**PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ**

**FACULTAD DE CIENCIAS E INGENIERÍA**


# ALGORITMO METAHEURÍSTICO PARA LA OPTIMIZACIÓN DE CONSULTAS SQL EN BASES DE DATOS DISTRIBUIDAS RELACIONALES

**Tesis para obtener el título profesional de Ingeniero Informático**

**AUTOR:**

Jesús Angel Eduardo Sangama Ramírez

**ASESOR:**

Mag. Rony Cueva Moscoso

Lima, Marzo, 2024

**Informe de Similitud**

Yo, …Rony Cueva Moscoso……,

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

…… ALGORITMO METAHEURÍSTICO PARA LA OPTIMIZACIÓN DE CONSULTAS SQL EN BASES DE DATOS DISTRIBUIDAS RELACIONALES……..,

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

…… Jesús Angel Eduardo Sangama Ramírez……,

……,

……,

dejo constancia de lo siguiente:

- El mencionado documento tiene un índice de puntuación de similitud de 13%. Así lo consigna el reporte de similitud emitido por el software *Turnitin* el 22/01/2024.
- He revisado con detalle dicho reporte y la Tesis o Trabajo de Suficiencia Profesional, y no se advierte indicios de plagio.
- Las citas a otros autores y sus respectivas referencias cumplen con las pautas académicas.

Lugar y fecha: …Lima, 13 de marzo del 2024……

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

## Resumen

En el contexto empresarial, los datos tienen una importancia significativa tanto para la operación del día a día en una organización como para la toma de decisiones dentro de esta. Por ello, resulta vital que consultar dichos datos sea un proceso lo más eficiente posible. Para las bases de datos relacionales, una forma de lograr esto es la optimización de consultas SQL, y entre los diferentes métodos de optimización se encuentran los algoritmos metaheurísticos.

El presente trabajo realiza una investigación de la literatura académica centrada en estos algoritmos aplicados a la optimización de consultas en bases de datos distribuidas relacionales y decide realizar una comparación entre el algoritmo genético (el cual cuenta con gran popularidad en este ámbito) y el memético, con el fin de evaluar si la aplicación de este último resulta viable para este tipo de optimización.

Para lograr lo planteado anteriormente, el trabajo busca definir variables, parámetros y restricciones del problema de optimización de consultas; las cuales posteriormente son utilizadas para implementar adaptaciones propias de los algoritmos genético y memético orientadas a dicho problema. Finalmente, se realiza una comparación de eficacia y eficiencia entre ambas implementaciones a través de experimentación numérica.

Tras finalizar todas las tareas anteriores, se concluye que se logró implementar un algoritmo memético para optimizar consultas SQL en bases de datos distribuidas relacionales cuyo rendimiento puede superar al algoritmo genético para escenarios de complejidad creciente (es decir, bases de datos con numerosas tablas y sitios).

## Tabla de contenidos

# Índice de tablas

# Índice de figuras

# 1. Problemática

En esta sección se desarrollará la problemática a tratar en el presente trabajo de tesis, junto al contexto bajo el que esta existe. De manera resumida, la problemática gira en torno a la necesidad de mejorar el rendimiento de consultas y a las dificultades para lograr esto de manera eficiente.

## 1.1. Árbol de Problemas


*Figura 1. Árbol de problemas*

## 1.2. Descripción

En el mundo empresarial actual, los datos han sido considerados unos de los recursos estratégicos más importantes para las organizaciones, lo que ha

impulsado a estas a utilizar sistemas de información que utilicen dichos datos para apoyarlas en la toma de decisiones. Un componente vital de cualquier sistema de información son los Sistemas de Administración de Bases de Datos o DBMS por sus siglas en inglés. En este sentido, la eficiencia de los sistemas de información depende enormemente del rendimiento del DBMS que utilizan (Wijesiriwardana C., Firdhous M., 2019). En este contexto, la existencia de bases de datos distribuidas cobra importancia. Estas se pueden definir como un conjunto de bases de datos repartidas en diferentes ubicaciones físicas e intercomunicadas por medio de una red, lo que permite mejoras de rendimiento, accesibilidad y modularidad respecto a las bases de datos centralizadas en casos de manejo de grandes volúmenes de datos (Panahi V., Navimipour N., 2019).

Ya sea para bases de datos centralizadas o distribuidas, es posible definir cuatro enfoques principales para mejoras de rendimiento: mejoras en la administración de la base de datos, mejoras en el sistema sobre el que funciona dicha base, mejoras en la red que utiliza la base de datos, y mejoras en las consultas SQL que realizan los usuarios. Sin embargo, se estima que el 80% de problemas de rendimiento en bases de datos pueden solucionarse con el último enfoque, es decir, realizando ajustes a consultas SQL (Myalapalli V.,Savarapu P., 2015). De esta forma queda evidenciada la relevancia de la optimización de consultas SQL como tema de investigación.

Un primer aspecto a considerar está relacionado a los factores o variables tomados en cuenta en la optimización de las consultas. De manera general, las consultas extraen datos de una tabla de la base de datos mediante el uso de operadores de selección y proyección. Los datos de diferentes tablas son relacionados mediante la aplicación de operadores *join*. Además, los datos que residen en almacenamiento no volátil (por ejemplo, discos duros) deben ser transferidos a memoria principal para poder ser procesados. Así, se comprende que las capacidades de I/O y de procesamiento de la base de datos pueden influir en el rendimiento de una consulta (Sharma M. et al, 2015). Por otro lado, en el contexto de bases de datos distribuidas, los datos se encuentran

repartidos, e incluso replicados, en diferentes sitios físicos. En estos casos, la selección de sitios de donde extraer los datos y la transmisión de los mismos entre diferentes sitios a través de la red también afectan la eficiencia de una consulta (Panahi V., Navimipour N., 2019). Incluso se pueden aprovechar las capacidades de paralelismo intersitio e intrasitio de una base de datos distribuida para acelerar el procesamiento (Zheng W. et al., 2018) y ejecución (Sharma M. et al, 2015) de consultas. A partir de lo anterior, se sabe que existen numerosos factores relevantes para el problema analizado, pero el peso o influencia de cada uno puede variar dependiendo de las características de la base de datos y de las prioridades del administrador y usuarios de la misma. En el ámbito académico, por ejemplo, existen estudios que enfocan la optimización en factores como el orden de *joins* del plan de ejecución (Yao M., 2018; Zheng W. et al., 2018; Han Y. et al, 2016) o la minimización de costos de transmisión entre sitios (Liu S., Xu X., 2016; Vijay T., Yadav M., 2017). Esta variabilidad de pesos para cada factor puede dar lugar al desarrollo de métodos de optimización que no sean aplicables para varias situaciones, es decir, que trabajen con asignaciones de pesos dinámicos para las variables de optimización.

A nivel técnico, una consulta SQL puede ser representada por diferentes estructuras equivalentes llamadas planes de ejecución (QEP, siglas para *Query Execution Plan*). Estos planes generan el mismo resultado, es decir, que todos recuperan los mismos datos que solicitó el usuario, pero difieren en el orden y forma en la que se ejecutan las diferentes operaciones necesarias para la obtención de los datos, siendo así el rendimiento diferente para cada uno (Abdalla M., Karabatak M., 2020). En este sentido, la optimización de una consulta implica seleccionar el plan de ejecución que tenga mejor rendimiento. Para consultas largas, es decir, que involucren numerosas tablas de donde extraen los datos, el número de QEPs equivalentes aumenta de manera exponencial, generándose espacios de búsqueda de gran tamaño. Debido a esto, la optimización de consultas con múltiples tablas es considerada un problema *NP-hard* (Zheng W. et al., 2018). Entre las estrategias de búsqueda

utilizadas para este problema, una categoría principal corresponde a los métodos exhaustivos, llamados así porque encuentran el plan de ejecución óptimo tras buscar en todo el espacio de búsqueda. Un ejemplo de estos es el algoritmo de *Dynamic Programming* (DP) o Programación Dinámica, el cual ha sido utilizado por algunos DBMS comerciales (Raipurkar A., Bamnote G., 2013). Lamentablemente, debido a su naturaleza, estos algoritmos tienen complejidad espacial y temporal exponencial, lo que resulta en tiempos exagerados para optimizar consultas grandes (Mohsin S. et al., 2021). Así, conforme aumenta la cantidad de tablas en una consulta, los métodos exhaustivos aumentarán enormemente su tiempo de ejecución y la cantidad de memoria que necesitan para evaluar todos los planes de búsqueda, produciéndose así problemas de rendimiento, concurrencia de consultas y bloqueo de recursos en la base de datos. Imponer una limitación de tiempo en la búsqueda exhaustiva anularía la garantía de obtener un plan óptimo o incluso cercano al óptimo, lo que podría traducirse a tiempos largos de ejecución de la consulta.

Si bien se ha hablado de las dificultades de los métodos exhaustivos para atacar la optimización de consultas de manera eficiente, existe otra categoría de estrategias de búsqueda que pueden hacerle frente de mejor manera: las metaheurísticas. Dentro de estas, un algoritmo que ha sido ampliamente estudiado y adaptado al problema de optimización de consultas es el algoritmo genético. Este trabaja con una población de cromosomas (cada uno representando un plan de ejecución válido para la consulta a optimizar), sobre la cual utiliza operadores genéticos de selección, cruce y mutación para producir cromosomas nuevos, los que a su vez vuelven a formar parte de la población representando nuevas generaciones. Esta estrategia permite encontrar soluciones buenas o incluso óptimas cubriendo solamente una fracción del espacio de búsqueda, lo que se traduce a una menor complejidad temporal y espacial. Diferentes trabajos han realizado mejoras y modificaciones al algoritmo genético tradicional para obtener mejores tiempos de optimización y/o una mayor calidad de soluciones (Sharma M. et al, 2015;

Sharma M. et al, 2016; Ban W. et al, 2016; Venkata S., Vatsavayi V., 2017). Sin embargo, resulta importante considerar la antigüedad del algoritmo: el estudio sobre el algoritmo genético aplicado en la optimización de consultas se remonta a 30 años atrás (Bennet K. et al., 1991), lo que nos lleva a pensar en buscar otras metaheurísticas desarrolladas posteriormente para utilizar en la optimización de consultas. Ya existen estudios recientes para algoritmos como el de Colonia de Hormigas (Mohsin S. et al., 2021) o el *Cuckoo* (Vijay T., Yadav M., 2017), en los cuales se han obtenido resultados favorables, pero no se han encontrado estudios en los que se realicen comparaciones de rendimiento entre las propuestas genéticas recientes y metaheurísticas como los algoritmos memético, murciélago, lobo gris, entre otros. Por otro lado, si bien el algoritmo genético puede ser considerado un ejemplo y un punto de comparación para la mejora continua de métodos de optimización de consultas, éste también abre paso a un problema: existen estudios que imponen limitaciones al alcance de los algoritmos que plantean. Por ejemplo, se ignoran costos de comunicación entre sitios (Yao M., 2018), capacidades de procesamiento local (Liu S., Xu X., 2016; Panahi V., Navimipour N., 2019) o incluso ambos factores (Han Y. et al., 2019) en la construcción de modelos de costo y lógicas de optimización.

En conclusión, los diferentes problemas o factores mencionados, agravados por las necesidades modernas de información, giran en torno a lo que ha sido escogido como el problema central a atacar en la tesis: las dificultades para optimizar consultas con una baja complejidad temporal.

## 1.3. Problema seleccionado

Dificultad para optimizar el rendimiento de consultas con baja complejidad temporal.

# 2. Objetivos

## 2.1. Objetivo General

Implementar un algoritmo memético para resolver el problema de optimización de consultas en bases de datos distribuidas relacionales.

## 2.2. Objetivos Específicos

Para el presente trabajo se tienen los siguientes objetivos específicos:

**O1:** Definir las variables, parámetros y restricciones del algoritmo, y diseñar estructuras de datos que las soportan para diseñar la función objetivo utilizada en la optimización.

**O2:** Adaptar el algoritmo metaheurístico memético para la optimización de consultas.

**O3:** Comparar el algoritmo memético propuesto y un algoritmo genético adaptado por el tesista, con el fin de obtener resultados mejores en términos de eficiencia y calidad de soluciones generadas.

## 2.3. Resultados Esperados

A continuación, se listan los resultados esperados para cada objetivo específico definido.

**(O1) Definir las variables, parámetros y restricciones del algoritmo, y diseñar estructuras de datos que las soporten para diseñar la función objetivo utilizada en la optimización:**

- **R1.1:** Variables, parámetros y restricciones que serán consideradas en el algoritmo propuesto.
- **R1.2:** Diseño de estructuras de datos para representar las variables y restricciones descritas.
- **R1.3:** Función objetivo utilizada en el algoritmo propuesto.

**(O2) Adaptar el algoritmo metaheurístico memético para la optimización de consultas:**

- **R2.1:** Pseudocódigo del algoritmo propuesto.
- **R2.2:** Codificación del algoritmo propuesto.
- **R2.3:** Calibración de variables utilizadas en el algoritmo propuesto.

**(O3) Comparar el algoritmo memético propuesto y un algoritmo genético adaptado por el tesista, con el fin de obtener resultados mejores en términos de eficiencia y calidad de soluciones generadas:**

- **R3.1:** Codificación de algoritmo genético con el cual se realizará la comparación.
- **R3.2:** Módulo de software para comparación de los algoritmos.
- **R3.3:** Experimentación numérica en base a los resultados del módulo de comparación.

## 2.4. Mapeo de objetivos, resultados y verificación

*Tabla 1. Medios de verificación e Indicadores objetivamente verificables*

| **O1:** Definir las variables, parámetros y restricciones del algoritmo, y diseñar estructuras de datos que las soporten para diseñar la función objetivo utilizada en la optimización | | |
|---|---|---|
| **Resultado** | **Medio de verificación** | **Indicador objetivamente verificable** |
| **R1.1:** Variables, parámetros y restricciones que serán consideradas en el algoritmo propuesto. | -Documento conteniendo el listado y descripción de parámetros, variables y restricciones a utilizar en el algoritmo | -Revisión y validación al 100% de variables, parámetros y restricciones a cargo de un especialista en bases de datos. |
| **R1.2:** Diseño de estructuras de datos para representar las variables y restricciones descritas | -Documento conteniendo el listado y descripción de estructuras de datos utilizadas para representar cada variable y restricción utilizada | -Revisión y validación al 100% de las estructuras de datos diseñadas a cargo de un especialista en algoritmos. |
| **R1.3:** Función objetivo utilizada en el algoritmo | -Documento conteniendo la fórmula matemática de | -Pruebas exitosas de comportamiento de la |

| | | |
|---|---|---|
| propuesto | la función objetivo y la descripción de los diferentes componentes de la misma | función objetivo utilizando datos aleatorios.<br><br>-Revisión y validación al 100% de la función objetivo a cargo de un especialista en algoritmos. |

| **O2**: Adaptar el algoritmo metaheurístico memético para la optimización de consultas | | |
|---|---|---|
| **Resultado** | **Medio de verificación** | **Indicador objetivamente verificable** |
| **R2.1:** Pseudocódigo del algoritmo propuesto | -Documento que contiene el pseudocódigo y la explicación del mismo para el algoritmo propuesto | -Pruebas de flujo exitosas siguiendo el pseudocódigo brindado.<br><br>-Revisión y validación al 100% del pseudocódigo por parte de un especialista en algoritmia metaheurística. |
| **R2.2:** Codificación del algoritmo propuesto | -Código fuente y ejecutable del algoritmo propuesto | -Pruebas unitarias exitosas del algoritmo propuesto. |
| **R2.3:** Calibración de variables utilizadas en el algoritmo propuesto | -Informe de calibración de las variables del algoritmo propuesto | -Revisión y validación al 100% del informe de calibración por parte de un especialista de algoritmia metaheurística. |

| **O3:** Comparar el algoritmo memético propuesto y un algoritmo genético adaptado por el tesista, con el fin de obtener resultados similares o mejores en términos de eficiencia y calidad de soluciones generadas | | |
|---|---|---|
| **Resultado** | **Medio de verificación** | **Indicador objetivamente verificable** |
| **R3.1:** Codificación de algoritmo genético con el cual se realizará la comparación | -Código fuente y ejecutable del algoritmo genético | -Pruebas unitarias exitosas del algoritmo genético. |

| | | |
|---|---|---|
| **R3.2:** Módulo de software para comparación de los algoritmos | -Código fuente y ejecutable del módulo de comparación | -Pruebas unitarias exitosas del módulo de comparación.<br><br>-Pruebas de integración exitosas del módulo de comparación con ambos algoritmos |
| **R3.3:** Experimentación numérica en base a los resultados del módulo de comparación | -Informe de experimentación numérica sobre la comparación entre el rendimiento de ambos algoritmos | -Revisión y validación al 100% del informe de experimentación numérica por parte de un especialista en algoritmos. |

## 2.5. Métodos y procedimientos

En la **tabla 2** se listan las herramientas, métodos y procedimientos a utilizar para cumplir con los resultados esperados desarrollados en las secciones anteriores.

*Tabla 2. Herramientas, métodos y procedimientos por resultado*

| **Resultado esperado** | **Herramientas** | **Metodologías, métodos y procedimientos** |
|---|---|---|
| **R1.1:** Variables, parámetros y restricciones que serán consideradas en el algoritmo propuesto. | No aplica | No aplica |
| **R1.2:** Diseño de estructuras de datos para representar las variables y restricciones descritas | No aplica | No aplica |
| **R1.3:** Función objetivo utilizada en el algoritmo propuesto | -Generador de datos aleatorios | No aplica |
| **R2.1:** Pseudocódigo del algoritmo propuesto | No aplica | No aplica |
| **R2.2:** Codificación del algoritmo propuesto | -Java | -Metodología Kanban<br>-Pruebas unitarias |
| **R2.3:** Calibración de variables | No aplica | No aplica |

| | | |
|---|---|---|
| utilizadas en el algoritmo propuesto | | |
| **R3.1:** Codificación de algoritmo genético con el cual se realizará la comparación | -Java | -Metodología Kanban<br>-Pruebas unitarias |
| **R3.2:** Módulo de software para comparación de los algoritmos | -Java | -Metodología Kanban<br>-Pruebas unitarias<br>-Pruebas de integración |
| **R3.3:** Experimentación numérica en base a los resultados del módulo de comparación | -R, RStudio | -Pruebas de significación estadística |

### 2.5.1. Herramientas

#### 2.5.1.1. Java

Java es un lenguaje de programación orientado a objetos, fuertemente tipado y que permite crear programas basados en clases que cumplan diferentes propósitos (Oracle, s.f.). Este lenguaje y la implementación más utilizada de la plataforma sobre la que funciona son mantenidos y actualizados en la actualidad por la compañía Oracle. Este soporte y la familiaridad del que escribe con el lenguaje son las razones principales por las que se decidió utilizarlo en desarrollo de la presente tesis.

#### 2.5.1.2. R y RStudio

R es un lenguaje y un entorno para la creación de gráficos y computación estadística. Este promete ser altamente extensible y brindar una amplia variedad de técnicas gráficas y estadísticas (R Project, s.f.).

RStudio es un entorno de desarrollo integrado o IDE para R compatible con todas las plataformas principales y que provee

una amplia variedad de funcionalidades que prometen aumentar la productividad del usuario (RStudio, 2021).

Se decidió utilizar R (y RStudio como IDE) debido a que facilitará la realización de la experimentación numérica gracias a los recursos estadísticos que provee y a que, nuevamente, el que escribe cuenta con cierto nivel de familiaridad con la herramienta.

### 2.5.2. Metodologías, métodos y procedimientos

#### 2.5.2.1. Metodología Kanban

Kanban es un término japonés que puede traducirse como "letrero", pero también hace referencia a un concepto nacido como un mecanismo de control de flujo dentro de un sistema de producción "*Just-in-time*" de Toyota en la década de 1950. En el mundo del desarrollo de software, Kanban como un método remonta sus orígenes al año 2004, cuando Microsoft le solicitó a David J. Anderson que apoye a un equipo pequeño de TI para que mejore su rendimiento. Este promueve la división del trabajo en tareas simples pero bien definidas y que añaden valor al producto final. Esto permite a los equipos estimar tiempos, organizar el trabajo y limitar las tareas en proceso en un determinado punto del tiempo (Ahmad O. et al., 2013; Corona E., Pani F. E., 2013; Kirovska N., Koceski S., 2015). Se considera que adoptar los principios de descomposición, control y limitación de trabajo de Kanban será beneficioso para la gestión y seguimiento del proceso de implementación de los algoritmos en el presente trabajo de tesis.

#### 2.5.2.2. Pruebas unitarias

Las pruebas de software pueden definirse como el proceso de ejecutar un programa con el fin de encontrar errores. Esto

implica una ejecución sistemática de dicho programa bajo una serie de circunstancias controladas o casos de prueba (con entradas, precondiciones y salidas esperadas definidas). El propósito de las pruebas es evaluar la calidad del software, mejorarla y asegurarse de que este cumple requerimientos específicos.

Las pruebas de software son de diferentes tipos, dependiendo de factores como la escala y las situaciones en las que se realizan. Uno de estos tipos es la llamada prueba unitaria o *unit test*. Esta implica probar las "piezas" más básicas o elementales del software que no sean triviales (pueden ser funciones, métodos, clases, etc.), las cuales se denominan unidades o módulos. En este sentido, permiten identificar errores en los niveles más bajos del software (Luo L., 2001; Jovanović I., 2006). Esta minuciosidad será beneficiosa para asegurar que los algoritmos desarrollados funcionen como se desea y para encontrar tempranamente posibles errores que puedan afectar el trabajo académico.

#### 2.5.2.3. Pruebas de integración

Habiéndose descrito lo que son las pruebas de software y su propósito en la sección de arriba, en la presente sección se hablará de otro tipo de pruebas: las de integración. Estas se realizan sobre las estructuras conformadas por múltiples módulos y sobre las interfaces que les permiten comunicarse entre sí (Luo L., 2001). Estas permiten comprobar el funcionamiento y detectar errores de los módulos cuando trabajan juntos en el software.

Las pruebas de integración serán útiles para detectar posibles errores en el módulo de comparación de los algoritmos, pues este se comunicará con cada algoritmo (los cuales son

unidades o módulos separados) para ejecutarlos y obtener datos de dichas ejecuciones.

#### 2.5.2.4. Pruebas de significación estadística

La técnica estadística de pruebas de hipótesis utiliza pruebas de significación estadística para determinar la probabilidad de que una afirmación o hipótesis realizada sobre una o más muestras de datos sea verdadera, y también establecer a qué grado de probabilidad dicha hipótesis será considerada realmente verdadera. El uso de estas pruebas de significación implica desarrollar las hipótesis nula y alternativa (generalmente sobre el comportamiento o características de la o las muestras) y establecer un nivel de significación $\alpha$ (que representa la probabilidad de que rechacemos la hipótesis nula cuando ésta es verdadera).

Existen diferentes pruebas de significación dependiendo de la hipótesis que se quiere probar (comportamiento de medias, comparación de medias o varianzas entre muestras, entre otros), teniendo cada una serie de requisitos para ser aplicada (siendo ejemplos la normalidad de una muestra, la igualdad de varianzas entre muestras, etcétera) (Massey A., Miller S. J., 2006).

Las pruebas de significación estadística tendrán un rol crítico en la experimentación numérica para realizar pruebas de hipótesis de comparación de tiempos de ejecución y costos de solución.

# 3. Marco Conceptual

## 3.1. Introducción

En la presente sección se describirán conceptos relevantes para la problemática abordada en el trabajo de tesis. Así, se hablará de bases de datos relacionales, consultas SQL, planes de ejecución, optimización de consultas y algoritmos metaheurísticos.

## 3.2. Desarrollo del Marco

### 3.2.1. Base de Datos Relacional

Una base de datos es un conjunto de datos cuyo almacenamiento se rige por una serie de reglas (Abdalla M., Karabatak M., 2020). Para una base de datos relacional, los datos son organizados en estructuras llamadas tablas, las cuales cuentan con filas (que representan un registro de datos) y columnas (que representan atributos para un registro de datos). Diferentes tablas pueden presentar relaciones entre sí basadas en campos comunes. Esta clase de base de datos está centrada en el concepto de consistencia: se busca que los mismos datos estén disponibles en el mismo estado para todos los usuarios en un punto específico del tiempo (Oracle, s.f.; IBM Cloud Learn Hub, 2019). El acceso, comunicación, consulta y administración de bases de datos computarizadas suelen realizarse por medio de Sistemas de Administración de Bases de Datos o DBMS, por sus siglas en inglés.

La estructura y el foco en la consistencia de las bases de datos relacionales han producido que estas sean ampliamente utilizadas para cubrir necesidades de almacenamiento de datos en el mundo empresarial: según el ranking hecho en abril de 2021 de DB-Engines, iniciativa desarrollada por la compañía australiana de

consultoría “solid IT”, 7 de los 10 DBMS más populares siguen el modelo relacional (db-engines.com, 2021). Es por esto que la optimización de consultas en bases de datos relacionales es un campo de estudio importante en la actualidad, lo que dio a su vez paso al desarrollo del presente trabajo.

Un ejemplo simple de una base de datos relacional serían tres tablas conteniendo datos de clientes, productos y pedidos de una pequeña empresa comercializadora. Cada tabla tiene un campo de ID que hace de llave, es decir, identifica de manera única cada registro de la tabla. La tabla de pedidos se relaciona con la de clientes al contener una columna con el ID del cliente que realizó el pedido, y también con la de productos, al contener otra columna con el ID del producto para el que se hizo el pedido.

### 3.2.2. Sistema de Administración de Base de Datos

Un Sistema de Administración de Base de Datos o DBMS por sus siglas en inglés (*Database Management System*) es un programa de software que sirve de interfaz entre una base de datos y usuarios u otros programas, facilitándoles la inserción, manipulación y consulta de los datos. También apoyan la administración y el control de la base de datos en sí, brindando funciones de monitoreo, configuración, respaldo y recuperación a administradores (Oracle, s.f.).

Si bien las bases de datos relacionales son ampliamente utilizadas en la actualidad, no todos los DBMS están dirigidos hacia este tipo de bases: también se han desarrollado sistemas de este tipo para bases de datos que no sigan el modelo relacional. Es más, hoy en día, el uso de DBMS multi-modelo (esto es, sistemas que controlan diferentes tipos de bases de datos en una plataforma unificada) está aumentando al punto de volverse una tendencia en el futuro (Lieponienė J., 2020). A continuación, se darán algunos

ejemplos de DBMS junto al tipo o tipos de base de datos a los que están dirigidos:

- Oracle Database: relacional, multi-modelo
- MongoDB: documentos, multi-modelo
- ElasticSearch: motor de búsqueda

Los DBMS se relacionan con el presente trabajo en el sentido en que la optimización de consultas SQL se realiza dentro de los DBMS de Bases de Datos relacionales.

### 3.2.3. Base de Datos Distribuida

El término Base de Datos Distribuida hace referencia a una arquitectura de base de datos en la que existen múltiples sitios separados físicamente, pero interconectados por una red en los cuales los datos se encuentran repartidos y/o replicados. Estas características pueden traer como ventajas mejoras de rendimiento, modularidad, accesibilidad y disponibilidad en contextos de volúmenes de datos grandes o sistemas de información también distribuidos (Liu S., Xu X., 2016; Panahi V., Navimipour N., 2019).

Un ejemplo teórico de una base de datos con esta arquitectura sería el siguiente: en la base de datos existen 10 tablas (T1 hasta T10) y se tienen 3 sitios diferentes. Estas 10 tablas se encuentran distribuidas entre los 3 sitios y tres de esas tablas están replicadas en al menos dos sitios. Se elaboró la **figura 2** para ilustrar esta estructura:

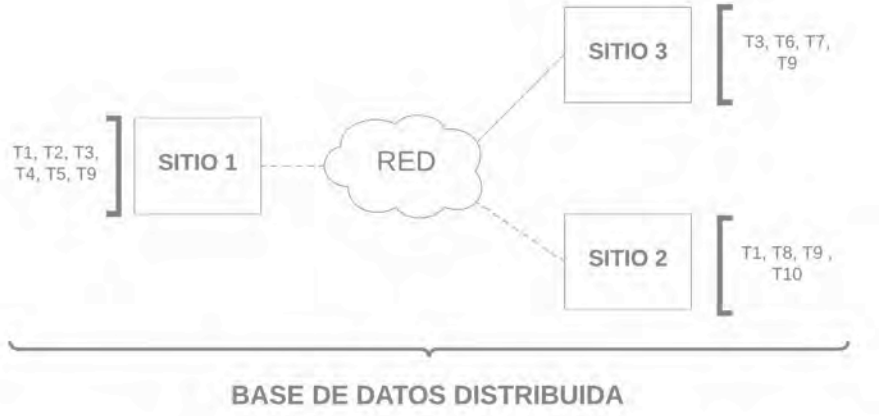



*Figura 2: Representación de BD distribuida*

Es importante tener en cuenta a las bases de datos distribuidas al hablar del problema de optimización de consultas en el presente trabajo, pues esta arquitectura trae consigo costos de transmisión de datos por la red, los cuales pueden afectar el rendimiento de una consulta.

### 3.2.4. Consulta SQL

En el contexto de bases de datos, una consulta o *query* hace referencia a una instrucción que un usuario envía al DBMS para obtener un conjunto de datos deseado. Para bases de datos relacionales, las consultas son escritas siguiendo la sintaxis del Lenguaje de Consultas Estructuradas o SQL (siguiendo las siglas de *Structured Query Language*). Entre el envío de una consulta y el retorno de los datos solicitados al usuario, ocurre el procesamiento de la consulta. A grandes rasgos, este proceso tiene tres pasos: en primer lugar, la consulta es analizada sintácticamente y traducida a una expresión de bajo nivel que el sistema puede entender con facilidad; en segundo lugar, esta expresión traducida pasa a ser optimizada, con lo que se decide la forma más eficiente de ejecutarla; y finalmente, se ejecuta el conjunto de operaciones necesarias para obtener los datos que desea el usuario de la forma

determinada en el paso anterior (Wijesiriwardana C., Firdhous M., 2019).

En su forma más simple, una consulta SQL se compone de tres operaciones: selección, proyección y *join*. La selección consiste en la obtención de un subconjunto de registros de datos de una tabla que cumplan con condiciones específicas. La proyección consiste en la obtención de un subconjunto de columnas o atributos de una tabla. La operación de *join* permite la unión de dos tablas relacionadas por medio de atributos compartidos (IBM, 2019). Esto se puede ilustrar en la **figura 3**:


*Figura 3: Descomposición de consulta SQL*

La consulta busca extraer datos de dos tablas: una de clientes y otra de países. La operación de proyección permite solo obtener el nombre, apellidos, edad y país de origen de los clientes en la base de datos. La operación de join permite unir las tablas de clientes y de países para obtener el nombre de un país en base al ID del mismo, un atributo común a las dos y por el cual se forma la relación entre ambas. Finalmente, la operación de selección solo retornará clientes cuya edad sea igual a 35 años.

Dentro del contexto de bases de datos relacionales, el presente trabajo girará en torno a mejorar el rendimiento de las consultas SQL.

### 3.2.5. Plan de Ejecución

Un plan de ejecución de consulta (QEP, siglas del término en inglés *Query Execution Plan*), también llamado plan de consulta, es una estructura que indica las operaciones que el DBMS realizará

para extraer los datos que solicita una consulta y el orden en que dichas operaciones se ejecutarán. En pocas palabras, un QEP indica el método a seguir para ejecutar una consulta (Abdalla M., Karabatak M., 2020). Cada operación del plan de consulta está relacionada a la extracción de registros de datos pertenecientes a tablas o a la preparación de dichos datos para enviarlos al usuario que envió la consulta. Según la Guía de Afinación de Consultas SQL en Oracle Database (Oracle, 2017), un plan de ejecución contiene o específica los siguientes aspectos:

- El orden de acceso y uso de las tablas relevantes en la consulta
- El método de acceso para cada tabla: se puede revisar todas las filas (método denominado *full scan*) o utilizar índices para ubicar filas específicas más rápidamente.
- El método de *join* para las tablas afectadas por operaciones de *join:* dependiendo del DBMS, pueden estar disponibles métodos que utilizan bucles anidados, tablas *hash*, entre otros.
- Operaciones de filtrado, ordenación y agregación de los datos
- Información sobre costos por operación, particionamiento y paralelismo de ejecución de operaciones

Para una misma consulta, es posible producir múltiples planes de ejecución factibles. Dichos planes son equivalentes semánticamente, es decir, generarán el mismo conjunto de datos, pero todos pueden diferir en su tiempo de ejecución y consumo de recursos (Chande S., 2019). Por ejemplo, para una consulta con tres tablas A, B y C, pueden existir dos planes de ejecución que difieran en el orden en el que operadores *join* son aplicados a las tres tablas: **A -> B -> C** versus **A -> C -> B**. Debido a factores como el tamaño de las tablas (o también su ubicación física, si hablamos de bases de datos distribuidas), el plan que utiliza el primer orden de *joins* tiene un tiempo de ejecución de 10 segundos, mientras que ejecutar el

otro plan tomaría 16 segundos. En este caso, es evidente que lo ideal sería utilizar el primer plan de ejecución.

Tras comentar lo anterior, podemos notar la relación entre el concepto de plan de ejecución de consulta y el presente trabajo: mejorar el rendimiento de una consulta SQL implica construir un plan de ejecución eficiente.

### 3.2.6. Complejidad Temporal/Espacial

En el contexto de algoritmos, los conceptos de complejidad temporal y espacial hacen referencia a medidas que buscan estimar el consumo de tiempo y unidades de memoria implicados en la ejecución de un determinado algoritmo en función del tamaño o longitud de los datos que se le brindan como entrada. De manera más específica, la complejidad temporal y espacial buscan dar una idea del crecimiento en el tiempo de ejecución y unidades de memoria ocupadas por un algoritmo cuando el tamaño de los datos de entrada aumenta. Este cálculo generalmente se realiza determinando el número de operaciones básicas que ejecuta el algoritmo en su ejecución. Sin embargo, los consumos de tiempo y memoria no solo dependen del tamaño de la entrada, sino también de sus características o naturaleza. Por ejemplo, para un algoritmo que busca un determinado valor en un arreglo de n números, sus tiempos de ejecución cuando el valor buscado se encuentra al inicio y al final del arreglo pueden ser muy diferentes. Es por esto que se plantean complejidades temporales para el mejor escenario, peor escenario y un escenario promedio por separado. Si bien las tres complejidades pueden ser utilizadas, generalmente la más útil es la complejidad en el peor escenario, pues permite acotar el consumo de recursos de un algoritmo. Así, cuando se habla de complejidad en general, generalmente nos referimos a complejidad en el peor escenario.

La acotación mencionada anteriormente se realiza con una nomenclatura asintótica llamada *big O*. Siendo T(n) una función que calcula la cantidad de operaciones básicas que realiza un algoritmo en base al tamaño n de su entrada, y g(n) una función cualquiera que también depende de n, se puede afirmar a grandes rasgos que T(n) (y por tanto, el tiempo y/o memoria consumidos por un algoritmo) pertenece a O(g(n)) si a partir de un determinado valor de n T(n) tiene un grado de crecimiento igual o menor que g(n) conforme n aumenta. Esto nos permite hacernos una idea de cómo puede crecer el tiempo o la memoria ocupada por un determinado algoritmo conforme crece el tamaño del problema a resolver (Levitin A., 2012). A continuación, se muestran complejidades temporales de algoritmos que resuelven ciertos problemas:

- Calcular el valor de la mediana en un arreglo ordenado de tamaño n: **O(1)**
- Encontrar el valor máximo en un arreglo no ordenado de tamaño n: **O(n)**

Los conceptos de complejidad temporal y espacial resultan importantes en el presente trabajo porque se busca hacer frente al problema de optimización de consultas con una solución tan o más eficiente que otros métodos ya existentes.

### 3.2.7. Problema *NP-Hard*

*NP-Hard* es un término utilizado para clasificar ciertos problemas de acuerdo a su dificultad o complejidad. En realidad, *NP-Hard* va acompañado de otras clases de complejidad, así que resulta conveniente describir algunas de estas clases brevemente.

