# ALGORITMO META-HEURÍSTICO MONO-OBJETIVO GUIADO POR VELOCIDAD, VOLUMEN Y LONGITUD DE COLAS PARA CALIBRAR MODELOS DE MICRO-SIMULACIÓN DE FLUJO DE TRÁFICO VEHICULAR CORSIM

**Carlos Armando Daza Rendón**
**Cristhian Martínez Rendón**

Director: PhD. Carlos Alberto Cobos Lozada
Co-Director: PhD. Martha Eliana Mendoza Becerra
Co-Director externo: PhD. Alexander Paz (University of Nevada)
Asesores: Carlos Gaviria y Cristian Arteaga (University of Nevada)

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Sistemas
Línea de Investigación de Sistemas Inteligentes
Popayán, febrero de 2017

## TABLA DE CONTENIDO

## LISTA DE FIGURAS

## LISTA DE TABLAS

## LISTA DE ANEXOS

**Anexo 1:** Aplicación software, código fuente en Java con la implementación de los algoritmos meta-heurísticos, incluye ejecutable basado en la nueva versión de la herramienta Calibration Tool.

**Anexo 2:** Artículo “Calibration of Microscopic Traffic Flow Simulation Models Using a Memetic Algorithm with Solis and Wets Local Search Chaining (MA-SW-Chains)”.

**Anexo 3:** Artículo “Algoritmo meta-heurístico mono-objetivo de Coevolución Cooperativa basada en Evolución Diferencial (DECC-G) guiado por velocidad, volumen y longitud de colas para la calibración de modelos de micro-simulación de flujo de tráfico vehicular”.

**Anexo 4:** Manuales de CORSIM.

**Anexo 5:** Resultados obtenidos en la experimentación.

# Resumen

En este trabajo se adaptan tres algoritmos meta-heurísticos mono-objetivo para abordar la calibración de modelos de micro-simulación de flujo de tráfico vehicular CORSIM. Estos tres algoritmos meta-heurísticos son especializados en problemas de alta complejidad que cuentan con diferentes enfoques: Coevolución cooperativa basada en evolución diferencial (DECC-G), Muestreo múltiple de descendientes (MOS) y un Algoritmo memético basado en encadenamiento de búsquedas locales (MA-SW-Chains). Para comparar el desempeño de estos tres algoritmos, se definieron tres experimentos con diferentes niveles de complejidad proporcionados por el Departamento de Transportes de Nevada (NDOT por sus siglas en inglés). Para determinar la calidad de la calibración, se tiene en cuenta la estadística GEH (aprobada por la Federal Highway Administration) y la discrepancia entre las variables de velocidad, volumen y longitud de colas de los modelos calibrados y los datos tomados en campo.

# Capítulo 1

## 1 INTRODUCCIÓN

### 1.1 PLANTEAMIENTO DEL PROBLEMA

La simulación es definida en [1] como la experimentación sobre un modelo (imitación simplificada) de un sistema real en un computador a través del tiempo, con el propósito de mejorar la comprensión y entendimiento de dicho sistema. Dentro del campo de la simulación, se contempla la micro-simulación o también llamada simulación microscópica, la cual ha sido aplicada en la última década con mayor frecuencia en el campo del análisis de los sistemas de transporte [2], en donde el comportamiento dinámico de los agentes individuales se simula de forma explícita tanto en tiempo y espacio para generar el comportamiento del sistema completo.

Actualmente existen herramientas software que permiten la creación de modelos de micro-simulación de flujo de tráfico vehicular, estos modelos son comúnmente utilizados por los profesionales e investigadores para diferentes aplicaciones en ingeniería de tráfico, como es el caso del análisis detallado del rendimiento de los sistemas de tráfico, un ejemplo de este tipo de software es CORSIM (CORridor SIMulation).

CORSIM es una herramienta para la elaboración y análisis de modelos de simulación microscópica, que ha sido diseñada para el análisis de autopistas, calles urbanas, corredores o redes que rastrea la posición y el movimiento de cada vehículo en la red [3]. Los modelos CORSIM, como la mayoría de modelos de simulación microscópica, necesitan de parámetros, para determinar el comportamiento del modelo, estos parámetros son de gran importancia ya que su variación involucra modificaciones en los resultados del proceso de simulación. Por consiguiente, dichos parámetros deben ser ajustados con el fin de que los resultados que arroje el proceso de simulación sean lo más aproximado a los datos de campo, es decir, a los datos que describen el comportamiento del sistema en la vida real, los cuales son capturados mediante sensores o herramientas específicas ubicados en diferentes partes del sistema que se pretende estudiar. Para lograr este ajuste, es necesario un proceso de calibración de los parámetros del modelo de micro-simulación de flujo de tráfico vehicular CORSIM.

En [4] se muestran diferentes metodologías que permiten identificar un conjunto de principios generales que se pueden tener en cuenta para realizar el proceso de calibración de un modelo de micro-simulación de tráfico. Estos principios son los siguientes: 1) Pre-calibración, 2) Selección de parámetros a calibrar y definición del rango posible de cada parámetro, 3) Selección de la medida de bondad de ajuste,

4) Especificar el procedimiento de calibración, y 5) Ejecución de la calibración, validación y análisis de resultados.

Para el procedimiento de calibración (etapa 4) se ha usado la optimización a nivel local o global como una estrategia factible y válida. En la optimización local se toma una pequeña región del espacio de búsqueda y se realiza el proceso de mejora hasta encontrar un conjunto de parámetros que al evaluarlos en la función objetivo obtiene la mejor solución (óptimo local) en dicha región. Por otra parte, en la optimización global, se busca en todo el espacio de búsqueda con el objetivo de encontrar los valores de los parámetros $x = \{x_1, x_2, \dots, x_D\}$, donde D es la dimensionalidad del problema abordado, tal que el valor de la evaluación de la función f(x) es óptimo, lo cual significa que x contiene el conjunto de parámetros con mejor valor de evaluación de la función que cualquier otro ajuste de parámetros en todo el espacio de búsqueda. Cuando la dimensionalidad D de un problema en particular es superior a cien dimensiones se considera un problema de optimización a gran escala [5] y su complejidad crece exponencialmente.

La optimización local o global puede ser aplicada mediante múltiples técnicas, una de las más usadas en la literatura como herramienta de calibración en diferentes campos de investigación son los algoritmos genéticos [4] [6] [7] [8] [9], aunque en [10] muestran que el algoritmo SPSA (Simultaneous Perturbation Stochastic Approximation) obtiene mejores resultados que los algoritmos genéticos en cuanto al tiempo de calibración y además consume menor número de recursos computacionales.

Otros trabajos han usado metodologías basadas en SPSA para calibrar modelos de simulación de flujo de tráfico microscópico, independiente de las características de cualquier modelo particular [11] y el uso de algoritmos meméticos (GASA = algoritmo genético + recocido simulado) [12] para calibrar modelos de micro-simulación de flujo de tráfico vehicular CORSIM teniendo en cuenta las variables de velocidad y volumen, obteniendo mejores resultados que los reportados por SPSA en [11].

Los trabajos previos en calibración de modelos de micro-simulación de flujo de tráfico vehicular CORSIM no han contemplado dos situaciones que son importantes en el proceso de optimización, a saber: 1) las meta-heurísticas usadas no son especializadas para resolver problemas de alta dimensionalidad, hecho que se refleja en [12] donde los resultados del algoritmo memético GASA y SPSA son buenos en problemas con menos de cien parámetros pero que se pueden mejorar considerablemente en problemas de mayor dimensionalidad, y 2) la función objetivo no contempla otros factores o componentes relacionados con los parámetros a calibrar, por ejemplo, las longitudes de las colas, considerada como la más relevante en el proceso de calibración [13] [14] [15].

Teniendo en cuenta lo anterior, en el presente trabajo se propuso usar el enfoque metodológico propuesto en [11], adaptando tres meta-heurísticas de optimización

continua especializadas en problemas de alta complejidad y definir una función de aptitud que oriente el proceso de optimización basada en la propuesta de [12], que contempla los componentes de volumen y velocidad, pero incorporando un nuevo componente relacionado con la longitud de colas. Para esta última, la medida utilizada fue NRMS (Normalized Root Mean Squared) y la estadística GEH que son constantemente usadas en procesos de calibración de modelos de micro-simulación de flujo vehicular y de hecho son medidas que recomiendan en [4], para utilizar en este proceso.

Para la selección de las tres meta-heurísticas de optimización continua especializadas en problemas de alta complejidad, se analizaron los resultados presentados en la competencia de la IEEE CEC realizada en Sendai-Japón en el 2015 [16]. En dicha competencia se seleccionan los mejores algoritmos para resolver diversos tipos de problemas de optimización continua, problemas representados con funciones unimodales, multimodales, separables y no separables. Los mejores resultados en las diferentes categorías incluyen a MOS, MA-SW-Chains, DECC-G, entre otros.

Por otro lado, el No-Free-Lunch Theorem para optimización establece que para un problema específico no se conoce cuál de las diferentes meta-heurísticas puede resolverlo de la mejor manera [17]. En este sentido, se evaluaron los algoritmos que presentan los mejores reportes a la fecha, para encontrar cuál de estos presenta un mejor resultado para el problema de calibración de un modelo de micro-simulación de tráfico, estos algoritmos son: 1) Muestreo múltiple de descendientes en optimización global a gran escala (MOS) [18], 2) Coevolución cooperativa basada en evolución diferencial para optimización a gran escala (DECC-G) [19] y 3) Algoritmo memético basado en encadenamiento de búsquedas locales para optimización continua global a gran escala (MA-SW-Chains) [20].

Teniendo en cuenta lo anterior, se planteó la siguiente pregunta de investigación: ¿Es posible obtener un algoritmo de optimización para calibrar modelos de simulación microscópica de flujo de tráfico vehicular CORSIM, basado en uno de los mejores algoritmos reportados en optimización continua de alta dimensionalidad (MOS, DECC-G y MA-SW-Chains), que sea más eficiente (tiempo de ejecución) y encuentre mejores resultados que los reportados en trabajos previos (GASA y SPSA) y cuente en su función objetivo con los componentes de volumen, velocidad y longitud de colas?.

## 1.2 APORTES DEL PROYECTO

Desde la perspectiva de investigación las contribuciones de este proyecto van encaminadas a generar nuevo conocimiento (exploratorio y descriptivo), dado que el algoritmo meta-heurístico DECC-G muestra los mejores resultados de calibración de modelos de tráfico vehicular CORSIM en comparación con los algoritmos meta-heurísticas MOS y MA-SW-Chains, además que MOS realiza mejor el proceso de calibración que MA-SW-Chains. Con respecto a los algoritmos del estado del arte

(SPSA y GASA), el algoritmo DECC-G es más eficiente en tiempo de ejecución y encuentra mejores resultados en las tres complejidades de los modelos CORSIM con los que se hicieron las pruebas. En comparación con la función objetivo usada en [12] y la propuesta en este trabajo, la segunda función objetivo presenta mejores resultados de calibración, mirando estos resultados como porcentajes de mejora en la calibración. Este conocimiento no se encontró reportado en ninguna de las bases de datos bibliográficas consultadas, a saber, IEEE, ScienceDirect, ACM y Springer Link.

En la literatura no existe reporte acerca de la implementación de los tres algoritmos meta-heurísticos (DECC-G, MOS y MA-SW-Chains) para el problema de calibración de modelos de tráfico vehicular, aspecto que hace innovador este proyecto.

El proceso de calibración se realizó sobre modelos de micro-simulación de flujo de tráfico vehicular CORSIM de Estados Unidos. Permitiendo a futuro que los algoritmos implementados se puedan usar para resolver problemas similares en diferentes partes del mundo.

## 1.3 OBJETIVOS

A continuación, se describen los objetivos del proyecto, conforme fueron aprobados por el Consejo de la Facultad de Ingeniería Electrónica y Telecomunicaciones de la Universidad del Cauca en el documento del anteproyecto.

### 1.3.1 Objetivo General

Definir un algoritmo meta-heurístico mono-objetivo guiado por velocidad, volumen y longitud de colas para calibrar modelos de micro-simulación de flujo tráfico vehicular CORSIM.

### 1.3.2 Objetivos Específicos

- Adaptar los algoritmos MA-SW-Chains [20], DECC-G [19] y MOS [18] para soportar el proceso de calibración de modelos de micro-simulación de flujo de tráfico vehicular CORSIM, usando los componentes (velocidad y volumen) de la función objetivo propuesta en [12].
- Definir una nueva función objetivo que contemple la función objetivo propuesta por Paz et al e incluya un componente relacionado con la longitud de las colas reportadas en los modelos de micro-simulación.
- Evaluar la calidad de la calibración de modelos de micro-simulación de flujo de tráfico vehicular realizada por los algoritmos propuestos con la nueva función objetivo utilizando las directrices FHWA [21], y finalmente comparar los resultados obtenidos en tiempo y mejores soluciones encontradas frente a los algoritmos SPSA y GASA.

## 1.4 RESULTADOS OBTENIDOS

A continuación, se resumen los principales resultados del presente trabajo de grado:

- Monografía del trabajo de grado, corresponde al presente documento que contiene el estado del arte sobre el problema junto con la descripción detallada de los algoritmos usados en este trabajo de grado. Luego presenta la adaptación de los algoritmos al proceso de calibración y la función objetivo propuesta. Después presenta el análisis de los resultados obtenidos por los algoritmos propuestos junto con la comparación con los algoritmos del estado del arte.
- Aplicación software con la implementación de los algoritmos en Java y su correspondiente documentación. Esta aplicación es una nueva versión de la herramienta Calibration Tool.
- Artículo presentado en evento internacional y publicado en revista internacional indexada por SJR y reconocida por COLCIENCIAS con categoría B en el PUBLINDEX. El artículo tiene la siguiente cita: “Calibration of Microscopic Traffic Flow Simulation Models using a Memetic Algorithm with Solis and Wets Local Search Chaining (MA-SW-Chains)” [22]. Proceedings of 15th edition of the Ibero-American Conference on Artificial Intelligence, IBERAMIA 2016. M. Montes y Gómez et al. (Eds.) Springer International Publishing. pp. 365-375. Series: Lecture Notes in Computer Science (LNCS), Subseries: Lecture Notes in Artificial Intelligence (LNAI). San Jose, Costa Rica, November 23-25, 2016, vol. 10022. ISSN: 0302-9743 (Print) 1611-3349 (Online). Available online at http://dx.doi.org/10.1007/978-3-319-47955-2_30
- Artículo “Algoritmo meta-heurístico mono-objetivo de Coevolución Cooperativa basada en Evolución Diferencial (DECC-G) guiado por velocidad, volumen y longitud de colas para la calibración de modelos de micro-simulación de flujo de tráfico vehicular” que resume toda la investigación y los resultados obtenidos, el cuál será enviado a revista internacional indexada JCR (ISI) a comienzos del año 2017.

# Capítulo 2

## 2 CONTEXTO TEÓRICO Y ESTADO DEL ARTE

### 2.1 CONTEXTO TEÓRICO

En esta sección se presentan los conceptos más relevantes para el desarrollo de este trabajo de grado, dando de esta forma mayor claridad en la lectura de este documento. Además, estos conceptos van acompañados de referencias bibliográficas para un mayor detalle de los mismos.

#### 2.1.1 Micro-simulación

La micro-simulación es una familia de simulaciones por computador que dependen de datos a nivel individual como entrada a uno o varios procesos que agrupan a los resultados a nivel macro. Una de las aplicaciones más populares de la micro-simulación son los sistemas de tráfico de automóviles, donde los registros individuales son vehículos [23].

En la simulación microscópica o micro-simulación [2] de tráfico, por cada vehículo que se encuentre en el modelo en cuestión, se realiza un seguimiento de su trayectoria, donde el movimiento de un vehículo cualquiera en el sistema, es determinado por las características del conductor, rendimiento del vehículo, sus interacciones con las avenidas, calles de la red y los vehículos que se encuentren a su alrededor. La simulación microscópica de tráfico es ampliamente utilizada para realizar análisis de las operaciones de tráfico, evaluar los méritos de las nuevas tecnologías de control y gestión de tráfico. La esencia de una simulación microscópica de tráfico es la lógica del seguimiento de los coches con sus comportamientos de cambio de carril [24].

Los modelos de simulación microscópica representan movimientos de vehículos individuales, que incluyen las influencias del comportamiento del conductor. Con estos modelos se pueden estudiar los efectos de estrategias muy detallados, tales como la reubicación de estaciones de autobuses o cambio de las restricciones de estacionamiento [21].

#### 2.1.2 CORSIM (CORridor SIMulation)

El desarrollo inicial de la lógica CORSIM inició a principios de 1970, bajo el patrocinio de la FHWA (Federal Highway Administration), CORSIM ha sido objeto de numerosas mejoras y actualizaciones importantes, tanto en la lógica de simulación como en ingeniería de software [24]. CORSIM es el componente básico de simulación y modelado de la herramienta TSIS (Traffic Software Integrated System). CORSIM consiste en un conjunto integrado de dos modelos de simulación microscópica (NETSIM y FRESIM) que representan el entorno de tráfico. NETSIM

representa el tráfico de las vías urbanas y FRESIM representa el tráfico de las autopistas o avenidas.

CORSIM aplica la simulación discreta con un paso de tiempo constante (un segundo) para describir las operaciones de tráfico. Cada vehículo es un objeto distinto que se mueve cada segundo. Cada variable de control (por ejemplo, señales de tráfico) y cada evento se actualizan cada segundo. CORSIM es un modelo estocástico, lo que significa que los números aleatorios se asignan a las características del conductor y del vehículo para la toma de decisiones. Las medidas de eficacia (estiman el rendimiento de la red) que se obtienen a partir de una simulación, son el resultado de un conjunto específico de semillas de números aleatorios. Para obtener una mejor comprensión del funcionamiento, la red debe ser simulada varias veces usando diferentes conjuntos de semillas de números aleatorios, esto genera una representación exacta del rendimiento de la red.

CORSIM es capaz de simular la mayoría de las geometrías de la autopista que sobresalen, incluyendo líneas principales de varios carriles de la autopista, conectores a otras autopistas, el radio de curvatura y elevación, adiciones de calzadas y carriles auxiliares, los cuales se utilizan por el tráfico para comenzar o terminar el proceso de cambio de carril o para entrar o salir de la autopista. Las características que cambian con el tiempo, tales como tiempos de señal y los volúmenes de tráfico, son representados por la división de la simulación en una secuencia de periodos de tiempo especificados por el usuario, durante la cual los flujos de tráfico, los controles de tráfico y la geometría se mantienen constantes [21].

En cuanto a su funcionamiento, CORSIM trabaja con un enfoque de caja negra, donde su entrada son archivos de extensión *.trf* y su salida son archivos de extensión *.out*. Por ser un software propietario y con enfoque de caja negra, se desconoce los procesos internos que ejecuta al realizar una simulación. Los archivos de entrada contienen la información del modelo y está compuesta por grupos de parámetros (denominados *record type*). Estos "*record type*" brindan la información necesaria para representar diversos comportamientos a través del modelo y su simulación. Los archivos de salida contienen diversos grupos de medida formadas a través de la suma de los comportamientos de los vehículos en el modelo, dichas medidas son obtenidas de las diferentes intersecciones del modelo, estas intersecciones se llaman nodos, los cuales a su vez se forman a partir de la unión de dos vías (denominadas *links*). La información contenida en estos archivos se encuentra sobre diferentes tablas que muestran a manera de resumen las estadísticas obtenidas de la simulación sobre diferentes nodos.

Los *records types* (tipos de registros) a su vez se conforman de un conjunto de datos que se denominan *entries* (entradas) que representan diversas formas en las que un comportamiento se puede exhibir sobre un enlace (*link*). Los modelos CORSIM son archivos planos de extensión *.trf*, en donde cada línea del archivo es un *record type* específico con sus respectivas entradas tal como se ilustra en la **Figura 1**.

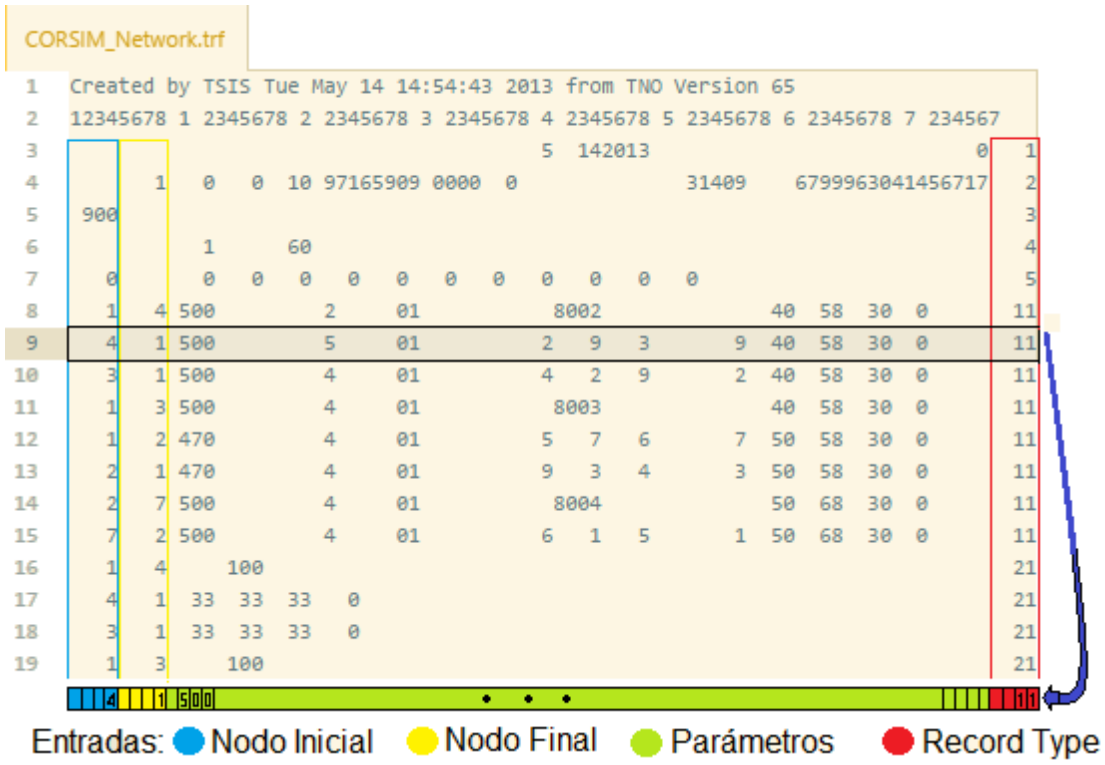



***Figura 1:*** *Estructura de un modelo CORSIM*

Cada registro o fila en el modelo CORSIM consta de un total de 80 columnas, de las cuales, la columna uno a la cuatro representa el nodo inicial, la columna cinco a la ocho el nodo final, entre las columnas 9 y 77 se indican los parámetros o características de cada *record type* para el enlace formado entre el nodo inicial y final, las últimas columnas (78, 79 y 80) indican el *record type*. En la **Figura 1** se ilustra en el recuadro rojo los diferentes *record type* que contiene el modelo, entre ellos el *record type* 11 que contiene la información sobre las características de una calle y el *record type* 21 que contiene la información sobre el volumen de vehículos que giran en una intersección. Esta información para cada *record type* se almacena en las diferentes entradas de cada uno de ellos (franja verde de la **Figura 1**).

Para el proceso de calibración se debe seleccionar las *entries* y los *records types* que se desean calibrar para cada uno de los enlaces que conforman el modelo CORSIM. El número de enlaces y las *entries* seleccionadas, determinan la dimensionalidad del problema. En [21] definen el conjunto de parámetros o entradas calibrables que son las referentes con el comportamiento del conductor y rendimiento.

Adicionalmente, CORSIM cuenta con un módulo importante de gráficas y animaciones conocido como TRAFVU que permite visualizar la información del sistema de tráfico que se desea modelar y obtener una mayor comprensión de cómo funciona la red. En la **Figura 2**, se ilustra los diferentes nodos y enlaces que conforman un modelo representado en el módulo TRAFVU.

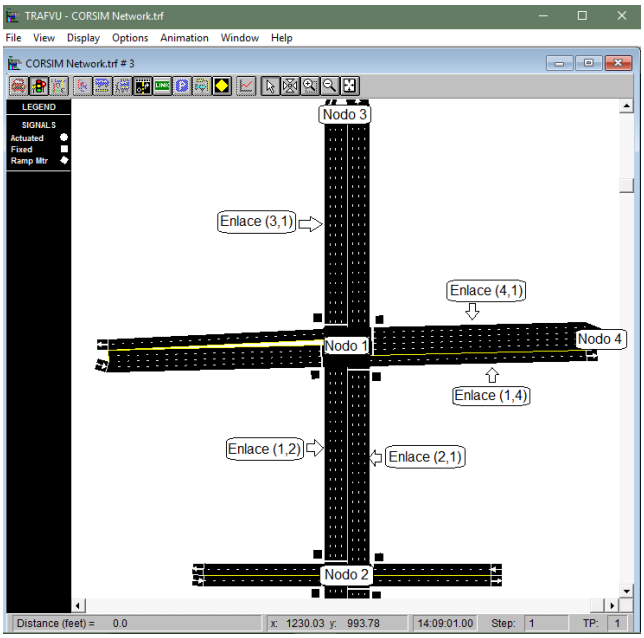



***Figura 2:*** *Simulación mediante TRAFVU*

### 2.1.3 Proceso de calibración

Según [4] en el proceso de calibración de un modelo de micro-simulación de tráfico vehicular CORSIM se identifica un conjunto de principios generales que se deben tener en cuenta en las siguientes etapas:

- **Pre-calibración:** se entiende el comportamiento del modelo y se determina el alcance del problema, es decir, posibilidad de descomponer el problema de calibración en sub-problemas independientes.
- **Selección de parámetros a calibrar y definición del rango posible de cada parámetro:** en el manual de referencia de Mctrans [25] se detallan cada uno de los parámetros que constituyen un modelo de micro-simulación de flujo de tráfico vehicular CORSIM, los cuales se agrupan en diversos "*record type*" y cada uno de ellos puede estar sujeto a dos tipos de restricciones. La primera es que los valores de cada parámetro están entre un valor mínimo y máximo. La segunda es que los valores de los parámetros de un mismo *record type* deben conservar coherencia. Las restricciones para cada *record type* se encuentran en el manual de referencia CORSIM [21].
- **Selección de la medida de bondad de ajuste:** esta medida es utilizada para medir la diferencia entre los resultados de la simulación y los datos de campo, es decir, los datos obtenidos por sensores sobre la vía (para esta investigación, longitud de colas, volumen y velocidad de los vehículos) del mismo lugar representado por la simulación.
- **Especificar el procedimiento de calibración:** una calibración manual es viable para un pequeño número de parámetros, es decir, no mayor a cinco. Pero dada la naturaleza de los modelos de micro-simulación de flujo vehicular CORSIM, se

requiere de un proceso automatizado para encontrar el mejor conjunto de parámetros que necesita la simulación para obtener como salida los datos más aproximados a los datos de campo en un espacio de posibles soluciones, dado por el rango que los diferentes parámetros pueden tomar. Un ejemplo de dicho proceso automatizado se conoce como la técnica de optimización.

- **Ejecución de la calibración, validación y análisis de resultados:** finalmente se determina el número de ejecuciones requeridas para encontrar una buena solución en el espacio de búsqueda, la ejecución de la calibración y se hace la respectiva validación estadística donde se debe demostrar un buen ajuste utilizando datos diferentes a los utilizados en el proceso de calibración.

### 2.1.4 Algoritmo meta-heurístico

Recientemente, el principal enfoque para resolver problemas de optimización han sido los algoritmos meta-heurísticos. Una de las principales razones para que este tipo de algoritmos tengan gran popularidad y éxito, es que han sido desarrollados imitando el proceso más exitoso de la naturaleza, incluyendo sistemas biológicos, físicos y químicos.

El notable rendimiento de los algoritmos meta-heurísticos es debido a como ellos imitan las mejores características de la naturaleza, donde la intensificación y diversificación son sus dos características fundamentales. La fase de intensificación, también llamada fase de explotación, busca las mejores soluciones actuales y selecciona el mejor o mejores soluciones candidatas, mientras que la fase de diversificación, también llamada fase de exploración, asegura que el algoritmo explore todo el espacio de búsqueda más eficientemente. La eficiencia general de un algoritmo meta-heurístico se debe a un buen balance entre estos dos componentes ya que el sistema puede quedar atrapado en un óptimo local si hay muy poca exploración, o si hay exceso de explotación, en este caso, sería muy difícil o incluso imposible encontrar el óptimo global. Por otro lado, si hay mucha exploración, pero poca explotación, puede ser difícil que el sistema converja, en este caso, desacelera el rendimiento de la búsqueda en general. Para obtener una mejor optimización al problema que se está solucionando, es necesario un equilibrio entre la explotación y la exploración. Durante la búsqueda, un mecanismo o criterio apropiado debe ser considerado para seleccionar las mejores soluciones. Actualmente, la “supervivencia del más apto” es uno de los criterios más comunes, que se basa en repetir varias veces la actualización de la mejor solución actual encontrada hasta un momento especifico del proceso de búsqueda.

Cada algoritmo meta-heurístico y sus variantes utilizan diferentes caminos para obtener el balance apropiado entre exploración y explotación. Cierta aleatorización en combinación con un procedimiento determinístico puede ser considerada un camino eficiente para lograr una buena exploración. Esto asegura que las soluciones generadas se distribuyan lo más diversamente posible en el espacio de búsqueda para que sea más viable encontrar el óptimo global [26].

### 2.1.5 DECC-G

La Coevolución Cooperativa basada en Evolución Diferencial (DECC-G) [19] es un marco de coevolución cooperativa que es capaz de optimizar problemas no separables a gran escala. DECC-G está diseñado para cambiar las estructuras de agrupación dinámicamente (agrupación aleatoria), lo que aumenta la posibilidad de la optimización de variables que interactúan entre sí, utilizando una estrategia de ponderación adaptativa, aplicando un peso a cada uno de los subcomponentes. Esta ponderación adaptativa proporciona coadaptación de los subcomponentes cuando son interdependientes.

DECC-G, utiliza para la optimización de los diferentes subcomponentes SaNSDE (Evolución Diferencial con Búsqueda Auto-adaptativa de Vecindario) y para la optimización de la ponderación adaptativa DE (Evolución Diferencial).

#### 2.1.5.1 Coevolución Cooperativa

La coevolución cooperativa [27] se aplica a problemas grandes y complejos utilizando la estrategia de divide y vencerás, descomponiendo un problema grande en subcomponentes más pequeños donde la interdependencia entre los diferentes subcomponentes sea lo más mínima posible. El objetivo es descomponer un problema de alta dimensionalidad en varios subcomponentes de baja dimensionalidad, evolucionando estos subcomponentes cooperativamente durante un número específico de ciclos, la cooperación se produce sólo durante la evaluación de la función objetivo.

La coevolución cooperativa para la optimización se resume de la siguiente manera:

- **Descomposición del problema:** descomponer el vector objetivo de alta dimensionalidad en subcomponentes más pequeños.
- **Optimización de los subcomponentes:** evolucionar cada subcomponente de forma independiente con algún Algoritmo Evolutivo.
- **Coadaptación de subcomponentes:** este paso tiene relación con las interdependencias entre los subcomponentes, capturándolas durante todo el proceso de optimización.

#### 2.1.5.2 Evolución Diferencial

Evolución Diferencial (DE) [28] es un método de búsqueda directa en paralelo que utiliza vectores como una población de cada generación. La población inicial se elige de forma aleatoria buscando cubrir todo el espacio de búsqueda. La Evolución Diferencial genera nuevos vectores mediante la adición de la diferencia ponderada entre dos vectores de la población, a este resultado se le suma un tercer vector, esta operación se llama mutación.

El vector mutado se mezcla con los valores de otro vector predeterminado (vector objetivo), para originar un vector de prueba, esta operación se llama cruce. Si el vector de prueba tiene un mejor valor de aptitud que el vector objetivo, el vector de

prueba sustituye al vector objetivo en la siguiente generación, esta última operación se llama selección.

#### 2.1.5.3 Evolución Diferencial con Búsqueda Auto-adaptativa de Vecindario

La Evolución Diferencial con Búsqueda Auto-adaptativa de Vecindario (SaNSDE) [29] es una mejora a la DE tradicional utilizando la estrategia de búsqueda de vecindario. Dicha mejora consiste en realizar una auto-adaptación de los parámetros Factor de Escala y Tasa de Cruce del algoritmo original. Para esto, SaNSDE adopta una auto-adaptación entre las distintas estrategias de mutación, ya que estas estrategias en DE dependen considerablemente del problema que se está solucionando. La auto-adaptación de la Tasa de Cruce la realiza mediante una ponderación y la del Factor de Escala con la distribución Normal o Gaussiana y la distribución de Cauchy.

### 2.1.6 MOS

MOS (Muestreo Múltiple de Descendientes) [18] es un algoritmo híbrido que combina las heurísticas Solis and Wets [30] y MTS-LS1 (Búsqueda de Múltiple Trayectoria - Búsqueda Local 1) [31] mostrando buenos resultados para problemas de optimización continua. El algoritmo MOS permite ajustar el número de evaluaciones de la función objetivo de forma dinámica y ejecuta las dos heurísticas en secuencia.

El algoritmo MOS tiene un enfoque especial, ya que no hace uso de algún algoritmo poblacional para realizar una exploración del espacio de búsqueda, sino que emplea dos algoritmos de búsqueda local con el fin de centrar la búsqueda en un pequeño número de soluciones candidatas (preferiblemente una solución), realizando diversas modificaciones a dichas soluciones. Estos dos algoritmos de búsqueda local son desacoplados del algoritmo MOS, para generar nuevas soluciones de forma independiente. De acuerdo al rendimiento que van teniendo las heurísticas Solis and Wets y MTS-LS1, MOS va asignando dinámicamente el porcentaje de participación para cada heurística. Referirse a [18] y [32], para encontrar más detalles de este algoritmo meta-heurístico.

#### 2.1.6.1 Algoritmo Solis and Wets

El algoritmo Solis and Wets es un Hill-Climber (ascenso de colina) aleatorio con un tamaño de paso adaptativo de acuerdo al número de éxitos o de fallos que se tenga cuando se realiza la búsqueda. En cada paso en una solución actual $\vec{x}$, se elige una desviación $\vec{d}$ de acuerdo a la distribución Normal o Gaussiana cuya desviación estándar está dada por un parámetro configurado por el usuario y su media es igual a cero.

