. Además, se reinicia el contador que indica el número de generaciones sin mejora

- Línea 22 - 23: en caso de que el nuevo mejor agente no supere al anterior mejor, se procede a aumentar el número de generaciones sin mejora
- Línea 24 – 25: finalmente se actualizan los contadores, tanto el de número de iteraciones como el de aplicación de búsqueda local cada N generaciones

- Línea 26: una vez terminado el bucle, se retorna el mejor agente que pudo obtenerse

Cabe mencionar que el diseño, la implementación y el criterio de elección del algoritmo de búsqueda local se encuentra detallado en una sección titulada "implementación del algoritmo memético", perteneciente al Capítulo 6 del presente proyecto de tesis, en la cual se realiza un conjunto de pruebas necesarias para asegurar la efectividad del algoritmo memético en conjunto con el algoritmo de búsqueda local elegido. Por ende, en esta sección se presenta el diseño del esquema general del algoritmo de búsqueda local, el cual puede visualizarse en la **Figura 10**.

Figura 10. *Esquema general del algoritmo de búsqueda local*

```
Inicio aplicarBusquedaLocal ()
1:  sActual = generarSolucionInicial ()
2:  Repetir
3:     sNueva = generarSolucionNueva (sActual)
4:     Si fObjetivo(sNueva) > fObjetivo(sActual):
5:        sActual = sNueva
6:  Hasta cumplirCriterioTerminacion ()
7:  Retornar sActual
Fin aplicarBusquedaLocal
```

Fuente: Adaptado de: "Handbook of Memetic Algorithms" por Neri, C., & Cotta, P. M. (2012)

Donde:

- Línea 1: se genera una solución inicial de manera aleatoria (para el caso del algoritmo memético se va a recibir esta solución como un parámetro)
- Línea 2: se inicia un bucle que termina solo cuando se cumpla una determinada condición, la cual puede ser cuando se alcance un número máximo de iteraciones,

cuando la solución deje de mejorar, cuando ya se haya explorado todo el conjunto de soluciones posibles, o inclusive puede usarse una combinación entre varios criterios.

- Línea 3: se genera una solución nueva en base a la original, la cual puede ser generada con diversos métodos como el intercambio de elementos, la mutación por “scramble”, el algoritmo k-opt, entre otros.
    - Línea 4 y 5: en caso de que la nueva solución generada sea mejor que la actual, esta es reemplazada como la nueva mejor solución
- Línea 6: se evalúa si se cumple el criterio de terminación
- Línea 7: se retorna la mejor solución encontrada

Finalmente, en el Anexo 7 se presenta el detalle de las pruebas de flujo de datos realizadas, las cuales consisten en llevar a cabo un examen cuidadoso de los detalles lógicos del algoritmo diseñado y tienen el objetivo de detectar posibles errores en el diseño del algoritmo a través de una simulación de su ejecución con datos conocidos. De estas pruebas se puede concluir lo siguiente:

i. El diseño planteado es correcto y no presenta errores que eviten su correcta implementación.
ii. Diseñar pruebas de flujo de datos resultó muy útil para entender más a fondo la naturaleza del algoritmo y poder corregir deficiencias en algunos puntos (los cuales se describen en el documento de pruebas).
iii. La función objetivo diseñada en el Capítulo 4 cumple un papel fundamental para el diseño del algoritmo memético, y ya que esta fue debidamente probada y validada, se usa como la función que describe la aptitud de las soluciones que brinda el algoritmo.

5.2.2.2 Métodos, medios de verificación e indicadores

El resultado alcanzado se encuentra plasmado en la sección de diseño del algoritmo memético mostrado en el presente capítulo. Dicha sección contiene una descripción detallada del diseño del algoritmo memético y sus principales funciones auxiliares, los cuales son el punto de partida para poder implementar la solución que se desea estudiar en el presente proyecto de tesis.

Además, este resultado ha sido revisado y validado en su totalidad por un especialista en el diseño algoritmos, el cual ha determinado que tanto el diseño como las pruebas presentadas son suficientes y el criterio empleado en la elección de los diversos métodos utilizados por el algoritmo memético es adecuado para un proyecto de fin de carrera. El documento de validación firmado puede visualizarse en el Anexo 8.

5.3 Discusión de resultados

En este capítulo se detallan los resultados alcanzados que evidencian el logro del segundo objetivo del presente proyecto de tesis: *Adaptar el algoritmo memético de tal manera que pueda ser aplicado en la optimización de la asignación de tablas a unidades de almacenamiento.* En primer lugar, se tiene a la definición de las estructuras de datos necesarias para el diseño del algoritmo memético, estas estructuras permiten organizar y agrupar a los diversos datos que se tiene, de tal manera que se facilite su utilización al momento de diseñar e implementar el algoritmo a estudiar. Para la realización de este resultado se toma como referencia a la sección de definición de variables y restricciones, en la cual se describe a las variables y restricciones más relevantes al momento de realizar un proceso de asignación exitoso. De dicha sección se logra obtener mucha de la información que posteriormente fue agrupada en las estructuras mostradas. Asimismo, se toma en cuenta la naturaleza propia de un algoritmo memético, en la cual se plantea una estructura “agente” que, a su vez, contiene la información del “cromosoma”, un elemento que resulta indispensable para el algoritmo y que contiene la solución que se trata de optimizar

iterativamente usando una serie de operadores como la selección, cruzamiento, mutación y búsqueda local.

Asimismo, para poder realizar la optimización de un cromosoma es necesario tener una manera de conocer la “calidad” o “aptitud” de la solución que este alberga, es por ello que se emplea la función objetivo planteada en el Capítulo 4, con la cual se vuelve inmediato el cálculo de la aptitud que posee cada cromosoma y se asegura que esta representa el objetivo que se quiere lograr en el presente proyecto de tesis, el cual es distribuir de manera eficiente las tablas de una base de datos, en los dispositivos de almacenamiento disponibles.

En segundo lugar, se tiene al diseño del algoritmo memético y sus principales funciones auxiliares, en el cual se presenta a las funciones a implementar de una forma sencilla y detallada, lo cual ayuda a comprender las características y responsabilidades que cumple cada una de estas, además de que ayuda a reducir la aparición de posibles errores en la fase de implementación. Adicionalmente a su definición, se presenta un conjunto de pruebas de flujo de datos que fueron elaboradas con el fin de examinar y simular el flujo de ejecución del algoritmo y así detectar otros posibles errores o deficiencias antes de realizar la implementación del mismo.

Estos resultados alcanzados en conjunto con las pruebas realizadas evidencian que fue posible adaptar y diseñar un algoritmo especializado de optimización, es decir el algoritmo memético, para que pueda ser empleado como una técnica de optimización en la asignación de tablas a unidades de almacenamiento. Abarcando, de esta manera, la ausencia en el empleo de técnicas de optimización adecuadas tanto por parte de los administradores de bases de datos como por las propias herramientas modernas de administración y optimización de bases de datos.

Asimismo, tal como se menciona en el capítulo 3, en la revisión de literatura no se encontraron investigaciones que hayan aplicado directamente al algoritmo memético en la optimización de la asignación de tablas a unidades de almacenamiento, sino que se emplean métodos basados en cálculos simples o heurísticas, por lo cual se espera que la adaptación de este

algoritmo centrado en la tarea de asignación de tablas signifique una contribución al estado del arte de la optimización de sistemas de base de datos relacionales.

Capítulo 6. Software de ejecución del algoritmo memético

6.1 Introducción

En el presente capítulo se desarrolla el tercer objetivo específico: *"Desarrollar una herramienta de software que permita la ejecución del algoritmo memético aplicado a la optimización de la asignación de tablas a unidades de almacenamiento de manera automática".* Para evidenciar que este objetivo ha sido logrado se presentan dos resultados alcanzados, los cuales son, por un lado, la codificación del algoritmo memético, que consiste en realizar una descripción de los aspectos principales de esta implementación y cómo se relaciona con el diseño del algoritmo presentado en el capítulo 5. Por otro lado, el segundo resultado alcanzado es la codificación de una interfaz que permita ejecutar el algoritmo memético, y consiste en describir y mostrar de manera detallada el diseño y la utilización de dicha interfaz de ejecución y cómo se relaciona con el resultado anterior. Adicionalmente, se tiene una sección en la cual se discuten los problemas causa abordados al alcanzar estos resultados.

6.2 Resultados alcanzados

6.2.1 Codificación del algoritmo memético

6.2.1.1 Descripción

Como se presentó en el Capítulo 5, el diseño del algoritmo memético consta de un amplio conjunto de estructuras, funciones y métodos auxiliares. En este capítulo se procede a implementar estos componentes siguiendo fielmente la lógica planteada en el diseño de los mismos. Además, se presenta el proceso de diseño, implementación, pruebas y elección final de uno de los algoritmos de búsqueda local que se toman en cuenta para formar parte del algoritmo memético planteado en el presente proyecto de tesis.

La elección de un algoritmo de búsqueda local adecuado para el problema de asignación de tablas a unidades de almacenamiento se realiza, como se mencionó anteriormente, mediante un proceso intensivo de pruebas sobre tres algoritmos seleccionados tomando en cuenta a

la investigación de P. Thainiam titulada en español como “Los efectos de los memes en los algoritmos meméticos para resolver problemas de asignación cuadrática” y al libro de Ferrante Neri y otros titulada en español como “Manual de algoritmos meméticos”, en las cuales se plantean diversos algoritmos “sofisticados” y “populares” de búsqueda local.

Las razones de aplicar un proceso intensivo de pruebas antes de tomar una decisión son: (i) para tener la certeza de que se elige el algoritmo de búsqueda local de mejor rendimiento para trabajar en conjunto con el algoritmo evolutivo y así tener una implementación lo suficientemente sustentada del algoritmo memético, (ii) debido a que no se tiene un punto de referencia para asegurar que, para el caso de la asignación de tablas a unidades de almacenamiento, algún método de búsqueda local es mejor que otro, y (iii) porque es sabido que una de las partes más importantes de los algoritmos meméticos es la búsqueda local, ya que es la característica que los diferencia de los típicos algoritmos evolutivos (Thainiam, 2019), por ende, se le presta una especial atención tanto a su diseño como a su implementación y pruebas.

Los algoritmos evaluados son: (i) Algoritmo de Enfriamiento Simulado, un método probabilístico que se inspira en la termodinámica y tiene la ventaja de que puede evitar los óptimos locales (Thainiam, 2019). (ii) Algoritmo de Búsqueda Local Iterada, un método que construye una secuencia de soluciones aplicando, de manera iterativa, un algoritmo de búsqueda local y una estrategia de perturbación (Thainiam, 2019). (iii) Algoritmo Búsqueda Tabú, un algoritmo que utiliza memoria para almacenar los “movimientos” o acciones realizadas anteriormente, lo cual permite guiar las búsquedas hacia entornos más prometedores. Asimismo, en el Anexo 10 se puede encontrar el diseño detallado de cada uno de estos algoritmos, junto a una especificación de sus componentes principales.

A continuación, en la **Tabla 9** se muestran los resultados obtenidos en las pruebas realizadas a cada uno de los algoritmos de búsqueda local evaluados, asimismo, para su correcto entendimiento debe considerarse lo siguiente:

- Los datos usados para las pruebas fueron generados de manera aleatoria usando un algoritmo elaborado por el tesista, estos datos se agrupan en “Instancias”. Para visualizar el detalle del proceso de generación de datos aleatorios puede dirigirse al Anexo 11.
- Cada “Instancia” hace referencia al conjunto de datos generado aleatoriamente sobre el cual se realizan las pruebas. En la tabla, el nombre de cada instancia posee dos números separados por guion (p. ej. 50-10), el primero de ellos hace referencia a la cantidad de tablas simuladas y el segundo a la cantidad de discos simulados para el caso de prueba.
- Cada columna de resultados en la tabla tiene títulos que hacen referencia al algoritmo que se está probando:
    - ABT: Algoritmo de Búsqueda Tabú
    - ABLI: Algoritmo de Búsqueda Local Iterada
    - AES: Algoritmo de Enfriamiento Simulado
- Cada algoritmo se ejecuta **diez veces** con cada una de las instancias que se está probando, por esta razón se muestran dos columnas de resultados, las cuales son:
    - Mejor: es el mejor resultado obtenido en las diez repeticiones
    - Promedio: es el promedio de todos los resultados obtenidos en cada repetición

Tabla 9. *Resultados de pruebas de algoritmos de búsqueda local*

Nro.	Instancia	ABT		ABLI		AES	
		Mejor	Promedio	Mejor	Promedio	Mejor	Promedio
1	20-5	4.93	4.91	4.72	4.64	4.64	4.50
2	30-10	4.64	4.61	4.98	4.89	4.47	4.33
3	40-10	10.20	10.17	9.01	8.75	8.09	7.63
4	80-20	22.97	22.94	19.83	19.56	19.22	18.60
5	100-20	24.05	24.01	19.61	19.30	19.97	19.52
6	100-25	25.47	25.42	18.80	18.45	19.06	18.05
7	120-40	29.29	29.24	26.27	25.77	24.91	24.56
8	150-30	30.52	30.47	27.07	26.75	25.29	24.83
9	150-50	36.69	36.65	29.21	28.85	28.56	27.80
10	160-40	39.43	39.39	34.40	33.58	31.11	30.34
11	165-55	41.04	41.01	36.34	35.97	33.93	33.45

12	200-40	47.18	47.13	38.71	38.11	38.18	37.62
13	225-45	56.03	55.94	42.78	42.06	47.43	46.82
14	240-60	60.49	60.37	49.42	48.83	48.26	46.08
15	250-50	69.95	69.88	56.95	55.86	53.93	53.40
Promedio Neto		**33.52**	**33.48**	**27.87**	**27.42**	**27.14**	**26.50**

Fuente: Elaboración propia

Asimismo, para un mayor detalle de las pruebas realizadas al integrar el algoritmo de búsqueda local al algoritmo memético propuesto consultar el Anexo 23. Finalmente, de los resultados obtenidos se puede concluir lo siguiente:

- El mejor rendimiento registrado es el correspondiente al algoritmo de Búsqueda Tabú, por ende, se elige este algoritmo para ser usado como parte del algoritmo memético que se desea estudiar en el presente proyecto de tesis.
- El resultado final promedio de todos los algoritmos de búsqueda local probados aumenta considerablemente al aplicarse en conjunto con el algoritmo evolutivo. Esto evidencia que la estrategia que usa el algoritmo memético al combinar estos métodos de optimización es efectivo y brinda una mejora sustancial en los resultados finales.
- Los parámetros empleados en las pruebas no fueron optimizados, por lo cual se espera que el algoritmo brinde aún mejores resultados luego del proceso de calibración de parámetros que se planea realizar en el Capítulo 7.

6.2.1.2 Métodos, medios de verificación e indicadores

Para poder implementar exitosamente al algoritmo memético se hizo uso, por un lado, del lenguaje de programación Java, y gracias a que este posee un paradigma orientado a objetos se pudo organizar de mejor manera a la gran cantidad de estructuras de datos y funciones auxiliares que posee el algoritmo. Por otro lado, se hizo uso de pruebas unitarias de software definiendo como una “unidad” a cada uno de los métodos implementados y usados directamente en el algoritmo memético. Estas pruebas ayudaron a lograr una gran confiabilidad en los resultados finales, ya que cada uno de los componentes del algoritmo fue probado de manera gradual e independiente para comprobar su adecuado funcionamiento.

Asimismo, para poder planificar y gestionar el proceso de implementación del algoritmo memético se hizo uso de un tablero Kanban brindado por la herramienta informática “Notion”, con ayuda de la cual se logró mostrar y clasificar las tareas según el nivel de avance de las mismas. Gracias al uso de este tablero se pudo medir fácilmente, y en todo momento, el avance de la implementación, además de que permitió identificar tareas difíciles que debían ser divididas en tareas más pequeñas con el objetivo de implementar funcionalidades cortas, pero de manera constante para no perder el ritmo de desarrollo.

Por otro lado, se hizo uso de algunos principios y prácticas recomendadas en el marco de referencia Extreme Programming tales como:

i. Realizar pruebas unitarias

   Esta práctica consiste en asegurar que todas las “unidades” planteadas sean debidamente probadas. Esto se logró planificando, para cada método implementado que forma parte del algoritmo memético, un conjunto de casos de prueba que debía superar. Luego, con ayuda de herramienta “Junit” en su versión 5, se procedió a implementar las pruebas, para finalmente ejecutarlas y registrar los resultados correspondientes.

ii. Mantener estándares de codificación,

   Esta práctica indica que el código debe tener un estilo consistente para que pueda ser fácil de entender y mantener, por lo cual se plantea utilizar un estándar de elaboración propia, cuya descripción detallada se encuentra en el Anexo 12. La implementación completa del algoritmo y sus funciones auxiliares se realizó siguiendo dicho estándar.

iii. Asumir la simplicidad

   Este principio consiste en que el programador debe centrarse en el trabajo más relevante en el momento, es decir que se debe priorizar aquello que se ha planteado como parte del entregable. En ese sentido, se priorizaron las tareas presentes en el tablero Kanban correspondientes al desarrollo del algoritmo memético.

iv. Cambio incremental

   Este principio consiste en que cuando se necesite agregar alguna funcionalidad, es

mejor tratar de aplicar cambios pequeños, ya que suele ser mejor que aplicar un cambio grande, pues estos pueden tener errores más difíciles de solucionar. En este caso, se procuró dividir las tareas de tal manera que se implemente primero las funciones auxiliares de manera independiente, para luego pasar a la implementación del algoritmo principal. Asimismo, cuando alguna función auxiliar necesitó funcionalidades extra, estas fueron consideradas como tareas nuevas, asegurando tener tareas valiosas pero pequeñas y fáciles de reparar en caso de detectar algún problema.

Cabe mencionar que el uso de estos principios y buenas prácticas ayudó a que el proceso de implementación sea eficiente en cuanto al tiempo disponible, efectivo en cuanto al cumplimiento de las funcionalidades y ordenado gracias a los estándares y a la priorización de tareas.

Este resultado alcanzado se encuentra codificado y almacenado en la nube y puede ser accedido mediante el enlace que se encuentra en el Anexo 13. Por otro lado, el documento de pruebas unitarias detallado puede ser visualizado en el Anexo 14, dicho documento contiene el detalle de las pruebas unitarias que fueron implementadas y ejecutadas sobre el algoritmo memético y sus funciones auxiliares.

Finalmente, este resultado ha sido revisado y validado en su totalidad por un especialista en el diseño algoritmos, el cual ha determinado que tanto la implementación como las pruebas realizadas son adecuadas para un proyecto de fin de carrera. Asimismo, se validó que las pruebas unitarias hayan sido exitosas en un 100%. El documento de validación firmado puede visualizarse en el Anexo 15.

6.2.2 Interfaz gráfica de ejecución del algoritmo memético

6.2.2.1 Descripción

En esta sección se describe el resultado alcanzado referente a la construcción de una interfaz que permita ejecutar el algoritmo memético desarrollado. Dicha interfaz sirve como una herramienta para poder aplicar el algoritmo memético al problema de asignación de tablas a

unidades de almacenamiento de manera rápida y sencilla. El detalle de cada una de las pantallas implementadas se muestra a continuación:

- **Pantalla de carga de datos**

  En esta pantalla se carga toda la información necesaria para poder realizar un proceso de asignación de tablas a unidades de almacenamiento, es decir, se carga el tamaño de las tablas, la frecuencia de uso de las mismas y sus respectivos coeficientes de importancia, asimismo, se carga la capacidad disponible de los discos y su rendimiento. Toda esta información debe estar distribuida en dos archivos de tipo "hoja de cálculo" cuyos formatos pueden visualizarse en el Anexo 16. Una representación visual de esta pantalla se muestra en la **Figura 11**.

Figura 11. *Pantalla de carga de datos*

Fuente: Elaboración propia

- **Pantalla de carga de parámetros del algoritmo memético**

  En esta pantalla se cargan todos los parámetros que utilizará el algoritmo memético para realizar su ejecución. Para organizar mejor la información, se decide separar a dichos parámetros en secciones, y estas son:

  i. Ejecución del algoritmo

Donde se encuentran los parámetros que controlan la ejecución del algoritmo y permiten, por un lado, limitar la cantidad de tiempo que se quiere ejecutar el algoritmo y, por otro lado, controlar el número de veces que se quiere correr el mismo.

ii. Algoritmo evolutivo

Este conjunto de parámetros es propio del algoritmo evolutivo, desde esta sección se puede controlar la cantidad de agentes que tendrá la población, el máximo número de iteraciones que se permitirá, el número máximo de generaciones sin mejorar la solución antes de parar el algoritmo, entre otros.

iii. Búsqueda local

Finalmente se encuentran los parámetros del algoritmo de búsqueda local, que tal y como se indicó en la sección anterior, se utiliza el algoritmo de Búsqueda Tabú. Con ayuda de estos parámetros se puede controlar cada cuántas generaciones se quiere aplicar la búsqueda local, sobre cuántos agentes se aplicará, el número máximo de iteraciones permitidas, entre otros.

Estas secciones sirven para para facilitar su entendimiento y además de ello se tiene un botón en el parte inferior llamado “Autocompletar campos”, el cual completa todos los campos con valores recomendados para este algoritmo. Una representación visual de esta pantalla se muestra en la **Figura 12**.

Figura 12. *Pantalla de carga de parámetros del algoritmo*

Fuente: Elaboración propia

- **Pantalla de progreso de la ejecución del algoritmo**

  En esta pantalla se encuentra un botón que inicializa la ejecución del algoritmo, seguido por el porcentaje de progreso de este y una barra de progreso, la cual muestra el avance que tiene la ejecución del algoritmo y permite verificar que está procesándose correctamente. Luego de finalizar la ejecución, se activa el botón de “Mostrar resultados” y puede pasarse a la siguiente pantalla. Una representación visual de la pantalla de ejecución puede mostrarse en la **Figura 13**.

Figura 13. *Pantalla de progreso de ejecución del algoritmo*

Fuente: Elaboración propia

- **Pantalla de resultados**

  En esta pantalla se muestran los resultados alcanzados por el algoritmo, estos resultados son: el tiempo total en segundos que demoró la ejecución, la cantidad total de procesos iterativos realizados y la mejor solución que se logró obtener. Esta última se representa mediante un cuadro en el cual cada columna representa a un disco del sistema de almacenamiento y cada fila está compuesta por las tablas asignadas a cada disco. En la **Figura 14** puede verse una representación gráfica de esta pantalla.

Figura 14. *Pantalla de resultados*

Fuente: Elaboración propia

6.2.2.2 Métodos, medios de verificación e indicadores

La implementación de la interfaz del software de ejecución se realizó haciendo uso del lenguaje de programación Java y la librería JavaFX en su versión 15. Asimismo, se hizo uso de pruebas unitarias automatizadas para validar la correctitud del software. Estas pruebas también ayudaron a validar que los componentes del software funcionan adecuadamente y son suficientes para completar el proceso de ejecución desde el punto de vista de un usuario. Para poder planificar y gestionar el proceso de implementación se hizo uso de un tablero Kanban brindado por la herramienta informática “Notion”, tal como se hizo en la sección anterior, en este tablero se clasificaron las tareas pendientes según el nivel de avance de las mismas. En este caso, el entregable completo fue separado en tareas más pequeñas para facilitar su implementación.

Asimismo, se tomaron en cuenta los siguiente principios y buenas prácticas del marco de referencia Extreme Programming:

i. Realizar pruebas unitarias

   Esta práctica consiste en asegurar de que cada una de las “unidades” implementadas sean debidamente probadas. Esto se logró, primero, definiendo como unidad a cada

una de las pantallas mostradas anteriormente, luego de ello, se procedió a diseñar un conjunto de casos de prueba que permita validar el correcto funcionamiento y diseño de cada componente al interior de la unidad, finalmente, con ayuda de la herramienta "TestFX" en su versión 4, se procedió a implementar las pruebas, para luego ejecutarlas y registrar los resultados correspondientes.

ii. Asumir la simplicidad

Este principio recomienda a los desarrolladores centrarse en el trabajo relevante en el momento, es decir que se debe priorizar aquello que se propuso como entregable y a pesar de que este pueda tener fallas, debe asumirse que será "sencillo" solucionarlas y se logrará la meta. En ese sentido, con ayuda del tablero Kanban, se priorizaron las tareas correspondientes a la implementación de la interfaz de ejecución, ya que es el entregable propuesto como objetivo.

iii. Mantener el diseño simple

Esta buena práctica consiste en diseñar las soluciones pensando en lo que es indispensable y no agregando funcionalidades innecesarias. En el presente entregable se diseñó la interfaz pensando en aquello que es completamente necesario para ejecutar el algoritmo de manera óptima, lo cual ayudó a tener un diseño más simple y que cumple con las expectativas del especialista.