En primer lugar, tenemos la clase P (cuyo nombre hace referencia a *Polinomial Time Complexity*), en la que se encuentran todos los problemas de decisión (un problema de optimización

puede ser transformado en uno de decisión) que pueden ser resueltos utilizando un algoritmo determinístico cuya complejidad de tiempo en el peor escenario es polinomial ($O(n^k)$, k es una constante >1). En segundo lugar, tenemos la clase NP (cuyo nombre hace referencia a *Nondeterministic Polinomial Time Complexity*), que incluye a todos los problemas de decisión cuyas soluciones pueden ser verificadas por un algoritmo no determinístico cuya complejidad de tiempo en el peor escenario es polinomial. En este sentido, P es un subconjunto de NP (pues si existe un algoritmo de complejidad polinomial capaz resolver un problema P, también existirá un algoritmo con complejidad polinomial capaz de verificar soluciones para dicho problema). Puede afirmarse así que los problemas NP que no están en P son complicados, pues no se han encontrado algoritmos de complejidad polinomial capaces de resolverlos.

En este contexto, se puede describir la clase *NP-Hard*: se dice que un problema es *NP-Hard* si un algoritmo capaz de resolverlo puede ser reducido en tiempo polinomial a otro algoritmo capaz de resolver cualquier problema NP. En este sentido, se sabe que los problemas *NP-Hard* son al menos tan difíciles como cualquiera de los problemas NP (Du K., Swamy M., 2016). Algunos ejemplos de problemas *NP-Hard* son:

- Problema del vendedor ambulante
- Problema para encontrar la ruta más larga entre dos nodos de un grafo

Resulta importante conocer estas diferentes clases para ser conscientes de cuán complejo puede ser el problema de optimización de consultas.

### 3.2.8. Optimización de Consultas SQL

Como se mencionó líneas más arriba, la optimización de consultas es uno de los pasos dentro del procesamiento de las

mismas. Considerando la existencia de múltiples QEP para una consulta, la optimización consiste en una tarea de búsqueda del mejor plan de ejecución dentro del conjunto de planes posibles equivalentes (Venkata S., Vatsavayi V., 2017).

En los DBMS, la optimización de consultas es responsabilidad de un módulo de software llamado, valga la redundancia, optimizador de consultas. Este está compuesto por tres componentes: un espacio de búsqueda, que hace referencia a un conjunto de QEPs equivalentes generados para una consulta; un modelo de costos, que permite la asociación de un costo a un plan QEP, generalmente mediante una función de costos; y una estrategia de búsqueda que se encarga de explorar el espacio de búsqueda y comparar los QEPs para seleccionar el de menor costo (Sharma M. et al, 2016).

Una característica del espacio de búsqueda de planes de ejecución es que su tamaño crece exponencialmente de acuerdo a la cantidad de tablas utilizadas en la consulta, lo que reduce enormemente la viabilidad de explorar el espacio completo para consultas largas y/o complejas. En este sentido, la búsqueda del mejor QEP en dicho espacio es considerada un problema de optimización *NP-hard* (Mohsin S. et al., 2021).

El presente trabajo, al buscar mejorar el rendimiento de consultas SQL, está buscando por consiguiente plantear un modelo de costos y una estrategia de búsqueda capaz de encontrar buenos planes de ejecución en espacios de búsqueda de gran tamaño.

Un ejemplo simple de un optimizador de consultas consiste en un modelo de costos que toma en cuenta la capacidad de procesamiento del servidor de base de datos y los tamaños de las tablas utilizadas en la consulta para plantear una función de costo, y una estrategia de búsqueda que explore todos los planes de

ejecución del espacio de búsqueda de manera secuencial para determinar el de menor costo.

### 3.2.9. Método Exhaustivo

En el ámbito de técnicas de búsqueda, los métodos exhaustivos o de fuerza bruta hacen referencia a algoritmos que evalúan todas las posibles soluciones a un problema y escogen la que satisface las condiciones del mismo y/o lo hace de manera óptima. Esto garantiza que, al terminar su ejecución, los algoritmos de búsqueda exhaustiva habrán encontrado siempre la mejor solución existente para el problema que atacan, pero también implica una alta complejidad temporal y espacial en el peor escenario (exponencial en ambos casos) (Mohsin S. et al., 2021). Otros factores que pueden promover el uso de estos métodos son su capacidad para atacar una variedad muy amplia de problemas y que existan problemas relativamente sencillos en los que no valga la pena implementar algoritmos más eficientes, pero complicados (Levitin A., 2012).

Los métodos exhaustivos de búsqueda se relacionan con la problemática en el sentido de que algoritmos de este tipo han sido utilizados anteriormente para la optimización de consultas en DBMS, pero con el aumento en la complejidad de consultas actual y el alto grado de crecimiento del espacio de búsqueda respecto a dicha complejidad, se han convertido en estrategias no viables (Mohsin S. et al., 2021). Algunos ejemplos de algoritmos exhaustivos son:

- Programación Dinámica / *Dynamic Programming*
- Ramificación y Poda / *Branch and Bound*
- *Backtracking*

### 3.2.10. Algoritmo Metaheurístico

Las metaheurísticas son un tipo de algoritmos inteligentes que encuentran soluciones cercanas al óptimo para problemas de optimización complicados. De manera conceptual, una metaheurística es un procedimiento de alto nivel capaz de determinar o construir una heurística (definida como una técnica basada en la experiencia para resolver problemas) de nivel más bajo que sea capaz de brindar soluciones suficientemente buenas para problemas de optimización. Para esto se basan en paradigmas inspirados en la naturaleza y en técnicas estocásticas o probabilísticas.

Al buscar entre un conjunto grande de soluciones factibles, los algoritmos metaheurísticos suelen encontrar buenas soluciones con una menor complejidad computacional que las de técnicas basadas en cálculos o en heurísticas (Du K., Swamy M., 2016). Considerando estas características, esta clase de algoritmos supone un campo de estudio importante para afrontar la optimización de consultas en el presente trabajo.

Los algoritmos heurísticos pueden clasificarse en diferentes grupos dependiendo de los procesos o comportamientos que reproduzcan. Algunos de estos grupos son:

- Basados en conceptos evolutivos, como por ejemplo los algoritmos genéticos
- Basados en conceptos científicos, siendo un ejemplo el algoritmo de recocido simulado o simulated annealing
- Basados en conceptos de inteligencia colectiva, siendo ejemplos los algoritmos de colonia de hormigas y de abejas artificiales

### 3.2.11. Algoritmo Genético

Un algoritmo genético es una metaheurística evolutiva, es decir, que sigue el paradigma neo-Darwiniano basado en la teoría de la evolución, la selección natural, y la genética de Mendel. Este algoritmo supone una buena estrategia para problemas de optimización gracias a su uso de poblaciones cuyos individuos ayudan a producir nuevas generaciones, cambiando así la población iterativamente.

De manera específica, cada individuo de la población es considerado un cromosoma, el cual representa una solución posible al problema atacado. Un cromosoma a su vez está compuesto por genes que determinan las características de esa solución específica. A su vez, cada cromosoma también posee un nivel o grado de *fitness* o adecuación, el cual determina la calidad de la solución respecto al modelo de costos planteado para el problema. Esta adecuación es utilizada tanto para determinar cuál es el mejor cromosoma en una determinada generación y también para aplicar operadores genéticos de selección, mutación y cruce, los cuales son los responsables de producir nuevos cromosomas en base a la población actual. Generalmente estos operadores incluyen lógicas probabilísticas o aleatorias para alterar cromosomas. Finalmente, el algoritmo genético termina al alcanzar una condición de parada, siendo algunos ejemplos un número máximo de generaciones o la falta de mejoras de *fitness* para los individuos en las últimas generaciones (Du K., Swamy M., 2016).

Existen múltiples problemas para los que se han aplicado algoritmos genéticos, generalmente en las ramas de búsqueda u optimización. Algunos ejemplos reales incluyen la planificación de tiempos en sistemas de manufactura (Ławrynowicz A., 2011) y

posicionamiento de espejos que reflejen luz a un punto en un sistema de energía solar (Gross B., 2003).

Los algoritmos genéticos se relacionan con la problemática y el presente trabajo al haber sido bastante estudiados para atacar la optimización de consultas.

### 3.2.12. Algoritmo Memético

Un algoritmo memético, también llamado búsqueda local genética, es también una metaheurística evolutiva que trabaja con poblaciones y cuya lógica parte del concepto del meme, el cual es la unidad básica de transmisión o imitación cultural. Así, se le puede catalogar como parte de los llamados algoritmos culturales. En un algoritmo memético se busca combinar una fase de evolución de una población con otra de aprendizaje individual. Así, es posible considerar al algoritmo memético como un algoritmo evolutivo al que se le suma búsqueda local. La fase de evolución de la población puede seguir una lógica similar o idéntica a la de un algoritmo genético, mientras que no es raro que la fase de búsqueda local se implemente con la lógica de algoritmos como *Hill Climbing, Simulated Annealing* o Búsqueda Tabú (Du K., Swamy M., 2016; Ryan C., 2003). Estas posibles combinaciones podrían evitar convergencias prematuras a un óptimo local en el espacio de soluciones del problema, como podría ocurrir al utilizarse un algoritmo genético. (Garg P., 2009).

Se han utilizado algoritmos meméticos para resolver una amplia gama de problemas. Algunos ejemplos son el reconocimiento de patrones (Aguilar J., Colmenares, A.,1998), la selección de variables de objetivo múltiple (Karkavitsas G. V., Tsihrintzis G. A., 2011), y el diseño de circuitos (Harris S. P., Ifeachor E. C., 1998).

El algoritmo memético se relaciona con el presente proyecto en el sentido en que se realizará la adaptación del mismo para atacar el problema de optimización de consultas.

# 4. Estado del Arte

## 4.1. Introducción

En la presente sección se presentan objetivos y la aplicación de una serie de estrategias relacionadas a la revisión de la literatura para el trabajo de tesis. Este proceso es de gran importancia para un trabajo académico, pues permite al investigador expandir y afianzar sus conocimientos sobre el tema en cuestión, y además presenta una oportunidad para aprovechar los avances y conocimientos brindados por la comunidad científica para desarrollar nuevas conclusiones y resultados.

## 4.2. Objetivos de Revisión

En el trabajo de tesis se realizará una revisión empírica, pues resulta vital analizar las experiencias y resultados de diferentes autores para obtener una perspectiva amplia de las soluciones existentes hoy en día para el problema propuesto, la que a su vez servirá de guía en el nacimiento de otra alternativa de solución. Así, se plantean los siguientes objetivos:

- Comprender los conceptos involucrados en la optimización de consultas SQL en Bases de Datos Relacionales (RDB).
- Identificar y comprender las diferentes propuestas de solución basadas en algoritmos metaheurísticos para el problema de optimización de consultas presentadas en los últimos años, teniendo en cuenta sus diferencias, ventajas y desventajas.
- Identificar las variables y restricciones relevantes para el problema.
- Identificar y comprender los principales métodos para el análisis de la calidad de las soluciones encontradas.

## 4.3. Preguntas de Revisión

En base a los objetivos planteados, se procede a aplicar el criterio PICOC (*Population, Intervention, Comparison, Outcome, Context*) para formular preguntas que guíen el proceso de investigación:

*Tabla 3. Desarrollo de criterio PICOC*

| **Criterio** | **Aplicación** |
|---|---|
| Población (*Population)* | Consultas SQL ineficientes en RDB |
| Intervención (*Intervention*) | Algoritmo metaheurístico/bioinspirado de optimización de consultas SQL |
| Comparación (*Comparison*) | Soluciones algorítmicas metaheurísticas/bioinspiradas más conocidas en la actualidad |
| Resultado (*Outcome)* | Casos de estudio que propongan algoritmos metaheurísticos/bioinspirados de optimización de consultas SQL |
| Contexto (*Context)* | Académico |

Considerando este criterio, se plantearon las siguientes preguntas de revisión:

**PR1:** ¿Cómo funcionan los algoritmos metaheurísticos que se han utilizado en los últimos años para resolver problemas de optimización de consultas SQL y por qué han sido propuestos?

**PR2:** ¿Cuáles son los factores más importantes en la optimización de consultas SQL y cómo afectan a los algoritmos en la optimización de las mismas?

**PR3:** ¿Cuáles son los principales criterios y métricas utilizados para determinar la eficacia y eficiencia de los algoritmos que optimizan las consultas SQL y cómo se utilizan para esta tarea?

## 4.4. Estrategia de Búsqueda

### 4.4.1. Motores de búsqueda a usar

Se utilizarán los siguientes motores para la revisión:

- IEEEXplore
- Scopus
- ACM Digital Library

### 4.4.2. Cadenas de búsqueda a usar

El desarrollo de los anteriores puntos debe traducirse a la producción de cadenas de búsqueda para los motores mencionados líneas arriba con el fin de asegurar búsquedas que nos provean de resultados útiles para poder responder las preguntas y cumplir los objetivos de la revisión. En la **tabla 4** se define una serie de conceptos clave alrededor de los cuales girará la revisión, y sus respectivos términos relacionados:

*Tabla 4. Términos relacionados a cada concepto*

| **Concepto** | **Términos relacionados** |
|---|---|
| **CC1:** Bases de Datos Relacionales | Relational Database, RDB, Distributed Relational Database, Database |
| **CC2:** Consultas SQL | SQL, query, SQL query, select statement, join |
| **CC3:** Optimización | Optimization, performance, efficiency, execution time, response time |
| **CC4:** Algoritmos metaheurísticos/bioinspirados | (metaheuristic, bioinspired, memetic, genetic, ant colony, bat, grey wolf, evolutionary, particle swarm, taboo, bee) + algorithm |

Se desea que la cadena de búsqueda incluya todos estos conceptos y, por lo tanto, la misma se estructurará tomando en cuenta los términos relacionados escogidos:

**CC1 AND CC2 AND CC3 AND CC4**

Esta misma se presentará en la sintaxis utilizada por cada motor de búsqueda escogido:

- IEEEXplore:
  (RDB OR "relational database" OR "distributed relational database" OR database) AND (SQL OR query OR "sql query" OR “select statement” OR join) AND (optimization OR performance OR efficiency OR "execution time" OR "response time") AND ((bioinspired OR metaheuristic OR memetic OR genetic OR “ant colony” OR bat OR “grey wolf” OR evolutionary OR “particle swarm” OR taboo OR bee) AND algorithm)
- Scopus:
  (TITLE-ABS-KEY(RDB) OR TITLE-ABS-KEY("relational database") OR TITLE-ABS-KEY("distributed relational database") OR TITLE-ABS-KEY(database)) AND (TITLE-ABS-KEY(SQL) OR TITLE-ABS-KEY(query) OR TITLE-ABS-KEY("sql query") OR TITLE-ABS-KEY(“select statement”) OR TITLE-ABS-KEY(join)) AND (TITLE-ABS-KEY(optimization) OR TITLE-ABS-KEY(performance) OR TITLE-ABS-KEY(efficiency) OR TITLE-ABS-KEY("execution time") OR TITLE-ABS-KEY("response time")) AND ((TITLE-ABS-KEY(bioinspired) OR TITLE-ABS-KEY(metaheuristic) OR TITLE-ABS-KEY(memetic) OR TITLE-ABS-KEY(genetic) OR TITLE-ABS-KEY(“ant colony”) OR TITLE-ABS-KEY(bat) OR TITLE-ABS-KEY(“grey wolf”) OR TITLE-ABS-KEY(evolutionary) OR TITLE-ABS-KEY(“particle swarm”) OR TITLE-ABS-KEY(taboo) OR TITLE-ABS-KEY(bee)) AND TITLE-ABS-KEY(algorithm))
- ACM Digital Library:

AllField:(RDB "relational database" "distributed relational database") AND AllField:("sql query" "select statement" join) AND AllField:(optimization performance efficiency "execution time" "response time") AND AllField:(metaheuristic bioinspired memetic genetic "ant colony" bat "grey wolf" evolutionary "particle swarm" taboo bee) AND AllFIeld:(algorithm)

### 4.4.3. Documentos encontrados por motor

Al utilizar la cadena de búsqueda propuesta en cada motor, y añadiendo un filtro de antigüedad de 6 años, se obtuvo lo siguiente:

*Tabla 5. Resultados encontrados por motor de búsqueda*

| **Motor de búsqueda** | **Cantidad de resultados** |
|---|---|
| IEEEXplore | 52 |
| Scopus | 254 |
| ACM Digital Library | 300 |
| **TOTAL** | 606 |

### 4.4.4. Criterios de inclusión/exclusión

En esta sección se detallarán los criterios de inclusión y exclusión elegidos para evaluar la adecuación de los resultados obtenidos en la búsqueda. Estos determinarán si un trabajo literario es considerado para su análisis a detalle o es descartado por no estar directamente relacionado a lo que se desea investigar.

#### 4.4.4.1. Criterios de inclusión

- El trabajo se centra en bases de datos, y no las usa como medio o componente para otro fin, pues esto se desvía del foco de la investigación.

- El trabajo considera bases de datos distribuidas: con consultas remotas y locales. En la investigación se busca evaluar la optimización para este tipo de consultas.
- El trabajo experimenta con motores de RDBs como Oracle, MicrosoftSQL, DB2, Postgres, MySQL, entre otros. La investigación no busca discriminar entre estos motores.

#### 4.4.4.2. Criterios de exclusión

- Si el trabajo presenta una solución para el problema de optimización de consultas SQL, esta no es un algoritmo metaheurístico o bioinspirado. Propuestas de cualquier otro tipo no aportarían en la elección y desarrollo de un nuevo algoritmo metaheurístico de optimización.
- La propuesta de solución del trabajo está basada en *machine* o *deep learning*. Esta clase de soluciones no son el foco de la investigación
- El trabajo discute mejoras en rendimiento en bases de datos por medio de ajustes de hardware o redes. Estos resultados no son relevantes porque no buscan atacar el problema planteado.
- El trabajo se centra en bases de datos NOSQL, ANYDB, OLAP, SPARK, entre otros. Estos tipos de base de datos no son el foco de la investigación.
- El trabajo no se encuentra redactado en inglés o español, pues esos son los lenguajes que el que escribe domina.
- El trabajo no fue publicado en los últimos 6 años. Solo se buscan propuestas recientes para estar seguros de que aún mantienen relevancia o superioridad en la actualidad.
- El acercamiento de optimización de consultas SQL está centrado en optimizar el orden de ejecución de un grupo de consultas independientes. El foco del estudio es la

optimización de una sola consulta, no determinar cómo ejecutar varias separadas.

- El trabajo solo busca hacer un resumen de la aplicación de un determinado algoritmo en la optimización de consultas, y no realiza un planteamiento detallado del problema de optimización. Un trabajo así no ayudaría a responder ninguna de las tres preguntas como literatura primaria.
- Los trabajos consideran bases de datos cifradas. Los mecanismos de encriptación y el foco en la seguridad y privacidad de los datos se alejan del foco de la investigación.

## 4.5. Formulario de Extracción

En la **tabla 6** se propone un formulario de extracción para el trabajo:

*Tabla 6. Estructura de formulario de extracción*

| **Campo** | **Descripción/posibles valores** | **Pregunta relacionada** |
|---|---|---|
| ID | Identificador alfanumérico del trabajo | No aplica |
| Fecha de extracción | Fecha en la que se encontró el trabajo | No aplica |
| Autor(es) | Nombre(s) de el o los autores | No aplica |
| Título | Título del trabajo | No aplica |
| Fuente literaria | Nombre de la revista, publicación o congreso en el que el trabajo fue publicado | No aplica |
| Año | Año de publicación | No aplica |
| Motor de búsqueda | Nombre del motor de búsqueda en el que se encontró el trabajo | No aplica |
| Enlace | URL del motor de búsqueda que corresponde | No aplica |

| | | |
|---|---|---|
| | al trabajo | |
| Resumen algoritmo propuesto | ¿Cómo funciona el algoritmo planteado para resolver el problema de optimización? | PR1 |
| Razón de elección de algoritmo | ¿Por qué se consideró que el algoritmo propuesto sería efectivo para enfrentar el problema de optimización? | PR1 |
| Factores para optimización de consultas | ¿Qué factores o conceptos considera importantes en la optimización de consultas SQL y cómo la afectan? | PR2 |
| Métricas de eficacia/eficiencia | ¿Qué criterios o métricas toman en cuenta para determinar la eficiencia del algoritmo y de la solución? | PR3 |
| Propósito de métricas | ¿Cómo se utilizan las métricas para determinar la calidad del algoritmo? | PR3 |

El formulario de extracción lleno en una hoja de excel puede encontrarse accediendo al enlace en el **anexo A.**

## 4.6. Resultados de la Revisión

Partiendo del total de 606 resultados, se procedió a descontar trabajos duplicados en ambos motores y a aplicar los criterios de inclusión y exclusión mencionados líneas más arriba sobre títulos y abstracts. Con estas medidas, la cantidad de resultados relevantes se redujo a 29. A continuación, se retiraron los trabajos para los que no se pudo obtener el texto completo, quedando así 19 resultados. Tras una segunda revisión y aplicación de criterios de inclusión/exclusión considerando el contenido de los trabajos, este número se redujo a 13 resultados, habiéndose encontrado un duplicado más con diferente

título y publicado en una diferente revista. Así, la división de resultados se puede observar en la **Tabla 7**:

*Tabla 7. Resultados finales por motor de búsqueda*

| **Motor de búsqueda** | **Cantidad de resultados** |
|---|---|
| IEEEXplore | 9 |
| Scopus | 4 |
| ACM Digital Library | 0 |
| **TOTAL** | 13 |

En la **Tabla 8** se muestra un listado de los trabajos académicos revisados y se indica las preguntas que ayudan a responder:

*Tabla 8: Listado de trabajos académicos revisados*

| **ID** | **Título** | **Autor(es)** | **Año** | **Preguntas relacionadas** |
|---|---|---|---|---|
| IE01 | Design and Analysis of Stochastic Query Optimizer for Biobank Databases | Sharma M, Singh G, Singh R, Singh J | 2015 | **PR1, PR2, PR3** |
| IE02 | Parametric Analysis of Different GA based Distributed DSS Query Optimizer Models | Sharma M, Singh G, Singh R | 2016 | **PR1, PR2, PR3** |
| IE03 | Query Optimization of Distributed Database Based on Parallel Genetic Algorithm and Max-Min Ant System | Ban W, Lin J, Ton J, Li S | 2016 | **PR1, PR2, PR3** |
| IE04 | To Review and Compare Evolutionary Algorithms in Optimization of Distributed Database Query | Abdalla M, Karabatak M | 2020 | **PR2, PR3** |
| IE05 | A distributed database query optimization method based on genetic algorithm and immune theory | Yao M | 2018 | **PR1, PR2, PR3** |
| IE06 | Distributed Database Query Based on Improved Genetic Algorithm | Liu S, Xu X | 2016 | **PR1, PR2, PR3** |

| | | | | |
|---|---|---|---|---|
| IE07 | Database Query Optimization Based on Parallel Ant Colony Algorithm | Zheng W, Jin X, Deng F et al | 2018 | **PR1, PR2, PR3** |
| IE08 | QIACO: A Quantum Dynamic Cost Ant System for Query Optimization in Distributed Database | Mohsin S, Darwish S, Younes A | 2021 | **PR1, PR2, PR3** |
| IE09 | The Multi-join Query Optimization for Smart Grid Data | Han Y, Miao Y, Zhang D | 2016 | **PR1, PR2, PR3** |
| SC01 | Database query optimization using genetic algorithms: A systematic literature review | Chande S | 2019 | **PR2** |
| SC02 | Generating distributed query plans using modified cuckoo search algorithm | Vijay Kumar T, Yadav M | 2017 | **PR1, PR2, PR3** |
| SC03 | Join query optimization in the distributed database system using an artificial bee colony algorithm and genetic operators | Panahi V, Navimipour N | 2019 | **PR1, PR2, PR3** |
| SC04 | Teacher-learner & multi-objective genetic algorithm based query optimization approach for heterogeneous distributed database systems | Venkata Lakshmi S, Vatsavayi V | 2017 | **PR1, PR2, PR3** |

Tras listar los trabajos estudiados y su respectiva relación a las preguntas de revisión, se procede a responder dichas preguntas con lo aprendido:

**PR1: ¿Cómo funcionan los algoritmos metaheurísticos que se han utilizado en los últimos años para resolver problemas de optimización de consultas SQL y por qué han sido propuestos?**

Entre los algoritmos utilizados en los últimos años para resolver el problema planteado en el presente trabajo, uno de los más estudiados es el Algoritmo Genético (GA, de *Genetic Algorithm*). Su lógica se basa en la creación de una población de cromosomas, estructuras que representan una posible solución para el problema planteado, la cual va cambiando de manera

iterativa mediante la aplicación de operadores genéticos (selección de padres, cruce de genes de padres para producir hijos y mutación de hijos) en lo que se denominan generaciones. Estos cambios buscan explorar diferentes posibles soluciones y aprovechar las características de buenos cromosomas para las generaciones futuras. Entre las razones que promueven la elección del GA para su estudio en el problema de optimización de consultas tenemos la evidencia académica/experimental de su eficacia para otros problemas de optimización complejos, su robustez y adaptabilidad en espacios de búsqueda muy grandes, entre otros.

Los factores principales que mantienen al GA como una solución viable para la optimización de consultas SQL son:

- Alteraciones e innovaciones en las lógicas de los operadores genéticos, cambiando parámetros como probabilidades de selección/cruce/mutación o incluyendo nuevas variables como la Entropía de la Teoría de la Información (Sharma M. et al, 2015; Sharma M. et al, 2016).
- Hibridización con otros algoritmos: se han realizado propuestas de solución que combinan la lógica del GA con la de otros algoritmos o en las que los resultados de uno de ellos son utilizados como entradas para el otro. Ejemplos de algoritmos híbridos incluyen el *Max-Min Ant System* o MMAS (Ban W. et al, 2016), el algoritmo inmune (Yao M., 2018), *Fuzzy C-means Clustering* o FCM (Liu S., Xu X., 2016), el algoritmo de colonia de abejas artificiales o ABC (Panahi V., Navimipour N., 2019), el algoritmo *Teacher-Learner* (Venkata S., Vatsavayi V., 2017) y el uso conjunto de algoritmos Guo Tao y *Particle Swarm* (Han Y. et al., 2016).

Otra metaheurística que, si bien no es tan estudiada como el GA, ha demostrado lograr soluciones de calidad en la optimización de consultas es el algoritmo Cuco. Su razón de estudio proviene de la evidencia de su eficiencia y eficacia para resolver problemas de optimización como el vendedor viajero. El algoritmo Cuco se basa en el uso de huevos de cuco para representar soluciones de optimización. De manera iterativa, se escogen fracciones de los

mejores y peores huevos: de los mejores se generan huevos de similar calidad que sustituirán de manera aleatoria a individuos de la población, y se busca transformar a los peores en huevos de mejor calidad. Estos cambios y transformaciones forman parte de un proceso conocido como vuelo de Lévy. Una modificación factible para este algoritmo consiste en el uso de valores y ecuaciones de vuelo de Lévy diferentes para los mejores y los peores huevos (Vijay T., Yadav M., 2017).

Un último ejemplo de algoritmos que en los últimos años han ofrecido buen rendimiento y resultados en el problema planteado es la Optimización de Colonia de Hormigas o *Ant Colony Optimization* (ACO). Este algoritmo también tiene una buena reputación en la resolución de problemas de optimización complejos. Su lógica se basa en la capacidad de las colonias de hormigas para encontrar las rutas más cortas a un determinado lugar. En el algoritmo se tiene el espacio de búsqueda planteado como diferentes nodos interconectados por caminos. Una ruta a través de estos nodos representa una solución de optimización. Las hormigas abstractas en el algoritmo se dedican a explorar las rutas existentes en búsqueda de su destino, y en el proceso generan feromonas (cuya intensidad depende de la calidad de la ruta) que otras hormigas utilizarán como guía. A esto se incluye un factor de evaporación de las feromonas para evitar el estancamiento en óptimos locales. Unas modificaciones bastante recientes de esta propuesta consisten en:

- El uso de paralelismo: la colonia global se subdivide en varias sub-colonias que exploran las rutas de manera simultánea y que comparten información entre ellas (Zheng W. et al., 2018).
- La inspiración en el principio de superposición de sistemas cuánticos para mejorar la búsqueda en el espacio de soluciones: en vez de utilizarse estados rígidos para las decisiones de movimiento de las hormigas, se utilizan estados cuánticos que no son definitivos y un mecanismo denominado puerta de negación parcial cuántica (Mohsin S. et al., 2021)

**PR2: ¿Cuáles son los factores más importantes en la optimización de consultas SQL y cómo afectan a los algoritmos en la optimización de las mismas?**

Los esfuerzos de optimización de consultas SQL, debido a las nuevas necesidades de información de los últimos años, se han enfocado en los siguientes factores:

En primer lugar, tenemos que el uso cada vez mayor de bases de datos distribuidas (que permiten la repartición y replicación de datos en sitios conectados por una red y ubicados en diferentes lugares) implica tomar en cuenta en qué sitio se ubica cada tabla usada en la consulta y cuál es la mejor forma de transmitir dichos datos por la red de sitios hacia el usuario, pues encontrar sitios adecuados y una buena estrategia para la transmisión de datos resultará en una reducción del tiempo de ejecución de la consulta.

En segundo lugar, pero con la misma importancia, se tiene que la cantidad de tablas utilizadas en una consulta puede ser considerablemente grande. Así, toma fuerza el efecto de las operaciones de *join* entre tablas sobre la eficiencia de dichas consultas y crece la necesidad de determinar un orden de *joins* óptimo que cruce las fuentes de datos de la forma menos costosa.

Si bien existen trabajos que solo se centran en uno de los dos factores (Yao M., 2018; Liu S., Xu X.,2016; Zheng W. et al., 2018; Vijay T., Yadav M., 2017; Han Y. et al, 2016), se evidencia que ambos suelen ir de la mano al plantear el problema de optimización (Sharma M. et al, 2015; Sharma M. et al, 2016; Ban W. et al, 2016; Abdalla M., Karabatak M., 2020; Chande S.,2019; Panahi V., Navimipour N., 2019; Mohsin S. et al., 2021; Venkata S., Vatsavayi V., 2017). Esto tiene sentido, pues en un contexto de datos repartidos y replicados en diferentes sitios, el uso de *joins* implica el uso de tablas remotas y la selección de sitios adecuados, y el orden de *joins* dictará cómo estos datos se transmiten entre dichos sitios.

Otro factor que ha sido discutido por ciertos trabajos en el contexto de las bases de datos distribuidas es el paralelismo intersitio e intrasitio. Se evalúa la posibilidad de ejecutar operaciones sobre los datos de manera simultánea en diferentes sitios y/o dentro de un mismo sitio con el fin de reducir el tiempo de ejecución de la consulta (Sharma M. et al, 2015; Sharma M. et al, 2016; Abdalla M., Karabatak M., 2020; Zheng W. et al., 2018).

**PR3: ¿Cuáles son los principales criterios y métricas utilizados para determinar la eficacia y eficiencia de los algoritmos que optimizan las consultas SQL y cómo se utilizan para esta tarea?**

De manera general, los costos de un determinado plan de ejecución de consulta se pueden dividir en Costos de Procesamiento Local (LPC) y Costos de Comunicación (Sharma M. et al, 2015; Sharma M. et al, 2016; Mohsin S. et al., 2021; Venkata S., Vatsavayi V., 2017). Los LPC se desglosan a su vez en costos de I/O y costos de CPU (para el procesamiento de operadores de selección, proyección y *join*). Los costos de comunicación nacen a partir de la transmisión de datos de un sitio a otro en el caso de bases de datos distribuidas, y algunos trabajos también incluyen el overhead de procesamiento para iniciar y concluir las transmisiones (Abdalla M., Karabatak M., 2020; Liu S., Xu X., 2016). Estos valores generalmente se suman para determinar el costo total del plan de ejecución.

Si bien todos esto criterios son válidos en el modelo de costos del problema de optimización, los trabajos estudiados muestran preferencias sobre los costos considerados o incluidos para realizar la optimización: en algunos casos se ignoran tanto los costos de I/O como el efecto de operadores de selección y proyección en el rendimiento de la consulta para construir un modelo basado solamente en el costo de cada *join* realizado (Ban W. et al, 2016; Yao M., 2018; Zheng W. et al., 2018; Han Y. et al, 2016). También existen propuestas que buscan solo centrarse en los costos de comunicación mediante el uso de métricas como el Costo de Proximidad de Consulta (QPC) que toma en cuenta la cantidad de sitios utilizados en una consulta y las veces

que se utiliza cada sitio para describir la concentración de las tablas en los sitios utilizados (Vijay T., Yadav M., 2017; Panahi V., Navimipour N., 2019).

Los diferentes modelos de costos planteados son utilizados en los algoritmos para comparar la calidad de las soluciones encontradas y también dentro de los procesos de construcción, selección y transformación de soluciones propios de cada algoritmo. En la evaluación experimental, los trabajos suelen comparar tiempos promedios de ejecución tanto de los algoritmos evaluados como de las soluciones generadas por dichos algoritmos, en base a factores como la configuración de parámetros de entrada, la cantidad de *joins* de la consulta, el número de tablas y sitios involucrados, etc. Algunos trabajos, en vez de promediar el tiempo de ejecución de la mejor solución para un número de corridas del algoritmo, proponen tomar las K mejores soluciones producidas por el algoritmo en una ejecución y promediar sus tiempos de ejecución (Vijay T., Yadav M., 2017; Panahi V., Navimipour N., 2019).

## 4.7. Conclusiones

Tras el proceso de revisión de literatura, podemos realizar conclusiones sobre diferentes aspectos del estudio del problema de optimización de consultas SQL en bases de datos relacionales:

- Estrategia utilizada: se sabe que la aplicación de algoritmos metaheurísticos en la optimización de consultas SQL es un tema que se ha estudiado y que ha ido progresando incluso en los últimos años. Si bien es posible observar un gran interés en utilizar el algoritmo genético para atacar la optimización, se sabe que este no es la única propuesta existente y que existen otros algoritmos que pueden utilizarse para producir buenos resultados, ya sea de manera independiente o mediante enfoques híbridos.
- Planteamiento del problema: dos grandes aspectos que se han mantenido relevantes en prácticamente todos los trabajos revisados son la importancia de las operaciones *join* y su orden en el rendimiento de una consulta, y la influencia de las bases de datos distribuidas y sus

capacidades de distribución, valga la redundancia, y replicación de datos en sitios separados físicamente. Ambos reflejan las necesidades y prácticas actuales de almacenamiento y recuperación de datos.

- Construcción de modelo de costos: con la revisión de los trabajos mencionados líneas arriba se determinó que existen diferentes costos relevantes para evaluar el rendimiento de las consultas optimizadas: aquellos relacionados al poder de procesamiento de los servidores de base de datos; aquellos relacionados a las diferentes operaciones relacionales que se realizan en una consulta; y aquellos relacionados a la transmisión de datos de un sitio a otro a través de redes. Si bien esta división es prácticamente un consenso, cada trabajo le atribuye diferentes pesos o importancias a cada componente de costo dependiendo de evidencia experimental, generalizaciones sobre las capacidades actuales de los sistemas de administración de base de datos y restricciones para no complicar innecesariamente el problema de optimización.

# 5. Definición de variables y diseño de función objetivo

## 5.1. Introducción

El presente capítulo detalla el desarrollo del primer objetivo específico **(O1): "Definir las variables, parámetros y restricciones del algoritmo, y diseñar estructuras de datos que las soporten para diseñar la función objetivo utilizada en la optimización."** Este objetivo está compuesto por tres resultados esperados: las variables, parámetros y restricciones del problema; el diseño de estructuras de datos para estas; y la función objetivo a utilizarse en el algoritmo memético.

## 5.2. Resultados alcanzados

### 5.2.1. Variables, parámetros y restricciones que serán consideradas en el algoritmo propuesto

En la presente sección se describirán las variables y restricciones a tomar en cuenta para el problema de optimización de una consulta SQL. Un formulario de validación de lo desarrollado llenado por un especialista en bases de datos puede encontrarse en el **anexo J**.

#### 5.2.1.1. Variables y parámetros del problema

Tras volver a revisar los trabajos obtenidos mediante la revisión sistemática, se identificaron las siguientes variables relevantes:

- **Cantidad de tablas utilizadas en la consulta**

  Es importante conocer la cantidad de tablas que contiene una consulta, pues esta se utiliza para determinar la cantidad de operaciones que conlleva dicha consulta. Por ejemplo, si una consulta utiliza 4 tablas, entonces su ejecución implica la realización de 3 operaciones *join*. Esta variable será referenciada como ***NumTab***.