Si $\vec{x} + \vec{d}$ o $\vec{x} - \vec{d}$ da como resultado una mejor solución, se realiza un movimiento hacia ésta solución y se registra un éxito, de lo contrario se registra un fallo y no se realiza ningún movimiento. Después de registrar varios éxitos consecutivos, la desviación estándar de la distribución Normal o Gaussiana se incrementa para

moverse más rápido durante la búsqueda. Si se registra varios fracasos consecutivos, la desviación estándar de la distribución Normal o Gaussiana se reducen para centrar la búsqueda.

Además, para encontrar mejores soluciones, se incluye un término $\overrightarrow{bias}$ para tener un impulso correcto de búsqueda en direcciones de éxito. Este es un método de búsqueda local simple que puede adaptar su tamaño de paso rápidamente [33]. Referirse a [30], para encontrar más detalles de este algoritmo.

#### 2.1.6.2 MTS-LS1

MTS (Búsqueda de Múltiple Trayectoria) utiliza múltiples individuos al mismo tiempo para encontrar la solución en el espacio de búsqueda. Cada individuo hace una búsqueda local iterativa utilizando uno de los tres métodos de búsqueda local que tiene MTS. Al elegir un método de búsqueda local que mejor se adapte a la zona de la solución, un individuo puede encontrar su camino hacia el óptimo global.

La búsqueda local 1 de MTS (MTS-LS1), recorre las dimensiones del individuo de manera secuencial para realizar modificaciones a cada dimensión de acuerdo al intervalo de búsqueda para encontrar mejores soluciones. El intervalo de búsqueda se define como la mitad del rango de cada dimensión. Referirse a [31] para encontrar más detalles de este algoritmo.

### 2.1.7 MA-SW-Chains

MA-SW-Chains es un algoritmo memético para la optimización continua a gran escala [20] y una adaptación del algoritmo MA-CMA-Chains [33], con el fin de mejorar el rendimiento en problemas de alta dimensionalidad. El principal enfoque de este algoritmo es encadenar diferentes búsquedas locales al mejor individuo de cada generación.

MA-SW-Chains utiliza un algoritmo genético de estado estacionario para explotar el espacio de búsqueda de soluciones. En este algoritmo genético de estado estacionario, uno o dos hijos son creados en cada generación y compiten con los individuos de la generación actual [34] para ser parte de la nueva población, esto hace que un individuo pueda sobrevivir a lo largo del proceso evolutivo durante varias generaciones.

MA-SW-Chains utiliza el algoritmo Solis and Wets [30] (Sección 2.1.6.1) para realizar las búsquedas locales encadenadas al mejor individuo de cada generación (explotación). A lo largo de la evolución se presentan este tipo de búsquedas locales con el fin de explotar con mayor intensidad las zonas más prometedoras localizadas por el algoritmo genético.

#### 2.1.7.1 Búsquedas locales encadenadas

En [35] definen las búsquedas locales encadenadas así: al finalizar la primera búsqueda local, el individuo termina con unos valores en sus componentes tales como genes, vector $\overrightarrow{bias}$ y desviación Gaussiana $\vec{d}$, los cuales han sido modificados

durante el proceso de búsqueda. Estos valores finales de la primera búsqueda, se utilizan como valores iniciales de la siguiente búsqueda local y así sucesivamente. De esta manera, se puede aplicar las búsquedas locales que sean necesarias, estableciendo una conexión sin interrupción, formando las búsquedas locales encadenadas. En la **Figura 3** se ilustra un ejemplo de búsqueda local encadenada.

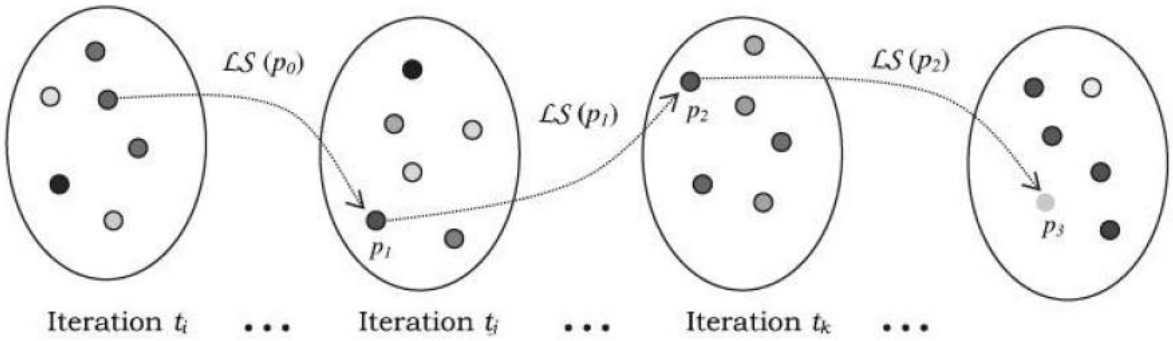



***Figura 3:*** *Búsqueda local encadenada, tomada de [35].*

$p_{i+1}$ (sucesor de $p_i$) son los valores de los componentes (genes, $\overrightarrow{bias}$, $\vec{d}$) cuando termina la búsqueda local, empezando con los valores de $p_i$, $p_0$ son los valores por defecto para iniciar la primera búsqueda local. Cuando se aplica una sola búsqueda local a $p_0$ con todos los componentes que equivalen a las $n$ optimizaciones locales, se debe obtener el mismo resultado que cuando se realiza $n$ búsquedas locales encadenadas.

## 2.2 ESTADO DEL ARTE

En la revisión de la literatura, se encuentra que los algoritmos genéticos y el algoritmo SPSA, son las técnicas más utilizadas recientemente para la calibración de modelos de tráfico vehicular, aunque también se reporta el uso del algoritmo de secuencia simple, algoritmos meméticos, métodos de ensayo y error, entre otros. A continuación, se describen los artículos de mayor relevancia, encontrados en la revisión de la literatura.

En [36] (2003) utilizan una metodología que se basa en el algoritmo de secuencia simple para calibrar modelos de micro-simulación CORSIM y TRANSIM de la ciudad de Houston, Texas. Para realizar la calibración utilizan la medida MAER (Mean Absolute Error Ratio), logrando buenos resultados en los modelos de regiones congestionadas, aunque el costo computacional es muy alto.

En [37] (2007) presentan un método de optimización estocástica basada en algoritmos genéticos para la optimización de señales de tráfico, es decir, el intercambio de derechos de vía entre el tráfico vehicular y peatonal. Los resultados de la simulación CORSIM muestran que la optimización estocástica producida a base del algoritmo genético proporciona mejores resultados en comparación con Synchro, un software de optimización de la coordinación de semáforos.

Luego en 2008 [7], se propone una metodología basada en el muestreo bayesiano para generar aleatoriamente matrices de demanda y condiciones de tráfico a partir de una distribución estadística observada en las variables. En cada iteración, la metodología de calibración estima los parámetros óptimos mediante el uso del algoritmo Enhanced Simultaneous Perturbation Stochastic Approximation (E-SPSA). El análisis bayesiano se aplica para representar la distribución de las características de tráfico. Con esta metodología, se calibró un modelo de simulación basado en la transmisión de célula de una porción de la autopista I-880 en California.

Después en 2009, en [38] presentan un desarrollo de un marco de sistema de control de señales de tránsito y control de velocidad que se representa en un modelo de simulación microscópica, un modelo de consumo de combustible y emisiones. Esta optimización se realiza mediante un algoritmo genético basado en optimización estocástica, reduciendo la congestión, el consumo del combustible y las emisiones, además de mejorar la movilidad de los carros reduciendo el tiempo total de las colas.

Este mismo año en [39] utilizan un algoritmo genético para encontrar los mejores valores para los parámetros de calibración de un modelo CORSIM y modelos de integración utilizando los datos de rendimiento empíricos recogidos en las carreteras. Para realizar la calibración usan la medida MAER. Con esta calibración se encontraron nuevos parámetros para los camiones a partir de datos empíricos y parámetros para representar el rendimiento promedio de los camiones.

Luego en 2010 [13] se muestra y explica el proceso de calibración de TSIS en una arteria de la ciudad de Santiago de Chile. Cuando definen las variables de interés para el proceso de calibración, consideran que la variable más relevante es la longitud máxima de cola (variable a considerar en este trabajo), es decir, la longitud de cola al inicio del verde o final del rojo del semáforo. Uno de los indicadores de calibración donde están involucradas las colas es el Indicador de Desempeño (ID). El proceso de calibración es realizado por el método de ensayo y error, usando la medida MANE (Mean Absolute Normalized Error), mejorando en más de una tercera parte la representación de la realidad en la arteria de la ciudad de Santiago de Chile.

En este mismo año [40] también se propone un nuevo enfoque para estimar conjuntamente los parámetros de los modelos de seguimiento de carros considerando múltiples fuentes de datos, con la finalidad de optimizar los resultados de la estimación cuando la información en las trayectorias individuales es limitado, esto permite un análisis estadístico de las estimaciones de los parámetros incluyendo el error estándar y la correlación de las estimaciones. Realizan comparación cruzada de los modelos de diferente complejidad mediante una prueba de razón de verosimidad. El enfoque es aplicado con éxito para estimar los parámetros del modelo GHR (Gazis, Herman y Rothery, uno de los modelos de coches de seguimiento más simples propuestos en la literatura) y de IDM (Modelo de Conducción Inteligente).

Posteriormente en 2012 se presentan dos trabajos, en el primero [41], proponen un modelo de simulación de tráfico microscópico basado en autómatas celulares, que puede ser acelerado. Para calibrar los parámetros del modelo de optimización utilizan un algoritmo genético, obteniendo como resultado todos los parámetros del modelo, aunque esta calibración se realiza para un modelo de simulación de tráfico microscópico simple con una cantidad limitada de datos reales. En el segundo [42] utilizan un algoritmo genético para calibrar un modelo de micro-simulación en Alemania y Reino Unido, la calibración se realiza para un subconjunto seleccionado de parámetros de entrada y salida que se utilizan en el proceso de calibración, para los cuales los datos regionales no estaban disponibles. Los resultados obtenidos a partir del modelo son muy parecidos a los datos reales, aunque en este trabajo no se centraron en encontrar mejores configuraciones a los operadores genéticos con el fin de obtener mejores resultados.

Para el 2013, en [43] señalan que las emisiones del tráfico en carreteras se pueden reducir mediante la optimización de control de tráfico realizando calibración. En el mayor de los casos, la calibración de los modelos de simulación se realiza utilizando volúmenes, tiempos de viajes y colas, pero para el objetivo de la calibración se utilizaron las características de salida que son críticos para la emisión: velocidad y aceleración. Utilizando un modelo del micro-simulador de trafico VISSIM, calibran los parámetros de comportamiento del conductor usando trayectorias de los vehículos. Una desventaja del método aplicado es que en ocasiones los vehículos que se detienen en distancias cortas generan una subestimación en el estado de velocidad de los vehículos. Mediante un estudio de sensibilización de parámetros, determinaron cuales son los parámetros más influyentes para garantizar que los resultados de la simulación fueron consistentes con el tráfico observado. El inconveniente de este método es que la interacción entre los parámetros sólo se puede identificar mediante la aplicación de este procedimiento para diferentes combinaciones de parámetros. Como alternativa a este problema sugirieron utilizar un algoritmo genético.

El mismo año en [44] utilizan un algoritmo genético para la calibración simultánea de todos los parámetros de entrada para un modelo de asignación dinámica de tráfico, estimando flujos de origen/destino y calibrando el comportamiento de los conductores. Esta metodología fue probada en una red compleja a gran escala de Toronto (Canadá), utilizando la medida NRMS y la estadística GEH (contempladas en este trabajo) para realizar la calibración (para acelerar este proceso aplican programación en paralelo), minimizando la discrepancia de tráfico observado y simulado. Para mejorar la precisión en la calibración y la eficiencia computacional, incluyen datos de velocidad (variable a considerar en este trabajo).

Además, en [45] estudian diferentes modelos computacionales que simulan procesos reales donde es preciso especificar los parámetros de cada modelo. Para realizar la calibración de estos parámetros, proponen diferentes algoritmos de aproximación estocástica. Los resultados de estos algoritmos son muy buenos al

igual que los resultados que arrojan los algoritmos genéticos, aunque los algoritmos de aproximación estocástica utilizan menos tiempo y recursos computaciones que los algoritmos genéticos.

En 2015, en [46] proponen una arquitectura para la calibración de modelos de tráfico vehicular, asistida por software para maximizar la practicidad, flexibilidad y facilidad de uso denominada SASCO (Sensitivity Analysis, Self-Calibration, and Optimization). SASCO cuenta con DBF (Directed Brute Force) y SPSA para realizar la calibración de los modelos, pero los resultados con DBF no son competitivos. Además, en SASCO se requiere que el usuario escoja las zonas de búsqueda para encontrar resultados óptimos.

Este mismo año (2015) en [47] se presenta un problema de optimización a gran escala no lineal con modelos de simulación de tráfico, donde se introduce una función objetivo que potencialmente tiene muchos óptimos locales (multimodal), usando una técnica de aproximación estocástica denominada Finite Difference Stochastic Approximation (FDSA) y SPSA. FDSA consume muchos recursos computacionales y recomiendan el uso de SPSA. En el trabajo se utiliza la medida NRMS y la estadística GEH para evaluar la calibración (contempladas en este trabajo).

También en este año, en [48] aplican el algoritmo Weighted Simultaneous Perturbation Stochastic Approximation (W-SPSA) ya que demuestra resultados eficientes en situaciones donde la correlación de variables no es homogénea, característica crucial en los modelos de simulación de tráfico. El problema de esta propuesta es el costo computacional de los métodos definidos para crear la matriz de pesos que se requieren, métodos como la derivación analítica, la aproximación basada en simulación y la aproximación numérica.

Además, en [49] presentan un algoritmo memético para optimizar un conjunto de parámetros aplicados en un modelo de cambio del uso del suelo y en la cobertura de la tierra basado en dinámica EGO. El algoritmo memético obtiene resultados de alta precisión y reduce significativamente los costos computacionales.

Posteriormente en 2015, en [50] plantean el problema a gran escala del flujo vehicular de taxis en Berlín (Alemania), ya que existen más de cinco mil taxis. Definen dos heurísticas para despachar taxis dentro de la simulación logrando optimizar los recursos de las empresas. Sin embargo, plantean estudiar otros algoritmos de despacho más sofisticados para resolver este tipo de problema.

En 2016, en [51] muestran que en los modelos microscópicos de tráfico existen problemas para comprender mejor los fenómenos del flujo vehicular que son impredecibles como las oscilaciones de aceleración y desaceleración de los carros, tiempo de reacción de los conductores, estos parámetros no son directamente observables a partir de datos de campo y pueden estar sujetos a errores y ruidos. Para estos inconvenientes, plantean un Método de Entropía Cruzada (CEM por sus siglas en inglés), además de presentar un Análisis de Sensibilidad Probabilístico

(PSA por sus siglas en inglés), un algoritmo basado en entropía relativa para identificar los parámetros importantes en el proceso de calibración. Este marco propuesto de calibración fue probado con varios estudios empíricos, mostrando buenos resultados encontrando el óptimo global. Se precisa resaltar que, en la función objetivo de esta calibración, se tienen en cuenta las variables de volumen y velocidad (variables que se consideran en este trabajo).

Además, en [52] proponen un algoritmo memético multi-objetivo (NSGA-II con recocido simulado) para la calibración de modelos de flujo de tráfico vehicular CORSIM. NSGA-II explora el espacio de búsqueda y obtiene un número de soluciones candidatas (Frente de Pareto), explotando dichas soluciones con recocido simulado. Este trabajo muestra que el algoritmo memético propuesto obtiene mejores resultados de calibración en menor tiempo que los algoritmos GASA y SPSA. Los dos objetivos propuestos para este estudio son volumen y velocidad (variables que se consideran en este trabajo) además de utilizar como criterio de calibración la estadística GEH (medida considerada en este trabajo).

Finalmente, en [53] desarrollan una metodología para calibración en línea de modelos de simulación microscópicos de tráfico para la predicción dinámica de múltiples pasos de medidas de tráfico y se aplica a los modelos de automóviles. Este enfoque tiene dos pasos principales, una fase de estimación y una fase de predicción. La fase de estimación se basa en un algoritmo de optimización global con restricciones. En cada paso del tiempo, la predicción se logra utilizando los valores estimados del anterior paso de tiempo, es decir, las mejores estimaciones disponibles de comportamiento de los conductores. Para poner a prueba esta metodología, se realiza una calibración estática y dinámica con datos de trayectoria disponibles a partir de un experimento realizado en Nápoles. La calibración estática se realiza con el fin de tener un punto de referencia para determinar el porcentaje de éxito de la calibración. En la calibración dinámica (también podría ser declarado como un problema de optimización multi-objetivo) los valores de los parámetros son constantes para los instantes de tiempo y las medidas de predicción. Para las dos calibraciones, se utiliza el algoritmo Improved Stochastic Ranking Evolution Strategy (ISRES). Los resultados de esta investigación sugieren que la calibración dinámica sea la más óptima para los modelos microscópicos de tráfico. En esta investigación, como medida de bondad se utiliza NRMS (medida considerada en este trabajo) y la velocidad como medida de eficacia (variable que se considera en este trabajo).

# Capítulo 3

## 3 PROPUESTA

### 3.1 INTRODUCCIÓN

En el presente capítulo se describe inicialmente la función objetivo a utilizar y el criterio a tener en cuenta para considerar que un modelo CORSIM ha sido calibrado. Luego, se muestra la representación de un individuo que da solución al problema de calibración de modelos CORSIM teniendo en cuenta las características que componen dichos modelos y las restricciones que se deben cumplir. Posteriormente, se presenta como fueron adaptados los tres algoritmos meta-heurísticos previamente mencionados en el capítulo 2 para solucionar el problema de calibración y la forma como se utilizan los hilos (threads) de ejecución en paralelo para optimizar el proceso de calibración.

### 3.2 FUNCIÓN OBJETIVO

Como función objetivo o medida de bondad de ajuste como se denota en el proceso de calibración descrito en [4] se adaptó la propuesta hecha por Paz en [12] y las recomendaciones de [4] y [47]. Esta función se conoce como Normalized Root Mean Squared (NRMS), pero en este trabajo se le agregó un componente relacionado con la longitud de las colas en los enlaces arteriales (*link*) de las carreteras. A continuación, en la **Ecuación (1)** se presenta la función NRMS ampliada.

$$NRMS = \frac{1}{\sqrt{N}}\left(\begin{array}{c}\alpha * \left(\sqrt{\sum_{i=1}^{N}\left(\frac{V_{i,t} - \tilde{V}_{(\theta)i,t}}{V_{i,t}}\right)^2}\right) + \beta * \left(\sqrt{\sum_{i=1}^{N}\left(\frac{S_{i,t} - \tilde{S}_{(\theta)i,t}}{S_{i,t}}\right)^2}\right) + \\ \gamma * \left(\sqrt{\sum_{i=1}^{N}\left(\frac{Q_{i,t} - \tilde{Q}_{(\theta)i,t}}{Q_{i,t}}\right)^2}\right)\end{array}\right) \quad (1)$$

Sujeto a que: $\alpha + \beta + \gamma = 1$

Donde:

$V_{i,t}$ = Conteo de vehículos reales para el enlace $i$ en el tiempo $t$.

$\tilde{V}_{(\theta)i,t}$ = Conteo de vehículos simulados para el enlace $i$ en el tiempo $t$.

$S_{i,t}$ = Velocidad real para el enlace $i$ en el tiempo $t$.

$\tilde{S}_{(\theta)i,t}$ = Velocidad simulada para el enlace $i$ en el tiempo $t$.

$Q_{i,t}$ = Conteo de vehículos reales que forman la longitud de cola para el enlace $i$ en el tiempo $t$.

$\tilde{Q}_{(\theta)i,t}$ = Conteo de vehículos simulados que forman la longitud de cola para el enlace $i$ en el tiempo $t$.

$\alpha$ = Peso para el componente de volumen.

$\beta$ = Peso para el componente de velocidad.

ɣ = Peso para el componente de longitud de colas.

$N$ = Número total de enlaces en el modelo.

$T$ = Número total de períodos de tiempo $t$.

Con esta función objetivo se obtiene una medida que indica que tan diferentes son los datos de salida de la simulación y los datos de campo de cada uno de los enlaces arteriales que conforman el modelo para cada una de las variables en las cuales se centra la presente investigación (volumen, velocidad y longitud de colas).

El simulador CORSIM, para poder generar los datos de salida necesarios para la comparación con los datos de campo en la función objetivo, requiere de un conjunto de parámetros que hacen parte de la representación de un individuo, el cual se describe posteriormente en la sección 3.4. De este modo, el propósito de los algoritmos radica en calibrar correctamente el individuo que contiene los parámetros necesarios para la calibración, y obtener los datos de salida simulados lo más parecido posible a los datos de campo, es decir, minimizar la función objetivo dada.

Cada uno de los pesos, $\alpha$, $\beta$ y ɣ en la función objetivo permiten que el proceso de calibración sea más flexible, ya que se establecen de acuerdo a la completitud y fiabilidad de los datos de campo de cada uno de los componentes (volumen, velocidad y longitud de colas).

## 3.3 CRITERIO DE CALIBRACIÓN

Para la presente investigación se adoptaron las directrices provistas por la Federal Highway Administration (FHWA), en donde se utiliza la estadística GEH, que se muestra a continuación en la **Ecuación (2)**.

$$GEH = \sum_{i=1}^{N} \sqrt{\frac{2(V_i - \tilde{V}_{(\theta)i})^2}{V_i + \tilde{V}_{(\theta)i}}} \qquad (2)$$

Donde:

$V_{i,t}$ = Datos de campo de volumen vehicular para el enlace $i$ en el tiempo $t$.

$\tilde{V}_{(\theta)i,t}$ = Datos simulados de volumen vehicular para el enlace $i$ en el tiempo $t$.

$N$ = Número total de enlaces en el modelo.

Se busca que la estadística GEH sea menor a 5 para al menos el 90% de los enlaces del modelo, ya que esta medida establece la diferencia entre el conteo (volumen)

de enlaces reales y los simulados. En las directrices de la FHWA un modelo se considera aceptablemente calibrado si cumple con un valor menor a 5 para al menos el 85% de los enlaces.

## 3.4 REPRESENTACIÓN DE UN INDIVIDUO

En la sección 2.1.2 se describió como se estructura un modelo CORSIM y los elementos que lo conforman. Uno de ellos son los *records types*, los cuales deben ser seleccionados por el usuario al iniciar el proceso de calibración. Adicionalmente, el usuario debe definir las entradas específicas del *record type* que desea calibrar para cada uno de los enlaces que contiene el modelo. La descripción detallada de todos los posibles *records types* y sus posibles entradas en los modelos CORSIM se encuentran en el manual de referencia CORSIM [21].

En la **Figura 4**, el recuadro verde indica que el usuario ha seleccionado el *record type* 11 y desea calibrar el valor de dicha entrada, cuyo valor está comprendido entre la columna 57 y 60 ($Start\ column$ y $End\ column$). El valor actual para esta entrada es de 80 ($Value$) y los posibles valores que podrá tomar están entre 0 y 99 ($Minimum\ Value$ y $Maximum\ Value$). Este registro se encuentra ubicado en la línea número 9 ($Line\ number$) del modelo y pertenece al enlace formado por los nodos 9 y 1 ($Initial\ node$ y $End\ node$). Del mismo modo, el recuadro azul hace referencia a un nuevo parámetro que ha sido seleccionado por el usuario para el proceso de calibración. Todo el conjunto de parámetros que seleccione el usuario para calibrar, formarán una matriz la cual se denota como los Genes del individuo donde se calibrarán específicamente los valores que se muestran en el recuadro rojo (**Figura 4**).

| Record type | Start column | End column | Minimum value | Maximum value | Value | Line number | Initial node | End node |
|---|---|---|---|---|---|---|---|---|
| 11 | 61 | 64 | 14 | 99 | 58 | 8 | 8001 | 9 |
| 11 | 57 | 60 | 0 | 99 | 80 | 9 | 9 | 1 |
| 56 | 25 | 28 | 10 | 1000 | 950 | 10 | 1 | 9 |
| 81 | 1 | 4 | 1 | 8 | 5 | 11 | 1 | 4 |
| 81 | 5 | 8 | 1 | 30 | 15 | 12 | 4 | 1 |
| 81 | 13 | 16 | 1 | 10 | 5 | 13 | 8003 | 3 |

***Figura 4:*** *Ejemplo de parámetros a calibrar (Genes del individuo)*

Adicional a esto, el individuo tiene una variable donde almacena su valor de aptitud o fitness (valor de NRMS), que se calcula procesando los genes del individuo en el simulador CORSIM para obtener los datos de salida referentes a las variables objetivo en la presente investigación (volumen, velocidad y longitud de colas) y

evaluarlos junto con los datos de campo en la función objetivo definido en la sección 3.2. A continuación, en la **Figura 5** se presenta un ejemplo de un individuo.

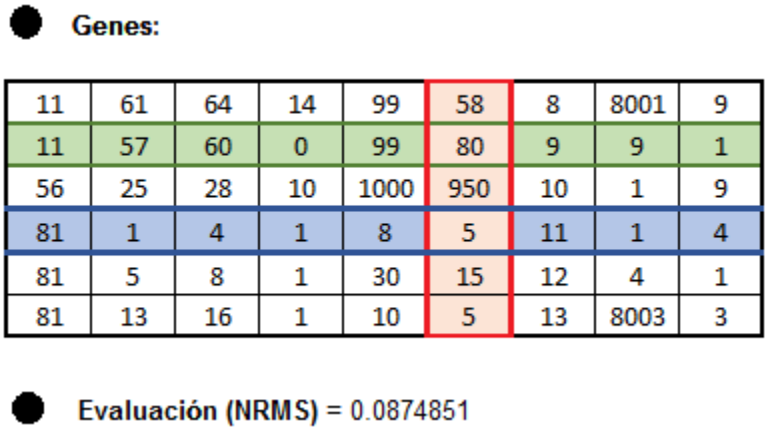



***Figura 5:*** *Ejemplo de representación del individuo*

Cabe aclarar que en el proceso de calibración sólo se realizan cambios a la columna de color rojo (columna número 6) de los individuos, ya que esta columna contiene los parámetros que se van a calibrar. Por esta razón, en la explicación de los algoritmos meta-heurísticos propuestos se toman los genes como un vector que representan dicha columna. Además, las otras columnas de la matriz son informativas (extraídas del modelo CORSIM), que entre otras cosas, ayudan a restringir los valores de la columna 6.

## 3.5 RESTRICCIONES DE UN INDIVIDUO

En los modelos CORSIM el conjunto de parámetros que conforman un *record type* conocidos como entradas, están sujetos a la restricción de un valor mínimo y máximo, tal como se muestra en la **Figura 4** de la sección anterior, donde la columna cuatro y cinco de la matriz de genes del individuo restringe el rango del valor que pueda tomar el parámetro o entrada.

Adicionalmente, se debe considerar una segunda restricción para las entradas de un mismo *record type*, en el cual los valores de estos parámetros están relacionados. Un ejemplo de esta restricción se aclara con el *record type* 21, el cual contiene cuatro entradas que representan el porcentaje de vehículos que giran a la derecha, a la izquierda, al frente y diagonal en un determinado cruce. Al tratarse de valores de porcentajes, las restricciones están dadas por un valor mínimo de cero, un valor máximo de cien y la suma de los cuatro porcentajes debe ser igual a cien.

Cada una de las restricciones en los diferentes *record type* de los modelos CORSIM se encuentran en el manual de referencia CORSIM [25] y han sido tenidas en cuenta en el desarrollo de este proyecto.

## 3.6 ADAPTACIÓN DE LOS ALGORITMOS META-HEURÍSTICOS

Las meta-heurísticas DECC-G, MOS y MA-SW-Chains, fueron diseñadas para resolver problemas de optimización continua de alta dimensionalidad sin restricciones, es decir, optimiza individuos cuyos genes son representados con valores decimales sin restricción alguna. Además, estos algoritmos meta-heurísticos realizan la evaluación de sus individuos de forma secuencial, no se proponen como meta-heurísticas paralelas.

En esta investigación, DECC-G, MOS y MA-SW-Chains se adaptaron para resolver un problema específico de optimización discreta con restricciones, es decir, que los individuos toman valores enteros en sus genes y estos individuos son revisados y reparados para asegurar que cumplen con una serie de restricciones ligadas a los modelos de micro-simulación CORSIM, luego cuando ya son soluciones válidas, se realiza la evaluación de la aptitud de dichos individuos. Para la evaluación de la función de aptitud, se utiliza la ejecución de hilos en paralelo, evaluando un individuo por cada hilo, esto con el fin de realizar mayor cantidad de evaluaciones de la función objetivo en menos tiempo, garantizando que, en todo el proceso de calibración, los diferentes algoritmos ejecuten la misma cantidad de hilos, incluidos los algoritmos del estado del arte, GASA y SPSA. Además, algunos parámetros o valores definidos en los algoritmos fueron ajustados al problema específico, como por ejemplo el número de ejecuciones que debe realizar Solis and Wets en el algoritmo MA-SW-Chains.

## 3.7 COEVOLUCIÓN COOPERATIVA BASADA EN EVOLUCIÓN DIFERENCIAL

En esta sección se explica la adaptación realizada al algoritmo meta-heurístico de Coevolución Cooperativa basada en Evolución Diferencial (DECC-G) para la calibración de modelos de micro-simulación de flujo de tráfico vehicular CORSIM. Para tener una visión general de la meta-heurística, la **Figura 6**.

Pseudocódigo de la meta-heurística DECC-G

**Entrada:** Tamaño de la población $NP$, Número de ciclos $ciclos$, Cantidad de subcomponentes $m$, Número de evaluaciones de la función objetivo para SaNSDE $FEs$, Número de evaluaciones de la función objetivo para Evolución Diferencial $wFEs$
**Salida:** Mejor individuo $Best$

1: $poblacion =$ Generar población inicial aleatoriamente
2: $poblacionPesos =$ Definir población de pesos
3: $evaluar\ (poblacion)$
4: $i = 0$
5: **Mientras** no se cumpla la condición de parada ($i < ciclos$) **hacer**
6: $indices$ = Permutación aleatoria ($n$) // $n$ es la cantidad de genes de los individuos
7: $j = 1$
8: **Mientras** no se cumpla la condición de parada ($j \leq m$) **hacer** // $m = \frac{n}{s}$
9: $l = \big((j-1) * s\big) + 1$ // $s$ es la cantidad de genes de cada subcomponente

10: $u = j * s$
11: $subPoblacion = poblacion[l, u]$ // Escoger los genes de todos los individuos desde // $l$ hasta $u$
12: $subPoblacion =$ Optimizar $subPoblacion$ con SANSDE $(subPoblacion, FEs)$
13: Inicializar pesos del subcomponente ($subPoblacion$)
14: $poblacion[l, u] = subPoblacion$
15: $j++$
16: **Fin mientras**
17: $Best =$ Encontrar Mejor Individuo ($poblacion$)
18: $rand =$ Encontrar Aleatorio Individuo ($poblacion$)
19: $peor =$ Encontrar Peor Individuo ($poblacion$)
20: $poblacion[posicion\_Best] =$ Optimizar $poblacionPesos$ y aplicarlos a $Best$ con DE $(Best, poblacionPesos, wFEs)$
21: $poblacion[posicion\_Rand] =$ Optimizar p$oblacionPesos$ y aplicarlos a $rand$ con DE $(rand, poblacionPesos, wFEs)$
22: $poblacion[posicion\_Peor] =$ Optimizar $poblacionPesos$ y aplicarlos a $peor$ con DE $(peor, poblacionPesos, wFEs)$
23: $Best =$ Encontrar Mejor Individuo ($poblacion$)
24: $i++$
25: **Fin mientras**
26: **Retornar** $\boldsymbol{Best}$

***Figura 6:*** *Pseudocódigo de la meta-heurística DECC-G*

En la línea 1 se genera la población inicial de individuos ($poblacion$) de forma aleatoria con un total de individuos definido por el usuario (tamaño de la población = $NP$). La dimensionalidad del individuo se forma de acuerdo a la cantidad de parámetros del modelo que se desea calibrar. Cada uno de estos parámetros tiene un rango de posibles valores definido en los modelos CORSIM, el cual es respetado en el proceso de creación aleatoria de sus valores al igual que las otras restricciones establecidas por dichos modelos. Además, las variables son discretas (enteras) y los datos generados también cumplen con esa regla.

En la línea 2 se define una población de vectores de pesos ($poblacionPesos$) de tamaño $NP$. Estos vectores de pesos, se inicializan con un valor de uno y tienen el mismo tamaño $n$ de los individuos de la población generada en la línea 1. Para mayor claridad, en la **Figura 7** se muestra la definición de una población de vectores de pesos.

| *Individuo* | *1* | *2* | *3* | *…* | *n - 2* | *n - 1* | *n* |
|---|---|---|---|---|---|---|---|
| *1* | 1 | 1 | 1 | … | 1 | 1 | 1 |
| *2* | 1 | 1 | 1 | … | 1 | 1 | 1 |
| *3* | 1 | 1 | 1 | … | 1 | 1 | 1 |
| *4* | 1 | 1 | 1 | … | 1 | 1 | 1 |
| *5 = NP* | 1 | 1 | 1 | … | 1 | 1 | 1 |

***Figura 7:*** *Definición de la población de vectores de pesos*

En la línea 3 se calcula el valor de aptitud basado en NRMS según la **Ecuación (1)** para todos los individuos que conforman la $poblacion$, esta evaluación se explica en la sección 3.10.1.

En la línea 4 se inicializa una variable $i = 0$ que controla el final del proceso evolutivo que realiza DECC-G. Desde la línea 5 hasta la línea 25 se encuentra dicho proceso evolutivo cuya condición de parada consiste en cumplir el número de ciclos que configure el usuario.