Por otro lado, la implementación de este resultado alcanzado, al igual que la del algoritmo memético, puede ser accedido mediante el enlace que se encuentra en el Anexo 13. Asimismo, el documento de pruebas unitarias detallado puede ser visualizado en el Anexo 17, dicho documento contiene una descripción detallada de todas las pruebas unitarias realizadas.

Además, el resultado completo con las pruebas unitarias y la integración de la interfaz con el algoritmo memético fue validado en su totalidad por un especialista en el diseño de algoritmos, el cual ha determinado que tanto la implementación como las pruebas realizadas son correctas para un proyecto de fin de carrera. El documento de validación se encuentra en el Anexo 18.

6.3 Discusión de resultados

En este capítulo se detalla el logro del tercer objetivo específico: *"Desarrollar un software que permita la ejecución del algoritmo memético aplicado a la optimización de la asignación de tablas a unidades de almacenamiento de manera automática".* El cual es evidenciado a través de la presentación detallada de dos resultados alcanzados.

En primer lugar, se tiene a la implementación del algoritmo memético, un resultado que consta en detallar los aspectos principales de la implementación del algoritmo a estudiar, y de los métodos, herramientas y validaciones empleados en su realización. Esta implementación fue realizada siguiendo fielmente el diseño de las estructuras de datos y las funciones auxiliares del algoritmo memético presentado en el Capítulo 5. Asimismo, se presenta el proceso de implementación y pruebas del algoritmo de búsqueda local empleado por el algoritmo memético, el cual, como se menciona en la descripción del resultado, es una de las partes más importantes del algoritmo a estudiar y, debido a que no hay suficiente evidencia para determinar qué algoritmo es el adecuado para este caso de estudio, se procede a realizar un conjunto intensivo de pruebas sobre tres algoritmos seleccionados, eligiendo finalmente aquel que obtuvo un mejor rendimiento: El Algoritmo de Búsqueda Tabú.

En segundo lugar, se tiene a la implementación de una interfaz que permita ejecutar el algoritmo memético implementado, el cual es un resultado que consiste en presentar el detalle de las pantallas implementadas que sirven para aplicar el algoritmo memético en la tarea de asignación de tablas de una forma efectiva y automática. En este resultado alcanzado se toma en cuenta las necesidades más básicas para aplicar esta tarea utilizando un algoritmo de optimización.

Una variedad de herramientas y métodos, tales como el lenguaje de programación Java, las herramientas informáticas para crear pruebas unitarias, la metodología Kanban y las buenas prácticas recomendadas en el marco Extreme Programming fueron claves para alcanzar los resultados mencionados, pues permitieron tener un estándar, orden y facilidad al momento de organizar y realizar la implementación de las soluciones planteadas.

Los resultados alcanzados en el presente capítulo evidencian que fue posible construir una

herramienta de software que permita el uso de un método de optimización especializado, es decir el algoritmo memético, en la tarea de asignación de tablas a unidades de almacenamiento de manera automática y no usando métodos rudimentarios o manuales que puedan estar propensos a errores que afecten gravemente el rendimiento del sistema de base de datos.

Capítulo 7. Comparación del desempeño de algoritmos

7.1 Introducción

En el presente capítulo se desarrolla el cuarto objetivo específico: *"Comparar el desempeño del algoritmo memético propuesto contra la solución más relevante encontrada en el estado del arte con respecto a la optimización de la asignación de tablas a unidades de almacenamiento con la finalidad de verificar que brinda buenos resultados y puede ser aplicado con éxito en dicha tarea de optimización".* Para evidenciar que este objetivo ha sido logrado se presentan cuatro resultados alcanzados, los cuales son, en primer lugar, el diseño del algoritmo GRASP encontrado en el estado del arte y adaptado al problema de asignación de tablas a unidades de almacenamiento, el cual consiste en mostrar el detalle del diseño de dicho algoritmo y sus principales funciones auxiliares. En segundo lugar, la codificación del algoritmo GRASP, el cual consiste en realizar una descripción de los aspectos principales de la implementación de este algoritmo y cómo se relaciona con el diseño del mismo. En tercer lugar, la calibración de las variables que utilizaran los algoritmos memético y GRASP, el cual consiste en describir las estrategias y resultados de las pruebas de calibración diseñadas. Finalmente, el cuarto resultado alcanzado es la experimentación numérica, la cual permite evaluar y comparar el desempeño de los dos algoritmos implementados, este resultado consiste en detallar el proceso de experimentación y el conjunto de pruebas utilizadas. Adicionalmente, se tiene una sección en la cual se discuten los problemas causa abordados al alcanzar estos resultados.

7.2 Resultados alcanzados

A continuación, se presenta el detalle de los resultados alcanzados en conjunto con los métodos, medios de verificación e indicadores correspondientes.

7.2.1 Diseño del algoritmo GRASP

7.2.1.1 Descripción

El nombre completo del algoritmo “GRASP” puede traducirse al español como: “Procedimiento Codicioso de Búsqueda Aleatoria Adaptativa”. Sin embargo, para una mayor simplicidad se prefiere usar el término “GRASP” para referirse a este. A continuación, se presenta el diseño del algoritmo GRASP y sus principales funciones auxiliares.

7.2.1.1.1 Algoritmo GRASP

El algoritmo GRASP es una metaheurística iterativa usada comúnmente en problemas de optimización complejos. Este algoritmo consiste generalmente de dos fases: la construcción y la búsqueda local. En la fase de construcción se genera una solución factible, que luego será mejorada al pasar por la fase de búsqueda local. Estas fases se repiten iterativamente y se conserva la mejor solución (Jean-Yvesspotvin, 2010). En la **Figura 15** puede visualizarse un diseño general del algoritmo GRASP.

Figura 15. *Diseño del algoritmo GRASP*

```
Inicio ejecutarAlgoritmoGRASP (P, param)
1:  maxIter = param
2:  nIter = 1
3:  sMejor
4:  sNueva, sMejorada = solucionesVacias ()
5:  Desde nIter Hasta maxIter Hacer:
6:     sNueva = construirNuevaSolucion (P, param)
7:     sMejorada = aplicarBusquedaLocal (P, sNueva, param)
8:     actualizarSolucion (sMejor, sMejorada)
9:  Retornar sMejor
Fin ejecutarAlgoritmoGRASP
```

Fuente: Adaptado de “*Manual de Algoritmos Metaheurísticos*” por Jean-Yvess Potvin y Michel Gendreau (2010).

Donde:

- Se tiene como parámetro a los siguientes elementos:

    - P es la estructura “Problema” y contiene toda la información del problema que se desea solucionar.
    - Param es el conjunto de parámetros necesarios para lograr una correcta ejecución del algoritmo
- Línea 1: se obtiene como parámetro al máximo número de iteraciones permitidas
- Línea 2: se inicializa el contador de iteraciones
- Línea 3: se declara la variable “sMejor”, la cual almacenará la mejor solución que logra obtener el algoritmo y se actualiza en cada iteración en la que esta solución logra ser mejorada
- Línea 4: se inicializan las variables “sNueva” y “sMejorada”, que servirán como variables intermedias en cada una de las fases del algoritmo
- Línea 5: Se inicia el bucle que no finaliza hasta alcanzar el número máximo de iteraciones permitidas
    - Línea 6: se aplica el proceso de construcción para generar una nueva solución factible, la cual se almacena en la variable “sNueva”
    - Línea 7: se aplica el proceso de búsqueda local sobre la solución recién generada. El resultado se almacena en la variable “sMejorada”
    - Línea 8: se actualiza la nueva mejor solución. Si la solución mejorada resulta ser mejor que la “mejor solución” actual, se procede a actualizar la variable sMejor para contener a la nueva mejor solución
- Línea 9: Una vez finalizadas todas las iteraciones, se procede a retornar la mejor solución encontrada

Asimismo, cabe mencionar que el diseño del algoritmo GRASP que se va a emplear en el presente proyecto de tesis está basado en el algoritmo propuesto en la investigación más relevante encontrada en el estado del arte con respecto a la optimización de la asignación de tablas a unidades de almacenamiento. Esta investigación se titula “Algoritmo GRASP

doblemente relajado para resolver el problema de asignación de tablas a dispositivos de almacenamiento", y emplea como **único** componente a la fase de construcción de la solución, es decir que en este caso no se contempla la utilización de una fase de mejora.

Por tal motivo, el algoritmo GRASP a emplearse en el presente proyecto de tesis tampoco contempla la fase de mejora, asegurando que se mantenga la coherencia con la investigación original encontrada en el estado del arte.

7.2.1.1.2 Fase de construcción

Tal como se menciona en la sección previa, la fase de construcción es el componente principal del algoritmo GRASP que se desea emplear en el presente proyecto de tesis. El método de construcción más básico consiste en generar una solución iterativamente, haciendo uso de lo que se llama como "lista restringida de candidatos" o RCL por sus siglas en inglés. Esta lista, como su propio nombre indica, contiene los elementos candidatos para formar parte de la solución final. Para construir esta lista es necesario conocer qué elementos ya forman parte de la solución y qué elementos siguen disponibles. Además, para asegurar que obtenga la mejor calidad posible en la solución (criterio codicioso), los elementos que se incorporan a la lista restringida de candidatos deben tener una calidad que se encuentre en un rango de valores determinado por un "parámetro de umbral" (Jean-Yvesspotvin, 2010). Este rango de valores es calculado tal como se muestra en la siguiente fórmula:

$$e^{mejor} - \alpha * (e^{mejor} - e^{peor}) \leq \text{e} \leq e^{mejor}$$

En donde "e" corresponde al elemento disponible para formar parte de la lista restringida de candidatos, "e^{mejor}" es el candidato de mejor calidad, "e^{peor}" es el candidato de peor calidad y "α" es el parámetro de umbral. Por lo tanto, se dice que aquel elemento "e" que pertenezca a dicho rango, pasará a formar parte de la lista restringida de candidatos. Luego de haber formado esta lista se elige aleatoriamente un elemento para formar parte de la solución final. Este proceso se repite hasta que la solución tenga todos sus elementos.

Dicho esto, cabe aclarar que, para el caso de la tarea de asignación de tablas a unidades de almacenamiento, la investigación que se está tomando como referencia propone un enfoque

diferente, haciendo uso de dos listas restringidas de candidatos, de las cuales la primera sirve para elegir una tabla adecuada haciendo uso de un parámetro de umbral “alfa”, y la segunda sirve para elegir un disco al cual dicha tabla debe ser asignada, haciendo uso, de igual manera, de un parámetro de umbral “beta”. El diseño adaptado de este enfoque se muestra en la **Figura 16**.

Figura 16. *Diseño del algoritmo de construcción*

```
Inicio construirNuevaSolucion (P, param)
1:  alfa, beta, nTablas, nDiscos = param, P
2:  resultado, tablasAsig = []
3:  tablasDisp, discosDisp = crearTodoDisponible ()
4:  puntajesT = calcularPuntajeTablas (P, nTablas)
5:  Mientras tamanho(tablasAsig) < nTablas:
6:     mejorT, peorT = obtenerMejorYPeorElemento (tablasDisp, puntajesT)
7:     RCLT = elegirElementos (tablasDisp, alfa, mejorT, peorT, puntajesT)
8:     iTabla = elegirAleatorio (RCLT)
9:     puntajesD = calcularPuntajeDiscos (iTabla, P)
10:    mejorD, peorD = obtenerMejorYPeorElemento (discosDisp, puntajesD)
11:    RCLD = elegirElementos (discosDisp, beta, mejorD, peorD, puntajesD)
12:    iDisco = elegirAleatorio (RCLD)
13:    agregar (resultado, iTabla, iDisco)
14:    agregar (tablasAsig, iTabla)
15: Retornar resultado
Fin construirNuevaSolucion
```

Fuente: Adaptado de *“Algoritmo GRASP doblemente relajado para resolver el problema de asignación de tablas a dispositivos de almacenamiento”* por R. Cueva y M. Tupia (2008).

Donde:

- Se tiene como parámetro a los siguientes elementos:
    - P es la estructura “Problema” y contiene toda la información acerca de las tablas y discos que se desea distribuir.
    - Param es el conjunto de parámetros necesarios para lograr una correcta ejecución del algoritmo.
- Línea 1: se obtienen los dos parámetros de umbral (alfa y beta) necesarios para crear las listas restringidas de candidatos, además del número de tablas y discos que se empleará.

- Línea 2: se inicializan como vacías las listas que contienen el resultado y las tablas asignadas, respectivamente.
- Línea 3: se crean las listas de control de disponibilidad: tablasDisp y discosDisp, las cuales almacenan, por cada elemento, si se encuentra disponible, al valor correspondiente a su índice y si no lo está, al número -1.
- Línea 4: se calcula previamente el puntaje de cada una de las tablas, el cual es necesario para saber qué tan prioritaria es una tabla con respecto a las demás. El diseño planteado en la investigación de R. Cueva y M. Tupia, propone que el administrador de la base de datos pueda ser quien decida si priorizar las tablas según su “tamaño” o su “número de lecturas y escrituras”. Por ende, se plantea lo mismo en esta ocasión.
- Línea 5: se inicia el bucle principal, el cual no se detiene hasta que todas las tablas hayan sido asignadas o, lo que es lo mismo, que la solución se encuentre completa.
- Línea 6: se obtiene el mejor y peor elemento de la lista de tablas disponibles, para lo cual se compara el puntaje de prioridad de cada una de las tablas.
- Línea 7: se filtran los elementos disponibles para formar la lista restringida de candidatos de las tablas o RCLT como se nombra en el diseño. Para ello se hace uso del mejor y peor elemento, y del parámetro de umbral “alfa”, con los cuales es posible realizar los cálculos correspondientes en base a la fórmula para hallar el rango de valores permitidos descrita al principio de esta sección, la cual también fue utilizada en la investigación que se toma como referencia.
- Línea 8: finalmente se elige aleatoriamente un elemento de la lista para ser la tabla que se desea asignar.
- Línea 9: se calcula el puntaje de cada disco. Esto se logra haciendo uso de la “Función discriminante” propuesta por los autores R. Cueva y M. Tupia en la investigación tomada como referencia. La fórmula de esta función es la siguiente:

$$Función\ Discriminante = \frac{CapacDisco}{TablaSelecc} * \left(1 - \frac{SumaTablasEnDisco}{SumaTotalTablas}\right)$$

Donde:

i. *CapacDisco*: la capacidad o rendimiento del disco que se está evaluando.

ii. *TablaSelecc*: es el puntaje de la tabla seleccionada para ser asignada.

iii. *SumaTablasEnDisco*: es la suma de todos los puntajes de las tablas presentes en el disco que se está evaluando.

iv. *SumaTotalTablas*: es la suma de todos los puntajes de las tablas.

- Línea 10: luego se llama a la función "**`obtenerMejorYPeorElemento`**" para obtener el mejor y peor disco.
- Línea 11: se forma la lista restringida de candidatos de los discos o RCLD como se nombra en el diseño. Para lo cual se hace uso del mejor y peor elemento, y del parámetro de umbral "beta". Con los cuales, al igual que en la Línea 7, es posible determinar el rango de facilidad para los elementos pertenecientes al RCLD.
- Línea 12: se elige aleatoriamente un elemento de la lista para ser el disco al cual se debe asignar la tabla seleccionada anteriormente.
- Línea 13-15: se actualiza el resultado final y la lista de tablas ya asignadas.
- Línea 16: una vez culminado el bucle, se retorna la solución contenida en la variable "resultado".

7.2.1.1.3 Pruebas de flujo de datos

Finalmente, en el Anexo 19 se presenta el detalle de las pruebas de flujo de datos realizadas, las cuales tienen el objetivo de detectar posibles errores en el diseño del algoritmo, a través de una simulación de su ejecución con datos conocidos. De estas pruebas se puede concluir lo siguiente:

i. El diseño planteado es correcto y no presenta errores que eviten su implementación.

ii. Diseñar pruebas de flujo de datos resultó muy útil para entender más a fondo la naturaleza del algoritmo.

iii. La lógica planteada en la investigación de R. Cueva y M. Tupia tiene un comportamiento adecuado para la tarea de asignación de tablas, sin embargo, se considera necesaria su implementación y la realización de un proceso de experimentación numérica para estudiar mejor su comportamiento en comparación al algoritmo Memético planteado en el presente proyecto de tesis.

7.2.1.2 Métodos, medios de verificación e indicadores

El resultado alcanzado se encuentra plasmado en la sección de diseño del algoritmo GRASP mostrado en el presente capítulo. Dicha sección contiene una descripción detallada del diseño del algoritmo y sus principales funciones auxiliares, lo cual sirve para entender de mejor manera al algoritmo y reducir la aparición de posibles errores en la fase de implementación.

Además, este resultado ha sido revisado y validado en su totalidad por un especialista en el diseño algoritmos, quien ha determinado que el diseño presentado es adecuado y las pruebas realizadas son suficientes para un proyecto de fin de carrera. El documento de validación firmado puede visualizarse en el Anexo 20.

7.2.2 Codificación del algoritmo GRASP

7.2.2.1 Descripción

En la sección previa se detalla el diseño del algoritmo GRASP encontrado en el estado del arte, así como sus funciones auxiliares principales. En esta sección se procede a implementar los componentes del algoritmo GRASP siguiendo fielmente la lógica planteada en el diseño de los mismos.

La codificación del algoritmo GRASP se realizó sin problemas gracias al diseño previo y al conjunto de métodos y herramientas que ayudaron a que se realice una correcta gestión del tiempo disponible para culminar la implementación.

Esta codificación se considera lo suficientemente precisa y adecuada como para realizar los procesos de calibración y experimentación numérica en las secciones subsiguientes.

7.2.2.2 Métodos, medios de verificación e indicadores

Para la codificación completa del algoritmo GRASP se hizo uso del lenguaje de programación Java, y gracias a que este posee un paradigma orientado a objetos se pudo organizar de mejor manera la información usada por este. Además, la experiencia ganada al implementar el algoritmo Memético ayudó a agilizar el proceso de codificación.

Asimismo, para poder planificar y gestionar el proceso de implementación del algoritmo GRASP se hizo uso de un tablero Kanban brindado por la herramienta informática “Notion”, con ayuda de la cual se logró mostrar y clasificar las tareas según el nivel de avance de las mismas.

Por otro lado, se hizo uso de algunos principios y prácticas recomendadas en el marco de referencia Extreme Programming tales como:

i. Realizar pruebas unitarias

   Esta práctica consiste en asegurar que todas las “unidades” planteadas sean debidamente probadas. En este caso se define como una “unidad” a cada uno de los métodos implementados y usados directamente en el algoritmo GRASP. Para cada unidad se planificó un conjunto de casos de prueba que debía superar. Luego, con ayuda de herramienta “Junit” en su versión 5, se procedió a implementar las pruebas, para finalmente ejecutarlas y registrar los resultados correspondientes. Estas pruebas ayudaron a lograr una gran confiabilidad en los resultados finales, ya que cada uno de los componentes del algoritmo fue probado de manera gradual e independiente para comprobar su adecuado funcionamiento.

ii. Mantener estándares de codificación

   Esta práctica indica que el código debe tener un estilo consistente para que pueda ser fácil de entender y mantener. Para cumplir esto se procede a utilizar los mismos estándares planteados en la ejecución del algoritmo Memético, los cuales se pueden encontrar en el Anexo 12. La implementación completa del algoritmo y sus funciones auxiliares se realizó siguiendo dicho estándar.

iii. Asumir la simplicidad

Este principio consiste en que el programador debe centrarse en el trabajo más relevante en el momento, es decir que se debe priorizar aquello que se ha planteado como parte del entregable y a pesar de que este pueda tener fallas, debe asumirse que será “sencillo” solucionarlas y se logrará la meta. En ese sentido, se priorizaron las tareas presentes en el tablero Kanban correspondientes al desarrollo del algoritmo GRASP.

iv. Cambio incremental

Este principio consiste en que cuando se necesite agregar alguna funcionalidad, es mejor tratar de aplicar cambios pequeños, ya que suele ser mejor que aplicar un cambio grande, pues estos pueden tener errores más difíciles de solucionar. En este caso, se procuró dividir las tareas para poder implementar al algoritmo según sus fases principales, codificando primero una estructura general del algoritmo, para luego pasar a cada una de las fases antes mencionadas.

El uso de estos principios y buenas prácticas ayudó a que el proceso de implementación sea eficiente en cuanto al tiempo disponible, efectivo en cuanto al cumplimiento de las funcionalidades y ordenado gracias a los estándares y a la priorización de tareas.

Este resultado alcanzado se encuentra codificado y almacenado en la nube y puede ser accedido mediante el enlace que se encuentra en el Anexo 13. Por otro lado, el documento de pruebas unitarias puede ser visualizado en el Anexo 21, dicho documento contiene el detalle de las pruebas unitarias que implementadas y probadas en el algoritmo GRASP.

Finalmente, este resultado ha sido revisado y validado en su totalidad por un especialista en el diseño algoritmos, el cual ha determinado que tanto la implementación como las pruebas realizadas son adecuadas para un proyecto de fin de carrera. Asimismo, se validó que las pruebas unitarias hayan sido exitosas en un 100%. El documento de validación firmado puede visualizarse en el Anexo 22.

7.2.3 Calibración de variables

La calibración de variables es un proceso empírico que permite encontrar la mejor combinación de variables a utilizar por los algoritmos de optimización. En esta oportunidad se realiza un proceso exhaustivo con el objetivo de tener el mejor rendimiento posible de los algoritmos que se desea estudiar.

7.2.3.1 Descripción

Para las pruebas de calibración realizadas se utiliza un conjunto de datos simulado con 40 tablas y 10 discos, esto debido a que suelen ser tamaños típicos promedio usados en contextos reales. En cada prueba de calibración el algoritmo se ejecuta 50 veces con cada valor y se registra el mejor resultado, el resultado promedio y la desviación estándar.

7.2.3.1.1 Resultados de la calibración de variables del algoritmo memético

El detalle de cada una de las pruebas de calibración puede visualizarse en el Anexo 24, en él se presenta un documento con cada una de las pruebas realizadas y los criterios de elección para cada variable. A continuación, en la **Tabla 10** se muestra el resumen de los resultados de calibración del algoritmo Memético.

Tabla 10. *Resultados de la calibración de variables del algoritmo Memético*

Variables del algoritmo evolutivo	
Variable calibrada	**Valor escogido**
Máximo número de iteraciones	800
Máximo número de generaciones sin mejora	200
Tamaño de la población	200
Tasa de casamiento	100%
Tasa de mutación	5%
Probabilidad de mutación	60%
Variables del algoritmo de búsqueda tabú	
Variable calibrada	**Valor escogido**
Número máximo de iteraciones de la BL	500
Número de vecinos a visitar	20
Cantidad de atributos tabú	100

Cada N generaciones	40
Sobre una cantidad M de agentes	90

7.2.3.1.2 Resultados de la calibración de variables del algoritmo GRASP

Con respecto al algoritmo GRASP, los valores elegidos para las variables utilizadas por el algoritmo son los mismos que se usan en la investigación con la cual se desea comparar el algoritmo Memético propuesto. A continuación, en la **Tabla 11** se muestran los valores de las variables a utilizar por el algoritmo GRASP.

Tabla 11. *Valores de las variables del algoritmo GRASP*

Variables del algoritmo GRASP	
Variable calibrada	**Valor escogido**
Máximo número de iteraciones	100
Parámetro de umbral Alfa	0.25
Parámetro de umbral Beta	0.25

7.2.3.2 Métodos, medios de verificación e indicadores

Para realizar las pruebas de calibración de variables se hizo uso del lenguaje de programación Java, con ayuda del cual se pudo correr iterativamente cada una de las pruebas y registrar los resultados correspondientes.

Asimismo, para poder planificar y gestionar el proceso de implementación y pruebas referentes a la calibración de variables se hizo uso de un tablero Kanban brindado por la herramienta informática “Notion”, con ayuda de la cual se logró mostrar y clasificar las tareas según el nivel de avance de las mismas.