- **Cantidad de sitios de la base de datos**

  Como se mencionó en capítulos anteriores, en una base de datos distribuida existen múltiples sitios separados físicamente que contienen tablas. Es importante conocer la

cantidad de sitios que posee la base de datos para poder construir planes de ejecución y determinar si será necesario transmitir datos de un sitio a otro. Esta variable será referenciada como ***NumSit***.

- **Distribución de tablas en los sitios de la base de datos**

  No solo es importante conocer las tablas y los sitios existentes en la base de datos, sino también como las primeras están distribuidas en estos sitios. La existencia o no existencia de una tabla en determinados sitios es importante para determinar la validez y calidad de un plan de ejecución. Por ejemplo, resulta menos costoso utilizar un solo sitio que contenga todas las tablas necesarias para ejecutar una consulta que traer las tablas de numerosos sitios. La replicación de tablas en múltiples sitios abre nuevas posibilidades para la generación de un plan de ejecución, pero también conlleva una mayor transmisión de datos a través de los sitios. Como este aspecto muestra una relación entre varias tablas y varios sitios, no se le va a representar como un número aislado, pero en adelante se le podría referenciar como ***DistTabSit***.

- **Tamaño de las tablas**

  El tamaño de una tabla, sea en cantidad de filas/columnas o tamaño en bits/bytes, es importante para determinar el peso que esta tiene sobre el procesamiento necesario para ejecutar una consulta. Mientras más grande sea una tabla, por ejemplo, más costoso será enviarla a otros sitios de la base de datos. El tamaño de una tabla i será referenciado como $\boldsymbol{NumFil_i}$ para su cantidad de filas, $\boldsymbol{NumCol_i}$ para su cantidad de columnas y $\boldsymbol{NumByte_i}$ para su cantidad de bytes.

- **Cardinalidad de las tablas**

En una parte de los trabajos académicos revisados, la cardinalidad de las tablas (definida como el número de valores distintos en una columna determinada) era necesaria para determinar el costo de realizar una operación *join* entre dos tablas. Esta variable será referenciada como $\boldsymbol{CardTab_{ij}}$ para una tabla i, considerando la columna j.

- **Costos de procesamiento**

  Son los costos incurridos por procesar las operaciones realizadas sobre las tablas durante la ejecución de una consulta en los diferentes sitios de la base de datos. En general, estos costos incluyen operaciones de selección, proyección y *join*, pero varios de los trabajos revisados centran esta parte de su modelo de costos meramente en los *joins* realizados. Tomando en cuenta que el procesamiento de datos puede realizarse en cualquier sitio de la base de datos, se decidió que esta variable se calcule para cada *join* realizado. Así, el costo de procesamiento para el *join* i sería referenciado como $\boldsymbol{CostProc_i}$.

- **Costos de comunicación**

  También denominados costos de transmisión, son aquellos incurridos por el envío y recepción de datos entre diferentes sitios. Para calcularlos se toman en cuenta las capacidades de transmisión entre cada par de sitios de la base de datos (en bytes por segundo) y el tamaño de los datos a ser transferidos, incluyéndose en algunos casos el overhead para iniciar la transmisión y para dar formato y procesar los mensajes de la misma. Así, se decidió referenciar el costo de comunicación como ***CostCom***, y la capacidad de transmisión entre dos sitios (en este caso, los sitios i y j) como $\boldsymbol{CapTrans_{ij}}$.

#### 5.2.1.2. Restricciones del problema

Las restricciones y condiciones descritas en esta sección hacen referencia al cálculo de las características del resultado de una operación *join*.

Al unir dos tablas en una consulta SQL por medio de una operación *join*, el resultado de esta operación también comparte las características de una tabla (tamaño, cardinalidades, ubicación en un sitio de la base de datos). El cálculo de algunas de estas características es relativamente trivial. Por ejemplo:

- ***NumCol*:** Es igual a la cantidad obtenida a partir de la unión de las columnas de las tablas participantes. Por ejemplo, si tabla T1 tiene 4 columnas, la tabla T2 tiene 5 columnas, y ambas comparten 2 columnas, el número de columnas del *join* resultante sería 7 (2 columnas propias de T1, 2 columnas compartidas y 3 columnas propias de T2).
- **Ubicación en la base de datos:** Si las tablas participantes se encuentran en el mismo sitio de la base de datos, el resultado del *join* también se ubicará en dicho sitio. Por otro lado, si las tablas participantes (T1 y T2, por ejemplo) se ubican en sitios diferentes, se ha decidido que el join resultante se ubique en el sitio de la segunda tabla (que en este ejemplo sería T2). Esta decisión resalta la importancia del orden en el que se efectúen los *joins*, pues esto afectará la ubicación de los resultados.

Sin embargo, se deben tener otras consideraciones para calcular otras variables:

- ***NumFil*:** En la práctica, determinar el número de filas depende de varios factores, siendo uno de los más importantes los datos y su distribución al interior de las tablas participantes en el *join*. Sin embargo, como en el algoritmo a desarrollarse en el

presente proyecto es de naturaleza académica/teórica, no se cuenta con esa información. Así, se ha decidido proponer una serie de condiciones para generalizar el cálculo de esta variable:

**-En una tabla con *NumFil* = n, para un conjunto de columnas con cardinalidades (c1,c2,...,cx), es posible representar dicho conjunto como una columna compuesta, cuya cardinalidad es igual a:**

$$min(\prod_{i=1}^{x} ci,\ n)$$

Por ejemplo, si la tabla T1 tiene 100 filas y se quiere representar a sus columnas A, B y C (de cardinalidades 20, 15 y 5, respectivamente) como una columna compuesta, la cardinalidad de esta última será igual al mínimo entre 20*15*5 (que representa el máximo de combinaciones únicas de los valores de cada columna) y 100. Así, la cardinalidad calculada será 100. De esta forma, esta condición nos permite utilizar la cardinalidad más alta posible para una columna compuesta.

**-Entre dos tablas T1 (n1 filas) y T2 (n2 filas) con una columna común de cardinalidades c1 y c2, respectivamente, se asume lo siguiente:**

**Si c1<c2, los c1 valores en T1 se encuentran en T2 una única vez. De esta manera, el número de filas del *join* resultante es igual a n1 (por tener la menor cardinalidad).**

**Si c1=c2, los c1 valores en T1 son los mismos que los de T2, y estos se encuentran distribuidos de manera uniforme en las filas de cada tabla. De esta manera, el número de filas del *join* resultante es igual a (n1*n2)/c1.**

**Si c2<c1, se pueden invertir los roles de T1 y T2 para aplicar el primer supuesto.**

Estas dos restricciones permiten generalizar el cálculo de *NumFil* para el resultado de un *join* en 3 fórmulas: en caso T1 y T2 cuenten con múltiples columnas comunes, estas columnas se pueden reducir a una sola con la primera restricción, y a partir de esta única columna y sus cardinalidades se puede utilizar la segunda restricción para calcular *NumFil*. Estas restricciones son un aporte que permite llevar a cabo la optimización de consultas SQL a un nivel aceptable para el ámbito académico y sin necesitar datos del contenido de una base de datos real.

- ***NumByte:*** Una vez obtenido el *NumFil* del resultado de un *join*, se podría calcular el tamaño en bytes del mismo de manera exacta si se conociera el tamaño en bytes de cada columna del problema. Sin embargo, para simplificar este proceso, se ha decidido utilizar un parámetro de entrada llamado ***tamPromCol*** que especifique el tamaño promedio de las columnas en la base de datos. De esta forma, *NumByte* se puede calcular con la siguiente fórmula:

$$NumByte = NumFil * NumCol * tamPromCol$$

- **Cardinalidades de las columnas (*CardTab*):** En uno de los trabajos leídos para la revisión sistemática se mencionó que la cardinalidad de las columnas del resultado de un *join* dependía del origen de dicha columna: si la columna era propia de una sola de las tablas participantes, se usaba su cardinalidad en dicha tabla; si era compartida por ambas tablas, se usaba el mínimo de las dos cardinalidades (Han Y. et al, 2016). Se decidió utilizar esta lógica, pero modificada para ser consistente con el cálculo de *NumFil* descrito anteriormente. Así, la cardinalidad de la columna *c* en un *join* a partir de las tablas T1 y T2 sería:

$$\text{Card(c)}_{join} \begin{cases} \min(\text{Card(c)}_{T1}, \text{NumFil}_{join}) & , c \in T1 \\ \min(\text{Card(c)}_{T2}, \text{NumFil}_{join}) & , c \in T2 \\ \min(\text{Card(c)}_{T1}, \text{Card(c)}_{T2}, \text{NumFil}_{join}) & , c \in T1,T2 \end{cases}$$

### 5.2.2. Estructuras de datos para variables y restricciones

En la presente sección se describirán las estructuras de datos propuestas para representar las variables y restricciones descritas líneas arriba dentro del algoritmo propuesto. Un formulario de validación de lo desarrollado llenado por un especialista en algoritmia puede encontrarse en el **anexo K**.

- **Cantidad de tablas utilizadas en la consulta (*NumTab*)**

  Al tratarse solamente de un número, esta variable no necesita representarse con una estructura de datos. Sin embargo, sí se utilizará para diseñar las estructuras de datos descritas más adelante.

- **Cantidad de sitios de la base de datos (*NumSit*)**

  De la misma forma que *NumTab*, esta variable tampoco necesita una estructura de datos, pero se utilizará para diseñar estructuras para las variables posteriores.

- **Distribución de tablas en los sitios de la base de datos (*DistTabSit*)**

  Esta variable puede representarse como una matriz bidimensional de tamaño ***NumSit***X***NumTab***. El elemento con posición (i,j) determinaría si en el sitio i se encuentra (valor 1) o no (valor 0) la tabla j. Así en caso de existir 5 tablas (T1, T2, …, T5) en una base de datos con 3 sitios (S1, S2, S3), un ejemplo de una matriz válida sería el siguiente:

| | **S1** | **S2** | **S3** |
|---|---|---|---|
| **T1** | 1 | 1 | 0 |
| **T2** | 0 | 0 | 1 |

| | | | |
|---|---|---|---|
| **T3** | 1 | 0 | 1 |
| **T4** | 1 | 1 | 1 |
| **T5** | 0 | 1 | 0 |

En dicho ejemplo, la Tabla 1 se encontraría en los sitios 1 y 2, y la tabla 2 solamente en el sitio 3.

- **Tamaño de las tablas (*NumFil, NumCol, NumByte*)**

  Las tres representaciones del tamaño de una tabla son numéricas, así que no se necesitarán estructuras de datos para representarlas.

- **Cardinalidad de las tablas (*CardTab*)**

  La cardinalidad de una tabla para todas las columnas existentes en el problema puede representarse con un arreglo cuyo tamaño es la cantidad total de columnas de todas las tablas. Así, el elemento i del arreglo determinaría la cardinalidad de la tabla para la columna i (Ci). Si la tabla no posee una determinada columna, entonces el valor contenido en el arreglo para la posición correspondiente es 0. Un ejemplo del arreglo para una tabla en un problema con un total de 5 columnas sería:

| **C1** | **C2** | **C3** | **C4** | **C5** |
|---|---|---|---|---|
| 1 000 | 631 | 6 | 0 | 892 |

  Si se desea acceder rápidamente a las cardinalidades para todas las tablas, se podría utilizar una matriz que contenga estos arreglos para cada tabla del problema (tamaño: número total de columnas x **NumTab**).

- **Costos de procesamiento (*CostProc)***

  Los costos de procesamiento de todos los *joins* podrían no necesitar ser representados con una estructura de datos en caso estos se calculen y se sumen directamente al momento de

determinar el costo total de un plan de ejecución. Sin embargo, si se desea almacenarlos para guardar un registro de cada *join* por separado, se podría utilizar un arreglo de tamaño (***NumTab* -1)**. Así, el costo de procesamiento calculado para el *join* i se almacenaría en el elemento i del arreglo. Un ejemplo simple para una consulta con 3 *joins* sería:

| $CostProc_1$ | $CostProc_2$ | $CostProc_3$ |
|---|---|---|
| 204.8 | 125.2 | 1 000.0 |

- **Costos de comunicación (*CostCom, CapTrans*)**

  Al ser un único valor numérico, los costos de comunicación no necesitan una estructura de datos para ser representados. Por otro lado, las capacidades de transmisión entre los sitios de la base de datos serán representadas con una matriz bidimensional de tamaño ***NumSit*X*NumSit***. Así, el elemento con posición (i,j) determinaría la capacidad de transmisión entre los sitios i y j. Se tendría el mismo valor en la posición (j,i), pues la transmisión es bidireccional. Además, los elementos con posición (i,i) siempre tendrían valor 0, pues no se dan transmisiones de datos por medio de la red al interior de un mismo sitio. Por ejemplo, una matriz válida para una base de datos con 3 sitios (S1, S2, S3) sería:

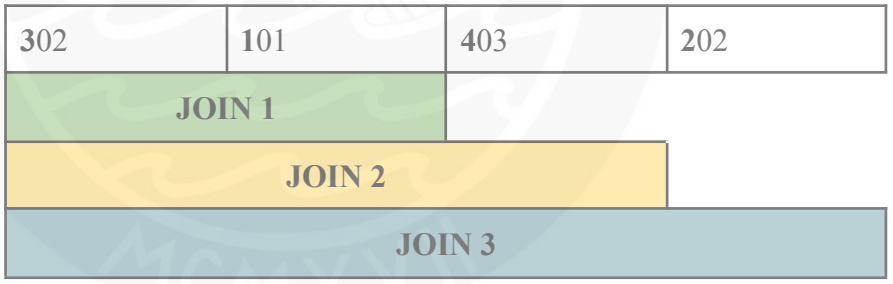
| | **S1** | **S2** | **S3** |
|---|---|---|---|
| **S1** | 0 | 7 000 | 12 000 |
| **S2** | 7 000 | 0 | 3 000 |
| **S3** | 12 000 | 3 000 | 0 |

Tras especificar estructuras para las variables de la sección anterior, a continuación se mostrará la estructura elegida para el cromosoma, el cual representará una solución al problema de optimización de una consulta. Se decidió utilizar un arreglo de números enteros de tamaño ***NumTab***:


Dentro de este arreglo, cada entero representa una combinación Tabla-Sitio de la siguiente manera:

**TT**SS

Donde **TT** representa el ID de la tabla y se obtiene dividiendo el número entre 100, y SS representa el ID del sitio del que proviene la tabla y se obtiene calculando **TT**SS%100. Así, el arreglo se va leyendo de izquierda a derecha para determinar el orden de los *joins* realizados. El sitio en el que se ubica el resultado de un join en específico es determinado por el sitio en el que se ubica la segunda tabla de dicho *join*. Un ejemplo aplicado a un caso con 4 tablas (T1, T2, T3, T4) y 3 sitios (S1, S2, S3) se muestra a continuación:




El cromosoma equivale al siguiente orden de joins:

1. T3 en S2 *join* T1 en S1 (formando el JOIN 1)
2. JOIN 1 en S1 *join* T4 en S3 (formando el JOIN 2)
3. JOIN 2 en S3 *join* T2 en S2 (formando el JOIN 3, el cual es el resultado final de la consulta)

Esta configuración del cromosoma soporta una cantidad de tablas cualquiera y como máximo 99 sitios de base de datos, por lo que cubre cualquier escenario práctico de consultas en bases de datos distribuidas. Cabe destacar además que, al incluir

valores enteros en lugar de binarios, los algoritmos metaheurísticos a desarrollar utilizando este cromosoma deberían considerarse algoritmos continuos.

### 5.2.3. Función objetivo a utilizar en el algoritmo

En la presente sección se describe la composición y fórmulas pertenecientes a la función objetivo a utilizarse para la optimización. Para esto, empezaremos definiendo el costo total de un plan de ejecución (***CostTotal***):

$$CostTotal = C_{COM} \star \frac{ln(1+CostCom)}{10} + C_{PROC} \star \frac{ln\,(1+\sum_{i=1}^{JOINS} CostProc_i)}{10} \quad \textbf{(1)}$$

Donde JOINS representa el número de joins realizados en la consulta y $C_{COM}$ y $C_{PROC}$ son coeficientes de costos de comunicación y procesamiento, respectivamente ($C_{COM}$ + $C_{PROC}$ = 1.0). Estos coeficientes se calculan a partir de un parámetro ***coefCom***. Al costo de comunicación y la sumatoria de costos de procesamiento de todos los joins se les ha aplicado logaritmo natural y se les ha dividido entre 10 para reducir sus valores a un rango pequeño, evitándose así que un costo domine al otro al calcular el costo total. El costo de comunicación (***CostCom***) se define de la siguiente manera:

$$CostCom = \sum_{i=1}^{TRANS} (OH + \frac{NumByte_i}{CapTrans}) \quad \textbf{(2)}$$

Donde i representa una transmisión de datos (una tabla o el resultado de un *join*) entre dos sitios (j y k, por ejemplo); TRANS representa el número de transmisiones realizadas; OH representa el tiempo de overhead de dicha transmisión (el cual será un dato de entrada del algoritmo); $NumByte_i$ representa la cantidad de bytes a ser transmitidos; y CapTrans representa la capacidad de transmisión entre los sitios j y k, en este caso. A continuación, el costo de

procesamiento de un *join* (***CostProc***$_i$) se define de la siguiente manera:

$$CostProc_i = \frac{NumFil_A \star NumFil_B}{\prod_{k \in C} max(CardTab_{Ak}, CardTab_{Bk})} \quad \textbf{(3)}$$

Donde A y B representan las tablas sobre las que se realiza el *join* y C representa las columnas comunes entre las tablas A y B. Se ha decidido utilizar esta fórmula para el costo de procesamiento de un join debido a que múltiples trabajos académicos dentro de los recogidos en la revisión sistemática la han utilizado en la composición de sus funciones objetivo (Han Y. et al, 2016; Ban W. et al, 2016; Venkata S., Vatsavayi V., 2017; Yao M., 2018).

Finalmente, como se busca que el costo se minimice en la optimización, la función objetivo (***FO***) se define como:

$$FO = \frac{1}{CostTotal} \quad \textbf{(4)}$$

El comportamiento de esta función objetivo fue evaluado sobre un problema de 5 tablas y 3 ciclos tomando un cromosoma como base y aplicando 3 cambios al mismo para observar el aumento o disminución del valor de la función objetivo sobre los cromosomas cambiados. Es posible acceder al documento con las pruebas de comportamiento en el **anexo C.** Además, en el **anexo L** puede encontrarse un formulario de validación de las fórmulas de la función objetivo llenado por un especialista en algoritmia.

## 5.3. Discusión

Tras el desarrollo del presente capítulo, se cuenta con una serie de variables relevantes para el problema de optimización de consultas SQL, las cuales están acompañadas en algunos casos de estructuras de datos que las representan; y también con la formulación matemática de una función objetivo a partir de dichas variables. El desarrollo de estos resultados esperados se

realizó tomando como base la literatura encontrada mediante la revisión sistemática y complementándola con las decisiones tomadas por el autor en conjunto con su asesor.

Los resultados esperados constituyen una base para entender el problema de optimización de consultas y representarlo de una manera simplificada y traducible a la lógica de un algoritmo metaheurístico. Sin embargo, es importante observar que existen limitaciones en lo desarrollado: un ejemplo es que no se han incluído todos los factores o variables encontrados en la literatura, pues existen apreciaciones variadas sobre qué variables son relevantes en el problema y también no todas las variables presentadas son compatibles entre sí (representan un mismo aspecto del problema pero de diferentes maneras). Además, se plantearon algunas restricciones para poder calcular variables que, en la práctica, necesitarían más datos que los que se han planteado para ser obtenidas.

# 6. Adaptación de algoritmo memético

## 6.1. Introducción

El presente capítulo detalla el desarrollo del segundo objetivo específico **(O2): "Adaptar el algoritmo metaheurístico memético para la optimización de consultas."** Este objetivo está compuesto por tres resultados esperados: el pseudocódigo del algoritmo memético, la codificación del mismo y la calibración de las variables o parámetros de ejecución.

## 6.2. Resultados alcanzados

### 6.2.1. Pseudocódigo del algoritmo memético

El pseudocódigo planteado para el algoritmo es el siguiente:

```
Datos: numTab, numSit, distTabSit, datos de tablas(id,numFil,
numCol, numByte), CardTab, capTrans, overhead, coefCom
param: numIter, tamPob, porcCromCasados, probMut, probBusq,
porcHijosIngresados, cantVecinosEval, porcIterEstanc
------------------------------------------------------------
INICIO algoritmoMemetico(Datos, param)
1:  pob = inicializarPoblacion(Datos, tamPob)
2:  calidadesPob = evaluarCalidad(pob, Datos)
3:  mejorCrom = encontrarMejor(pob, calidadesPob)
4:  i= 0, iterEstanc= numIter*porcIterEstanc, contEstanc= 0
5:  Mientras i<numIter && contEstanc<iterEstanc Hacer:
6:  | padres= seleccionarPadres(pob, porcCromCasados,
    |                            calidadesPob)
7:  | hijos= casarPadres(padres)
8:  | hijos= mutarHijos(hijos, probMut)
9:  | hijos= busquedaLocal(hijos, probBusq, cantVecinosEval,
    |                      Datos)
10: | calidadesHijos= evaluarCalidad(hijos, Datos)
11: | pob= evolucionarPoblacion(pob, hijos, mejorCrom,
    |                           calidadesPob)
12: | antiguoMejor= mejorCrom
13: | mejorCrom= encontrarMejor(pob, calidadesPob)
14: | Si antiguoMejor==mejorCrom Entonces:
15: | | contEstanc+= 1
16: | Sino:
17: | | contEstanc= 0
18: | FinSi
19: FinMientras
20: Retornar mejorCrom
```

```
FIN algoritmoMemetico
```

Ahora se procederá a explicar los fragmentos del pseudocódigo:

- **Inicialización de la población (línea 1):** Se crea un conjunto de cromosomas con soluciones válidas de manera aleatoria. La población tendrá un tamaño de *tamPob* cromosomas. La lógica resumida para la inicialización es la siguiente:

```
INICIO inicializarPoblacion(Datos, tamPob)
1:  pob= [], i= 0, distTablas= Datos.distTabSit
2:  Mientras i<tamPob Hacer:
3:  | crom= []
4:  | Para cada tabla en Datos.tablas Hacer:
5:  | | indice, idTabla= tabla.id, idSitio
6:  | | Hacer:
7:  | | | indice= numEnteroAleatorioEntre(0, numTab-1)
8:  | | Mientras crom[indice] esté ocupado
9:  | | Hacer:
10: | | | idSitio= numEnteroAleatorioEntre(1,numSit)
11: | | Mientras distTablas[idTabla-1][idSitio-1] == 0
12: | | crom[indice]= idTabla*100 + idSitio
13: | FinPara
14: | pob[i]= crom
15: | i++
16: FinMientras
17: Retornar pob
FIN inicializarPoblacion
```

- **Evaluación de calidad de los individuos (líneas 2 y 3):** La función objetivo descrita en el capítulo anterior se aplica sobre los cromosomas pertenecientes a la población utilizando la solución que cada uno representa y la información de tablas y sitios. Tras calcular la calidad, se determina cuál de los individuos es el de mayor calidad. La lógica resumida del cálculo de calidad es la siguiente:

```
INICIO evaluarCalidad(arrCrom, Datos)
1:  calidades= [], i= 0
2:  Para cada crom en arrCrom Hacer:
3:  | costProc= 0, costCom= 0 ,idTabla= crom[0]/100-1, j=1
4:  | tabla1= Datos.tablas[idTabla], idSit1= crom[0]%100
5:  | Mientras j<Datos.numTab Hacer:
```

```
6:  | | tabla2= datos.tablas[(crom[j]/100)-1]
7:  | | idSit1= crom[j-1]%100, idSit2= crom[j]%100
8:  | | costProc+= calcCostProc(tabla1,tabla2,Datos.cardTab)
9:  | | Si idSit1!=idSit2 Entonces:
10: | | | costCom+= calcCostCom(tabla1,tabla2,idSit1,idSit2,
    | | |                       Datos.capTrans,Datos.overhead)
11: | | FinSi
12: | | //Se forma una tabla para el join de 1 y 2
13: | | tabla1= formarJoin(tabla1,tabla2,Datos.cardTab)
14: | | j++
15: | FinMientras
16: | calidadCrom= aplicarFormula(costProc,costCom,Datos.coefCom)
17: | calidades[i] = calidadCrom
18: | i++
19: FinPara
20: Retornar calidades
FIN evaluarCalidad
```

- **Condiciones de parada (línea 5):** El resto de la lógica del algoritmo se realiza de manera iterativa dentro de un bucle. Dicho bucle se detiene cuando se cumple la condición o condiciones de parada. Para el algoritmo propuesto, las condiciones a utilizarse son un límite de iteraciones (tras la iteración número 500 el algoritmo se detiene, por ejemplo) y una parada por estancamiento (por ejemplo, si en las últimas 100 iteraciones la calidad de la mejor solución no ha aumentado, el algoritmo se detiene). Solamente es necesario que se cumpla una de estas condiciones para dar fin al bucle.
- **Selección de padres (línea 6):** Una fracción de la población es seleccionada de manera aleatoria. A los cromosomas elegidos se les llamará padres. En la implementación del algoritmo se decidió utilizar el método de ruleta para la selección de padres. En este método, se escoge un cromosoma de la población de manera aleatoria, pero cada cromosoma tiene una probabilidad de ser elegido proporcional a su calidad calculada. La lógica básica está representada en el siguiente pseudocódigo:

```
INICIO seleccionarPadres(pob, porcCromCasados, calidades)
1:  padres= [], i=0, sumaCalidades= suma(calidades)
2:  cantidadPadres= tamaño(pob)*porcCromCasados //Debe ser par
```

```
3:  Mientras i<cantidadPadres Hacer:
4:  | valSeleccion= numRealAleatorioEntre(0,sumaCalidades)
5:  | j= 0, sumaParcial= 0
6:  | Mientras j<tamaño(pob) Hacer:
7:  | | sumaParcial+= calidades[j]
8:  | | Si sumaParcial>=valSeleccion) Entonces:
9:  | | | break
10: | | FinSi
11: | | j++
12: | FinMientras
13: | padres[i] = pob[j]
14: | i++
15: FinMientras
16: Retornar padres
FIN seleccionarPadres
```

- **Casamiento de cromosomas (línea 7):** A los cromosomas padres se les aplica un operador de casamiento para poder producir cromosomas hijos. Este operador necesita dos padres y puede producir uno o dos hijos dependiendo de la implementación. En este caso, se decidió utilizar como implementación el método *Partially Mapped Crossover* o PMX. Este copia un segmento de un padre e intenta mantener el orden del otro padre para los elementos faltantes al crear un hijo (Eiben A., Smith J., 2003). La lógica detrás del PMX es la siguiente:

```
INICIO casarPadres(padres)
1:  hijos= [], i= 0
2:  Mientras i<tamaño(padres) Hacer:
3:  | hijo= [], padre1= padres[i], padre2= padres[j]
4:  | indice1= numEnteroAleatorioEntre(0,tamaño(padre1)-1)
5:  | indice2= numEnteroAleatorioEntre(indice1,tamaño(padre1)-1)
6:  | //Se copia el segmento de padre1 entre los índices al hijo
7:  | copiarSegmento(padre1,hijo,indice1,indice2)
8:  | j= indice1
9:  | Mientras j<=indice2 Hacer:
10: | | idTabPadre2= padre2[j]/100
11: | | //Se busca el id de tabla en el segmento del padre1
12: | | Si indiceNumEnSegmento(idTabPadre2,padre1,indice1,
    | | |                      indice2)==-1 Entonces:
13: | | | indiceColocacion= j
14: | | | Hacer:
15: | | | | idTabPadre1= padre1[indiceColocacion]/100
16: | | | | indiceColocacion= indiceNumEnSegmento(idTabPadre1,
17: | | | |                   padre2,0,tamaño(padre2)-1)
18: | | | Mientras hijo[indiceColocacion] esté ocupado
19  | | | hijo[indiceColocacion]= padre2[j]
```

```
20: | | FinSi
21: | | j++
22: | FinMientras
23: | //Se llenan los espacios vacíos con los elem. de padre2
24: | j= 0
25: | Mientras j<tamaño(padre2) Hacer:
26: | | Si hijo[j] no está ocupado Hacer:
27: | | | hijo[j]= padre2[j]
28: | | FinSi
29: | | j++
30: | FinMientras
31: | hijos[i/2]= hijo
32: | i+= 2 //Se toman dos padres en cada iteración
33: FinMientras
34: Retornar hijos
FIN casarPadres
```

- **Mutación de cromosomas hijos (línea 8):** A cada cromosoma hijo generado por el casamiento se le aplica un operador de mutación, el cual altera parte de su estructura sin información de otro cromosoma. En este caso, se decidió que la implementación de la mutación solamente escoja dos posiciones al azar del cromosoma e intercambie los elementos en dichas posiciones. Este método es conocido como Mutación por Intercambio o *Swap Mutation* (Eiben A., Smith J., 2003). Cabe resaltar que la aplicación del operador a cada cromosoma hijo se realiza o no dependiendo de una probabilidad ingresada como parámetro, la cual llamaremos *pMut*. La lógica de la mutación se detalla a continuación:

```
INICIO mutarHijos(hijos, probMut)
1:  hijos2= [], i=0
2:  Mientras i<tamaño(hijos) Hacer:
3:  | hijoOrig= hijos[i], hijoMutado= copiar(hijoOrig)
4:  | aleatorio= numRealAleatorioEntre(0,1)
5:  | Si aleatorio<=probMut Hacer:
6:  | | indice1= numEnteroAleatorioEntre(0,tamaño(hijoOrig))
7:  | | indice2
8:  | | Hacer:
9:  | | | indice2= numEnteroAleatorioEntre(0,tamaño(hijoOrig))
10: | | Mientras indice1==indice2
11: | | temp= hijoMutado[indice1]
12: | | hijoMutado[indice1]= hijoMutado[indice2]
13: | | hijoMutado[indice2]= hijoMutado[temp]
14: | FinSi
```

```
15: | hijos2[i]= hijoMutado
16: | i++
17: FinMientras
18: Retornar hijos2
FIN mutarHijos
```

- **Mejora mediante búsqueda local (línea 9):** Para cada cromosoma hijo se aplica un proceso de búsqueda local en la cual se determinan soluciones similares o "vecinas" y se las compara con el cromosoma original para determinar la de mayor calidad. En este caso, para la implementación se decidió que las soluciones vecinas se generen cambiando los sitios de los que se traen las tablas. Para esto será necesario acceder a la distribución de tablas en sitios (*DistTabSit*) con el fin de no generar soluciones inválidas. Al igual que con la mutación, la aplicación de búsqueda local para un cromosoma depende de un parámetro de probabilidad, al cual llamaremos *pBusq*. La lógica que se busca implementar para la búsqueda local es la siguiente:

```
INICIO busquedaLocal(hijos, probBusq, cantVecinosEval, Datos)
1:  hijos2= [], i=0, distTablas= Datos.distTabSit
2:  Mientras i<tamaño(hijos) Hacer:
3:  | hijoOrig= hijos[i], hijoBusq= copiar(hijoOrig)
4:  | aleatorio= numRealAleatorioEntre(0,1)
5:  | Si aleatorio<=probBusq Hacer:
6:  | | j=0, ganador= copiar(hijoOrig)
7:  | | Mientras j<cantVecinosEval Hacer:
6:  | | | vecino= copiar(ganador)
7:  | | | indCambio= numEnteroAleatorioEntre(0,tamaño(vecino))
8:  | | | idTabla= vecino[indCambio]/100, indSitNuevo
9:  | | | Hacer:
10: | | | | idSitNuevo= numEnteroAleatorioEntre(1,
    | | | |                    Datos.numSit)
11: | | | Mientras distTablas[idTabla-1][idSitNuevo-1]==0
12: | | | vecino[indCambio]= idTabla*100 + idSitNuevo
13: | | | cal1= evaluarCalidadCrom(ganador, Datos)
14: | | | cal2= evaluarCalidadCrom(vecino, Datos)
15: | | | Si cal2>cal1 Entonces:
16: | | | | ganador= vecino
17: | | | FinSi
18: | | | j++
18: | | FinMientras
19: | | hijoBusq= ganador
20: | FinSi
21: | hijos2[i]= hijoBusq
```

```
22: | i++
23: FinMientras
24: Retornar hijos2
FIN busquedaLocal
```

- **Evaluación de calidad de los hijos (línea 10):** Proceso similar al de la línea 2, pero esta vez aplicado sobre el grupo de cromosomas hijos.
- **Evolución de la población (líneas 11 y 12):** Se seleccionan cromosomas hijos para reemplazar individuos de la población, obteniéndose así una nueva generación. Los hijos e individuos serán seleccionados aleatoriamente, pero se impedirá que el mejor individuo de la población sea expulsado para no perder soluciones de buena calidad. Si bien se busca que cada vez que se vaya a concretar un reemplazo, se evalúe si la calidad del nuevo integrante de la población supera a la del mejor individuo actualmente para así actualizar la mejor solución, en el pseudocódigo la actualización del mejor cromosoma se hace después de la evolución de la población.
- **Evaluación de estancamiento (líneas 13-16):** En caso no se haya encontrado una nueva mejor solución, el contador de estancamiento (*contEstanc*) se incrementa en 1, caso contrario se reinicia a 0. Este contador es utilizado para determinar el cumplimiento de la condición de parada por estancamiento.

Al pseudocódigo anterior se le han realizado pruebas de flujo con datos y parámetros controlados. Se puede acceder al documento con las pruebas en el **anexo D**. Además, lo desarrollado en esta sección ha sido validado por un especialista en algoritmia metaheurística, y este último ha llenado un formulario de validación que puede encontrarse en el **anexo M.**

### 6.2.2. Codificación del algoritmo memético

El algoritmo memético propuesto como pseudocódigo en el resultado anterior ha sido implementado en el lenguaje de programación Java. Este recibe datos de entrada a partir de un archivo de texto de formato CSV que utiliza como separador el caracter **";"**. Dichos datos de entrada se pueden clasificar en:

- **Datos del problema de optimización de consultas:** Datos relacionados a las tablas y sitios de la base de datos sobre la que se realiza la optimización de consultas y a algunas variables utilizadas en el cálculo de la función objetivo.
- **Parámetros de ejecución del algoritmo:** Datos que condicionan la ejecución del algoritmo memético (características de la población; condiciones de parada; funcionamiento y uso de operadores de casamiento, mutación y búsqueda local; entre otros).