En la línea 6 se llena un vector $indices$ con una permutación aleatoria de tamaño $n$ (tamaño de los individuos). Cada elemento del vector $indices$ indica las posiciones de los genes de cada subcomponente en que se va dividir los individuos de la $poblacion$. La **Figura 8** ilustra un vector inicializado con una permutación aleatoria donde $n = 8$.

| 6 | 4 | 8 | 2 | 1 | 7 | 3 | 5 |
|---|---|---|---|---|---|---|---|

***Figura 8:*** *Vector inicializado con una permutación aleatoria*

En la línea 7 se inicializa una variable $j = 1$ que controla y garantiza la formación de los $m$ subcomponentes que el usuario desee. Desde la línea 8 hasta la línea 16 se forman uno a uno los $m$ de subcomponentes (líneas 9, 10 y 11) y cada uno de ellos es optimizado por el algoritmo SaNSDE (línea 12). Para dar mayor claridad acerca de la formación de los subcomponentes ($subPoblacion$), se supondrá que el individuo tiene un tamaño de ocho dimensiones ($n = 8$), el usuario ha decidido formar tres subcomponentes ($m = 3$) y que la permutación aleatoria arrojó como resultado el vector de la **Figura 8**, entonces los dos primeros subcomponentes tendrán un tamaño de tres dimensiones y el tercer subcomponente tendrá un tamaño de dos dimensiones, luego la población quedará dividida en subcomponentes como lo muestra la **Figura 9** donde cada color indica cada subcomponente que se va optimizar uno por uno con el algoritmo SaNSDE. Para el primer subcomponente, se escogen las tres primeras posiciones (6, 4, 8) del vector $indices$ de la **Figura 8**, por lo tanto este subcomponente lo conforma los genes en las posiciones 4, 6 y 8 de todos los individuos de la población, luego se escogen los tres siguientes (2, 1, 7) para formar el segundo subcomponente, de este modo el siguiente subcomponente lo conforma los genes en las posiciones 1, 2 y 7 de todos los individuos de la población y por último se escogen los dos siguientes (3, 5) para formar el tercer subcomponente, por lo tanto este subcomponente lo conforma los genes en las posiciones 3 y 5 de todos los individuos de la población.

| *Individuo* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* |
|---|---|---|---|---|---|---|---|---|
| *1* | 94 | 41 | 79 | 44 | 58 | 3 | 41 | 76 |
| *2* | 11 | 55 | 70 | 6 | 83 | 28 | 89 | 56 |
| *3* | 31 | 93 | 16 | 24 | 91 | 79 | 97 | 42 |
| *4* | 84 | 60 | 1 | 41 | 90 | 69 | 57 | 97 |
| *5* | 35 | 95 | 61 | 24 | 60 | 46 | 91 | 25 |

***Figura 9:*** *Población dividida en subcomponentes*

Luego de que cada subcomponente es optimizado por SaNSDE, a cada uno de ellos se inicializa de forma aleatoria el vector de pesos (línea 13). La inicialización de los vectores de peso se realiza con valores aleatorios entre cero y uno. En la **Figura 10** se muestra la inicialización del primer subcomponente (6, 4, 8) de la población de los vectores de pesos, es decir de las columnas amarillas, la 4, 6 y 8.

| *Individuo* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* |
|---|---|---|---|---|---|---|---|---|
| *1* | 1 | 1 | 1 | 0,50 | 1 | 0,41 | 1 | 0,36 |
| *2* | 1 | 1 | 1 | 0,58 | 1 | 0,22 | 1 | 0,13 |
| *3* | 1 | 1 | 1 | 0,54 | 1 | 0,41 | 1 | 0,71 |
| *4* | 1 | 1 | 1 | 0,95 | 1 | 0,27 | 1 | 0,89 |
| *5* | 1 | 1 | 1 | 0,03 | 1 | 0,45 | 1 | 0,25 |

***Figura 10:*** *Inicialización de un subcomponente de la población de vectores de pesos*

Cuando cada subcomponente es optimizado se reemplaza en la $poblacion$, teniendo así nuevos individuos con un subcomponente optimizado en la población (línea 14). La línea 15 indica que un subcomponente ha sido optimizado.

En la línea 17, se encuentra el mejor individuo (valor de aptitud más bajo porque se está minimizando) y su posición en la $poblacion$, luego en la línea 18 se selecciona en forma aleatoria un individuo y su posición en la $poblacion$, y después en la línea 19 se encuentra el peor individuo (valor de aptitud más alto) y su posición en la $poblacion$. En las líneas 20, 21 y 22 se optimiza la $poblacionPesos$ y se aplican estos vectores de pesos a los individuos encontrados en las líneas 17, 18 y 19 (mejor, aleatorio y peor) respectivamente con Evolución Diferencial.

En la línea 23 se encuentra el mejor individuo o $Best$ y su posición en la $poblacion$, en la línea 24 se indica que se ha cumplido un ciclo en el proceso evolutivo y la línea 26 retorna el mejor individuo ($Best$) encontrado por la meta-heurística DECC-G.

### 3.7.1 Adaptación de Evolución Diferencial con Búsqueda Auto-adaptativa de Vecindario

Como se mencionó anteriormente, los subcomponentes de todos los individuos formados por DECC-G son optimizados por SaNSDE, de acuerdo al pseudocódigo adaptado para soportar el proceso de calibración que se presenta en la **Figura 11**.

Pseudocódigo del algoritmo SaNSDE

**Entrada:** Número de evaluaciones de la función objetivo $FEs$, Periodo de aprendizaje de P y Fp $periodoPyFp$, Periodo de aprendizaje de CR $periodoCR$
**Salida:** Población optimizada $Poblacion$

1: $contadorFEs = 0, contadorPyFP = 0, contadorCRm = 0$
2: $p = 0.5, fp = 0.5, CRm = 0.5$
3: $Poblacion =$ Inicializar población // Esta población es la que viene de DECC-G
4: $indices =$ Establecer las posiciones de los genes de los individuos que conforman el subcomponente que se va optimizar
5: **Mientras** no se cumpla la condición de parada ($contadorFEs < FEs$) **hacer**
6: $mutados$ = Mutar los genes que pertenecen a $indices$ de $Poblacion$
7: $cruzados$ = Cruzar los genes que pertenecen a $indices$ de los $mutados$ y de $Poblacion$
8: $evaluar\ (cruzados)$
9: $Poblacion =$ Seleccionar ($Poblacion, cruzados$)
10: $contadorPyFP = contadorPyFP + Tamaño\ de\ cruzados$
11: $contadorCRm = contadorCRm + Tamaño\ de\ cruzados$
12: **Si** $contadorPyFP \geq$ periodoPyFp **entonces**
13: $p =$ Actualizar probabilidad $p$
14: $fp =$ Actualizar probabilidad $fp$
15: $contadorPyFP = 0$
16: **Fin si**
17: **Si** $contadorCRm \geq$ periodoCR **entonces**
18: $CRm =$ Actualizar probabilidad de la Tasa de Cruce $CRm$
19: $contadorCRm = 0$
20: **Fin si**
21: $contadorFEs = contadorFEs + Tamaño\ de\ cruzados$
22: **Fin mientras**
23: **Retornar** $\boldsymbol{Poblacion}$

***Figura 11:*** *Pseudocódigo del algoritmo SaNSDE*

En las líneas 1 y 2 se inicializan las variables que cuentan el número de evaluaciones de la función objetivo que se van realizando, además de inicializar en 0.5 el control de estrategias de mutación ($p$), la probabilidad de auto adaptación del Factor de Escala ($fp$) y la probabilidad de auto adaptación de la tasa de cruce ($CRm$) que son utilizadas para la mutación y el cruce. Este valor inicial de dichas probabilidades, se toma según recomendaciones de los autores de [29].

En la línea 3, se establece $subPoblación$ de la meta-heurística DECC-G en $Poblacion$, vale recordar que $subPoblación$ contiene el subcomponente que se va optimizar. En la línea 4 se establece el vector $indices$ que contiene las posiciones de los genes del subcomponente que se va optimizar.

Desde la línea 5 hasta la línea 22, se realiza el proceso evolutivo del algoritmo SaNSDE, utilizando mutación, cruce, selección y auto adaptación de parámetros según el periodo de aprendizaje que haya establecido el usuario para cada parámetro, además, la condición de parada consiste en completar el número de evaluaciones de la función objetivo ($FEs$).

En la línea 6 se realiza la operación de mutación. Para generar un individuo mutado $V_i$, se tiene en cuenta el vector $indices$ que contiene las posiciones del subcomponente que se va optimizar, es decir, sólo estas posiciones se van a modificar en el proceso evolutivo. Para la mutación se establece el Factor de Escala ($F$) de acuerdo al parámetro de auto adaptación de $F$ ($fp$). El valor del Factor de Escala se genera basado en la **Ecuación (3)**.

$$F = \begin{cases} N(0.3, 0.5), Si\ Aleatorio(0,1) < fp \\ Cauchy(0,1), \quad en\ otro\ caso \end{cases} \qquad (3)$$

Donde:

$Aleatorio(0,1) =$ Número aleatorio entre 0 y 1.

$N(0.3, 0.5) =$ Valor de la Distribución Normal o Gaussiana con media = 0.3 y desviación estándar = 0.5.

$Cauchy(0,1) =$ Valor de la Distribución Cauchy con $X_0 = 0$ y Escala = 1.

SaNSDE tiene dos estrategias de mutación, el algoritmo determina cual estrategia utilizar de acuerdo a una probabilidad de control de las estrategias de mutación ($p$) como se muestra en la **Ecuación (4)**.

$$V_i = \begin{cases} X_1 + F * (X_2 - X_3), si\ Aleatorio(0,1) < p \\ X_i + F * (X_{best} - X_i) + F(X_1 - X_2), \ \ en\ otro\ caso \end{cases} \qquad (4)$$

Donde:

$V_i =$ Individuo mutado

$Aleatorio(0,1) =$ Número aleatorio entre 0 y 1.

$X_1, X_2, X_3 =$ Son tres individuos aleatorios de la población original diferentes entre sí.

$X_i =$ Individuo en la posición $i$ de la población.

$X_{best} =$ Mejor individuo de la población.

De esta manera se genera un individuo mutado, al cual se le revisan y corrigen las restricciones necesarias para evitar valores incoherentes, luego se agrega a la población de individuos mutados ($mutados$), este proceso se repite hasta que esta población tenga el mismo tamaño de la $poblacion$ ($NP$).

En la línea 7 se realiza la operación de cruce; se establece la Tasa de Cruce ($CR$) con un valor de la distribución Gaussiana o Normal con media = $CRm$ y desviación

estándar = 0.1, donde $CRm$ es el parámetro de auto adaptación de $CR$. Se tiene en cuenta el vector $indices$ que contiene las posiciones del subcomponente que se va optimizar, es decir, sólo estas posiciones se van a modificar en el proceso evolutivo. Para generar el primer individuo cruzado o de prueba $U_i$, se obtiene el primer individuo de la población $X_i$ ($Poblacion$) y el primer individuo de la población de individuos mutados $V_i$ ($mutados$) y se realiza según la **Ecuación (5)**.

$$U_i[j] = \begin{cases} V_i[j], si\ Aleatorio(0,1) < CR \\ X_i[j], \quad en\ otro\ caso \end{cases} \qquad (5)$$

Donde:

$U_i[j] =$ Valor del gen en la posición $j$ del individuo cruzado o de prueba en la posición $i$ de la población de individuos cruzados.

$V_i[j] =$ Valor del gen en la posición $j$ del individuo mutado en la posición $i$ de la población de individuos mutados.

$X_i[j] =$ Valor del gen en la posición $j$ del individuo en la posición $i$ de la población.

$Aleatorio(0,1) =$ Número aleatorio entre 0 y 1.

Así se genera un individuo cruzado o de prueba, al cual se le aplica las restricciones necesarias para evitar valores incoherentes, luego se agrega a la población de individuos cruzados ($cruzados$), este proceso se repite hasta que esta población tenga el mismo tamaño de la población original ($NP$). En la línea 8, esta población de individuos cruzados se evalúa como se explicará en la sección 3.10.1.

En la línea 9 se realiza la operación de selección; aquí se seleccionan los mejores individuos entre la $Poblacion$ y la población de individuos cruzados ($cruzados$) para formar la nueva $Poblacion$. Dicha selección se realiza comparando uno a uno los individuos de la población original con los individuos de la población de cruzados de forma secuencial. La selección se realiza según la **Ecuación (6)**.

$$X'_i = \begin{cases} U_i, si\ Aptitud\ de\ U_i < Aptitud\ de\ X_i \\ X_i, \quad en\ otro\ caso \end{cases} \qquad (6)$$

Donde:

$X'_i =$ Individuo que ocupará la posición $i$ en la siguiente generación.

$U_i =$ Individuo cruzado en la posición $i$ de la población de individuos cruzados.

$X_i =$ Individuo en la posición $i$ de la población.

Cada vez que un individuo cruzado pasa a la siguiente generación, se calcula la mejora ($Aptitud\ de\ X_i - Aptitud\ de\ U_i$) que obtuvo este individuo cruzado sobre el individuo original, estas mejoras se almacenan en el vector llamado $frec$. Este vector se utiliza para la auto adaptación de la Tasa de Cruce ($CR$). Además, se debe contar el número de individuos cruzados que pasan a la siguiente generación y han

sido creados por la estrategia de mutación número 1 ($X_1 + F * (X_2 - X_3)$), contar el número de individuos cruzados descartados para la siguiente generación y han sido creados por la estrategia de mutación número 1, contar el número de individuos cruzados que pasan a la siguiente generación y han sido creados por la estrategia de mutación número 2 ($(X_i + F * (X_{best} - X_i) + F(X_1 - X_2)$), contar el número de individuos cruzados descartados para la siguiente generación y han sido creados por la estrategia de mutación número 2. Estos contadores se utilizan para actualizar el control de estrategias de mutación ($p$). De manera similar, se crean contadores para actualizar la probabilidad de auto adaptación del Factor de Escala ($fp$).

En las líneas 10 y 11, se actualizan las variables $contadorPyFP$ y $contadorCRm$ de acuerdo al número de evaluaciones de la función objetivo que se realizaron, para determinar si cumplieron el período de aprendizaje ($periodoPyFP$ y $periodoCR$) para actualizar las probabilidades $p$, $fp$ y $CRm$.

Desde la línea 12 hasta la línea 16, se actualizan el control de estrategias de mutación ($p$) y la probabilidad de auto adaptación del Factor de Escala ($fp$), si ha cumplido con el periodo de aprendizaje ($periodoPyFP$). El control de estrategias de mutación ($p$) se actualiza de acuerdo a la **Ecuación (7)**.

$$p = \frac{ns1 * (ns2 + nf2)}{ns2 * (ns1 + nf1) + ns1 * (ns2 + nf2)} \quad (7)$$

Donde:

$ns1 =$ Número de individuos correctamente introducidos en la nueva generación creados por la estrategia de mutación número 1.

$ns2 =$ Número de individuos correctamente introducidos en la nueva generación creados por la estrategia de mutación número 2.

$nf1 =$ Número de individuos descartados en la nueva generación creados por la estrategia de mutación número 1.

$nf2 =$ Número de individuos descartados en la nueva generación creados por la estrategia de mutación número 2.

Luego de actualizar el control de estrategias de mutación ($p$), se actualiza la probabilidad de auto adaptación del Factor de Escala ($fp$) de acuerdo a la **Ecuación (8)**.

$$fp = \frac{fp\_ns1 * (fp\_ns2 + fp\_nf2)}{fp\_ns2 * (fp\_ns1 + fp\_nf1) + fp\_ns1 * (fp\_ns2 + fp\_nf2)} \quad (8)$$

Donde:

$fp\_ns1 =$ Número de veces en que el Factor de Escala $F$ permitió ingresar con éxito un individuo en la nueva generación creado por la estrategia de mutación número 1.

$fp\_ns2 =$ Número de veces en que el Factor de Escala $F$ permitió ingresar con éxito un individuo en la nueva generación creado por la estrategia de mutación número 2.

$fp\_nf2 =$ Número de veces en que el Factor de escala $F$ no ingreso un individuo en la nueva generación creado por la estrategia de mutación número 1.

$fp\_nf2 =$ Número de veces en que el Factor de escala $F$ no ingreso un individuo en la nueva generación creado por la estrategia de mutación número 2.

Desde la línea 17 hasta la línea 20 se actualiza la probabilidad de auto adaptación de $CR$ ($CRm$), si ha cumplido con el periodo de aprendizaje ($periodoCR$). La probabilidad de auto adaptación de $CR$ ($CRm$) se actualiza de acuerdo a la **Ecuación (9)**.

$$CRm = \sum_{k=1}^{n} w_k * CR_{Rec_k} \qquad (9)$$

$$w_k = \frac{frec_k}{\sum_{j=0}^{n} frec_j}$$

Donde:

$frec =$ Es un vector que almacena las mejoras en el valor de aptitud de los individuos cruzados sobre los originales (este vector se llena en el proceso de selección).

$CR_{Rec} =$ Es un vector que almacena todos los valores de Tasa de Cruce ($CR$) generados por la distribución Gaussiana en el proceso de cruce.

En la línea 21 se actualiza la variable $contadorFEs$ de acuerdo al número de evaluaciones de la función objetivo que se realizaron y la línea 23 retorna la $Poblacion$ optimizada por SaNSDE.

### 3.7.2 Adaptación de Evolución Diferencial

Como se mencionó anteriormente, el algoritmo de Evolución Diferencial optimiza los vectores de pesos del mejor, peor y un individuo aleatoriamente seleccionado de la población de DECC-G. Para realizar esta optimización se usa Evolución Diferencial como se ilustra en la **Figura 12**.

Pseudocódigo de Evolución Diferencial

**Entrada:** Número de evaluaciones de la función objetivo $wFEs$, Individuo $inicial$
**Salida:** Mejor individuo $Best$
1: $contadorFEs = 0$
2: $poblacionPesos =$ Inicializar Población de Pesos // Esta inicialización la hace DECC-G

3: $poblacion =$ Aplicar $poblacionPesos$ a $inicial$
4: $evaluar\ (poblacion)$
5: $contadorFEs = contadorFEs + Tamaño\ de\ poblacion$
6: **Mientras** no se cumpla la condición de parada ($contadorFEs < wFEs$) **hacer**
7: $poblacionPesosMutados$ = Mutar los individuos de $poblacionPesos$
8: $poblacionPesosCruzados =$ Cruzar los individuos de $poblacionPesos$ y $poblacionPesosMutados$
9: $poblacionCruzados$ = Aplicar $poblacionPesosCruzados$ a $inicial$
10: $evaluar\ (poblacionCruzados)$
11: $poblacion$ = Seleccionar ($poblacion, poblacionCruzados$)
12: $poblacionPesos$ = Seleccionar ($poblacionPesos, poblacionPesosCruzados$)
13: $contadorFEs = contadorFEs + Tamaño\ de\ poblacionCruzados$
14: **Fin mientras**
15: $Best =$ Encontrar Mejor Individuo ($poblacion$)
16: **Si** $Aptitud\ de\ inicial < Aptitud\ de\ Best$) **entonces**
17: $Best = inicial$
18: **Fin si**
19: **Retornar** $\boldsymbol{Best}$

***Figura 12:*** *Pseudocódigo de Evolución Diferencial*

En la línea 1 se inicializa la variable que cuenta el número de evaluaciones de la función objetivo realizadas por Evolución Diferencial.

En la línea 2 se realiza la inicialización de la población de los vectores de pesos ($poblacionPesos$), esto lo hace la meta-heurística DECC-G en la línea 13 de su pseudocódigo (**Figura 6**), luego de ir optimizando cada subcomponente.

En la línea 3 se aplica toda la población de vectores de pesos al individuo que se va optimizar para generar una población de nuevos individuos que dan solución al problema de calibración, es decir, multiplicar cada uno de los vectores de pesos al individuo que se va optimizar. Para tener claridad, suponga que la **Figura 13** ilustra el individuo que va optimizar por Evolución Diferencial y la **Figura 14** representa una población de vectores de pesos.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 34 | 67 | 45 | 89 | 12 | 64 | 55 | 13 |

***Figura 13:*** *Individuo a optimizar por Evolución Diferencial*

| *Individuo* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* |
|---|---|---|---|---|---|---|---|---|
| *1* | 0,15 | 0,36 | 0,88 | 0,68 | 0,38 | 0,94 | 0,60 | 0,04 |
| *2* | 0,12 | 0,39 | 0,04 | 0,73 | 0,05 | 0,42 | 0,03 | 0,73 |
| *3* | 0,25 | 0,41 | 0,67 | 0,77 | 0,88 | 0,37 | 0,13 | 0,21 |
| *4* | 0,21 | 0,12 | 0,35 | 0,38 | 0,76 | 0,26 | 0,78 | 0,26 |
| *5* | 0,59 | 0,95 | 0,90 | 0,18 | 0,20 | 0,64 | 0,10 | 0,84 |

***Figura 14:*** *Población de vectores de pesos*

Entonces se multiplica el individuo que se va optimizar con cada uno de los vectores de pesos aplicando las restricciones pertinentes a estos nuevos individuos para

evitar valores incoherentes, obteniendo como resultado una nueva población de individuos con el tipo de dato correcto y dentro del rango definido, que darán solución al problema de calibración, como lo ilustra la **Figura 15**.

| *Individuo* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* |
|---|---|---|---|---|---|---|---|---|
| *1* | 5 | 24 | 40 | 61 | 5 | 60 | 33 | 1 |
| *2* | 4 | 26 | 2 | 65 | 1 | 27 | 2 | 9 |
| *3* | 8 | 28 | 30 | 69 | 11 | 24 | 7 | 3 |
| *4* | 7 | 8 | 16 | 34 | 9 | 17 | 43 | 3 |
| *5* | 20 | 64 | 41 | 16 | 2 | 41 | 6 | 11 |

***Figura 15:*** *Población luego de aplicar los vectores de pesos a un individuo*

En la línea 4 se calcula el valor de aptitud basado en NRMS para todos los individuos que conforman la $poblacion$, la cual fue obtenida en la línea 3, esta evaluación se explica en la sección 3.10.1. En la línea 5 se actualiza la variable $contadorFEs$ de acuerdo al número de evaluaciones de la función objetivo que se realizaron.

Desde la línea 6 hasta la línea 14 se realiza el proceso evolutivo de Evolución Diferencial, utilizando mutación, cruce y selección cuya condición de parada consiste en completar el número de evaluaciones de la función objetivo ($wFEs$).

En la línea 7 se realiza la operación de mutación; para generar un individuo mutado $V_i$, se tiene en cuenta el Factor de Escala ($F$) que toma un valor de 0.5 [29] (parámetro configurado por el usuario), se escogen tres vectores de pesos de forma aleatoria de la población de pesos (estos vectores deben ser diferentes entre sí) y se opera conforme a la **Ecuación (10)**.

$$V_i = |X_1 + F * (X_2 - X_3)| \qquad (10)$$

Donde:

$V_i =$ Vector mutado.

$X_1, X_2, X_3 =$ Son tres vectores aleatorios de $poblacionPesos$ diferentes entre sí.

Cabe mencionar que se toma el valor absoluto, ya que se están optimizando los vectores de pesos. De esta manera se genera un vector mutado, el cual se agrega a la población de pesos mutados ($poblacionPesosMutados$), este proceso se repite hasta que la población de individuos mutados tenga el mismo tamaño de $poblacionPesos$ ($NP$).

En la línea 8 se realiza la operación de cruce; para generar el primer individuo cruzado o de prueba $U_i$, se estable la Tasa de Cruce ($CR$) con un valor de 0.9 [29], además se obtiene el primer individuo de la población $X_i$ ($poblacionPesos$) y el primer individuo de la población de individuos cruzados $V_i$ ($poblacionPesosMutados$) y se realiza el cruce de acuerdo a la **Ecuación (5)**. Este proceso se repite hasta que

la población de individuos cruzados tenga el mismo tamaño de $poblacionPesos$ ($NP$).

En la línea 9, se aplica toda la población de vectores de pesos, ($poblacionPesosCruzados$) al individuo que se va optimizar para generar una población de nuevos individuos que dan solución al problema de calibración ($poblacionCruzados$), es decir, multiplicar cada uno de los vectores de pesos al individuo que se va optimizar, aplicando las restricciones pertinentes a estos nuevos individuos para evitar valores incoherentes. Este proceso es similar al explicado en la línea 3. En la línea 10, se calcula el valor de aptitud basado en NRMS para todos los individuos que conforman $poblacionCruzados$, la cual fue obtenida en línea 9, esta evaluación se explica en la sección 3.10.1.

En la línea 11, se realiza la operación de selección; aquí se seleccionan los mejores individuos entre la $poblacion$ y la población de individuos cruzados ($cruzados$) para formar la nueva $poblacion$. Dicha selección, se realiza comparando el valor de NRMS del primer individuo de la población original, con el primer individuo de la población de cruzados y de forma secuencial, para el resto de individuos de la población. Una vez seleccionado el individuo, se obtiene su respectivo peso aplicado para ingresarlo a la nueva población de pesos ($poblacionPesos$) que será la inicial para la siguiente generación.

En la línea 13, se actualiza la variable $contadorFEs$ de acuerdo al número de evaluaciones de la función objetivo realizadas. En la línea 15 se encuentra el mejor individuo de $población$. Si el individuo $inicial$ tiene un mejor valor de aptitud que el mejor individuo encontrado en la línea anterior, entonces el mejor individuo será $inicial$ (líneas 16, 17 y 18). La línea 19, retorna el mejor individuo optimizado ($Best$) con la ayuda de los vectores de pesos por Evolución Diferencial.

## 3.8 MUESTREO DE MÚLTIPLES DESCENDIENTES

En esta sección se explica la adaptación realizada al algoritmo meta-heurístico de Muestreo de Múltiples Descendientes (MOS) para la calibración de modelos de micro-simulación de flujo de tráfico vehicular CORSIM. Para tener una visión general de la meta-heurística, la **Figura 16** ilustra el pseudocódigo de MOS.

Pseudocódigo de la meta-heurística MOS

**Entrada:** Individuo $inicial$, Número de pasos $pasos$, Porcentaje de individuos $porcentajeIndividuos$
**Salida:** Mejor individuo $Best$
1: $Best = inicial$
2: $promedioAptitudSW = 0, promedioAptitudMTS = 0, i = 1$
3: **Mientras** no se cumpla la condición de parada ($i \leq pasos$) **hacer**
4: $EFOs =$ Calcular el número de evaluaciones de la función objetivo a cada técnica ($i, promedioAptitudSW, promedioAptitudMTS$)
5: $bestSW =$ Ejecutar la técnica Solis and Wets ($Best, EFOs$)

6: $bestMTS =$ Ejecutar la técnica MTS-LS1 $(Best, EFOs)$
7: $promedioAptitudSW =$ Calcular el promedio de aptitud de la técnica Solis and Wets $(porcentajeIndividuos)$
8: $promedioAptitudMTS =$ Calcular el promedio de aptitud de la técnica MTS-LS1 $(porcentajeIndividuos)$
9: **Si** $Aptitud\ de\ bestSW < Aptitud\ de\ bestMTS$ **entonces**
10: $Best = bestSW$
11: **Fin si**
12: **Sino**
13: $Best = bestMTS$
14: **Fin si**
15: $i++$
16: **Fin mientras**
17: **Retornar** $\boldsymbol{Best}$

***Figura 16:*** *Pseudocódigo de la meta-heurística MOS*

Luego de tener una visión general de la meta-heurística MOS, a continuación, se explica con más detalle cada una de las líneas del pseudocódigo.

En la línea 1, se toma el individuo inicial aplicando restricciones para evitar valores no permitidos. Cada parámetro a calibrar, tiene un rango previamente definido y es respetado en el proceso de creación de sus valores. Además, las variables son discretas (enteras) y los datos generados también cumplen con esa regla. A este individuo inicial, se le calcula el valor de aptitud basado en NRMS según la **Ecuación (1)**. Este individuo inicial se selecciona como la mejor solución actual $(Best)$.

En la línea 2, se inicializan las variables $promedioAptitudSW$ y $promedioAptitudMTS$ para guardar los promedios de aptitud de los individuos generados por las técnicas o algoritmos Solis and Wets y Búsqueda de Múltiple Trayectoria - Búsqueda Local 1 (MTS-LS1) respectivamente. La variable $i$ sirve para tener el control de los $pasos$ que va ejecutar MOS. Desde la línea 3 hasta la línea 16, se encuentra el ciclo para garantizar la ejecución de todos los $pasos$ (parámetro configurado por el usuario) cuya condición de parada es cumplir el número de $pasos$ que configure el usuario.

En la línea 4, se calcula el número de evaluaciones de la función objetivo $(EFOs)$, el cual le corresponde a cada técnica, de acuerdo a la mejora que logre hacer a $Best$ en el paso $i$, identificando cuál es la mejor técnica (la mejor técnica es aquella que tenga un valor promedio de aptitud NRMS menor).

En la línea 5, se ejecuta la técnica o algoritmo Solis and Wets para explotar $Best$, fijando el valor de $rho$ (parámetro configurado por el usuario), inicializando $\overrightarrow{bias}$ con el vector cero $(\vec{0})$ para empezar a orientar la búsqueda hacia el óptimo global y estableciendo el número de EFOs que le corresponde a esta técnica calculado en la línea 4. En la línea 6, se ejecuta la técnica o algoritmo MTS-LS1 para explotar $Best$, estableciendo el número de EFOs que le corresponde a esta técnica calculado en la línea 4.

En la línea 7, se calcula el promedio de aptitud (NRMS) de la técnica Solis and Wets, este promedio se calcula a un porcentaje de todos los valores de aptitud generados por la técnica Solis and Wets (este porcentaje es configurado por el usuario). En la línea 8, se calcula el promedio de aptitud (NRMS) de la técnica MTS-LS1, este promedio se calcula a un porcentaje de todos los valores de aptitud generados por la técnica MTS-LS1 (este porcentaje es configurado por el usuario).

Desde la línea 9 hasta la línea 14, se identifica cuál de las dos técnicas encontró un mejor individuo (aquel que tenga menor valor de aptitud NRMS), este individuo pasa a ser la mejor solución actual ($Best$).

En la línea 15 se indica que se ha cumplido un paso y la línea 17 retorna el mejor individuo ($Best$) encontrado por la meta-heurística MOS.

La principal característica de la meta-heurística MOS, es asignar dinámicamente el número de EFOs a cada técnica de acuerdo al rendimiento que hayan tenido en los $pasos$, por esta razón se muestra el pseudocódigo de esta función en la **Figura 17**.

Pseudocódigo de la función calcular número de EFOs de la meta-heurística MOS

**Entrada:** Paso $i$, Número de EFOs por paso $num$, Porcentaje de incremento/disminución $\varepsilon$, Promedio de aptitud de la peor técnica $Q_{max}$, Promedio de aptitud de la mejor técnica $Q_{min}$, Número de EFOs de la mejor técnica en el paso i $ratio$
**Salida:** Número de EFOs de la técnica Solis and Wets $EFOsSW$, Número de EFOs de la técnica MTS-LS1 $EFOsMTS$

1: **Si** $i = 1$ entonces
2: $EFOsSW = \frac{num}{2}$
3: $EFOsMTS = \frac{num}{2}$
4: **Fin si**
5: **Sino**
6: $NumeroEvaluaciones = \varepsilon^{*\frac{Q_{max}-Q_{min}}{Q_{max}}} * ratio$
7: A la mejor técnica sumarle a su número de EFOs el $NumeroEvaluaciones$
8: A la mejor peor restarle a su número de EFOs el $NumeroEvaluaciones$
9: **Fin si**
10: **Retornar $EFOsSW$ y $EFOsMTS$**

***Figura 17:*** *Pseudocódigo de la función calcular número de EFOs de la meta-heurística MOS*

Desde la línea 1 hasta la línea 4, es el caso para el primer paso, donde el número de EFOs por $paso$ (parámetros configurado por el usuario) se divide en la mitad para que las dos técnicas tengan igual EFOs.

Desde la línea 5 hasta la línea 9, es el caso para los demás pasos, el número EFOs por paso se van distribuyendo de acuerdo al rendimiento que haya tenido cada técnica en el paso $i$. Para esto se utiliza la **Ecuación (11)** para determinar cuántas evaluaciones se debe incrementar y/o disminuir a cada técnica (línea 6).

$$Número\ de\ Evaluaciones = \varepsilon * \frac{Q_{max} - Q_i}{Q_{max}} * ratio \qquad (11)$$

Donde:

$\varepsilon =$ Porcentaje de incremento/disminución del número de EFOs (parámetro configurado por el usuario).

$Q_{max} =$ Promedio de aptitud de la peor técnica (este promedio se calcula a un porcentaje de todos los valores de aptitud generados por la peor técnica, este porcentaje es configurado por el usuario).

$Q_i =$ Promedio de aptitud de la mejor técnica (este promedio se calcula a un porcentaje de todos los valores de aptitud generados por la mejor técnica, este porcentaje es configurado por el usuario).

$ratio =$ Número de EFOs de la mejor técnica en el $paso\ i$.

En las líneas 7 y 8, se actualiza el número de EFOs que puede ejecutar cada técnica en el paso $i$, la mejor técnica (la mejor técnica es aquella que tenga un valor promedio de aptitud NRMS menor) en el $paso\ i$, se le aumenta al número de EFOs el resultado de la **Ecuación (11)**. A la peor técnica (la peor técnica es aquella que tenga un valor promedio de aptitud NRMS mayor) en el $paso\ i$, se le disminuye al número de EFOs el resultado de la **Ecuación (11)**. En la línea 10 se retorna el número de EFOs para cada técnica en el $paso\ i$.

### 3.8.1 Adaptación del Algoritmo Solis and Wets

La representación de un individuo para este algoritmo es diferente, ya que representa un vector de genes que contienen una solución para el problema de optimización, además de tener una medida de aptitud NRMS (eficacia), un vector bias que indica la dirección de la búsqueda local para encontrar mejores soluciones, una desviación estándar (rho) que se utiliza en la Distribución Normal o Gaussiana dentro del algoritmo Solis and Wets para generar el vector de desviación que ayuda a cambiar la dirección del vector bias. La representación de un individuo, se ilustra en la **Figura 18**.

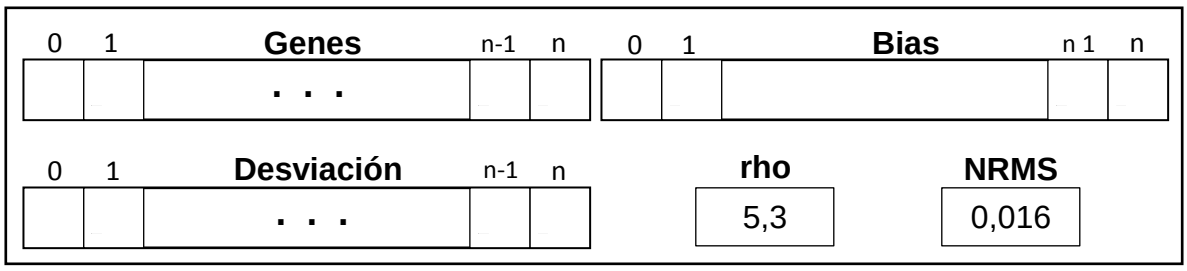



***Figura 18:*** *Representación de un individuo en el algoritmo Solis and Wets*

Como se observa en la **Figura 18**, los vectores de los genes, bias y desviación tienen el mismo tamaño *n*.