Finalmente, este resultado ha sido revisado y validado en su totalidad por un especialista en el diseño algoritmos, el cual ha determinado que tanto la implementación como las pruebas de calibración realizadas son adecuadas para un proyecto de fin de carrera. El documento de validación firmado puede visualizarse en el Anexo 25.

7.2.4 Experimentación numérica

La experimentación numérica empleada en el presente proyecto de tesis tiene por objetivo estudiar el comportamiento de los algoritmos mediante la aplicación de diversas pruebas estadísticas sobre los resultados obtenidos por los mismos. En la presente sección se describe a detalle el proceso de experimentación y las conclusiones obtenidas de cada prueba realizada.

7.2.4.1 Descripción

7.2.4.1.1 Generación de las muestras

Para realizar el proceso de experimentación numérica y para poder analizar el comportamiento de los algoritmos propuestos, se generaron dos conjuntos de datos, lo cuales poseen 40 muestras de datos cada uno. Para simular los datos de cada muestra primero se generó un conjunto grande de datos aleatorios con las características que se muestran en la **Tabla 12**.

Tabla 12. *Valores del conjunto de datos generado aleatoriamente*

Dato	Número de valores	Rango de valores
Frecuencia de uso de las tablas	120	[155.0, 975.0]
Tamaño de las tablas	120	[45.0, 1055.0]
Rendimiento de los discos	40	[255.0, 1055.0]
Capacidades de los discos	40	[59169.0, 59760.0]

Luego, teniendo esta cantidad de valores generados aleatoriamente, se procede a seleccionar 80 muestras también de manera aleatoria, de las cuales las 40 primeras contienen 80 tablas y 10 discos y las 40 siguientes, 40 tablas y 10 discos. Esto con el objetivo de probar los algoritmos ante dos problemas de diferente magnitud. Los resultados mostrados en esta sección corresponden a las pruebas realizadas sobre el primer conjunto de datos (80 tablas y 10 discos), mientras que el segundo conjunto de datos (40 tablas y 10 discos) será utilizado en la sección de pruebas adicionales.

Cabe mencionar que en esa ocasión se está priorizando a cada tabla en función únicamente

de su frecuencia de uso, esto debido a que el algoritmo GRASP encontrado en el estado del arte no contempla la combinación de variables de las tablas al momento de calcular su importancia. Esto no significa una limitante pues ambos algoritmos están correctamente diseñados para procesar las entradas de datos descritas y brindar resultados adecuados en función de ellas.

7.2.4.1.2 Resultados

En la **Tabla 13** se muestran los resultados obtenidos por cada algoritmo estudiado sobre el conjunto de datos que contiene 40 muestras acerca de 80 tablas y 10 discos. Tal como se indicó anteriormente, cada algoritmo se ejecuta 20 veces, registrándose el mejor resultado, el resultado promedio y la desviación estándar.

Tabla 13. *Resultados de la experimentación*

Muestra	MEMÉTICO			GRASP		
	Mejor	Promedio	Desv. Estd.	Mejor	Promedio	Desv. Estd.
1	25.8018	25.7954	0.0049	24.9402	24.9002	0.0264
2	26.9714	26.9692	0.0018	26.0858	26.0244	0.0311
3	24.8545	24.8505	0.0031	23.6070	23.5447	0.0367
4	27.8194	27.8130	0.0043	27.2038	27.1648	0.0263
5	26.0893	26.0881	0.0010	24.7509	24.7150	0.0311
6	27.4689	27.4671	0.0021	26.6503	26.5270	0.0620
7	27.9787	27.9696	0.0091	27.3226	27.2613	0.0314
8	27.0776	27.0662	0.0075	26.4658	26.4197	0.0279
9	28.8920	28.8898	0.0026	28.1936	28.1540	0.0241
10	27.2752	27.2680	0.0049	26.7089	26.6746	0.0184
11	25.9797	25.9752	0.0023	25.4211	25.3722	0.0245
12	25.0477	25.0473	0.0006	23.4316	23.3728	0.0310
13	24.6244	24.6214	0.0024	23.6342	23.6038	0.0190
14	28.1421	28.1317	0.0077	27.6135	27.5479	0.0303
15	29.7722	29.7544	0.0119	29.2304	29.1912	0.0212
16	28.5919	28.5910	0.0010	27.7759	27.7421	0.0241
17	27.7411	27.7308	0.0054	27.0467	26.9934	0.0312
18	28.1294	28.1267	0.0021	27.4734	27.3884	0.0427
19	26.6800	26.6799	0.0001	25.7729	25.7436	0.0215
20	23.0409	23.0402	0.0008	22.0082	21.9446	0.0344
21	25.9856	25.9850	0.0011	24.7276	24.6729	0.0231
22	28.2026	28.1994	0.0014	27.3393	27.2540	0.0347
23	26.7061	26.7053	0.0007	25.8463	25.7909	0.0323

24	29.2515	29.2255	0.0177	28.7481	28.7066	0.0249
25	25.8858	25.8794	0.0048	25.2668	25.2410	0.0157
26	26.6465	26.6451	0.0011	25.6750	25.6274	0.0321
27	30.5315	30.5302	0.0018	29.9825	29.9414	0.0200
28	27.8055	27.8014	0.0031	27.2093	27.1213	0.0319
29	28.0225	28.0202	0.0019	27.1772	27.1067	0.0444
30	24.9482	24.9475	0.0009	23.7707	23.6988	0.0433
31	25.1226	25.1209	0.0028	23.9378	23.9130	0.0113
32	27.0665	27.0658	0.0012	25.8188	25.7843	0.0188
33	27.0723	27.0697	0.0014	25.8207	25.7727	0.0218
34	27.7610	27.7567	0.0062	26.6609	26.6381	0.0140
35	27.2876	27.2802	0.0052	26.7556	26.6844	0.0287
36	28.1687	28.1634	0.0045	27.5515	27.4732	0.0348
37	27.7712	27.7683	0.0024	27.1980	27.1564	0.0283
38	26.0574	26.0498	0.0092	25.4462	25.3560	0.0394
39	22.3179	22.3178	0.0001	19.7162	19.5978	0.0518
40	27.8456	27.8410	0.0026	27.0203	26.9849	0.0201
Promedio Neto	**26.9609**	**26.9562**	**0.0036**	**26.0751**	**26.0202**	**0.0292**

7.2.4.1.3 Prueba Shapiro-Wilk

Esta prueba permite evaluar la normalidad del conjunto de datos conformado por los mejores resultados mostrados en la sección anterior. Esta prueba es necesaria, ya que verifica que se cumpla el primer requisito para aplicar la prueba Z: que los datos se encuentren normalmente distribuidos, es decir, que no haya presencia de valores anómalos que puedan afectar el análisis. Dicho esto, a continuación, se presentan los resultados de las pruebas realizadas:

- **Prueba para el algoritmo memético**

Para esta prueba se plantean las siguientes hipótesis:

- H0: los valores de la muestra del algoritmo memético siguen una distribución normal
- H1: los valores de la muestra del algoritmo memético no siguen una distribución normal

En la **Tabla 14** se muestran los resultados calculados para la prueba.

Tabla 14. *Prueba Shapiro-Wilk de los resultados del Algoritmo Memético*

Estadísticos descriptivos del conjunto de datos	
Media	27.18
Desviación Estándar	1.67
Varianza	2.79
Mínimo	22.32
Máximo	30.53
Cantidad de datos	40.00
Resultados de la prueba	
Grado de significancia	0.05
P-valor	0.23

Como puede notarse en los resultados, el p-valor obtenido es mayor al nivel de significancia, por ende, se concluye que la hipótesis nula no es rechazada y la muestra sigue una distribución normal.

- **Prueba para el algoritmo GRASP**

Para esta prueba se plantean las siguientes hipótesis:

- H0: los valores de la muestra del algoritmo GRASP siguen una distribución normal
- H1: los valores de la muestra del algoritmo GRASP no siguen una distribución normal

En la **Tabla 15** se muestran los resultados calculados para la prueba.

Tabla 15. *Prueba Shapiro-Wilk de los resultados del Algoritmo GRASP*

Estadísticos descriptivos del conjunto de datos	
Media	26.08
Desviación Estándar	1.97
Varianza	3.86
Mínimo	19.72
Máximo	29.98
Cantidad de datos	40.00
Resultados de la prueba	
Grado de significancia	0.05
P-valor	0.06

De igual manera para los resultados del algoritmo GRASP, como puede notarse en los resultados, el p-valor obtenido es mayor al nivel de significancia, por ende, se concluye que

la hipótesis nula no es rechazada y la muestra sigue una distribución normal.

7.2.4.1.4 Prueba F

El segundo requisito para realizar una prueba Z exitosa es que las varianzas de las muestras sean significativamente homogéneas, es decir, que la variabilidad de sus resultados sea similar entre sí. Para comprobar ello, se realiza la prueba F de Fisher.

Para esta prueba se plantean las siguientes hipótesis:

- H0: las varianzas de los resultados son significativamente homogéneas
- H1: las varianzas de los resultados son diferentes

En la **Tabla 16** se muestran los resultados alcanzados en esta prueba.

Tabla 16. *Prueba F de Fisher de los algoritmos Memético y GRASP*

	Memético	GRASP
Media	27.18	26.08
Desviación Estándar	1.67	1.97
Varianza	2.79	3.86
Mínimo	22.32	19.72
Máximo	30.53	29.98
Cantidad de datos	40.00	
F	0.72	
Grado de significancia	0.05	
P-valor	0.32	

Como puede notarse en los resultados, el p-valor es mayor al nivel de significancia, por lo cual se concluye que no hay suficiente evidencia para rechazar la hipótesis nula. Por tal motivo, se concluye que las varianzas de los resultados de los algoritmos son significativamente homogéneas.

7.2.4.1.5 Prueba Z

Una vez comprobada la normalidad de las muestras y que estas poseen varianzas significativamente homogéneas, se puede proceder a realizar la prueba Z. Esta prueba permite comparar las medias de dos muestras de datos con tamaños significativos (mayores a 30 observaciones). En esta prueba se plantean dos pares de hipótesis, primero se busca verificar si las medias de ambas muestras son homogéneas usando la “prueba de dos colas”

y segundo, si la media de la muestra de resultados del algoritmo Memético es superior a la del algoritmo GRASP usando la “prueba de una cola”.

- **Prueba de dos colas**

Para esta prueba se plantean las siguientes hipótesis:

- H0: la media del algoritmo Memético es igual a la media del algoritmo GRASP
- H1: la media del algoritmo Memético es distinta a la media del algoritmo GRASP

En la **Tabla 17** pueden visualizarse los resultados de esta prueba adicional

Tabla 17. *Resultados de pruebas adicionales de Z de dos colas*

	Memético	GRASP
Media	27.18	26.08
Desviación Estándar	1.67	1.97
Varianza	2.79	3.86
Mínimo	22.32	19.72
Máximo	30.53	29.98
Cantidad de datos	40.00	
Prueba de dos colas		
Z	2.17	
Grado de significancia	0.05	
P-valor	0.03	

Como puede observarse, el p-valor resultante de la prueba de dos colas es menor al nivel de significancia, por lo que se concluye que se rechaza la hipótesis nula y que la media de los resultados obtenidos por los algoritmos propuestos es distinta.

- **Prueba de una cola**

Para esta prueba se plantean las siguientes hipótesis:

- H0: la media del algoritmo GRASP es mayor a la media del algoritmo Memético
- H1: la media del algoritmo GRASP es menor a la media del algoritmo Memético

En la **Tabla 18** pueden visualizarse los resultados de esta prueba adicional

Tabla 18. *Resultados de pruebas adicionales de Z de una cola*

	Memético	GRASP
Media	27.18	26.08
Desviación Estándar	1.67	1.97
Varianza	2.79	3.86

Mínimo	22.32	19.72
Máximo	30.53	29.98
Cantidad de datos	40.00	
Prueba de una cola		
Z	2.17	
Grado de significancia	0.05	
P-valor	0.01	

En esta prueba puede observarse que el p-valor resultante es menor que el nivel de significancia, por ende, se rechaza la hipótesis nula y se concluye que la media de los resultados obtenidos por el algoritmo GRASP es menor a la obtenida por el algoritmo Memético propuesto.

7.2.4.1.6 Pruebas adicionales

Adicionalmente a las pruebas con muestras de 80 tablas y 10 discos, se realizó la experimentación numérica considerando 40 tablas y 10 discos, esto con el objetivo de estudiar el comportamiento de los algoritmos en un problema de menor magnitud. Los resultados de esta experimentación y las pruebas de estadísticas pueden visualizarse en el Anexo 26. Las conclusiones de las pruebas realizadas se resumen a continuación

- En ambos casos (para el algoritmo Memético y GRASP) los resultados sobre las 40 muestras tienen una distribución normal.
- Las varianzas de los resultados de los algoritmos son significativamente homogéneas.
- Las medias de los resultados de los algoritmos Memético y GRASP son igualmente buenas

Finalmente, se concluye que para problemas de magnitud normal (40 tablas y 10 discos en este caso) ambos algoritmos brindan buenos resultados, sin embargo, para problemas de mayor magnitud (80 tablas y 10 discos), el algoritmo Memético es el que obtiene los mejores resultados en comparación al algoritmo GRASP.

7.2.4.2 Métodos, medios de verificación e indicadores

Para realizar las pruebas estadísticas se hizo uso de pruebas comúnmente utilizadas en

experimentación numérica, tales como la prueba de Shapiro-Wilk, la prueba F de Fisher y la prueba Z, con ayuda de las cuales pudo evaluarse de mejor manera al verdadero comportamiento de los algoritmos estudiados en el presente proyecto de tesis.

Asimismo, para poder ejecutar estas pruebas se hizo uso del software R Studio, el cual dispone de un conjunto de funcionalidades que facilitaron el proceso de experimentación. Finalmente, este resultado ha sido revisado y validado en su totalidad por un especialista en el diseño algoritmos, el cual ha determinado que las pruebas de experimentación realizadas son adecuadas para un proyecto de fin de carrera. El documento de validación firmado puede visualizarse en el Anexo 27.

7.3 Discusión de resultados

En este capítulo se detalla el logro del cuarto objetivo específico: "*Comparar el desempeño del algoritmo memético propuesto contra la solución más relevante encontrada en el estado del arte con respecto a la optimización de la asignación de tablas a unidades de almacenamiento con la finalidad de verificar que este brinda buenos resultados y puede ser aplicado con éxito en dicha tarea de optimización".* El cual es evidenciado a través de la presentación detallada de cuatro resultados alcanzados cuyas implicaciones se resumen a continuación:

En primer lugar, se tiene al diseño del algoritmo GRASP encontrado en el estado del arte. Este diseño se hizo siguiendo fielmente la lógica planteada en el estudio encontrado en la literatura, ya que este ha comprobado su efectividad en el problema de asignación de tablas a unidades de almacenamiento y se quiere mantener un alto nivel de coherencia con el mismo. Asimismo, se realizaron pruebas de caja blanca sobre el diseño del algoritmo para poder verificar que la lógica planteada en este sea correcta.

En segundo lugar, se tiene a la implementación del algoritmo GRASP diseñado, el cual es un resultado que consiste en utilizar el diseño planteado en el resultado anterior, para codificar el algoritmo GRASP en el lenguaje de programación Java, y así poder ser utilizado en futuras pruebas experimentales. Luego, se realizaron pruebas unitarias que validaron el correcto

funcionamiento del algoritmo.

En tercer lugar, se tiene al proceso de calibración de variables, el cual consiste en mostrar de manera detallada al proceso para encontrar la mejor combinación de valores de los parámetros para los algoritmos que se desea estudiar. Estos valores son usados en las pruebas de experimentación numérica correspondientes al cuarto resultado alcanzado.

En cuarto lugar, se tiene al proceso de experimentación numérica, el cual es un resultado que consiste en probar los algoritmos planteados en el presente proyecto de tesis con un amplio conjunto de muestras de datos y compararlos haciendo uso de pruebas estadísticas de hipótesis. Esto permite verificar la validez del algoritmo memético propuesto y estudiar su comportamiento en comparación con el algoritmo GRASP encontrado en el estado del arte.

Los resultados alcanzados en el presente capítulo evidencian que fue posible comparar el desempeño del algoritmo memético propuesto con la solución más relevante encontrada en el estado del arte y que pudo verificarse que el algoritmo propuesto brinda buenos resultados y puede ser empleado con éxito en la tarea de asignación de tablas a unidades de almacenamiento.

Capítulo 8. Conclusiones y trabajos futuros

En el presente capitulo se muestran las conclusiones alcanzadas al culminar los objetivos planteados en el proyecto de tesis. Asimismo, se tiene una sección de trabajos futuros en donde se proponen posibles extensiones a tomar en cuenta a partir de la investigación realizada.

8.1 Conclusiones

El presente proyecto de tesis busca abordar el problema central referente a la “deficiente asignación de tablas a unidades de almacenamiento de bases de datos relacionales”, para lo cual se plantea el objetivo general de “implementar un algoritmo memético para optimizar la asignación de tablas a unidades de almacenamiento de bases de datos relacionales”. En ese sentido, a continuación, se muestran las conclusiones alcanzadas al término del proyecto de tesis:

- Con respecto a la definición de una relación adecuada entre las variables o factores más relevantes relacionados a la asignación de tablas a unidades de almacenamiento, se realizaron entrevistas con especialistas en administración de bases de datos, gracias a las cuales las cuales pudo entenderse más a profundidad el problema de asignación de tablas y los factores más influyentes para éste en contextos reales. De esta manera, se logró identificar y definir los principales factores y variables del problema para luego ser incorporados a la definición de una función objetivo que relaciona todas estas variables y permite que puedan ser usadas para la optimización de la asignación de tablas a unidades de almacenamiento.
- En relación a la adaptación del algoritmo memético al problema de asignación de tablas, se procedió, en primer lugar, a identificar las estructuras de datos a ser usadas por este, luego, se realizó la adaptación de cada una de las funciones auxiliares del algoritmo memético, haciendo uso de las estructuras de datos planteadas, fuentes encontradas en el estado del arte acerca del diseño de algoritmos meméticos y la función objetivo diseñada como parte del objetivo anterior. Logrando, de esta manera, tener un diseño

detallado del algoritmo memético que se desea estudiar.

- En relación al desarrollo de una herramienta de software que permita ejecutar al algoritmo memético planteado, se procedió, en primer lugar, a codificar el diseño del algoritmo memético presentado en el objetivo anterior. Asimismo, se procedió a realizar un proceso intensivo de pruebas sobre la implementación de tres algoritmos de búsqueda local encontrados en el estado del arte, concluyendo que el algoritmo de búsqueda tabú es el que obtiene los mejores resultados y puede ser usado en conjunto con el algoritmo memético planteado. Luego, se procedió a incorporar este algoritmo al flujo principal del algoritmo memético, terminando, de esta manera, la codificación del algoritmo que se desea estudiar. En segundo lugar, se realizó el diseño e implementación de una interfaz desde la cual puede realizarse todo el proceso de ejecución del algoritmo memético, lo cual incluye el ingreso de datos del problema, la introducción de los parámetros del algoritmo y la visualización de los resultados. Para ambos resultados se hizo uso del lenguaje de programación Java, el marco de referencia Extreme Programming, el método Kanban y se realizaron pruebas unitarias que verifican su correcto funcionamiento.
- En relación a la comparación entre el algoritmo memético propuesto y la solución más relevante encontrada en el estado del arte, se procedió, en primer lugar, a diseñar y codificar esta solución siguiendo fielmente la lógica planteada en el estudio original. Para lograr este objetivo se hizo uso del lenguaje de programación Java y de diversas pruebas tales como pruebas de caja blanca (para comprobar que la lógica planteada en el diseño sea correcta) y pruebas unitarias de software (para verificar el correcto funcionamiento del algoritmo). Luego, se procedió a realizar un proceso de calibración de variables para obtener la combinación de valores de parámetros óptima para el algoritmo Memético y así poder comparar los algoritmos que se desea estudiar en las mejores condiciones posibles. Este proceso se hizo de manera empírica realizando 50 ejecuciones con cada una de las variables y sus respectivos valores propuestos. Luego, se registró el mejor resultado, el resultado promedio y la desviación estándar, los cuales son factores que se tomaron en cuenta para la elección final del valor que toma la variable en cuestión.

Finalmente, se procedió a realizar el proceso de experimentación numérica, primero, creando 40 muestras de datos acerca de 40 tablas y 10 discos y también 40 muestras acerca de 80 tablas y 10 discos, esto para poder probar a los algoritmos ante problemas no triviales de diferente magnitud. Luego, ambos algoritmos se corrieron 20 veces sobre cada una de las muestras, registrándose el mejor resultado, el resultado promedio, y la desviación estándar. Una vez que se obtuvo todos los resultados correspondientes, se procedió a realizar un conjunto de pruebas estadísticas sobre estos. Estas pruebas son: la prueba de normalidad de Shapiro-Wilk, la prueba F para comparar varianzas y la prueba Z para comparar las medias de los resultados obtenidos por los algoritmos. De dichas se obtienen los siguientes resultados: (i) Para el caso de las muestras con 40 tablas y 10 discos, ambos algoritmos (Memético y GRASP) brindan resultados igual de buenos y (ii) Para el caso de las muestras con 80 tablas y 10 discos, el algoritmo Memético es superior al algoritmo GRASP, por lo que se concluye que el algoritmo Memético tiene un mejor comportamiento en comparación al algoritmo GRASP cuando se trata de problemas de gran magnitud.

Finalmente, luego de haber evaluado la efectividad del algoritmo Memético propuesto frente al algoritmo GRASP encontrado en el estado del arte, se concluye que su aplicación en el problema de asignación de tablas a unidades de almacenamiento es viable para ser utilizado en contextos reales y en problemas con tamaños de gran magnitud. Asimismo, se concluye que pudo lograrse el objetivo general referente a la implementación de un algoritmo memético para optimizar la asignación de tablas a unidades de almacenamiento, con el cual se plantea una propuesta de solución que permite abarcar el problema central referente a la realización de una deficiente asignación de tablas a unidades de almacenamiento y, debido a que es un algoritmo que no fue aplicado con anterioridad al problema de asignación de tablas, se considera como un aporte para la comunidad científica.

8.2 Trabajos futuros

A continuación, se proponen algunas extensiones o trabajos futuros asociados al presente

proyecto de tesis:

- En primer lugar, con respecto a la aplicabilidad del diseño del algoritmo presentado, al ser construido en base a un problema de asignación generalizado, este puede ser extendido a diversos recursos como, por ejemplo, servidores de archivos y servidores de correo electrónico en donde los recursos a asignar sean los archivos y los correos electrónicos, respectivamente. Por lo tanto, un posible trabajo futuro es aplicar el Algoritmo Memético en problemas de alcance similar, esperándose tener resultados aceptables.
- En segundo lugar, se podría realizar la extensión del algoritmo propuesto para ser aplicado para asignar otros recursos de la base de datos tales como los índices y las vistas materializadas, los cuales podrían tener sus propias consideraciones y ayudaría a mejorar aún más el desempeño del sistema de base de datos.
- Finalmente, tanto el algoritmo Memético como la interfaz de ejecución del mismo pueden ser integrados como parte de un sistema de información empresarial o como una extensión de un sistema de administración de bases de datos con el objetivo de ser utilizada de manera directa por sus administradores.

Referencias

Ahmad, M. O., Markkula, J., & Oivo, M. (2013). Kanban in software development: A systematic literature review. *Proceedings - 39th Euromicro Conference Series on Software Engineering and Advanced Applications, SEAA 2013*, 9–16. https://doi.org/10.1109/SEAA.2013.28

Allen, G. (2012). *Beginning DB2: from novice to professional*. Apress : Distributed to the book trade worldwide by Springer Science+Business Media New York.