La estructura del archivo y el resumen de los datos que contiene se puede observar en la **figura 4**:

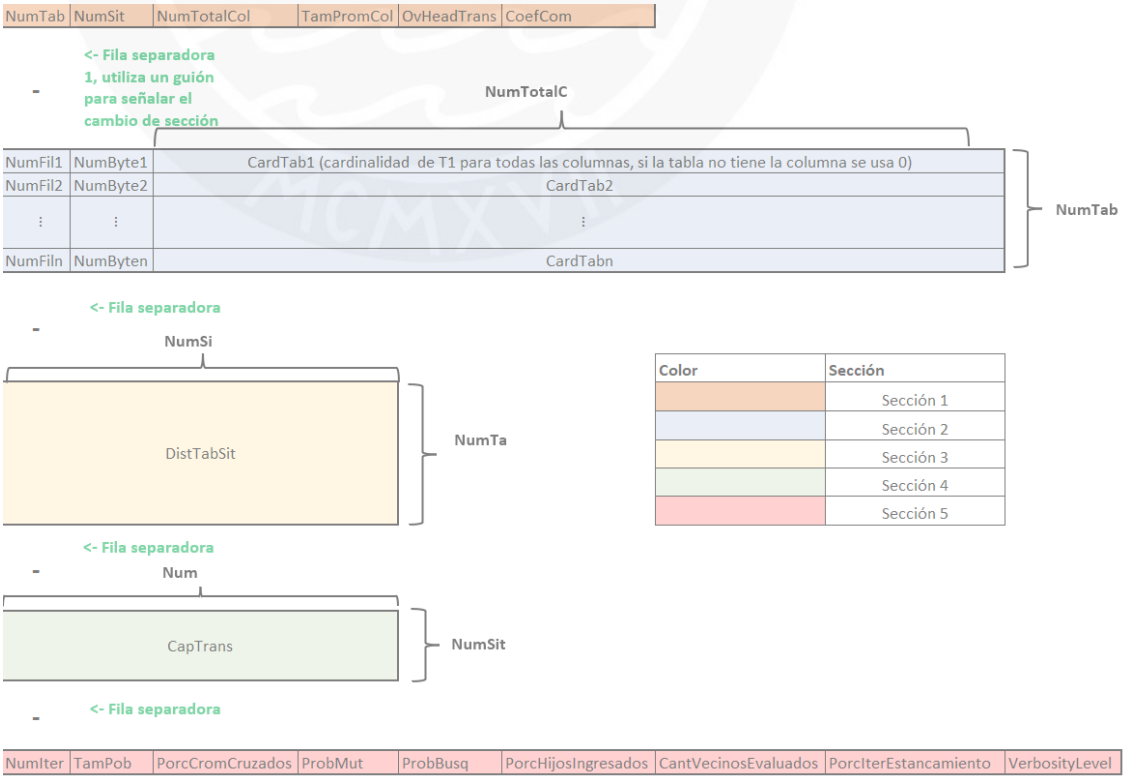


*Figura 4. Estructura de archivo de datos de entrada*

Se pueden observar 5 secciones separadas por colores. A continuación se describirán los contenidos de cada una:

- **Sección 1:** Contiene el número de tablas de la consulta (**NumTab**), el número de sitios en la base de datos (**NumSit**), el número total de todas las diferentes columnas de las tablas de la consulta (**NumTotalCol**), el tamaño promedio en bytes de cada columna (**TamPromCol**), el tiempo de overhead por transmisión entre sitios en milisegundos (**OvHeadTrans**) y el coeficiente de comunicación utilizado en la función objetivo (**CoefCom**).
- **Sección 2:** Contiene los datos específicos de cada tabla usada en la consulta. Cada fila de esta sección contiene el número de filas de una tabla (**NumFil**), el número de bytes de la misma (**NumByte**) y sus cardinalidades por cada columna (**CardTab**). En esta última parte, si la tabla en cuestión no contiene una columna, su cardinalidad será 0. Así, el número de columnas de una tabla en cuestión equivale al número de cardinalidades diferentes de 0 en **CardTab**.
- **Sección 3:** Contiene la distribución de tablas en sitios de la base de datos (DistTabSit) como una matriz de **NumTab** filas y **NumSit** columnas. El contenido de dicha matriz sigue la misma lógica utilizada en la sección que desarrolla las estructuras de datos para las variables del problema en el capítulo anterior.
- **Sección 4:** Contiene las capacidades de transmisión entre sitios en bytes por segundo (**CapTrans**), representadas en una matriz cuadrada de tamaño **NumSit**. Su contenido también utiliza la lógica planteada en la sección de estructuras de datos del capítulo anterior.
- **Sección 5:** Contiene los parámetros de ejecución del algoritmo. Estos son el número de iteraciones (**NumIter**), el tamaño de la población (**TamPob**), el porcentaje de los individuos de la

población que formará el grupo de cromosomas padres en cada iteración (**PorcCromCruzados**), la probabilidad de que se aplique mutación sobre un cromosoma hijo (**ProbMut**), la probabilidad de que se aplique búsqueda local sobre un cromosoma hijo (**ProbBusq**), el porcentaje del total de hijos generados en una iteración que pasará a formar parte de la población (**PorcHijosIngresados**), la cantidad de vecinos que se generarán iterativamente para un cromosoma durante la búsqueda local (**CantVecinosEvaluados**), el porcentaje del total de iteraciones que determinará el estancamiento de una ejecución del algoritmo (**PorcIterEstancamiento**), y el nivel de verbosidad de la ejecución del algoritmo (**VerbosityLevel**), el cual determina el detalle de los mensajes impresos en la ejecución. Normalmente, esta sección solo consistirá en una fila, pero es posible ingresar múltiples filas con diferentes valores de los parámetros de ejecución en caso se quiera ejecutar el algoritmo para múltiples configuraciones de dichos parámetros.

Puede observarse además que entre cada sección de datos existe una fila de separación que contiene un caracter **"-"**. Su propósito es apoyar en la identificación de estructuras inválidas cuando el algoritmo lea el archivo de texto. Se puede encontrar un ejemplo de archivo de datos de entrada en el **anexo E**.

Una vez descritos los datos de entrada del algoritmo, se mostrará la estructura del proyecto de Java que se codificó para el mismo en la **figura 5**:

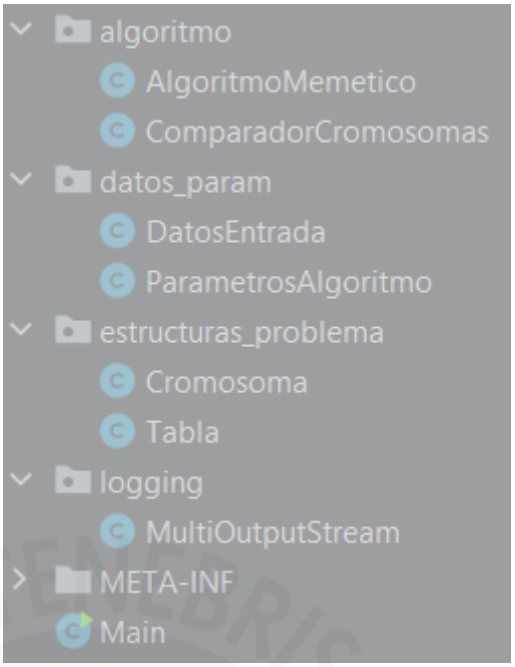



*Figura 5. Estructura del proyecto en Java del algoritmo memético*

Es posible identificar 4 paquetes además del archivo que contiene la clase Main. Como cada archivo contiene estrictamente una sola clase, se referirá a cada uno directamente como una clase en vez de como un archivo. Se describirá brevemente cada paquete a continuación:

- **algoritmo:** Incluye la clase **AlgoritmoMemetico**, la cual se usa para ejecutar el algoritmo en sí, y la clase **ComparadorCromosomas**, la cual se utiliza para ordenar descendentemente a la población en base al fitness para identificar los k mejores cromosomas al final de la ejecución del algoritmo.
- **datos_param:** Incluye las clases **DatosEntrada** (que lee el archivo de datos de entrada y almacena su contenido en atributos) y **ParametrosAlgoritmo** (que contiene específicamente los parámetros de ejecución del algoritmo y está instanciada como objeto en uno de los atributos de **DatosEntrada**). Se ha utilizado el patrón de diseño *Singleton* y un método estático para obtener una instancia única de **DatosEntrada** que pueda ser accedida desde distintos lugares del código.

- **estructuras_problema:** Incluye las clases **Cromosoma** (que representa una posible solución al problema de optimización de consultas y se utiliza durante la ejecución del algoritmo) y **Tabla** (que contiene los datos específicos de una tabla que se encontraban en la sección 2 del archivo de datos de entrada).
- **logging:** Incluye la clase **MultiOutputStream**, la cual se utiliza para configurar la salida y el error estándar del programa de tal forma que ambos impriman tanto a la consola como a archivos de texto a la vez.

Tras detallar la estructura del proyecto, es posible explicar a alto nivel lo que el programa realiza al ejecutarse. En la clase main, primero se configura la salida y error estándar para que realicen impresiones simultáneas a consola y a archivos de texto. Para esto se utiliza la clase **MultiOutputStream**. A continuación, se genera una instancia de la clase **DatosEntrada**, para lo cual se lee el archivo de datos de entrada y se almacena su contenido en los diferentes atributos de dicha instancia. Este proceso utiliza objetos de las clases **Tabla** y **ParametrosAlgoritmo**. Después de esto, se crea un objeto **AlgoritmoMemetico** que usa los parámetros de ejecución leídos del archivo de datos de entrada y se ejecuta el algoritmo. Durante esta ejecución se utilizarán objetos de las clases **DatosEntrada**, **Cromosoma** y **Tabla**. El algoritmo recibe también el parámetro *verbosityLevel* para configurar el detalle de los mensajes que se imprimen durante la ejecución (sin mensajes, mensajes al final de la ejecución, mensajes al final de cada iteración y mensajes de progreso incluso al interior de cada iteración). Finalmente, el algoritmo imprimirá un resumen de su ejecución conteniendo el tiempo de ejecución en milisegundos y el mejor cromosoma generado junto a su *fitness* calculada.

Ahora se procederá a listar y resumir los atributos y métodos de cada clase del proyecto en la **tabla 9**. En esta tabla se obviará la

clase **MultiOutputStream**, pues no es relevante para el algoritmo, y los métodos que no estén directamente relacionados a la ejecución del mismo (*setters, getters, toString*, entre otros). Además, la primera columna de la tabla determina si la fila hace referencia a un atributo (A), método (M), o constructor (C).

*Tabla 9. Atributos y métodos de clases del algoritmo memético*

| **Clase: Tabla** | | | |
|---|---|---|---|
| A/M/C | nombre | Tipo de dato o de retorno | Descripción breve |
| A | id | int | ID de la tabla. |
| A | numFilas | int | Número de filas. |
| A | numColumnas | int | Número de columnas. |
| A | numBytes | int | Número de Bytes. |
| A | cardColumnas | int[] | Card. de las columnas. |
| A | isJoin | boolean | Determina si el objeto representa el resultado de un join en lugar de una tabla normal (usado en el cálculo de función objetivo para un cromosoma). |
| **Clase: ParametrosAlgoritmo** | | | |
| A/M/C | nombre | Tipo de dato o de retorno | Descripción breve |
| A | numIter | int | Número de iteraciones. |
| A | tamPob | int | Tamaño de población. |
| A | porcCromCruzados | float | Porcentaje de la población que formará parte de los padres en cada iteración. |
| A | probMut | float | Probabilidad de aplicación de mutación a un cromosoma hijo. |
| A | probBusq | float | Probabilidad de aplicación de búsqueda local a un cromosoma hijo. |
| A | porcHijosIngresados | float | Porcentaje de los hijos que ingresarán a la población (sustituyendo individuos de la misma) en cada iteración. |
| A | cantVecinosEvalu | int | Cantidad de vecinos generados |

| | | | |
|---|---|---|---|
| | ados | | iterativamente durante cada búsqueda local. |
| A | porcIterEstanc | float | Porcentaje del total de iteraciones para considerar estancamiento. |
| **Clase: DatosEntrada** | | | |
| A/M/C | nombre | Tipo de dato o de retorno | Descripción breve |
| A | numTablas | int | Número de tablas de la consulta. |
| A | numSitios | int | Número de sitios de la base de datos. |
| A | numTotalColumnas | int | Número total de columnas. |
| A | distTabSit | boolean[][] | Matriz de distribución de tablas en sitios. |
| A | tablas | Tabla[] | Arreglo de tablas de la consulta. |
| A | cardTablas | int[][] | Matriz de cardinalidades de cada tabla para todas las columnas. |
| A | capTransSitios | int[][] | Matriz de capacidades de transmisión entre todos los sitios. |
| A | tamPromColumna | int | Tamaño promedio de cada columna en bytes. |
| A | overheadTrans | int | Tiempo de overhead por transmisión realizada en milisegundos. |
| A | coefCom | float | Coeficiente de costos de comunicación utilizado en la función objetivo. |
| A | coefProc | float | Coeficiente de costos de procesamiento utilizado en la función objetivo. |
| A | paramAlg | ParametrosAlgoritmo[] | Arreglo con objetos con los parámetros de ejecución del algoritmo. |
| A | verbosityLevel | int | Nivel de verbosidad de los |

| | | | |
|---|---|---|---|
| | | | mensajes durante la ejecución del algoritmo. |
| A | INSTANCIA | DatosEntrada | Instancia *Singleton* de esta clase. |
| M | loadInstance | DatosEntrada | Lee el archivo de datos desde un path brindado como parámetro y devuelve un objeto **DatosEntrada**. |
| M | getInstance | DatosEntrada | Se usa para obtener la instancia *Singleton* de la clase (llamará a loadInstance si esta no ha sido creada aún). |
| **Clase: Cromosoma** | | | |
| A/M/C | nombre | Tipo de dato o de retorno | Descripción breve |
| A | cromosoma | int[] | Arreglo que representa la solución al problema de optimización (descrito en el capítulo anterior). |
| A | indicesTab | int[] | Arreglo que indica el índice que cada tabla ocupa en el cromosoma (por ejemplo, si la tabla 3 ocupa el índice 1 en el cromosoma, el tercer elemento de indicesTab será igual a 1). |
| A | costProc | double[] | Arreglo con los costos de procesamiento incurridos en cada *join* realizado en la solución descrita por el cromosoma. |
| A | listaJoins | Tabla[] | Arreglo con objetos tabla que representan los *joins* realizados en la solución descrita por el cromosoma. |
| A | costCom | double | Costo de comunicación acumulado calculado. |
| A | fitness | double | Valor de la función objetivo calculado para el cromosoma. |
| A | fitnessCalculada | boolean | Flag que indica si ya se calculó el *fitness* del cromosoma. |

| | | | |
|---|---|---|---|
| C | Cromosoma (primer constructor, sin parámetros) | No aplica | Constructor que genera un objeto Cromosoma con una posible solución determinada aleatoriamente. |
| M | calcularCostProc | double | Calcula el costo de procesamiento de cada join, almacenándolo en el atrib. costProc y retornando el costo acumulado. |
| M | calcularCostCom | double | Calcula el costo de comunicación de cada transmisión entre sitios, almacenándolo en el atrib. costCom y retornándolo. |
| M | calcularFitness | void | Llama a calcularCostProc y calcularCostCom para calcular ambos costos y les aplica la fórmula de la función objetivo. El valor obtenido se almacena en el atrib. fitness. |
| M | getFitness | dobule | Retorna el atributo fitness, pero si no ha sido calculado, llama al método calcularFitness. |
| M | clonar | Cromosoma | Retorna un objeto Cromosoma con los atributos cromosoma e indicesTab copiados del objeto original. |
| **Clase: AlgoritmoMemetico (una parte de los atributos son compartidos con la clase ParametrosAlgoritmo, así que solo se están listando aquellos propios de AlgoritmoMemetico)** | | | |
| A/M/C | nombre | Tipo de dato o de retorno | Descripción breve |
| A | cantCromCruzados | int | Cantidad de cromosomas padres que serán elegidos de la población en cada iteración. |
| A | cantHijosIngresados | int | Cantidad de cromosomas hijos que serán introducidos en la población (sustituyendo individuos de la misma) en cada iteración. |
| A | poblacion | Cromosoma[] | Arreglo de cromosomas que conforma la población que el |

| | | | |
|---|---|---|---|
| | | | algoritmo cambiará iterativamente. |
| A | fitnessSumadaPoblacion | double | *Fitness* acumulada de todos los individuos de la población. Cambia en cada iteración. |
| A | mejorSolucion | Cromosoma | Cromosoma con el mayor *fitness* dentro de la población en cada iteración. |
| A | fitnessMejorSolucion | double | *Fitness* del mejor cromosoma de la población. |
| A | indiceMejorSolucion | int | Índice dentro de la población en el que se encuentra el mejor cromosoma. |
| A | tiempoEjecucion | double | Tiempo de ejecución del algoritmo en milisegundos. |
| C | AlgoritmoMemetico (primer constructor, sin parámetros) | No aplica | Llena y calcula los atributos relacionados a parámetros de ejecución a partir del primer objeto ParametrosAlgoritmo de la instancia de la clase DatosEntrada. |
| C | AlgoritmoMemetico (segundo constructor, con parámetros) | No aplica | Recibe los parámetros de ejecución del algoritmo y los usa para llenar y calcular los atributos relacionados del objeto. |
| M | incializarPoblacion | void | Genera una población de tamaño **tamPob** utilizando el constructor vacío de la clase Cromosoma. Determina simultáneamente el mejor cromosoma de la población generada. |
| M | seleccionXRuleta | Cromosoma[] | Selecciona aleatoriamente **cantCromCruzados** cromosomas de la población mediante el método de selección por ruleta y retorna un arreglo con los cromosomas seleccionados (padres). |
| M | casamientoPMX | Cromosoma[] | Recibe el arreglo de padres y les aplica *Partially Mapped Crossover* para generar un hijo por cada par de padres. Finalmente, retorna el arreglo de cromosomas hijos. |

| | | | |
|---|---|---|---|
| M | swapMutation | Cromosoma[] | Recibe el arreglo de hijos y le aplica *swap mutation* a cada uno con una probabilidad de **probMut**. Finalmente, retorna el arreglo de hijos modificado. |
| M | busqLocal | Cromosoma[] | Recibe el arreglo de hijos y le aplica búsqueda local a cada uno (generando **cantVecinosEvaluados** vecinos por cada aplicación) con una probabilidad de **probBusq**. Finalmente, retorna el arreglo de hijos modificado. |
| M | evolPoblacion | boolean | Recibe el arreglo de hijos y toma **cantHijosIngresados** cromosomas hijos para sustituir aleatoriamente individuos de la población. Siempre evita sustituir al mejor cromosoma. Finalmente, retorna un valor booleano indicando si se encontró un nuevo mejor cromosoma o no. |
| M | ejecutar | Cromosoma | Ejecuta el algoritmo memético utilizando los métodos antes descritos para generar la población y alterarla iterativamente. Finalmente, retorna al mejor cromosoma encontrado en todas las iteraciones. |
| **Clase: Main** | | | |
| A/M/C | nombre | Tipo de dato o de retorno | Descripción breve |
| M | configurarSalida | void | Configura la salida y el error estándar para la impresión simultánea a consola y archivos utilizando la clase **MultiOutputStream**. |
| M | main | void | Recibe como argumento el *path* del archivo de datos de entrada. Genera la instancia *Singleton* de **DatosEntrada** utilizando el archivo en dicho *path*. Finalmente, ejecuta el algoritmo memético. |

Si se desea leer el código fuente para comprender su estructura y lógica a detalle, en el **anexo F** se encuentra el enlace al archivo comprimido que contiene el proyecto en Java del algoritmo memético.

La codificación del algoritmo memético fue una tarea cuya complejidad implicó el uso de herramientas y metodologías de apoyo. Las metodologías utilizadas fueron:

- **Metodología Kanban**: Se tomaron algunos aspectos de la metodología Kanban para realizar la fragmentación del desarrollo en tareas y el seguimiento del progreso de dichas tareas. Para su aplicación, se utilizó como herramienta la plataforma Trello.

En primer lugar se determinaron dos categorías para las tareas a realizarse: tareas de codificación y otras tareas (que no impliquen escribir código, pueden incluir investigación de documentación y frameworks, configuración del IDE, manejo de archivos de texto, entre otros). A continuación, se creó un tablero Kanban dentro de Trello con las siguientes columnas: "**backlog"** (incluye todas las tareas al inicio), "**por hacer"** (incluye las tareas a realizarse en la semana), "**codificación/en proceso"** (incluye las tareas que se están realizando en la actualidad), "**pruebas"** (incluye las tareas de codificación que se concluyeron y están probando), **"revisión y correcciones"** (incluyen las tareas de codificación que necesitan cambios después de pasar por las pruebas) y **"completado"** (para tareas listas). Finalmente, se creó un límite de trabajo en progreso (el número máximo de tareas simultáneas entre las columnas "codificación/en proceso" y "revisión y correcciones") de dos tareas. En la **tabla 10** se mostrará el número de tareas que fueron asignadas para cada clase programada en Java:

*Tabla 10. Cantidad de tareas por clase*

| **Clase(s)** | **Cantidad de tareas relacionadas** |
|---|---|
| Tabla | 1 |
| DatosEntrada (incluyendo ParametrosAlgoritmo) | 1 |
| Cromosoma | 2 |
| MultiOutputStream | 1 |
| AlgoritmoMemetico | 6 |
| Sin codificación (creación de archivo de datos de entrada) | 1 |
| **TOTAL** | **12** |

- **Pruebas unitarias:** Se crearon pruebas unitarias automatizadas para las clases que ejecutaban lógicas y cálculos relacionados a la ejecución del algoritmo memético. Esto se logró utilizando la librería **JUnit5**. En la **tabla 11** se muestra resumen de las pruebas unitarias creadas para las clases del algoritmo:

*Tabla 11. Pruebas unitarias por clase*

| **Clase** | **Cantidad de pruebas unitarias** | **Descripción** |
|---|---|---|
| DatosEntrada | 3 | Se probó la lectura del archivo de texto brindando un *path* personalizado como argumento del programa, utilizándose un *path* por defecto relativo a la ubicación del programa, y brindando archivos inexistentes o con estructura inválida. |
| Cromosoma | 2 | Se probó la creación automática del cromosoma a partir del constructor vacío (tamaños de arreglo consistentes con datos de entrada, no repetición de |

| | | |
|---|---|---|
| | | tablas en el arreglo **cromosoma**, uso de sitios válidos según **distTabSit** y consistencia entre los índices de **indicesTab** y el arreglo **cromosoma**) y el cálculo correcto de la función objetivo. |
| AlgoritmoMemetico | 6 | Se probó el funcionamiento de los 6 operadores del algoritmo (inicialización de población, selección de padres, casamiento, mutación, búsqueda local y evolución de población). Uno de los puntos de prueba era la consistencia de tamaños de arreglos de cromosomas de entrada y salida (si entran 10 hijos a mutación, el método debe devolver también 10 hijos, por ejemplo) y otro la validez de los cromosomas creados, seleccionados o modificados (utilizando los mismos criterios que en las pruebas unitarias de la clase Cromosoma). |
| **TOTAL** | 11 | - |

Si se desea observar el código de las pruebas mencionadas, este se encuentra junto al resto del código fuente del algoritmo memético en el **anexo F**.

### 6.2.3. Calibración de parámetros de ejecución

Para proceder con la calibración de parámetros, se decidió escoger entre 1 y 3 niveles para cada uno de los 8 parámetros de ejecución del algoritmo memético. En la **tabla 12** se muestran los valores elegidos para cada parámetro:

*Tabla 12. Elección de valores para parámetros de ejecución*

| **Parámetro** | **Valor bajo** | **Valor medio** | **Valor alto** |
|---|---|---|---|

| | | | |
|---|---|---|---|
| NumIter | - | 5 000 | 10 000 |
| TamPob | - | 100 | 200 |
| PorcCromCruzados | - | 0.5 | 0.75 |
| ProbMut | 0.5 | 0.75 | 1.00 |
| ProbBusq | - | 0.5 | 0.6 |
| PorcHijosIngresados | - | 0.75 | 0.9 |
| CantVecinosEvaluados | - | 40 | 50 |
| PorcIterEstancamiento | - | 0.3 | - |

Para cada combinación de valores de los 8 parámetros se ejecutará el algoritmo memético 10 veces y finalmente se compararán los promedios de tiempo de ejecución y *fitness* de la mejor, 10 mejores y 20 mejores soluciones para decidir una combinación ganadora. En todos los casos se realizará la optimización sobre un problema con 10 tablas y 6 sitios. Tras realizar las ejecuciones, se muestran en la **figura 5** las 10 mejores configuraciones según una priorización de fitness promedio de la mejor solución, fitness promedio de las 10 mejores soluciones, fitness promedio de las 20 mejores soluciones y tiempo de ejecución promedio:

| NumIter | TamPob | PorcCrom Cruzados | Prob Mut | Prob Busq | PorcHijosIn gresados | CantVecinos Eval | PorcIter Estanc | Fitness prom. mejor cromosoma | Fitness prom. mejores 10 cromosomas | Fitness prom. mejores 20 cromosomas | Tiempo ejec. prom. |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 10000 | 200 | 0.75 | 1.00 | 0.50 | 0.90 | 40 | 0.30 | 3.418393917 | 2.347092195 | 2.286961326 | 2151.385 |
| 10000 | 200 | 0.75 | 0.50 | 0.50 | 0.75 | 50 | 0.30 | 3.415233636 | 2.356624629 | 2.287517683 | 2396.209 |
| 10000 | 200 | 0.75 | 1.00 | 0.60 | 0.90 | 50 | 0.30 | 3.41125558 | 2.328608747 | 2.267848179 | 2159.734 |
| 5000 | 200 | 0.75 | 0.75 | 0.60 | 0.90 | 50 | 0.30 | 3.410748748 | 2.360951333 | 2.289709106 | 1437.239 |
| 10000 | 200 | 0.75 | 1.00 | 0.50 | 0.75 | 50 | 0.30 | 3.410149242 | 2.353422998 | 2.283890703 | 2362.58 |
| 5000 | 200 | 0.75 | 1.00 | 0.60 | 0.75 | 50 | 0.30 | 3.407815719 | 2.380500517 | 2.30931452 | 1129.536 |
| 10000 | 200 | 0.75 | 0.75 | 0.50 | 0.90 | 50 | 0.30 | 3.407812718 | 2.379315401 | 2.302254674 | 2233.305 |
| 10000 | 200 | 0.50 | 0.75 | 0.60 | 0.75 | 50 | 0.30 | 3.407779477 | 2.364561175 | 2.296618154 | 1925.817 |
| 10000 | 200 | 0.75 | 0.50 | 0.50 | 0.90 | 40 | 0.30 | 3.407489144 | 2.364850953 | 2.29626379 | 2060.819 |
| 10000 | 200 | 0.50 | 1.00 | 0.60 | 0.90 | 40 | 0.30 | 3.407334262 | 2.349753076 | 2.28995922 | 1602.49 |

*Figura 6. 10 mejores configuraciones de parámetros*

En base a estos resultados, la configuración de parámetros de ejecución que se utilizará es la siguiente:

- **NumIter:** 10 000
- **TamPob:** 200
- **PorcCromCruzados:** 0.75
- **ProbMut:** 1.00
- **ProbBusq:** 0.50
- **PorcHijosIngresados:** 0.90
- **CantVecinosEvaluados:** 40
- **PorcIterEstancamiento:** 0.30

El procedimiento de calibración ha sido validado por un especialista en algoritmia metaheurística mediante el llenado de un formulario de validación, el cual se encuentra en el **anexo N**.

## 6.3. Discusión

Tras el desarrollo del presente capítulo, se cuenta con una adaptación propia del algoritmo memético para tratar el problema de optimización de consultas SQL. La estructura y lógica del mismo ha sido planteada en pseudocódigo y la configuración de sus parámetros de ejecución ha sido seleccionada entre varias en base a la calidad de las soluciones que brindó. En el siguiente capítulo se implementará una adaptación del algoritmo genético para comparar sus resultados con los de este algoritmo con el fin de encontrar puntos de mejora y evaluar si los algoritmos meméticos son una propuesta viable para la optimización de consultas SQL en bases de datos distribuidas relacionales.

En el contexto de la literatura académica, el algoritmo ha sido desarrollado tomando en cuenta elementos de múltiples trabajos y conocimiento general de algoritmos evolutivos, pero estos han sido adaptados y alterados de tal forma que se haya creado una adaptación propia del algoritmo sin que se pierdan las contribuciones pasadas en el campo. Cabe resaltar que existen limitaciones en lo desarrollado: debido a falta de información en la literatura elegida, algunos aspectos del funcionamiento del algoritmo han sido basados en lógicas propias conversadas con el asesor de

tesis; y con respecto al rendimiento del algoritmo, este puede verse afectado por las especificaciones técnicas del computador del autor y por un conjunto limitado de configuraciones de parámetros y ejecuciones para cada una de estas.

# 7. Comparación con algoritmo genético

## 7.1. Introducción

El presente capítulo detalla el desarrollo del tercer objetivo específico **(O3): "Comparar el algoritmo memético propuesto y un algoritmo genético adaptado por el tesista, con el fin de obtener resultados mejores en términos de eficiencia y calidad de soluciones generadas."** Este objetivo está compuesto por tres resultados esperados: la codificación de la adaptación propia del algoritmo genético, la codificación de un módulo que permita la ejecución de ambos algoritmos para una misma configuración del problema de optimización y la experimentación numérica sobre los resultados del módulo anterior para determinar qué algoritmo tiene mejor rendimiento.

## 7.2. Resultados alcanzados

### 7.2.1. Codificación de algoritmo genético

La codificación de la adaptación del algoritmo genético ha reutilizado múltiples clases creadas para el algoritmo memético (**Tabla**, **Cromosoma**, **DatosEntrada**, **ParametrosAlgoritmo**. **MultiOutputStream** y **ComparadorCromosomas**). Esto se debe en gran medida a que la forma de definir el problema de optimización (las características de tablas y sitios de la base de datos) y la estructura de un cromosoma es la misma para ambos algoritmos. Así, los principales cambios se encuentran en la clase propia del algoritmo en cuestión, en este caso llamada **AlgoritmoGenetico**. La estructura del proyecto de Java se puede observar en la **figura 7**:


*Figura 7. Estructura del proyecto en Java del algoritmo genético*

La lógica resumida del programa es prácticamente la misma que la del algoritmo memético. Se empieza por configurar la salida y el error estándar para que impriman tanto a consola como a archivos de texto utilizando la clase **MultiOutputStream**, para luego leerse un archivo de datos de entrada, cuyo contenido se almacenará en objetos de las clases **DatosEntrada**, **Tabla** y **ParametrosAlgoritmo**. Se decidió utilizar la misma estructura del archivo de datos descrita en el capítulo anterior con el fin de facilitar el funcionamiento del módulo de ejecución de ambos algoritmos. Finalmente, se crea un objeto **AlgoritmoGenetico** que usa los parámetros almacenados en la instancia del objeto *singleton* **DatosEntrada** y se ejecuta el algoritmo.

Las diferencias principales entre las implementaciones de ambos algoritmos (en las clases **AlgoritmoMemetico** y **AlgoritmoGenetico**), aparte del uso de un operador de búsqueda local para el memético, son las siguientes:

- **Parámetros de ejecución utilizados:** Puesto que el algoritmo genético no utiliza un operador de búsqueda local, el mismo no

considera los parámetros ***probBusq*** y ***cantVecinosEvaluados***, presentes en el archivo de datos de entrada.

- **Operador de selección de cromosomas padres**: Se ha decidido utilizar el método de selección por torneo en lugar del anterior método de selección por ruleta. Para la elección de un padre en este nuevo método, se toman dos individuos diferentes de la población de manera aleatoria y se selecciona el que tiene mayor *fitness* de los dos. El pseudocódigo del nuevo operador de selección es el siguiente:

```
INICIO seleccionarPadres(pob, porcCromCasados, calidades)
1:  padres= [], i=0, tamPob = tamaño(pob)
2:  cantidadPadres= tamaño(pob)*porcCromCasados //Debe ser par
3:  Mientras i<cantidadPadres Hacer:
4:  | indice1= numEnteroAleatorioEntre(0,tamPob)
5:  | indice2= numEnteroAleatorioEntre(0,tamPob)
6:  | padreSeleccionado, crom1= pob[indice1]
7:  | crom2= pob[indice2]
8:  | Si (calidades[indice1]>=calidades[indice2]) Entonces:
9:  | | padreSeleccionado= crom1
10: | Sino:
11: | | padreSeleccionado= crom2
12: | FinSi
13: | padres[i] = padreSeleccionado
14: | i++
15: FinMientras
16: Retornar padres
FIN seleccionarPadres
```

- **Operador de mutación:** Se ha decidido utilizar el método de mutación por inversión en lugar del anterior método de mutación por intercambio. La mutación de un cromosoma en este nuevo método consiste en la elección de dos índices diferentes del arreglo de pares tabla-sitio del cromosoma, los cuales delimitan un segmento del mismo. A continuación, el orden de los elementos en dicho segmento del arreglo será invertido. El cambio de operador se debe a que la inversión también es válida para cromosomas basados en permutaciones de valores enteros y a que la misma añadirá exploración de

soluciones respecto a la mutación por intercambio con la finalidad de compensar la carencia de búsqueda local en el algoritmo genético. El pseudocódigo del nuevo operador de mutación es el siguiente:

```
INICIO mutarHijos(hijos, probMut)
1:  hijos2= [], i=0
2:  Mientras i<tamaño(hijos) Hacer:
3:  | hijoOrig= hijos[i], hijoMutado= copiar(hijoOrig)
4:  | aleatorio= numRealAleatorioEntre(0,1)
5:  | Si aleatorio<=probMut Hacer:
6:  | | indiceIni= numEnteroAleatorioEntre(0,tamaño(hijoOrig)-1)
7:  | | indiceFin
8:  | | Hacer:
9:  | | | indiceFin= numEnteroAleatorioEntre(0,tamaño(hijoOrig))
10: | | Mientras indiceIni>=indiceFin
11: | | invertirSegmentoArreglo(hijoMutado,indiceIni,indiceFin)
12: | FinSi
13: | hijos2[i]= hijoMutado
14: | i++
15: FinMientras
16: Retornar hijos2
FIN mutarHijos
```

Si se desea leer el código fuente, en el **anexo G** se encuentra el enlace al archivo comprimido que contiene el proyecto en Java del algoritmo genético.

Para el desarrollo del presente capítulo también se realizó una calibración de parámetros de ejecución del algoritmo genético. Se decidió escoger entre 1 y 3 niveles para cada uno de los 6 parámetros de ejecución del algoritmo genético. En la **tabla 13** se muestran los valores elegidos para cada parámetro:

*Tabla 13. Elección de valores para parámetros de ejecución (genético)*

| **Parámetro** | **Valor bajo** | **Valor medio** | **Valor alto** |
|---|---|---|---|
| NumIter | - | 5 000 | 10 000 |
| TamPob | - | 100 | 200 |
| PorcCromCruzados | - | 0.5 | 0.75 |

| | | | |
|---|---|---|---|
| ProbMut | 0.5 | 0.75 | 1.00 |
| PorcHijosIngresados | 0.5 | 0.75 | 0.9 |
| PorcIterEstancamiento | - | 0.3 | - |

La calibración se realizará ejecutando el algoritmo 10 veces para cada combinación de valores de los 6 parámetros. Al igual que con el algoritmo memético, la ejecución se realiza sobre un problema de optimización con una base de datos de 10 tablas y 6 sitios. Los resultados de las ejecuciones para las 10 mejores configuraciones se pueden observar en la **figura 8**:

| NumIter | TamPob | PorcCrom Cruzados | ProbMut | PorcHijos Ingresad os | PorcIter Estanc | Fitness prom. mejor cromosoma | Fitness prom. mejores 10 cromosomas | Fitness prom. mejores 20 cromosomas | Tiempo ejec. prom. |
|---|---|---|---|---|---|---|---|---|---|
| 5000 | 200 | 0.75 | 0.50 | 0.90 | 0.30 | 3.137348012 | 2.363792567 | 2.292987474 | 411.0727 |
| 10000 | 200 | 0.75 | 1.00 | 0.75 | 0.30 | 3.132882683 | 2.363594186 | 2.299217125 | 695.4376 |
| 10000 | 200 | 0.75 | 1.00 | 0.90 | 0.30 | 3.111311001 | 2.356157161 | 2.289651939 | 750.2449 |
| 10000 | 200 | 0.75 | 0.50 | 0.90 | 0.30 | 3.110182929 | 2.354088233 | 2.2865951 | 701.1574 |
| 10000 | 200 | 0.75 | 0.75 | 0.75 | 0.30 | 3.102959821 | 2.349212109 | 2.284335025 | 750.576 |
| 10000 | 100 | 0.75 | 0.75 | 0.90 | 0.30 | 3.102358074 | 2.29180263 | 2.211069387 | 282.0345 |
| 10000 | 200 | 0.50 | 1.00 | 0.90 | 0.30 | 3.097285222 | 2.353087629 | 2.285738669 | 487.3602 |
| 10000 | 200 | 0.50 | 1.00 | 0.50 | 0.30 | 3.095145248 | 2.357709696 | 2.293791792 | 393.0754 |
| 10000 | 200 | 0.50 | 0.50 | 0.90 | 0.30 | 3.092820692 | 2.351643535 | 2.287595874 | 500.1813 |
| 5000 | 200 | 0.50 | 1.00 | 0.90 | 0.30 | 3.09080112 | 2.355055407 | 2.29265826 | 269.7116 |

*Figura 8. 10 mejores configuraciones de parámetros (genético)*

En base a estos resultados, la configuración de parámetros de ejecución que se utilizará es la siguiente:

- **NumIter:** 5 000
- **TamPob:** 200
- **PorcCromCruzados:** 0.75
- **ProbMut:** 0.50
- **PorcHijosIngresados:** 0.90
- **PorcIterEstancamiento:** 0.30

Finalmente, se describirá brevemente el uso de metodologías para el desarrollo de este resultado. Comenzando con la metodología Kanban, se utilizó la misma configuración de tablero

Kanban descrita en la sección de codificación del algoritmo memético, y se crearon solamente 3 tareas: la **adaptación de la estructura de la clase AlgoritmoMemetico** (que implicaba cambiar los parámetros de ejecución que se toman del objeto **DatosEntrada**), la **codificación del operador de selección por torneo** y la **codificación del operador de mutación por inversión**. Por otro lado, para las pruebas unitarias fue posible reutilizar las codificadas para el algoritmo memético, pues las mismas verificaban principalmente que los operadores del algoritmo retornen arreglos con cromosomas válidos, condición que se busca mantener para los nuevos operadores del algoritmo genético.