Para entender la forma que Solis and Wets genera nuevos individuos, la **Figura 19** muestra el vector de genes del individuo que se va a generar soluciones vecinas (individuo original).

| | | | | | | |
|---|---|---|---|---|---|---|
| $X_1$ | $X_2$ | $X_3$ | … | $X_i$ | … | $X_n$ |

***Figura 19:*** *Representación de los genes de un individuo*

Luego se calcula el rango ($\Delta_{xi} = X_{imax} - X_{imin}$) de cada una de los valores ($X_1$, $X_2$, $X_3$, …, $X_n$) del vector genes, formando el vector de rangos que muestra la **Figura 20**.

| | | | | | | |
|---|---|---|---|---|---|---|
| $\Delta_{x1}$ | $\Delta_{x2}$ | $\Delta_{x3}$ | … | $\Delta_{xi}$ | … | $\Delta_{xn}$ |

***Figura 20:*** *Representación de los rangos de los genes de un individuo*

Inicialmente, el vector bias es el vector cero ($\vec{0}$), este vector va cambiando de acuerdo a los éxitos que tenga la búsqueda a la hora de encontrar mejores soluciones. El vector desviación se inicializa con valores de la Distribución Normal o Gaussiana ($N(0, rho)$), cuyos parámetros son media igual a cero y desviación estándar ($rho$).

El algoritmo Solis and Wets genera individuos cercanos de dos maneras diferentes:

$$\overrightarrow{X'} = \vec{X} + \overrightarrow{bias} + \overrightarrow{desviación} \quad (12)$$

$$\overrightarrow{X'} = \vec{X} - \overrightarrow{bias} - \overrightarrow{desviación} \quad (13)$$

Donde:

$\overrightarrow{X'}$ = Vector de genes del nuevo individuo.

$\vec{X}$ = Vector de genes del individuo original.

$\overrightarrow{bias}$ = Vector bias del individuo original.

$\overrightarrow{desviación}$ = Vector desviación del individuo original.

Luego de conocer la forma de crear nuevos individuos por parte del algoritmo Solis and Wets, se explica la adaptación realizada al algoritmo Solis and Wets para la calibración de modelos de micro-simulación de flujo de tráfico vehicular CORSIM. Para tener una visión general del algoritmo, la **Figura 21** ilustra el pseudocódigo de Solis and Wets, tal como se adaptó para soportar el proceso de calibración.

Pseudocódigo del algoritmo Solis and Wets

**Entrada:** Tramo de intensidad $istr$, Individuo $inicial$, Tamaño del vecindario $N$
**Salida:** Mejor individuo $Best$
1: $Best = inicial$
2: $i = 0, exitos = 0, fallos = 0$

3: **Mientras** no se cumpla la condición de parada ($i < istr$) **hacer**
4: $vecindario =$ Generar vecindario ($N, Best$)
5: $S1 =$ Obtener mejor vecino ($vecindario$)
6: **Si** $Aptitud\ de\ S1 < Aptitud\ de\ Best$ **entonces**
7: **Si** $S1$ es generado por la Ecuación (12) **entonces**
8: $\overrightarrow{bias} = \left(0.2 * \overrightarrow{bias}\ de\ S1\right) + (0.4 * (\overrightarrow{desviación}\ de\ S1 + \overrightarrow{bias}\ de\ S1))$
9: **Fin si**
10: **Si** $S1$ es generado por la Ecuación (13) **entonces**
11: $\overrightarrow{bias} = \overrightarrow{bias}\ de\ S1 - (0.4 * (\overrightarrow{desviación}\ de\ S1 + \overrightarrow{bias}\ de\ S1))$
12: **Fin si**
13: $\overrightarrow{bias}\ de\ S1 = \overrightarrow{bias}$
14: $Best = S1$
15: $exitos++, fallos = 0$
16: **Fin si**
17: **Sino**
18: $fallos++, exitos = 0$
19: **Fin si**
20: **Si** $exitos > 2$ **entonces**
21: $rho\ de\ Best = 2 * rho\ de\ Best$
22: $exitos = 0$
23: **Fin si**
24: **Si** $fallos > 1$ **entonces**
25: $rho\ de\ Best = \frac{rho\ de\ Best}{2}$
26: **Si** $rho\ de\ Best < 1$ **entonces**
27: $rho\ de\ Best = 1$
28: **Fin si**
29: $fallos = 0$
30: **Fin si**
31: $i = i + N$
32: **Fin mientras**
33: **Retornar** $\boldsymbol{Best}$

***Figura 21:*** *Pseudocódigo del algoritmo Solis and Wets*

En la línea 1 se toma como mejor, el individuo inicial aplicando restricciones para evitar valores no permitidos. Cada parámetro a calibrar tiene un rango previamente definido y es respetado en el proceso de creación de sus valores. Además, las variables son discretas (enteras) y los datos generados también cumplen con esa regla. Este individuo inicial se toma como la mejor solución actual ($Best$).

En la línea 2 se inicializa la variable $i$ para controlar el número de EFOs que realiza el algoritmo y poder completar el tramo de intensidad $istr$, el cual es la condición de parada del ciclo que lo comprenden las líneas desde la 3 hasta la 32. Las variables $exitos$ y $fracasos$ corresponde a contadores que llevan el control del número de éxitos que tenga la búsqueda local a la hora de encontrar mejores soluciones (valor de aptitud NRMS menor) que $Best$ y el número de fracasos que tenga la búsqueda local a la hora de encontrar peores soluciones (valor de aptitud NRMS mayor) que $Best$.

En la línea 4 se genera un vecindario de individuos cercanos a partir del mejor individuo ($Best$), este vecindario tiene tamaño $N$ que es el número de hilos de

ejecución que configure el usuario (esta es una adaptación del algoritmo, ya que crea varios individuos cercanos a partir de $Best$). En la línea 5 se encuentra el mejor individuo del vecindario ($S1$). Desde la línea 6 hasta la línea 16 se identifica si $S1$ tiene una menor medida de aptitud que $Best$ (es mejor solución al problema), entonces se debe preguntar si $S1$ fue creado por la **Ecuación (12)** (línea 7 hasta línea 9) o si fue creado por la **Ecuación (13)** (línea 10 hasta línea 12). Si $S1$ fue creado por la **Ecuación (12)** entonces $\overrightarrow{bias}$ se actualiza usando la siguiente ecuación: (0.2 * $\overrightarrow{bias}$ + 0.4 * ($\overrightarrow{desviacion}$ + $\overrightarrow{bias}$)) (línea 8). Si $S1$ fue creado por la **Ecuación (13)** entonces $\overrightarrow{bias}$ se actualiza usando la siguiente ecuación: ($\overrightarrow{bias}$ - 0.4 * ($\overrightarrow{desviacion}$ + $\overrightarrow{bias}$)) (línea 11). Las constantes incluidas para actualizar el vector *bias* fueron tomadas de acuerdo a las investigaciones realizadas en [33]. En la línea 13 se establece $\overrightarrow{bias}$ a $S1$. En la línea 14, $S1$ pasa a ser la mejor solución actual ($Best$). En la línea 15 aumenta en uno el número de éxitos y se inicializa en cero el número de fracasos.

Las líneas 17, 18 y 19 es el caso de que $S1$ no es mejor solución que $Best$, entonces se aumenta en uno el número de fracasos y se inicializa en cero el número de éxitos (línea 18). Desde la línea 20 hasta la línea 23 se pregunta si la cantidad de éxitos consecutivos es mayor a dos, para duplicar el valor de la desviación estándar ($rho$) de la mejor solución actual (línea 21) y se inicializa en cero el número de éxitos (línea 22). Desde la línea 24 hasta la línea 30 se pregunta si el número de fallos consecutivos es mayor a uno para reducir a la mitad el valor de la desviación estándar ($rho$) de la mejor solución actual (línea 25). Si el valor de la desviación estándar ($rho$) de la mejor solución actual ($Best$) es menor a uno, entonces el valor de la desviación estándar se hace igual a uno (líneas 26, 27 y 28) y se inicializa en cero el número de fracasos (línea 29). Esto se debe a que hay parámetros de calibración que oscilan entre un rango pequeño de valores enteros

La línea 31 suma a la variable $i$ el valor de $N$ (tamaño del vecindario) creado en la línea 4, para indicar el número de EFOs que se realizaron y cumplir el tramo de intensidad $istr$. La línea 33 retorna el mejor individuo $Best$ encontrado por el algoritmo Solis and Wets.

La **Figura 22** muestra el pseudocódigo de la función generar vecindario del algoritmo Solis and Wets, permitiendo ver detalles del proceso de generación de nuevos individuos del vecindario (esta es una adaptación del algoritmo, ya que en la versión original se crea un solo vecino por ciclo, mientras que en esta investigación se crean varios individuos cercanos a partir de $Best$).

Pseudocódigo de la función generar vecindario del algoritmo Solis and Wets

**Entrada:** Tamaño del vecindario $N$, Individuo $inicial$
**Salida:** Vecindario $V$
1: $V = \emptyset, i = 0$

2: **Mientras** no se cumpla la condición de parada ($i < N$) **hacer**
3: $\overrightarrow{desviacion}$ = Distribución Gaussiana (0, $rho\ de\ inicial$)
4: $individuo1 = \overrightarrow{genes}\ de\ inicial + \overrightarrow{bias}\ de\ inicial + \overrightarrow{desviacion}$ // Ecuación (12)
5: Indicar que $individuo1$ fue creado por la Ecuación (12)
6: $\overrightarrow{desviacion}$ de $individuo1 = \overrightarrow{desviacion}$
7: Adicionar $individuo1$ a $V$
8: $individuo2 = \overrightarrow{genes}\ de\ inicial - \overrightarrow{bias}\ de\ inicial - \overrightarrow{desviacion}$ // Ecuación (13)
9: Indicar que $individuo2$ fue creado por la Ecuación (13)
10: $\overrightarrow{desviacion}$ de $individuo2 = \overrightarrow{desviacion}$
11: Adicionar $individuo2$ a $V$
12: $i = i + 2$
13: **Fin mientras**
14: Evaluar ($V$)
15: **Retornar** $V$

***Figura 22:*** *Pseudocódigo de la función generar vecindario del algoritmo Solis and Wets*

En la línea 1 se inicializa la lista donde se van almacenar los individuos que se van creando $V$. La variable $i$ es un contador para crear el número $N$ de individuos que es la condición de parada del ciclo que lo comprenden las líneas desde la 2 hasta la 13.

En la línea 3 se inicializa $\overrightarrow{desviacion}$ con valores de la Distribución Gaussiana, cuyos parámetros son media igual a cero, y la desviación estándar ($rho$) del individuo $inicial$. La línea 4 crea un individuo ($individuo1$) mediante la **Ecuación (12)**. A este nuevo individuo se le reparan los valores para cumplir con las restricciones de rango y tipo de dato. En la línea 5 se indica que $individuo1$ fue creado por la **Ecuación (12)**. La línea 6 establece $\overrightarrow{desviacion}$ a $individuo1$. La línea 7 adiciona el $individuo1$ al vecindario $V$. La línea 8 crea otro individuo ($individuo2$) mediante la **Ecuación (13)**. A este nuevo individuo se le reparan los valores para cumplir con las restricciones de rango y tipo de dato. En la línea 9 se indica que $individuo2$ fue creado por la **Ecuación (13)**. La línea 10 establece $\overrightarrow{desviacion}$ a $individuo2$. La línea 11 adiciona el $individuo2$ al vecindario $V$.

En la línea 12 se indica que fueron creados dos individuos, en la línea 14 se evalúa el vecindario, esta evaluación se explica en la sección 3.10.2. En la línea 15 se retorna el vecindario creado.

### 3.8.2 Adaptación del Algoritmo Búsqueda de Múltiple Trayectoria - Búsqueda Local 1

En esta sección se explica la adaptación realizada al algoritmo Búsqueda de Múltiple Trayectoria - Búsqueda Local 1 (MTS-LS1) para la calibración de modelos de micro-simulación de flujo de tráfico vehicular CORSIM. Para tener una visión general del algoritmo, la **Figura 23** ilustra el pseudocódigo de MTS-LS1.

Pseudocódigo del algoritmo MTS-LS1

**Entrada:** Número de evaluaciones de la función objetivo $EFOs$, Individuo $inicial$, Tamaño del vecindario $N$
**Salida:** Mejor individuo $Best$

1: $Best = inicial, i = 0$
2: **Mientras** no se cumpla la condición de parada ($i < EFOs$) **hacer**
3: $vecindario =$ Generar vecindario ($N, Best$)
4: $S1 =$ Obtener mejor vecino ($vecindario$)
5: **Si** $Aptitud\ de\ S1 < Aptitud\ de\ Best$ **entonces**
6: $Best = S1$
7: Indicar que $Best$ ha mejorado
8: **Fin si**
9: **Sino**
10: Indicar que $Best$ no ha mejorado
11: **Fin si**
12: $i = i + N$
13: **Fin mientras**
14: **Retornar** $Best$

***Figura 23:*** *Pseudocódigo del algoritmo MTS-LS1*

Al igual que en la línea 1 del algoritmo Solis and Wets, se toma como la mejor solución actual ($Best$) un individuo inicial al cual se le han aplicado sus respectivas restricciones. Además, se inicializa la variable $i$ que sirve para controlar el número de EFOs que realiza el algoritmo, esta es la condición de parada del ciclo que lo comprenden las líneas desde la 2 hasta la 13.

En la línea 3 se genera un vecindario de individuos cercanos a partir del mejor individuo ($Best$), este vecindario tiene tamaño $N$ que es el número de hilos de ejecución que configure el usuario (esta es una adaptación del algoritmo, ya que crea varios individuos cercanos a partir de $Best$). En la línea 4 se encuentra el mejor individuo del vecindario ($S1$). Desde la línea 5 hasta la línea 8 se identifica si $S1$ tiene una menor medida de aptitud que $Best$ (es mejor solución al problema). En la línea 6, $S1$ pasa a ser la mejor solución actual ($Best$) y en la línea 7 se indica que $Best$ ha mejorado, es decir tuvo una disminución es su valor de aptitud NRMS. Las líneas 9, 10 y 11 es el caso de que $S1$ no es mejor solución que $Best$, entonces se indica que $Best$ no ha mejorado (línea 10).

La línea 12 suma a la variable $i$ el valor de $N$ (tamaño del vecindario) creado en la línea 3, para indicar el número de EFOs que se realizaron y cumplir la condición de parada ($i < EFOs$). La línea 14 retorna el mejor individuo $Best$ encontrado por el algoritmo MTS-LS1.

La **Figura 24** muestra el pseudocódigo de la función generar vecindario del algoritmo MTS-LS1, que permite detallar la forma como se generan nuevos individuos en el vecindario (esta es una adaptación del algoritmo, ya que en el original sólo se crea uno y en este investigación se crean varios individuos cercanos a partir de $Best$).

Pseudocódigo de la función generar vecindario del algoritmo MTS-LS1

**Entrada:** Tamaño del vecindario $N$, Individuo $inicial$
**Salida:** Vecindario $V$
1: $V = \emptyset, i = 0$
2: **Mientras** no se cumpla la condición de parada ($i < N$) **hacer**
3: $individuo1 = inicial, individuo2 = inicial$
4: $posicion =$ Número aleatorio $(0, tamaño\ de\ inicial - 1)$
5: $SR = \frac{inicial[posicion]_{max} - inicial[posicion]_{min}}{2}$
6: **Si** $inicial$ no ha mejorado **entonces**
7: $SR = \frac{SR}{2}$
8: **Si** $SR < 2$ **entonces**
9: $SR = 2$
10: **Fin si**
11: **Fin si**
12: $individuo1[posicion] = inicial[posicion] - SR$
13: Adicionar $individuo1$ a $V$
14: $individuo2[posicion] = inicial[posicion] + \frac{SR}{2}$
15: Adicionar $individuo2$ a $V$
16: $i = i + 2$
17: **Fin mientras**
18: Evaluar ($V$)
19: **Retornar** $V$

***Figura 24:*** *Pseudocódigo de la función generar vecindario del algoritmo MTS-LS1*

Luego de tener una visión general de la función generar vecindario del algoritmo MTS-LS1, a continuación, se explica con más detalle cada una de las líneas del pseudocódigo.

En la línea 1 se inicializa la lista donde se van almacenar los individuos que se van creando $V$. La variable $i$ es un contador para crear el número $N$ de individuos que es la condición de parada del ciclo que lo comprenden las líneas desde la 2 hasta la 17.

En la línea 3 se crean dos copias de $inicial$ que se guardan en $individuo1$ e $individuo2$. En la línea 4 se escoge de forma aleatoria una posición del vector de los genes de $inicial$. La **Figura 25** muestra que la posición 3 fue la posición seleccionada aleatoriamente.

| *0* | *1* | *2* | *3* | *4* | *5* | *6* | *7* |
|---|---|---|---|---|---|---|---|
| 34 | 67 | 45 | 89 | 12 | 64 | 55 | 13 |

***Figura 25:*** *Selección de un gen aleatorio de un individuo*

En la línea 5 se calcula el intervalo de búsqueda $SR$ que es igual a la mitad del rango de la posición aleatoria escogida anteriormente, como lo muestra le **Ecuación (14)**:

$$SR = \frac{X_{3máximo} - X_{3mínimo}}{2} \qquad (14)$$

Desde la línea 6 hasta la línea 11 se pregunta si $inicial$ no ha mejorado, es decir que no ha disminuido su valor de aptitud NRMS, entonces el valor de $SR$ se reduce a la mitad (línea 7). Si el valor del intervalo de búsqueda ($SR$) es menor a dos, entonces el intervalo de búsqueda se hace igual a dos (líneas 8, 9 y 10). Esto se debe a que hay parámetros de calibración que oscilan entre un rango pequeño de valores enteros

En la línea 12 se modifica el gen de la posición seleccionada en la línea 4 del individuo $inicial$, restándole a este gen el valor de $SR$ (**Figura 26**). A este nuevo individuo se le reparan sus valores para que cumpla con las restricciones de rango y tipo, luego se adiciona al vecindario (línea 13).

| *0* | *1* | *2* | *3* | *4* | *5* | *6* | *7* |
|---|---|---|---|---|---|---|---|
| 34 | 67 | 45 | $89 - SR$ | 12 | 64 | 55 | 13 |

***Figura 26:*** *Generación de un nuevo individuo por el algoritmo MTS-LS1 (1 de 2)*

En la línea 14 se modifica el gen de la posición seleccionada en la línea 4 del individuo $inicial$, sumándole a este gen el valor de $\frac{SR}{2}$ (**Figura 27**). A este nuevo individuo se le reparan sus valores para que cumpla con las restricciones de rango y tipo, luego se adiciona al vecindario (línea 15).

| *0* | *1* | *2* | *3* | *4* | *5* | *6* | *7* |
|---|---|---|---|---|---|---|---|
| 34 | 67 | 45 | $89 + \frac{SR}{2}$ | 12 | 64 | 55 | 13 |

***Figura 27:*** *Generación de un nuevo individuo por el algoritmo MTS-LS1 (2 de 2)*

En la línea 16 se indica que fueron creados dos individuos, en la línea 18 se evalúa el vecindario, esta evaluación se explica en la sección 3.10.2. En la línea 19 se retorna el vecindario creado.

## 3.9 ADAPTACIÓN DEL ALGORITMO MA-SW-CHAINS

En esta sección se explica la adaptación realizada a la meta-heurística MA-SW-Chains para la calibración de modelos de micro-simulación de flujo de tráfico vehicular CORSIM. Para tener una visión general de la meta-heurística, la **Figura 28** ilustra el pseudocódigo de MA-SW-Chains.

Pseudocódigo de la meta-heurística MA-SW-Chains

**Entrada:** Tamaño de la población $NP$, Número de generaciones $numGeneraciones$, Generaciones en un óptimo local $GOptimoLocal$, Tramo de intensidad $istr$
**Salida:** Mejor individuo $Best$
1: $poblacion =$ Generar población inicial
2: evaluar ($poblacion$)
3: $i = 0, noMejora = 0$
4: **Mientras** no se cumpla la condición de parada ($i < numGeneraciones$) **hacer**

5: $seleccionados =$ Seleccionar $(poblacion)$
6: $cruzados =$ Cruzar $(seleccionados)$
7: $mutados =$ Mutar $(cruzados)$
8: evaluar $(mutados)$
9: $poblacion =$ Seleccionar los mejores individuos entre $poblacion$ y $mutados$
10: $Best =$ Obtener el mejor individuo $(poblacion)$
11: **Si** $noMejora = GOptimoLocal$ o a $Best$ no se le ha aplicado búsqueda local **entonces**
12: $rho\ de\ Best = \frac{Distancia\ al\ vecino\ más\ cercano\ (Best)}{2}$
13: **Si** $rho\ de\ Best < 1$ **entonces**
14: $rho\ de\ Best = 1$
15: **Fin si**
16: $\overrightarrow{bias}\ de\ Best = \vec{0}$
17: $noMejora = 0$
18: **Fin si**
19: $Best' =$ Ejecutar el algoritmo Solis and Wets $(Best, istr)$
20: **Si** $Aptitud\ de\ Best' = Aptitud\ de\ Best$ **entonces**
21: $noMejora++$
22: **Fin si**
23: **Sino**
24: $noMejora = 0$
25: **Fin si**
26: Reemplazar $Best$ por $Best'$ en $poblacion$
27: $i++$
28: **Fin mientras**
29: **Retornar** $\boldsymbol{Best}$

***Figura 28:*** *Pseudocódigo de la meta-heurística MA-SW-Chains*

Al igual que en la línea 1 de la meta-heurística DECC-G, se genera la población inicial de forma aleatoria. A todos los individuos de la población se les calcula el valor de aptitud basado en NRMS (línea 2), esta evaluación se explica en la sección 3.10.3. El tamaño de la población ($NP$) es un parámetro que se define previamente por parte del usuario.

En la línea 3 se inicializa la variable $i$ que sirve para controlar el número de generaciones $numGeneraciones$ que realiza la meta-heurística que es la condición de parada del ciclo que lo comprenden las líneas desde la 4 hasta la 28. En la líneas 5, 6 y 7 se generan nuevos individuos (la cantidad de individuos la determina los parámetros del algoritmo configurados por parte del usuario) utilizando los operadores genéticos de selección por torneo, cruce uniforme y mutación multigen [12]. Estos nuevos individuos cumplen las restricciones para evitar valores no permitidos. En la línea 8, se evalúan los nuevos individuos generados ($mutados$) basados en NRMS, esta evaluación se explica en la sección 3.10.3. Teniendo en cuenta, que se busca encontrar el menor valor de NRMS, se eliminan de la población los individuos con valores de aptitud altos. Se eliminan tantos individuos como sea necesario para mantener el tamaño de la población definida por el usuario (línea 9). En la línea 10 se selecciona el mejor individuo de la población (este individuo tiene la menor medida de aptitud porque se está buscando minimizar NRMS).

Desde la línea 11 hasta la línea 18, se establecen los componentes de desviación estándar ($rho$) y $\overrightarrow{bias}$ de $Best$ para realizar la búsqueda local encadenada, si el mejor individuo no ha mejorado su valor de aptitud durante el número de generaciones (parámetro del algoritmo configurado por parte del usuario, esta es una adaptación del algoritmo que permite romper el encadenamiento de búsquedas locales cuando se estanca en un óptimo local) o si a este mejor individuo $Best$ no se le ha aplicado previamente búsqueda local (línea 11), entonces se inicializan los componentes del mejor individuo con valores por defecto, en la línea 12 la desviación estándar ($rho$) es igual a la mitad de la distancia del vecino más cercano al mejor individuo de la población [33]. Esta distancia se calcula mediante la **Ecuación (15)**.

$$distancia = \sqrt{\sum_{i=0}^{n} (Best_i - individuoCercano_i)^2} \quad (15)$$

Donde:

$n =$ Es la cantidad de genes que tiene el mejor individuo $Best$.

$Best_i =$ Valor del gen en la posición $i$ del mejor individuo de la población.

$individuo_i =$ Valor del gen en la posición $i$ del individuo más cercano a $Best$.

Si el valor de la desviación estándar ($rho$) es menor a uno, se hace igual a uno debido a que hay parámetros de calibración que oscilan entre un rango pequeño de valores enteros (líneas 13, 14 y 15). En la línea 16, $\overrightarrow{bias}$ de $Best$ se inicializa con el vector cero. En la línea 17 la variable $noMejora$ se inicializa en cero.

En la línea 19 se explota al mejor individuo de la población ($Best$) con el algoritmo de búsqueda local Solis and Wets (sección 3.8.1) con un tramo de intensidad $istr$ ($istr$ es el número de evaluaciones de la función objetivo que ejecutará la búsqueda local y es previamente definido por el usuario). Esta explotación genera un nuevo individuo que se guarda en $Best'$, al cual se le indica que se le aplicó búsqueda local. Desde la línea 20 hasta la línea 25 se identifica si la búsqueda local mejoró el valor de aptitud del mejor individuo, si $Best'$ tiene igual valor de aptitud que $Best$, esto indica que la búsqueda local no mejoró el valor de aptitud de $Best$, por tal razón se aumenta en uno la variable $noMejora$ (líneas 20, 21 y 22), en caso contrario, si $Best'$ tiene un valor de aptitud menor que $Best$, esto indica que la búsqueda local mejoró el valor de aptitud de $Best$, por tal razón la variable $noMejora$ se inicializa en cero (líneas 23, 24 y 25). En la línea 26 se reemplaza el individuo $Best$ por $Best'$ en la $poblacion$.

En la línea 27 se indica que se ha cumplido una generación y la línea 29 retorna el mejor individuo ($Best$) encontrado por la meta-heurística MA-SW-Chains.

### 3.9.1 Adaptación del Algoritmo Genético

El algoritmo genético en MA-SW-Chains realiza la exploración del espacio de búsqueda, encontrando individuos prometedores para luego ser explotados con el algoritmo de búsqueda local Solis and Wets. Para realizar la exploración, es necesario generar varios individuos, en este caso, se utilizan los operadores genéticos de selección por torneo, cruce uniforme y mutación multigen recomendados en [12]. A continuación se explica el proceso de cada operador genético utilizado.

#### 3.9.1.1 Selección por torneo

La población de individuos es ordenada de forma ascendente (este problema consiste en minimizar la aptitud NRMS de los individuos), de acuerdo a su valor de aptitud, de esta manera el mejor individuo de la población va ser el primero (menor NRMS) y el peor individuo va ser el último (mayor NRMS). Suponiendo que el usuario ha configurado un 60% como porcentaje de selección y una población de 75 individuos de la población, entonces se calcula la posición de aceptación conforme a la **Ecuación (16)**.

$$Posición\ de\ Aceptación = \frac{Tamaño\ de\ la\ población * Porcentaje\ de\ selección}{100} \qquad (16)$$

$$Posición\ de\ Aceptación = \frac{75 * 60}{100} = 45$$

El valor de la posición de aceptación, en este caso 45, indica que se va seleccionar dos individuos contiguos de forma aleatoria entre los primeros 45 mejores individuos. La idea es seleccionar tantos individuos como el usuario configure.

#### 3.9.1.2 Cruce uniforme

Luego se realiza el cruce uniforme a los individuos seleccionados. Se recorre uno a uno los individuos seleccionados y se genera un número aleatorio, si este número aleatorio es mayor al porcentaje de cruce (parámetro configurado por el usuario), entonces no se realiza el cruce, de lo contrario, se cruza de la siguiente manera:

Se genera un vector máscara de forma aleatoria de ceros y unos del mismo tamaño del vector genes de los individuos, con una probabilidad del 50%, es decir la misma cantidad de ceros y unos, como lo ilustra la **Figura 29**.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 0 | … | 1 | 0 | 0 |

***Figura 29:*** *Representación de un vector máscara*

Se recorre cada posición del vector máscara, si el valor es igual a cero, los genes en esa posición de los dos individuos se mantienen. Si el valor de la posición de la máscara es igual a uno, se intercambian los valores de los genes en esa posición, como se ilustra en la **Figura 30** y **Figura 31**.

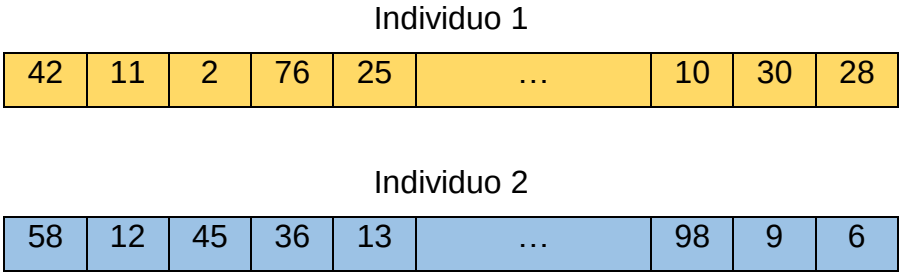



***Figura 30:*** *Representación de dos individuos antes de ser cruzados*

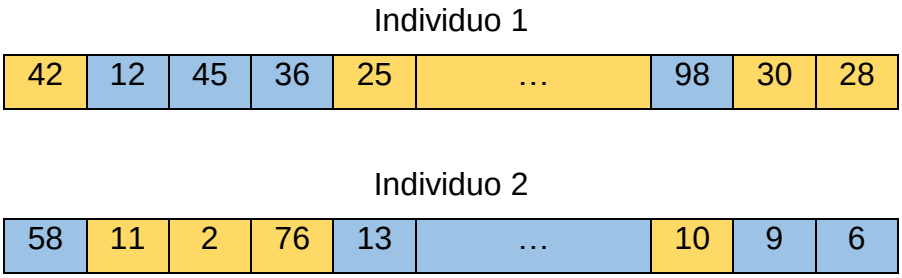



***Figura 31:*** *Representación de dos individuos después de ser cruzados*

#### 3.9.1.3 Mutación multigen

Luego se procede a realizar la mutación multigen a los individuos cruzados. Se recorre uno a uno todos los genes de todos los individuos cruzados, cada vez que se recorre un gen se genera un número aleatorio, si este número aleatorio es mayor al porcentaje de mutación (parámetro configurado por el usuario), entonces no se modifica este gen, de lo contrario, se muta de la siguiente manera:

Se genera un número aleatorio entre cero y uno, si este número es menor a 0.5, entonces a este gen se le va restar un valor de alteración, si este número es mayor a 0.5, entonces a este gen se le va sumar un valor de alteración. Para calcular este valor de alteración, se hace lo siguiente:

Calcular el rango al gen en cuestión, de acuerdo a la **Ecuación (17)**.

$$\Delta_{xi} = X_{imax} - X_{imin} \qquad (17)$$

Teniendo en cuenta el valor del rango, la alteración puede tomar dos valores según la **Ecuación (18)**.

$$Alteración = \begin{cases} 1, Si\ \Delta_{xi} \leq 10 \\ \dfrac{\Delta_{xi} * Paso\ de\ la\ mutación}{100}, en\ otro\ caso \end{cases} \qquad (18)$$

En caso de que el rango sea menor a 10, el valor de la alteración va tomar un valor de 1, ya que el rango indica que este gen varía muy poco y en caso de hacer un cambio, se debe realizar un cambio mínimo.

## 3.10 HILOS DE EJECUCIÓN

En este trabajo, el uso de hilos de ejecución es muy importante para hacer un mejor uso de los recursos hardware (procesador, velocidad de procesamiento, núcleos del

procesador, memoria RAM, CPU) del servidor facilitado por la Universidad de Nevada, Las Vegas. Además, los experimentos tienen una complejidad considerable, teniendo en cuenta el tiempo de ejecución de los algoritmos meta-heurísticos para completar la calibración. Para evitar el conflicto de acceder al mismo tiempo a un mismo espacio de la memoria, cada hilo de ejecución tiene un espacio de memoria RAM asignado mediante una copia del archivo con extensión *.trf* (modelo CORSIM). El cálculo del valor de aptitud NRMS de los individuos se realiza almacenando los parámetros del individuo en un archivo de extensión *.trf* para luego realizar la simulación en CORSIM.

Para los tres algoritmos meta-heurísticos, los hilos de ejecución en paralelo se utilizan en la evaluación de la función objetivo, ya que es la parte que requiere de más recursos computacionales debido a que realiza el llamado al simulador CORSIM. El número de hilos se establece en quince para hacer un buen uso del servidor y no llegar a saturarlo, es decir, la implementación de la herramienta garantiza al máximo lanzamientos de bloques de quince hilos para realizar evaluaciones de la función objetivo, aunque la aplicación software solicita al usuario el número de hilos que desea ejecutar.

Ahora se explica cómo se utilizan los hilos de ejecución en los algoritmos meta-heurísticos propuestos, vale aclarar que para esta sección $n$ es el número de hilos configurado por el usuario.

### 3.10.1 Uso de hilos en la meta-heurística DECC-G

En la meta-heurística DECC-G, tanto el algoritmo SaNSDE como Evolución Diferencial calculan el valor de aptitud NRMS de la población o conjunto de individuos mediante la ejecución de hilos en paralelo, formando bloques de ejecución de $n$ hilos, como lo muestra la **Ecuación (19)**. Por medio de esta ecuación se calcula el número de bloques de ejecución (cada uno de $n$ hilos) para evaluar la población. Sea $NP$ la cantidad de individuos de la población, si $NP < n$, entonces esta población se evaluará en un solo bloque de ejecución de $NP$ hilos. Además, sea $EFOs$ el número de evaluaciones que faltan por realizar, si $EFOs < \frac{NP}{2}$, entonces estas $EFOs$ evaluaciones se pierden, en caso contrario, $EFOs$ se iguala a $NP$ para alcanzar a realizar un bloque de ejecución adicional de $NP$ hilos.

$$Número\ de\ bloques\ de\ ejecución = \frac{Cantidad\ de\ individuos\ de\ la\ población}{Cantidad\ de\ hilos} \qquad (19)$$

### 3.10.2 Uso de hilos en la meta-heurística MOS

Como se mencionó anteriormente, la meta-heurística MOS utiliza dos técnicas para realizar la calibración, estas técnicas son Solis and Wets y Búsqueda de Múltiple Trayectoria - Búsqueda Local 1 (MTS-LS1).