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

Alsultanny, Y. (2010). Database management and partitioning to improve database processing performance. *Journal of Database Marketing & Customer Strategy Management*, *17*(3–4), 271–276. https://doi.org/10.1057/dbm.2010.14

Berg, K. L., Seymour, T., & Goel, R. (2012). History Of Databases. *International Journal of Management & Information Systems (IJMIS)*, *17*(1), 29–36. https://doi.org/10.19030/ijmis.v17i1.7587

Booth, A. (2012). Systematic Approaches to a Successful Literature Review. In *Journal of the Canadian Health Libraries Association / Journal de l'Association des bibliothèques de la santé du Canada* (Vol. 34, Issue 1). https://doi.org/10.5596/c13-009

Burbank, D., & Knight, M. (2019). *Trends in Data Management A 2019 DATAVERSITY Report*. https://content.dataversity.net/rs/656-WMW-918/images/TrendsinDM_2019.pdf

Canim, M., Mihaila, G. A., Bhattacharjee, B., Ross, K. A., & Lang, C. A. (2009). An Object Placement Advisor for DB2 Using Solid State Storage. *Proc. VLDB Endow.*, *2*(2), 1318–1329. https://doi.org/10.14778/1687553.1687557

Cincy, W. C., & Jeba, J. R. (2018). A method of A-BAT algorithm based query optimization for crowd sourcing system. *International Journal of Intelligent Systems and Applications*, *10*(3), 33–40. https://doi.org/10.5815/ijisa.2018.03.04

Colley, D., & Stanier, C. (2017). Identifying New Directions in Database Performance Tuning. *Procedia Computer Science*, *121*, 260–265. https://doi.org/10.1016/j.procs.2017.11.036

Connolly, T. M., & Begg, C. E. (2016). *Database systems: a practical approach to design, implementation, and management* (6th ed). Pearson.

Cotta, C. (2016). *Una visión general de los algoritmos meméticos*. *January 2007*.

Cueva, R., & Tupia, M. (2008). Double Relaxation -- GRASP Algorithm for Solve the Table Assignment Problem on Data Base Store Devices. *2008 Fourth International Conference on Networked Computing and Advanced Information Management*, *2*(May 2015), 418–421. https://doi.org/10.1109/NCM.2008.186

Du, K.-L., & Swamy, M. N. S. (2016). *Search and optimization by metaheuristics techniques and algorithms inspired by nature*. Springer International Publishing. http://dx.doi.org/10.1007/978-3-319-41192-7

Extreme Programming. (2013). *Extreme Programming: A gentle introduction*. http://www.extremeprogramming.org/

Fallas, J. (2012). *Análisis de Varianza*. Universidad para la Cooperación Internacional. http://www.ucipfg.com/Repositorio/MGAP/MGAP-05/BLOQUE-ACADEMICO/Unidad-2/complementarias/analisis_de_varianza_2012.pdf

Fritchey, G. (2018). *SQL Server 2017 Query Performance Tuning*. Apress. https://doi.org/10.1007/978-1-4842-3888-2

Garcia-Molina, H., Ullman, J. D., & Widom, J. (2014). *Database systems: the complete book* (2. edition). Pearson.

Golubchik, L., Khanna, S., Khuller, S., Thurimella, R., & Zhu, A. (2009). Approximation algorithms for data placement on parallel disks. *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, *5*(4), 223–232. https://doi.org/10.1145/1597036.1597037

Grier, D. A. (2012). *The Relational Database and the Concept of the Information System*. 9–17. https://doi.org/10.1109/MAHC.2012.70

Harrington, J. L. (2016). *Relational database design and implementation, fourth edition*.

http://www.totalboox.com/book/id-2922977465074637430

Haupt, R. L., & Haupt, S. E. (2004). The Continuous Genetic Algorithm. *Practical Genetic Algorithms*, 51–66. https://doi.org/10.1002/0471671746.ch3

IBM Cloud Education. (2019). *Relational Databases*. https://www.ibm.com/cloud/learn/relational-databases

Idhaim, H. A. (2019). Selecting and tuning the optimal query form of different SQL commands. *International Journal of Business Information Systems*, *30*(1), 1. https://doi.org/10.1504/IJBIS.2019.097041

Jean-Yvesspotvin, M. (2010). *Handbook of Metaheuristics Third Edition*. http://www.springer.com/series/6161

Kamatkar, S. J., Kamble, A., Viloria, A., Hernández-Fernandez, L., & García Cali, E. (2018). Database performance tuning and query optimization. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics): Vol. 10943 LNCS* (pp. 3–11). https://doi.org/10.1007/978-3-319-93803-5_1

Kechar, M., & Nait-Bahloul, S. (2019). Bringing Together Physical Design and Fast Querying of Large Data Warehouses. *Proceedings of the 4th International Conference on Big Data and Internet of Things*, 1–8. https://doi.org/10.1145/3372938.3372947

Kundakcioglu, O. E. (2009). Encyclopedia of Optimization. *Encyclopedia of Optimization*, *January 2008*. https://doi.org/10.1007/978-0-387-74759-0

Lin, L., Zhu, Y., Yue, J., Cai, Z., & Segee, B. (2011). Hot random off-loading: A hybrid storage system with dynamic data migration. *IEEE International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems - Proceedings*, 318–325. https://doi.org/10.1109/MASCOTS.2011.41

Lindberg, S., & Strandberg, F. (2006). *The Development and Evaluation of a Unit Testing Methodology*. 161. https://pdfs.semanticscholar.org/24e6/5805ae9d920131a59d3973af59d89b594309.pdf

Luke, S., Domeniconi, C., Jong, K. De, Grefenstette, J., Vo, C., Harrison, J., Sullivan, K.,

Hrolenok, B., Langdon, B., Wiegand, R. P., Absetz, B., Branly, J., Compton, J., Donnelly, S., Haddon, W., Jamil, B., Kangas, E., Beirne, J. O., Rupakheti, P., … Alstrup, A. O. (2011). *Essentials of Metaheuristics A Set of Undergraduate Lecture Notes by*.

Maabreh, K. S. (2018). Optimizing Database Query Performance Using Table Partitioning Techniques. *ACIT 2018 - 19th International Arab Conference on Information Technology*, 1–4. https://doi.org/10.1109/ACIT.2018.8672584

Martí, R. (2003). *Procedimientos Metaheurísticos en Optimización Combinatoria* (pp. 2–60). Facultad de Matemáticas de la Universidad de Valencia. https://www.uv.es/rmarti/paper/docs/heur1.pdf

Massey, A., & Miller, S. J. (2016). Tests of Hypotheses Using Statistics. *Mathematics Department, Brown University, Providence, RI 2912*, 1–32.

Matalqa, S., & Mustafa, S. (2016). The effect of horizontal database table partitioning on query performance. *International Arab Journal of Information Technology*, *13*(1A), 184–189.

Myalapalli, V. K., Chakravarthy, A. S. N., & Reddy, K. P. (2015). Accelerating SQL queries by unravelling performance bottlenecks in DBMS engine. *2015 International Conference on Energy Systems and Applications*, 7–12. https://doi.org/10.1109/ICESA.2015.7503304

Myalapalli, V. K., Totakura, T. P., & Geloth, S. (2016). Augmenting database performance via SQL tuning. *2015 International Conference on Energy Systems and Applications*, 13–18. https://doi.org/10.1109/ICESA.2015.7503305

Neri, C., & Cotta, P. M. (2012). *Handbook of Memetic Algorithms. Studies in Computational Intelligence.*

Neri, F., Cotta, C., Moscato, P., & Kacprzyk, J. (Eds.). (2012). *Handbook of memetic algorithms* (Vol. 379). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-23247-3

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

Null, L., & Lobur, J. (2015). *The essentials of computer organization and architecture, fourth edition*. Jones & Bartlett Learning. http://www.books24x7.com/marc.asp?bookid=69820

Onwubolu, G. C., & Babu, B. V. (2004). *New Optimization Techniques in Engineering Springer-Verlag Berlin Heidelberg GmbH Studies in Fuzziness and Soft Computing , Volume 141*.

Oracle. (2019). *SQL Tuning Guide*. https://docs.oracle.com/en/database/oracle/oracle-database/12.2/tgsql/sql-tuning-guide.pdf

Oracle. (2020). *The Java Language Specification* (pp. 2–9). https://docs.oracle.com/javase/specs/jls/se14/jls14.pdf

Ou, J., Shu, J., Lu, Y., Yi, L., & Wang, W. (2014). EDM: An endurance-aware data migration scheme for load balancing in SSD storage clusters. *Proceedings of the International Parallel and Distributed Processing Symposium, IPDPS*, 787–796. https://doi.org/10.1109/IPDPS.2014.86

Paredaens, J., Bra, P., Gyssens, M., & Gucht, D. (1989). *The Structure of the Relational Database Model*. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-69956-6

Piccoli, G., & Pigni, F. (2016). *Information systems for managers: with cases* (Edition 3.). Prospect Press.

R Project. (n.d.). *What is R?* https://www.r-project.org/about.html

Raja Kumar Reddy, N., & Naidu, M. M. (2015). A genetic algorithmic approach for determining optimal secondary index set for querying on multiple relations. *International Journal of Applied Engineering Research*, *10*(8), 19195–19212. https://www.scopus.com/inward/record.uri?eid=2-s2.0-84929890949&partnerID=40&md5=773933f45766154ed8145b40b09cb3c5

Razali, N. M., & Wah, Y. B. (2011). Power comparisons of Shapiro-Wilk , Kolmogorov-Smirnov, Lilliefors and Anderson-Darling tests. *Journal of Statistical Modeling and Analytics*, *2*(1), 21–33. https://doi.org/doi:10.1515/bile-2015-0008

RStudio. (2020). *Take control of your R code*. https://rstudio.com/products/rstudio/#rstudio-desktop

Schnaitter, K., & Polyzotis, N. (2012). Semi-automatic index tuning. *Proceedings of the VLDB Endowment*, *5*(5), 478–489. https://doi.org/10.14778/2140436.2140444

Shi, H., Arumugam, R. V., Foh, C. H., & Khaing, K. K. (2013). Optimal disk storage allocation for multitier storage system. *IEEE Transactions on Magnetics*, *49*(6), 2603–2609. https://doi.org/10.1109/TMAG.2013.2250936

Silberschatz, A., Galvin, P. B., & Gagne, G. (2011). *Operating system concepts essentials*. John Wiley & Sons Inc.

Silberschatz, A., Korth, H. F., & Sudarshan, S. (2011). *Database system concepts* (6th ed). McGraw-Hill.

Stallings, W. (2015). *Operating systems: internals and design principles* (Eighth Edi). Pearson.

Sumathi, S., & Esakkirajan, S. (2007). *Fundamentals of relational database management systems* (Issue v. 47). Springer.

Szulc, I., Stencel, K., & Wiśniewski, P. (2017). Using Genetic Algorithms to Optimize Redundant Data. In *Communications in Computer and Information Science* (Vol. 716, pp. 165–176). https://doi.org/10.1007/978-3-319-58274-0_14

Thainiam, P. (2019). The Effects of Memes on Memetic Algorithms for Solving Quadratic Assignment Problem. *IEEE International Conference on Industrial Engineering and Engineering Management*, 1334–1338. https://doi.org/10.1109/IEEM44572.2019.8978780

Troncoso, C. E., & Daniele, E. G. (2004). Las entrevistas semiestructuradas como instrumentos de recolección de datos: una aplicación en el campo de las ciencias naturales. *Universidad Nacional Del Comahue - Consejo Provincial de Educación de Neuquen*, *4*(2), 12. http://artedialogico.com/sumak.cl/docto/2Ciencias/3Ciencias_Sociales/Metodologia/entrevistas semiestructuradas.pdf%0Afile:///C:/Users/GREICI/Downloads/3223-2923-1-PB(1).pdf

Valacich, J. S., & Schneider, C. (2017). *Information systems today: managing in the digital world* (8th ed). Pearson.

Wajszczyk, B., & Gruszka, I. M. (2020). Analysis of possibilities to increase the efficiency of the relative database management system using the methods of parallel processing. *Proceedings of SPIE - The International Society for Optical Engineering*, *11442*(May), 54. https://doi.org/10.1117/12.2565744

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

Wu, C.-H., Huang, C.-W., & Chang, C.-Y. (2019). A data management method for databases using hybrid storage systems. *ACM SIGAPP Applied Computing Review*, *19*(1), 34–47. https://doi.org/10.1145/3325061.3325064

Yoon, D. Y., Mozafari, B., & Brown, D. P. (2015a). DBSeer: Pain-free database administration through workload intelligence. *Proceedings of the VLDB Endowment*, *12*, 2036–2039. https://doi.org/10.14778/2824032.2824130

Yoon, D. Y., Mozafari, B., & Brown, D. P. (2015b). DBSeer. *Proceedings of the VLDB Endowment*, *8*(12), 2036–2039. https://doi.org/10.14778/2824032.2824130

Zhong-zhuang, W., Lun-dan, D., Yun-ting, W., & Zhi-wen, H. (2012). The Optimization of Large Information Management Database Bottlenecks. In *Advances in Intelligent and Soft Computing: Vol. 117 AISC* (Issue 127 VOL. 2, pp. 197–202). https://doi.org/10.1007/978-3-642-25437-6_28

Anexos

Anexo 1: Formulario de extracción de datos y resultado de la revisión sistemática

En la **Tabla 19** puede visualizarse el detalle del formulario de extracción usado en el presente proyecto de tesos, este es usado con el objetivo de obtener un mejor acercamiento a las respuestas de las preguntas de investigación planteadas en el Capítulo 3.

Tabla 19. *Formulario de extracción*

Campo	Descripción	Pregunta
Id	E[número] Por ejemplo: E001	General
Fecha de extracción		General
Autores		General
Título		General
Año de publicación		General
Causa de los problemas de rendimiento	Qué causa del problema de rendimiento se está investigando.	P1
Propuesta de solución a los problemas de rendimiento	Qué solución se propone al problema de rendimiento que se está investigando.	P2
Métodos de solución a los problemas de rendimiento	Qué método y/o tecnología se usó para la solución al problema de rendimiento que se está investigando.	P2
Implicancia de las unidades de almacenamiento en la investigación	La solución propuesta implica a las unidades de almacenamiento en el proceso de mejora del rendimiento de las bases de datos relacionales. (Sí/No)	P3
Implicancia de métodos o algoritmos	La solución implica utilizar algún método, algoritmo metaheurístico o algoritmo memético para la asignación	P3

de asignación en la investigación	en unidades de almacenamiento de bases de datos. (Sí/No)	

Un ejemplo del proceso de llenado del formulario se muestra en la **Tabla 20**.

Tabla 20. *Ejemplo de un formulario de extracción con datos reales*

Id	E001
Fecha de extracción	29/mayo/2020
Autores	Kamatkar, S.J. Kamble, A. Viloria, A. Hernández-Fernandez, L. García Cali, E.
Título	Database performance tuning and query optimization
Año de publicación	2018
Causa de los problemas de rendimiento	El autor divide los problemas de rendimiento de las bases de datos relacionales en 5 rubros: - Cuellos de botella de la CPU -Problemas con la estructura de la memoria - Capacidad de E/S del disco - Problemas en el diseño de la base de datos -Problemas relacionados a la indexación en una base de datos
Propuesta de solución a los problemas de rendimiento	Optimización de consultas
Método de solución a los problemas de rendimiento	La investigación brinda una serie de técnicas para optimizar las consultas.
Implicancia de las unidades de almacenamiento en la investigación	No
Implicancia de métodos o algoritmos de asignación en la investigación	No

Para observar el formulario de extracción completo dirigirse al siguiente enlace: "Formulario de extracción".

Finalmente, en la **Tabla 21** se muestra el resultado de la revisión sistemática realizada, cada una de las investigaciones seleccionadas será evaluada haciendo uso del formulario de extracción presentado anteriormente.

Tabla 21. *Documentos relevantes seleccionados*

Id	Referencia (formato APA)
E001	Kamatkar, S. J., Kamble, A., Viloria, A., Hernández-Fernandez, L., & García Cali, E. (2018). Database performance tuning and query optimization. In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics): Vol. 10943 LNCS. https://doi.org/10.1007/978-3-319-93803-5_1
E002	Saleh Maabreh, K., & Maabreh, K. S. S. (2018). Optimizing Database Query Performance Using Table Partitioning Techniques. 2018 International Arab Conference on Information Technology (ACIT), 1–4. https://doi.org/10.1109/ACIT.2018.8672584
E003	Wajszczyk, B., & Gruszka, I. M. (2020). Analysis of possibilities to increase the efficiency of the relative database management system using the methods of parallel processing. In P. Kaniewski & J. Matuszewski (Eds.), Radioelectronic Systems Conference 2019 (Vol. 11442, p. 54). SPIE. https://doi.org/10.1117/12.2565744
E004	Colley, D., & Stanier, C. (2017). Identifying New Directions in Database Performance Tuning. Procedia Computer Science, 121, 260–265. https://doi.org/10.1016/j.procs.2017.11.036
E005	Myalapalli, V. K. V. K. K., Totakura, T. P. T. P. P., & Geloth, S. (2015). Augmenting database performance via SQL tuning. 2015 International Conference on Energy Systems and Applications, 13–18. https://doi.org/10.1109/ICESA.2015.7503305
E006	Cincy, W. C., & Jeba, J. R. (2018). A Method of A-BAT Algorithm Based Query Optimization for Crowd Sourcing System. International Journal of Intelligent Systems and Applications, 10(3), 33–40. https://doi.org/10.5815/ijisa.2018.03.04
E007	Myalapalli, V. K., Chakravarthy, A. S. N., & Reddy, K. P. (2015). Accelerating SQL queries by unravelling performance bottlenecks in DBMS engine. 2015 International Conference on Energy Systems and Applications, 7–12. https://doi.org/10.1109/ICESA.2015.7503304
E008	Neuhaus, P., Couto, J., Wehrmann, J., Ruiz, D., & Meneguzzi, F. (2019). GADIS: A Genetic Algorithm for Database Index Selection (S). Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE, 2019-July, 39–42. https://doi.org/10.18293/SEKE2019-135
E009	Idhaim, H. A. (2019). Selecting and tuning the optimal query form of different SQL commands. International Journal of Business Information Systems, 30(1), 1. https://doi.org/10.1504/IJBIS.2019.097041
E010	Yoon, D. Y., Mozafari, B., & Brown, D. P. (2015). DBSeer: Pain-free Database Administration through Workload Intelligence. Proceedings of the VLDB Endowment, 8(12), 2036–2039. https://doi.org/10.14778/2824032.2824130
E011	Matalqa, S., & Mustafa, S. (2016). The effect of horizontal database table partitioning on query performance. International Arab Journal of Information Technology, 13(1A), 184–189.
E012	Canim, M., Mihaila, G. A., Bhattacharjee, B., Ross, K. A., & Lang, C. A. (2009). An Object Placement Advisor for DB2 Using Solid State Storage. Proc. VLDB Endow., 2(2), 1318–1329. https://doi.org/10.14778/1687553.1687557
E013	Zhong-zhuang, W., Lun-dan, D., Yun-ting, W., & Zhi-wen, H. (2012). The Optimization of Large Information Management Database Bottlenecks. In Advances in Intelligent and Soft

	Computing: Vol. 117 AISC (Issue 127 VOL. 2, pp. 197–202). https://doi.org/10.1007/978-3-642-25437-6_28
E014	Cueva, R., & Tupia, M. (2008). Double relaxation - GRASP algorithm for solve the table assignment problem on data base store devices. Proceedings - 4th International Conference on Networked Computing and Advanced Information Management, NCM 2008, 2(October), 418–421. https://doi.org/10.1109/NCM.2008.186
E015	Golubchik, L., Khanna, S., Khuller, S., Thurimella, R., & Zhu, A. (2009). Approximation algorithms for data placement on parallel disks. ACM Transactions on Algorithms, 5(4), 1–26. https://doi.org/10.1145/1597036.1597037
E016	Wu, C.-H., Huang, C.-W., & Chang, C.-Y. (2019). A data management method for databases using hybrid storage systems. ACM SIGAPP Applied Computing Review, 19(1), 34–47. https://doi.org/10.1145/3325061.3325064
E017	Schnaitter, K., & Polyzotis, N. (2012). Semi-automatic index tuning. Proceedings of the VLDB Endowment, 5(5), 478–489. https://doi.org/10.14778/2140436.2140444
E018	Alsultanny, Y. (2010). Database management and partitioning to improve database processing performance. Journal of Database Marketing & Customer Strategy Management, 17(3–4), 271–276. https://doi.org/10.1057/dbm.2010.14
E019	Kechar, M., & Nait-Bahloul, S. (2019). Bringing Together Physical Design and Fast Querying of Large Data Warehouses. Proceedings of the 4th International Conference on Big Data and Internet of Things, 1–8. https://doi.org/10.1145/3372938.3372947
E020	Raja Kumar Reddy, N., & Naidu, M. M. (2015). A genetic algorithmic approach for determining optimal secondary index set for querying on multiple relations. International Journal of Applied Engineering Research, 10(8),19195–19212.https://www.scopus.com/inward/record.uri?eid=2-s2.0-84929890949&partnerID=40&md5=773933f45766154ed8145b40b09cb3c5
E021	Szulc, I., Stencel, K., & Wiśniewski, P. (2017). Using Genetic Algorithms to Optimize Redundant Data. In Communications in Computer and Information Science (Vol. 716, pp. 165–176). https://doi.org/10.1007/978-3-319-58274-0_14
E022	Lin, L., Zhu, Y., Yue, J., Cai, Z., & Segee, B. (2011). Hot random off-loading: A hybrid storage system with dynamic data migration. IEEE International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems - Proceedings, 318–325. https://doi.org/10.1109/MASCOTS.2011.41
E023	Ou, J., Shu, J., Lu, Y., Yi, L., & Wang, W. (2014). EDM: An endurance-aware data migration scheme for load balancing in SSD storage clusters. Proceedings of the International Parallel and Distributed Processing Symposium, IPDPS, 787–796. https://doi.org/10.1109/IPDPS.2014.86
E024	Shi, H., Arumugam, R. V., Foh, C. H., & Khaing, K. K. (2013). Optimal disk storage allocation for multitier storage system. IEEE Transactions on Magnetics, 49(6), 2603–2609. https://doi.org/10.1109/TMAG.2013.2250936

Anexo 2: Plan de proyecto

1) Justificación

En la actualidad, los datos tienen un gran valor para las organizaciones, es por esta razón que las empresas han buscado la manera de almacenarlos y organizarlos de forma segura y eficiente para su uso posterior. Este objetivo se ha alcanzado gracias a la creación de los sistemas de administración de base de datos relaciones los cuales permiten almacenar, organizar, acceder y modificar los datos de manera eficiente, es por ello que hoy en día estos sistemas son considerados como la fuente principal para administrar una empresa. Sin embargo, debido al rápido desarrollo de las tecnologías de información, cada vez más sistemas de información de gran escala generan enormes cantidades de datos, haciendo que la tarea de organizarlos, almacenarlos y acceder a ellos se vuelva altamente desafiante. En ese sentido, mejorar y mantener el rendimiento del sistema de administración de base de datos es considerado como el aspecto más crítico dentro de la administración de un sistema de información, pues permite mejorar el rendimiento del mismo y hace que la empresa pueda gestionar sus datos de mejor manera y satisfacer sus necesidades de procesamiento de información.

A lo largo de los años han surgido diversas técnicas para la optimización del sistema de base de datos, muchas de ellas se enfocan en la mejora del tiempo de respuesta al momento de ingresar u obtener datos directamente de la base de datos. Una de estas técnicas es conocida como la asignación de datos, la cual consiste en distribuir de manera eficiente las tablas de base de datos en los dispositivos de almacenamiento secundario disponibles, esto resulta útil ya que permite mejorar el rendimiento de entrada/salida de disco, el cual es considerado como el recurso más limitado en un sistema de base dase de datos relacional.

Sin embargo, muchas de las herramientas de optimización ofrecidas por sistemas de bases de datos modernos no contemplan a la asignación de tablas como un método de optimización,

haciendo que esta tarea se tenga que hacer de manera manual por los administradores de bases de datos, quienes usualmente se basan en su experiencia pasada o solo en algunas variables para tomar una decisión. Además, muchos de los estudios realizados acerca de este método proponen soluciones que pueden brindar resultados que no son óptimos para la tarea de asignación de tablas a unidades de almacenamiento.

En ese sentido, lo que se busca en el presente proyecto es implementar una nueva propuesta que permita optimizar la asignación de tablas a unidades de almacenamiento, esta propuesta se basa en el uso de un algoritmo memético, el cual ha demostrado tener buenos resultados en tareas de optimización de similar complejidad (F. Neri et al., 2012), pertenece a una clase de algoritmos de alto nivel que pueden proporcionar soluciones apropiadas a problemas de optimización usando menos esfuerzo computacional que los métodos propuestos en la literatura, y además no ha sido aplicado directamente en la tarea de asignación de datos.