### 7.2.2. Codificación de módulo para comparación de algoritmos

El desarrollo del módulo para la comparación de algoritmos solamente supuso crear un proyecto de Java que incluya todas las clases desarrolladas en los proyectos de ambos algoritmos. Una vez que se comprobó que ambos algoritmos se podían ejecutar dentro de dicho proyecto a partir de un mismo archivo de datos de entrada, se procedió a codificar un método en la clase Main que se encargue de ejecutar ambos algoritmos un número determinado de veces. Cada algoritmo debe ejecutarse con su respectiva configuración de parámetros decidida tras las calibraciones de los mismos, por lo que la estructura del archivo de prueba ha cambiado ligeramente: ahora, en la **sección 5** del archivo pueden existir múltiples filas de parámetros de ejecución. El programa será capaz de almacenar todas las diferentes configuraciones en un arreglo de objetos **ParametrosAlgoritmo**. Esto nos permite ingresar dos configuraciones diferentes: una para el algoritmo memético y otra para el genético.

La lógica que el nuevo método codificado sigue es la siguiente: tras leerse el archivo de datos de entrada, se toman las dos

primeras configuraciones de parámetros de ejecución (si hay configuraciones adicionales, estas son ignoradas) y se instancia un objeto **AlgoritmoMemetico** con la primera configuración y otro objeto **AlgoritmoGenetico** con la segunda. A continuación, se entra a un bucle cuya cantidad de iteraciones es un parámetro llamado *cantEjecuciones*. En dicho bucle se ejecutan el algoritmo memético primero y luego el genético. Tras ambas ejecuciones, se imprime el *fitness* de la mejor solución, el *fitness* promedio de las 10 y 20 mejores soluciones y el tiempo de ejecución para ambos algoritmos. De esta manera, se obtiene un archivo de texto separado por caracteres ";" que contiene los resultados de la cantidad de ejecuciones deseada para ambos algoritmos. El código fuente del proyecto en Java de este módulo puede ser accedido desde el enlace ubicado en el **anexo H**.

### 7.2.3. Experimentación numérica con resultados de ejecución de algoritmos

Esta sección se estructurará en 4 subsecciones para explicar el desarrollo de la experimentación numérica.

**Obtención de datos:**

El conjunto de datos a utilizarse en la experimentación numérica se obtuvo utilizando el módulo de comparación desarrollado en la sección anterior con las siguientes configuraciones:

- Número de ejecuciones: **1 000**
- Problema a optimizar: 3 configuraciones (**C1 con 5 tablas y 3 sitios**; **C2 con 8 tablas y 4 sitios**; y **C3 con 10 tablas y 6 sitios**)

Con esto se obtuvieron 1000 registros de ejecución para cada pareja algoritmo-configuración del problema, formándose así un conjunto de datos de 6000 registros de ejecución para los algoritmos

memético y genético, cada uno ejecutándose sobre las configuraciones C1, C2 y C3. Cada registro contiene 3 variables dependientes: el *fitness* de la mejor solución obtenida, el *fitness* promediado de las 10 mejores soluciones obtenidas y el tiempo de ejecución en milisegundos. Véase el **anexo I** si se desea observar el conjunto de datos.

**Consideraciones:**

- **Objetivo:**

  Determinar si el algoritmo utilizado en la optimización tiene un efecto sobre cada una de las variables de respuesta a evaluar.
- **Población:**

  Datos de ejecución de los algoritmos obtenidos mediante el módulo de comparación.
- **Variables de respuesta:**

  -*Fitness* de la mejor solución encontrada

  -*Fitness* promedio de las 10 mejores soluciones encontradas

  -Tiempo de ejecución del algoritmo utilizado
- **Factores:**

  -Algoritmo utilizado (F1): Memético, Genético.

  -Configuración del problema (F2): C1, C2, C3.
- **Tratamientos:**

  Los tratamientos corresponden a todas las combinaciones de los factores F1 y F2. Para cada tratamiento se tiene 1000 mediciones de las variables de respuesta.
- **Nivel de significación:**

  Para las pruebas a realizarse se utilizará un nivel de significación estadística de **5% (0.05)**.
- **Independencia de los datos:**

  Los tratamientos (combinación algoritmo-configuración del problema) son independientes entre sí, pues cada uno

corresponde a una configuración del problema diferente o a un algoritmo diferente. Incluso para un mismo algoritmo, el factor de semi aleatoriedad introducido por los operadores brinda independencia a cada ejecución.

- **Normalidad de los datos:**

  Para evaluar la normalidad de cada tratamiento se utilizará la prueba de normalidad de Pearson. Así, se plantean las siguientes hipótesis:

  **H0**: Los datos del tratamiento T tienen una distribución normal.

  **H1**: Los datos del tratamiento T no tienen una distribución normal.

  Tras realizar la prueba sobre cada variable de respuesta de cada tratamiento, se obtuvieron un conjunto de *p-values*, los cuales se muestran en la **tabla 14**.

*Tabla 14. p-values de pruebas de normalidad*

| **Configuración problema** | **Algoritmo** | **Fitness mejor sol.** | **Fitness 10 mejores** | **Tiempo ejecución** |
|---|---|---|---|---|
| C1 | Memético | <2.22e-16 | 0.00466 | <2.22e-16 |
| | Genético | <2.22e-16 | **0.12630** | <2.22e-16 |
| C2 | Memético | <2.22e-16 | **0.11504** | <2.22e-16 |
| | Genético | <2.22e-16 | **0.36354** | <2.22e-16 |
| C3 | Memético | <2.22e-16 | **0.77764** | <2.22e-16 |
| | Genético | 0.0000018 | **0.49916** | <2.22e-16 |

De todas las pruebas realizadas, solamente 5 resultaron en un *p-value* mayor a 0.05. Así, con un nivel de significación de 5% es posible afirmar que 13 de los subconjuntos de datos no presentan normalidad (pues se rechaza la hipótesis nula), y 5 sí lo hacen (pues no se puede rechazar la hipótesis nula).

Ya que no todos los subconjuntos de datos presentan normalidad, no es posible utilizar pruebas de significación estadística que tengan como condición la normalidad de datos. Así, se decidió utilizar la prueba U de Mann-Whitney, la cual compara medianas de 2 conjuntos de datos.

**Experimentación realizada:**

La prueba U de Mann-Whitney se aplicará sobre 2 subconjuntos de datos que coincidan en la configuración del problema y en la variable de respuesta medida (por ejemplo, los tiempos de ejecución de algoritmos memético y genético para la configuración C1).

- **Fitness de mejor solución encontrada:**

  Se definen las siguientes variables estadísticas:

  $\mathbf{A_i}$ = *Fitness* de mejor solución para algoritmo memético con configuración Ci

  $\mathbf{B_i}$ = *Fitness* de mejor solución para algoritmo genético con configuración Ci

  donde i = 1, 2, 3. Ahora se procede a definir las hipótesis:

  **H0**: $Me(A_i) = Me(B_i)$

  **H1**: $Me(A_i) > Me(B_i)$

  Luego de esto, se procede a realizar las pruebas sobre los pares de subconjuntos. A continuación se muestran los *p-values* obtenidos para cada prueba:

  -Configuración C1: **1**

  -Configuración C2: **< 2.2e-16**

  -Configuración C3: **< 2.2e-16**

A partir de estos resultados y con un nivel de significación de 5% es posible afirmar que la mediana del *fitness* de la mejor solución encontrada por el algoritmo memético es mayor que la de la encontrada por el algoritmo genético para las configuraciones C2 y C3 (pues se rechaza la hipótesis nula). Para la configuración C1, no es posible afirmar esto. Para observar el comportamiento del *fitness*, en la **figura 9** se tiene un gráfico de líneas.

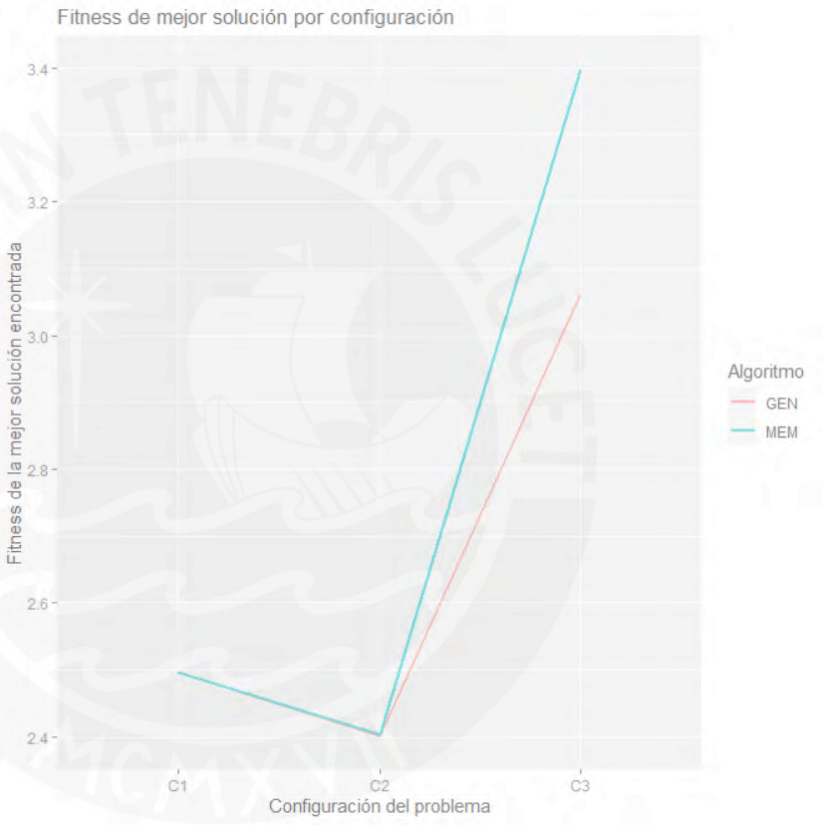



*Figura 9. Gráfico de líneas de fitness de mejor solución por configuración del problema*

- **Fitness de 10 mejores soluciones:**

  Se definen las siguientes variables estadísticas:

  $\mathbf{A_i}$ = *Fitness* promedio de 10 mejores soluciones para algoritmo memético con configuración Ci

  $\mathbf{B_i}$ = *Fitness* promedio de 10 mejores soluciones genético con configuración Ci

donde i = 1, 2, 3. Ahora se procede a definir las hipótesis:

**H0**: $Me(A_i) = Me(B_i)$

**H1**: $Me(A_i) > Me(B_i)$

Luego de esto, se procede a realizar las pruebas sobre los pares de subconjuntos. A continuación se muestran los *p-values* obtenidos para cada prueba:

-Configuración C1: **0.04073**

-Configuración C2: **0.06779**

-Configuración C3: **0.08236**

A partir de estos resultados y con un nivel de significación de 5% es posible afirmar que la mediana del *fitness* promedio de las 10 mejores soluciones encontradas por el algoritmo memético es mayor que la de las encontradas por el algoritmo genético para la configuración C1 (pues se rechaza la hipótesis nula). No es posible afirmar esto para las configuraciones C2 y C3. Para observar el comportamiento del *fitness* promedio, en la **figura 10** se tiene un segundo gráfico de líneas.

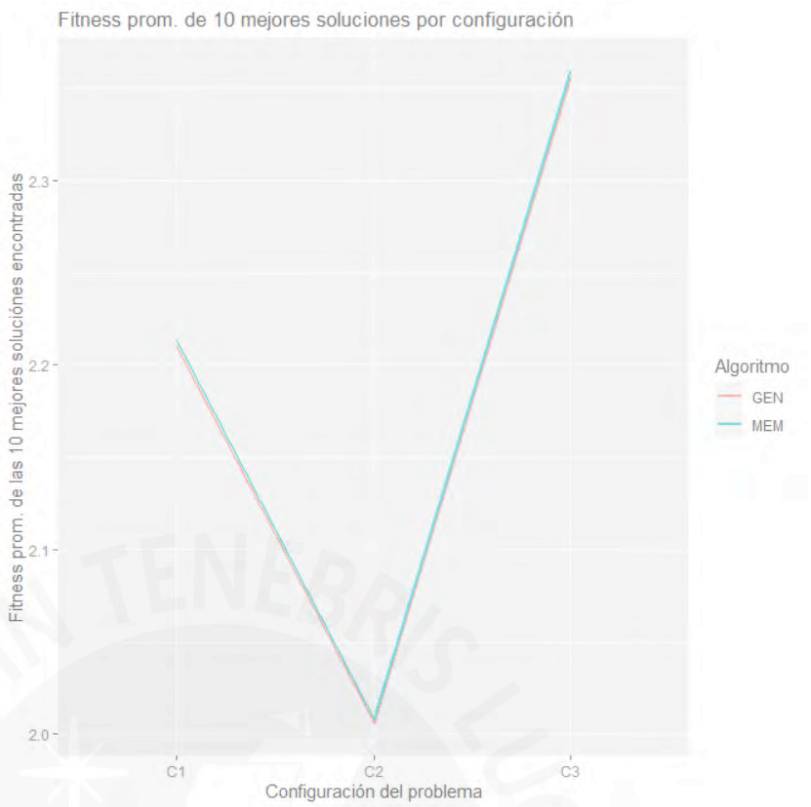



*Figura 10. Gráfico de líneas de fitness promedio de 10 mejores soluciones por configuración del problema*

- **Tiempo de ejecución:**

Se definen las siguientes variables estadísticas:

$\mathbf{A_i}$ = Tiempo de ejecución del algoritmo memético con configuración Ci

$\mathbf{B_i}$ = Tiempo de ejecución del algoritmo genético con configuración Ci

donde i = 1, 2, 3. Ahora se procede a definir las hipótesis:

**H0**: $Me(A_i) = Me(B_i)$

**H1**: $Me(A_i) > Me(B_i)$

Luego de esto, se procede a realizar las pruebas sobre los pares de subconjuntos. A continuación se muestran los *p-values* obtenidos para cada prueba:

-Configuración C1: **< 2.2e-16**

-Configuración C2: **< 2.2e-16**

-Configuración C3: **< 2.2e-16**

A partir de estos resultados y con un nivel de significación de 5% es posible afirmar que la mediana del tiempo de ejecución del algoritmo memético es mayor que la del obtenido por el algoritmo genético para las tres configuraciones C1, C2 y C3 (pues se rechaza la hipótesis nula). Para observar el comportamiento del tiempo de ejecución, en la **figura 11** se tiene un tercer gráfico de líneas.

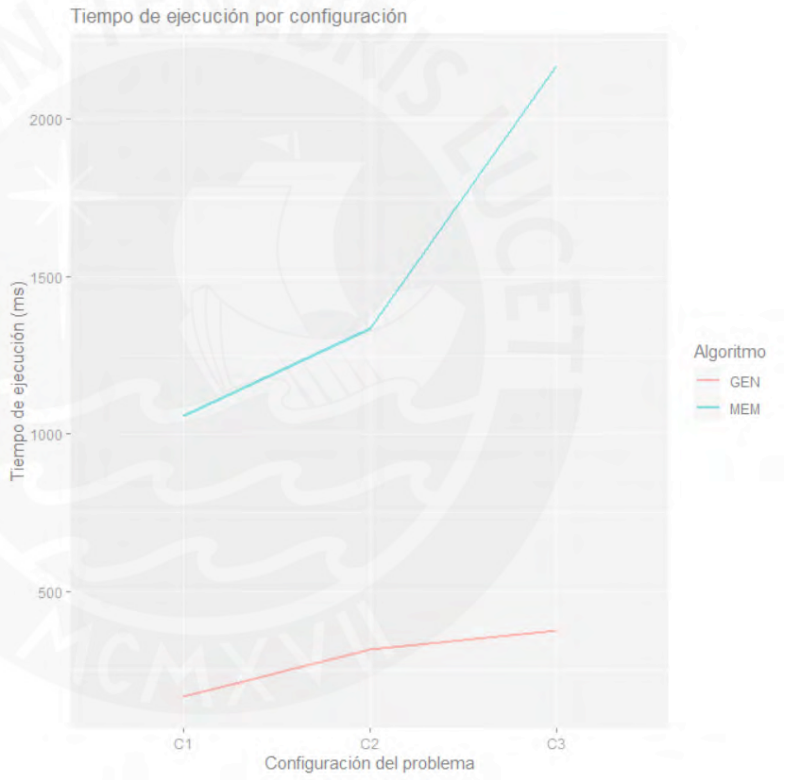



*Figura 11. Gráfico de líneas de tiempo de ejecución por configuración del problema*

**Conclusiones:**

A partir de las pruebas realizadas y los resultados obtenidos, es posible concluir que el algoritmo memético produce mejores soluciones que el genético para problemas de complejidad media o mayor. En problemas de baja complejidad, ambos algoritmos

pueden generar soluciones de la misma calidad. Sin embargo, las diferencias de calidad entre ambos algoritmos dejan de ser considerables cuando se considera un conjunto de n mejores soluciones de las poblaciones finales (para la presente experimentación numérica se utilizó n=10). Además, es importante considerar que la obtención de mejores soluciones en el algoritmo memético viene de la mano con tiempos de ejecución mayores que los obtenidos normalmente en el algoritmo genético.

La experimentación realizada en esta sección ha sido validada por un especialista en algoritmia metaheurística, cuya constancia es un formulario de validación que se encuentra en el **anexo O**.

## 7.3. Discusión

Tras el desarrollo del presente capítulo, se cuenta con una adaptación del algoritmo genético para tratar el problema de optimización de consultas SQL. Dicha adaptación se ha utilizado como punto de comparación para evaluar el rendimiento de la adaptación propia del algoritmo memético (desarrollada en el capítulo anterior) mediante herramientas como el módulo de comparación implementado en Java y la realización de experimentación numérica. Los resultados de esta última han sido positivos para el algoritmo memético, pero también resaltaron posibles puntos de mejora como el tiempo de ejecución del mismo.

En el contexto de la literatura académica, si bien la adaptación del algoritmo genético no está directamente basada en un trabajo específico, la misma sigue conceptos de modelo de costos utilizados para la implementación del algoritmo memético y utiliza operadores de modificación de cromosomas que son conocidos y ampliamente usados en el ámbito académico.

Finalmente, respecto a limitaciones de lo desarrollado en este capítulo, podría resaltarse que debido a la no normalidad de varios de los conjuntos de datos de la experimentación numérica, fue necesario utilizar una prueba no

paramétrica para evaluar el comportamiento de *fitness* obtenido y tiempo de ejecución de ambos algoritmos. Si bien las pruebas no paramétricas presentan una mayor robustez que las paramétricas (pues son utilizables en una mayor variedad de escenarios al no contar con tantos supuestos), también tienen una menor potencia estadística que las pruebas paramétricas, lo que implica que tienen una menor probabilidad de rechazar la hipótesis nula (para demostrar lo que se desea evaluar).

# 8. Conclusiones

Tras el desarrollo del presente proyecto de tesis, se obtuvo lo siguiente respecto a los objetivos específicos especificados:

- **O1:** Una serie de variables y restricciones relacionadas al problema de optimización de consultas SQL en bases de datos distribuidas relacionales, las cuales se tuvieron en cuenta para construir una función objetivo a optimizar mediante un algoritmo metaheurístico. Algunas de las variables más importantes para la optimización fueron los datos específicos a cada tabla (tamaños de tabla y cardinalidades de columnas) y la distribución de tablas en sitios, que es la variable que caracteriza el caso de bases de datos distribuidas.
- **O2:** Una adaptación propia del algoritmo memético para realizar la optimización de la función objetivo mencionada anteriormente que utiliza una configuración de parámetros de ejecución concebida a partir de un proceso de calibración.
- **O3:** Una comparación entre el algoritmo memético y una adaptación, también propia, del algoritmo genético en términos de calidad de soluciones y tiempos de ejecución. En este caso, se decidió optar por una adaptación propia del algoritmo genético en lugar de una partiendo de la literatura debido a que los trabajos encontrados, si bien explicaban a grandes rasgos la lógica de optimización utilizada, no entraban en detalles respecto a la implementación en código de los algoritmos propuestos en cada trabajo.

La revisión y análisis de la literatura académica relacionada al tema del proyecto fue de vital importancia para determinar puntos de importancia y tomar decisiones durante el desarrollo de los resultados esperados ligados al planteamiento del problema de optimización (variables/restricciones, estructuras de datos y función objetivo), pero para algunos de estos puntos también fueron importantes las conversaciones con el asesor del proyecto y los aportes propios, esto con el fin de superar problemas ocasionados por la falta de información y ofrecer nuevas ideas respecto al problema analizado.

Por otro lado, las fases de adaptación, codificación y comparación de los algoritmos memético y genético fueron sostenidas en mayor medida por aportes y

decisiones propias del tesista, pero siempre considerando buenas prácticas y métodos conocidos en el campo (siendo ejemplos de esto el seguimiento de trabajo mediante un tablero Kanban, la creación y uso de pruebas unitarias, y la realización de experimentación numérica con pruebas de significación estadística).

Tomando en cuenta lo anterior y todo lo desarrollado en el presente trabajo, es posible afirmar que se cumplieron todos los objetivos específicos planteados: se determinaron las variables del problema, estructuras de datos que las representen, y se construyó una función objetivo exitosamente; las variables y función objetivo fueron utilizadas para adaptar la lógica de los algoritmos memético y genético de tal forma que estos puedan optimizar una abstracción del problema de optimización de consultas SQL; y los dos algoritmos adaptados fueron comparados mediante una metodología estructurada. Sin embargo, se buscaba inicialmente que dicha comparación resultara en una victoria del algoritmo memético tanto en términos de calidad de soluciones como de tiempos de ejecución, pero solamente se logró dicha victoria en el primer aspecto. No obstante, los tiempos de ejecución alcanzados para ambos algoritmos fueron completamente aceptables para un escenario práctico (sobrepasando ligeramente los dos segundos en el peor de los casos), por lo que podría considerarse que, al fin y al cabo, la comparación obtuvo resultados favorables.

Finalmente, ya que todos los objetivos específicos fueron alcanzados durante el desarrollo del proyecto, es posible afirmar que el objetivo general, "**implementar un algoritmo memético para resolver el problema de optimización de consultas en bases de datos distribuidas relacionales**", también fue alcanzado. Si bien el algoritmo memético adaptado no es directamente utilizable en un escenario práctico (pues considera una abstracción del problema a optimizar y el proyecto no considera dentro de su alcance la integración del algoritmo a un sistema de administración de base de datos), su desarrollo ciertamente indica que esta clase de algoritmo metaheurístico es una opción completamente viable (y capaz de brindar mejores soluciones que el algoritmo genético cuando la complejidad del problema crece) para afrontar la optimización de consultas SQL en bases de datos distribuidas relacionales.

## 9. Referencias

Oracle, s.f. *Java™ Programming Language*. Recuperado el 13 de junio del 2021 a partir de: https://docs.oracle.com/javase/7/docs/technotes/guides/language/

R Project, s.f. *What is R?* Recuperado el 13 de junio del 2021 a partir de: https://www.r-project.org/about.html

RStudio, 2021. *Frequently asked questions: What is RStudio?* Recuperado el 13 de junio del 2021 a partir de: https://support.rstudio.com/hc/en-us/articles/200486548-Frequently-Asked-Questions

Ahmad, M. O., Markkula, J., & Oivo, M. (2013, September). *Kanban in software development: A systematic literature review.* In 2013 39th Euromicro conference on software engineering and advanced applications (pp. 9-16). IEEE.

Corona, E., & Pani, F. E. (2013). *A review of lean-kanban approaches in the software development*. WSEAS transactions on information science and applications, 10(1), 1-13.

Kirovska, N., & Koceski, S. (2015). *Usage of Kanban methodology at software development teams*. Journal of applied economics and business, 3(3), 25-34.

Luo, L. (2001). *Software testing techniques*. Institute for software research international Carnegie mellon university Pittsburgh, PA, 15232(1-19), 19.

Jovanović, I. (2006). *Software testing methods and techniques*. The IPSI BgD Transactions on Internet Research, 30.

Massey, A., & Miller, S. J. (2006). *Tests of hypotheses using statistics*. Mathematics Department, Brown University, Providence, RI, 2912, 1-32.

Bennett, K. P., Ferris, M. C., & Ioannidis, Y. E. (1991). *A genetic algorithm for database query optimization*. University of Wisconsin-Madison Department of Computer Sciences.

Oracle, s.f. *What is a Relational Database?*. Recuperado el 04 de mayo del 2021 a partir de: https://www.oracle.com/database/what-is-a-relational-database/

IBM Cloud Learn Hub, 2019. *Relational Databases Explained*. Recuperado el 04 de mayo del 2021 a partir de: https://www.ibm.com/cloud/learn/relational-databases

IBM, 2019. *Documentation: Informix Servers*. Recuperado el 04 de mayo del 2021 a partir de: https://www.ibm.com/docs/en/informix-servers/12.10?topic=overview-some-basic-concepts

Oracle, 2017. *Database SQL Tuning Guide*. Recuperado el 04 de mayo del 2021 a partir de: https://docs.oracle.com/database/121/TGSQL/toc.htm

LEVITIN A., 2012. *Introduction to The Design and Analysis of Algorithms. 3ra edición*. USA: Pearson,. ISBN 0-13-231681-1

Myalapalli, V. K., & Savarapu, P. R. (2014, December). *High performance SQL*. In *2014 Annual IEEE India Conference (INDICON)* (pp. 1-6). IEEE.

Wijesiriwardana, C., & Firdhous, M. F. M. (2019, September). *An Innovative Query Tuning Scheme for Large Databases*. In 2019 International Conference on Data Science and Engineering (ICDSE) (pp. 154-159). IEEE.

Du, K. L., & Swamy, M. N. S. (2016). *Search and optimization by metaheuristics. Techniques and Algorithms Inspired by Nature*.

Ławrynowicz, A. (2011). *Genetic algorithms for solving scheduling problems in manufacturing systems*. Ladies and Gentlemen, 7.

Gross, B. (2003). *A solar energy system that tracks the sun*. TED. Recuperado de: https://www.ted.com/talks/bill_gross_a_solar_energy_system_that_tracks_the_sun

Ryan, C. (2003). *Evolutionary Algorithms and Metaheuristics*. In Encyclopedia of Physical Science and Technology (Third Edition) (pp. 673-685).

Garg, P. (2009). *A Comparison between Memetic algorithm and Genetic algorithm for the cryptanalysis of Simplified Data Encryption Standard algorithm*. In International Journal of Network Security & Its Applications (IJNSA).

Aguilar, J., & Colmenares, A. (1998). Resolution of pattern recognition problems using a hybrid genetic/random neural network learning algorithm. Pattern Analysis and Applications, 1(1), 52-61.

Karkavitsas, G. V., & Tsihrintzis, G. A. (2011). Automatic music genre classification using hybrid genetic algorithms. In Intelligent Interactive Multimedia Systems and Services (pp. 323-335). Springer, Berlin, Heidelberg.

Harris, S. P., & Ifeachor, E. C. (1998). Automatic design of frequency sampling filters by hybrid genetic algorithm techniques. IEEE Transactions on Signal Processing, 46(12), 3304-3314.

Sharma, M., Singh, G., Singh, R., & Singh, J. (2015, June). *Design and analysis of stochastic query optimizer for biobank databases*. In *2015 15th International Conference on Computational Science and Its Applications* (pp. 47-51). IEEE.

Sharma, M., Virk, R., & Singh, G. (2016, March). *Parametric analysis of different GA based distributed DSS Query Optimizer models*. In *2016 International Conference on Computational Techniques in Information and Communication Technologies (ICCTICT)* (pp. 25-31). IEEE.

Ban, W., Lin, J., Tong, J., & Li, S. (2015, December). Query optimization of distributed database based on parallel genetic algorithm and max-min ant system. In *2015 8th International Symposium on Computational Intelligence and Design (ISCID)* (Vol. 2, pp. 581-585). IEEE.

Abdalla, M. H., & Karabatak, M. (2020, June). To Review and Compare Evolutionary Algorithms in Optimization of Distributed Database Query. In *2020 8th International Symposium on Digital Forensics and Security (ISDFS)* (pp. 1-5). IEEE.

Yao, M. (2017, November). A distributed database query optimization method based on genetic algorithm and immune theory. In *2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS)* (pp. 762-765). IEEE.

Liu, S., & Xu, X. (2016, July). Distributed database query based on improved genetic algorithm. In *2016 3rd International Conference on Information Science and Control Engineering (ICISCE)* (pp. 348-351). IEEE.

Zheng, W., Jin, X., Deng, F., Mo, S., Qu, Y., Yang, Y., ... & Xie, Z. (2018, June). Database query optimization based on parallel ant colony algorithm. In 2018 IEEE 3rd International Conference on Image, Vision and Computing (ICIVC) (pp. 653-656). IEEE.

Mohsin, S. A., Darwish, S. M., & Younes, A. (2021). QIACO: A Quantum Dynamic Cost Ant System for Query Optimization in Distributed Database. *IEEE Access*, *9*, 15833-15846.

Yinghua, H., Yanchun, M., & Dongfang, Z. (2015, June). The multi-join query optimization for smart grid data. In *2015 8th International Conference on Intelligent Computation Technology and Automation (ICICTA)* (pp. 1004-1007). IEEE.

Chande, S. V. (2019). Database Query Optimization using Genetic Algorithms: A Systematic Literature Review. *International Journal of Advanced Trends in Computer Science and Engineering*, *8*(5), 1903-1913.

Kumar, T. V., & Yadav, M. (2017). Generating Distributed Query Plans Using Modified Cuckoo Search Algorithm. In *Proceedings of Sixth International Conference on Soft Computing for Problem Solving* (pp. 128-140). Springer, Singapore.

Panahi, V., & Navimipour, N. J. (2019). Join query optimization in the distributed database system using an artificial bee colony algorithm and genetic operators. *Concurrency and Computation: Practice and Experience*, *31*(17), e5218.

LAKSHMI, S. V., & VATSAVAYI, V. K. (2017). TEACHER-LEARNER & MULTI-OBJECTIVE GENETIC ALGORITHM BASED QUERY OPTIMIZATION

APPROACH FOR HETEROGENEOUS DISTRIBUTED DATABASE SYSTEMS. *Journal of Theoretical & Applied Information Technology*, *95*(8).

Eiben, A. E., & Smith, J. E. (2003). Introduction to evolutionary computing (Vol. 53, p. 18). Berlin: springer

# Anexos

## Anexo A: Enlace al formulario de extracción en excel

https://docs.google.com/spreadsheets/d/1TK-r82KOYMkT-5NRM5LmuU6bb5AY1FAPLfkLulOsb7E/edit?usp=sharing

# Anexo B: Plan de Proyecto

## 1. Justificación

En la actualidad, los datos tienen una importancia enorme para las organizaciones tanto para realizar y mantener sus operaciones como para desarrollar estrategias que les permitan obtener ventajas competitivas y aprovechar oportunidades. En este sentido, el contar con Sistemas de Administración de Bases de Datos (o DBMS por sus siglas en inglés) que soporten las necesidades crecientes de las organizaciones es algo crítico.

Existen múltiples caminos en los que se puede mejorar el rendimiento de los DBMS o resolver problemas relacionados al mismo, dirigidos por ejemplo a la administración de las estructuras de base de datos o a la red que permite la conexión de la misma con los usuarios. Sin embargo, se estima que el 80% de los problemas de rendimiento en bases de datos pueden solucionarse realizando ajustes y mejoras dirigidas a las sentencias SQL que utilizan los usuarios (Myalapalli V.,Savarapu P., 2015).

Tras considerar lo anterior, no es razonable pedirles a los usuarios que conozcan y utilicen siempre las sentencias SQL más eficientes para cada caso particular de obtención de datos. El que la extensión y complejidad de las consultas crezca a la vez que lo hace el volumen de datos que manejan las organizaciones solo reafirma este punto. Entonces, es necesario desarrollar mecanismos que optimicen las consultas realizadas por los usuarios para brindarles los datos que buscan en el menor tiempo posible. Podemos afirmar así que el desarrollo y adaptación de algoritmos para la optimización de consultas SQL tiene relevancia como un tema de investigación.

El presente proyecto de tesis busca tiene como propósito proponer e implementar una adaptación del algoritmo memético para la optimización de consultas SQL, y compararla con una propuesta de algoritmo genético encontrada en la literatura académica sobre el tema. La realización de dicho proyecto puede producir beneficios en el sentido de: plantear un nuevo método

de solución viable (y potencialmente aplicable) para el problema de optimización de consultas; contribuir al conocimiento existente en este campo de investigación; y servir de influencia y/o referencia para estudios futuros en la comunidad académica.

## 2. Viabilidad

En esta sección se escribirá sobre la viabilidad del proyecto de tesis en los aspectos temporal, económico y técnico:

### 2.1. Viabilidad temporal

Se busca ejecutar y culminar el proyecto de tesis durante el semestre 2021-2. El cronograma desarrollado en la **sección 8** hace de sustento para esto. Además, se ha conversado con el asesor respecto a la posibilidad de iniciar la ejecución del proyecto durante el periodo entre los semestres 2021-1 y 2021-2 con el fin de asegurar el cumplimiento del proyecto en el plazo establecido.

### 2.2. Viabilidad económica

Las herramientas a ser utilizadas en el proyecto son gratuitas o cuentan con licencia educativa. Por otro lado, se accedió a todos los documentos académicos revisados a la actualidad sin costo alguno, y se espera que el acceso futuro a trabajos tampoco tenga costo gracias a los beneficios brindados por la universidad en diferentes bases de datos académicas. Así, es posible afirmar que se incurrirá en un costo mínimo para la ejecución del proyecto de tesis.

### 2.3. Viabilidad técnica

El tesista cuenta con:

- Conocimiento previo sobre algoritmos metaheurísticos y consultas SQL.
- El apoyo de su asesor, quien es un especialista en el campo de algoritmia.

- Documentación considerable acerca de la optimización de consultas SQL, las características/funcionamiento de algoritmos memético y genético, y la aplicación de este último para realizar dicha optimización.

## 3. Alcance

Como se afirmó en la sección anterior, el proyecto de tesis realizará una implementación de un algoritmo memético adaptado para la optimización de consultas y comparado con una adaptación del algoritmo genético descrita en otro trabajo académico, perteneciendo así al área de ciencias de la computación. La adaptación del algoritmo memético supone: la definición y/o diseño de variables relevantes para el problema, de estructuras de datos que las soporten, y de una función objetivo utilizada para realizar la optimización de consultas; el diseño e implementación del algoritmo memético en sí; y la calibración de parámetros de ejecución de este último. Cabe resaltar en esta etapa dos puntos importantes sobre el algoritmo adaptado: este funcionará para consultas simples (es decir, que cuentan con las operaciones de selección, proyección y *join*, y no incluyen consultas anidadas) y no se encargará de interpretar instrucciones SQL, sino que trabajará directamente con planes de ejecución, o al menos una representación lógica de los mismos.

Por otro lado, la comparación con el algoritmo genético implica: la implementación de este algoritmo y de un módulo de software sencillo que facilite realizar múltiples ejecuciones de ambos algoritmos; la ejecución de estos últimos mediante dicho módulo bajo los mismos datos de entrada y considerándose escenarios con bases de datos distribuidas (una mezcla de bases locales y remotas); la comparación de los resultados de ejecución de ambos algoritmos mediante experimentación numérica; y la redacción de conclusiones en base a los resultados de dicha experimentación.