La técnicas o algoritmos Solis and Wets y MTS-LS1 en MOS generan vecindarios de $n$ individuos para ser evaluados. El número de vecindarios generados se ve

limitado hasta agotar el número de evaluaciones de la función objetivo que se haya asignado a cada técnica.

Cuando MOS está asignando el número de evaluaciones de la función a cada técnica en los diferentes pasos, si alguna técnica llega a tener como número de evaluaciones de la función objetivo la mitad del número de hilos o menos ($EFOs \leq \frac{n}{2}$), entonces esta técnica se le asigna como número de evaluaciones de la función objetivo, la mitad del número de hilos más uno ($EFOs = \frac{n}{2} + 1$) y a la otra técnica se le asigna el número de evaluaciones de la función objetivo por paso menos la mitad del número de hilos más uno. Esto se hace para darle una mínima oportunidad a la peor técnica y no desecharla por completo.

En la ejecución de algunas de las dos técnicas, el número de evaluaciones de la función objetivo que faltan por realizar es menor a la mitad del número de hilos ($EFOs < \frac{n}{2}$), entonces estas evaluaciones se pierden, en caso contrario, $EFOs$ se iguala a $n$ para alcanzar a realizar un bloque de ejecución adicional de $n$ hilos.

### 3.10.3 Uso de hilos en la meta-heurística MA-SW-Chains

En la meta-heurística MA-SW-Chains, el algoritmo genético calcula el valor de aptitud NRMS de la población o conjunto de individuos mediante la ejecución de hilos en paralelo, formando bloques de ejecución de $n$ hilos, tal como evalúa la población la meta-heurística DECC-G (sección 3.10.1). Para realizar la explotación de las soluciones más prometedoras, MA-SW-Chains utiliza el algoritmo Solis and Wets y el uso de hilos para esta búsqueda local se explica en la sección 3.10.2.

### 3.10.4 Uso de hilos en GASA y SPSA

Para realizar una comparación justa entre los algoritmos meta-heurísticos propuestos y los algoritmos del estado del arte (GASA y SPSA), a los algoritmos del estado del arte también se implementó el uso de hilos en paralelo.

En el algoritmo memético GASA, el algoritmo genético calcula el valor de aptitud NRMS de la población o conjunto de individuos mediante la ejecución de hilos en paralelo, formando bloques de ejecución de $n$ hilos, tal como evalúa la población la meta-heurística DECC-G (sección 3.10.1). Para realizar la explotación de las soluciones más prometedoras, GASA utiliza el algoritmo Recocido Simulado, utilizando el mismo proceso que los algoritmos Solis and Wets y MTS-LS1.

El algoritmo SPSA genera de forma paralela $\frac{n}{3}$ soluciones y cada una de estas soluciones ejecuta tres evaluaciones que corresponden a modelos que genera SPSA ($\frac{n}{3} * 3 = n$) (así es la lógica de este algoritmo), alcanzando el mismo número de hilos de ejecución en paralelo que los demás algoritmos (DECC-G, MOS, MA-SW-Chains y GASA).

# Capítulo 4

## 4 EXPERIMENTACIÓN

En este capítulo se presentan los modelos CORSIM utilizados para la evaluación y comparación de los algoritmos meta-heurísticos propuestos con los algoritmos del estado del arte (GASA y SPSA). Además, se muestra el afinamiento de parámetros realizado a las meta-heurísticas propuestas con el fin de obtener mejores resultados de calibración. Luego de esto, se muestran los resultados obtenidos teniendo en cuenta la función propuesta por Paz et al que contempla las variables de volumen y velocidad, realizando un análisis minucioso de los resultados. Después, se muestra la generación de datos para el componente de longitud de colas, la manera de seleccionar los pesos adecuados para los tres componentes para poner a prueba los algoritmos en la función propuesta en esta investigación que contempla los componentes de volumen, velocidad y longitud de colas.

La ejecución de los experimentos fue realizada en un servidor facilitado por la Universidad de Nevada, Las Vegas, Estados Unidos, cuyas características son las siguientes: Windows Server 2008 R2 de 64 bits; Procesador: Intel Xeon CPU X7560 2.26 GHz de 64 núcleos; Memoria RAM: 256 Gigabytes DDR3; Software: Java Virtual Machine V1.7.60.

### 4.1 DISEÑO DE LOS EXPERIMENTOS

Para poner a prueba los tres algoritmos meta-heurísticos propuestos, se definen tres experimentos de complejidad baja, media y alta. Esta complejidad se determina en primera medida, de acuerdo a los recursos computacionales consumidos por parte de los algoritmos meta-heurísticos DECC-G, MOS, MA-SW-Chains para calibrar cada uno de los modelos. Además, igual que en [12], también se tienen en cuenta la cantidad de enlaces que tenga cada modelo, en general, los modelos que tienen entre 0 y 99 enlaces, se consideran modelos de complejidad baja; entre 100 y 300 enlaces, modelos de complejidad media, y más de 300 enlaces, modelos de complejidad alta.

El conjunto de parámetros a calibrar en cada modelo están relacionados con el comportamiento del conductor y el rendimiento del vehículo, estos parámetros se encuentran en [21] y también se utilizan en [12]. Para realizar el proceso de calibración, se toman las salidas de la simulación relacionadas con el conteo de vehículos (volumen) medidos como la cantidad de vehículos por hora (vph), la velocidad de los vehículos medida en millas por hora (mph) y la longitud de las colas medidas en número de vehículos. El mejor conjunto de parámetros calibrados son los que logran disminuir la diferencia entre los valores simulados y los datos de campo, además de cumplir el criterio de calibración GEH. Los datos de campo (volumen y velocidad) y los modelos fueron facilitados por el Departamento de

Transportes de Nevada (NDOT por sus siglas en inglés). Cabe resaltar que para los modelos facilitados, no se contó con los datos de campo de longitud de colas, por lo cual fue necesario crear un procedimiento (sección 4.4) para la generación de estos datos de campo en las diferentes redes.

### 4.1.1 Modelo de complejidad baja: McTrans network

Este modelo facilitó el afinamiento de parámetros para los tres algoritmos meta-heurísticos propuestos, debido a su baja complejidad y a que requiere menos tiempo de ejecución para realizar la calibración, además de servir para definir los pesos para los componentes de volumen, velocidad y longitud de colas en la función objetivo. Este modelo contiene 20 enlaces. Teniendo en cuenta los datos de volumen y velocidad, el valor inicial de NRMS es de 0.291182. Incluyendo los valores de longitud de colas, NRMS inicia con 0.456479. La **Figura 32** muestra el modelo de baja complejidad.

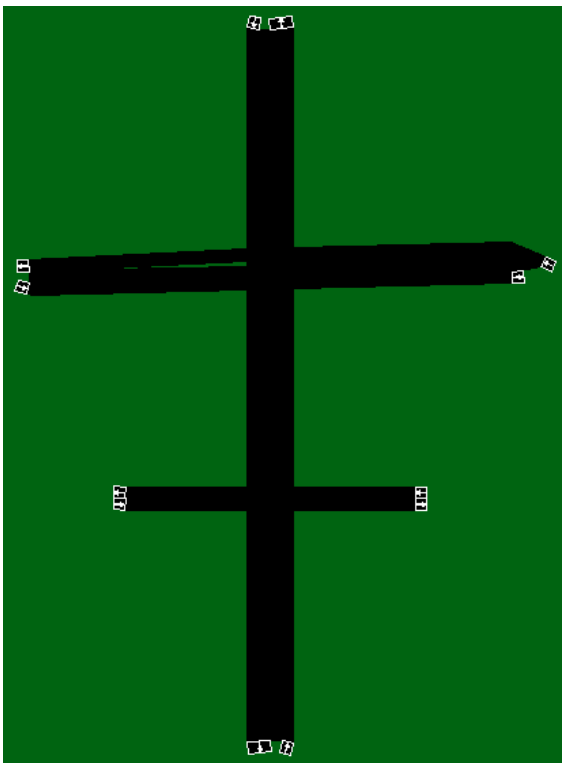


***Figura 32:*** *Modelo McTrans Viewer en el software TRAFVU*

### 4.1.2 Modelo de complejidad media: Reno network

Este modelo CORSIM representa la autopista Pyramid en Reno, Nevada, Estados Unidos. Este modelo contiene 126 enlaces y dispone datos de campo para 45 (volumen y velocidad) de los enlaces. Teniendo en cuenta los datos de volumen y velocidad, el valor inicial de NRMS es de 0.221791. Incluyendo los valores de longitud de colas, NRMS inicia con 1.424487. La **Figura 33** muestra el modelo de complejidad media.


***Figura 33:*** *Autopista Pyramid en Reno, Nevada. Tomada de Google Maps y TRAFVU Viewer*

### 4.1.3 Modelo de complejidad alta: I-75 network

Este modelo CORSIM representa una porción de la autopista interestatal I-75 en Miami, Florida, Estados Unidos. Este modelo contiene 703 enlaces y dispone de datos de campo para 346 (volumen) de los enlaces. Los datos de campo de velocidad fueron simulados a partir de los parámetros por defecto ya que para este modelo no existen datos reales de velocidad. Teniendo en cuenta los datos de volumen y velocidad, el valor inicial de NRMS es de 0.331824. La **Figura 34** muestra el modelo de complejidad alta.

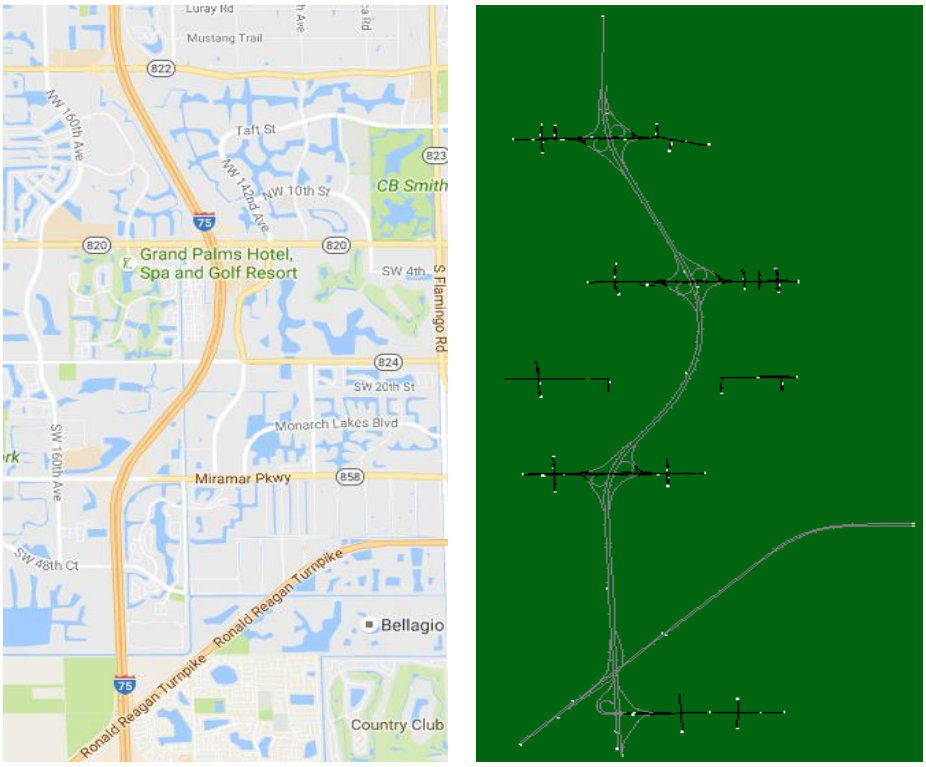



***Figura 34:*** *Autopista I-75 en Miami, Florida. Tomada de Google Maps y TRAFVU Viewer*

## 4.2 AFINAMIENTO DE PARÁMETROS

El objetivo de afinar los parámetros de los algoritmos meta-heurísticos DECC-G, MOS y MA-SW-Chains es alcanzar una mejor calibración en cada uno de los experimentos. Para realizar este afinamiento de parámetros, las pruebas se realizaron en el modelo de complejidad baja (McTrans Network), ya que este experimento demora menos tiempo en realizar la calibración y consume menos recursos computacionales.

Para realizar el afinamiento de parámetros de MOS y MA-SW-Chains, se utilizaron Arreglos de Cobertura (CA por sus siglas en inglés). Los Arreglos de Cobertura son estructuras combinatorias especificadas como una matriz de $N$ filas y $k$ columnas sobre un alfabeto de $v$ símbolos tales que para cada conjunto de $t$ columnas (llamadas la fuerza del arreglo) cubre cada t-tupla de símbolos. El número de filas del Arreglo de Cobertura indica el número de experimentos que se realizarán, este número de pruebas depende de la cantidad de columnas ($k$) que indican los parámetros que intervienen en la prueba, el vocabulario seleccionado para la prueba ($v$), este vocabulario indica la cantidad de valores que se probarán para cada parámetro y la fuerza de la prueba ($t$) que indica la cantidad de columnas que se agrupan para la posterior combinación. Por lo anterior, un Arreglo de Cobertura de denota como $CA(t, k, v)$. Para alcanzar a cumplir con el tiempo, la fuerza de la prueba fue igual a dos ($t = 2$) ya que garantiza un número de pruebas que se

pueden realizar para alcanzar a cumplir los objetivos de esta investigación en el tiempo definido para la misma.

Recientemente han sido utilizados para representar conjuntos de pruebas de interacción para pruebas de software dado que proporcionan casos de prueba de tamaño económico mientras se conservan importantes capacidades de detección de fallos. Los Arreglos de Cobertura buscan tener cardinalidad mínima para reducir al mínimo el número de pruebas y cobertura al máximo para garantizar cubrir todas las combinaciones de cierto tamaño (nivel de fuerza o interacción) entre los parámetros de entrada [54].

Como cada algoritmo meta-heurístico tiene diferente número de parámetros para afinar, es necesario aplicar un arreglo de cobertura diferente a cada algoritmo. La fuerza igual a dos y el alfabeto de tres símbolos, son las componentes en común que tienen los arreglos de cobertura aplicados a cada algoritmo. Estos valores de la fuerza y el alfabeto de símbolos fueron seleccionados de acuerdo al tiempo tardado (aproximadamente de 6 a 7 horas) en ejecutar 30 veces cada una de las pruebas que resultaron de los arreglos de cobertura sobre el mismo modelo. Las 30 ejecuciones permiten tener valores promedio que se comporten como una distribución normal como lo establece el teorema del límite central en estadística.

### 4.2.1 Afinamiento de parámetros para DECC-G

Para el algoritmo meta-heurístico DECC-G, no se aplicó un Arreglo de Cobertura directamente por la naturaleza de la meta-heurística, ya que al realizar las combinaciones de los diferentes valores que pueden tomar los diferentes parámetros del algoritmo, la meta-heurística no realizaba el número de evaluaciones de la función objetivo esperado (entre 5200 y 5400) y que permite hacer una comparación justa con los demás algoritmos. Por esta razón se hicieron algunas pruebas aleatorias para realizar el afinamiento de parámetros de DECC-G. La **Tabla 1** muestra las pruebas realizadas para afinar los parámetros de DECC-G, con los valores que tomaron los parámetros y el valor de promedio NRMS de cada prueba.

| PRUEBA | PARÁMETROS | | | | PROMEDIO NRMS |
|---|---|---|---|---|---|
| | Ciclos | Subcomponentes | Evaluaciones de SaNSDE | Evaluaciones de DE | |
| 1 | 3 | 6 | 165 | 255 | 0.016744 |
| **2** | **5** | **4** | **195** | **75** | **0.016301** |
| 3 | 3 | 6 | 240 | 105 | 0.018159 |
| 4 | 4 | 5 | 210 | 90 | 0.016584 |
| 5 | 3 | 9 | 150 | 120 | 0.017728 |

***Tabla 1:*** *Afinamiento de parámetros de DECC-G*

Se observa que la prueba dos obtuvo un mejor valor de NRMS (se está minimizando), por tal razón, los valores de los parámetros en la prueba dos son los más adecuados para realizar la calibración de los diferentes modelos con el algoritmo meta-heurístico DECC-G, estos parámetros son los siguientes:

- Tamaño de la población: 15
- Número de ciclos: 5
- Número de subcomponentes: 4
- Número de evaluaciones de la función objetivo para SaNSDE: 195
- Periodo de aprendizaje de $p$ y $fp$ = 60
- Periodo de aprendizaje de $CR$ = 30
- Número de evaluaciones de la función objetivo para Evolución Diferencial = 75
- Factor de Escala ($F$) = 0.5
- Tasa de Cruce ($CR$) = 0.9

El tamaño de la población, periodo de aprendizaje de $p$ y $fp$, Periodo de aprendizaje de $CR$, Factor de Escala y Tasa de Cruce se tomaron directamente de los reportados en la literatura en [29].

### 4.2.2 Afinamiento de parámetros para MOS

Para el algoritmo meta-heurístico MOS se define un Arreglo de Cobertura de fuerza igual a dos, cuatro parámetros y un vocabulario de tres $CA(2,4,3)$, esto originó un arreglo con 12 pruebas. La **Tabla 2** muestra el arreglo de cobertura utilizado.

| | | | |
|---|---|---|---|
| 0 | 0 | 1 | 2 |
| 0 | 1 | 1 | 0 |
| 0 | 2 | 0 | 0 |
| 1 | 0 | 2 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 2 | 1 | 2 |
| 2 | 0 | 0 | 1 |
| 2 | 1 | 2 | 2 |
| 2 | 2 | 1 | 1 |
| 0 | 2 | 2 | 1 |
| 2 | * | * | 0 |
| * | * | 0 | 2 |

***Tabla 2:*** *$CA(2,4,3)$ para MOS*

Los asteriscos (*) son comodines en el arreglo, lugares que pueden tomar cualquier valor permitido del vocabulario del arreglo de cobertura. La **Tabla 3** muestra el vocabulario de este arreglo.

| **PARÁMETROS** | **VALORES** | | |
|---|---|---|---|
| (1) Número de pasos | 11 | 16 | 21 |
| (2) Porcentaje de incremento de evaluaciones de la función objetivo | 10 | 25 | 40 |
| (3) Porcentaje de individuos | 10 | 25 | 40 |
| (4) rho | 60 | 95 | 130 |

***Tabla 3:*** *Vocabulario para MOS*

Reemplazando el vocabulario en el arreglo de cobertura y adicionando el promedio del valor NRMS generado por cada prueba, se obtiene el Arreglo de Cobertura para MOS (**Tabla 4**).

| **PRUEBA** | **PARÁMETROS** | | | | **PROMEDIO NRMS** |
|---|---|---|---|---|---|
| | **(1)** | **(2)** | **(3)** | **(4)** | |
| 1 | 11 | 10 | 25 | 130 | 0.021121 |
| 2 | 11 | 25 | 25 | 60 | 0.023880 |
| 3 | 11 | 40 | 10 | 60 | 0.022819 |
| 4 | 16 | 10 | 40 | 60 | 0.024987 |
| 5 | 16 | 25 | 10 | 95 | 0.021099 |
| 6 | 16 | 40 | 25 | 130 | 0.022819 |
| 7 | 21 | 10 | 10 | 95 | 0.023076 |
| 8 | 21 | 25 | 40 | 130 | 0.023269 |
| **9** | **21** | **40** | **25** | **95** | **0.019762** |
| 10 | 11 | 40 | 40 | 95 | 0.021279 |
| 11 | 21 | 10 | 25 | 60 | 0.248898 |
| 12 | 21 | 40 | 10 | 130 | 0.022656 |

***Tabla 4:*** *Arreglo de Cobertura para MOS*

Se observa que la prueba nueve obtuvo un mejor valor de NRMS, por tal razón, los valores de los parámetros en la prueba nueve y sus pequeñas variaciones son los más adecuados para realizar la calibración de los diferentes modelos con el algoritmo MOS, estos parámetros son los siguientes:

- Número de evaluaciones de la función objetivo por paso: 255
- Número de pasos: 21

- Porcentaje de incremento de evaluaciones de la función objetivo: 40%
- Porcentaje de individuos: 25%
- rho = 95

Cabe resaltar que el número de evaluaciones de la función objetivo por paso no se tuvo en cuenta en el arreglo de cobertura, ya que este parámetro es calculado a partir del número de pasos para mantener el tiempo estimado en la literatura para la calibración de cada uno de los modelos de prueba.

### 4.2.3 Afinamiento de parámetros para MA-SW-Chains

Para el algoritmo MA-SW-Chains se define un Arreglo de Cobertura de fuerza igual a dos, cinco parámetros y un vocabulario de tres $CA(2,5,3)$, esto originó un arreglo de trece pruebas. La **Tabla 5** muestra el arreglo de cobertura utilizado.

| | | | | |
|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 2 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 2 | 0 | 0 | 0 |
| 1 | 0 | 2 | 0 | 0 |
| 1 | 1 | 0 | 1 | 2 |
| 1 | 2 | 1 | 2 | 1 |
| 2 | 0 | 0 | 1 | 1 |
| 2 | 1 | 2 | 2 | 0 |
| 2 | 2 | 1 | 1 | 0 |
| 0 | 2 | 2 | 1 | 2 |
| 2 | * | * | 0 | 2 |
| * | * | 0 | 2 | * |
| * | * | 2 | * | 1 |

***Tabla 5:*** *$CA(2,5,3)$ para MA-SW-Chains*

La **Tabla 6** muestra el vocabulario de este arreglo:

| **PARÁMETROS** | **VALORES** | | |
|---|---|---|---|
| (1) Tamaño de la Población | 45 | 60 | 75 |
| (2) Porcentaje de Selección | 50 | 60 | 70 |
| (3) Porcentaje de Cruce | 65 | 75 | 85 |
| (4) Porcentaje de Mutación | 5 | 7 | 10 |
| (5) Generaciones en un óptimo local | 1 | 2 | 3 |

***Tabla 6:*** *Vocabulario para MA-SW-Chains*

Reemplazando el vocabulario en el arreglo de cobertura y adicionando el promedio del valor NRMS generado por cada prueba, se obtiene el Arreglo de Cobertura para MA-SW-Chains (**Tabla 7**).

| PRUEBA | PARÁMETROS | | | | | PROMEDIO NRMS |
|---|---|---|---|---|---|---|
| | (1) | (2) | (3) | (4) | (5) | |
| 1 | 45 | 50 | 75 | 10 | 3 | 0.022376 |
| 2 | 45 | 60 | 75 | 5 | 2 | 0.023268 |
| 3 | 45 | 70 | 65 | 5 | 1 | 0.028648 |
| 4 | 60 | 50 | 85 | 5 | 1 | 0.023356 |
| 5 | 60 | 60 | 65 | 7 | 3 | 0.029046 |
| 6 | 60 | 70 | 75 | 10 | 2 | 0.021703 |
| 7 | 75 | 50 | 65 | 7 | 2 | 0.027145 |
| **8** | **75** | **60** | **85** | **10** | **1** | **0.021011** |
| 9 | 75 | 70 | 75 | 7 | 1 | 0.022438 |
| 10 | 45 | 70 | 85 | 7 | 3 | 0.025338 |
| 11 | 75 | 60 | 85 | 5 | 3 | 0.026131 |
| 12 | 60 | 50 | 65 | 10 | 1 | 0.022551 |
| 13 | 60 | 50 | 85 | 5 | 2 | 0.023905 |

***Tabla 7:*** *Arreglo de Cobertura para MA-SW-Chains*

Se observa que en la prueba ocho, se obtuvo un mejor valor de NRMS, por tal razón, los valores de los parámetros en la prueba ocho son los más adecuados para realizar la calibración de los diferentes modelos con el algoritmo MA-SW-Chains, estos parámetros corresponden a:

- Tamaño de la población: 75
- Número de generaciones: 16
- Número de padres: 15
- Porcentaje de selección: 60%
- Porcentaje de cruce: 85%
- Porcentaje de mutación: 10%
- Paso de mutación: 2
- Tramo de intensidad ($istr$): 300
- Generaciones en un óptimo local: 1

El número de generaciones y el tramo de intensidad no se tuvieron en cuenta en el arreglo de cobertura, ya que con los anteriores valores de estos dos parámetros garantizan un número de evaluaciones de la función objetivo que al evaluarlas no

supera el tiempo estimado en la literatura para la calibración de cada uno de los modelos de prueba. El criterio que se tuvo en cuenta para conformar el arreglo de cobertura, fue seleccionar los parámetros que tienen un mayor impacto en la efectividad del algoritmo según la literatura, por tal motivo se descartó el número de padres y el paso de mutación, permitiendo además un conjunto de pruebas más reducido y efectivo.

## 4.3 RESULTADOS OBTENIDOS Y COMPARACIÓN

A continuación, se presentan los resultados obtenidos por los algoritmos meta-heurísticos propuestos en los modelos CORSIM presentados previamente. En esta etapa de experimentación, se contempló un rango de iteraciones entre 5200 y 5400 para todos los algoritmos y el uso de hilos de ejecución para obtener un mayor rendimiento de cada uno de ellos. Para determinar este rango del número de evaluaciones de la función objetivo, se consideró el tiempo de ejecución que tardan los algoritmos del estado del arte para realizar la calibración de los modelos CORSIM y el número de hilos de ejecución con que se ejecutan las meta-heurísticas (estas pruebas fueron realizadas con quince hilos de ejecución en paralelo). Una vez obtenida la mejor instancia de cada meta-heurística se realizaron 30 ejecuciones con el fin de cumplir con el teorema del límite central [55] y evidenciar el comportamiento de los algoritmos bajo diferentes valores aleatorios definidos durante la ejecución de la meta-heurística sobre los modelos de prueba para la función objetivo.

La primera fase de la experimentación se centró en evaluar los algoritmos meta-heurísticos propuestos, teniendo en cuenta solamente los componentes de volumen y velocidad en la función objetivo. Para ello, se asignó un peso de 0.7 (70%) para el componente de volumen, 0.3 (30%) para velocidad y 0% para el componente de longitud de colas. Estos pesos fueron utilizados en [12] y se utilizan en la presente investigación para realizar una comparación justa entre los algoritmos propuestos y los presentados en el estado del arte. En la segunda fase de la experimentación se realizó un análisis para determinar los pesos de los tres componentes de la función objetivo y se ejecutó nuevamente los algoritmos de la misma forma que en la primera fase, es decir, cada algoritmo fue ejecutado nuevamente en la función objetivo con los nuevos pesos 30 veces y con los parámetros definidos previamente en los arreglos de cobertura para cada uno de ellos. Las redes en las cuales se puso a prueba cada uno de los algoritmos meta-heurísticos para la segunda fase fueron McTrans y Reno. En la red I-75 no se utilizó la función objetivo con un peso al componente de longitud de colas ya que no se contó con datos de campo (FRESIM) en el componente de longitud de colas y ya los datos de velocidad habían sido definidos artificialmente.

Para comparar los resultados obtenidos por todos los algoritmos, se utilizaron dos métodos de análisis estadístico mediante el software Keel [56], las pruebas no paramétricas de Wilcoxon y Friedman.

### 4.3.1 Resultados y comparación para el modelo McTrans con los componentes de volumen y velocidad

A continuación se presentan los resultados obtenidos por cada uno de los algoritmos en el modelo de complejidad baja.

#### 4.3.1.1 Resultados obtenidos con DECC-G

Para realizar la ejecución del algoritmo meta-heurístico DECC-G con el modelo McTrans y obtener los resultados finales, se tomaron los parámetros que mejor dieron resultado en el afinamiento de parámetros descrito en la sección 4.2.1 y se realizaron un total de 30 ejecuciones del algoritmo. Los parámetros utilizados fueron: tamaño de la población: 15, ciclos = 5, número de subcomponentes = 4, número de evaluaciones de la función objetivo para SANSDE = 195, periodo de aprendizaje de $p$ y $fp$ cada 60 evaluaciones de la función objetivo y el periodo de aprendizaje de $CR$ cada 30, numero de evaluaciones de la función objetivo para Evolución Diferencial = 75, factor de escala ($F$)= 0.5 y tasa de cruce ($CR$) = 0.9.

La ejecución de las 30 pruebas realizadas iniciaron con un NRMS de 0.291182, en ellas se obtuvo un promedio de NRMS de 0.016301, el mejor valor encontrado fue de 0.013634 y el peor valor encontrado fue de 0.019504, estos resultados arrojan una desviación estándar de 0.001393 lo cual indica poca dispersión en los valores de las 30 ejecuciones con respecto al promedio de NRMS. Teniendo en cuenta el valor inicial y promedio de NRMS, DECC-G logra una mejora del 94.4% aproximadamente. Con respecto a los tiempos obtenidos, el promedio de tiempo para cumplir el criterio de parada del algoritmo fue de 11:16, el mejor tiempo registrado fue de 11:03 y el peor tiempo fue de 11:31, lo cual permite observar a nivel general que el algoritmo cumple con el criterio de parada en un estimado de 11 minutos para la red de complejidad baja.

#### 4.3.1.2 Resultados obtenidos con MOS

Para la ejecución del algoritmo MOS en la red McTrans se definieron los siguientes parámetros en la sección 4.2.2: número de evaluaciones de la función objetivo por paso = 255, número de pasos = 21, porcentaje de incremento de evaluaciones de la función objetivo = 40, porcentaje de individuos = 25 y rho = 95.

El promedio de NRMS de ejecutar el algoritmo meta-heurístico en 30 pruebas fue de 0.019762, el mejor valor de NRMS encontrado fue de 0.011759 y el peor encontrado fue 0.027190, estos resultados arrojan una desviación estándar de 0,003994 lo cual indica poca dispersión en los valores de las 30 ejecuciones con respecto al promedio de NRMS. Teniendo en cuenta el valor inicial y promedio de NRMS, MOS logra una mejora del 93.21% aproximadamente. El tiempo promedio de ejecución del total de experimentos fue de 13:28, el mejor tiempo registrado fue de 13:12 y el peor fue de 13:50.

#### 4.3.1.3 Resultados obtenidos con MA-SW-Chains

El conjunto de parámetros que se definieron en la sección 4.2.3 para ejecutar el algoritmo MA-SW-Chains fueron: tamaño de la población = 75, número de

generaciones = 16, número de padres = 15, porcentaje de selección = 60, porcentaje de cruce = 85, porcentaje de mutación = 10, paso de mutación = 2, tramo de intensidad = 300 y generaciones en un óptimo local = 1. Bajo esta configuración de parámetros el comportamiento del algoritmo meta-heurístico, se realizaron 30 ejecuciones.

Con las diferentes ejecuciones se obtuvo un promedio de NRMS de 0.021011, el mejor valor de NRMS registrado fue de 0.014450 y el peor de 0.037628, estos resultados arrojan una desviación estándar de 0.005168 lo cual indica poca dispersión en los valores de las 30 ejecuciones con respecto al promedio de NRMS. Teniendo en cuenta el valor inicial y promedio de NRMS, MA-SW-Chains logra una mejora del 92.78% aproximadamente. Por otro lado, los registros referentes al tiempo de ejecución del algoritmo determinaron un promedio de 12:09, el mejor y peor tiempo registrado fueron 11:56 y 14:02 respectivamente.

#### 4.3.1.4 Comparación de los algoritmos propuestos con GASA y SPSA

Con respecto a los algoritmos referenciados en el estado del arte GASA y SPSA, con cada uno de ellos se realizó 30 ejecuciones para este modelo de complejidad baja, donde GASA obtuvo un promedio de NRMS de 0.023591, el mejor valor encontrado fue de 0.011926 y el peor valor encontrado fue de 0.035454, estos resultados arrojan una desviación estándar de 0.005046 lo cual indica poca dispersión en los valores de las 30 ejecuciones con respecto al promedio de NRMS. Teniendo en cuenta el valor inicial y promedio de NRMS, GASA logra una mejora del 91.90% aproximadamente. Con respecto a los tiempos obtenidos, el promedio de tiempo para cumplir el criterio de parada del algoritmo fue de 11:48, el mejor tiempo registrado fue de 11:08 y el peor tiempo fue de 18:12.

El algoritmo SPSA obtuvo un promedio de NRMS de 0.082146, el mejor valor encontrado fue de 0.024845 y el peor valor encontrado fue de 0.216319, estos resultados arrojan una desviación estándar de 0.040124 lo cual indica una dispersión mayor con respecto a los algoritmos meta-heurísticos propuestos en los valores de las 30 ejecuciones con respecto al promedio de NRMS. Teniendo en cuenta el valor inicial y promedio de NRMS, SPSA logra una mejora del 71.79% aproximadamente. Con respecto a los tiempos obtenidos, el promedio de tiempo para cumplir el criterio de parada del algoritmo fue de 12:13, el mejor tiempo registrado fue de 11:07 y el peor tiempo fue de 14:08.

Para realizar una comparación justa, se tuvo en cuenta las siguientes condiciones: en primer lugar, se seleccionó la red McTrans con un conjunto de parámetros ya establecidos en anteriores investigaciones [12], permitiendo de esta forma que el valor inicial de NRMS para todos los algoritmos iniciara en el mismo valor (0.291182). Por otro lado, cada algoritmo ejecuta entre 5200 y 5400 evaluaciones de la función objetivo. Finalmente, se estableció la misma cantidad de hilos (15) que podría ejecutar cada algoritmo en paralelo. Los resultados promedio que se

registraron en las 30 ejecuciones para cada algoritmo en diferentes intervalos de tiempo se ilustran en la **Figura 35**.