Los principales beneficios de la aplicación de este algoritmo en la optimización de la asignación de tablas a unidades de almacenamiento son:

- Permite optimizar el rendimiento de los sistemas de administración de base de datos relacionales, y con ello mejorar el rendimiento de los sistemas de información que los usan.
- Brinda soporte a los administradores de bases de datos en la tarea de asignación de tablas, haciendo que esta sea menos tediosa y propensa a errores, pues se realizaría de manera automática.
- Permite a las empresas aprovechar de mejor manera el potencial del hardware actual que posee, reduciendo gastos innecesarios en actualizaciones de hardware.

2) Viabilidad

Las herramientas de desarrollo planteadas para el presente proyecto de tesis tales como el

lenguaje Java, Kanban, entre otras, son conocidas por el autor, ya que fueron aplicadas a lo largo de la carrera de Ingeniería Informática y en su vida profesional. Asimismo, en internet existe suficiente documentación acerca de estas, lo cual puede contribuir en caso de que se presente alguna dificultad. Sumado a esto, se cuenta con el apoyo del asesor quien domina el tema de algoritmos metaheurísticos y temas relacionados a la administración de bases de datos.

Por otro lado, se usarán metodologías que brindan un conjunto de buenas prácticas para el desarrollo de proyectos de esta naturaleza y servirán de apoyo para cumplir los objetivos planteados, en ese sentido, se puede decir que no habrá obstáculos para el desarrollo normal del proyecto en el aspecto técnico.

Asimismo, en la sección de Cronograma del proyecto, la Tabla 23 muestra el cronograma del proyecto como una evidencia de la viabilidad temporal del mismo. En él se detalla la planificación de actividades que se llevarán a cabo durante la ejecución del proyecto de tesis. En ese sentido, el desarrollo correcto del presente proyecto se considera viable a nivel temporal.

Finalmente, para el desarrollo del presente proyecto no se requerirá de una inversión económica significativa, ya que se utilizan herramientas de libre acceso y recursos ya disponibles. En la sección de Lista de recursos, se describen los recursos requeridos para el proyecto. En ese sentido, se considera que se tienen los recursos necesarios para el desarrollo correcto del presente proyecto de tesis.

3) Alcance

El presente proyecto de tesis pertenece al área de ciencias de la computación, más específicamente a la rama de algoritmos de optimización y tiene como objetivo principal implementar un algoritmo memético para optimizar la asignación de tablas a unidades de

almacenamiento de bases de datos relacionales. Este algoritmo permitirá asignar las tablas de base de datos a los dispositivos de almacenamiento secundario disponibles, para así maximizar el rendimiento de entrada/salida de disco y mejorar el rendimiento de la base de datos. Cabe mencionar que no se considerará la asignación de otros objetos de bases de datos como, por ejemplo, los índices.

En ese sentido, en primer lugar, se procederá a realizar un análisis y planteamiento de las variables, factores, y sus respectivas relaciones más importantes a considerar al momento de realizar esta tarea. Posteriormente, se procederá a definir las estructuras de datos necesarias para implementar el algoritmo memético, luego se diseñará el algoritmo en base a la información recopilada.

A continuación, se implementará el algoritmo memético y adicionalmente, se realizará el diseño e implementación de un algoritmo GRASP encontrado en la literatura, esto con el objetivo de ser comparado con el algoritmo memético planteado y corroborar la validez de su aplicación en la tarea de asignación de tablas a unidades de almacenamiento. La comparación de algoritmos se realizará mediante un proceso de experimentación numérica sobre los resultados obtenidos, esta experimentación se llevará a cabo mediante el uso de diversas pruebas estadísticas.

Finalmente, se implementará una interfaz que facilite la ejecución de los algoritmos planteados, esta recibirá la información necesaria para correr los algoritmos y también mostrará los resultados de la ejecución, sin embargo, cabe mencionar que el alcance de este proyecto no comprende el desarrollo de un sistema de información que contenga como funcionalidad a los algoritmos implementados.

4) Limitaciones

Debido a que este es un proyecto de desarrollo de algoritmos, la principal limitante tiene que

ver con la capacidad de procesamiento de la computadora en la cual se ejecutan los mismos, por esta razón, se incluye como limitante a las características de hardware del equipo en el cual se ejecutarán los algoritmos.

5) Identificación de los riesgos del proyecto

En la **Tabla 22** se muestran los riesgos identificados junto con la probabilidad de ocurrencia, el impacto sobre el proyecto, la severidad del riesgo y el plan de contingencia en caso de que este se concrete.

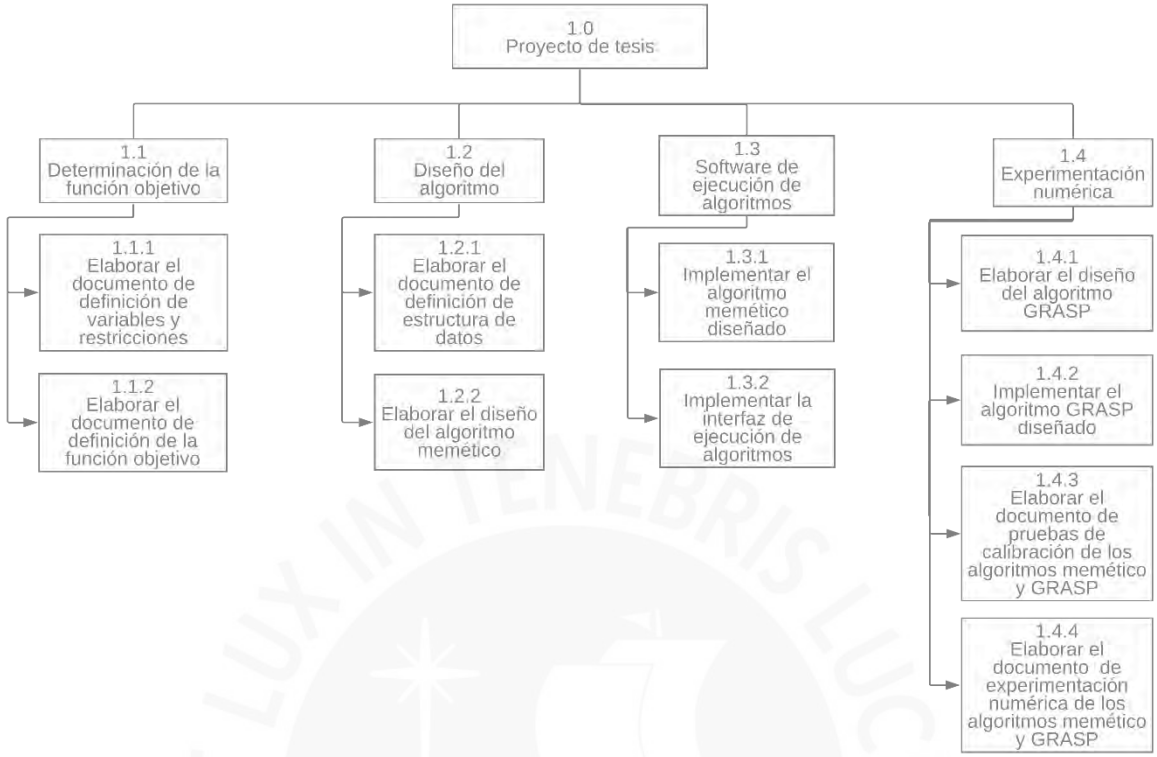
Tabla 22. *Riesgos del proyecto*

Descripción	Síntomas	P	I	S	Mitigación	Contingencia
Poco o nulo acceso a información de terceros (información acerca de bases de datos reales)	No se obtiene respuestas de las personas que brindan la información de terceros	2	2	4	Gestionar el acceso a la información de manera oportuna	En caso de que no se consiga, generar información propia de manera aleatoria
Problemas de conectividad al momento de realizas actividades programadas	Fallas constantes en la conectividad en los días previos a las actividades programadas	2	1	2	Contratar un plan celular para acceder a la reunión	Activar el plan celular y acceder a la reunión programada
Especialistas no realizan la validación de los entregables en los tiempos establecidos	No se obtiene respuesta de los especialistas en las comunicaciones previas a la fecha de entrega	2	3	6	Planificar las reuniones de revisión de manera oportuna	Solicitar ayuda de un especialista alterno para la revisión de entregables
Especialistas no se encuentran disponibles para realizar una entrevista presencial	No se obtiene una respuesta de los especialistas en las comunicaciones previas a la fecha de la entrevista, o estos cancelan la entrevista en los días previos	3	1	3	Reservar un horario para la entrevista con al menos 4 días de anticipación	Usar una herramienta como “Formularios de Google” para realizar la entrevista no presencial, y procurar enfocarse en los temas centrales.

6) Estructura de descomposición del trabajo

La estructura de descomposición del trabajo puede ser visualizada en la **Figura 17.**

Figura 17. *Estructura de descomposición del trabajo*



7) Lista de tareas

A continuación, en la **Tabla 23** se presenta la lista de tareas en base a la estructura de descomposición planteada.

Tabla 23. *Lista de tareas*

Nro.	Tarea	Duración estimada (días)	Esfuerzo asociado (horas/persona)	Costo estimado (Soles)
1	Entregable 4 de Tesis1	0	0	0
2	Realizar entrevista semiestructurada con un especialista en administración de base de datos	1	6	120
3	Definir las variables o factores más relevantes	1	6	120
4	Definir la función objetivo	1	6	120
5	Reunión de validación con el asesor y realizar ajustes	2	4	80
7	Validar la definición de las variables o factores más relevantes con un especialista	1	2	40
8	Validar la definición de la función objetivo con un especialista	1	2	40
	Subtotal	**7**	**26**	**520**
9	Definir la estructura de datos	2	12	240
10	Reunión de validación con el asesor y realizar ajustes	1	4	80
11	Validar estructura de datos con un especialista en algoritmia	1	2	40
12	Exposición 1	1	4	80
13	Diseñar el pseudocódigo del algoritmo memético	4	24	480
14	Reunión de validación con el asesor y realizar ajustes	1	4	80
15	Realizar pruebas de flujo de datos sobre el diseño del algoritmo memético	2	12	240
16	Validar pseudocódigo del algoritmo por un especialista en algoritmia	1	4	80
17	Exposición 2	1	2	40
	Subtotal	**14**	**68**	**1360**
18	Codificar el algoritmo memético	5	40	800
19	Realizar pruebas unitarias del algoritmo memético	1	6	120
20	Reunión de validación con el asesor y realizar ajustes	1	4	80
21	Realizar correcciones	1	5	100
22	Exposición 3	1	4	80
23	Codificar módulo de ejecución de los algoritmos	2	16	320
24	Realizar pruebas unitarias del módulo de ejecución de algoritmos	1	6	120

25	Reunión de validación con el asesor y realizar ajustes	1	4	80
27	Exposición 4	1	2	40
	Subtotal	**14**	**87**	**1740**
28	Avance parcial y correcciones finales	5	30	600
29	Exposición parcial	1	4	80
	Subtotal	**6**	**34**	**680**
30	Diseñar el pseudocódigo del algoritmo GRASP	4	24	480
31	Reunión de validación con el asesor y realizar ajustes	1	4	80
32	Realizar pruebas de flujo de datos sobre el diseño del algoritmo GRASP	2	12	240
33	Validar pseudocódigo del algoritmo por un especialista en algoritmia	1	4	80
34	Exposición 5	1	2	40
35	Codificar el algoritmo GRASP	5	40	800
36	Realizar pruebas unitarias del algoritmo GRASP	2	12	240
37	Reunión de validación con el asesor y realizar ajustes	1	4	80
39	Realizar pruebas de calibración de los algoritmos memético y GRASP	3	24	480
	Subtotal	**20**	**126**	**2520**
40	Realizar experimentación numérica	8	40	800
41	Exposición 6	1	4	80
	Subtotal	**9**	**44**	**880**
42	Correcciones finales	6	24	480
43	Entregable final	1	4	80
	Subtotal	**7**	**28**	**560**
	TOTAL	**77**	**413**	**8260**

8) Cronograma del proyecto

A continuación, en la **Tabla 24** se presenta el cronograma del proyecto en base a la estructura de descomposición de trabajo y la lista de tareas.

Tabla 24. *Cronograma del proyecto*

Sem.	Fechas	Tarea
1	del 29-03-	Realizar correcciones del entregable 4 de Tesis 1
		Realizar plan del proyecto para el curso de Tesis 2

	2021 al 05-04-2021	**Objetivo Específico 1: Determinar una relación adecuada entre las variables o factores más relevantes relacionados a la asignación de tablas a unidades de almacenamiento**
		Resultado 1: Definición de variables y restricciones más relevantes a tomar en cuenta para la tarea de asignación de tablas a unidades de almacenamiento
		Realizar entrevista semiestructurada con un especialista en administración de base de datos
		Identificar las variables y restricciones en base a las opiniones del especialista
		Resultado 2: Diseño de la función objetivo a ser usada en la optimización de la tarea asignación de tablas a unidades de almacenamiento
		Definir la función objetivo usando las variables y restricciones identificadas
		Reunión de validación con el asesor y realizar ajustes
		Validar la definición de las variables o factores más relevantes con un especialista
		Validar la definición de la función objetivo con el especialista
		Redactar Capítulo 4 completo del documento de tesis
2	del 06-04-2021 al 12-04-2021	Exposición 1
		Levantar observaciones de la semana 1
		Objetivo Específico 2: Adaptar el algoritmo memético de tal manera que pueda ser aplicado en la optimización de la asignación de tablas a unidades de almacenamiento
		Resultado 3: Definición de las estructuras de datos que utilizará el algoritmo memético
		Definir las estructuras principales del algoritmo memético
		Resultado 4: Diseño de un algoritmo memético adaptado al problema de asignación de tablas a unidades de almacenamiento
		Diseñar el algoritmo memético y sus principales funciones auxiliares
		Reunión de validación con el asesor y realizar ajustes
		Validar diseño completo del algoritmo con el especialista
		Redactar Capítulo 5 completo del documento de tesis
3	del 13-04-2021 al 19-04-2021	Exposición 2
		Levantar observaciones de la semana 2
		Objetivo Específico 3: Desarrollar una herramienta de software que permita la ejecución del algoritmo memético aplicado a la optimización de la asignación de tablas a unidades de almacenamiento de manera automática
		Resultado 5: Codificación del algoritmo memético adaptado al problema de asignación de tablas a unidades de almacenamiento
		Implementar el algoritmo memético ya diseñado y aprobado
		Realizar pruebas unitarias del algoritmo memético
		Reunión de validación con el asesor y realizar ajustes
		Validar codificación del algoritmo con el especialista
		Redactar Capítulo 6 del documento de tesis (solo la sección de codificación del algoritmo memético)
4	del 20-04-	Exposición 3
		Levantar observaciones de la semana 3

	2021 al 26-04-2021	**Objetivo Específico 3: Desarrollar una herramienta de software que permita la ejecución del algoritmo memético aplicado a la optimización de la asignación de tablas a unidades de almacenamiento de manera automática**
		Resultado 6: Interfaz gráfica que permita ejecutar el algoritmo memético aplicado al problema de asignación de tablas a unidades de almacenamiento
		Implementar el software de ejecución del algoritmo memético
		Realizar pruebas unitarias del software de ejecución del algoritmo memético
		Reunión de validación con el asesor y realizar ajustes
		Validar el software de ejecución con el especialista
		Redactar Capítulo 6 del documento de tesis (culminar documento agregando la sección de desarrollo de la herramienta de software de ejecución)
5	del 27-04-2021 al 03-05-2021	Exposición 4
		Levantar observaciones de la semana 4
		Objetivo Específico 4: Comparar el desempeño del algoritmo memético propuesto contra la solución más relevante encontrada en el estado del arte con respecto a la optimización de la asignación de tablas a unidades de almacenamiento con la finalidad de verificar que este brinda buenos resultados y puede ser aplicado con éxito en dicha tarea de optimización
		Resultado 7: Diseño del algoritmo GRASP adaptado al problema de asignación de tablas a unidades de almacenamiento
		Diseñar el algoritmo GRASP y sus principales funciones auxiliares
		Reunión de validación con el asesor y realizar ajustes
		Validar diseño del algoritmo con el especialista
		Redactar Capítulo 7 del documento de tesis (solo la sección del diseño del algoritmo GRASP)
6	del 04-05-2021 al 10-05-2021	Redactar el avance parcial
		Reunión de validación con el asesor y realizar ajustes
		Exposición parcial
7	del 11-05-2021 al 17-05-2021	Redactar el avance parcial
		Reunión de validación con el asesor y realizar ajustes
		Exposición parcial
8	del 18-05-2021 al 24-05-2021	Exposición 5
		Levantar observaciones del entregable parcial
		Objetivo Específico 4: Comparar el desempeño del algoritmo memético propuesto contra la solución más relevante encontrada en el estado del arte con respecto a la optimización de la asignación de tablas a unidades de almacenamiento con la finalidad de verificar que este brinda buenos resultados y puede ser aplicado con éxito en dicha tarea de optimización
		Resultado 8: Codificación del algoritmo GRASP encontrado en el estado del arte
		Implementar el algoritmo GRASP ya diseñado y aprobado
		Realizar pruebas unitarias del algoritmo GRASP
		Reunión de validación con el asesor y realizar ajustes

		Validar implementación del algoritmo con el especialista
		Redactar Capítulo 7 del documento de tesis (solo la sección de codificación del algoritmo GRASP encontrado en el estado del arte)
9	del 25-05-2021 al 30-05-2021	**Semana de exámenes parciales**
		Objetivo Específico 4: Comparar el desempeño del algoritmo memético propuesto contra la solución más relevante encontrada en el estado del arte con respecto a la optimización de la asignación de tablas a unidades de almacenamiento con la finalidad de verificar que este brinda buenos resultados y puede ser aplicado con éxito en dicha tarea de optimización
		Resultado 9: Calibración de variables que se usarán en los algoritmos memético y GRASP
		Realizar proceso de calibración de variables a usar en ambos algoritmos
		Reunión de validación con el asesor y realizar ajustes
		Validar proceso de calibración con el especialista
		Redactar Capítulo 7 del documento de tesis (solo la sección de calibración de variables)
10	del 01-06-2021 al 07-06-2021	Exposición 6
		Levantar observaciones de la semana 8
		Objetivo Específico 4: Comparar el desempeño del algoritmo memético propuesto contra la solución más relevante encontrada en el estado del arte con respecto a la optimización de la asignación de tablas a unidades de almacenamiento con la finalidad de verificar que este brinda buenos resultados y puede ser aplicado con éxito en dicha tarea de optimización
		Resultado 10: Experimentación numérica a modo de evaluación del desempeño de los algoritmos memético y GRASP
		Realizar proceso de experimentación numérica para evaluar los algoritmos
		Reunión de validación con el asesor y realizar ajustes
		Validar proceso de experimentación con el especialista
		Redactar Capítulo 7 del documento de tesis y entregable final (culminar capítulo y agregar la sección de experimentación numérica de algoritmos)
11	del 08-06-2021 al 14-06-2021	Presentación del entregable final
12	del 15-06-2021 al 28-06-2021	Revisión del jurado
13		
14	del 29-06-2021 al 12-07-2021	Levantar observaciones
15		
16	del 13-07-	Exposiciones finales

17	2021 al 26-07-2021	

9) **Lista de recursos**

- **Personas involucradas y necesidades de capacitación**

  En cuanto a las personas involucradas en el desarrollo del proyecto se tienen principalmente dos:

  - **El tesista**

    Es el alumno que elabora el proyecto de tesis con el objetivo de obtener el título universitario.

  - **El asesor**

    Es el especialista encargado de guiar al alumno tesista en el desarrollo del proyecto de tesis.

- **Materiales requeridos para el proyecto**

  No aplica.

- **Estándares utilizados en el proyecto**

  En relación a los estándares utilizados en el proyecto se tiene lo siguiente:

  - **PMBOK parte 2: El estándar para la dirección de proyectos**

    Uso de algunas buenas prácticas para la dirección de proyectos propuestas por el instituto de administración de proyectos PMI en la guía PMBOK 6ta edición.

- **Equipamiento**

  Con respecto al equipamiento necesario para el desarrollo del proyecto se tiene lo siguiente:

  - **Laptop**

    Este equipamiento está disponible para el tesista del proyecto y es necesario para la realización de todos los componentes del proyecto de tesis.

- **Herramientas requeridas**

  Con respecto a las herramientas requeridas se tiene las siguientes:

  - **IDE para el lenguaje java**

    Esta herramienta de construcción de software es necesaria para la codificación de los algoritmos propuestos en el presente proyecto de tesis.

  - **IDE para el lenguaje R**

    Esta herramienta es necesaria para realizar la experimentación numérica sobre los resultados obtenidos por los algoritmos planteados.

  - **Herramienta en la nube para control de versiones de software**

    Es necesaria una herramienta de control de versiones para mantener seguros los avances del desarrollo del proyecto de tesis y gestionar sus cambios de manera óptima.

  - **Herramienta virtual para el uso de Kanban**

    Se necesita una herramienta virtual para el uso de Kanban, ya que, esta permite aprovechar de mejor manera los beneficios de este método y compartir información con los involucrados en el proyecto de manera fácil y segura.

  - **Herramienta de comunicación (Zoom, Google Meets, WhatsApp, entre otros)**

    Se requiere de herramientas de comunicación para poder realizar las reuniones de revisión con el asesor del proyecto y las entrevistas que se planea tener con especialistas en administración de bases de datos y algoritmia.

  - **Herramienta de ofimática (Word y Excel)**

    Finalmente se requiere de herramientas de ofimática para la redacción de los documentos necesarios en el presente proyecto de tesis.

10) Costeo del proyecto

A continuación, en la **Tabla 25** se plantea un análisis de costos para el desarrollo del

presente proyecto de tesis.

Tabla 25. *Análisis de costos*

Ítem	Descripción	Unidad	Cantidad	Valor unitario (S/.)	Monto parcial (S/.)	Monto total (S/.)
1.	**Personas involucradas**	**---**	**---**	**---**	**---**	**11,360**
1.1	Estudiante 1	Horas	413	20	8260	--
1.2	Asesor 1	Horas	62	50	3100	
2.	**Bienes y equipos**	**---**	**---**	**---**	**---**	**826**
2.1	Laptop	Horas	413	2	826	--
3.	**Servicios**	**---**	**---**	**---**	**---**	**480**
3.1	Internet	Mes	4	120	480	---
	Total	**---**	**---**	**---**	**---**	**12,666**

Anexo 3: Entrevista semiestructurada acerca de la tarea de asignación de tablas

En el presente anexo se detalla una estructura general de las preguntas más relevantes realizadas en las entrevistas semiestructuradas con el especialista en administración de bases de datos. El objetivo de estas entrevistas es detectar los factores más relevantes relacionados a la asignación de tablas a unidades de almacenamiento, además, estas entrevistas permiten un mayor entendimiento de la aplicación de este método en contextos y situaciones reales.

Datos generales

- ¿Cuál es su nombre?
- ¿Por cuánto tiempo trabaja o trabajó en el rubro de la administración de bases de datos?

Experiencia con el uso de la asignación de tablas en contextos reales

- ¿En su opinión, cuáles son los aspectos que más pueden influir en el rendimiento de un sistema de base de datos?
- ¿Ha utilizado el método de asignación de tablas para poder mejorar el rendimiento de un sistema de base de datos?
- ¿Qué dificultades ha encontrado al momento de realizar una correcta asignación de tablas?
- ¿Cuánto tiempo en promedio toma realizar un proceso de asignación de tablas?

Aspectos más importantes al momento de realizar un proceso de asignación de tablas

- ¿Qué aspectos deben considerarse para realizar una apropiada asignación de tablas?
- ¿Considera que el tamaño de las tablas a asignar es un factor que debe ser considerado?
- ¿Cómo se realiza la medición de este valor y en qué unidades se mide?
- ¿Considera que la frecuencia de uso de las tablas a asignar es un factor que debe ser considerado?
- ¿Cómo se realiza la medición de este valor y en qué unidades se mide?

- ¿Considera que la capacidad de almacenamiento de los discos es un factor que debe ser considerado?
- ¿Cómo se realiza la medición de este valor y en qué unidades se mide?
- ¿Considera que el rendimiento o velocidad de procesamiento de los discos es un factor que debe ser considerado?
- ¿Cómo se realiza la medición de este valor y en qué unidades se mide?