Finalmente, algo no contemplado en el presente proyecto es la integración del algoritmo desarrollado a un DBMS, pues este está orientado a fines académicos y no comerciales.

## 4. Limitaciones

Considerando la naturaleza del trabajo de tesis, una de las principales limitantes existentes es que los resultados de ejecución de los algoritmos (tanto propio como el implementado a partir de otro trabajo académico) dependen en gran medida de las capacidades técnicas de la computadora a utilizar (en el presente proyecto se utilizará la laptop personal del que escribe). Esto puede, por un lado, dificultar la repetibilidad y comparación de resultados en trabajos futuros, y por otro, producir una desviación o sesgo (ya sea para peor o para mejor) de la aparente calidad de dichos resultados.

## 5. Identificación de los riesgos del proyecto

En la **tabla B1** se detallarán los riesgos identificados para el proyecto de tesis. Se decidió que tanto la posibilidad de ocurrencia (**P**) como el impacto (**I**) de un riesgo se determinen en base a tres categorías: bajo, medio y alto, equivalentes a los niveles numéricos 1, 2 y 3, respectivamente. A su vez, la severidad (**S**), es el producto matemático de la posibilidad y el impacto (**P x I**), y se utiliza para comparar y priorizar los riesgos.

*Tabla B1. Riesgos del proyecto*

| **Descripción** | **Síntomas** | **P** | **I** | **S** | **Mitigación** | **Contingencia** |
|---|---|---|---|---|---|---|
| Falta de tiempo de especialistas para validar resultados esperados | Especialistas ocupados; baja frecuencia de comunicación con estos | 2 | 3 | 6 | Planificación con antelación de la validación con los especialistas | Recurrir a especialistas alternativos para las validaciones que falten |
| Avería de la laptop utilizada en el proyecto | Disminución del rendimiento del equipo, aumento de apagados o reinicios inesperados del mismo | 1 | 3 | 3 | Mantenimiento preventivo periódico al equipo | Uso de una laptop de otro miembro de la familia |
| Borrado accidental de | Desaparición de entregables o de sus | 1 | 3 | 3 | Respaldos periódicos de los | Acceso a los documentos |

| | | | | | | |
|---|---|---|---|---|---|---|
| avances de entregables del proyecto | versiones más actualizadas del repositorio de tesis en Google Drive | | | | entregables en un medio de almacenamiento local | respaldados |
| Falta de tiempo del tesista o asesor para avances del proyecto antes del inicio del siguiente ciclo | Bajo nivel de comunicación entre tesista y asesor, y tardanzas o inasistencias a reuniones durante el periodo indicado. | 2 | 2 | 4 | Planificación clara de las reuniones y actividades a realizar en conjunto con el asesor, y avisos con antelación de posibles impedimentos o retrasos a dichas reuniones/actividades | Reprogramación de las reuniones y actividades a realizar durante el ciclo siguiente |

## 6. EDT

En la **figura B1** se muestra el EDT del proyecto:

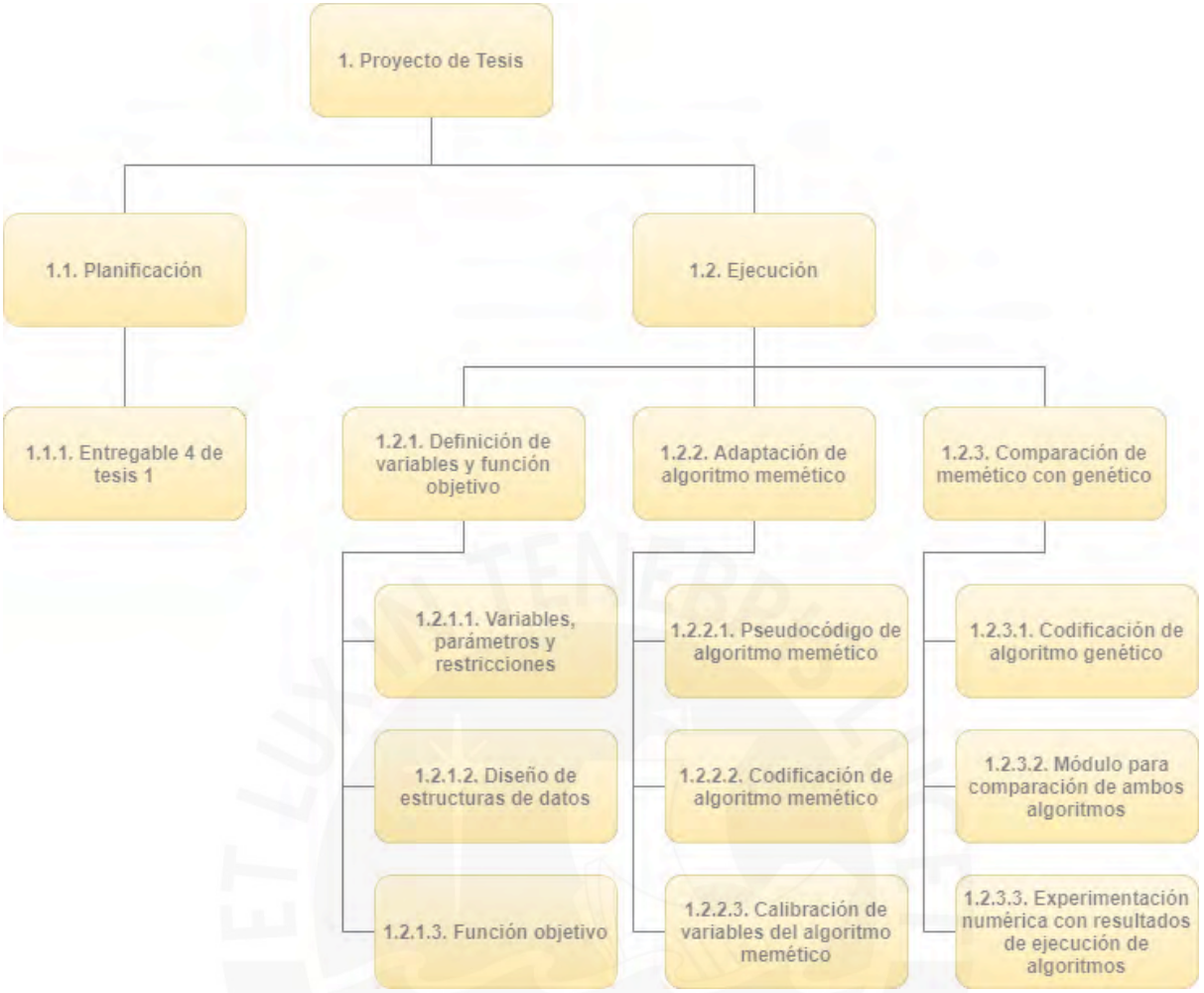



*Figura B1. EDT*

Se procederá ahora a detallar cada componente en la **tabla B2**:

*Tabla B2. Diccionario de EDT*

| **1.1.1** | Entregable 4 de tesis 1 |
|---|---|
| **Descripción** | **Medio de verificación** |
| Entregable final del curso de tesis 1 con todas sus observaciones subsanadas. | Documento con las secciones: problemática; objetivos; métodos y procedimientos; marco conceptual; estado del arte; y referencias. El plan de proyecto es un anexo del documento. |
| **1.2.1.1** | Variables, parámetros y restricciones |
| **Descripción** | **Medio de verificación** |
| Determinación de las variables, parámetros y restricciones a considerar para la optimización de consultas SQL realizada por el algoritmo memético. | Documento conteniendo el listado y descripción de parámetros, variables y restricciones a utilizar en el algoritmo |

| 1.2.1.2 | Diseño de estructuras de datos |
|---|---|
| **Descripción** | **Medio de verificación** |
| Selección y diseño de estructuras de datos para representar las variables y restricciones determinadas en el componente anterior. | Documento conteniendo el listado y descripción de estructuras de datos utilizadas para representar cada variable y restricción utilizada |
| **1.2.1.3** | Función objetivo |
| **Descripción** | **Medio de verificación** |
| Diseño de una función objetivo que el algoritmo utilizará para realizar la optimización y para medir y comparar la calidad de sus soluciones. | Documento conteniendo la fórmula matemática de la función objetivo y la descripción de los diferentes componentes de la misma |
| **1.2.2.1** | Pseudocódigo del algoritmo memético |
| **Descripción** | **Medio de verificación** |
| Pseudocódigo que permite entender a alto nivel el funcionamiento y la lógica detrás de la adaptación del algoritmo memético para la optimización de consultas. | Documento que contiene el pseudocódigo y la explicación del mismo para el algoritmo memético |
| **1.2.2.2** | Codificación de algoritmo memético |
| **Descripción** | **Medio de verificación** |
| Implementación en el lenguaje Java del algoritmo memético adaptado. | Código fuente y ejecutable del algoritmo memético |
| **1.2.2.3** | Calibración de variables del algoritmo memético |
| **Descripción** | **Medio de verificación** |
| Determinación de los valores para los parámetros del algoritmo que ofrecen los mejores resultados. | Informe de calibración de las variables del algoritmo propuesto |
| **1.2.3.1** | Codificación del algoritmo genético |
| **Descripción** | **Medio de verificación** |
| Implementación en el lenguaje Java del algoritmo genético seleccionado para la comparación con el memético. | Código fuente y ejecutable del algoritmo genético |
| **1.2.3.2** | Módulo para comparación de ambos algoritmos |
| **Descripción** | **Medio de verificación** |

| | |
|---|---|
| Implementación en el lenguaje Java de un programa que permite ejecutar cualquiera de los dos algoritmos una cantidad determinada de veces con los mismos datos de entrada y configuración de parámetros. | Código fuente y ejecutable del módulo de comparación |

| **1.2.3.3** | Experimentación numérica con resultados de ejecución de ambos algoritmos |
|---|---|
| **Descripción** | **Medio de verificación** |
| Comparación de los resultados de ejecución de ambos algoritmos mediante experimentación numérica para determinar cuál de los dos ofrece mejores resultados en términos de tiempo de ejecución y calidad de soluciones generadas. | Informe de experimentación numérica sobre la comparación entre el rendimiento de ambos algoritmos |

## 7. Lista de Tareas

*Tabla B3. Lista de tareas*

| Nro. | Tarea | Duración estimada (horas) | Esfuerzo estimado (horas-persona) | Costo estimado (soles) |
|---|---|---|---|---|
| 1 | Corrección de entregable 4 de Tesis 1 | 2 | 3 | 100 |
| 2 | Determinación de variables y restricciones de optimización | 8 | 10 | 240 |
| 3 | Validación de variables/restricciones por especialista en bases de datos | 2 | 4 | 290 |
| 4 | Diseño de estructuras de datos para cada variable y restricción | 6 | 7 | 140 |
| 5 | Validación de estructuras de datos por especialista en algoritmos | 3 | 5 | 390 |
| 6 | Redacción de los resultados esperados avanzados en el documento de tesis para exposición 1 | 2 | 3 | 100 |
| 7 | **Exposición 1** | 3 | 4 | 110 |
| 8 | Levantamiento de observaciones de exposición 1 | 2 | 3 | 100 |
| 9 | Diseño de función objetivo para la optimización | 4 | 5 | 120 |

| | | | | |
|---|---|---|---|---|
| 10 | Validación de función objetivo por especialista en algoritmos | 2 | 4 | 290 |
| 11 | Creación de pseudocódigo para algoritmo memético | 4 | 5 | 120 |
| 12 | Realización de pruebas de flujo para el pseudocódigo creado | 2 | 3 | 100 |
| 13 | Validación del pseudocódigo por especialista en algoritmia metaheurística | 2 | 4 | 290 |
| 14 | Avance de implementación de algoritmo memético (30%) | 6 | 7 | 140 |
| 15 | Redacción de los resultados esperados avanzados en el documento de tesis para exposición 2 | 2 | 3 | 100 |
| 16 | **Exposición 2** | 3 | 4 | 110 |
| 17 | Levantamiento de observaciones de exposición 2 | 2 | 3 | 100 |
| 18 | Avance de implementación de algoritmo memético (60%) | 6 | 7 | 140 |
| 19 | Redacción de los resultados esperados avanzados en el documento de tesis para exposición 3 | 2 | 3 | 100 |
| 20 | **Exposición 3** | 3 | 4 | 110 |
| 21 | Levantamiento de observaciones de exposición 3 | 2 | 3 | 100 |
| 22 | Implementación completa de algoritmo memético | 8 | 10 | 240 |
| 23 | Realización de pruebas unitarias para verificación del resultado esperado (algoritmo memético) | 3 | 4 | 110 |
| 24 | Determinación de posibles configuraciones de parámetros para el algoritmo memético | 2 | 3 | 100 |
| 25 | Ejecución de algoritmo para las diferentes configuraciones de parámetros | 4 | 4 | 40 |
| 26 | Creación de informe de calibración de parámetros | 2 | 3 | 100 |

| | | | | |
|---|---|---|---|---|
| 27 | Validación de infore de calibración por especialista en algoritmia metaheurística | 2 | 4 | 290 |
| 28 | Redacción de los resultados esperados avanzados en el documento de tesis para exposición 4 | 2 | 3 | 100 |
| 29 | **Exposición 4** | 3 | 4 | 110 |
| 30 | Levantamiento de observaciones de exposición 4 | 2 | 3 | 100 |
| 31 | Avance de implementación de algoritmo genético(70%) | 10 | 12 | 260 |
| 32 | Redacción de los resultados esperados avanzados en el documento de tesis para el avance parcial | 3 | 5 | 190 |
| 33 | **Exposición de avance parcial** | 2 | 3 | 100 |
| 34 | Implementación completa de algoritmo genético | 5 | 6 | 130 |
| 35 | Realización de pruebas unitarias para verificación del resultado esperado (algoritmo genético) | 3 | 4 | 110 |
| 36 | Implementación de módulo de comparación de algoritmos | 4 | 4 | 40 |
| 37 | Realización de pruebas unitarias para verificación del resultado esperado (módulo de comparación) | 2 | 2 | 20 |
| 38 | Levantamiento de observaciones de exposición de avance parcial | 4 | 5 | 120 |
| 39 | Realización de múltiples ejecuciones de ambos algoritmos mediante el módulo de comparación | 10 | 10 | 100 |
| 40 | Redacción de los resultados esperados avanzados en el documento de tesis para la exposición 5 | 3 | 4 | 110 |
| 41 | **Exposición 5** | 2 | 3 | 100 |
| 42 | Levantamiento de observaciones de exposición 5 | 2 | 3 | 100 |

| | | | | |
|---|---|---|---|---|
| 43 | Realización de experimentación numérica con resultados obtenidos de las ejecuciones de algoritmos | 3 | 4 | 110 |
| 44 | Redacción de informe de experimentación numérica | 3 | 4 | 110 |
| 45 | Validación de informe de experimentación numérica por especialista en algoritmos | 2 | 4 | 290 |
| 46 | Redacción de los resultados esperados avanzados en el documento de tesis para la exposición 6 | 3 | 4 | 110 |
| 47 | **Exposición 6** | 2 | 3 | 100 |
| 48 | Levantamiento de observaciones de exposición 6 | 2 | 3 | 100 |
| 49 | Redacción y presentación del entregable final de tesis | 8 | 10 | 240 |
| 50 | Levantamiento de observaciones de entregable final | 7 | 8 | 150 |
| 51 | Preparación de exposición final | 8 | 9 | 160 |
| 52 | **Exposición final** | 2 | 2 | 20 |
| | **TOTAL** | **186** | **244** | **7250** |

## 8. Cronograma

*Tabla B4. Cronograma del proyecto*

| Nro. Semana | Nro. Tarea | Tarea |
|---|---|---|
| **1** | 1 | Corrección de entregable 4 de Tesis 1 |
| | 2 | Actualización cronograma de progreso de entregables |
| **2** | 3 | Determinación de variables y restricciones de optimización |
| **3** | 4 | **Exposición 1** |
| | 5 | Diseño de estructuras de datos para cada variable y restricción |
| | 6 | Diseño de función objetivo para la optimización |
| | 7 | Validación de variables/restricciones por especialista en bases de datos |
| **4** | 8 | Levantamiento de observaciones de exposición 1 |

| | | |
|---|---|---|
| | 9 | Validación de estructuras de datos por especialista en algoritmos |
| | 10 | Validación de función objetivo por especialista en algoritmos |
| | 11 | Creación de pseudocódigo para algoritmo memético |
| | 12 | Realización de pruebas de flujo para el pseudocódigo creado |
| | 13 | Validación del pseudocódigo por especialista en algoritmia metaheurística |
| | 14 | Avance de implementación de algoritmo memético: implementación de clases a utilizar en el algoritmo (sin considerar la lógica del algoritmo mismo) y cálculo de función de costo/*fitness* |
| **5** | 15 | Avance de implementación de algoritmo memético: implementación de operadores para la alteración y mejora de individuos de la población |
| | 16 | **Exposición 2** |
| **6** | 17 | Levantamiento de observaciones de exposición 2 |
| | 18 | Implementación completa de algoritmo memético |
| | 19 | Realización de pruebas unitarias para verificación del resultado esperado (algoritmo memético) |
| | 20 | Determinación de posibles configuraciones de parámetros para el algoritmo memético |
| | 21 | Ejecución de algoritmo para las diferentes configuraciones de parámetros |
| **7** | 22 | **Exposición de avance parcial** |
| | 23 | Creación de informe de calibración de parámetros |
| | 24 | Validación de informe de calibración por especialista en algoritmia metaheurística |
| | 25 | Avance de implementación de algoritmo genético: implementación de clases a usar en algoritmo y cálculo de función de costo/*fitness* |
| **8** | 26 | Levantamiento de observaciones de exposición de avance parcial |
| | 27 | Implementación completa de algoritmo genético |
| | 28 | Realización de pruebas unitarias para verificación del resultado esperado (algoritmo genético) |
| | 29 | Implementación de módulo de comparación de algoritmos |
| | 30 | Realización de pruebas unitarias para verificación del resultado esperado (módulo de comparación) |
| **9** | **EXÁMENES PARCIALES** | |
| **10** | 31 | **Exposición 3** |

| | | |
|---|---|---|
| | 32 | Realización de múltiples ejecuciones de ambos algoritmos mediante el módulo de comparación |
| | 33 | Realización de experimentación numérica con resultados obtenidos de las ejecuciones de algoritmos |
| **11** | 34 | Validación de informe de experimentación numérica por especialista en algoritmos |
| | 35 | Levantamiento de observaciones de exposición 3 |
| **12, 13** | **REVISIÓN DEL JURADO** | |
| **14, 15** | 36 | Levantamiento de observaciones de entregable final |
| | 37 | Preparación de exposición final |
| **16, 17** | 38 | **Exposición final** |

## 9. Lista de recursos

### 9.1. Personas involucradas

- **Tesista:** Alumno que elabora el proyecto de tesis para recibir un título universitario.
- **Asesor:** Especialista responsable de guiar al tesista a lo largo de la ejecución del proyecto de tesis.
- **Especialista en bases de datos:** Brinda su apoyo en la validación de resultados relacionados a las bases de datos (como la selección de variables y restricciones para la optimización de consultas SQL).
- **Especialista en algoritmia metaheurística:** Brinda su apoyo en la validación de resultados relacionados al diseño, adaptación y desarrollo de algoritmos metaheurísticos.
- **Especialista en algoritmos:** Brinda su apoyo en la validación de resultados relacionados a aspectos generales de algoritmos (como el diseño de estructuras de datos y de la función objetivo a utilizar en el algoritmo memético). Puede ser la misma persona que el especialista en algoritmia metaheurística.

## 9.2. Materiales requeridos

No aplica.

## 9.3. Estándares utilizados

- **Metodología Kanban en desarrollo de software:** Permitirá organizar y hacer un seguimiento granular del avance en el desarrollo de los módulos de software del proyecto de tesis.

## 9.4. Equipamiento requerido

- **Laptop:** Computadora portátil que el tesista utilizará para todas las tareas relacionadas al proyecto de tesis (incluida la ejecución de los algoritmos).

## 9.5. Herramientas requeridas

- **IDE para el lenguaje Java:** Importante para facilitar el desarrollo y pruebas de los módulos de software del proyecto. Se planea utilizar el IDE **IntelliJ IDEA Community Edition**.
- **IDE para el lenguaje R:** Importante para facilitar la realización de experimentación numérica mediante las herramientas que provee el lenguaje R y sus librerías. Se planea utilizar el IDE **RStudio**.
- **Herramienta digital para realizar un tablero kanban**: Se decidió que el tablero Kanban se cree y actualice en un medio digital en lugar de uno físico. Se planea utilizar la plataforma **Trello** para este propósito.
- **Herramientas de comunicación instantánea:** Vital para mantener una comunicación fluida con el asesor y los especialistas. Se planea utilizar las aplicaciones **WhatsApp** y **Zoom**.
- **Herramientas de ofimática:** Vitales para redactar los documentos necesarios en el proyecto. Se utiliza y planea utilizar la plataforma de **Google Docs**.

## 10. Costeo del proyecto

En la **Tabla B5**, se mostrará una descomposición del costeo del proyecto de tesis de acuerdo con sus diferentes elementos:

*Tabla B5. Descomposición de costeo del proyecto*

| Ítem | Descipción | Unidad de medida | Cantidad | Valor unitario (soles) | Monto (soles) |
|---|---|---|---|---|---|
| **0** | **Costo total del proyecto** | -- | -- | -- | **7 998.95** |
| **1** | **Personas involucradas** | -- | -- | -- | **7 250** |
| 1.1 | Tesista | Hora | 179 | 10 | 1 790 |
| 1.2 | Asesor | Hora | 52 | 80 | 4 160 |
| 1.3 | Especialista en bases de datos | Hora | 2 | 100 | 200 |
| 1.4 | Especialista en algoritmia metaheurística | Hora | 4 | 100 | 400 |
| 1.5 | Especialista en algoritmos | Hora | 7 | 100 | 700 |
| **2** | **Equipamiento** | -- | -- | -- | **348.95** |
| 2.1 | Laptop (operación) | Hora | 179 | 0.05 | 8.95 |
| 2.2 | Laptop (mantenimiento) | Semana | 17 | 20 | 340 |
| **3** | **Herramientas requeridas** | -- | -- | -- | **0** |
| 3.1 | Licencias de software | Licencia | 0 | 0 | 0 |
| **4** | **Servicios** | -- | -- | -- | **400** |
| 4.1 | Internet | Mes | 4 | 100 | 400 |

## Anexo C: Pruebas de comportamiento de la función objetivo

Enlace del documento:

https://docs.google.com/spreadsheets/d/19cm2TdM6Tqqa2rgvFsQp6wwMS_mu64Y5i0hKkUZcyyo/edit?usp=sharing

Introducción:

**INTRODUCCCIÓN**

En el presente documento se realizarán pruebas de comportamiento de la función objetivo construida, la cual corresponde al Resultado Esperado **R1.3** en el trabajo de tesis. Su redacción y explicación se encuentran en el capítulo 5 del documento de tesis principal.

Estas pruebas se realizarán sobre una configuración del problema pequeña, lo cual permite analizar sin mucha complejidad el comportamiento de la función objetivo . Entre los datos más importantes, tenemos:

-Consulta que usa 5 tablas en la base de datos(T1, T2, T3, T4, T5)
-3 sitios en la base de datos (S1, S2, S3)
-Un límite de iteraciones de 3.
-Los arreglos manejados para la estructura del cromosoma inician con índice 0 en su primer elemento.

La especificación completa de datos es la siguiente:

Datos del problema:

| | |
|---|---|
| **Número de tablas (*NumTab*)** | 5 |
| **Número de sitios (*NumSit*)** | 3 |
| **Tamaño prom. de columna (*TamProm*)** | 5 bytes |

| **Tamaño de tablas** | *NumFil* | *NumCol* | *NumByte* |
|---|---|---|---|
| T1 | 1000 | 6 | 24000 |
| T2 | 300 | 5 | 6000 |
| T3 | 1800 | 4 | 36000 |
| T4 | 50 | 2 | 400 |
| T5 | 700 | 5 | 28000 |

| **Dist. tablas en sitios** | S1 | S2 | S3 |
|---|---|---|---|
| T1 | 1 | 1 | 1 |
| T2 | 0 | 1 | 1 |
| T3 | 1 | 0 | 0 |
| T4 | 0 | 1 | 1 |
| T5 | 1 | 1 | 0 |

| **Cardinalidad de tablas** | T1 | T2 | T3 | T4 | T5 |
|---|---|---|---|---|---|
| C1 | 1000 | 0 | 0 | 0 | 0 |
| C2 | 800 | 0 | 1800 | 0 | 0 |
| C3 | 300 | 300 | 210 | 0 | 0 |
| C4 | 50 | 40 | 30 | 50 | 20 |
| C5 | 380 | 0 | 0 | 0 | 0 |
| C6 | 530 | 0 | 0 | 0 | 700 |
| C7 | 0 | 120 | 500 | 0 | 200 |
| C8 | 0 | 90 | 0 | 0 | 0 |
| C9 | 0 | 200 | 0 | 0 | 0 |
| C10 | 0 | 0 | 0 | 50 | 0 |
| C11 | 0 | 0 | 0 | 0 | 350 |
| C12 | 0 | 0 | 0 | 0 | 500 |

| **Cap. trans. de sitios** | S1 | S2 | S3 |
|---|---|---|---|
| S1 | 0 | 500 | 1000 |
| S2 | 500 | 0 | 200 |
| S3 | 1000 | 200 | 0 |

Cromosomas a comparar:

| Cromosomas a comparar | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Un cromosoma es un arreglo de tamaño *NumTab*. Cada elemento del arreglo es un número entero representado como (ID de Tabla)*100 + (ID de sitio) | | | | | | | | | |
| Cromosoma | Elem. 1 | Elem. 2 | Elem. 3 | Elem. 4 | Elem. 5 | | | | |
| Base | 101 | 203 | 301 | 403 | 502 | | | | |
| Cambio A | **103** | 203 | 301 | 403 | 502 | | | | |
| Cambio B | **203** | **101** | 301 | 403 | 502 | | | | |
| Cambio C | **403** | 203 | 301 | **101** | 502 | | | | |
| *coefCom* | 0,5 | => | Ccom | **0,5** | Cproc | **0,5** | | Overhead trans. (seg) | 0,00002 |

Cálculo de F.O. para crom. base:

| Cálculo de func. objetivo crom. base | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Individuo | Cost. Proc. Join 1 | Cost. Proc. Join 2 | Cost. Proc. Join 3 | Cost. Proc. Join 4 | Cost. Proc. | Cant. trans. | Cost. Com. | Cost. Tot. | **Valor func. Objetivo** |
| Base | 20 | 0,00005 | 300 | 0,012 | 320,01205 | 4 | 126,00008 | 0,53078331 | **1,884007964** |

| Información de joins intermedios | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | Cardinalidades | | | | | | | | | | |
| # Join | NumFil | NumCol | NumBytes | C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 | C11 | C12 |
| Join 1 | 300 | 9 | 13500 | 300 | 300 | 300 | 40 | 300 | 300 | 120 | 90 | 200 | 0 | 0 | 0 |
| Join 2 | 300 | 9 | 13500 | 300 | 300 | 210 | 30 | 300 | 300 | 120 | 90 | 200 | 0 | 0 | 0 |
| Join 3 | 300 | 10 | 15000 | 300 | 300 | 210 | 30 | 300 | 300 | 120 | 90 | 200 | 50 | 0 | 0 |
| Join 4 | 300 | 12 | 18000 | 300 | 300 | 210 | 20 | 300 | 300 | 120 | 90 | 200 | 50 | 300 | 300 |

Cálculo de F.O para cambio A:

**Cambio A:** Alterar el sitio de procedencia de una de las tablas

| Cromosoma | Elem. 1 | Elem. 2 | Elem. 3 | Elem. 4 | Elem. 5 |
|---|---|---|---|---|---|
| Base | 101 | 203 | 301 | 403 | 502 |
| Cambio A | **103** | 203 | 301 | 403 | 502 |

El orden de los joins no se altera, así que los costos de procesamiento serán los mismos. Por otro lado, el cambio de sitio en este caso reduce el número de transmisiones necesarias, por lo que los costos de comunicación deberían reducirse.

**Cálculo de func. objetivo crom. cambio A**

| Individuo | Cost. Proc. Join 1 | Cost. Proc. Join 2 | Cost. Proc. Join 3 | Cost. Proc. Join 4 | Cost. Proc. | Cant. trans. | Cost. Com. | Cost. Tot. | Valor func. Objetivo | |
|---|---|---|---|---|---|---|---|---|---|---|
| Base | 20 | 0,00005 | 300 | 0,012 | 320,01205 | 4 | 126,00008 | ),530783318 | **1,884007964** | |
| Cambio A | 20 | 0,00005 | 300 | 0,012 | 320,01205 | **3** | **102,00006** | 0,52031041 | **1,921929636** | **<-Se redujo el costo de comunicación** |

**Información de joins intermedios**

| | | | | Cardinalidades | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| # Join | NumFil | NumCol | NumBytes | C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 | C11 | C12 |
| Join 1 | 300 | 9 | 13500 | 300 | 300 | 300 | 40 | 300 | 300 | 120 | 90 | 200 | 0 | 0 | 0 |
| Join 2 | 300 | 9 | 13500 | 300 | 300 | 210 | 30 | 300 | 300 | 120 | 90 | 200 | 0 | 0 | 0 |
| Join 3 | 300 | 10 | 15000 | 300 | 300 | 210 | 30 | 300 | 300 | 120 | 90 | 200 | 50 | 0 | 0 |
| Join 4 | 300 | 12 | 18000 | 300 | 300 | 210 | 20 | 300 | 300 | 120 | 90 | 200 | 50 | 300 | 300 |

Cálculo de F.O. para cambio B:

**Cambio B:** Intercambiar la posición de dos tablas contiguas en el arreglo

| Cromosoma | Elem. 1 | Elem. 2 | Elem. 3 | Elem. 4 | Elem. 5 |
|---|---|---|---|---|---|
| Base | 101 | 203 | 301 | 403 | 502 |
| Cambio B | **203** | **101** | 301 | 403 | 502 |

Si bien existe un cambio en el orden de tablas, como las tablas intercambiadas son contiguas y se encuentran al inicio del arreglo, el orden de joins no se ve afectado tampoco, pero el número de transmisiones se reduce en 1. Por ello, el costo de comunicación nuevo será menor al base.

**Cálculo de func. objetivo crom. cambio B**

| Individuo | Cost. Proc. Join 1 | Cost. Proc. Join 2 | Cost. Proc. Join 3 | Cost. Proc. Join 4 | Cost. Proc. | Cant. trans. | Cost. Com. | Cost. Tot. | Valor func. Objetivo | |
|---|---|---|---|---|---|---|---|---|---|---|
| Base | 20 | 0,00005 | 300 | 0,012 | 320,01205 | 4 | 126,00008 | ,530783318 | **1,884007964** | |
| Cambio B | 20 | 0,00005 | 300 | 0,012 | 320,01205 | **3** | **94,50006** | 0,51653027 | **1,935994935** | **<-Se redujo el costo de comunicación** |

**Información de joins intermedios**

| | | | | Cardinalidades | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| # Join | NumFil | NumCol | NumBytes | C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 | C11 | C12 |
| Join 1 | 300 | 9 | 13500 | 300 | 300 | 300 | 40 | 300 | 300 | 120 | 90 | 200 | 0 | 0 | 0 |
| Join 2 | 300 | 9 | 13500 | 300 | 300 | 210 | 30 | 300 | 300 | 120 | 90 | 200 | 0 | 0 | 0 |
| Join 3 | 300 | 10 | 15000 | 300 | 300 | 210 | 30 | 300 | 300 | 120 | 90 | 200 | 50 | 0 | 0 |
| Join 4 | 300 | 12 | 18000 | 300 | 300 | 210 | 20 | 300 | 300 | 120 | 90 | 200 | 50 | 300 | 300 |

Cambio de F.O. para cambio C:

**Cambio C:** Intercambiar la posición de dos tablas no contiguas en el arreglo

| Cromosoma | Elem. 1 | Elem. 2 | Elem. 3 | Elem. 4 | Elem. 5 |
|---|---|---|---|---|---|
| Base | 101 | 203 | 301 | 403 | 502 |
| Cambio C | **403** | 203 | 301 | **101** | 502 |

El cambio realizado, además de reducir el número de transmisiones en 2 (reduciéndo así el costo de comunicación), también altera el orden en que se aplican joins sobre las tablas, por lo que los costos de procesamiento cambiarán respecto a los del caso base.

**Cálculo de func. objetivo crom. cambio C**

| Individuo | Cost. Proc. Join 1 | Cost. Proc. Join 2 | Cost. Proc. Join 3 | Cost. Proc. Join 4 | Cost. Proc. | Cant. trans. | Cost. Com. | Cost. Tot. | Valor func. Objetivo | |
|---|---|---|---|---|---|---|---|---|---|---|
| Base | 20 | 0,00005 | 300 | 0,012 | 320,01205 | 4 | 126,00008 | ,530783318 | **1,884007964** | |
| Cambio C | 300 | 0,09 | 0,025 | 0,05 | **300,165** | **2** | **39,00004** | 0,46982693 | **2,128443308** | **<-Se redujeron costos de procesamiento y comunicación** |

**Información de joins intermedios**

| | | | | Cardinalidades | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| # Join | NumFil | NumCol | NumBytes | C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | C10 | C11 | C12 |
| Join 1 | 300 | 6 | 9000 | 0 | 0 | 300 | 40 | 0 | 0 | 120 | 90 | 200 | 50 | 0 | 0 |
| Join 2 | 300 | 7 | 10500 | 0 | 300 | 210 | 30 | 0 | 0 | 120 | 90 | 200 | 50 | 0 | 0 |
| Join 3 | 300 | 10 | 15000 | 300 | 300 | 210 | 30 | 300 | 300 | 120 | 90 | 200 | 50 | 0 | 0 |
| Join 4 | 300 | 12 | 18000 | 300 | 300 | 210 | 20 | 300 | 300 | 120 | 90 | 200 | 50 | 300 | 300 |

Resultados y conclusiones:

**Cromosoma base y cambios**

| Cromosoma | Elem. 1 | Elem. 2 | Elem. 3 | Elem. 4 | Elem. 5 | Cost. Proc. Join 1 | Cost. Proc. Join 2 | Cost. Proc. Join 3 | Cost. Proc. Join 4 | Cost. Proc | Cant. tran | Cost. Com | Cost. Tot. | Valor func. Objetivo |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Base | 101 | 203 | 301 | 403 | 502 | 20 | 0,00005 | 300 | 0,012 | 320,0120 | 4 | 126,0000 | 0,530783 | **1,884007964** |
| Cambio A | **103** | 203 | 301 | 403 | 502 | 20 | 0,00005 | 300 | 0,012 | 320,0120 | **3** | **102,0000** | 0,520310 | **1,921929636** |
| Cambio B | **203** | **101** | 301 | 403 | 502 | 20 | 0,00005 | 300 | 0,012 | 320,0120 | **3** | **94,50006** | 0,516530 | **1,935994935** |
| Cambio C | **403** | 203 | 301 | **101** | 502 | 300 | 0,09 | 0,025 | 0,05 | **300,165** | **2** | **39,00004** | 0,469826 | **2,128443308** |

**Conclusiones**

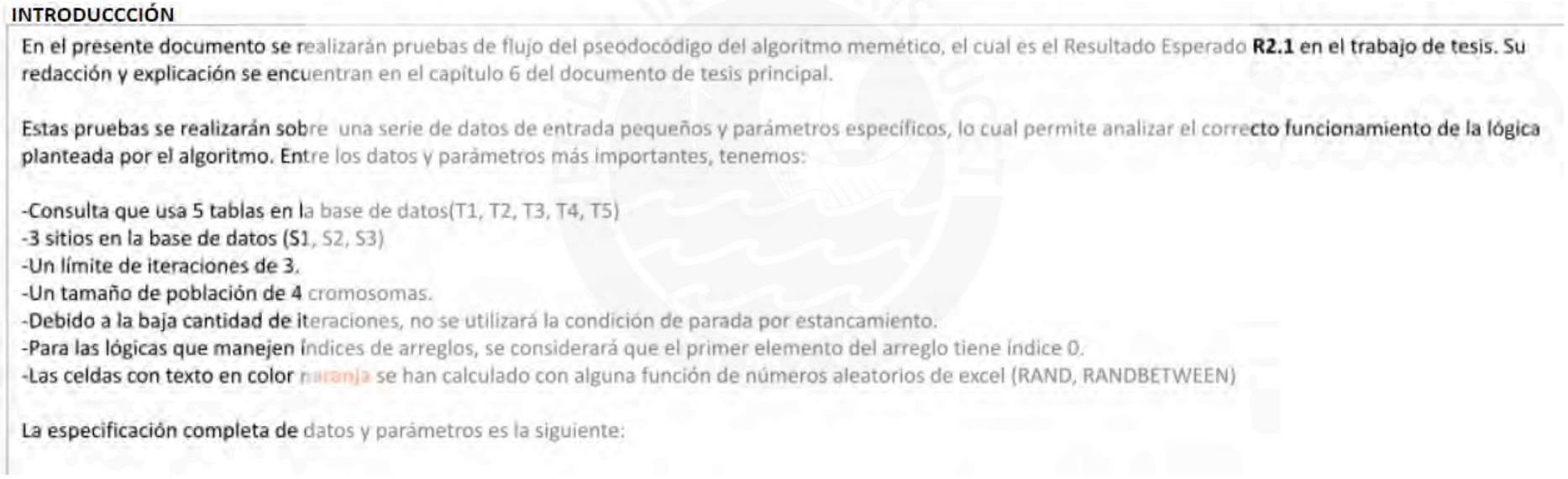
Se observa que la función objetivo cambia su valor de acuerdo a la solución presentada en cada cromosoma. De manera general, una reducción en costos de procesamiento y/o de comunicación implica un aumento en el valor de la función objetivo. No obstante, mientras que los cambios en los costos de comunicación son relativamente predecibles (pues se relacionan directamente al número de transmisiones, algo que es posible determinar simplemente observando el contenido del cromosoma), los cambios en costos de procesamiento son más difíciles de predecir, pues dependen de numerosos factores (columnas comunes, múltiples cardinalidades, características de los joins parciales formados, entre otros). Es por esto que la función objetivo planteada ofrece una herramienta viable (sobre todo al ser calculada mediante cómputo) para cuantificar la calidad de una determinada solución.