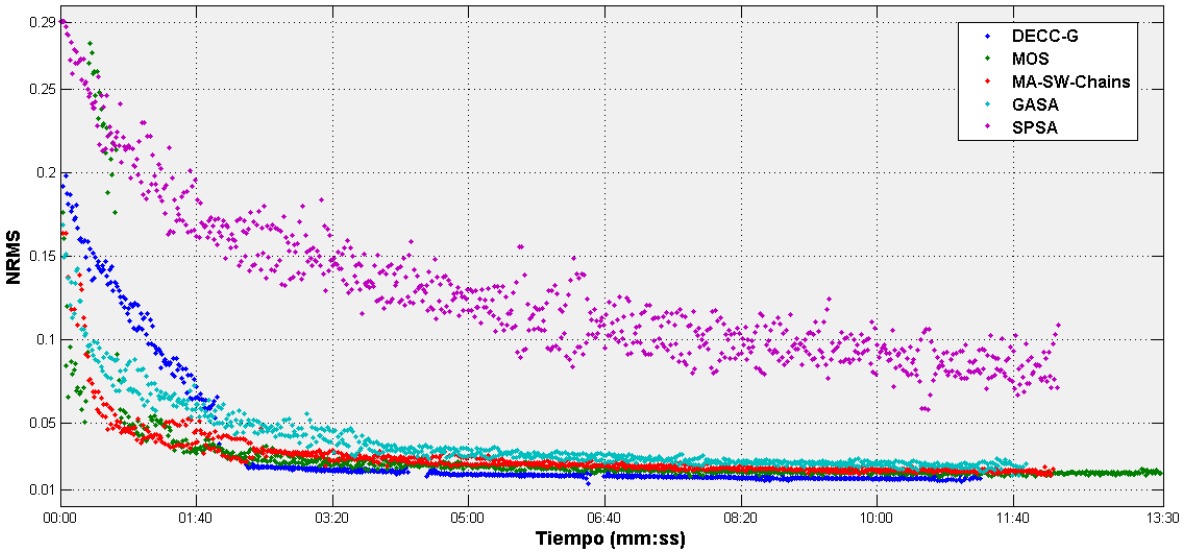



***Figura 35:*** *Gráfica NRMS vs Tiempo en el modelo McTrans de los algoritmos propuestos y los del estado del arte (volumen y velocidad)*

Como se puede apreciar, los algoritmos que convergen rápidamente y obtienen mejores resultados en el proceso inicial de calibración corresponden a MOS y MA-SW-Chains. Sin embargo, en poco tiempo (aproximadamente 2 minutos) son superados por DECC-G y éste logra alcanzar soluciones mucho más óptimas hasta el final del proceso de calibración en menor tiempo que el resto de algoritmos. Aunque MOS muestra buenos resultados en el inicio de la calibración, se observa que es el algoritmo más demorado para terminar el proceso de optimización. Con respecto a GASA, tiene un funcionamiento estable durante todo el proceso de calibración, pero no logra resultados tan óptimos como DECC-G, MOS y MA-SW-Chains, aunque se demora menos en su proceso que MOS y MA-SW-Chains. Con respecto al algoritmo SPSA se evidencia que tiene el comportamiento más disperso de los algoritmos y no alcanza buenos resultados de NRMS y tiene un tiempo de ejecución similar a MA-SW-Chains. Por lo anterior, se afirma que DECC-G es el mejor algoritmo para el modelo de complejidad baja mostrando un promedio NRMS menor, menos tiempo de ejecución y la menor desviación estándar.

Si bien el algoritmo MOS es el que más rápido converge a una solución prometedora al igual que MA-SW-Chains, en menos de dos minutos el algoritmo DECC-G los supera durante el tiempo restante del proceso de calibración. Sin embargo, los algoritmos meta-heurísticos propuestos cumplen satisfactoriamente con el criterio de calibración, en el cual el 100% de los enlaces logran tener un GEH menor a 5, partiendo que el valor inicial de GEH es menor a 5 para el 55% de los enlaces. La **Figura 36** muestra el GEH de las mejores ejecuciones de cada algoritmo meta-heurístico propuesto.

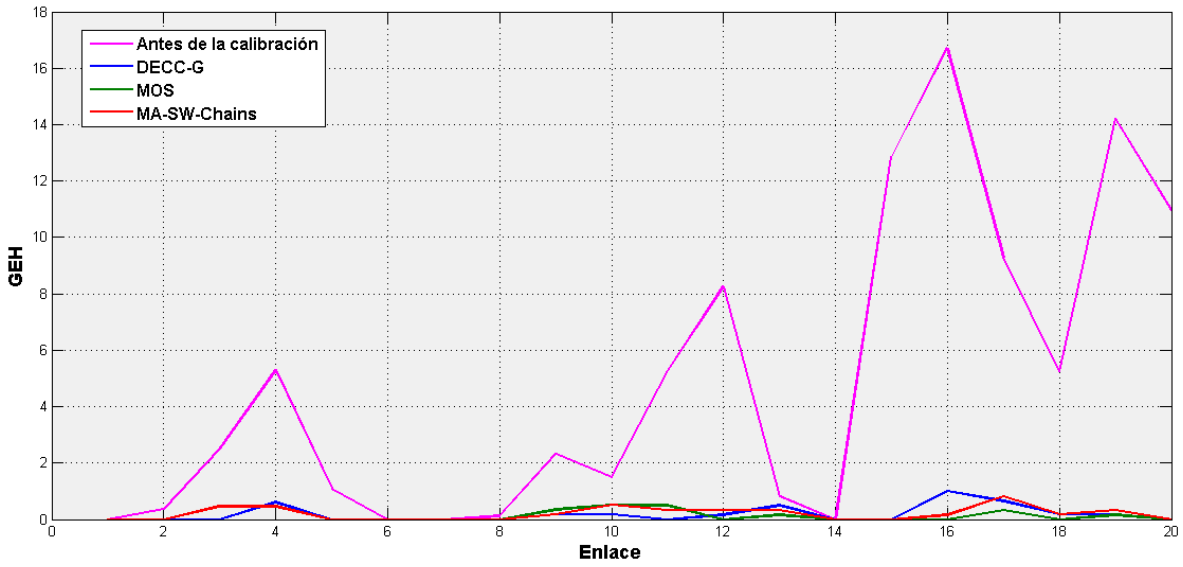



***Figura 36:*** *Gráfica GEH vs Enlace en el modelo McTrans de los algoritmos meta-heurísticos propuestos (volumen y velocidad)*

La **Figura 37** ilustra el GEH de las mejores ejecuciones del algoritmo meta-heurístico DECC-G y los del estado del arte, donde se ve claramente reflejado que los algoritmos obtienen un valor de GEH menor a 5 para el 100% de los enlaces del modelo McTrans, cumpliendo con el criterio de calibración de esta investigación.

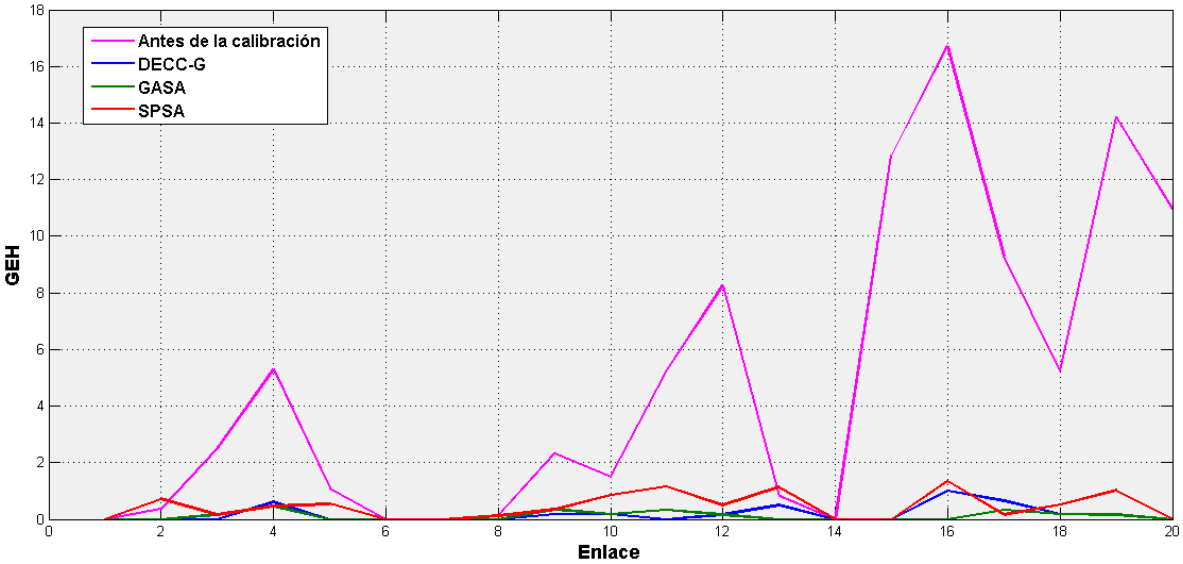



***Figura 37:*** *Gráfica GEH vs Enlace en el modelo McTrans de DECC-G y algoritmos del estado del arte (volumen y velocidad)*

La **Tabla 8** resume los resultados de los algoritmos propuestos y los del estado del arte para el modelo de baja complejidad McTrans.

| ALGORITMO | NRMS | | | | GEH DEL MEJOR RESULTADO | TIEMPO PROMEDIO (mm:ss) |
|---|---|---|---|---|---|---|
| | PROMEDIO | MEJOR RESULTADO | PEOR RESULTADO | DESVIACIÓN ESTÁNDAR | | |
| DECC-G | 0.016301 | 0.013634 | 0.019504 | 0.001393 | 100% | 11:16 |
| MOS | 0.019762 | 0.011759 | 0.027190 | 0.003994 | 100% | 13:28 |
| MA-SW-Chains | 0.021011 | 0.014450 | 0.037628 | 0.005168 | 100% | 12:09 |
| GASA | 0.023591 | 0.011926 | 0.035454 | 0.005046 | 100% | 11:48 |
| SPSA | 0.082146 | 0.024845 | 0.216319 | 0.040124 | 100% | 12:13 |

***Tabla 8:*** *Resultados en el modelo McTrans (volumen y velocidad)*

#### 4.3.1.5 Análisis estadístico

Al finalizar las 30 ejecuciones de cada algoritmo para la red de baja complejidad en la fase uno descrita anteriormente, se realizó un análisis estadístico con la prueba de Wilcoxon. El propósito de la prueba es determinar la dominancia de un algoritmo sobre los demás. Los resultados de la prueba se evidencian en la **Figura 38** en donde el punto negro indica que el algoritmo de la fila domina al algoritmo de la columna, el punto blanco indica que el algoritmo de la columna domina al de la fila y la casilla vacía indica que no es posible establecer una dominancia de un algoritmo sobre el otro. Los resultados por debajo de la diagonal tienen un nivel de significancia del 0.95 y los que están por encima de 0.90.

| | (1) | (2) | (3) | (4) | (5) |
|---|---|---|---|---|---|
| DECC-G (1) | - | ● | ● | ● | ● |
| MOS (2) | ○ | - | | ● | ● |
| MA-SW-Chains (3) | ○ | | - | ● | ● |
| GASA (4) | ○ | ○ | ○ | - | ● |
| SPSA (5) | ○ | ○ | ○ | ○ | - |

***Figura 38:*** *Resultados de la prueba de Wilcoxon para el modelo McTrans (volumen y velocidad)*

Como se puede apreciar, el algoritmo que claramente domina a todos los algoritmos es el DECC-G. Adicionalmente, la prueba revela que todos los algoritmos meta-heurísticos propuestos en la presente investigación dominan a los encontrados en la literatura (GASA y SPSA). Además, entre MOS y MA-SW-Chains no se puede establecer un dominio entre uno y otro.

Adicionalmente, se realizó la prueba de Friedman en la cual se busca establecer un ranking de los algoritmos asignando el menor valor al algoritmo que presenta mejores resultados. Los resultados obtenidos se muestran en la **Tabla 9**, en la cual se evidencia claramente que el mejor algoritmo entre los propuestos y los reportados en el estado del arte es el algoritmo meta-heurístico propuesto DECC-

G, ratificando así los resultados obtenidos por la anterior prueba y los resultados descritos anteriormente. Se puede notar nuevamente que todos los algoritmos propuestos en la presente investigación superan notablemente los algoritmos encontrados en el estado del arte. A diferencia de la anterior prueba, en esta se puede resaltar que el algoritmo meta-heurístico propuesto MOS supera por una pequeña diferencia al algoritmo meta-heurístico MA-SW-Chains. Lo anterior permite establecer junto con la prueba de Wilcoxon que de los algoritmos propuestos el mejor es el DECC-G seguido por el algoritmo MOS para resolver el problema de calibración de complejidad baja McTrans con la función objetivo contemplando los componentes de volumen y velocidad.

| ALGORITMO | RANKING |
|---|---|
| DECC-G | 1.3333 |
| MOS | 2.5333 |
| MA-SW-Chains | 2.7667 |
| GASA | 3.4333 |
| SPSA | 4.9333 |

***Tabla 9:*** *Resultados de la prueba de Friedman para el modelo McTrans (volumen y velocidad)*

### 4.3.2 Resultados y comparación para el modelo Reno con los componentes de volumen y velocidad

A continuación se presentan los resultados obtenidos por cada uno de los algoritmos en el modelo de complejidad media.

#### 4.3.2.1 Resultados obtenidos con DECC-G

Para la ejecución del algoritmo DECC-G con el modelo Reno y obtener los resultados finales, se realizó el mismo procedimiento descrito con en el modelo McTrans, es decir, la cantidad de experimentos y los parámetros del algoritmo no cambiaron para esta nueva red. Esto con el fin de poner a prueba la misma configuración del algoritmo, en una red más grande donde la cantidad de enlaces que se tiene en Reno es mayor que en McTrans.

La ejecución de las 30 pruebas realizadas iniciaron con un NRMS de 0.221791, en ellas se obtuvo un promedio de NRMS de 0.076795, el mejor valor encontrado fue de 0.072939 y el peor valor encontrado fue de 0.079446, estos resultados arrojan una desviación estándar de 0.001551, lo cual indica poca dispersión en los valores de las 30 ejecuciones con respecto al promedio de NRMS. Teniendo en cuenta el valor inicial y promedio de NRMS, DECC-G logra una mejora del 65.38% aproximadamente. Con respecto a los tiempos obtenidos, el promedio de tiempo para cumplir el criterio de parada del algoritmo fue de 57:58, el mejor tiempo registrado fue de 56:30 y el peor tiempo fue de 1:09:06, lo cual revela a nivel general

que el algoritmo cumple con el criterio de parada en un estimado de 57 minutos para la red de complejidad media.

#### 4.3.2.2 Resultados obtenidos con MOS

Para realizar las pruebas con el algoritmo meta-heurístico MOS en la red Reno se definieron los siguientes parámetros: número de evaluaciones de la función objetivo por paso = 480, número de pasos = 11, porcentaje de incremento de evaluaciones de la función objetivo = 25, porcentaje de individuos = 25 y rho = 5. Estos parámetros varían un poco con respecto a McTrans con el objetivo de obtener mejores resultados. En el arreglo de cobertura definido en la sección 4.2.2, los parámetros definidos para este modelo, también arrojan buenos resultados, además se identifica que rho debe tomar un valor pequeño a medida que va creciendo el modelo.

El promedio de NRMS para este algoritmo en las 30 pruebas fue de 0.085525, el mejor valor de NRMS encontrado fue de 0.076537 y el peor encontrado 0.097568, estos resultados arrojan una desviación estándar de 0.005557 lo cual indica poca dispersión en los valores de las 30 ejecuciones con respecto al promedio de NRMS. Teniendo en cuenta el valor inicial y promedio de NRMS, MOS logra una mejora del 61.44% aproximadamente. El tiempo promedio de ejecución del total de experimentos fue de 58:58, el mejor tiempo registrado fue de 58:04 y el peor tiempo fue de 59:57.

#### 4.3.2.3 Resultados obtenidos con MA-SW-Chains

El conjunto de parámetros para ejecutar el algoritmo MA-SW-Chains fueron: tamaño de la población = 15, número de generaciones = 16, número de padres = 15, porcentaje de selección = 60, porcentaje de cruce = 85, porcentaje de mutación = 10, paso de mutación = 2, tramo de intensidad = 300 y generaciones en un óptimo local = 1. Bajo esta configuración de parámetros el comportamiento del algoritmo meta-heurístico, se realizaron las 30 ejecuciones. El único cambió de parámetro con respecto a McTrans fue el tamaño de la población, ya que se identificó que la población debe ser menor a medida que va creciendo la red para alcanzar mejores resultados.

Con las diferentes ejecuciones se obtuvo un promedio de NRMS de 0.095523, el mejor valor de NRMS registrado fue de 0.078747 y el peor de 0.135866, estos resultados arrojan una desviación estándar de 0.012504 lo cual indica mayor dispersión en los valores de las 30 ejecuciones con respecto al promedio de NRMS. Teniendo en cuenta el valor inicial y promedio de NRMS, MA-SW-Chains logra una mejora del 56.93% aproximadamente. Por otro lado, los registros referentes al tiempo de ejecución del algoritmo determinaron un promedio de 01:06:18, el mejor y peor tiempo registrado fueron 01:01:58 y 01:10:39 respectivamente.

#### 4.3.2.4 Comparación de los algoritmos propuestos con GASA y SPSA

Con respecto a los algoritmos del estado del arte GASA y SPSA, para cada uno de ellos, también se realizaron 30 ejecuciones para este modelo de complejidad media,

donde GASA obtuvo un promedio de NRMS de 0.086270, el mejor valor encontrado fue de 0.079550 y el peor valor encontrado fue de 0.093249, estos resultados arrojan una desviación estándar de 0.003521 lo cual indica poca dispersión en los valores de las 30 ejecuciones con respecto al promedio de NRMS. Teniendo en cuenta el valor inicial y promedio de NRMS, GASA logra una mejora del 61.10% aproximadamente. Con respecto a los tiempos obtenidos, el promedio de tiempo para cumplir el criterio de parada del algoritmo fue de 01:00:56, el mejor tiempo registrado fue de 59:44 y el peor tiempo fue de 01:02:07.

El algoritmo SPSA obtuvo un promedio de NRMS de 0.152618, el mejor valor encontrado fue de 0.105134 y el peor valor encontrado fue de 0.219641, estos resultados arrojan una desviación estándar de 0.025917 lo cual indica una dispersión mayor con respecto a los algoritmos meta-heurísticos propuestos en los valores de las 30 ejecuciones con respecto al promedio de NRMS. Teniendo en cuenta el valor inicial y promedio de NRMS, SPSA logra una mejora del 31.19% aproximadamente. Con respecto a los tiempos obtenidos, el promedio de tiempo para cumplir el criterio de parada del algoritmo fue de 40:47, el mejor tiempo registrado fue de 39:06 y el peor tiempo fue de 44:49.

Las condiciones que se tuvieron en cuenta para la comparación justa corresponden a las mismas que las descritas anteriormente en el modelo McTrans. El valor inicial de NRMS para esta red fue de 0.221791 al iniciar el proceso de calibración, para cada uno de los algoritmos. La cantidad de evaluaciones de la función objetivo al igual que en la red McTrans se estableció entre 5200 y 5400. El comportamiento de cada algoritmo se evidencia en la **Figura 39**.

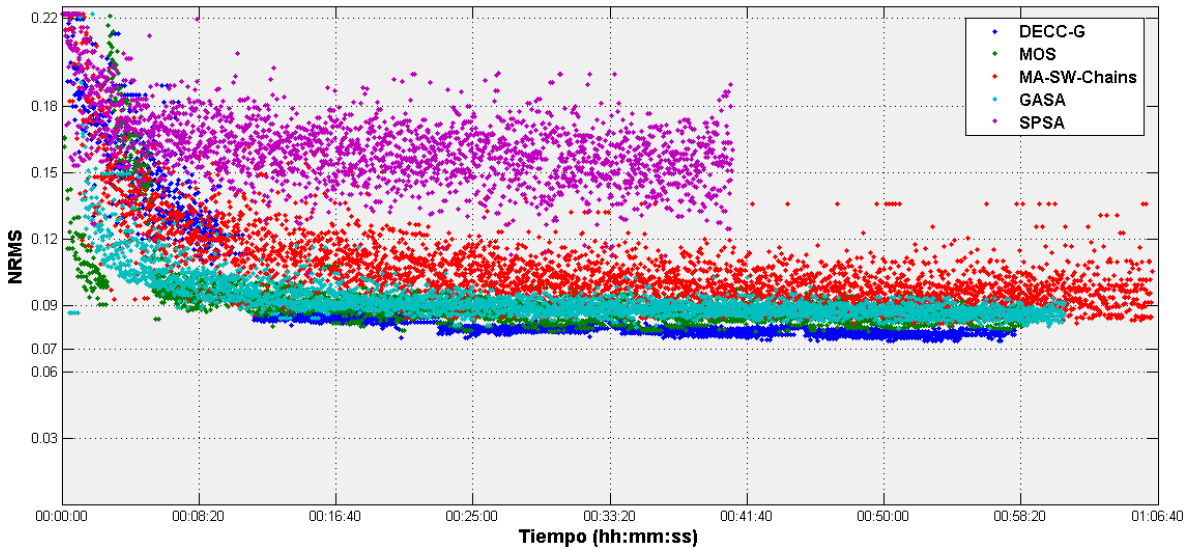



***Figura 39:*** *Gráfica NRMS vs Tiempo en el modelo Reno de los algoritmos propuestos y los del estado del arte (volumen y velocidad)*

Los algoritmos que convergen rápidamente y obtienen mejores resultados al inicio del proceso de calibración son MOS y GASA, aunque MOS alcanza mejores

resultados de NRMS en menos tiempo a lo largo del proceso de calibración que GASA. Sin embargo, DECC-G luego de 12 minutos (menos del 20% del tiempo de calibración) alcanza mejores soluciones que MOS y GASA en menor tiempo que el resto de algoritmos. Los algoritmos MA-SW-Chains y SPSA presentan mayor dispersión en sus resultados. MA-SW-Chains es el algoritmo más demorado en este modelo, aunque alcanza mejores resultados de NRMS que SPSA. El algoritmo SPSA registra menos tiempo de ejecución, pero es el algoritmo que presenta mayor dispersión en sus datos sin encontrar buenos resultados de NRMS. Se observa claramente que el algoritmo que mejor encuentra soluciones prometedoras durante un alto porcentaje del proceso de calibración es DECC-G, considerando además que obtiene soluciones mucho más homogéneas que los otros algoritmos. A pesar de que GASA es el algoritmo que más rápido converge, para un proceso de calibración de aproximadamente una hora dada las características del modelo y la dimensionalidad del problema, el hecho de que DECC-G lo supere en menos de 12 minutos se considera realmente rápido el tiempo en el cual DECC-G logra alcanzar mejores soluciones que los algoritmos reportados en la literatura. Por lo anterior, se afirma que DECC-G es el mejor algoritmo para el modelo de complejidad media Reno mostrando un promedio NRMS menor, menos tiempo de ejecución y la menor desviación estándar.

Los algoritmos meta-heurísticos propuestos cumplen satisfactoriamente con el criterio de calibración, en el cual el 100% de los enlaces logra tener un GEH menor a 5, partiendo que el valor inicial de GEH es menor a 5 para el 47% aproximadamente de los enlaces. La **Figura 40** muestra el GEH de las mejores ejecuciones de cada algoritmo meta-heurístico propuesto.

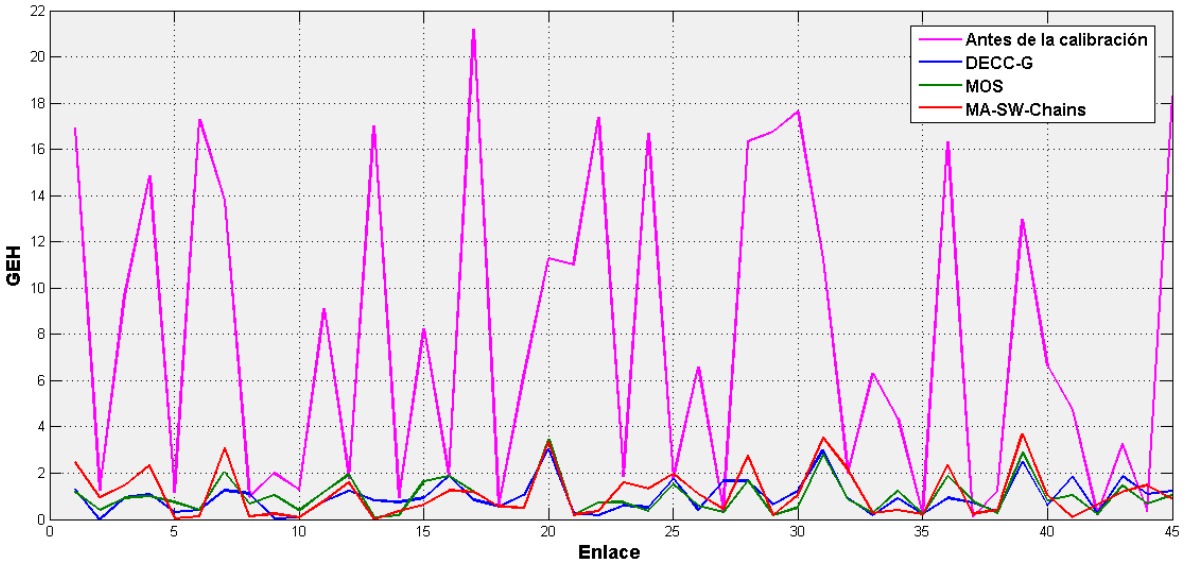



***Figura 40:*** *Gráfica GEH vs Enlace en el modelo Reno de los algoritmos meta-heurísticos propuestos (volumen y velocidad)*

La **Figura 41** ilustra el GEH de las mejores ejecuciones del algoritmo meta-heurístico DECC-G y los encontrados en el estado del arte, donde se ve que DECC-G y GASA obtienen un valor de GEH menor a 5 para el 100% de los enlaces del modelo Reno, cumpliendo con el criterio de calibración, mientras que SPSA obtiene un valor de GEH menor a 5 para el 91% aproximadamente el cual no cumple el criterio de calibración definido en esta investigación.

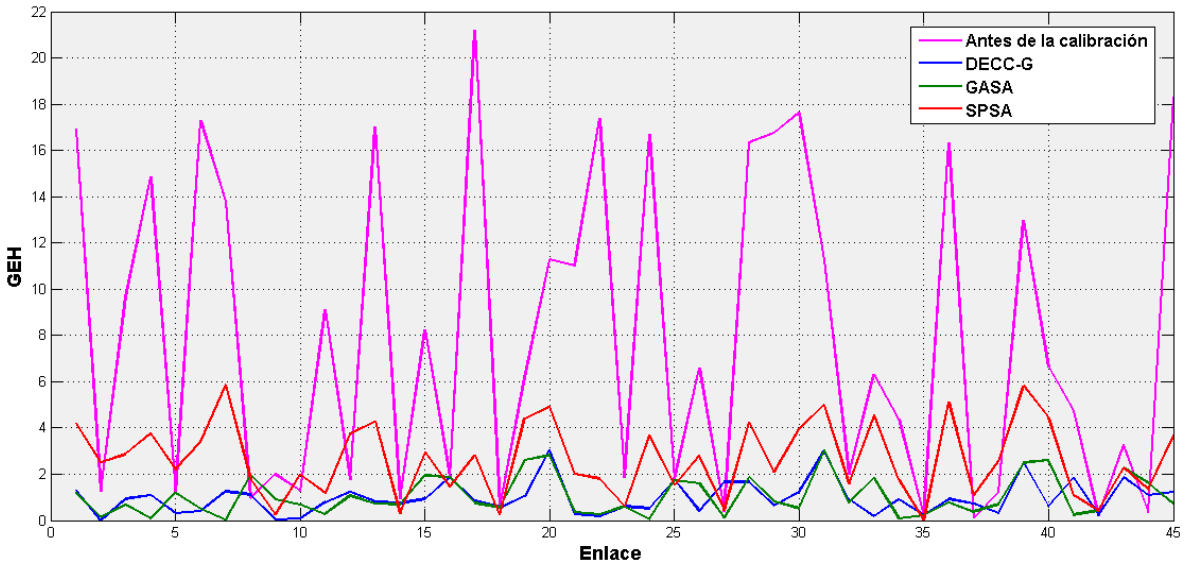



***Figura 41:*** *Gráfica GEH vs Enlace en el modelo Reno de DECC-G y de los algoritmos del estado del arte (volumen y velocidad)*

La **Tabla 10** resume los resultados de los algoritmos propuestos y los del estado del arte para el modelo de complejidad media, Reno.

| ALGORITMO | NRMS | | | | GEH DEL MEJOR RESULTADO | TIEMPO PROMEDIO (hh:mm:ss) |
|---|---|---|---|---|---|---|
| | PROMEDIO | MEJOR RESULTADO | PEOR RESULTADO | DESVIACIÓN ESTÁNDAR | | |
| DECC-G | 0.076795 | 0.072939 | 0.079446 | 0.001551 | 100% | 00:57:58 |
| MOS | 0.085525 | 0.076537 | 0.097568 | 0.005557 | 100% | 00:58:58 |
| MA-SW-Chains | 0.095523 | 0.078747 | 0.135866 | 0.012504 | 100% | 01:06:18 |
| GASA | 0.086270 | 0.079550 | 0.093249 | 0.003521 | 100% | 01:00:56 |
| SPSA | 0.152618 | 0.105134 | 0.219641 | 0.025917 | 91% | 00:40:47 |

***Tabla 10:*** *Resultados en el modelo Reno (volumen y velocidad)*

### 4.3.2.5 Análisis estadístico

Similar a lo realizado con la red de baja complejidad McTrans, se realizó un análisis estadístico con la prueba de Wilcoxon y de Friedman para determinar cuál de los algoritmos meta-heurísticos propuestos junto con los encontrados en el estado del arte presentan mejores resultados y si las diferencias en los resultados son estadísticamente significativas. La primera prueba se realizó con Wilcoxon, en la

cual se establece una dominancia de un algoritmo sobre otro. Importante recordar que el punto negro indica que el algoritmo de la fila domina al algoritmo de la columna, el punto blanco indica que el algoritmo de la columna domina al de la fila y la casilla vacía indica que no es posible establecer una dominancia de un algoritmo sobre el otro. Los resultados por debajo y encima de la diagonal tienen un nivel de significancia del 0.95 y 0.90 respectivamente. El resultado de la prueba se muestra en la **Figura 42**.

| | (1) | (2) | (3) | (4) | (5) |
|---|---|---|---|---|---|
| DECC-G (1) | - | ● | ● | ● | ● |
| MOS (2) | ○ | - | ● | | ● |
| MA-SW-Chains (3) | ○ | ○ | - | ○ | ● |
| GASA (4) | ○ | | ● | - | ● |
| SPSA (5) | ○ | ○ | ○ | ○ | - |

***Figura 42:*** *Resultados de la prueba de Wilcoxon para el modelo Reno (volumen y velocidad)*

La **Figura 42** registra nuevamente que el algoritmo que domina a todos los demás es el DECC-G. Sin embargo, a diferencia del anterior análisis estadístico hecho en el modelo McTrans se logra evidenciar que no todos los algoritmos meta-heurísticos propuestos superan a los algoritmos reportados en la literatura. Es el caso del MA-SW-Chains el cual es dominado por GASA más no por SPSA. Entre el algoritmo propuesto MOS y GASA no es posible determinar una dominancia. Adicionalmente, se muestra como el algoritmo SPSA es dominado por todos los demás algoritmos.

Por otro lado, los resultados obtenidos en la prueba de Friedman la cual establece un ranking de los algoritmos (**Tabla 11**), revela de nuevo y ratifica que el algoritmo claramente ganador para el proceso de calibración en redes de complejidad media, es el DECC-G, el cual supera notablemente al resto de algoritmos meta-heurísticos propuestos y los presentados en el estado del arte. Al igual que la prueba de Wilcoxon y a diferencia del análisis hecho en McTrans, el algoritmo propuesto MA-SW-Chains no logra superar a GASA pero si a SPSA. A diferencia de la anterior prueba, se observa que MOS logra superar a GASA, conservando la segunda posición que obtuvo en el análisis con la red McTrans, lo cual permite afirmar que DECC-G y MOS obtienen los mejores resultados en redes de complejidad baja y media contemplando los componentes de volumen y velocidad en la función objetivo.

| ALGORITMO | RANKING |
|---|---|
| DECC-G | 1 |
| MOS | 2.6667 |
| GASA | 2.9 |
| MA-SW-Chains | 3.5 |
| SPSA | 4.9333 |

***Tabla 11:*** *Resultados de la prueba de Friedman para el modelo Reno (volumen y velocidad)*

### 4.3.3 Resultados y comparación para el modelo I-75 con los componentes de volumen y velocidad

A continuación se presentan los resultados que se obtuvieron para la red de complejidad alta I-75 con cada uno de los algoritmos meta-heurísticos propuestos y los presentados en la literatura.

#### 4.3.3.1 Resultados obtenidos con DECC-G

Para la ejecución del algoritmo DECC-G con el modelo I-75 y obtener los resultados finales, se realizó el mismo procedimiento descrito con los modelos McTrans y Reno, es decir, la cantidad de experimentos y los parámetros del algoritmo no cambiaron para esta nueva red. Esto con el fin de poner a prueba la misma configuración del algoritmo en una red más grande donde la cantidad de enlaces que se tiene en I-75 es mayor que en Reno y McTrans.

Las ejecuciones de las 30 pruebas realizadas iniciaron con un NRMS de 0.331824, en ellas se obtuvo un promedio de NRMS de 0.083380, el mejor valor encontrado fue de 0.071467 y el peor valor encontrado fue de 0.091165, estos resultados arrojan una desviación estándar de 0.004144 lo cual indica poca dispersión en los valores de las 30 ejecuciones con respecto al promedio de NRMS. Teniendo en cuenta el valor inicial y promedio de NRMS, DECC-G logra una mejora del 74.87% aproximadamente. Con respecto a los tiempos obtenidos, el promedio de tiempo para cumplir el criterio de parada del algoritmo fue de 04:10:42, el mejor tiempo registrado fue de 04:07:11 y el peor tiempo fue de 04:19:19.

#### 4.3.3.2 Resultados obtenidos con MOS

Los parámetros para ejecutar el algoritmo MOS en este modelo de complejidad alta, fueron los mismos que en el modelo Reno. Los resultados que presenta MOS para el modelo de complejidad alta I-75 son los siguientes: el promedio de NRMS de ejecutar el algoritmo en 30 pruebas fue de 0.218630, el mejor valor de NRMS encontrado fue de 0.078374 y el peor encontrado 0.285472, estos resultados arrojan una desviación estándar de 0.067822 lo cual indica mayor dispersión en los valores de las 30 ejecuciones con respecto al promedio de NRMS. Teniendo en cuenta el valor inicial y promedio de NRMS, MOS logra una mejora del 34.11% aproximadamente. El tiempo promedio de ejecución del total de experimentos fue

de 04:38:29, el mejor tiempo registrado fue de 04:27:26 y el peor tiempo fue de 04:49:23.