Aspectos más específicos acerca de un posible futuro diseño de la función objetivo

- ¿Considera que el tamaño de las tablas y su frecuencia de uso son igualmente relevantes para decidir qué tabla es más importante que otra?
- ¿Cuáles son sus razones para esta respuesta?
- ¿Considera que es importante que una solución permita al administrador poder decidir en función a qué aspecto (tamaño o frecuencia de uso) quiere priorizar cada tabla?
- ¿Cuáles son sus razones para esta respuesta?

Aspectos acerca del proceso real de asignación como experiencia propia

- ¿Se suele emplear todas las variables mencionadas anteriormente en un proceso de asignación real?
- ¿De qué maneras es común realizar la asignación? ¿Cómo se usan las variables y factores al realizarla?

Anexo 4: Resultados y conclusiones de las entrevistas semiestructuradas acerca de la tarea de asignación de tablas

En el presente anexo se detallan los resultados y conclusiones obtenidos de las entrevistas realizadas con dos especialistas en administración de bases de datos. Se espera que estos sirvan de referencia para poder redactar el capítulo correspondiente. Estas entrevistas fueron realizadas en las fechas 30/03/2021, 2/04/2021 y 9/04/2021.

Datos generales de la primera entrevista

La entrevista fue realizada al especialista Rony Cueva Moscoso, quien posee 12 años de experiencia en trabajos relacionados con la administración de bases de datos

Experiencia con el uso de la asignación de tablas en contextos reales

De esta sección se tienen las siguientes conclusiones:

1. El entrevistado opina que los aspectos que más pueden influir en el rendimiento de un sistema de bases de datos son:
    a. El mal uso de los recursos disponibles
    b. La dificultad para aplicar técnicas de optimización adecuadas
2. El entrevistado opina que una de las mayores dificultades para realizar una correcta asignación de tablas es que la cantidad de tablas y unidades de almacenamiento disponibles era muy grande, lo cual provocaba que se realice un proceso de análisis con una demora de 5 días de trabajo en promedio.

Aspectos más importantes al momento de realizar un proceso de asignación de tablas

De esta sección se tienen las siguientes conclusiones:

1. El entrevistado considera que los factores más importantes a tomar en cuenta en la asignación de tablas son:
    a. El tamaño de las tablas, el cual puede ser obtenido a través de comandos SQL
    b. La frecuencia de uso de las tablas, el cual puede ser obtenido calculando el volumen de operaciones de entrada y salida que se realiza

c. La capacidad de almacenamiento de los discos (HDD o SSD), el cual puede ser obtenido con las herramientas del sistema operativo
d. La velocidad de procesamiento de los discos, el cual es brindado por el fabricante

Aspectos más específicos acerca de un posible futuro diseño de la función objetivo

De esta sección se tiene la siguiente conclusión:

1. El entrevistado considera que tanto el tamaño de las tablas, como su frecuencia de uso son igualmente relevantes para un proceso de asignación. Por lo tanto, una solución debe permitir al administrador elegir si desea priorizar tablas en base al tamaño, frecuencia de uso o considerando ambos factores en igual proporción

Aspectos acerca del proceso real de asignación como experiencia propia

De esta sección se tienen las siguientes conclusiones:

1. Usualmente no se toman en cuenta a todos los factores más importantes, tales como la frecuencia de uso, el tamaño de una tabla, el rendimiento de los discos y su capacidad de almacenamiento, al momento de realizar el proceso de asignación. Ya que, esto le suma complejidad al proceso que normalmente se realiza de forma manual por los administradores de base de datos.
2. Las técnicas más comunes empleadas para tratar de simplificar el proceso de asignación de tablas son las siguientes:
    a. Colocar todas las tablas al disco más veloz
    b. Colocar solo las tablas más usadas en el disco más veloz

    Sin embargo, estos métodos resultan no ser apropiados porque no significan una optimización real para la base de datos.

Datos generales de la segunda entrevista

La entrevista fue realizada al especialista Maisa Carranza, quien posee 8 años de experiencia en trabajos relacionados con la administración de bases de datos

Experiencia con el uso de la asignación de tablas en contextos reales

De esta sección se tienen las siguientes conclusiones:

1. El entrevistado opina que los aspectos que más pueden influir en el rendimiento de un sistema de bases de datos son:
    a. El mal uso de los recursos disponibles
    b. La dificultad para aplicar técnicas de optimización adecuadas
    c. Muchas herramientas no toman en cuenta a esta técnica de optimización
2. El entrevistado opina que una de las mayores dificultades para realizar una correcta asignación de tablas es que no existe un criterio comprobado que ayude a realizar el proceso de asignación y que, además, no se cuenta con una herramienta que facilite dicho proceso.

Aspectos más importantes al momento de realizar un proceso de asignación de tablas

De esta sección se tienen las siguientes conclusiones:

1. El entrevistado considera que los factores más importantes a tomar en cuenta en la asignación de tablas son:
    a. El tamaño de las tablas, el cual puede ser obtenido a través de consultas a los segmentos de tables e índices
    b. La frecuencia de uso de las tablas, el cual puede ser obtenido mediante métricas de las tablas
    c. La capacidad de almacenamiento de los discos (HDD o SSD), el cual puede ser obtenido revisando los datos de configuración para el almacenamiento de la base de datos
    d. La velocidad de procesamiento de los discos

Aspectos más específicos acerca de un posible futuro diseño de la función objetivo

De esta sección se tiene la siguiente conclusión:

1. El entrevistado no considera que el tamaño de las tablas y su frecuencia de uso sean igualmente relevantes para un proceso de asignación ya que, esto depende de qué es lo que se quiera priorizar (espacio o frecuencia de uso)

2. El entrevistado considera que puede ser muy útil una herramienta que permita decidir en función de qué aspecto (tamaño o frecuencia de uso) quiere priorizar cada tabla, pues esto permite gestionar de mejor manera la asignación del espacio.

Aspectos acerca del proceso real de asignación como experiencia propia

De esta sección se tienen las siguientes conclusiones:

1. Usualmente no se toman en cuenta a todos los factores más importantes, tales como la frecuencia de uso, el tamaño de una tabla, el rendimiento de los discos y su capacidad de almacenamiento, al momento de realizar el proceso de asignación.
2. Las técnicas más comunes empleadas para tratar de simplificar el proceso de asignación de tablas son las siguientes:
    a. Colocar solo las tablas más usadas en el disco más veloz
    b. Colocar las tablas más grandes en los discos con más memoria

    Sin embargo, estos métodos pueden no ser apropiados en todos los casos.

Anexo 5: Documento de pruebas realizadas sobre la función objetivo

Para visualizar las pruebas de la función objetivo puede ingresar al siguiente enlace: “pruebas realizadas”

Anexo 6: Documento de validación de la función objetivo firmado por un especialista en administración de bases de datos

Para observar la validación por un especialista dirigirse al siguiente enlace: “Documento de validación”.

Anexo 7: Documento de pruebas de flujo de datos realizadas sobre el diseño del algoritmo memético

Para visualizar el documento de pruebas realizadas puede dirigirse al siguiente enlace: "Pruebas de flujo de datos".

Anexo 8: Documento de validación del diseño del algoritmo memético firmado por un especialista

Para observar la validación por un especialista dirigirse al siguiente enlace: “Documento de validación”.

Anexo 9: Documento con el diseño de las funciones auxiliares del algoritmo memético

El diseño del algoritmo memético viene separado en diversas funciones auxiliares, cada una de las cuales cumple con una responsabilidad definida, la cual será explicada a detalle y representada mediante pseudocódigo.

- **Función para inicializar una población**

La inicialización de una población es uno de los primeros pasos en el algoritmo memético. Esta función se encarga de generar una "población" de agentes sobre la cual se puedan aplicar los procesos de evolución y mejora propios de este algoritmo. La principal característica de esta función auxiliar es que logra generar poblaciones con una alta diversidad, es decir, poblaciones en las cuales todos los agentes son únicos. El diseño de esta función se muestra en la **Figura 18**.

Figura 18. *Función para generar una población inicial*

```
Inicio generarPoblacionInicialDiversa (tamanho, P)
1:  llaves = []
2:  agentes = []
3:  Para _ en [1…tamanho]
4:     Hacer
5:       cromosoma = generarCromosoma(P)
6:       llave = generarLlave(cromosoma)
7:     Mientras (esRepetida (llave, llaves))
8:     agregar (llave, llaves)
9:     agregar (cromosoma, agentes)
10:
11: pob = Poblacion (tamanho, agentes)
12: Retornar pob
Fin generarPoblacionInicialDiversa
```

Fuente: Elaboración propia

Donde:

- Los parámetros "tamanho" y "P" vienen a ser el tamaño de la población que se quiere generar y la estructura **Problema** (que contiene los datos necesarios para crear un cromosoma), respectivamente.

- Línea 1: se inicializa una lista llamada “llaves” como vacía
- Línea 2: se inicializa una lista llamada “agentes” como vacía
- Línea 3: se inicia un bucle en el cual se itera una cantidad de veces equivalente al tamaño que va a tener la población
    - Línea 4: se inicia un nuevo bucle que solo termina si el cromosoma generado no ha sido repetido anteriormente
        - Línea 5: se genera un nuevo cromosoma, tomando como parámetro al problema
        - Línea 6: se genera una llave única que identifica al cromosoma y se guarda por separado en una lista
        - Línea 7: se comprueba que esta llave no haya sido repetida con anterioridad, en caso sea repetida se continúa iterando hasta generar cromosomas completamente nuevos
    - Línea 8 y 9: una vez que se genera el cromosoma nuevo, se almacena su llave única en la lista “llaves” para evitar repeticiones en la siguiente iteración, además este nuevo cromosoma se almacena en la lista “agentes”.
- Línea 11 y 12: una vez que se generaron todos los cromosomas, se crea la población con ellos y se retorna como resultado.

- **Función para generar un cromosoma**

Esta función tiene como objetivo crear “agentes” o “cromosomas” de manera que puedan ser incluidos como parte de una población. En la creación de nuevos cromosomas se asegura: (i) que los nuevos cromosomas generados sean válidos, es decir, que las tablas asignadas no sobrepasen la capacidad del disco que las almacena y (ii) que los nuevos cromosomas sean generados aleatoriamente, ya que es altamente recomendable tener un grado significativo de aleatoriedad en la población (Luke et al., 2011). El diseño de esta función se muestra en la **Figura 19**.

Figura 19. *Función para generar un cromosoma*

```
Inicio generarCromosoma (P)
1:  nTablas, nDiscos = P
2:  genes = []
3:  Hacer
4:     Para _ en [0 … nTablas-1]
5:        pos = generarPosicionAleatoriaSinRepetir ()
6:        genes[pos] = obtenerDiscoAleatorio (nDiscos)
7:  Mientras (sobrepasaCapacidad (genes, P))
8:
9:  Retornar Cromosoma(genes)
Fin generarCromosoma
```

Fuente: Elaboración propia

Donde:

- El parámetro “P” viene a ser la estructura **Problema** (que contiene los datos necesarios para crear un cromosoma).
- Línea 1: se obtiene el número de tablas y número de discos que usará el algoritmo.
- Línea 2: se inicializa una lista llamada “genes” como vacía
- Línea 3: se inicia un bucle en el cual se itera hasta tener una lista de genes que no sobrepase la capacidad de los discos
    - Línea 4: se inicia un nuevo bucle que itera una cantidad de veces igual al número de tablas que se pretende asignar
        - Línea 5: se genera una posición aleatoria no repetida dentro de la lista de genes
        - Línea 6: se obtiene el índice de un disco aleatorio y se coloca en la posición generada anteriormente
- Línea 7: se comprueba que la lista de genes generada no sobrepasa la capacidad de los discos, en caso se sobrepase la capacidad se dispone a descartar la solución y a continuar buscando una nueva.
- Línea 8: una vez que generada la lista de genes, se crea el cromosoma con este y se retorna como resultado.

- **Función de evaluación de la factibilidad de un cromosoma**

Al momento de generar aleatoriamente una población inicial pueden obtenerse soluciones infactibles debido a que alguna de las unidades de almacenamiento sobrepasa su capacidad disponible, por ello, en esta sección se propone una función que permita evaluar si el agente creado es válido y puede ser considerado para formar parte de la población. A continuación, en la **Figura 20** se muestra el diseño de esta función.

Figura 20. *Función para evaluar la factibilidad de un cromosoma*

```
Inicio sobrepasaCapacidad (genes, P)
1:  conjuntoGenes = convertirAConjunto(genes)
2:  Para gen en conjuntoGenes:
3:     disco = obtenerDisco (gen, P)
4:     tamanhoTotal = 0
5:     Para i en posicionesDe (gen, genes)
6:        tabla = obtenerTabla (i, P)
7:        tamanhoTotal = tamanhoTotal + obtenerTamanho(tabla)
8:     Si obtenerCapacidad(disco) <= tamanhoTotal
9:        Retornar verdadero
10: Retornar falso
Fin sobrepasaCapacidad
```

Fuente: Elaboración propia

Donde:

- Los parámetros “genes” y “P” vienen a ser la lista de genes generados aleatoriamente y la estructura **Problema** (que contiene los datos necesarios para evaluar las capacidades de los discos y los tamaños de las tablas), respectivamente
- Línea 1: se obtiene un conjunto de identificadores de los discos usados
- Línea 2: se inicia un bucle en el cual se itera sobre cada elemento del conjunto de genes
    - Línea 3: se obtiene el disco usando el identificador contenido en “gen”
    - Línea 4: se inicializa el tamaño total en cero, esta variable servirá para acumular el tamaño de las tablas asignadas al disco extraído

- Línea 5: se inicia un nuevo bucle en el que se pretende evaluar cada una de las tablas que han sido asignadas al disco en cuestión, por lo cual se obtiene las posiciones del gen en la lista de genes
  - Línea 6: se obtiene una tabla contenida en el disco evaluado
  - Línea 7: se acumula el tamaño de esta tabla en la variable “tamaño total”
- Línea 8 y 9: una vez que se acumula el tamaño de todas las tablas contenidas en el disco, se procede a comparar la capacidad del disco versus este tamaño, en caso se sobrepase la capacidad se retorna “verdadero”

- Línea 10: si luego de haber evaluado todos los discos se determina que ninguno ha sobrepasado su capacidad, se procede a retornar “falso”

- **Función para calcular la aptitud de una población**

Otra función auxiliar importante usada en el algoritmo memético es la evaluación de la aptitud de una población, la cual tiene como objetivo calcular el puntaje de aptitud que posee cada agente dentro de la población. Este valor es necesario porque permite priorizar a los “mejores” agentes. En la **Figura 21** se muestra el diseño de esta función.

Figura 21. *Función para calcular la aptitud de una población*

```
Inicio evaluarAptitudDeUnaPoblacion (poblacion, P)
1:  Para agente en poblacion:
2:     agregarPuntajeDeAptitud (agente, calcularAptitud (agente, P))
Fin evaluarAptitudDeUnaPoblacion
```

Fuente: Elaboración propia

Donde:

- Los parámetros población y P vienen a ser las estructuras **Población** (que contiene la lista de cromosomas generados aleatoriamente) y **Problema** (que contiene los datos necesarios para evaluar la aptitud del cromosoma), respectivamente.
- Línea 1: se inicia un bucle en el que se itera en cada elemento de la población
    - Línea 2: se evalúa la aptitud del elemento o "agente" y esta es agregada como uno de sus atributos

- **Función para calcular la aptitud de un cromosoma o agente**

La función de aptitud es la representación del cálculo de la función objetivo definida en el capítulo anterior, esta función permitirá evaluar las bondades de un cromosoma y saber qué tan buena es la solución presente en este. En la **Figura 22** se muestra el diseño de esta función.

Figura 22. *Función para calcular la aptitud de un cromosoma*

```
Inicio calcularAptitud (C, P)
1:  aptitudTotal = 0
2:  listaPenaliz = obtenerListaDePenalizacion (C, P)
3:  Para indice en [0…tamanho(C)]:
4:    gen = C[indice]
5:    tabla = obtenerTabla (P, indice)
6:    disco = obtenerDisco (P, gen)
7:    puntPriorTabla = calcularPuntajePriorTabla(tabla)
8:    penaliz = listaPenaliz[gen]
9:    aptitudIndiv = calcularAptitudIndividual (
                      puntPriorTabla,
                      obtenerRendimiento(disco),
                      penaliz)
10:   aptitudTotal = aptitudTotal + aptitudIndiv
11: retornar aptitudTotal
Fin calcularAptitud
```

Fuente: Elaboración propia

Donde:

- Los parámetros C y P vienen a ser las estructuras **Cromosoma** (que contiene los datos de la asignación realizada) y **Problema** (que contiene los datos necesarios para evaluar la aptitud del cromosoma), respectivamente.
- Línea 1: se inicializa en cero al puntaje de aptitud total del cromosoma C

- Línea 2: se calcula por adelantado la penalización individual de cada asignación y se almacenan en una lista
- Línea 3: se inicia un bucle en el cual se itera en los índices del cromosoma C
    - Línea 4: se extrae el gen del cromosoma
    - Línea 5 y 6: se obtiene las estructuras tabla y disco haciendo uso del índice y gen del cromosoma C
    - Línea 7: se calcula el puntaje de prioridad neto que tiene la tabla extraída
    - Línea 8: se obtiene la penalización pre calculada
    - Línea 9 y 10: se calcula el puntaje de aptitud individual del gen evaluado y este es acumulado en la variable fitness del cromosoma
- Línea 11: una vez que se procesaron todos los genes del cromosoma, se retorna el puntaje de aptitud total calculado

- **Función para la selección de agentes**

Una vez que se tiene a toda la población evaluada según su aptitud, se procede a iniciar con los procesos de "evolución" o mejora. La selección de agentes es una función que se encarga de extraer agentes de una población, estos son normalmente llamados "padres", ya que son agentes que fueron elegidos para dar origen a nuevas soluciones. Existen diversos métodos para poder realizar una selección de agentes tales como la Selección por Torneo, Selección por Ruleta, Selección Aleatoria, entre otras. En el presente proyecto de tesis se utiliza el método de Selección por Ruleta debido a que este brinda mayores posibilidades de conservar las mejores soluciones durante el proceso de evolución de una población. En la **Figura 23** se muestra el diseño de esta función.

Figura 23. *Función para seleccionar un agente de una población*

```
Inicio Seleccionar (poblacion, aptAcumulada)
1:  tamanho = obtenerTamanho(poblacion)
2:  limInferior = 0
3:  limSuperior = obtenerAptAcumulada (aptAcumulada, tamanho – 1)
4:  valor = elegirNumeroAleatorioEntre (limInferior, limSuperior)
5:  indice = busquedaBinaria (aptAcumulada, valor, 0, tamanho - 1)
6:  Retornar obtenerAgente (poblacion, indice)
Fin seleccionar
```

Fuente: Elaboración propia

Donde:

- Los parámetros población y aptAcumulada vienen a ser la estructura **Población** (que contiene los agentes) y una lista en la que cada elemento consiste en la aptitud acumulada desde el primer agente hasta el agente actual, respectivamente.
- Línea 1: se obtiene el tamaño de la población
- Línea 2 y 3: se guarda el límite inferior como 0 y al límite superior como el último elemento de la lista de aptitudes acumuladas (el cual viene a ser la suma de todas las aptitudes de la población)
- Línea 4: se elige un valor aleatorio entre el límite superior e inferior, este puede ser un valor decimal
- Línea 5: se usa un algoritmo de búsqueda binaria para buscar en qué índice se encuentra el elemento aleatorio elegido (como la búsqueda se realiza en la lista acumulada, aquel elemento que posea un puntaje de aptitud más alto tendrá más probabilidades de ser elegido)
- Línea 6: se retorna como resultado al agente que fue elegido

Asimismo, en la **Figura 24** se presenta un ejemplo gráfico del comportamiento de la función de selección, en el cual se puede notar cada una de las fases descritas anteriormente.

Figura 24. *Ejemplo de aplicación del algoritmo de selección*

Fuente: Elaboración propia

- **Función para el cruce de agentes**

La función de cruce o cruzamiento es una operación que permite combinar la información de dos agentes seleccionados, también llamados “padres”, para crear nuevos agentes, los cuales serán una mezcla de las características de sus “padres”. Existen diversos métodos para realizar esta operación tales como el Cruzamiento en Un Punto, Cruzamiento en Dos Puntos, Cruzamiento Uniforme, entre otros. En el presente proyecto de tesis se utiliza el método de Cruzamiento Uniforme debido a que, a diferencia de los métodos de cruzamiento en Un Punto y Dos Puntos, este método no tiende a romper constantemente con las buenas

soluciones que dependen de la combinación de varios genes, y además trata a los genes de manera más “justa” al cruzar cada punto independientemente uno del otro (Luke et al., 2011). En la **Figura 25** se muestra el diseño de esta función.

Figura 25. *Función de cruce de agentes*

```
Inicio reproducir (padreU, padreD)
1:  prob = 0.5
2:  hijos, genesU, genesD = [], obtenerGenes (padreU, padreD)
3:  tamanho = obtenerTamanho (padreU)
4:  Para i en [0…tamanho]:
5:     Si prob > numeroAleatorioEntre (0, 1):
6:        intercambiar (genesU, genesD, i)
7:  hijoU = Cromosoma(genesU)
8:  hijoD = Cromosoma(genesD)
9:  hijos = [hijoU, hijoD]
10: Retornar Descendencia(hijos)
Fin reproducir
```

Fuente: Elaboración propia

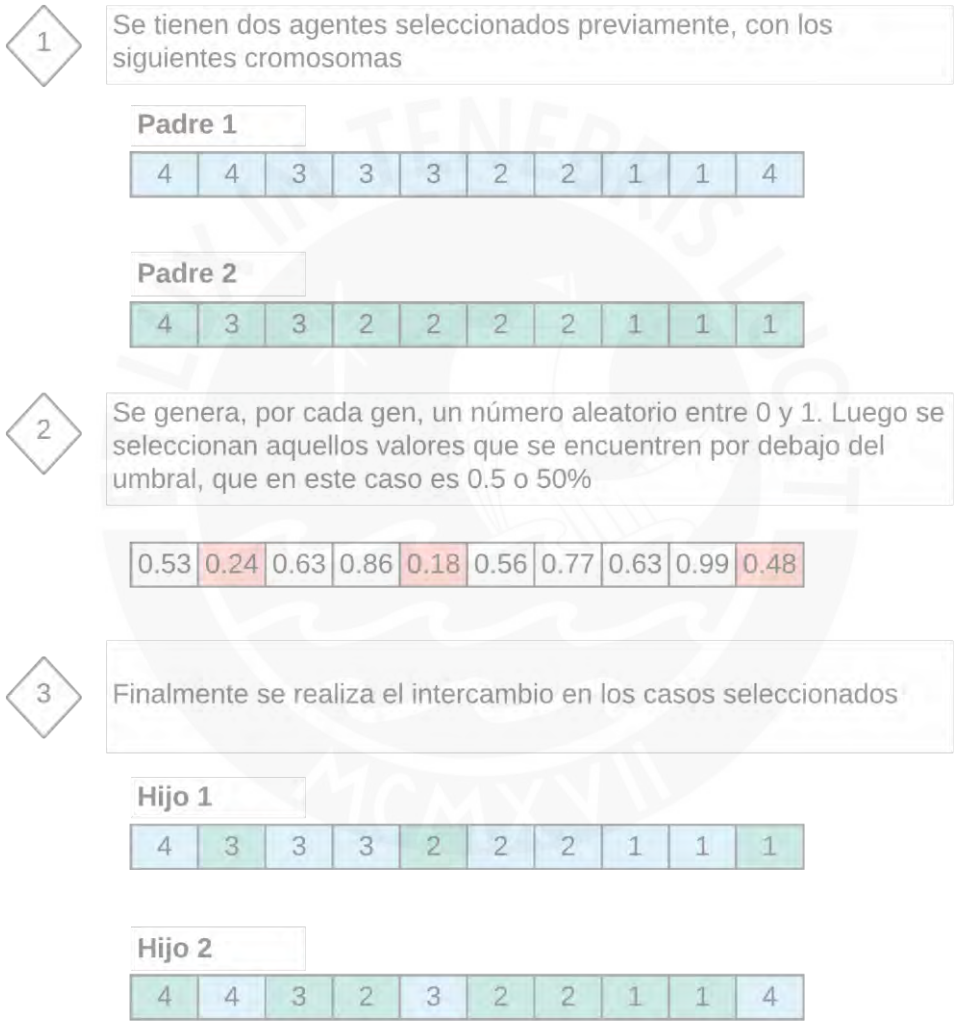
Donde:

- Los parámetros padreU y padreD vienen a ser elementos que fueron elegidos de la población mediante el método de selección.
- Línea 1: se inicializa la probabilidad de aplicar un intercambio de genes en 50%
- Línea 2: se inicializa una lista llamada “hijos” como vacía y se almacenan los genes de los cromosomas padreU y padreD
- Línea 3: se obtienen el tamaño de la lista de genes del padreU, el cual se asume igual que el del padreD
- Línea 4: se inicia un bucle en el cual se itera sobre todos los índices del cromosoma
    - Línea 5 y 6: se genera un numero aleatorio entre 0 y 1, y se compara este con la probabilidad, en caso de que el valor sea menor que la variable “prob” se procede a intercambiar los genes en el índice el cromosoma actual.
- Línea 7, 8 y 9: una vez que se realizaron todos los intercambios, se procede a generar la lista de “hijos” en base a los genes que han sido modificados

- Línea 10: finalmente se retorna la lista de agentes generados en una estructura llamada “Descendencia”

Asimismo, en la **Figura 26** se presenta un ejemplo gráfico del comportamiento de la función de cruce, en la cual se puede notar las fases principales de este procedimiento.