## Anexo D: Pruebas de flujo de pseudocódigo para el algoritmo memético

Enlace del documento:

https://docs.google.com/spreadsheets/d/1aLa13G7rLqqxFh3wDINpYLuUrvZjZNQx/edit?usp=sharing&ouid=117223103864494113454&rtpof=true&sd=true

Introducción:



**INTRODUCCCIÓN**

En el presente documento se realizarán pruebas de flujo del pseodocódigo del algoritmo memético, el cual es el Resultado Esperado **R2.1** en el trabajo de tesis. Su redacción y explicación se encuentran en el capítulo 6 del documento de tesis principal.

Estas pruebas se realizarán sobre una serie de datos de entrada pequeños y parámetros específicos, lo cual permite analizar el correcto funcionamiento de la lógica planteada por el algoritmo. Entre los datos y parámetros más importantes, tenemos:

-Consulta que usa 5 tablas en la base de datos(T1, T2, T3, T4, T5)
-3 sitios en la base de datos (S1, S2, S3)
-Un límite de iteraciones de 3.
-Un tamaño de población de 4 cromosomas.
-Debido a la baja cantidad de iteraciones, no se utilizará la condición de parada por estancamiento.
-Para las lógicas que manejen índices de arreglos, se considerará que el primer elemento del arreglo tiene índice 0.
-Las celdas con texto en color naranja se han calculado con alguna función de números aleatorios de excel (RAND, RANDBETWEEN)

La especificación completa de datos y parámetros es la siguiente:

Datos del problema y parámetros de ejecución:

| | |
|---|---|
| **Número de tablas (*NumTab*)** | 5 |
| **Número de sitios (*NumSit*)** | 3 |
| **Tamaño prom. de columna (*TamProm*)** | 5 bytes |

| **Tamaño de tablas** | *NumFil* | *NumCol* | *NumByte* |
|---|---|---|---|
| T1 | 1000 | 6 | 24000 |
| T2 | 300 | 5 | 6000 |
| T3 | 1800 | 4 | 36000 |
| T4 | 50 | 2 | 400 |
| T5 | 700 | 5 | 28000 |

| **Dist. tablas en sitios** | S1 | S2 | S3 |
|---|---|---|---|
| T1 | 1 | 1 | 1 |
| T2 | 0 | 1 | 1 |
| T3 | 1 | 0 | 0 |
| T4 | 0 | 1 | 1 |
| T5 | 1 | 1 | 0 |

| **Cardinalidad de tablas** | T1 | T2 | T3 | T4 | T5 |
|---|---|---|---|---|---|
| C1 | 1000 | 0 | 0 | 0 | 0 |
| C2 | 800 | 0 | 1800 | 0 | 0 |
| C3 | 300 | 300 | 210 | 0 | 0 |
| C4 | 50 | 40 | 30 | 50 | 20 |
| C5 | 380 | 0 | 0 | 0 | 0 |
| C6 | 530 | 0 | 0 | 0 | 700 |
| C7 | 0 | 120 | 500 | 0 | 200 |
| C8 | 0 | 90 | 0 | 0 | 0 |
| C9 | 0 | 200 | 0 | 0 | 0 |
| C10 | 0 | 0 | 0 | 50 | 0 |
| C11 | 0 | 0 | 0 | 0 | 350 |
| C12 | 0 | 0 | 0 | 0 | 500 |

| **Cap. trans. de sitios** | S1 | S2 | S3 |
|---|---|---|---|
| S1 | 0 | 500 | 1000 |
| S2 | 500 | 0 | 200 |
| S3 | 1000 | 200 | 0 |

| | |
|---|---|
| Número de iteraciones | 3 |
| Tamaño de población | 4 |

Inicialización de población:

| Población inicial generada | | Cada individuo es un arreglo de tamaño *NumTab*. Cada elemento del arreglo es un número entero representado como (ID de Tabla)*100 + (ID de sitio) | | | |
|---|---|---|---|---|---|
| Individuo | Elem. 1 | Elem. 2 | Elem. 3 | Elem. 4 | Elem. 5 |
| Individuo 1 | 101 | 203 | 301 | 403 | 502 |
| Individuo 2 | 202 | 501 | 101 | 403 | 301 |
| Individuo 3 | 501 | 301 | 103 | 403 | 202 |
| Individuo 4 | 301 | 403 | 502 | 202 | 102 |

| *coefCom* | 0,5 | => | Ccom | 0,5 | Cproc | 0,5 | Overhead trans. (seg) | 0,00002 |
|---|---|---|---|---|---|---|---|---|

OBS

Se está considerando que el resultado de un join entre dos tablas (T1 y T2) presenta:

-*NumFil* se calcula con la fórmula descrita en el R1.1
-*NumCol = NumColT1* + (# col que solo T2 tiene)
-*NumBytes = NumFil * NumCol* * TamPromCol
-Las cardinalidades de cada columna:
Son iguales a max(cardT1,cardT2) si no es una columna compartida
Son iguales a min(cardT1,cardT2) si es una columna compartida
Si sobrepasan NumFil, se les sustituye por NumFil

| Evaluación de calidad de la población | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Individuo | Cost. Proc. Join 1 | Cost. Proc. Join 2 | Cost. Proc. Join 3 | Cost. Proc. Join 4 | Cost. Proc. | Cost. Com. | Cost. Tot. | Valor func. Objetivo |
| Individuo 1 | 20 | 0,00005 | 300 | 0,012 | 320,01205 | 126,00008 | 0,530783 | **1,884007964** |
| Individuo 2 | 1,875 | 0,02857142857 | 300 | 0,00006666666667 | 301,903638 | 46,50006 | 0,478707 | **2,088959226** |
| Individuo 3 | 84 | 0,00003703703704 | 700 | 0,00625 | 784,006287 | 199,50004 | 0,598325 | **1,671331608** |
| Individuo 4 | 1800 | 84 | 0,0875 | 0,00001587301587 | 1884,08751 | 261,00004 | 0,655503 | **1,525544381** |

| Mejor individuo | |
|---|---|
| Individuo | Val. Func. Objetivo |
| Individuo 2 | 2,088959226 |

Iteración 1, población inicial y selección de padres:

| Población actual | | | | | | |
|---|---|---|---|---|---|---|
| Individuo | Elem. 1 | Elem. 2 | Elem. 3 | Elem. 4 | Elem. 5 | Valor func. Obj. |
| Individuo 1 | 101 | 203 | 301 | 403 | 502 | 1,884007964 |
| Individuo 2 | 202 | 501 | 101 | 403 | 301 | 2,088959226 |
| Individuo 3 | 501 | 301 | 103 | 403 | 202 | 1,671331608 |
| Individuo 4 | 301 | 403 | 502 | 202 | 102 | 1,525544381 |

| Fuentes de aleatoriedad | |
|---|---|
| Probabilidad | 0,230080054 |
| Valor min. | 0 |
| Valor max. | 4 |
| Entero | 1 |

| Mejor individuo actual | |
|---|---|
| Individuo | Val. Func. Objetivo |
| Individuo 2 | 2,088959226 |

v solo se casará a 2 individuos

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *pMut* | 0,7 | *pBusq* | 0,5 | % población a casar | 0,5 | # vecinos evaluados búsqueda | 2 |

| Casamiento de población | | | | | | |
|---|---|---|---|---|---|---|
| Individuos a casar | 2 | Padre 1 | 3 | Padre 2 | 1 | |
| Individuo | Elem. 1 | Elem. 2 | Elem. 3 | Elem. 4 | Elem. 5 | Valor func. Obj. |
| Individuo 3 | 501 | 301 | 103 | 403 | 202 | 1,671331608 |
| Individuo 1 | 101 | 203 | 301 | 403 | 502 | 1,884007964 |

Iteración 1, aplicación de *Partially Mapped Crossover* sobre padres y mutación por intercambio del hijo:

| ***Partially mapped crossover*** | | | | | |
|---|---|---|---|---|---|
| Índice inicio | 0 | Índice fin | 2 | | |
| **Los elementos del primer padre que pertenecen al segmento marcado por ambos índices se copian en el cromosoma hijo** | | | | | |
| Individuo | Elem. 1 | Elem. 2 | Elem. 3 | Elem. 4 | Elem. 5 |
| Individuo 3 | **501** | **301** | **103** | 403 | 202 |
| Individuo 1 | 101 | 203 | 301 | 403 | 502 |
| **Hijo** | **501** | **301** | **103** | | |
| **Se evalúa qué elementos del padre 2 en el segmento no están en el segmento para el padre 1** | | | | | |
| Individuo | Elem. 1 | Elem. 2 | Elem. 3 | Elem. 4 | Elem. 5 |
| Individuo 3 | 501 | 301 | 103 | 403 | 202 |
| Individuo 1 | 101 | 203 | 301 | 403 | 502 |
| **Hijo** | 501 | 301 | 103 | | |
| **Por cada elemento faltante, se analiza el elemento del padre 1 que está en la misma posición que el faltante. Se descubre la posición de dicho elemento en el padre 2. En dicha posición del hijo se colocará el faltante. En caso dicha posición del hijo ya esté ocupada, el proceso se repite para dicha posición** | | | | | |
| Individuo | Elem. 1 | Elem. 2 | Elem. 3 | Elem. 4 | Elem. 5 |
| Individuo 3 | 501 | 301 | 103 | 403 | 202 |
| | | | | | |
| Individuo 1 | 101 | **203** | 301 | 403 | 502 |
| **Hijo** | 501 | 301 | 103 | | **203** |
| **El resto de elementos se copian desde el padre 2** | | | | | |
| Individuo | Elem. 1 | Elem. 2 | Elem. 3 | Elem. 4 | Elem. 5 |
| Individuo 3 | 501 | 301 | 103 | 403 | 202 |
| Individuo 1 | 101 | **203** | 301 | **403** | 502 |
| **Hijo** | 501 | 301 | 103 | **403** | 203 |

| **Swap Mutation** | | | | | |
|---|---|---|---|---|---|
| Valor aleatorio | 0,474547132 | Mutación? | **SI** | | |
| Índice 1 | 1 | Índice 2 | 4 | | |
| Individuo | Elem. 1 | Elem. 2 | Elem. 3 | Elem. 4 | Elem. 5 |
| Hijo actual | 501 | **301** | 103 | 403 | **203** |
| **Hijo mutado** | 501 | **203** | 103 | 403 | **301** |

Iteración 1, operador de búsqueda local:

| **Búsqueda local** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Valor aleatorio | 0,357874767 | Búsqueda? | SI | | | | | |
| Indice elemento | 3 | Nuevo sitio | 2 | <- se valida que coherencia con dist. de tablas en sitios | | | | |
| Individuo | Elem. 1 | Elem. 2 | Elem. 3 | Elem. 4 | Elem. 5 | | | |
| Hijo actual | 501 | 203 | 103 | 403 | 301 | | | |
| Vecino 1 | 501 | 203 | 103 | **402** | 301 | | | |
| **Se compara** la calidad de ambos cromosomas y se mantiene el mejor | | | | | | | | |
| Individuo | Cost. Proc. Join 1 | Cost. Proc. Join 2 | Cost. Proc. Join 3 | Cost. Proc. Join 4 | Cost. Proc. | Cost. Com. | Cost. Tot. | **Valor Func.** |
| Hijo actual | 26,25 | 0,028571428 | 300 | 0,000066666 | 326,2786381 | 46,00004 | 0,482048 | **2,07448213** |
| Vecino 1 | 26,25 | 0,028571428 | 300 | 0,000066666 | 326,2786381 | 200,50006 | 0,554830 | **1,80235361** |
| **Ganador 1** | **Hijo actual** | | | | | | | |
| Indice elemento | 1 | Nuevo sitio | 2 | | | | | |
| Individuo | Elem. 1 | Elem. 2 | Elem. 3 | Elem. 4 | Elem. 5 | | | |
| Hijo Actual | 501 | 203 | 103 | 403 | 301 | | | |
| Vecino 2 | 501 | **202** | 103 | 403 | 301 | | | |
| **Se compara** la calidad de ambos cromosomas y se mantiene el mejor | | | | | | | | |
| Individuo | Cost. Proc. Join 1 | Cost. Proc. Join 2 | Cost. Proc. Join 3 | Cost. Proc. Join 4 | Cost. Proc. | Cost. Com. | Cost. Tot. | **Valor Func. Objetivo** |
| Hijo Actual | 26,25 | 0,028571428 | 300 | 0,000066666 | 326,2786381 | 46,00004 | 0,482048 | **2,07448213** |
| Vecino 2 | 26,25 | 0,028571428 | 300 | 0,000066666 | 326,2786381 | 134,00006 | 0,534804 | **1,86984265** |
| **Ganador 2** | **Hijo Actual** | | | | | | | |
| **Salida búsqueda** | Elem. 1 | Elem. 2 | Elem. 3 | Elem. 4 | Elem. 5 | | | |
| **Hijo Actual** | 501 | 203 | 103 | 403 | 301 | | | |

Iteración 1, evaluación de calidad de crom. hijo y evolución de la población:

| Evaluación de calidad de hijos | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Individuo | Cost. Proc. | Cost. Proc. | Cost. Proc. | Cost. Proc. | Cost. Proc. | Cost. Com. | Cost. | Valor Func. |
| Hijo modificado | 26,25 | 0,028571428 | 300 | 0,000066666 | 326,2786381 | 46,00004 | 0,4820480 | 2,07448213 |

| Evolución de la población | | <- se selecciona un individuo de la población para ser reemplazado por el hijo. Si el individuo elegido es el de mayor calidad actualmente, se vuelve a elegir otro individuo aleatoriamente hasta que ya no se trate del mejor | | | |
|---|---|---|---|---|---|
| Índice de población | 2 | Es el mejor individuo? | NO | Reemplazo? | SI |

| Nueva población | | | | | | |
|---|---|---|---|---|---|---|
| Individuo | Elem. 1 | Elem. 2 | Elem. 3 | Elem. 4 | Elem. 5 | Valor func. Obj. |
| Individuo 1 | 101 | 203 | 301 | 403 | 502 | 1,884007964 |
| Individuo 2 | 202 | 501 | 101 | 403 | 301 | 2,088959226 |
| Individuo 3 | 501 | 203 | 103 | 403 | 301 | 2,074482131 |
| Individuo 4 | 301 | 403 | 502 | 202 | 102 | 1,525544381 |

| Nuevo mejor individuo | |
|---|---|
| Individuo | Val. Func. Objetivo |
| Individuo 2 | 2,088959226 |

Iteración 2, población inicial y selección de padres:

| Población actual | | | | | | |
|---|---|---|---|---|---|---|
| Individuo | Elem. 1 | Elem. 2 | Elem. 3 | Elem. 4 | Elem. 5 | Valor func. Obj. |
| Individuo 1 | 101 | 203 | 301 | 403 | 502 | 1,884007964 |
| Individuo 2 | 202 | 501 | 101 | 403 | 301 | 2,088959226 |
| Individuo 3 | 501 | 203 | 103 | 403 | 301 | 2,074482131 |
| Individuo 4 | 301 | 403 | 502 | 202 | 102 | 1,525544381 |

| Fuentes de aleatoriedad | |
|---|---|
| Probabilidad | 0,420513066 |
| Valor min. | 0 |
| Valor max. | 3 |
| Entero | 2 |

| Mejor individuo actual | |
|---|---|
| Individuo | Val. Func. Objetivo |
| Individuo 2 | 2,088959226 |

y solo se casará a 2 individuos

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *pMut* | 0,7 | *pBusq* | 0,5 | % población a casar | 0,5 | # vecinos evaluados búsqueda | 2 |

| Casamiento de población | | | | | | |
|---|---|---|---|---|---|---|
| Individuos a casar | 2 | Padre 1 | 4 | Padre 2 | 2 | |
| Individuo | Elem. 1 | Elem. 2 | Elem. 3 | Elem. 4 | Elem. 5 | Valor func. Obj. |
| Individuo 4 | 301 | 403 | 502 | 202 | 102 | 1,525544381 |
| Individuo 2 | 202 | 501 | 101 | 403 | 301 | 2,088959226 |

Iteración 2, aplicación de *Partially Mapped Crossover* sobre padres y mutación por intercambio del hijo:

| ***Partially mapped crossover*** | | | | | |
|---|---|---|---|---|---|
| Índice inicio | 3 | Índice fin | 3 | | |
| **Los elementos del primer padre que pertenecen al segmento marcado por ambos índices se copian en el cromosoma hijo** | | | | | |
| Individuo | Elem. 1 | Elem. 2 | Elem. 3 | Elem. 4 | Elem. 5 |
| Individuo 4 | 301 | 403 | 502 | **202** | 102 |
| Individuo 2 | 202 | 501 | 101 | 403 | 301 |
| **Hijo** | | | | **202** | |
| **Se evalúa qué elementos del padre 2 en el segmento no están en el segmento para el padre 1** | | | | | |
| Individuo | Elem. 1 | Elem. 2 | Elem. 3 | Elem. 4 | Elem. 5 |
| Individuo 4 | 301 | 403 | 502 | 202 | 102 |
| Individuo 2 | 202 | 501 | 101 | **403** | 301 |
| **Hijo** | | | | 202 | |
| **Por cada elemento faltante, se analiza el elemento del padre 1 que está en la misma posición que el faltante. Se descubre la posición de dicho elemento en el padre 2. En dicha posición del hijo se colocará el faltante. En caso dicha posición del hijo ya esté ocupada, el proceso se repite para dicha posición** | | | | | |
| Individuo | Elem. 1 | Elem. 2 | Elem. 3 | Elem. 4 | Elem. 5 |
| Individuo 4 | 301 | 403 | 502 | 202 | 102 |
| | | | | | |
| Individuo 2 | 202 | 501 | 101 | **403** | 301 |
| **Hijo** | **403** | | | 202 | |
| **El resto de elementos se copian desde el padre 2** | | | | | |
| Individuo | Elem. 1 | Elem. 2 | Elem. 3 | Elem. 4 | Elem. 5 |
| Individuo 4 | 301 | 403 | 502 | 202 | 102 |
| Individuo 2 | 202 | **501** | **101** | 403 | **301** |
| **Hijo** | 403 | **501** | **101** | 202 | **301** |

| **Swap Mutation** | | | | | |
|---|---|---|---|---|---|
| Valor aleatorio | 0,348694526 | Mutación? | **SI** | | |
| Índice 1 | 2 | Índice 2 | 0 | | |
| Individuo | Elem. 1 | Elem. 2 | Elem. 3 | Elem. 4 | Elem. 5 |
| Hijo actual | **403** | 501 | **101** | 202 | 301 |
| **Hijo mutado** | **101** | 501 | **403** | 202 | 301 |

Iteración 2, operador de búsqueda local:

| Búsqueda local | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Valor aleatorio | 0,959334 | Búsqueda? | NO | | | | | |
| Indice elemento | | Nuevo sitio | | <- se valida que coherencia con dist. de tablas en sitios | | | | |
| Individuo | Elem. 1 | Elem. 2 | Elem. 3 | Elem. 4 | Elem. 5 | | | |
| | | | | | | | | |
| | | | | | | | | |
| **Se compara la calidad de ambos cromosomas y se mantiene el mejor** | | | | | | | | |
| Individuo | Cost. Proc. Join 1 | Cost. Proc. Join 2 | Cost. Proc. Join 3 | Cost. Proc. Join 4 | Cost. Proc. | Cost. Com. | Cost. Tot. | **Valor Func.** |
| | | | | | | | | |
| | | | | | | | | |
| **Ganador 1** | | | | | | | | |
| Indice elemento | | Nuevo sitio | | | | | | |
| Individuo | Elem. 1 | Elem. 2 | Elem. 3 | Elem. 4 | Elem. 5 | | | |
| | | | | | | | | |
| | | | | | | | | |
| **Se compara la calidad de ambos cromosomas y se mantiene el mejor** | | | | | | | | |
| Individuo | Cost. Proc. Join 1 | Cost. Proc. Join 2 | Cost. Proc. Join 3 | Cost. Proc. Join 4 | Cost. Proc. | Cost. Com. | Cost. Tot. | **Valor Func. Objetivo** |
| | | | | | | | | |
| | | | | | | | | |
| **Ganador 2** | | | | | | | | |
| **Salida búsqueda** | Elem. 1 | Elem. 2 | Elem. 3 | Elem. 4 | Elem. 5 | | | |
| **Hijo mutado** | 101 | 501 | 403 | 202 | 301 | | | |

Iteración 2, evaluación de calidad de crom. hijo y evolución de la población:

| Evaluación de calidad de hijos | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Individuo | Cost. Proc. Join 1 | Cost. Proc. Join 2 | Cost. Proc. Join 3 | Cost. Proc. Join 4 | Cost. Proc. | Cost. Com. | Cost. Tot. | **Valor Func. Objetivo** |
| Hijo mutado | 20 | 700 | 0,0875 | 0,000066666666 | 720,0875667 | 242,50006 | 0,6037938885 | **1,65619430** |

| Evolución de la población | | <- se selecciona un individuo de la población para ser reemplazado por **el hijo. Si el individuo elegido es el de mayor calidad actualmente, se vuelve a elegir otro individuo aleatoriamente hasta que ya no se trate del mejor** | | | |
|---|---|---|---|---|---|
| Índice de población | 0 | Es el mejor individuo? | **NO** | Reemplazo? | SI |

| Nueva población | | | | | | |
|---|---|---|---|---|---|---|
| Individuo | Elem. 1 | Elem. 2 | Elem. 3 | Elem. 4 | Elem. 5 | Valor func. **Obj.** |
| Individuo 1 | 101 | 501 | 403 | 202 | 301 | **1,656194306** |
| Individuo 2 | 202 | 501 | 101 | 403 | 301 | **2,088959226** |
| Individuo 3 | 501 | 203 | 103 | 403 | 301 | **2,074482131** |
| Individuo 4 | 301 | 403 | 502 | 202 | 102 | **1,525544381** |

| Nuevo mejor individuo | |
|---|---|
| Individuo | Val. Func. Objetivo |
| Individuo 2 | **2,088959226** |

Iteración 3, población inicial y selección de padres:

| Población actual | | | | | | |
|---|---|---|---|---|---|---|
| Individuo | Elem. 1 | Elem. 2 | Elem. 3 | Elem. 4 | Elem. 5 | Valor func. Obj. |
| Individuo 1 | 101 | 501 | 403 | 202 | 301 | 1,656194306 |
| Individuo 2 | 202 | 501 | 101 | 403 | 301 | 2,088959226 |
| Individuo 3 | 501 | 203 | 103 | 403 | 301 | 2,074482131 |
| Individuo 4 | 301 | 403 | 502 | 202 | 102 | 1,525544381 |

| Fuentes de aleatoriedad | |
|---|---|
| Probabilidad | 0,697025123 |
| Valor min. | 0 |
| Valor max. | 3 |
| Entero | 0 |

| Mejor individuo actual | |
|---|---|
| Individuo | Val. Func. Objetivo |
| Individuo 2 | 2,088959226 |

y solo se casará a 2 individuos

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *pMut* | 0,7 | *pBusq* | 0,5 | % población a casar | 0,5 | # vecinos evaluados búsqueda | 2 |

| Casamiento de población | | | | | | |
|---|---|---|---|---|---|---|
| Individuos a casar | 2 | Padre 1 | 1 | Padre 2 | 4 | |
| Individuo | Elem. 1 | Elem. 2 | Elem. 3 | Elem. 4 | Elem. 5 | Valor func. Obj. |
| Individuo 1 | 101 | 501 | 403 | 202 | 301 | 1,656194306 |
| Individuo 4 | 301 | 403 | 502 | 202 | 102 | 1,525544381 |

Iteración 3, aplicación de *Partially Mapped Crossover* sobre padres y mutación por intercambio del hijo:

| *Partially mapped crossover* | | | | | |
|---|---|---|---|---|---|
| Índice inicio | 0 | Índice fin | 3 | | |
| **Los elementos del primer padre que pertenecen al segmento marcado por ambos índices se copian en el cromosoma hijo** | | | | | |
| Individuo | Elem. 1 | Elem. 2 | Elem. 3 | Elem. 4 | Elem. 5 |
| Individuo 1 | **101** | **501** | **403** | **202** | 301 |
| Individuo 4 | 301 | 403 | 502 | 202 | 102 |
| **Hijo** | **101** | **501** | **403** | **202** | |
| **Se evalúa qué elementos del padre 2 en el segmento no están en el segmento para el padre 1** | | | | | |
| Individuo | Elem. 1 | Elem. 2 | Elem. 3 | Elem. 4 | Elem. 5 |
| Individuo 1 | 101 | 501 | 403 | 202 | 301 |
| Individuo 4 | 301 | 403 | 502 | 202 | 102 |
| **Hijo** | 101 | 501 | 403 | 202 | |
| **Por cada elemento faltante, se analiza el elemento del padre 1 que está en la misma posición que el faltante. Se descubre la posición de dicho elemento en el padre 2. En dicha posición del hijo se colocará el faltante. En caso dicha posición del hijo ya esté ocupada, el proceso se repite para dicha posición** | | | | | |
| Individuo | Elem. 1 | Elem. 2 | Elem. 3 | Elem. 4 | Elem. 5 |
| Individuo 1 | 101 | 501 | 403 | 202 | 301 |
| | | | | | |
| Individuo 4 | **301** | 403 | 502 | 202 | 102 |
| **Hijo** | 101 | 501 | 403 | 202 | **301** |
| **El resto de elementos se copian desde el padre 2** | | | | | |
| Individuo | Elem. 1 | Elem. 2 | Elem. 3 | Elem. 4 | Elem. 5 |
| Individuo 1 | 101 | 501 | 403 | 202 | 301 |
| Individuo 4 | 301 | 403 | 502 | 202 | 102 |
| **Hijo** | 101 | 501 | 403 | 202 | 301 |

| **Swap Mutation** | | | | | |
|---|---|---|---|---|---|
| Valor aleatorio | 0,845669455 | Mutación? | NO | | |
| Índice 1 | | Índice 2 | | | |
| Individuo | Elem. 1 | Elem. 2 | Elem. 3 | Elem. 4 | Elem. 5 |
| | | | | | |
| | | | | | |

Iteración 3, operador de búsqueda local:

**Búsqueda local**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Valor aleatorio | 0,134673977 | Búsqueda? | SI | | | | | |
| Indice elemento | 4 | Nuevo sitio | 1 | <- se valida que coherencia con dist. de tablas en sitios | | | | |
| Individuo | Elem. 1 | Elem. 2 | Elem. 3 | Elem. 4 | Elem. 5 | | | |
| Hijo actual | 101 | 501 | 403 | 202 | 301 | | | |
| Vecino 1 | 101 | 501 | 403 | 202 | **301** | | | |
| **Se compara la calidad de ambos cromosomas y se mantiene el mejor** | | | | | | | | |
| Individuo | Cost. Proc. Join 1 | Cost. Proc. Join 2 | Cost. Proc. Join 3 | Cost. Proc. Join 4 | Cost. Proc. | Cost. Com. | Cost. Tot. | **Valor Func.** |
| Hijo actual | 20 | 700 | 0,0875 | 0,00006666666 | 720,0875667 | 242,5000 | 0,603793 | **1,65619430** |
| Vecino 1 | 20 | 700 | 0,0875 | 0,00006666666 | 720,0875667 | 242,5000 | 0,603793 | **1,65619430** |
| **Ganador 1** | **Vecino 1** | | | | | | | |
| Indice elemento | 0 | Nuevo sitio | 3 | | | | | |
| Individuo | Elem. 1 | Elem. 2 | Elem. 3 | Elem. 4 | Elem. 5 | | | |
| Vecino 1 | 101 | 501 | 403 | 202 | 301 | | | |
| Vecino 2 | **103** | 501 | 403 | 202 | 301 | | | |
| **Se compara la calidad de ambos cromosomas y se mantiene el mejor** | | | | | | | | |
| Individuo | Cost. Proc. Join 1 | Cost. Proc. Join 2 | Cost. Proc. Join 3 | Cost. Proc. Join 4 | Cost. Proc. | Cost. Com. | Cost. Tot. | **Valor Func. Objetivo** |
| Vecino 1 | 20 | 700 | 0,0875 | 0,00006666666 | 720,0875667 | 242,5000 | 0,603793 | **1,65619430** |
| Vecino 2 | 20 | 700 | 0,0875 | 0,00006666666 | 720,0875667 | 266,5000 | 0,608494 | **1,64340151** |
| **Ganador 2** | Vecino 1 | | | | | | | |
| **Salida búsqueda** | Elem. 1 | Elem. 2 | Elem. 3 | Elem. 4 | Elem. 5 | | | |
| **Vecino 1** | 101 | 501 | 403 | 202 | 301 | | | |

Iteración 3, evaluación de calidad de crom. hijo y evolución de la población:

| Evaluación de calidad de hijos | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Individuo | Cost. Proc. | Cost. Proc. | Cost. Proc. | Cost. Proc. | Cost. Proc. | Cost. | Cost. | **Valor Func.** |
| Hijo modificado | 20 | 700 | 0,0875 | 0,00006666666 | 720,0875667 | 242,5000 | 0,603793 | **1,65619430** |

| Evolución de la población | | <- se selecciona un individuo de la población para ser reemplazado por el hijo. Si el individuo elegido es el de mayor calidad actualmente, se vuelve a elegir otro individuo aleatoriamente hasta que ya no se trate del mejor | | | |
|---|---|---|---|---|---|
| Índice de población | 1 | Es el mejor individuo? | SI | Reemplazo? | NO |
| Índice de población | 3 | Es el mejor individuo? | NO | Reemplazo? | SI |

| Nueva población | | | | | | |
|---|---|---|---|---|---|---|
| Individuo | Elem. 1 | Elem. 2 | Elem. 3 | Elem. 4 | Elem. 5 | Valor func. Obj. |
| Individuo 1 | 101 | 501 | 403 | 202 | 301 | 1,656194306 |
| Individuo 2 | 202 | 501 | 101 | 403 | 301 | 2,088959226 |
| Individuo 3 | 501 | 203 | 103 | 403 | 301 | 2,074482131 |
| Individuo 4 | 101 | 501 | 403 | 202 | 301 | 1,656194306 |

| Nuevo mejor individuo | |
|---|---|
| Individuo | Val. Func. Objetivo |
| Individuo 2 | 2,088959226 |

Resultados de las pruebas y conclusiones:

| Población inicial generada | | | | | | |
|---|---|---|---|---|---|---|
| Individuo | Elem. 1 | Elem. 2 | Elem. 3 | Elem. 4 | Elem. 5 | Valor func. Obj. |
| Individuo 1 | 101 | 203 | 301 | 403 | 502 | 1,884007964 |
| Individuo 2 | 202 | 501 | 101 | 403 | 301 | 2,088959226 |
| Individuo 3 | 501 | 301 | 103 | 403 | 202 | 1,671331608 |
| Individuo 4 | 301 | 403 | 502 | 202 | 102 | 1,525544381 |

| Población final generada | | | | | | |
|---|---|---|---|---|---|---|
| Individuo | Elem. 1 | Elem. 2 | Elem. 3 | Elem. 4 | Elem. 5 | Valor func. Obj. |
| Individuo 1 | 101 | 501 | 403 | 202 | 301 | 1,656194306 |
| Individuo 2 | 202 | 501 | 101 | 403 | 301 | 2,088959226 |
| Individuo 3 | 501 | 203 | 103 | 403 | 301 | 2,074482131 |
| Individuo 4 | 101 | 501 | 403 | 202 | 301 | 1,656194306 |

| Mejor individuo inicial | |
|---|---|
| Individuo | Val. Func. Objetivo |
| Individuo 2 | 2,088959226 |

| Mejor individuo final | |
|---|---|
| Individuo | Val. Func. Objetivo |
| Individuo 2 | 2,088959226 |

**Conclusiones**

Se observa que la ejecución del algoritmo con un conjunto de datos pequeños y parámetros controlados permitió mantener la calidad de la mejor solución y mejorar la calidad promedio de la población en lugar de empeorarlas. Para lograr esto, aplicó los operadores de casamiento, mutación y búsqueda local (estos dos últimos de manera probabilística) para producir nuevas generaciones de la población.

En base a lo anterior, se puede concluir que la lógica del algoritmo realmente permite la optimización de soluciones posibles para la optimización de consultas SQL.