#### 4.3.3.3 Resultados obtenidos con MA-SW-Chains

Con respecto al algoritmo MA-SW-Chains, los parámetros a utilizar son los mismo que en el modelo Reno. Al terminar las 30 ejecuciones se obtuvo un promedio de NRMS de 0.255762, el mejor valor que se encontró fue de 0.106086 y el peor encontrado 0.298015, estos resultados arrojan una desviación estándar de 0.047830 lo cual indica mayor dispersión en los valores de las 30 ejecuciones con respecto al promedio de NRMS. Teniendo en cuenta el valor inicial y promedio de NRMS, MA-SW-Chains logra una mejora del 22.92% aproximadamente. El tiempo promedio de ejecución del total de experimentos fue de 04:34:11, el mejor tiempo registrado fue de 04:21:33 y el peor tiempo fue de 04:59:18.

#### 4.3.3.4 Comparación de los algoritmos propuestos con GASA Y SPSA

Con respecto a los algoritmos del estado del arte GASA y SPSA, con cada uno de ellos se realizó nuevamente 30 ejecuciones para este modelo de complejidad alta, donde GASA obtuvo un promedio de NRMS de 0.098343, el mejor valor encontrado fue de 0.082160 y el peor valor encontrado fue de 0.148625, estos resultados arrojan una desviación estándar de 0.011792 lo cual indica poca dispersión en los valores de las 30 ejecuciones con respecto al promedio de NRMS. Teniendo en cuenta el valor inicial y promedio de NRMS, GASA logra una mejora del 70.36% aproximadamente. Con respecto a los tiempos obtenidos, el promedio de tiempo para cumplir el criterio de parada del algoritmo fue de 04:34:36, el mejor tiempo registrado fue de 04:25:33 y el peor tiempo fue de 04:39:43.

El algoritmo SPSA obtuvo un promedio de NRMS de 0.328595, el mejor valor encontrado fue de 0.311928 y el peor valor encontrado fue de 0.331824, estos resultados arrojan una desviación estándar de 0.006821 lo cual indica una dispersión mayor con respecto a los algoritmos meta-heurísticos propuestos en los valores registrados en las 30 ejecuciones con respecto al promedio de NRMS. Teniendo en cuenta el valor inicial y promedio de NRMS, SPSA logra una mejora del 0.97% aproximadamente. Con respecto a los tiempos obtenidos, el promedio de tiempo para cumplir el criterio de parada del algoritmo fue de 02:58:40, el mejor tiempo registrado fue de 02:32:44 y el peor tiempo fue de 03:26:42.

Las condiciones tenidas en cuenta para una comparación justa son las mismas descritas para los modelos de McTrans y Reno. El valor inicial de NRMS para esta red fue de 0.331824 al iniciar el proceso de calibración para cada uno de los algoritmos. La cantidad de evaluaciones de la función objetivo al igual que en la red McTrans y Reno se estableció entre 5200 y 5400. El comportamiento de cada algoritmo se evidencia en la **Figura 43**.

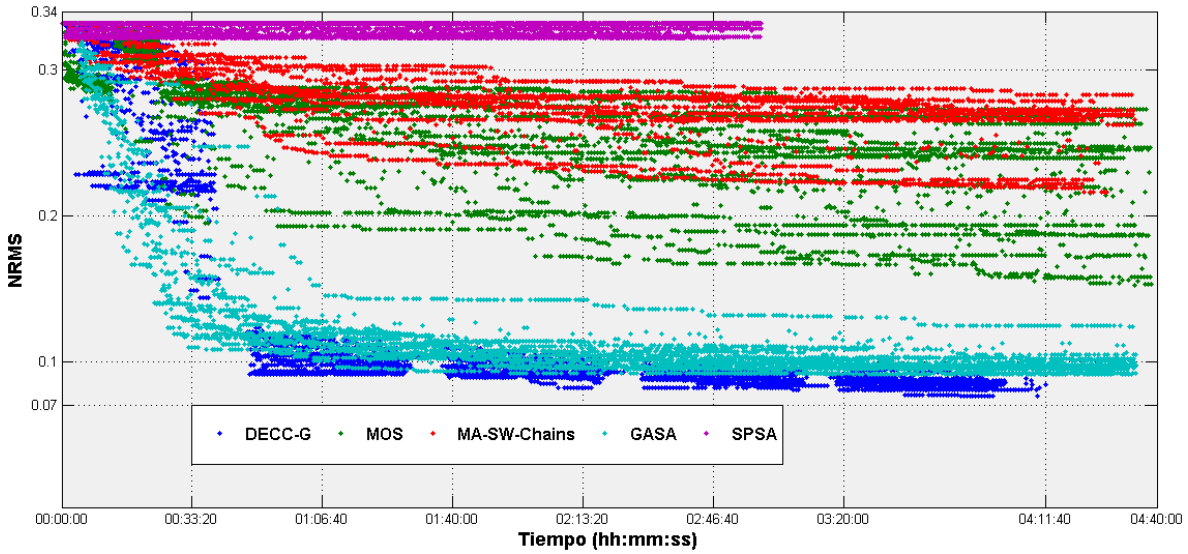



***Figura 43:*** *Gráfica NRMS vs Tiempo en el modelo I-75 de los algoritmos propuestos y los algoritmos encontrados en el estado del arte*

Los algoritmos que convergen rápidamente y obtienen mejores resultados en el proceso inicial de calibración son DECC-G y GASA, pero aproximadamente entre 40 y 50 minutos, DECC-G empieza a encontrar mejores resultados que GASA y termina más rápido el proceso de calibración. Con respecto a las meta-heurísticas MOS y MA-SW-Chains, son los algoritmos que presentan resultados más dispersos y los valores NRMS no logran ser tan óptimos como los de GASA, pero si superan los resultados de SPSA. MOS encuentra mejores resultados que MA-SW-Chains, aunque se demore un poco más en su proceso, además que MA-SW-Chains tiene un tiempo de ejecución similar a GASA. Con relación a SPSA, sus resultados muestran poca dispersión, pero es el algoritmo con los resultados menos favorables de NRMS, aunque es el algoritmo más rápido para terminar el proceso de calibración. Se puede observar cómo el algoritmo DECC-G a diferencia de los análisis realizados en los otros modelos, logra converger rápidamente a soluciones prometedoras y correlacionadas entre sí. Por lo anterior, se afirma que DECC-G es el mejor algoritmo para el modelo de complejidad alto mostrando un promedio NRMS menor, menos tiempo de ejecución y la menor desviación estándar.

Partiendo que el valor inicial de GEH es menor a 5 para el 40% aproximadamente de los enlaces, la **Figura 44** muestra el GEH de las mejores ejecuciones de cada algoritmo meta-heurístico propuesto, donde la meta-heurística DECC-G obtiene un valor de GEH menor a 5 para el 97% de los enlaces, cumpliendo con el criterio de calibración, MOS obtiene un valor de GEH menor a 5 para el 96% de los enlaces, cumpliendo con el criterio de calibración y MA-SW-Chains obtiene un valor de GEH menor a 5 para el 90% de los enlaces, de esta manera cumple con el criterio de calibración.

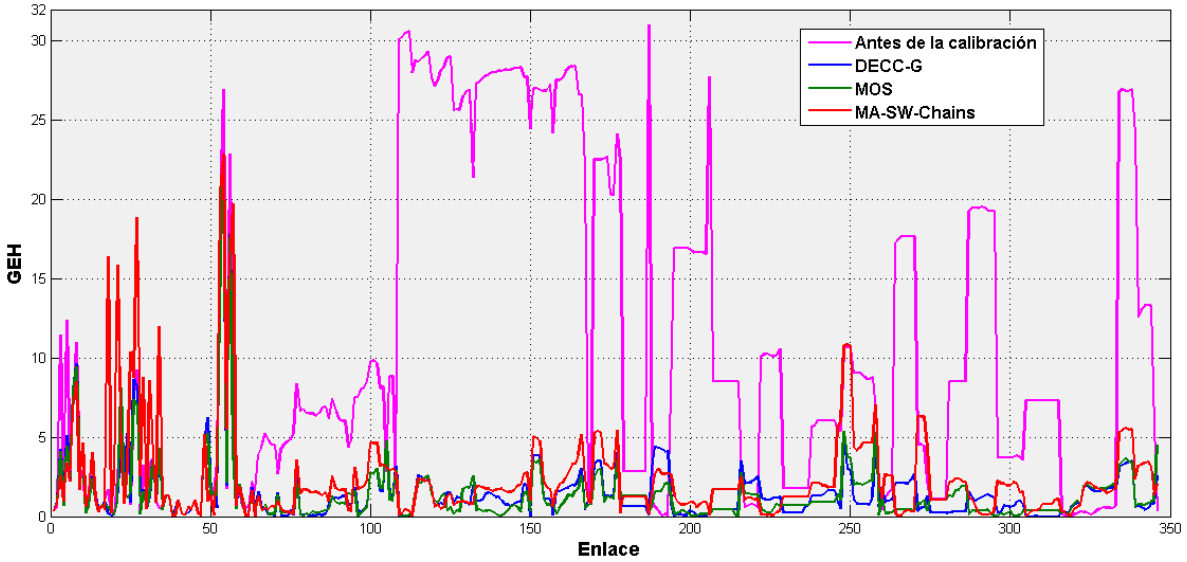



***Figura 44:*** *Gráfica GEH vs Enlace en el modelo I-75 de los algoritmos meta-heurísticos propuestos*

La **Figura 45** ilustra el GEH de las mejores ejecuciones del algoritmo meta-heurístico DECC-G y de los algoritmos encontrados en estado del arte, donde se ve que DECC-G obtiene un valor de GEH menor a 5 para el 97% de los enlaces, cumpliendo con el criterio de calibración, GASA obtiene un valor de GEH menor a 5 para el 93% de los enlaces, sin cumplir con el criterio de calibración y SPSA obtiene un valor de GEH menor a 5 para el 42% de los enlaces, sin cumplir con el criterio de calibración.

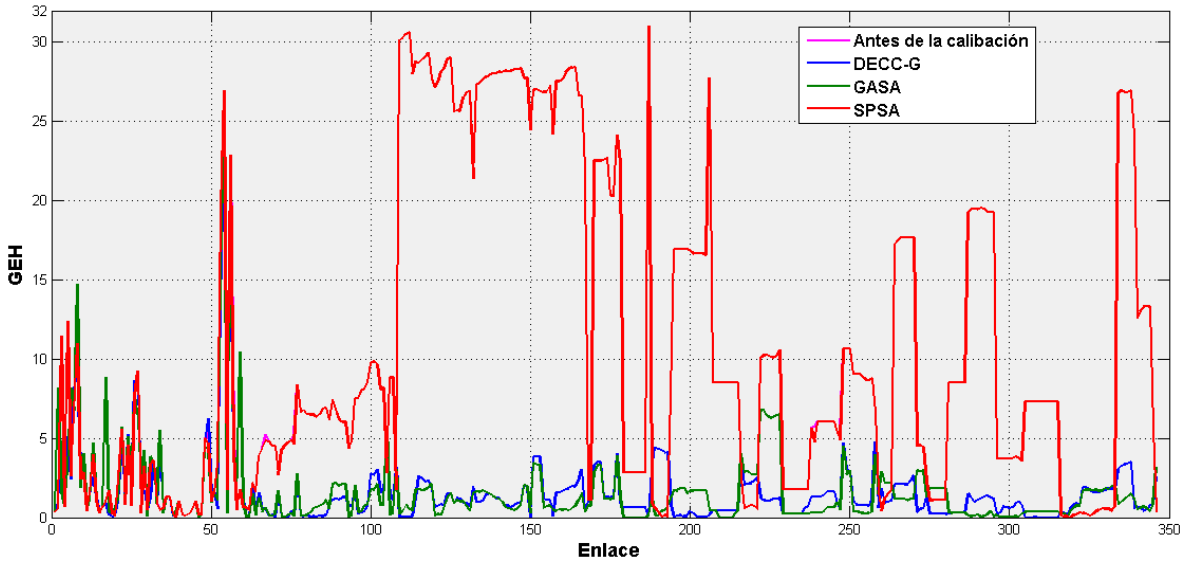



***Figura 45:*** *Gráfica GEH vs Enlace en el modelo I-75 de DECC-G y los algoritmos encontrados en el estado del arte*

La **Tabla 12** resume los resultados de los algoritmos propuestos y los algoritmos encontrados en el estado del arte para el modelo de complejidad alta I-75.

| ALGORITMO | NRMS | | | | GEH DEL MEJOR RESULTADO | TIEMPO PROMEDIO (hh:mm:ss) |
|---|---|---|---|---|---|---|
| | PROMEDIO | MEJOR RESULTADO | PEOR RESULTADO | DESVIACIÓN ESTÁNDAR | | |
| DECC-G | 0.083380 | 0.071467 | 0.091165 | 0.004144 | 97% | 04:10:42 |
| MOS | 0.218630 | 0.078374 | 0.285472 | 0.067822 | 96% | 04:38:29 |
| MA-SW-Chains | 0.255762 | 0.106086 | 0.298015 | 0.047830 | 90% | 04:34:11 |
| GASA | 0.098343 | 0.082160 | 0.148625 | 0.011792 | 93% | 04:34:36 |
| SPSA | 0.328595 | 0.311928 | 0.331824 | 0.006821 | 42% | 02:58:40 |

***Tabla 12:*** *Resultados en el modelo I-75*

#### 4.3.3.5 Análisis estadístico

Al igual que en los dos modelos anteriores, para la red I-75 se realizó un análisis estadístico con la prueba de Wilcoxon y de Friedman para determinar cuál de los algoritmos meta-heurísticos propuestos junto con los presentados en el estado del arte presenta mejores resultados. El resultado de la prueba de Wilcoxon se muestra en la **Figura 46**.

| | (1) | (2) | (3) | (4) | (5) |
|---|---|---|---|---|---|
| DECC-G (1) | - | ● | ● | ● | ● |
| MOS (2) | ○ | - | ● | ○ | ● |
| MA-SW-Chains (3) | ○ | ○ | - | ○ | ● |
| GASA (4) | ○ | ● | ● | - | ● |
| SPSA (5) | ○ | ○ | ○ | ○ | - |

***Figura 46:*** *Resultados de la prueba de Wilcoxon para el modelo I-75*

Los resultados de la prueba reflejan que el algoritmo que tiene dominancia sobre el resto de algoritmos es el DECC-G, al igual que el análisis realizado para la red de baja complejidad McTrans y la de complejidad media Reno. Estos resultados permiten afirmar que el algoritmo dominante para resolver el problema de calibración sin importar la complejidad de la red es la propuesta ganadora DECC-G. Sin embargo, como ocurrió en el anterior análisis hecho para el modelo Reno, GASA es superior al algoritmo propuesto MA-SW-Chains y en esta ocasión también supera al algoritmo MOS. Nuevamente el algoritmo SPSA es dominado por todos los demás algoritmos. Lo cual, permite afirmar que sin importar el tamaño de la red o modelo CORSIM, el desempeño del algoritmo estocástico (SPSA) en el proceso de calibración es deficiente.

En cuanto al ranking que establece la prueba de Friedman para el modelo I-75, se puede evidenciar en la **Tabla 13** que el algoritmo claramente ganador para el

proceso de calibración en redes de complejidad alta como I-75 es el DECC-G, el cual supera notablemente al resto de algoritmos meta-heurísticos propuestos y los presentados en el estado del arte. Al igual que la prueba de Wilcoxon, los algoritmos propuestos MA-SW-Chains y MOS no logran superar a GASA pero si a SPSA.

| ALGORITMO | RANKING |
|---|---|
| DECC-G | 1.0667 |
| GASA | 2.1 |
| MOS | 3.2 |
| MA-SW-Chains | 3.6333 |
| SPSA | 5 |

***Tabla 13:*** *Resultados de la prueba de Friedman para el modelo I-75*

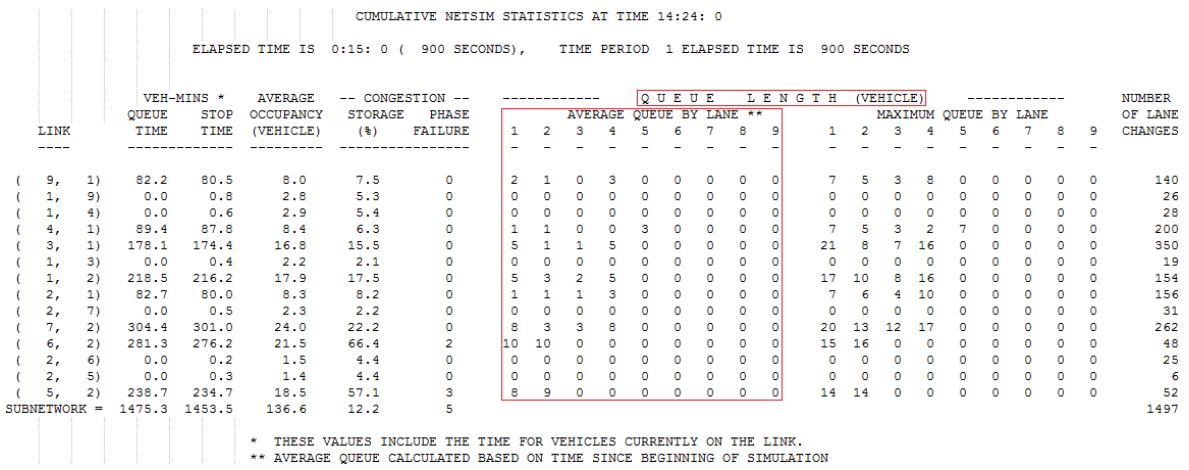
Los resultados hasta aquí presentados comprenden la primera fase de la experimentación en la cual se evalúa cada uno de los algoritmos en cada una de las redes de diferentes complejidades para la función objetivo en la cual sólo se tiene en cuenta volumen y velocidad, obteniendo claramente un algoritmo meta-heurístico ganador (DECC-G) en tiempo y resultados obtenidos.

## 4.4 DATOS DE CAMPO PARA LONGITUD DE COLAS

El Departamento de Transportes de Nevada no tiene datos reales para la variable longitud de colas en ninguno de los anteriores modelos de prueba, por esta razón es necesario generar estos datos a partir de las simulaciones con los datos de volumen y velocidad.

Los datos de campo para longitud de colas fueron generados para los modelos McTrans Network y Reno Network de la siguiente manera: se selecciona el archivo de salida con extensión *.out*, el cual se obtiene de la mejor calibración realizada por los tres algoritmos meta-heurísticos propuestos en los dos modelos antes mencionados. Para el caso del modelo McTrans, la mejor calibración la realizó el algoritmo MOS con un NRMS de 0.011759 y para el modelo Reno, la mejor calibración la realizó el algoritmo meta-heurístico DECC-G con un NRMS de 0.072939. Estas dos calibraciones tienen los datos más similares a la realidad para el volumen, velocidad y longitud de colas.

Se toma el archivo de salida de las anteriores simulaciones y se ubica la parte de “QUEUE LENGTH (VEHICLE)”, como se muestra en la **Figura 47**.



CUMULATIVE NETSIM STATISTICS AT TIME 14:24: 0

ELAPSED TIME IS 0:15: 0 ( 900 SECONDS), TIME PERIOD 1 ELAPSED TIME IS 900 SECONDS

| LINK | VEH-MINS * QUEUE TIME | VEH-MINS * STOP TIME | AVERAGE OCCUPANCY (VEHICLE) | -- CONGESTION -- STORAGE (%) | -- CONGESTION -- PHASE FAILURE | QUEUE LENGTH (VEHICLE) AVERAGE QUEUE BY LANE ** 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | MAXIMUM QUEUE BY LANE 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | NUMBER OF LANE CHANGES |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ( 9, 1) | 82.2 | 80.5 | 8.0 | 7.5 | 0 | 2 | 1 | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 7 | 5 | 3 | 8 | 0 | 0 | 0 | 0 | 0 | 140 |
| ( 1, 9) | 0.0 | 0.8 | 2.8 | 5.3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 26 |
| ( 1, 4) | 0.0 | 0.6 | 2.9 | 5.4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 28 |
| ( 4, 1) | 89.4 | 87.8 | 8.4 | 6.3 | 0 | 1 | 1 | 0 | 0 | 3 | 0 | 0 | 0 | 0 | 7 | 5 | 3 | 2 | 7 | 0 | 0 | 0 | 0 | 200 |
| ( 3, 1) | 178.1 | 174.4 | 16.8 | 15.5 | 0 | 5 | 1 | 1 | 5 | 0 | 0 | 0 | 0 | 0 | 21 | 8 | 7 | 16 | 0 | 0 | 0 | 0 | 0 | 350 |
| ( 1, 3) | 0.0 | 0.4 | 2.2 | 2.1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 19 |
| ( 1, 2) | 218.5 | 216.2 | 17.9 | 17.5 | 0 | 5 | 3 | 2 | 5 | 0 | 0 | 0 | 0 | 0 | 17 | 10 | 8 | 16 | 0 | 0 | 0 | 0 | 0 | 154 |
| ( 2, 1) | 82.7 | 80.0 | 8.3 | 8.2 | 0 | 1 | 1 | 1 | 3 | 0 | 0 | 0 | 0 | 0 | 7 | 6 | 4 | 10 | 0 | 0 | 0 | 0 | 0 | 156 |
| ( 2, 7) | 0.0 | 0.5 | 2.3 | 2.2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 31 |
| ( 7, 2) | 304.4 | 301.0 | 24.0 | 22.2 | 0 | 8 | 3 | 3 | 8 | 0 | 0 | 0 | 0 | 0 | 20 | 13 | 12 | 17 | 0 | 0 | 0 | 0 | 0 | 262 |
| ( 6, 2) | 281.3 | 276.2 | 21.5 | 66.4 | 2 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 15 | 16 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 48 |
| ( 2, 6) | 0.0 | 0.2 | 1.5 | 4.4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 25 |
| ( 2, 5) | 0.0 | 0.3 | 1.4 | 4.4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 6 |
| ( 5, 2) | 238.7 | 234.7 | 18.5 | 57.1 | 3 | 8 | 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 14 | 14 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 52 |
| SUBNETWORK = | 1475.3 | 1453.5 | 136.6 | 12.2 | 5 | | | | | | | | | | | | | | | | | | | 1497 |

* THESE VALUES INCLUDE THE TIME FOR VEHICLES CURRENTLY ON THE LINK.
** AVERAGE QUEUE CALCULATED BASED ON TIME SINCE BEGINNING OF SIMULATION

***Figura 47:*** *Longitud de colas y promedio de colas por carril en un archivo de salida CORSIM*

Como los datos de campo de volumen y velocidad son valores promedios que se detectaron con sensores, entonces para la variable longitud de colas es necesario tener valores promedios, para esto se ubica el bloque que dice “AVERAGE QUEUE BY LANE”. Este bloque indica el número de vehículos promedio que forman la longitud de colas en cada uno de los carriles. Es de aclarar que cada enlace (*link*) cuenta con un número diferente de carriles, para conocer con exactitud este número de carriles, se observa hasta cual carril existen datos de colas, por ejemplo para el enlace (9, 1) existen datos de colas hasta el carril 4, es decir que este enlace tiene 4 carriles, para el enlace (1, 9) su valor de longitud de colas es 0 al igual que para el enlace (1, 4), para el enlace (4, 1) existe datos de colas hasta el carril 5, es decir que este enlace tiene 5 carriles y así sucesivamente con cada uno de los enlaces.

Teniendo en cuenta el promedio de vehículos que forman la longitud de cola en cada uno de los carriles en los diferentes enlaces y el número total de carriles de cada enlace, se calcula el promedio de longitud de cola para cada enlace de acuerdo a la **Ecuación (20)**.

$$Dato\ de\ longitud\ de\ cola = \frac{\sum_{i=1}^{n} Promedio\ de\ vehículos\ en\ el\ carril_i}{n} \qquad (20)$$

Donde:

$n =$ Número total de carriles para cada enlace.

Por ejemplo, el enlace (9, 1) tiene 4 carriles y en cada uno de ellos tiene un número de vehículos promedio (carril 1 = 2, carril 2 = 1, carril 3 = 0, carril 4 = 3), entonces el dato de longitud de cola para este enlace sería.

$$Dato\ de\ longitud\ de\ cola = \frac{2+1+0+3}{4} = \frac{6}{4} = 1.5$$

Este valor de 1.5 indica que para el enlace (9, 1) en promedio se forma una longitud de colas de 1.5 vehículos, de esta manera, se calcula los datos existentes de

longitud de colas para cada enlace en los diferentes enlaces en los modelos de McTrans y Reno. Con respecto al modelo I-75, el simulador CORSIM no genera datos específicos para longitud de colas en el tráfico de las autopistas o avenidas (FRESIM), por tal razón, I-75 no se tuvo presente para evaluar la función objetivo con el componente de longitud de colas.

## 4.5 ANÁLISIS Y SELECCIÓN DE LOS PESOS DE LA FUNCIÓN OBJETIVO CON EL COMPONENTE DE LONGITUD DE COLAS

Al incluir el componente de longitud de colas en la función objetivo, es necesario definir los pesos que corresponden a volumen, velocidad y longitud de colas. Para esto se hicieron un total de 35 pruebas en el modelo de complejidad baja McTrans para determinar el mejor conjunto de pesos para aplicar la función objetivo. Estas pruebas se ejecutaron con el algoritmo meta-heurístico DECC-G, el cual ha mostrado los mejores resultados en esta investigación, con los parámetros definidos en la sección 4.2.1.

Las 35 pruebas se definieron de la siguiente manera: se formularon las pruebas manteniendo la relación de pesos de volumen y velocidad aplicada en [12] (70% para volumen y 30% para velocidad), es decir, se le quita peso a volumen y velocidad (manteniendo la proporción de 70% a 30%), para darle peso a longitud de colas. Se ejecutaron cuatro pruebas que tuvieran pesos cercanos a los que presentaron mejores resultados en las primeras diez pruebas. Se realizaron 21 pruebas teniendo en cuenta las ecuaciones diofánticas [57].

Una ecuación diofántica es un polinomio que tiene $n$ incógnitas (en este caso $n = 3$ porque son tres pesos que se van a buscar) que puede tener soluciones finitas, soluciones infinitas o tener una solución no trivial, de acuerdo a la forma del polinomio. Para este caso, la ecuación diofántica tiene soluciones infinitas y es representada por la restricción de la **Ecuación (1)**, es decir, $\alpha + \beta + \gamma = 1$. Al tener infinitas soluciones, se utiliza el concepto de maya que es valor porcentual para reducir los posibles valores que pueden tener las incógnitas, por ejemplo, una maya del 20% indica que una de las tres incógnitas (peso de volumen, peso de velocidad y peso de longitud de colas) no va cambiar su valor, una de las otras dos variables incrementará en 20% y la restante disminuirá el mismo porcentaje. De manera similar, se utiliza la maya del 30%.

La **Tabla 14** muestra el resultado de las 35 pruebas que se realizaron para encontrar los pesos más óptimos para la función objetivo.

| PRUEBA | PESOS | | | PROMEDIO NRMS | | | DISTANCIA AL PUNTO (0, 0, 0) |
|---|---|---|---|---|---|---|---|
| | VOLUMEN | VELOCIDAD | COLAS | VOLUMEN | VELOCIDAD | COLAS | |
| 1 | 70 | 30 | 0 | 0,013478 | 0,022890 | 0,053229 | 0,059489 |
| 2 | 70 | 20 | 10 | 0,014008 | 0,023534 | 0,031406 | 0,041670 |
| 3 | 60 | 20 | 20 | 0,015481 | 0,024083 | 0,025742 | 0,038501 |
| 4 | 60 | 10 | 30 | 0,015576 | 0,029168 | 0,024842 | 0,041358 |
| 5 | 60 | 0 | 40 | 0,017196 | 0,032933 | 0,020397 | 0,042383 |
| 6 | 50 | 0 | 50 | 0,018171 | 0,031490 | 0,020666 | 0,041819 |
| 7 | 40 | 0 | 60 | 0,018946 | 0,032544 | 0,019606 | 0,042455 |
| 8 | 30 | 0 | 70 | 0,021167 | 0,030135 | 0,018935 | 0,041408 |
| 9 | 20 | 0 | 80 | 0,020318 | 0,031435 | 0,021715 | 0,043273 |
| 10 | 10 | 0 | 90 | 0,019881 | 0,032754 | 0,019672 | 0,043070 |
| 11 | 50 | 40 | 10 | 0,016862 | 0,020115 | 0,034840 | 0,043621 |
| 12 | 80 | 10 | 10 | 0,012223 | 0,029076 | 0,038201 | 0,049539 |
| 13 | 45 | 10 | 45 | 0,018906 | 0,031566 | 0,020033 | 0,041895 |
| 14 | 40 | 30 | 30 | 0,018215 | 0,021007 | 0,023093 | 0,036144 |
| 15 | 0 | 20 | 80 | 0,022327 | 0,029122 | 0,018612 | 0,041145 |
| 16 | 0 | 40 | 60 | 0,022679 | 0,022924 | 0,022210 | 0,039156 |
| 17 | 0 | 60 | 40 | 0,022299 | 0,018570 | 0,025950 | 0,038929 |
| 18 | 0 | 80 | 20 | 0,021016 | 0,015926 | 0,031837 | 0,041339 |
| 19 | 0 | 100 | 0 | 0,022084 | 0,015485 | 0,049311 | 0,056205 |
| 20 | 20 | 20 | 60 | 0,018693 | 0,027406 | 0,019560 | 0,038511 |
| 21 | 20 | 40 | 40 | 0,020137 | 0,022614 | 0,023220 | 0,038159 |
| 22 | 20 | 60 | 20 | 0,020938 | 0,017917 | 0,027022 | 0,038595 |
| 23 | 20 | 80 | 0 | 0,020744 | 0,014869 | 0,046642 | 0,053169 |
| 24 | 40 | 20 | 40 | 0,019796 | 0,026675 | 0,018062 | 0,037811 |
| 25 | 40 | 40 | 20 | 0,017663 | 0,019344 | 0,027361 | 0,037879 |
| 26 | 40 | 60 | 0 | 0,017993 | 0,016750 | 0,050866 | 0,056495 |
| 27 | 60 | 40 | 0 | 0,013664 | 0,022493 | 0,056084 | 0,061952 |
| 28 | 80 | 0 | 20 | 0,013558 | 0,033605 | 0,031342 | 0,047911 |
| 29 | 100 | 0 | 0 | 0,009941 | 0,063969 | 0,091776 | 0,112311 |
| 30 | 0 | 30 | 70 | 0,020714 | 0,026954 | 0,018497 | 0,038701 |
| 31 | 0 | 90 | 10 | 0,023366 | 0,016414 | 0,038404 | 0,047857 |
| **32** | **30** | **30** | **40** | **0,019691** | **0,023633** | **0,017757** | **0,035518** |
| 33 | 30 | 60 | 10 | 0,019746 | 0,015777 | 0,038923 | 0,046410 |
| 34 | 60 | 30 | 10 | 0,014564 | 0,021641 | 0,032945 | 0,042022 |
| 35 | 90 | 0 | 10 | 0,010905 | 0,042327 | 0,047093 | 0,064251 |

***Tabla 14:*** *Resultados de los diferentes pesos aplicados en la función objetivo*

Cada prueba consta de 30 ejecuciones en el modelo McTrans con la meta-heurística DECC-G. La **Tabla 14** divide el valor promedio NRMS que obtuvo por cada componente, la última columna calcula la distancia que tuvo el punto (promedio NRMS de volumen, promedio NRMS de velocidad, promedio NRMS de colas) al punto de origen de coordenadas (0, 0, 0), esta distancia se calcula de acuerdo a la distancia euclidiana con la **Ecuación (21)**.

$$Distancia\ al\ punto\ (0,0,0) = \sqrt{V^2 + S^2 + Q^2} \qquad (21)$$

Donde:

$V =$ Promedio NRMS de volumen.

$S =$ Promedio NRMS de velocidad.

$Q =$ Promedio NRMS de longitud de colas.

Como el problema de calibración es de minimización, se optó por calcular la distancia al origen de coordenadas, por tal razón, la mejor prueba es la que tiene resultados más cercano al punto (0, 0, 0), en este caso la prueba 32. De acuerdo a lo anterior, el conjunto de pesos óptimos para la función objetivo es: peso para volumen = 30%, peso para velocidad = 30% y peso para longitud de colas = 40%. Estos pesos están en consonancia con la importancia que debe tener el componente de longitud de colas según lo presentado en el estado del arte [13] [14] [15].

## 4.6 RESULTADOS OBTENIDOS Y COMPARACIÓN APLICANDO LA FUNCIÓN OBJETIVO CON EL COMPONENTE DE LONGITUD DE COLAS

Para poner a prueba la función objetivo en la cual se considera un peso para la variable de longitud de colas, se aplicó el mismo procedimiento descrito anteriormente, cada uno de los algoritmos fue ejecutado 30 veces en cada una de los modelos presentados anteriormente. Los parámetros de los algoritmos son los mismos con los que se ejecutó la función objetivo contemplando solamente volumen y velocidad en los modelos de McTrans y Reno. Con respecto a los tiempos de ejecución, son similares a los de la fase uno de la experimentación ya que se utiliza la misma función objetivo. De este modo en la presente sección solo se indican los resultados que se obtuvieron de cada algoritmo aplicando los nuevos pesos en la función objetivo. Al final de la sección se realiza un análisis comparativo y estadístico entre los algoritmos meta-heurísticos propuestos y los presentados en el estado del arte.

### 4.6.1 Resultados y comparación para el modelo McTrans

A continuación se presentan los resultados que se obtuvieron para la red de complejidad baja McTrans con cada uno de los algoritmos meta-heurísticos propuestos y los presentados en la literatura.

#### 4.6.1.1 Resultados obtenidos con DECC-G

La ejecución de las 30 pruebas realizadas para el algoritmo meta-heurístico DECC-G en la red McTrans iniciaron con un NRMS de 0.456479, en ellas se obtuvo un promedio de NRMS de 0.020099, el mejor valor encontrado fue de 0.012159 y el peor valor encontrado fue de 0.025346, estos resultados arrojan una desviación estándar de 0.003264 lo cual indica poca dispersión en los valores de las 30 ejecuciones con respecto al promedio de NRMS. Teniendo en cuenta el valor inicial y promedio de NRMS, DECC-G logra una mejora del 95.60% aproximadamente.