Figura 26. *Ejemplo de aplicación del algoritmo de cruce*



Fuente: Elaboración propia

- **Función de casamiento de agentes**

La función de casamiento consolida a la función de selección y a la función de reproducción descritas anteriormente. Esta permite aplicar dichas funciones de manera repetida sobre una

población de agentes, con el objetivo de generar una nueva población con información nueva y relevante que permita encontrar una mejor solución. El diseño de esta función puede visualizarse en la **Figura 27**.

Figura 27. *Función de casamiento de agentes*

```
Inicio casamiento (poblacion, tasa)
1:  tam = obtenerTamanho(poblacion)
2:  nroCasamientos = tam * tasa
3:  aptAcumulada = obtenerAptitudAcumulada(poblacion)
4:  descendencia = []
5:  Para i en [0…nroCasamientos]:
6:     padreU = seleccionar (poblacion, aptAcumulada)
7:     padreD = seleccionar (poblacion, aptAcumulada)
8:     hijos = reproducir (padreU, padreD)
9:     agregar (hijos, descendencia)
10: nTam = obtenerTamanho(descendencia)
11: Retornar Poblacion (descendencia, nTam)
Fin casamiento
```

Fuente: Elaboración propia

Donde:

- Los parámetros población y tasa vienen a ser la estructura **Población** (que contiene los agentes) y un valor entre 0 y 1 que permite calcular cuántas operaciones de casamiento se realizará
- Línea 1 y 2: se obtiene el tamaño de la población y se calcula el número de casamientos a aplicar
- Línea 3 y 4: se obtiene la lista de aptitudes acumuladas y se inicializa una lista vacía de agentes llamada descendencia
- Línea 5: se inicia un bucle que se repetirá una cantidad de veces igual al número de casamientos que se desea realizar
    - Línea 6 y 7: se seleccionan dos agentes de la población haciendo uso de la función seleccionar
    - Línea 8: se usa la función reproducir para obtener dos nuevas soluciones

    - Línea 9: se incorporan las nuevas soluciones generadas a lista de agentes llamada descendencia
- Línea 10: una vez finalizado el proceso de casamiento, se procede a calcular el tamaño de la lista de agentes generada
- Línea 11: se genera una estructura “Población” con la lista de agentes y el tamaño calculado en el paso anterior, devolviendo una nueva población con características distintas y, probablemente, mejoradas en forma de respuesta

- **Funciones de mutación de agentes**

Como se mencionó anteriormente, el algoritmo memético tiene la particularidad de que realiza una mejora o “evolución” iterativa de una población haciendo uso de diversos operadores tales como la selección y cruce de agentes, sin embargo, es posible que, al realizar muchas veces los procesos de cruce y selección, la población converja, es decir, que los agentes de la población lleguen a ser muy similares entre sí (Luke et al., 2011). Para tratar de solucionar este problema se tiene a la función de mutación de genes, la cual se encarga de modificar partes de los agentes de la población de manera aleatoria para así tener una mayor diversidad y que las soluciones puedan explorar otros espacios de búsqueda. Las estrategias consideradas en el presente proyecto de tesis fueron la mutación por aleatorización de genes y la mutación por intercambio de genes, finalmente se eligió la mutación por aleatorización debido a que esta trata a cada gen del cromosoma de manera individual y es considerado como un método adecuado cuando se tiene una representación lineal entera para un cromosoma (Luke et al., 2011), tal como la que se propone en el presente proyecto de tesis. Asimismo, algunos aspectos a resaltar tienen que ver con los parámetros “tasa de mutación” y “probabilidad de mutación”, los cuales sirven para calcular la cantidad de veces que se ejecutará el proceso de mutación sobre la población y la probabilidad de ocurrencia de la mutación sobre los genes de un agente, respectivamente. En la **Figura 28** se muestra el diseño de la función aplicada a una población y en la **Figura 29** se muestra el diseño de esta función aplicada a un agente individual.

Figura 28. *Función de mutación de una población*

```
Inicio mutarPoblacion (Pob, Prob, tasaMutac, probMutac)
1:  tamanhoPob = obtenerTamanho (Pob)
2:  nMutaciones = tamanhoPob * tasaMutac
3:  agentes = Pob.agentes
4:  Para _ en [0…nMutaciones]:
5:     iAg = numeroAleatorioEntre (0, tamanhoPob)
6:     agente = agentes[iAg]
7:     mutar (agente, Prob, probMutac)
Fin mutarPoblacion
```

Fuente: Elaboración propia

Donde:

- Los parámetros son los siguientes
  - Pob: la estructura **Población** (que contiene los agentes a mutar)
  - Prob: la estructura **Problema** (que contiene toda la información acerca de las tablas y discos)
  - tasaMutac: la tasa de mutación que permite calcular el número de mutaciones a realizar
  - probMutac: la probabilidad de mutación que permite definir la probabilidad de ocurrencia de una mutación
- Línea 1: se obtiene el tamaño de la población
- Línea 2: se calcula el número de mutaciones a realizar
- Línea 3: se separa previamente la lista de agentes en una variable
- Línea 4: se inicia un bucle que itera una cantidad de veces igual al número de iteraciones que se quiere realizar
  - Línea 5 y 6: se obtiene un agente aleatorio
  - Línea 7: se realiza el proceso de mutación del agente

Figura 29. *Función de mutación de un agente*

```
Inicio mutarCromosoma (A, P, probMutacion)
1:  nDiscos = obtenerNroDeDiscos(P)
2:  genes = obtenerGenes(A)
3:  Para i en [0…obtenerTamanho(genes)]:
4:     Si probMutacion >= numeroAleatorioEntre (0, 1):
5:        dAleatorio = numeroAleatorioEntre (0, nDiscos)
6:        agregarEn (genes, i, dAleatorio)
7:  reemplazarGenes (A, genes)
Fin mutarCromosoma
```

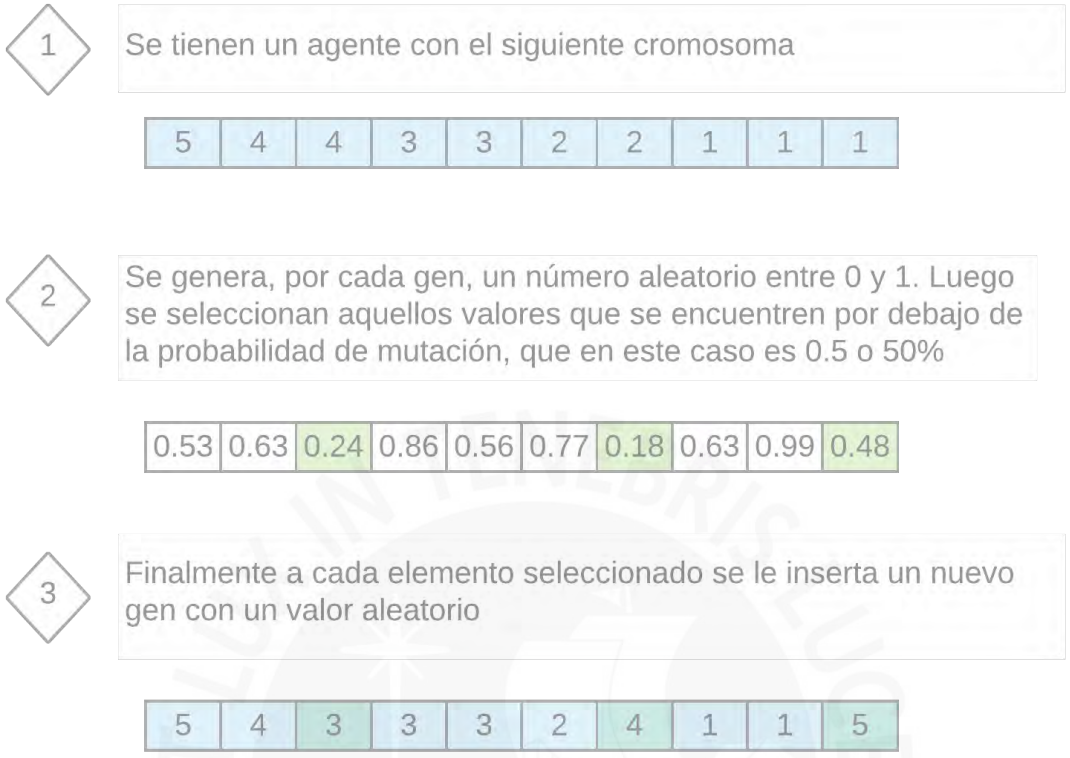
Fuente: Elaboración propia

Donde:

- Los parámetros son los siguientes
    - A: agente que se va a mutar
    - P: la estructura **Problema** (que contiene toda la información acerca de las tablas y discos)
    - probMutacion: la probabilidad de mutación que permite definir la probabilidad de ocurrencia de una mutación
- Línea 1: se obtiene el número de discos
- Línea 2: se obtiene la lista de genes del agente a mutar
- Línea 3: se inicia un bucle que itera una cantidad de veces igual al tamaño de la lista de genes del agente a mutar
    - Línea 4: se evalúa si al generar un número aleatorio, este llega a ser menor que la probabilidad de mutación
        - Línea 5 y 6: en caso de que se cumpla la condición se procede a reemplazar un gen de la lista de genes, usando un valor aleatorio
- Línea 7: finalmente los genes del agente son reemplazados por los genes que fueron mutados

Asimismo, en la **Figura 30** se presenta un ejemplo gráfico del comportamiento de la función de mutación, en la cual se puede notar las fases principales de este procedimiento.

Figura 30. *Ejemplo de aplicación del algoritmo de mutación*



Fuente: Elaboración propia

- **Estrategia de reemplazo de agentes**

Esta estrategia permite determinar qué agentes de los que se generaron mediante las funciones de selección, cruzamiento y mutación, ingresarán a la nueva población o también llamada “nueva generación”. Un método posible consiste en reemplazar por completo la población antigua por la nueva generación, sin embargo, este método tiene la desventaja de que se pierden las mejores soluciones existentes en la población anterior, en ese sentido, se prefiere elegir otro enfoque: El Elitismo. La estrategia de elitismo permite que se pueda evitar reemplazar a los mejores agentes de la población, conservando sus características para que estas puedan ser usadas en las nuevas generaciones. El detalle del diseño de esta función se puede visualizar en la **Figura 31**.

Figura 31. *Función para realizar el reemplazo usando Elitismo*

```
Inicio reemplazo (Pob, Des)
1:  tamanhoFinal = obtenerTamanho (Pob)
2:  agentesFinal = []
3:  agentesMezclados = mezclarAgentes (Pob, Des)
4:  ordenar(agentesMezclados)
5:  contElem = 0
6:  Mientras contElem < tamanhoFinal:
7:     agregar (agentesFinal, agentesMezclados[contElem])
8:     contElem = contElem + 1
9:  Retornar Poblacion (agentesFinal, obtenerTamanho(agentesFinal))
Fin reemplazo
```

Fuente: Elaboración propia

Donde:

- Los parámetros son los siguientes
    - Pob: estructura **Población**, la cual contiene la población original
    - Des: estructura **Población**, la cual contiene los agentes pertenecientes a la nueva generación producto de las operaciones realizadas anteriormente
- Línea 1, 2: se obtiene el tamaño de la población y se inicia una lista de agentes como vacía, la cual servirá para albergar a los agentes finalmente seleccionados
- Línea 3: se mezclan los agentes de las dos poblaciones (Pob y Des) en una misma lista de agentes
- Línea 4: se ordena la lista de agentes de manera descendente según su puntaje de aptitud
- Línea 5: se inicializa el contador que determina la cantidad de elementos que se va a seleccionar de la lista mezclada
- Línea 6: se inicia un bucle que acaba cuando el contador alcance el tamaño de la población original
    - Línea 7: se agrega a la lista de “agentesFinal” el agente que corresponde al índice contElem, es decir que se agregan solo los mejores agentes a esta nueva lista
    - Línea 8: se actualiza el contador

- Línea 9: una vez finalizada la selección se procede a generar una nueva población con los agentes seleccionados y se retorna en forma de resultado

- **Estrategia de parada**

La estrategia de parada permite especificar los criterios que harán que la ejecución del algoritmo memético se detenga. En el presente proyecto de tesis, la estrategia de parada se compone de tres criterios:

1. Detener la ejecución sí ya se ha alcanzado el número de N generaciones sin mejora, siendo N un número brindado por el modelador del algoritmo.
2. Detener la ejecución luego de un número M de iteraciones.
3. Detener la ejecución luego de una cierta cantidad de tiempo.

Anexo 10: Diseño de los algoritmos de búsqueda local implementados

- **Algoritmo de Enfriamiento Simulado**

  Es un método probabilístico que se inspira en la termodinámica y tiene la ventaja de que puede evitar los óptimos locales (Thainiam, 2019). Esto lo logra usando un criterio probabilístico basado en dos valores: el "delta" que es la diferencia entre las aptitudes de la solución "actual" y la solución candidata, y la "temperatura", un valor inicialmente alto que disminuye conforme aumenta las iteraciones mediante un proceso de "enfriamiento", el cual tiene la siguiente fórmula:

  $$nuevaTemp = \frac{temperatura}{(1 + log(nIter))}$$

  Donde "temperatura" es la temperatura que se desea "enfriar" y "nIter" es el número de la iteración en la que se encuentra el algoritmo. Asimismo, en cada iteración, el algoritmo "visita" un conjunto de vecinos y para decidir si va a aceptar alguno de ellos toma en cuenta dos criterios: (i) probabilidad de aceptación y (ii) mejoría de la calidad de la solución "actual". Si se cumple el segundo criterio, es decir, la solución "actual" se ve mejorada, la nueva solución se acepta automáticamente. En cambio, si la nueva solución generada, es peor que la "actual", existe una probabilidad de que, aun así, esta sea aceptada como la nueva solución "actual". El cálculo de esta probabilidad se realiza siguiendo la siguiente fórmula:

  $$P_{aceptación} = e^{\frac{-delta}{T}}$$

  Donde "delta" es la diferencia entre la solución actual y la solución nueva y "T" es el valor de la temperatura. Entonces, de esta fórmula se puede concluir lo siguiente:

  i. Si la temperatura es alta, entonces el valor de la probabilidad de aceptación también, por ende, al inicio del algoritmo habrá más posibilidades de aceptar soluciones peores, la cual va bajando conforme el algoritmo avanza.
  ii. Si el delta es pequeño, entonces hay una mayor probabilidad de aceptación, por lo tanto, es más probable que se acepte una solución peor, lo cual quiere decir

que mientras más cercanos estén en aptitud la solución actual y la solución nueva, habrá una mayor probabilidad de aceptar una solución peor.

Estos criterios de aceptación de soluciones peores, ayudan a que el algoritmo no caiga en óptimos locales y pueda explorar nuevos espacios de soluciones. En la **Figura 32** puede visualizarse el diseño de este algoritmo.

Figura 32. *Diseño del algoritmo de enfriamiento simulado*

```
Inicio enfriamientoSimulado (P, sInicial, param)
1:  maxIter, tempInicial, maxVecinos = param
2:  maxExitos = maxVecinos * tasaExitos
3:  nIter, nVecinos, nExitos = 0
4:  tempActual = tempInicial
5:  sActual, sMejor = sInicial
6:  Mientras nIter < maxIter Hacer:
7:     Mientras nExitos < maxExitos Y nVecinos < maxVecinos Hacer
8:        sNueva = generarVecino(sActual)
9:        deltaE = aptitud(sActual) – aptitud(sNueva)
10:       aleatorio = generarAleatorioEntre (0, 1)
11:       Si aleatorio < exp(-deltaE/tempActual) O deltaE < 0
12:          sActual = sNueva
13:          nExitos + 1
14:       nVecinos + 1
15:    nExitos, nVecinos = 0
16:    tempActual = enfriar (tempInicial, nIter)
17:    Si aptitud(sActual) > aptitud(sMejor):
18:       sMejor = sActual
19:    nIter + 1
20: Retornar sMejor
Fin enfriamientoSimulado
```

Donde:

- Línea 1: se obtienen los parámetros que definen el máximo número de iteraciones, la “temperatura inicial” y la cantidad de vecinos a explorar
- Línea 2: se calcula el máximo número de “éxitos” que se permitirá, es decir el número máximo de veces que se podrá actualizar la solución actual por una nueva en cada iteración
- Línea 3-5: se inicializan los contadores, la temperatura actual, la solución actual y la mejor solución antes de iniciar la búsqueda
- Línea 6: se inicia el bucle que solo finaliza si se alcanza el número máximo de iteraciones

permitidas

- Línea 7: se inicia el bucle que acaba si se alcanza el número máximo de “éxitos” o reemplazos de la solución actual, o si se alcanza el número máximo de vecinos explorados
  - Línea 8: se genera una solución vecina en base a la solución actual, aplicando un proceso simple de intercambio de elementos dentro de la solución
  - Línea 9: se obtiene el delta entre las aptitudes de la solución actual y la nueva
  - Línea 10: se genera un número aleatorio entre 0 y 1
  - Línea 11-13: si el número generado es menor que el exponencial del negativo de deltaE dividido entre la temperatura actual, o si el deltaE resulta ser negativo, se procese a actualizar la solución actual y reemplazarla por la solución nueva. Este criterio asegura que, si la nueva solución es peor que la actual, existirá una probabilidad de tomar esta solución de menor calidad, pero, si la nueva solución es mejor que la actual, se acepta automáticamente.
  - Línea 14: luego de ello se aumenta el número de vecinos visitados
- Línea 15: una vez que se termina de explorar a los vecinos, se reinician los contadores de éxitos y de vecinos visitados
- Línea 16: luego se aplica el proceso de “enfriamiento” de la temperatura inicial
- Línea 17-19: se evalúa si la nueva solución actual, es mejor que la mejor solución, si es así, esta es colocada como la nueva mejor. Luego se aumenta el contador de iteraciones.

- Línea 20: cuando se alcanza el número máximo de iteraciones, se retorna la mejor solución que se pudo encontrar
- **Algoritmo de Búsqueda Local Iterada**

  Es un método que construye una secuencia de soluciones aplicando, de manera iterativa,

un algoritmo de búsqueda local y una estrategia de perturbación (Thainiam, 2019). Para este caso, la estrategia de perturbación consiste en elegir aleatoriamente un subconjunto de elementos de una solución y revolverlos aleatoriamente. Para lograr esto se usa el "grado de perturbación", un valor que define la cantidad de elementos que tendrá el subconjunto a revolver. El cálculo de este grado de perturbación se realiza como "n/12" donde "n" es la longitud de la solución a analizar, este valor es elegido en base a los experimentos realizados por O. Martin y otros en la investigación titulada en español como "Búsqueda local iterada: marco y aplicaciones", en la cual se prueban diversos grados de perturbación y se muestra que "n/12" es un valor adecuado. Asimismo, se eligió al algoritmo de enfriamiento simulado para ser usado como algoritmo de búsqueda local. En la **Figura 33** puede visualizarse un ejemplo del proceso de perturbación y en la **Figura 34** el diseño del algoritmo de búsqueda local completo.

Figura 33. *Ejemplo de aplicación del proceso de perturbación de soluciones*

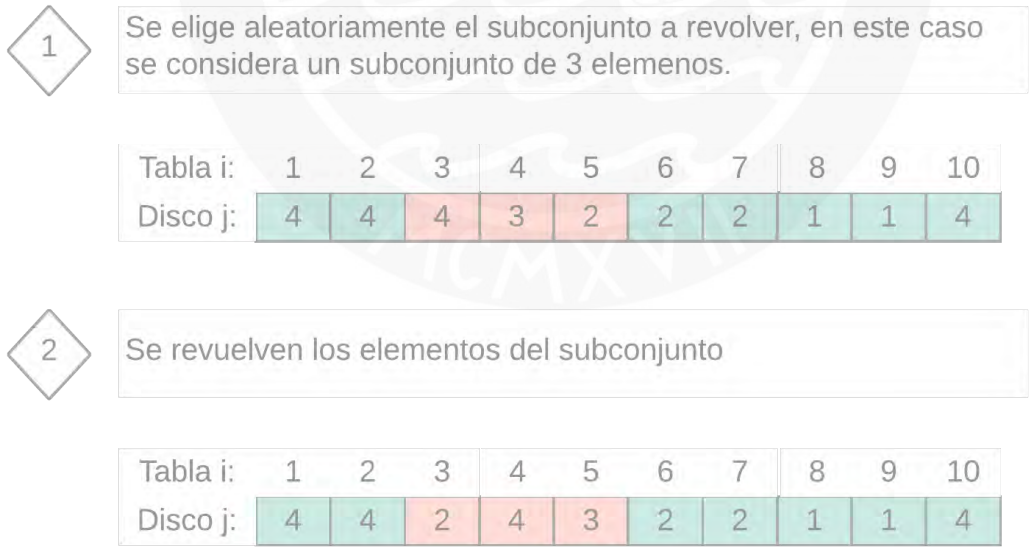


Figura 34. *Diseño del algoritmo de búsqueda local iterada*

```
Inicio busquedaLocalIterada (sInicial, param)
1:  maxIter = param
2:  gradoPerturb = obtenerGradoDePerturb(sInicial)
3:  nIter = 0
4:  sActual = busquedaLocal (sInicial, param)
5:  sMejor = sActual
6:  Mientras nIter < maxIter Hacer:
7:     aplicarPerturbacion (sActual, gradoPerturb)
8:     sNueva = busquedaLocal (sActual, param)
9:     Si aptitud(sNueva) > aptitud(sActual):
10:       sActual = sNueva
11:    Si aptitud(sActual) > aptitud(sMejor):
12:       sMejor = sActual
13:    nIter + 1
14: Retornar sMejor
Fin busquedaLocalIterada
```

Donde:

- Línea 1: se obtiene el máximo número de iteraciones a realizar
- Línea 2: se calcula el grado de perturbación a utilizar
- Línea 3: se inicializa el contador de iteraciones
- Línea 4: se realiza un proceso de búsqueda local sobre la solución inicial y se guarda como la solución “actual”
- Línea 5: se inicializa la solución mejor con el valor de la solución actual
- Línea 6: se inicia el bucle principal que solo termina si se alcanza el número máximo de iteraciones permitidas
    - Línea 7: se aplica la perturbación a la solución actual
    - Línea 8: se aplica el proceso de búsqueda local a la solución actual perturbada y se genera una nueva solución
    - Línea 9-10: si esta solución es mejor que la actual, entonces se actualiza como la nueva solución actual
    - Línea 11-12: si la solución actual es mejor que la “mejor solución” entonces se actualiza el valor de la mejor solución
    - Línea 13: se actualiza el número de iteraciones
- Línea 14: finalmente se retorna la mejor solución hallada

- **Algoritmo Búsqueda Tabú**

  Es un tipo de algoritmo de búsqueda local que utiliza memoria para almacenar los "movimientos" o acciones realizadas anteriormente, lo cual permite guiar las búsquedas hacia entornos más prometedores, este elemento guardado es llamado "el elemento tabú". En este caso el atributo guardado contiene una lista compuesta por dos listas, la primera es una lista con los índices de los discos intercambiados, y la segunda es una lista con las tablas nuevas asignadas a estos discos. En otras palabras, lo que se almacena es el movimiento previo que se realizó para llegar a la solución actual. En la **Figura 35** se puede visualizar un ejemplo de esta representación del atributo tabú.