## Anexo E: Ejemplo de archivo de datos de entrada al algoritmo memético

| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 5 | 3 | 12 | 5 | 10 | 0.5 | | | | | | | | | |
| 2 | - | | | | | | | | | | | | | | |
| 3 | 1000 | 24000 | 1000 | 800 | 300 | 50 | 380 | 530 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 4 | 300 | 6000 | 0 | 0 | 300 | 40 | 0 | 0 | 120 | 90 | 200 | 0 | 0 | 0 | |
| 5 | 1800 | 36000 | 0 | 1800 | 210 | 30 | 0 | 0 | 500 | 0 | 0 | 0 | 0 | 0 | |
| 6 | 50 | 400 | 0 | 0 | 0 | 50 | 0 | 0 | 0 | 0 | 0 | 50 | 0 | 0 | |
| 7 | 700 | 28000 | 0 | 0 | 0 | 20 | 0 | 700 | 200 | 0 | 0 | 0 | 350 | 500 | |
| 8 | - | | | | | | | | | | | | | | |
| 9 | 1 | 1 | 1 | | | | | | | | | | | | |
| 10 | 0 | 1 | 1 | | | | | | | | | | | | |
| 11 | 1 | 0 | 0 | | | | | | | | | | | | |
| 12 | 0 | 1 | 1 | | | | | | | | | | | | |
| 13 | 1 | 1 | 0 | | | | | | | | | | | | |
| 14 | - | | | | | | | | | | | | | | |
| 15 | 0 | 500 | 1000 | | | | | | | | | | | | |
| 16 | 500 | 0 | 200 | | | | | | | | | | | | |
| 17 | 1000 | 200 | 0 | | | | | | | | | | | | |
| 18 | - | | | | | | | | | | | | | | |
| 19 | 10000 | 1000 | 0.5 | 0.7 | 0.6 | 0.9 | 25 | 0.3 | 1 | | | | | | |
| 20 | | | | | | | | | | | | | | | |

## Anexo F: Código fuente del algoritmo memético

Enlace del archivo comprimido con el proyecto de Java:

https://drive.google.com/file/d/18gS-LGKwmjZ2ScRy62G_Zk414cerFA4V/view?usp=sharing

## Anexo G: Código fuente del algoritmo genético

Enlace del archivo comprimido con el proyecto de Java:

https://drive.google.com/file/d/1qWGnRm1AqmVvVksfBcSxcse8egTCA0tO/view?usp=sharing

## Anexo H: Código fuente del proyecto de software de comparación de algoritmos

Enlace del archivo comprimido con el proyecto de Java:

https://drive.google.com/file/d/14gz6s2XWmahqeGf1CMCIge2-FUHVpfYB/view?usp=sharing

## Anexo I: Conjunto de datos de ejecución de algoritmos para experimentación numérica

Enlace de archivo de excel con el conjunto de datos de ejecución:

https://docs.google.com/spreadsheets/d/1ZOpk9tvVw5L_u7krpLtx1mHjgmPtuyX1/edit?usp=sharing&ouid=117223103864494113454&rtpof=true&sd=true

Promedios de datos de ejecución para cada combinación algoritmo + configuración del problema (1000 registros por cada combinación):

| Algoritmo | Problema | Fitness1 | Fitness10 | Fitness20 | Tiempo |
|---|---|---|---|---|---|
| MEM | C1 | 2,49649153 | 2,213201651 | 2,156892142 | 1056,351457 |
| GEN | C1 | 2,49649153 | 2,210156734 | 2,155444664 | 166,8759943 |
| MEM | C2 | 2,404214625 | 2,008088557 | 1,96831637 | 1333,005673 |
| GEN | C2 | 2,40108293 | 2,005961366 | 1,966627157 | 315,9301921 |
| MEM | C3 | 3,397766916 | 2,359572609 | 2,291211181 | 2168,363428 |
| GEN | C3 | 3,062917314 | 2,356448097 | 2,288630161 | 375,4428084 |

Muestra de los datos (10 registros para cada combinación algoritmo + configuración del problema:

| Algoritmo | Problema | Fitness1 | Fitness10 | Fitness20 | Tiempo |
|---|---|---|---|---|---|
| MEM | C1 | 2,49649153 | 2,192227035 | 2,149468033 | 1243,4992 |
| MEM | C1 | 2,49649153 | 2,196516209 | 2,150057005 | 1050,6272 |
| MEM | C1 | 2,49649153 | 2,321413136 | 2,248500874 | 987,4506 |
| MEM | C1 | 2,49649153 | 2,253768211 | 2,175787541 | 983,6177 |
| MEM | C1 | 2,49649153 | 2,207197297 | 2,154800109 | 984,9224 |
| MEM | C1 | 2,49649153 | 2,227252958 | 2,17820956 | 990,7757 |
| MEM | C1 | 2,49649153 | 2,197361208 | 2,137631001 | 981,9019 |
| MEM | C1 | 2,49649153 | 2,159162028 | 2,109759829 | 985,8041 |
| MEM | C1 | 2,49649153 | 2,209573749 | 2,142770778 | 988,9366 |
| MEM | C1 | 2,49649153 | 2,207255711 | 2,150735538 | 985,2781 |
| GEN | C1 | 2,49649153 | 2,248518299 | 2,183792397 | 175,2151 |
| GEN | C1 | 2,49649153 | 2,214098801 | 2,162679119 | 173,6302 |
| GEN | C1 | 2,49649153 | 2,189944332 | 2,1438906 | 165,146 |
| GEN | C1 | 2,49649153 | 2,239743096 | 2,175197848 | 166,1142 |
| GEN | C1 | 2,49649153 | 2,209795407 | 2,170630315 | 165,0781 |
| GEN | C1 | 2,49649153 | 2,147115833 | 2,116834647 | 163,4606 |
| GEN | C1 | 2,49649153 | 2,286029927 | 2,198839072 | 172,7083 |
| GEN | C1 | 2,49649153 | 2,202041989 | 2,161823029 | 172,7174 |
| GEN | C1 | 2,49649153 | 2,19121797 | 2,153349416 | 164,2745 |

| | | | | | |
|---|---|---|---|---|---|
| GEN | C1 | 2,49649153 | 2,216750978 | 2,155832457 | 165,5814 |
| MEM | C2 | 2,404217602 | 1,989672923 | 1,952024776 | 1766,9973 |
| MEM | C2 | 2,404217602 | 2,011737116 | 1,970488773 | 2081,1261 |
| MEM | C2 | 2,404217602 | 2,002119592 | 1,956130815 | 1088,5344 |
| MEM | C2 | 2,404217602 | 1,998717599 | 1,954980594 | 1062 |
| MEM | C2 | 2,404217602 | 2,038894722 | 1,991825682 | 1318,9683 |
| MEM | C2 | 2,404217602 | 1,993192136 | 1,964902157 | 1263,9714 |
| MEM | C2 | 2,404217602 | 2,043099567 | 1,981697623 | 1645,6083 |
| MEM | C2 | 2,404217602 | 1,960007067 | 1,930859515 | 1293,9684 |
| MEM | C2 | 2,404217602 | 2,018520952 | 1,982518671 | 1287,5313 |
| MEM | C2 | 2,404075843 | 2,027973244 | 1,975465739 | 1183,5352 |
| GEN | C2 | 2,404075628 | 2,006406374 | 1,977208107 | 199,3069 |
| GEN | C2 | 2,404075628 | 1,964378147 | 1,935889003 | 514,0148 |
| GEN | C2 | 2,40407553 | 2,002479607 | 1,965030024 | 229,2053 |
| GEN | C2 | 2,404025399 | 2,019841441 | 1,976005571 | 383,1543 |
| GEN | C2 | 2,404075747 | 1,992200894 | 1,958294891 | 472,648101 |
| GEN | C2 | 2,40368488 | 2,012001822 | 1,976292205 | 219,4075 |
| GEN | C2 | 2,40407553 | 1,971765863 | 1,941937228 | 221,014499 |
| GEN | C2 | 2,386896528 | 2,016339356 | 1,968598443 | 239,3739 |
| GEN | C2 | 2,404075628 | 1,989552616 | 1,949968806 | 294,7139 |
| GEN | C2 | 2,404075875 | 1,997790613 | 1,958146796 | 403,0654 |
| MEM | C3 | 3,346295168 | 2,353230808 | 2,267288219 | 2387,2806 |
| MEM | C3 | 3,420690419 | 2,310626755 | 2,258115797 | 1917,7147 |
| MEM | C3 | 3,420690419 | 2,460750744 | 2,381492403 | 1297,1658 |
| MEM | C3 | 3,400909684 | 2,454620101 | 2,349211406 | 1990,8455 |
| MEM | C3 | 3,366196164 | 2,369329683 | 2,290284251 | 1407,9559 |
| MEM | C3 | 3,366193163 | 2,363173376 | 2,283920339 | 1811,035 |
| MEM | C3 | 3,42095553 | 2,417084587 | 2,301384356 | 1798,2151 |
| MEM | C3 | 3,42095553 | 2,396152683 | 2,333417446 | 1832,4185 |
| MEM | C3 | 3,422227925 | 2,355714529 | 2,307698975 | 2953,0457 |
| MEM | C3 | 3,422227985 | 2,350291977 | 2,287636473 | 2708,6197 |
| GEN | C3 | 3,257557728 | 2,351197917 | 2,283106579 | 557,8498 |
| GEN | C3 | 3,020709987 | 2,33794989 | 2,279619826 | 235,9179 |
| GEN | C3 | 3,214005697 | 2,396285186 | 2,312368065 | 309,4263 |
| GEN | C3 | 3,114080513 | 2,377108032 | 2,326246268 | 413,4733 |
| GEN | C3 | 3,253146351 | 2,390047507 | 2,290069725 | 347,1831 |
| GEN | C3 | 3,079559471 | 2,351191638 | 2,294930305 | 259,982 |
| GEN | C3 | 3,101857301 | 2,325895787 | 2,268480406 | 517,0152 |
| GEN | C3 | 3,026393441 | 2,404924158 | 2,326536898 | 417,8475 |
| GEN | C3 | 3,142653118 | 2,359177239 | 2,302442575 | 488,5782 |
| GEN | C3 | 3,017847785 | 2,350991633 | 2,277608597 | 481,4933 |

**Anexo J: Formulario de validación de resultado R1.1 llenado por experto en bases de datos**

cesar.aguilera@pucp.edu.pe

3 de 3

No se pueden editar las respuestas

# Proyecto de Tesis 2 - Validación resultado esperado 1.1

Tema de tesis: Algoritmo metaheurístico para la optimización de consultas en bases de datos relacionales
Asesor: Rony Cueva Moscoso
Tesista: Jesús Angel Sangama Ramírez

Breve resumen: El proyecto de tesis tiene como fin desarrollar un algoritmo memético para realizar la optimización de consultas/queries SQL en el contexto de bases de datos relacionales distribuidas (múltiples sitios/servidores separados físicamente en los que tablas se encuentran distribuidas y/o replicadas).

El presente formulario tiene como objetivo determinar si el resultado esperado 1.1 (Variables, parámetros y restricciones que serán consideradas en el algoritmo propuesto) es válido bajo el punto de vista de un experto en la materia (en este caso, en bases de datos).

*Obligatorio

Correo *

cesar.aguilera@pucp.edu.pe

Por favor, ingrese su nombre: *

César Aguilera Serpa

A continuación, se mostrarán los factores (variables, parámetros) a considerarse en el proyecto de tesis para realizar la optimización de consultas SQL. Seleccione aquellos que considera relevantes para este propósito. *

- [x] Cantidad de tablas usadas en la consulta SQL
- [x] Cantidad de sitios/servidores en la base de datos
- [x] Distribución de las tablas en los sitios (en qué sitio o sitios se puede encontrar cada tabla, ej: Tabla 1 se encuentra en sitios 1 y 3 de la BD, Tabla 2 se encuentra solo en el sitio 2, etc.)
- [x] Tamaño de las tablas (número de filas, columnas, bytes)
- [x] Cardinalidades de cada tabla (cantidad de valores únicos que cada columna posee)
- [x] Costos de procesamiento (costo incurrido para procesar operaciones de join entre tablas en los sitios de la base de datos)
- [x] Capacidad de transmisión entre los sitios de la base de datos (en bytes por segundo)
- [x] Costos de comunicación (costo incurrido al enviar una tabla de un sitio a otro)

¿Tiene usted comentarios u observaciones respecto al resultado entregado (dudas, puntos a mejorar, puntos faltantes, etc.)?

Quizás precisar el tipo de arquitectura, si es estructurada pura u otra.

¿A qué porcentaje diría usted que aprueba el resultado presentado? *

- ( ) 0% (no lo aprueba)
- ( ) 20%
- ( ) 40%
- ( ) 60%
- ( ) 80%
- (•) 100%

Enviado: 6/11/21 6:24

## Anexo K: Formulario de validación de resultado R1.2 llenado por experto en algoritmia

tupia.mf@pucp.edu.pe

2 de 2

No se pueden editar las respuestas

# Proyecto de Tesis 2 - Validación resultado esperado 1.2

Tema de tesis: Algoritmo metaheurístico para la optimización de consultas en bases de datos relacionales
Asesor: Rony Cueva Moscoso
Tesista: Jesús Angel Sangama Ramírez

Breve resumen: El proyecto de tesis tiene como fin desarrollar un algoritmo memético en el lenguaje Java para realizar la optimización de consultas/queries SQL en el contexto de bases de datos relacionales distribuidas (múltiples sitios/servidores separados físicamente en los que tablas se encuentran distribuidas y/o replicadas).

El presente formulario tiene como objetivo determinar si el resultado esperado 1.2 (estructuras de datos para representar variables del problema en el algoritmo propuesto) es válido bajo el punto de vista de un experto en la materia (en este caso, en algoritmia).

*Obligatorio

Correo *

tupia.mf@pucp.edu.pe

Por favor, ingrese su nombre: *

MANUEL TUPIA ANTICONA

## Descripción de estructuras de datos por variable a considerar

A continuación se mostrará una lista con las variables consideradas para el problema de optimización de consultas SQL. Cada una irá acompañada de la estructura de datos escogida para representarla, y con una imagen para ilustrar dicha estructura, de ser necesario.

Formato de presentación:

Número - Descripción de la variable (Nombre de la variable)

Propósito de la variable en el algoritmo

Estructura de datos a utilizar para representar la variable

Imagen opcional

### 1. Cantidad de tablas usadas en la consulta SQL (NumTab)

Propósito:
Esta variable se utiliza para calcular la cantidad de operaciones que conllevará una determinada consulta, y también será necesaria para crear estructuras de datos descritas más adelante.

Estructura de datos:
Como se trata simplemente de un número entero, se le puede almacenar en una variable de tipo de dato int.

### 2. Cantidad de sitios/servidores en la base de datos (NumSit)

Propósito:
La cantidad de sitios se utilizará para la construcción de posibles soluciones y para determinar si será necesario realizar transmisiones de datos entre sitios diferentes. También es necesaria para crear estructuras de datos descritas más adelante

Estructura de datos:
Al tratarse también de un número entero, se le puede almacenar en una variable de tipo de dato int.

### 3. Distribución de tablas en los sitios de la base de datos (DistTabSit)

Propósito:
Como el problema de optimización de consultas SQL planteado involucra la distribución y replicación de tablas en diferentes sitios de una base de datos, es necesario conocer en qué sitio o sitios se encuentra cada tabla para poder construir soluciones al problema, identificar la validez de una solución, y determinar el orden en el que los datos pueden transmitirse entre sitios.

Estructura de datos:
Se decidió representar esta variable con una matriz bidimensional de valores binarios de tamaño NumTab x NumSit. Cada elemento de la matriz indica si la tabla i se encuentra (1) o no (0) en el sitio j. Un ejemplo de la matriz se muestra a continuación:

Ejemplo de matriz de distribución de tablas en sitios (N tablas y M sitios)

| | S1 | S2 | ... | Sm |
|---|---|---|---|---|
| T1 | 1 | 0 | ... | 1 |
| T2 | 0 | 1 | ... | 0 |
| ... | ... | ... | ... | ... |
| Tn | 1 | 1 | ... | 1 |

### 4. Tamaño de una tabla (NumFil, NumCol, NumByte)

Propósito:
Conocer el tamaño de una tabla, sea en cantidad de filas/columnas (NumFil y NumCol) o tamaño en bytes (NumByte), es importante para determinar el peso que dicha tabla tendrá en el procesamiento necesario para ejecutar una consulta SQL. Por ejemplo, mientras más grande sea una tabla, más costoso resultará transmitir sus datos de un sitio a otro.

Estructura de datos:
Las tres variables de tamaño de una tabla son numéricas y enteras, por lo que pueden ser almacenadas en variables de tipo de dato int. Es importante considerar que estas 3 variables existen para cada tabla del problema, por lo que en la implementación en Java en realidad serán atributos de una clase Tabla.

### 5. Cardinalidades de una tabla (CardTab)

Propósito:
En el proyecto, se define cardinalidad como la cantidad de valores distintos que existen en una columna de una tabla específica. Siguiendo los trabajos académicos revisados, conocer estas cardinalidades es necesario para poder calcular el costo de procesamiento de un join entre dos tablas.

Estructura de datos:
Tomando en consideración que para una tabla existen múltiples columnas, se decidió que las cardinalidades de una tabla específica sean almacenadas en un arreglo cuyo tamaño sea la cantidad total de columnas de todas las tablas. En la posición i del arreglo se almacenará la cardinalidad de la columna i para la tabla a la que referencia el arreglo. Sin embargo, también se consideró tener las cardinalidades de todas las tablas juntas para facilitar el acceso a dichos datos. Así, también es posible unir los arreglos de cada tabla en otro arreglo de tamaño NumTab (formándose así una matriz). A continuación se muestra un ejemplo tanto de un arreglo solo como de la matriz para todas las tablas:

Ejemplo de arreglo y matriz de cardinalidades (N tablas y M columnas en total)

## Arreglo para T1:

| C1 | C2 | ... | Cm |
|---|---|---|---|
| **50** | **1000** | **...** | **652** |

## Matriz para todas las tablas:

| | C1 | C2 | ... | Cm |
|---|---|---|---|---|
| T1 | **50** | **1000** | **...** | **652** |
| T2 | **0** | **600** | **...** | **1000** |
| ... | **...** | **...** | **...** | **...** |
| Tn | **50** | **0** | **...** | **0** |

### 6. Costos de procesamiento (CostProc)

Propósito:
Esta variable hace referencia al costo computacional de realizar una operación de join entre 2 tablas. Como para una consulta se pueden realizar múltiples joins secuenciales, es importante conocer el costo de procesamiento de cada join de una solución generada por el algoritmo. Estos costos se utilizarán para calcular el fitness de dicha solución.

Estructura de datos:
Se ha decidido representar esta variable con un arreglo de números reales de tamaño NumTab-1 (pues para una consulta con X tablas se realizarán X-1 joins). Cada elemento del arreglo contendrá el costo de procesamiento de uno de los joins de la solución generada. Un ejemplo del arreglo se muestra a continuación:

Ejemplo de arreglo de costos de procesamiento (N tablas)

| T1 | T2 | T3 | ... | Tn |
|---|---|---|---|---|
| 204.8 | 8.7 | 1000.0 | ... | 127.2 |

### 7. Costos de comunicación (CostCom)

Propósito:
El costo de comunicación es el incurrido al realizar transmisiones de datos entre dos sitios de la base de datos, generalmente para enviar el contenido de una tabla a otro sitio en el que se realizará una operación join con una segunda tabla. Este costo es otro componente necesario para calcular el fitness de una determinada solución generada por el algoritmo.

Estructura de datos:
Considerando que la cantidad de transmisiones en una solución no es fija (depende de los sitios de donde se traiga cada tabla a consultar), se decidió calcular los costos de comunicación como un único valor numérico real. Por lo tanto, para almacenarlos solo será necesaria una variable de tipo double.

### 8. Capacidades de transmisión entre sitios (CapTrans)

Propósito:
El conocer las capacidades de transmisión entre los diferentes sitios de la base de datos (en bytes por segundo) será necesario para calcular los costos de comunicación incurridos para ejecutar una determinada solución que genere el algoritmo.

Estructura de datos:
Esta variable será representada por una matriz cuadrada de valores reales de tamaño NumSit x NumSit. El elemento en una posición arbitraria (i,j) representa la capacidad de transmisión en bytes por segundo entre el sitio i y el sitio j. Considerando esto, esta matriz siempre tendrá 0s en la diagonal mayor, pues no es razonable realizar transmisiones de datos entre un único sitio. Además, como estas capacidades son las mismas sin importar el origen y el destino, los elementos de la matriz estarán reflejados. Un ejemplo se muestra a continuación:

Ejemplo de matriz de capacidades de transmisión (N sitios)

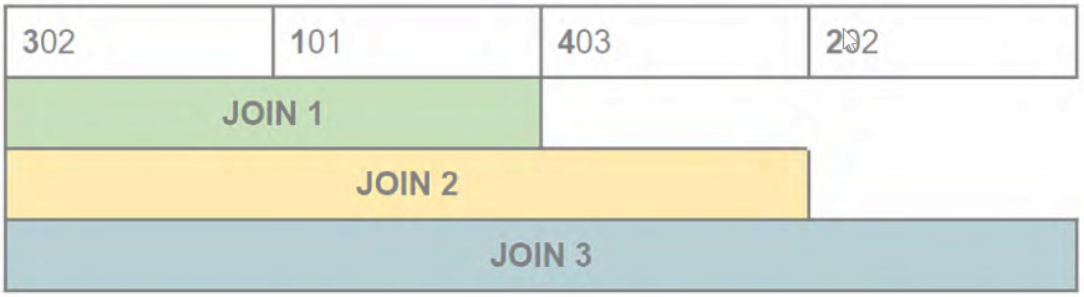
| | S1 | S2 | ... | Sn |
|---|---|---|---|---|
| S1 | 0 | 1000 | ... | 5000 |
| S2 | 1000 | 0 | ... | 12000 |
| ... | ... | ... | ... | ... |
| Sn | 5000 | 12000 | ... | 0 |

## 9. Estructura del cromosoma

Los algoritmos evolutivos utilizan una entidad denominada cromosoma, la cual representa una posible solución al problema a optimizar. En este caso, las soluciones al problema de optimización de consultas SQL son estructuras que representen el orden en que se les aplicarán joins a las tablas y los sitios desde donde se obtendrá cada tabla.

Considerando lo anterior, se decidió en conjunto con el asesor de tesis que la estructura del cromosoma sea un arreglo de números enteros de tamaño NumTab. Cada elemento del arreglo representa un par tabla-sitio (Las centenas del número son el número de la tabla y las unidades son el número del sitio desde donde se obtiene dicha tabla. Así, cada elemento contiene la información de los sitios y el orden de los elementos del arreglo contiene la información del orden de joins realizados. Cabe resaltar que, como el cromosoma contiene valores enteros en lugar de binarios, el algoritmo a desarrollar entra en la categoría de algoritmos metaheurísticos continuos. Un ejemplo del cromosoma es el siguiente:

Ejemplo de cromosoma para un problema con 4 tablas (T1-T4) y 3 sitios (S1-S3)



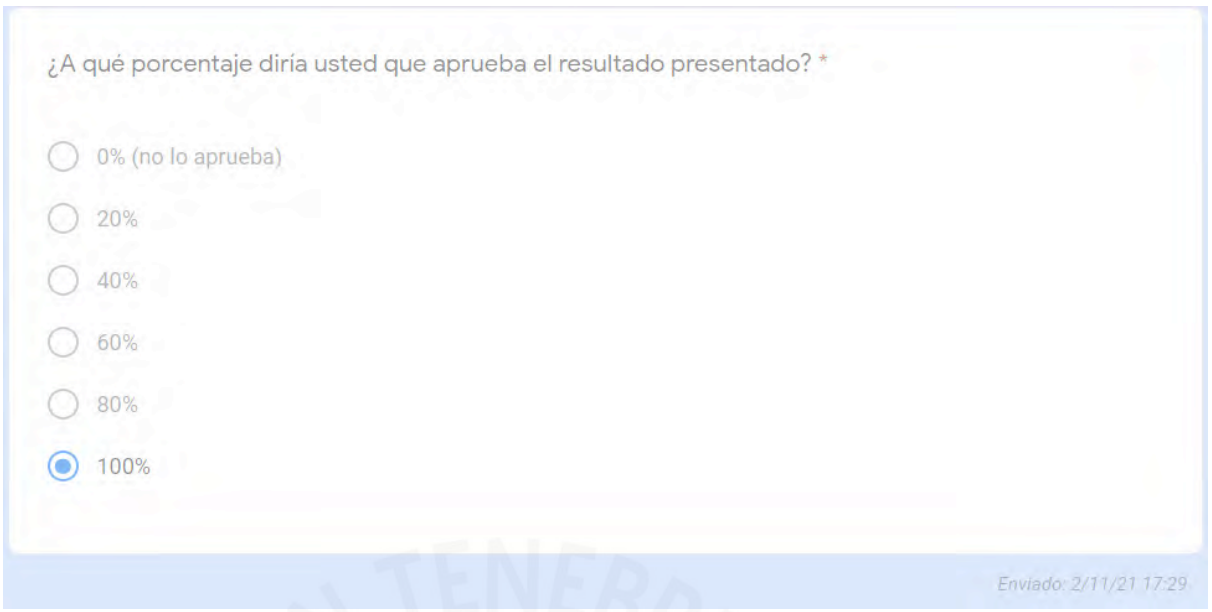

El cromosoma equivale al siguiente orden de joins: **T3 en S2** con **T1 en S1** forman el **JOIN1**; el **JOIN1 en S1** con **T4 en S3** forman el **JOIN2**; y finalmente el **JOIN2 en S3** con **T2 en S2** forman el **JOIN3 en S2**, el cual contiene el resultado final de la consulta.

A continuación, se muestra un listado de las variables descritas anteriormente. Seleccione aquellas para las cuales considera que se ha escogido una estructura de datos válida. *

- [x] Cantidad de tablas usadas en la consulta SQL
- [x] Cantidad de sitios/servidores en la base de datos
- [x] Distribución de las tablas en los sitios (en qué sitio o sitios se puede encontrar cada tabla, ej: Tabla 1 se encuentra en sitios 1 y 3 de la BD, Tabla 2 se encuentra solo en el sitio 2, etc.)
- [x] Tamaño de las tablas (número de filas, columnas, bytes)
- [x] Cardinalidades de las tablas (número de valores diferentes en una columna de una tabla)
- [x] Costos de procesamiento (costo incurrido para procesar operaciones de join entre tablas en los sitios de la base de datos)
- [x] Costos de comunicación (costo incurrido al enviar una tabla de un sitio a otro)
- [x] Capacidad de transmisión entre los sitios de la base de datos (en bytes por segundo)
- [x] Estructura del cromosoma para el algoritmo

¿Tiene usted comentarios u observaciones respecto al resultado entregado (dudas, puntos a mejorar, puntos faltantes, etc.)?

Ninguna duda


¿A qué porcentaje diría usted que aprueba el resultado presentado? *
0% (no lo aprueba)
20%
40%
60%
80%
100%
Enviado: 2/11/21 17:29

## Anexo L: Formulario de validación de resultado R1.3 llenado por experto en algoritmia

tupia.mf@pucp.edu.pe

1 de 1

No se pueden editar las respuestas

# Proyecto de Tesis 2 - Validación resultado esperado 1.3

Tema de tesis: Algoritmo metaheurístico para la optimización de consultas en bases de datos relacionales
Asesor: Rony Cueva Moscoso
Tesista: Jesús Angel Sangama Ramírez

Breve resumen: El proyecto de tesis tiene como fin desarrollar un algoritmo memético en el lenguaje Java para realizar la optimización de consultas/queries SQL en el contexto de bases de datos relacionales distribuidas (múltiples sitios/servidores separados físicamente en los que tablas se encuentran distribuidas y/o replicadas).

El presente formulario tiene como objetivo determinar si el resultado esperado 1.3 (función objetivo a utilizar en el algorimo) es válido bajo el punto de vista de un experto en la materia (en este caso, en algoritmia).

*Obligatorio

Correo *

tupia.mf@pucp.edu.pe

Por favor, ingrese su nombre: *

MANUEL TUPIA ANTICONA

## Descripción de la función objetivo

A continuación se explicará la estructura de la fórmula utilizada para calcular el fitness de las soluciones (representadas por cromosomas) que el algoritmo generará durante su ejecución. En primer lugar, se tiene la fórmula para calcular el valor de la función objetivo para un determinado cromosoma (FO):

1. Valor de la función objetivo para un cromosoma

$$FO = \frac{1}{CostTotal}$$

Se puede observar que el valor de la FO es la inversa del costo total calculado para el cromosoma en cuestión (CostTotal). Esto se debe a que el algoritmo busca maximizar el valor de la función objetivo, pero el costo total es un valor que debería minimizarse. Esta primera fórmula siempre resultará en valores válidos porque el costo total siempre será mayor que 0. A continuación, se mostrará la fórmula para calcular el costo total (CostTotal):

2. Costo total de ejecutar una consulta SQL siguiendo la información de un cromosoma:

$$CostTotal = C_{COM} \star \frac{ln(1+CostCom)}{10} + C_{PROC} \star \frac{ln\left(1+\sum_{i=1}^{JOINS} CostProc_i\right)}{10}$$

En la fórmula, CostCom y CostProc i hacen referencia a los costos de comunicación (aquellos incurridos por transmisiones de datos de una tabla entre sitios de la base de datos) y a los costos de procesamiento de una determinada operación join, respectivamente. Al costo de comunicación total y a la sumatoria de costos de procesamiento de todos los joins se le ha aplicado la función log(1+X)/10 Esto con el objetivo de que uno de los costos no vuelva insignificante al otro debido a la escala de sus valores (ya que esta transformación convertirá cada costo a un valor entre 0 y 5 aproximadamente). Esta transformación fue aplicada por recomendación del asesor de tesis.

Además, Ccom y Cproc son coeficientes de peso para estos dos costos que se pueden cambiar según se desee (Ccom+Cproc=100%). Esto da libertad al que ejecute el algoritmo de decidir si quiere centrar la optimización en base a uno de los dos costos o si quiere que ambos tengan el mismo peso.

Ahora se procederá a mostrar la fórmula para calcular los costos de comunicación totales del cromosoma (CostCom):

3. Costo de comunicación total de un cromosoma:

$$CostCom = \sum_{i=1}^{TRANS} (OH + \frac{NumByte_i}{CapTrans})$$

En la fórmula, TRANS representa el número de transmisiones realizadas en la solución representada por el cromosoma. El costo de cada transmisión se muestra en la expresión al interior de la sumatoria y está compuesta por:

-OH: Un tiempo de overhead para iniciar y concluir la transmisión (constante para todas las transmisiones.

-NumByte i: Número de bytes a ser enviados en la transmisión i.

-CapTrans: La capacidad de transmisión (en bytes por segundo) entre los sitios de la base de datos que participan en la transmisión

Finalmente, se mostrará y explicará la fórmula utilizada para calcular el costo de procesamiento de una determinada operación join (CostProc i):

4. Costo de procesamiento de una operación join:

$$CostProc_i = \frac{NumFil_A * NumFil_B}{\prod_{k \in C} max(CardTab_{Ak}, CardTab_{Bk})}$$

En la fórmula, A y B representan las dos tablas participantes en la operación join, y C representa la lista de columnas que son comunes a ambas tablas. Así, la expresión para calcular el costo de procesamiento es la división entre:

-El producto de los números de filas de las tablas A y B

-El producto de las cardinalidades máximas entre las tablas A y B para cada una de sus columnas comunes. En el proyecto de tesis, se define cardinalidad como la cantidad de valores distintos que existen en una columna de una tabla específica.

Esta fórmula fue elegida debido a que se observó su uso para representar el costo de procesamiento de un join en varios de los trabajos académicos leídos durante la revisión sistemática de la literatura.

¿Tiene usted comentarios u observaciones respecto al resultado entregado (dudas, puntos a mejorar, puntos faltantes, etc.)?

Ninguna corrección, las fórmulas son correctas

¿A qué porcentaje diría usted que aprueba el resultado presentado? *

- ○ 0% (no lo aprueba)
- ○ 20%
- ○ 40%
- ○ 60%
- ○ 80%
- ◉ 100%

*Enviado: 4/11/21 10:06*

## Anexo M: Formulario de validación de resultado R2.1 llenado por experto en algoritmia metaheurística

Este y los formularios de validación posteriores representan un acta de validación tras una reunión por videollamada con el experto en cuestión.

tupia.mf@pucp.edu.pe 1 de 1

No se pueden editar las respuestas

Proyecto de Tesis 2 - Validación resultado esperado 2.1

Tema de tesis: Algoritmo metaheurístico para la optimización de consultas en bases de datos relacionales
Asesor: Rony Cueva Moscoso
Tesista: Jesús Angel Sangama Ramírez

Breve resumen: El proyecto de tesis tiene como fin desarrollar un algoritmo memético en el lenguaje Java para realizar la optimización de consultas/queries SQL en el contexto de bases de datos relacionales distribuidas (múltiples sitios/servidores separados físicamente en los que tablas se encuentran distribuidas y/o replicadas).

El presente formulario tiene como objetivo determinar si el resultado esperado 2.1 (pseudocódigo del algoritmo memético) es válido bajo el punto de vista de un experto en la materia (en este caso, en algoritmia metaheurística).

*Obligatorio

Correo *

tupia.mf@pucp.edu.pe

Por favor, ingrese su nombre: *

MANUEL TUPIA ANTICONA

Por favor, confirme si se reunió con el tesista Jesús Sangama Ramírez para la explicación del resultado esperado en la siguiente fecha: 05/11/2021 *

- (●) Sí nos reunimos
- ( ) No nos reunimos
- ( ) Nos reunimos, pero en otra fecha

Si se reunió con el tesista en una fecha diferente a la especificada anteriormente, por favor, ingrese dicha fecha:

DD MM AAAA

05 / 11 / 2021

¿Tiene usted comentarios u observaciones respecto al resultado entregado (dudas, puntos a mejorar, puntos faltantes, etc.)?

Ninguna, el resultado mostrado es correcto

¿A qué porcentaje diría usted que aprueba el resultado presentado? *

- ( ) 0% (no lo aprueba)
- ( ) 20%
- ( ) 40%
- ( ) 60%
- ( ) 80%
- (●) 100%

Enviado: 5/11/21 21:42

## Anexo N: Formulario de validación de resultado R2.3 llenado por experto en algoritmia metaheurística

tupia.mf@pucp.edu.pe 1 de 1

No se pueden editar las respuestas

# Proyecto de Tesis 2 - Validación resultado esperado 2.3

Tema de tesis: Algoritmo metaheurístico para la optimización de consultas en bases de datos relacionales
Asesor: Rony Cueva Moscoso
Tesista: Jesús Angel Sangama Ramírez

Breve resumen: El proyecto de tesis tiene como fin desarrollar un algoritmo memético en el lenguaje Java para realizar la optimización de consultas/queries SQL en el contexto de bases de datos relacionales distribuidas (múltiples sitios/servidores separados físicamente en los que tablas se encuentran distribuidas y/o replicadas).

El presente formulario tiene como objetivo determinar si el resultado esperado 2.3 (calibración de variables y parámetros del algoritmo memético) es válido bajo el punto de vista de un experto en la materia (en este caso, en algoritmia metaheurística).

*Obligatorio

Correo *

tupia.mf@pucp.edu.pe

Por favor, ingrese su nombre: *

MANUEL TUPIA ANTICONA

Por favor, confirme si se reunió con el tesista Jesús Sangama Ramírez para la explicación del resultado esperado en la siguiente fecha: 05/11/2021 *

- (●) Sí nos reunimos
- ( ) No nos reunimos
- ( ) Nos reunimos, pero en otra fecha

Si se reunió con el tesista en una fecha diferente a la especificada anteriormente, por favor, ingrese dicha fecha:

DD MM AAAA

/ /

¿Tiene usted comentarios u observaciones respecto al resultado entregado (dudas, puntos a mejorar, puntos faltantes, etc.)?

Ninguna, el resultado se realizó de manera correcta

¿A qué porcentaje diría usted que aprueba el resultado presentado? *

- ( ) 0% (no lo aprueba)
- ( ) 20%
- ( ) 40%
- ( ) 60%
- ( ) 80%
- (●) 100%

*Enviado: 5/11/21 21:43*

## Anexo O: Formulario de validación de resultado R3.3 llenado por experto en algoritmia

tupia.mf@pucp.edu.pe 1 de 1

No se pueden editar las respuestas

# Proyecto de Tesis 2 - Validación resultado esperado 3.3

Tema de tesis: Algoritmo metaheurístico para la optimización de consultas en bases de datos relacionales
Asesor: Rony Cueva Moscoso
Tesista: Jesús Angel Sangama Ramírez

Breve resumen: El proyecto de tesis tiene como fin desarrollar un algoritmo memético en el lenguaje Java para realizar la optimización de consultas/queries SQL en el contexto de bases de datos relacionales distribuidas (múltiples sitios/servidores separados físicamente en los que tablas se encuentran distribuidas y/o replicadas).

El presente formulario tiene como objetivo determinar si el resultado esperado 3.3 (experimentación numérica con resultados de ejecución de algoritmos memético y genético) es válido bajo el punto de vista de un experto en la materia (en este caso, en algoritmia).

*Obligatorio

Correo *

tupia.mf@pucp.edu.pe

Por favor, ingrese su nombre: *

MANUEL TUPIA ANTICONA

Por favor, confirme si se reunió con el tesista Jesús Sangama Ramírez para la explicación del resultado esperado en la siguiente fecha: 05/11/2021 *

- (●) Sí nos reunimos
- ( ) No nos reunimos
- ( ) Nos reunimos, pero en otra fecha

Si se reunió con el tesista en una fecha diferente a la especificada anteriormente, por favor, ingrese dicha fecha:

DD MM AAAA

__ / __ / ____

¿Tiene usted comentarios u observaciones respecto al resultado entregado (dudas, puntos a mejorar, puntos faltantes, etc.)?

Ninguna, el resultado mostrado es correcto

¿A qué porcentaje diría usted que aprueba el resultado presentado? *

- ( ) 0% (no lo aprueba)
- ( ) 20%
- ( ) 40%
- ( ) 60%
- ( ) 80%
- (●) 100%

Enviado: 5/11/21 21:44