Figura 35. *Ejemplo del atributo tabú*

Para saber si un elemento está presente en la lista tabú, solo se debe verificar si el movimiento que lo llevó a esa solución ya fue realizado con anterioridad, si esto es así, se considera como tabú y tendrá que realizar otro movimiento distinto. Asimismo, en la **Figura 36** puede visualizarse el detalle del diseño de este algoritmo.

Figura 36. *Diseño del algoritmo de búsqueda tabú*

```
Inicio busquedaTabu (P, sInicial, param)
1:  kOpt, nVecinos, mAtrib, pIter = param
2:  nIter = 0
3:  listaTabu = []
4:  sActual = sInicial
5:  sMejor = sInicial
6:  Mientras nIter < pIter Y sActual ≠ nulo Hacer:
7:     sNueva = nulo
8:     atrNuevo = nulo
9:     Para vecino En generarVecinos (sActual, nVecinos, kOpt):
10:       sVecina = vecino
11:       estaEnListaTabu = contiene (listaTabu, vecino)
12:       Si ((No estaEnListaTabu Y (sNueva == nulo O sVecina > sNueva)) O
(sVecina ≥ sMejor)):
13:          sNueva = sVecina
14:          atrNuevo = Atributo (obtenerAtributo (vecino))
15:    sActual = sNueva
16:    Si sActual ≠ nulo:
17:       Si sActual ≥ sMejor:
18:          sMejor = sActual
19:       Si listaTabu.tamanho == mAtrib:
20:          quitarPrimero (listaTabu)
21:       agregarAlFinal (listaTabu, atrNuevo)
22:    nIter + 1
23: Retornar sMejor
Fin busquedaTabu
```

Donde:

- Línea 1: se obtienen los parámetros
- Línea 2 y 3: se inicializa el contador de iteraciones y la lista tabú
- Línea 4 y 5: se inicializa la solución actual y la mejor solución con la solución inicial que se recibe como parámetro
- Línea 6: se inicia el bucle principal que solo para cuando se llega a un número máximo de iteraciones permitidas o cuando ningún movimiento puede superar a la mejor solución actual
    - Línea 7 y 8: la “nueva solución” y el “atributo tabú” se inicializan como nulos
    - Línea 9: se inicia un nuevo bucle que recorre todo el espacio de vecinos generado gracias a un algoritmo k-opt, el cual toma la solución enviada como parámetro y elige aleatoriamente “k” elementos para ser intercambiados

- Línea 10 y 11: se extrae la solución nueva y se evalúa si está presente en la lista tabú
- Línea 12-14: se evalúa si el vecino no está en la lista tabú y la solución nueva no fue actualizada aún o el vecino es mejor que la solución nueva o si el vecino es mejor que la mejor solución hasta el momento. En caso se cumpla la condición se actualiza la nueva solución con los datos del vecino.

- Línea 15: luego de que se exploran todos los vecinos se actualiza la solución actual con la solución nueva
- Línea 16: En caso de que la solución actual no sea nula, se evalúa las siguientes líneas
  - Línea 17-18: si la solución actual resulta ser mejor que la “mejor solución”, se coloca como nueva mejor a esta
  - Línea 19-20: si la lista tabú llega al límite de elementos, se elimina el más antiguo
  - Línea 21: Se agrega el atributo de la solución nueva
- Línea 22: se actualiza el número de iteraciones

- Línea 23: una vez finalizado el buble principal se retorna la mejor solución encontrada

Anexo 11: Detalle del proceso de generación de datos aleatorios

La generación de datos aleatorios se realiza mediante un algoritmo de creación de datos, este algoritmo se llama “creador de instancias” y tiene la capacidad de crear tres tipos de instancias:

i. **Instancias pequeñas**: las cuales constan de un número de discos entre 5 y 20, y un número de tablas entre 15 y 100
ii. **Instancias medianas**: las cuales constan de un número de discos entre 21 y 40, y un número de tablas entre 75 y 240
iii. **Instancias grandes**: las cuales constan de un número de discos entre 41 y 60, y un número de tablas entre 135 y 360

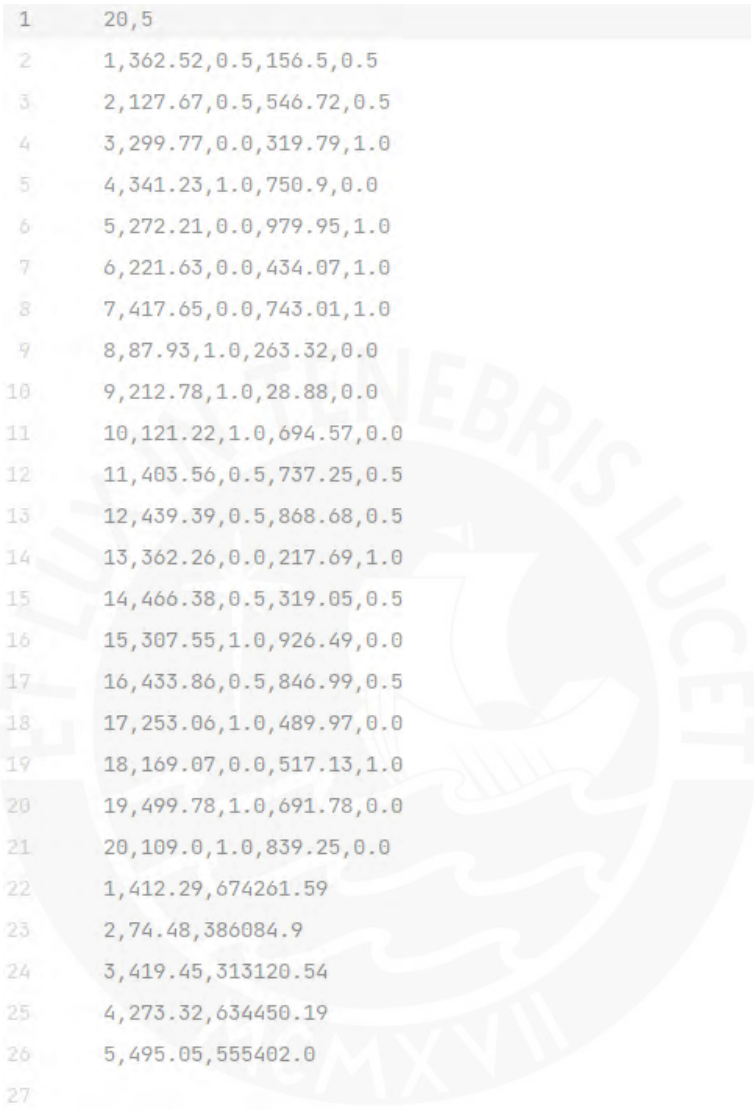
Al momento de generar cada instancia, independientemente del tipo que esta tenga, se toma en cuenta las siguientes consideraciones:

i. La capacidad de las unidades de almacenamiento generadas tendrá un valor suficientemente alto, generado aleatoriamente entre 250000 y 1000000. Esto se realiza para asegurar que se simula un “espacio” disponible suficientemente alto para albergar una gran cantidad de tablas en cada disco.
ii. El rendimiento de las unidades de almacenamiento tendrá un valor entre 50 y 500, el cual será generado aleatoriamente.
iii. El tamaño de las tablas es generado aleatoriamente entre 50 y 500.
iv. La frecuencia de uso de las tablas simuladas tiene un valor generado aleatoriamente entre 10 y 1000.
v. El valor de los coeficientes de importancia, tanto para el tamaño de las tablas como para la frecuencia de uso es generado aleatoriamente entre los valores 0.5, 0.0 y 1.0. Cabe mencionar que, tal como se diseñó esta variable, ambos coeficientes suman 1.0 en todo momento.

Finalmente, la información de la instancia es almacenada en un archivo, el cual está listo para ser usado en cualquiera de las posteriores pruebas. En la **Figura 37** se muestra un ejemplo

de la estructura del archivo generado.

Figura 37. *Estructura del archivo de datos generados aleatoriamente*



```
20,5
1,362.52,0.5,156.5,0.5
2,127.67,0.5,546.72,0.5
3,299.77,0.0,319.79,1.0
4,341.23,1.0,750.9,0.0
5,272.21,0.0,979.95,1.0
6,221.63,0.0,434.07,1.0
7,417.65,0.0,743.01,1.0
8,87.93,1.0,263.32,0.0
9,212.78,1.0,28.88,0.0
10,121.22,1.0,694.57,0.0
11,403.56,0.5,737.25,0.5
12,439.39,0.5,868.68,0.5
13,362.26,0.0,217.69,1.0
14,466.38,0.5,319.05,0.5
15,307.55,1.0,926.49,0.0
16,433.86,0.5,846.99,0.5
17,253.06,1.0,489.97,0.0
18,169.07,0.0,517.13,1.0
19,499.78,1.0,691.78,0.0
20,109.0,1.0,839.25,0.0
1,412.29,674261.59
2,74.48,386084.9
3,419.45,313120.54
4,273.32,634450.19
5,495.05,555402.0
```

Como se puede notar, en la primera línea se tiene a la cantidad de tablas y discos simulados, separados por comas. Luego se encuentra la información de las tablas en el formato: “índice de la tabla, coeficiente de importancia del tamaño, tamaño, coeficiente de importancia de la frecuencia de uso, frecuencia de uso”. Finalmente se encuentra la información de los discos, separada por comas siguiendo el formato: “índice del disco, rendimiento del disco, capacidad disponible del disco”.

Anexo 12: Estándares empleados en la programación

Para la elaboración de este anexo se empleó como referencia al documento titulado "Convenciones de Código en Java", publicado por la empresa creadora del lenguaje Java: Sun Microsystems. Sin embargo, se modificaron algunos aspectos sobre el idioma para lograr una mejor claridad del código. El detalle de los estándares se muestra en la **Tabla 26**.

Tabla 26. *Estándares de programación*

Nro.	Elemento	Estándar	Ejemplo
1	Nombres de archivos	Minúsculas, español	controlador
2	Nombre de las clases	La primera letra en mayúsculas, español	Tabla
3	Nombre de los métodos	La primera letra en minúsculas, español	correrAlgoritmo
3	Nombre de atributos	La primera letra en minúsculas, español	frecuenciaUso
4	Nombre de constantes	Todo en mayúsculas, español	TASA_MUTACION
5	Sangría o "Indentation"	4 espacios	-
6	Comentarios	Usando /**/ y //, español	// comentario ejemplo

Además de lo presentado anteriormente se tienen algunas normas a tomar en cuenta:

- Declarar las variables y atributos de clase en líneas distintas, es decir cada variable o atributo en cada línea
- Evitar el uso de paréntesis en las declaraciones "retorno"
- Evitar omitir los corchetes en la declaración de condicionales "If-Else"
- Separar adecuadamente los métodos mediante una línea en blanco
- Evitar que se acumulen las variables en una declaración, separar por espacio siempre que se requiera una mejor visualización de los procedimientos programados
- Omitir el alineamiento horizontal innecesario

Anexo 13: Código fuente del proyecto de tesis

Para visualizar la implementación de los algoritmos puede ingresar al siguiente enlace: "código fuente del proyecto de tesis".

Anexo 14: Documento de pruebas unitarias sobre el algoritmo memético

Para visualizar el documento de pruebas unitarias puede ingresar al siguiente enlace: "Documento de pruebas unitarias".

Anexo 15: Documento de validación de la implementación del algoritmo memético firmado por un especialista

Para visualizar el documento de validación puede ingresar al siguiente enlace: "Documento de validación"

Anexo 16: Formato de los archivos de entrada para el software de ejecución

En este anexo se presenta el detalle de los archivos de entrada recibidos por el software de ejecución del algoritmo memético, los cuales contienen la información necesaria para completar el proceso de asignación. En la **Figura 38** se muestra un ejemplo simple de la estructura del archivo que contiene la información acerca de las tablas de base de datos que se desean distribuir.

Figura 38. *Ejemplo del archivo de entrada con los datos de las tablas*

Nro	Tamaño	Coeficiente de importancia del tamaño	Frecuencia de uso	Coeficiente de importancia de la frecuencia de uso
1	873.88	1.00	103.32	0.00
2	780.21	0.50	149.15	0.50
3	764.72	0.50	238.20	0.50
4	709.07	0.50	253.48	0.50
5	635.82	1.00	289.15	0.00
6	607.70	1.00	293.75	0.00
7	530.51	1.00	339.85	0.00
8	520.30	0.00	382.30	1.00
9	313.91	0.00	478.51	1.00
10	296.51	1.00	521.29	0.00

En el ejemplo se puede ver que cada columna corresponde a un dato acerca de las tablas y cada fila representa a una tabla de la base de datos. Asimismo, en la **Figura 39** se tiene un ejemplo de la estructura del archivo que contiene la información necesaria acerca de las unidades de almacenamiento a utilizar.

Figura 39. *Ejemplo del archivo de entrada con los datos de los discos*

Nro	Rendimiento	Capacidad
1	181.38	9600.00
2	190.14	9390.00
3	317.75	8600.00

En el ejemplo se puede notar que cada columna corresponde a un dato acerca da los discos y cada fila representa a un “disco” o unidad de almacenamiento. Cabe mencionar que ambos

formatos mostrados deben respetarse al momento de utilizar el software de ejecución, pues de lo contrario este no entregará resultados adecuados.

Anexo 17: Documento de las pruebas unitarias realizadas sobre el software de ejecución de algoritmos

Para visualizar el documento de pruebas dirigirse al siguiente enlace: “Documento de pruebas unitarias”.

Anexo 18: Documentos de validación del software de ejecución del algoritmo memético

Para visualizar los documentos de validación del software de ejecución puede acceder al siguiente enlace: “Documentos de validación”.

Anexo 19: Documento de pruebas de flujo de datos realizadas sobre el diseño del algoritmo GRASP

Para visualizar el documento de pruebas realizadas puede dirigirse al siguiente enlace: “Pruebas de flujo de datos”.

Anexo 20: Documento de validación del diseño del algoritmo GRASP

Para visualizar los documentos de validación del diseño del algoritmo GRASP puede acceder al siguiente enlace: “Documentos de validación”.

Anexo 21: Documento de pruebas unitarias sobre el algoritmo GRASP

Para visualizar el documento de pruebas unitarias puede ingresar al siguiente enlace: "Documento de pruebas unitarias".

Anexo 22: Documento de validación de la codificación del algoritmo GRASP

Para visualizar los documentos de validación de la codificación del algoritmo GRASP puede acceder al siguiente enlace: “Documentos de validación”.

Anexo 23: Resultados de la incorporación de los algoritmos de búsqueda local propuestos al algoritmo memético

En la **Tabla 27** se muestran los resultados de cada algoritmo de búsqueda local cuando es integrado al algoritmo memético.

Tabla 27. *Resultados de pruebas del algoritmo memético*

Nro.	Instancia	AM + ABT		AM + ABLI		AM + AES	
		Mejor	Promedio	Mejor	Promedio	Mejor	Promedio
1	20-5	5.09	5.09	5.08	5.06	5.09	5.06
2	30-10	5.74	5.74	5.7	5.66	5.71	5.67
3	40-10	10.55	10.53	10.45	10.38	10.46	10.37
4	80-20	24.81	24.73	24.44	24.34	24.52	24.38
5	100-20	26.26	26.17	25.81	25.66	25.79	25.62
6	100-25	28.06	27.94	27.58	27.4	27.68	27.45
7	120-40	33.9	33.68	33.33	33.08	33.31	33.07
8	150-30	36.78	36.55	36.05	35.6	35.9	35.56
9	150-50	40.44	40.13	39.4	38.88	39.5	39.07
10	160-40	45.96	45.63	44.95	44.54	44.98	44.48
11	165-55	51.44	51.15	49.43	48.76	49.87	48.99
12	200-40	51.49	51.22	49.86	49.34	50.2	49.54
13	225-45	62.3	61.64	59.6	59.07	59.89	59.29
14	240-60	66.77	66.33	64.25	63.67	64.58	63.7
15	250-50	77.01	76.27	73.65	73.02	74.64	73.41
Promedio Neto		**37.77**	**37.52**	**36.64**	**36.3**	**36.81**	**36.38**

Fuente: Elaboración propia

Adicionalmente, en la **Tabla 28** se muestran los parámetros utilizados para ejecutar las pruebas.

Tabla 28. *Tabla de parámetros usados en las pruebas*

Algoritmo	Parámetro	Valor
Búsqueda tabú	Atributos tabúes	20
	Vecinos a explorar	10
	Criterio de terminación	No mejora, 50 * tamaño del cromosoma
Enfriamiento simulado	Temperatura inicial	1000
	Vecinos a explorar	25
	Criterio de terminación	50 * tamaño del cromosoma
Búsqueda local iterada	Criterio de terminación	50 * tamaño del cromosoma

Algoritmo memético	Tamaño de la población	50
	Tasa de casamiento	0.5
	Tasa de mutación	0.5
	Probabilidad de mutación	0.5
	Máximo número de generaciones sin mejora	30
	Criterio de terminación	50

Fuente: Elaboración propia

Anexo 24: Documento de calibración de variables

Para visualizar el documento de calibración de variables del algoritmo memético puede dirigirse al siguiente enlace: “Documento de calibración”

Anexo 25: Documento de validación da la calibración de variables

Para visualizar los documentos de validación de la calibración de variables del algoritmo Memético puede acceder al siguiente enlace: “Documentos de validación”

Anexo 26: Pruebas adicionales de experimentación numérica

En la **Tabla 29** se muestran los resultados obtenidos por cada algoritmo estudiado sobre el conjunto de datos que contiene 40 muestras acerca de 40 tablas y 10 discos. Al igual que con las pruebas mencionadas en la sección de Experimentación numérica, cada algoritmo se corre 20 veces, registrándose el mejor resultado, el resultado promedio y la desviación estándar.

Tabla 29. *Resultados de las pruebas adicionales*

Muestra	MEMÉTICO			GRASP		
	Mejor	Promedio	Desv. Estd.	Mejor	Promedio	Desv. Estd.
1	12.6836	12.6803	0.0033	12.4071	12.3324	0.0482
2	10.4904	10.4875	0.0027	10.2654	10.2125	0.0215
3	11.5188	11.5170	0.0019	11.3221	11.2726	0.0292
4	11.6196	11.6176	0.0016	11.4032	11.3508	0.0284
5	12.6008	12.5972	0.0028	12.4174	12.3321	0.0344
6	11.4439	11.4373	0.0047	11.2734	11.2240	0.0253
7	13.1238	13.1191	0.0039	12.8997	12.8375	0.0299
8	12.2783	12.2739	0.0037	12.0109	11.9544	0.0274
9	11.1659	11.1627	0.0019	10.9700	10.9117	0.0282
10	11.8851	11.8804	0.0035	11.6147	11.5705	0.0241
11	9.3975	9.3945	0.0027	9.1586	9.1036	0.0294
12	11.7316	11.7278	0.0022	11.4974	11.4525	0.0321
13	10.7498	10.7443	0.0027	10.5600	10.4596	0.0368
14	10.0582	10.0580	0.0004	9.7425	9.7096	0.0199
15	12.2576	12.2566	0.0012	12.0497	12.0104	0.0248
16	10.4372	10.4357	0.0021	10.3009	10.2393	0.0256
17	9.0279	9.0271	0.0008	8.8050	8.7506	0.0283
18	10.9987	10.9948	0.0036	10.6925	10.6550	0.0213
19	11.1989	11.1960	0.0024	11.0274	10.9755	0.0262
20	10.0293	10.0278	0.0017	9.7190	9.6587	0.0291
21	8.7855	8.7838	0.0021	8.4396	8.4020	0.0235
22	14.2114	14.2077	0.0027	14.0234	13.9213	0.0417
23	12.0001	11.9980	0.0028	11.7456	11.7032	0.0203
24	14.5456	14.5406	0.0041	14.3449	14.2946	0.0252
25	9.3375	9.3354	0.0017	9.0952	9.0524	0.0273
26	13.6387	13.6341	0.0041	13.4679	13.3762	0.0348
27	13.0120	13.0068	0.0036	12.7777	12.7247	0.0369
28	12.6651	12.6639	0.0018	12.4216	12.3639	0.0342
29	8.7135	8.7082	0.0036	8.4713	8.4047	0.0311
30	11.6978	11.6921	0.0046	11.5119	11.4801	0.0189

31	11.4425	11.4412	0.0018	11.0844	11.0405	0.0215
32	10.1934	10.1925	0.0009	9.9255	9.8639	0.0243
33	10.4935	10.4931	0.0003	10.2284	10.1839	0.0241
34	11.7081	11.7021	0.0051	11.4708	11.4275	0.0187
35	12.1842	12.1822	0.0024	11.9399	11.8834	0.0326
36	12.2908	12.2877	0.0028	12.0671	12.0056	0.0326
37	11.8105	11.8066	0.0043	11.5550	11.5125	0.0213
38	13.6314	13.6175	0.0077	13.3717	13.3161	0.0206
39	11.5160	11.5003	0.0087	11.3346	11.2992	0.0171
40	9.7787	9.7766	0.0034	9.5714	9.5071	0.0267
Promedio Neto	**11.4588**	**11.4551**	**0.0030**	**11.2246**	**11.1694**	**0.0276**

Cabe mencionar que para la experimentación numérica se toma en cuenta a los mejores resultados de ambos algoritmos. Dicho esto, a continuación, se muestra un resumen de las pruebas de estadísticas realizadas:

- **Prueba de Shapiro-Wilk**

En las **Tablas 30 y 31** se muestran los resultados obtenidos de esta prueba en cada uno de los algoritmos propuestos.

Tabla 30. *Resultados pruebas adicionales de Shapiro-Wilk del algoritmo Memético*

Estadísticos descriptivos del conjunto de datos	
Media	11.46
Desviación Estándar	1.44
Varianza	2.08
Mínimo	8.71
Máximo	14.55
Cantidad de datos	40.00
Resultados de la prueba	
Grado de significancia	0.05
P-valor	0.80

Tabla 31. *Resultados de pruebas adicionales de Shapiro-Wilk del algoritmo GRASP*

Estadísticos descriptivos del conjunto de datos	
Media	11.22
Desviación Estándar	1.45
Varianza	2.12
Mínimo	8.44
Máximo	14.34
Cantidad de datos	40.00

Resultados de la prueba	
Grado de significancia	0.05
P-valor	0.79

Debido a que el p-valor obtenido es mayor al nivel de significancia, se concluye que la hipótesis nula no es rechazada y la muestra sigue una distribución normal.

- **Prueba F**

En la **Tabla 32** pueden visualizarse los resultados de esta prueba adicional.

Tabla 32. *Resultados de pruebas adicionales de F de Fisher*

	Memético	GRASP
Media	11.46	11.22
Desviación Estándar	1.44	1.45
Varianza	2.08	2.12
Mínimo	8.71	8.44
Máximo	14.55	14.34
Cantidad de datos	40.00	
F	0.98	
Grado de significancia	0.05	
P-valor	0.95	

Se puede notar que el p-valor obtenido es mayor al nivel de significancia, por tal motivo, se concluye que las varianzas de los resultados de los algoritmos son significativamente homogéneas.

- **Prueba Z**

Para esta prueba se plantean las siguientes hipótesis:

  - H0: la media del algoritmo Memético es igual a la media del algoritmo GRASP
  - H1: la media del algoritmo Memético es distinta a la media del algoritmo GRASP

Dicho esto, en la **Tabla 33** se muestran los resultados alcanzados en esta prueba.

Tabla 33. *Prueba Z de los algoritmos Memético y GRASP*

	Memético	GRASP
Media	11.46	11.22
Desviación Estándar	1.44	1.45
Varianza	2.08	2.12

Mínimo	8.71	8.44
Máximo	14.55	14.34
Cantidad de datos	40.00	
Prueba de dos colas		
Z	0.72	
Grado de significancia	0.05	
P-valor	0.47	

Como puede observarse en la **Tabla 33**, el p-valor resultante de la prueba de dos colas es mayor al nivel de significancia, por lo que se concluye que no se rechaza la hipótesis nula y que la media de los resultados obtenidos por los algoritmos propuestos es igual, es decir, que ambos algoritmos brindan resultados igual de buenos cuando se trata de un problema con 40 tablas y 10 discos.

Anexo 27: Documento de validación da la experimentación numérica

Para visualizar los documentos de validación de la experimentación numérica realizada para comparar los algoritmos Memético y GRASP, puede acceder al siguiente enlace: "Documentos de validación"