#### 4.6.1.2 Resultados obtenidos con MOS

La ejecución de las 30 pruebas realizadas para el algoritmo meta-heurístico MOS en el modelo McTrans iniciaron con un NRMS de 0.456479, en ellas se obtuvo un promedio de NRMS de 0.034282, el mejor valor encontrado fue de 0.016222 y el peor valor encontrado fue de 0.062056, estos resultados arrojan una desviación estándar de 0.011123 lo cual indica mayor dispersión en los valores de las 30 ejecuciones con respecto al promedio de NRMS. Teniendo en cuenta el valor inicial y promedio de NRMS, MOS logra una mejora del 92.49% aproximadamente.

#### 4.6.1.3 Resultados obtenidos con MA-SW-Chains

La ejecución de las 30 pruebas realizadas para el algoritmo meta-heurístico MA-SW-Chains en el modelo McTrans iniciaron con un NRMS de 0.456479, en ellas se obtuvo un promedio de NRMS de 0.031406, el mejor valor encontrado fue de 0.018773 y el peor valor encontrado fue de 0.052120, estos resultados arrojan una desviación estándar de 0.008404 lo cual indica poca dispersión en los valores de las 30 ejecuciones con respecto al promedio de NRMS. Teniendo en cuenta el valor inicial y promedio de NRMS, MOS logra una mejora del 93.12% aproximadamente.

#### 4.6.1.4 Comparación de los algoritmos propuestos con GASA Y SPSA

Con respecto a los algoritmos del estado del arte GASA y SPSA, cada uno de ellos también ser realizaron 30 ejecuciones para este modelo de complejidad baja, donde GASA obtuvo un promedio de NRMS de 0.032590, el mejor valor encontrado fue de 0.0161140 y el peor valor encontrado fue de 0.052067, estos resultados arrojan una desviación estándar de 0.008507 lo cual indica poca dispersión en los valores de las 30 ejecuciones con respecto al promedio de NRMS. Teniendo en cuenta el valor inicial y promedio de NRMS, GASA logra una mejora del 92.86% aproximadamente.

El algoritmo SPSA registró un promedio de NRMS de 0.097927, el mejor valor encontrado fue de 0.051916 y el peor valor encontrado fue de 0.218370, estos resultados arrojan una desviación estándar de 0.047274 lo cual indica una dispersión mayor con respecto a los algoritmos meta-heurísticos propuestos en los valores de las 30 ejecuciones con respecto al promedio de NRMS. Teniendo en cuenta el valor inicial y promedio de NRMS, SPSA logra una mejora del 78.55% aproximadamente.

Se tuvieron en cuenta las mismas condiciones establecidas en la fase 1 de la experimentación con los tres modelos de prueba, con el fin de realizar una

comparación justa entre los diferentes algoritmos. Al aplicar un nuevo peso en el componente de longitud de colas en función objetivo, el valor inicial de NRMS para todos los algoritmos fue de 0.456479, aumentando el valor con respecto a la función objetivo que originalmente sólo contemplaba volumen y velocidad. La cantidad de evaluaciones de la función objetivo está entre el rango de 5200 y 5400. Finalmente, se estableció la misma cantidad de hilos que podría ejecutar cada algoritmo durante su ejecución. El comportamiento de cada algoritmo se evidencia en la **Figura 48**.

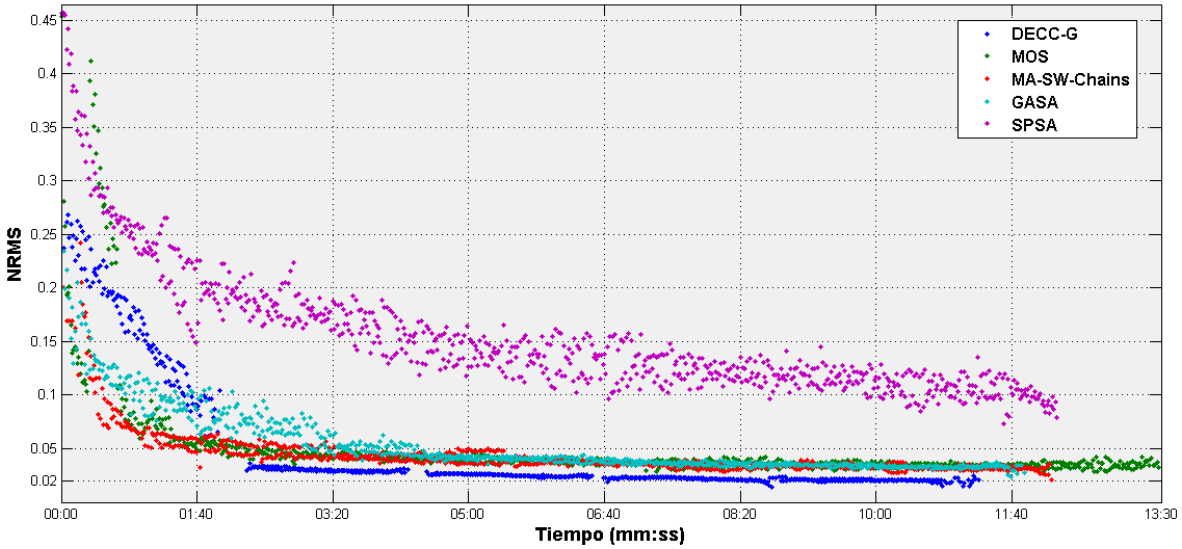



***Figura 48:*** *Gráfica NRMS vs Tiempo en el modelo McTrans entre los algoritmos propuestos y los del estado del arte (volumen, velocidad y longitud de colas)*

Como se observa, los algoritmos que convergen rápidamente y obtiene mejores resultados en la parte inicial del proceso de calibración son MOS y MA-SW-Chains, aunque MA-SW-Chains alcanza mejores resultados de NRMS en menos tiempo a lo largo del proceso de calibración que MOS y GASA. Sin embargo, GASA termina el proceso de calibración en menos tiempo. Con respecto a SPSA, es el algoritmo que presenta mayor dispersión en sus resultados y el que encuentra valores de NRMS menos óptimos, además tiene un tiempo de ejecución similar a MA-SW-Chains. Se evidencia claramente que el algoritmo ganador es el DECC-G teniendo en cuenta no solo el mejor valor encontrado en NRMS sino también en promedio y tiempo. DECC-G alcanza soluciones mucho más óptimas hasta el final del proceso de calibración. Por lo anterior, se afirma que DECC-G es el mejor algoritmo mostrando un promedio NRMS menor, menos tiempo de ejecución y la menor desviación estándar.

Los algoritmos meta-heurísticos propuestos cumplen satisfactoriamente con el criterio de calibración, en el cual el 100% de los enlaces logra tener un GEH menor a 5, partiendo que el valor inicial de GEH es menor a 5 para el 55% de los enlaces. La **Figura 49** muestra el GEH de las mejores ejecuciones de cada algoritmo meta-heurístico propuesto.

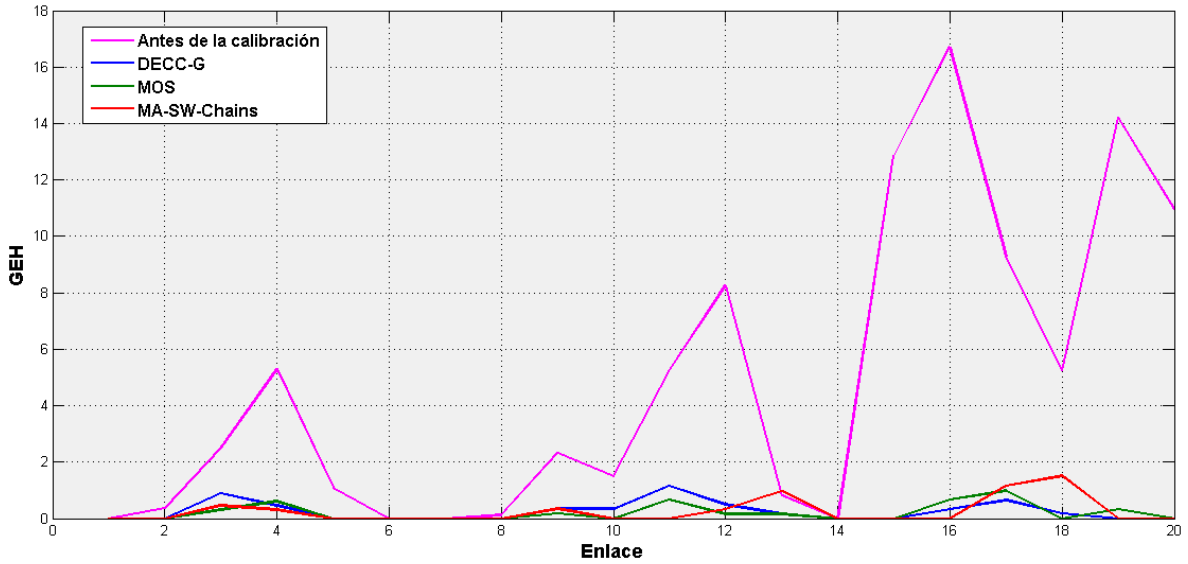



***Figura 49:*** *Gráfica GEH vs Enlace en el modelo McTrans de los algoritmos meta-heurísticos propuestos (volumen, velocidad y longitud de colas)*

La **Figura 50** ilustra el GEH de las mejores ejecuciones del algoritmo meta-heurístico DECC-G y los algoritmos encontrados en el estado del arte, donde se ve claramente reflejado que los algoritmos obtienen un valor de GEH menor a 5 para el 100% de los enlaces del modelo McTrans, cumpliendo con el criterio de calibración de esta investigación.

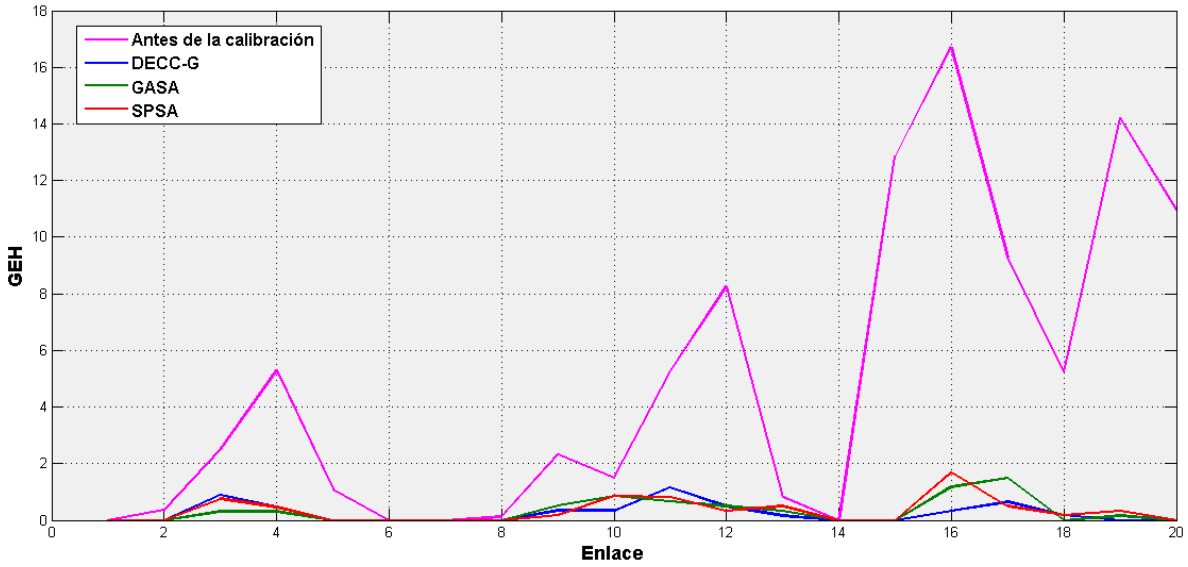



***Figura 50:*** *Gráfica GEH vs Enlace en el modelo McTrans de DECC-G y de los algoritmos del estado del arte (volumen, velocidad y longitud de colas)*

La **Tabla 15** resume los resultados de los algoritmos propuestos y los del estado del arte para el modelo de baja complejidad McTrans con los componentes de volumen, velocidad y longitud de colas.

| ALGORITMO | NRMS | | | | GEH DEL MEJOR RESULTADO | TIEMPO PROMEDIO (mm:ss) |
|---|---|---|---|---|---|---|
| | PROMEDIO | MEJOR RESULTADO | PEOR RESULTADO | DESVIACIÓN ESTÁNDAR | | |
| DECC-G | 0.020099 | 0.012159 | 0.025346 | 0.003264 | 100% | 11:16 |
| MOS | 0.034282 | 0.016222 | 0.062056 | 0.011123 | 100% | 13:28 |
| MA-SW-Chains | 0.031406 | 0.018773 | 0.052120 | 0.008404 | 100% | 12:09 |
| GASA | 0.032590 | 0.016114 | 0.052067 | 0.008507 | 100% | 11:48 |
| SPSA | 0.097927 | 0.051916 | 0.218370 | 0.047274 | 100% | 12:13 |

***Tabla 15:*** *Resultados en el modelo McTrans (volumen, velocidad y longitud de colas)*

#### 4.6.1.5 Análisis estadístico

Con la función objetivo incluyendo el componente de longitud de colas y las 30 ejecuciones realizados por cada algoritmo en la red de complejidad baja, se realizó el mismo análisis estadístico presentado en las secciones anteriores. Los resultados de la prueba de Wilcoxon se evidencian en la **Figura 51**.

| | (1) | (2) | (3) | (4) | (5) |
|---|---|---|---|---|---|
| DECC-G (1) | - | ● | ● | ● | ● |
| MOS (2) | ○ | - | | | ● |
| MA-SW-Chains (3) | ○ | | - | | ● |
| GASA (4) | ○ | | | - | ● |
| SPSA (5) | ○ | ○ | ○ | ○ | - |

***Figura 51:*** *Resultados de la prueba de Wilcoxon para el modelo McTrans (volumen, velocidad y longitud de colas)*

Como se puede apreciar, el algoritmo que claramente domina a todos los algoritmos es el DECC-G. A diferencia de la primera prueba realizada en la función objetivo contemplando solamente volumen y velocidad, esta vez no se puede establecer una dominancia entre dos de los algoritmos meta-heurísticos propuestos (MOS y MA-SW-Chains) con el mejor reportado en la literatura (GASA). Sin embargo, la prueba revela que todos los algoritmos meta-heurísticos propuestos en la presente investigación dominan al algoritmo estocástico SPSA.

Adicionalmente, se realizó la prueba de Friedman en la cual se busca establecer un ranking de los algoritmos asignando el menor valor al algoritmo que presenta mejores resultados. Los resultados obtenidos se muestran en la **Tabla 16**, en la cual se evidencia claramente que el mejor algoritmo entre los propuestos y los reportados en el estado del arte es el algoritmo meta-heurístico DECC-G, ratificando así los resultados obtenidos por la anterior prueba y los resultados descritos anteriormente. Esta vez sí es posible notar que todos los algoritmos meta-

heurísticos propuestos en la presente investigación superan los resultados de los algoritmos encontrados en el estado del arte. A diferencia de la anterior prueba, en esta se puede notar que los algoritmos propuestos MOS y MA-SW-Chains superan por una pequeña diferencia al algoritmo GASA, lo que permite establecer junto con la prueba de Wilcoxon que de los algoritmos meta-heurísticos propuestos el mejor es el DECC-G seguido por el algoritmo MOS.

| ALGORITMO | RANKING |
|---|---|
| DECC-G | 1.3 |
| MA-SW-Chains | 2.7333 |
| MOS | 2.8667 |
| GASA | 3.1 |
| SPSA | 5 |

***Tabla 16:*** *Resultados de la prueba de Friedman para el modelo McTrans (volumen, velocidad y longitud de colas)*

### 4.6.2 Resultados y comparación para el modelo Reno

A continuación se presentan los resultados que se obtuvieron para el modelo de complejidad media Reno con cada uno de los algoritmos propuestos y los presentados en la literatura.

#### 4.6.2.1 Resultados obtenidos con DECC-G

La ejecución de las 30 pruebas realizadas iniciaron con un NRMS de 1.424487, en ellas se obtuvo un promedio de NRMS de 0.137525, el mejor valor encontrado fue de 0.128027 y el peor valor encontrado fue de 0.143980, estos resultados arrojan una desviación estándar de 0.004188 lo cual indica poca dispersión en los valores de las 30 ejecuciones con respecto al promedio de NRMS. Teniendo en cuenta el valor inicial y promedio de NRMS, DECC-G logra una mejora del 90.35% aproximadamente.

#### 4.6.2.2 Resultados obtenidos con MOS

La ejecución de cada uno de las 30 ejecuciones iniciaron con un NRMS de 1.424487, en ellas se obtuvo un promedio NRMS de 0.171576, el mejor valor encontrado fue de 0.134442 y el peor valor encontrado fue de 0.333030, estos resultados arrojan una desviación estándar de 0.044118 lo cual indica mayor dispersión en los valores registrados en las 30 ejecuciones con respecto al promedio de NRMS. Teniendo en cuenta el valor inicial y promedio de NRMS, DECC-G logra una mejora del 87.96% aproximadamente.

#### 4.6.2.3 Resultados obtenidos con MA-SW-Chains

Los resultados que se obtuvieron al realizar las 30 ejecuciones con MA-SW-Chains fueron los siguientes: un promedio de NRMS de 0.224657, el mejor valor de NRMS registrado fue de 0.135373 y el peor valor de 0.363337. Por otro lado, estos

resultados arrojan una desviación estándar de 0.069602 lo cual indica mayor dispersión en los valores registrados en las 30 ejecuciones con respecto al promedio de NRMS. Teniendo en cuenta el valor inicial y promedio de NRMS, DECC-G logra una mejora del 84.23% aproximadamente.

#### 4.6.2.4 Comparación de los algoritmos propuestos

Con respecto a los algoritmos del estado del arte GASA y SPSA, cada uno de ellos también ser realizaron 30 ejecuciones para este modelo de complejidad media, donde GASA obtuvo un promedio de NRMS de 0.227865, el mejor valor encontrado fue de 0.145302 y el peor valor encontrado fue de 0.380963, estos resultados arrojan una desviación estándar de 0.064727 lo cual indica mayor dispersión en los valores registrados en las 30 ejecuciones con respecto al promedio de NRMS. Teniendo en cuenta el valor inicial y promedio de NRMS, GASA logra una mejora del 84% aproximadamente. El algoritmo SPSA obtuvo un promedio de NRMS de 0.817488, el mejor valor encontrado fue de 0.585524 y el peor valor encontrado fue de 1.098432, estos resultados arrojan una desviación estándar de 0.149849 lo cual indica mayor dispersión mayor con respecto a los algoritmos meta-heurísticos propuestos en los valores de las 30 ejecuciones con respecto al promedio de NRMS. Teniendo en cuenta el valor inicial y promedio de NRMS, SPSA logra una mejora del 42.61% aproximadamente.

Se tuvieron en cuenta las mismas condiciones establecidas en la fase 1 de la experimentación con los tres modelos de prueba, con el fin de realizar una comparación justa entre los diferentes algoritmos. Al incluir el componente de longitud de colas, el valor inicial de NRMS para todos los algoritmos en la red Reno fue de 1.424487, aumentando respecto a la función objetivo que solo consideraba volumen y velocidad. El comportamiento de cada algoritmo propuesto se evidencia en la **Figura 52**.

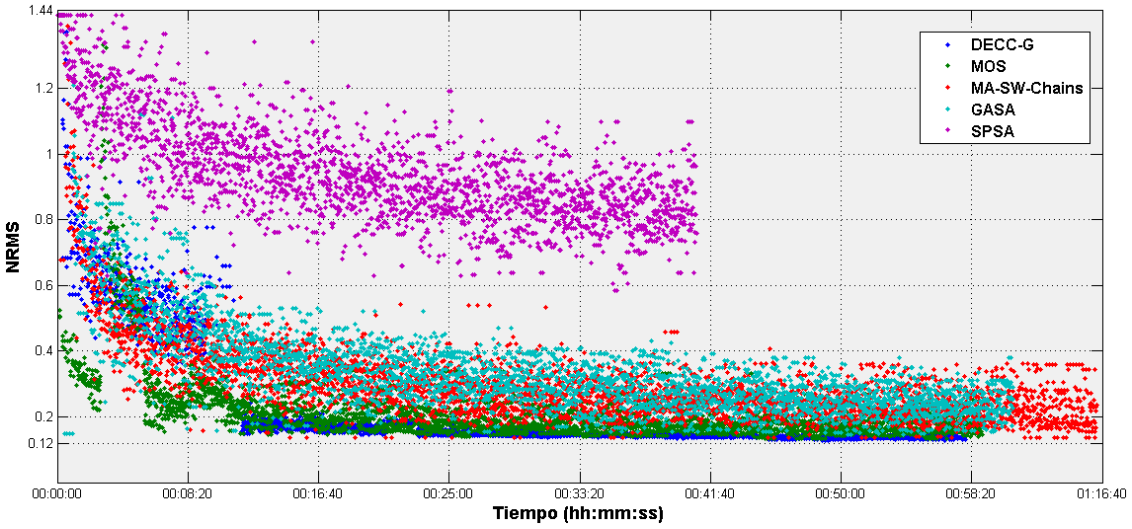



***Figura 52:*** *Gráfica NRMS vs Tiempo en el modelo Reno entre los algoritmos propuestos y los del estado del arte (volumen, velocidad y longitud de colas)*

El algoritmo meta-heurístico que converge rápidamente y obtiene mejores resultados en la parte inicial del proceso de calibración es MOS, alcanzado mejores resultados que MA-SW-Chains y GASA en menor tiempo. MA-SW-Chains y GASA muestran una dispersión considerable en los resultados obtenidos, aunque MA-SW-Chains alcanza mejores resultados en mayor tiempo de ejecución. Con respecto a SPSA, es el algoritmo que presenta mayor dispersión en sus datos y es el peor algoritmo para encontrar resultados de NRMS, aunque su tiempo es menor a comparación de los demás al cumplir el criterio de parada establecido para todos los algoritmos. Si bien los resultados en la gráfica anterior no definen un claro ganador, los resultados descritos anteriormente permiten concluir que el algoritmo que obtiene mejores resultados tanto en promedio de NRMS como la mejor solución encontrada, es el algoritmo propuesto DECC-G. El único algoritmo que logra estabilizarse (menor dispersión) luego de aproximadamente 12 minutos y obtener las soluciones más prometedoras durante el proceso de calibración es el algoritmo propuesto DECC-G. Por lo anterior, se afirma que DECC-G es el mejor algoritmo mostrando un promedio NRMS menor, menos tiempo de ejecución y la menor desviación estándar.

Los algoritmos meta-heurísticos propuestos cumplen satisfactoriamente con el criterio de calibración, partiendo que el valor inicial de GEH es menor a 5 para el 47% aproximadamente de los enlaces. DECC-G logra aproximadamente el 98% de los enlaces tengan un GEH menor a 5, cumpliendo con el criterio de calibración propuesto en esta investigación. Con respecto a MOS y MA-SW-Chains cumplen satisfactoriamente con el criterio de calibración, en el cual el 100% de los enlaces logra tener un GEH menor a 5. La **Figura 53** muestra el GEH de las mejores ejecuciones de cada algoritmo meta-heurístico propuesto.

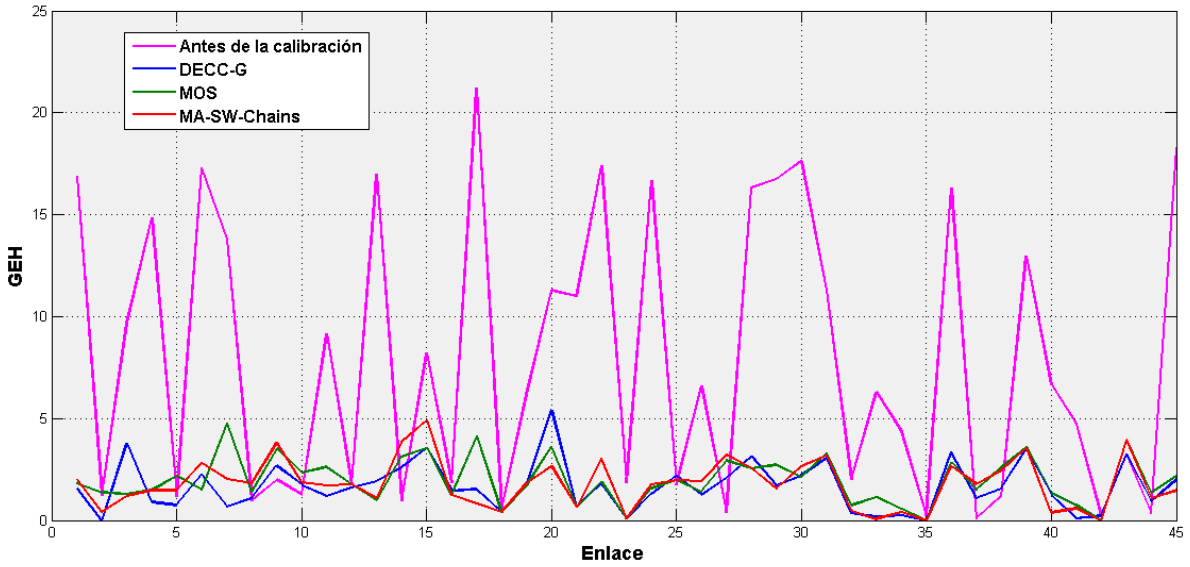



***Figura 53:*** *Gráfica GEH vs Enlace en el modelo Reno de los algoritmos meta-heurísticos propuestos (volumen, velocidad y longitud de colas)*

La **Figura 54** ilustra el GEH de las mejores ejecuciones del algoritmo meta-heurístico DECC-G y los algoritmos encontrados en el estado del arte, donde se ve que DECC-G logra aproximadamente el 98% de los enlaces tengan un GEH menor a 5, cumpliendo con el criterio de calibración, GASA obtienen un valor de GEH menor a 5 para el 100% de los enlaces del modelo Reno cumpliendo con el criterio de calibración, mientras que SPSA obtiene un valor de GEH menor a 5 para el 48% aproximadamente.

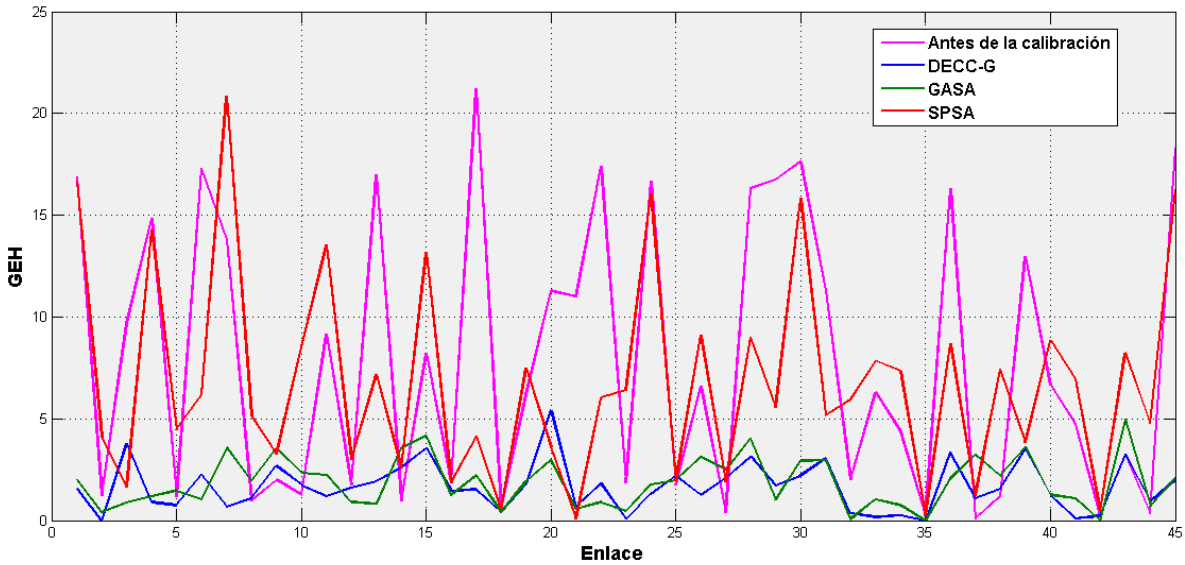



***Figura 54:*** *Gráfica GEH vs Enlace en el modelo Reno de DECC-G con los algoritmos del estado del arte (volumen, velocidad y longitud de colas)*

La **Tabla 17** resume los resultados de los algoritmos propuestos y los algoritmos encontrados en el estado del arte para el modelo de complejidad media Reno con los componentes de volumen, velocidad y longitud de colas.

| ALGORITMO | NRMS | | | | GEH DEL MEJOR RESULTADO | TIEMPO PROMEDIO (hh:mm:ss) |
|---|---|---|---|---|---|---|
| | PROMEDIO | MEJOR RESULTADO | PEOR RESULTADO | DESVIACIÓN ESTÁNDAR | | |
| DECC-G | 0.137525 | 0.128027 | 0.143980 | 0.004188 | 98% | 00:57:58 |
| MOS | 0.171576 | 0.134442 | 0.333030 | 0.044118 | 100% | 00:58:58 |
| MA-SW-Chains | 0.224657 | 0.135373 | 0.363337 | 0.069602 | 100% | 01:06:18 |
| GASA | 0.227865 | 0.145302 | 0.380963 | 0.064727 | 100% | 01:00:56 |
| SPSA | 0.817488 | 0.585524 | 1.098432 | 0.149849 | 48% | 00:40:47 |

***Tabla 17:*** *Resultados en el modelo Reno (volumen, velocidad y longitud de colas)*

Aunque DECC-G obtiene un 98% de sus enlaces un valor menor a 5, y no supera los resultados obtenidos por MOS, MA-SW-Chains y GASA, DECC-G se considera mejor que los demás algoritmos ya que cumple con el criterio de calibración de esta

investigación y encuentra mejores resultados de NRMS en menor tiempo, en promedio y estos resultados son menos dispersos.

#### 4.6.2.5 Análisis estadístico

Con el fin de determinar claramente cual algoritmo es más eficaz se realiza a igual que los análisis anteriores en las diferentes redes un análisis estadístico aplicando las pruebas de Wilcoxon y Friedman. Los resultados de las pruebas de Wilcoxon y Friedman son presentados en la **Figura 55** y **Tabla 18** respectivamente.

| | (1) | (2) | (3) | (4) | (5) |
|---|---|---|---|---|---|
| DECC-G (1) | - | ● | ● | ● | ● |
| MOS (2) | ○ | - | ● | ● | ● |
| MA-SW-Chains (3) | ○ | ○ | - | | ● |
| GASA (4) | ○ | ○ | | - | ● |
| SPSA (5) | ○ | ○ | ○ | ○ | - |

***Figura 55:*** *Resultados de la prueba de Wilcoxon para el modelo Reno (volumen, velocidad y longitud de colas)*

| ALGORITMO | RANKING |
|---|---|
| DECC-G | 1.1333 |
| MOS | 2.2333 |
| MA-SW-Chains | 3.2667 |
| GASA | 3.3667 |
| SPSA | 5 |

***Tabla 18:*** *Resultados de la prueba de Friedman para el modelo Reno (volumen, velocidad y longitud de colas)*

En las dos pruebas estadísticas, el algoritmo DECC-G es el ganador, es dominante sobre el resto de algoritmos y obtiene el puesto número uno en el ranking de Friedman. Sin ninguna duda, con este resultado se puede afirmar que el algoritmo DECC-G es el algoritmo más eficaz para resolver el problema de calibración sin importar el tamaño de la red y teniendo en cuenta los tres componentes de volumen, velocidad y longitud de colas en la función objetivo. Del mismo modo, el algoritmo meta-heurístico MOS logra superar los resultados de los algoritmos reportados en la literatura y al algoritmo propuesto MA-SW-Chains.

Aunque en la prueba de Wilcoxon no se logra establecer una dominancia entre MA-SW-Chains y GASA, con la prueba de Friedman es posible afirmar que todos los algoritmos meta-heurísticos propuestos logran superar a los reportados en el estado del arte.

## 4.7 ANÁLISIS DE RESULTADOS

En los resultados anteriores, se evidencia la superioridad del algoritmo meta-heurístico DECC-G sobre las otras meta-heurísticas, MOS y MA-SW-Chains, y sobre los algoritmos del estado del arte, GASA y SPSA, independientemente de los componentes que se tengan en cuenta (volumen, velocidad y longitud de colas), DECC-G encuentra mejores valores de NRMS en menos tiempo de ejecución con respecto a los demás algoritmos, aunque en los modelos Reno e I-75, SPSA tuvo un menor tiempo de ejecución que DECC-G. MOS y MA-SW-Chains son mejores encontrando valores de NRMS que SPSA, independientemente de los componentes que se tengan en cuenta pero SPSA demora menos tiempo para ejecutar la calibración.

MOS encuentra mejores valores de NRMS que GASA en las redes de complejidad baja y media, en la red de complejidad alta, GASA es mejor que MOS, a nivel general, GASA ejecuta el proceso de calibración más rápido que MOS, independientemente de los componentes que se tengan en cuenta en la función objetivo. MA-SW-Chains es mejor encontrando valores NRMS que GASA teniendo en cuenta los tres componentes en la función objetivo, pero su proceso de optimización es más demorado que GASA.

Con respecto al criterio de calibración, los algoritmos propuestos lo cumplen satisfactoriamente en todos los modelos sin importar los componentes que se tengan en cuenta en la función objetivo.

Al observar los porcentajes de mejora en las dos fases de experimentación (fase uno contemplando los componentes de volumen y velocidad, fase dos contemplando los componentes de volumen, velocidad y longitud de colas), es notorio que al incluir el componente de longitud de colas, los algoritmos meta-heurísticos propuestos logran mayores porcentajes de mejora en su valor de NRMS.

## 4.8 VALIDACIÓN INTERNA Y EXTERNA DE LOS EXPERIMENTOS

Con el fin de excluir las explicaciones alternativas de los resultados obtenidos en esta investigación, se realizó primero una validación interna de los experimentos. En términos de los resultados simulados obtenidos, se tiene como variable independiente (VI) los parámetros que conforman el modelo, los cuales fueron obtenidos por cada uno de los diferentes algoritmos, y por otro lado, se tiene la variable dependiente (VD) que serán los resultados obtenidos o datos simulados por el simulador CORSIM al procesar los parámetros del modelo.

Dado que el simulador CORSIM es un software de caja negra desarrollado por terceros, usado a nivel mundial desde hace varios años, no es posible conocer los procesos internos que realiza para obtener los datos de la simulación. Sin embargo, este software cuenta con diferentes parámetros de configuración que podrían afectar la experimentación realizada. En esta investigación, se realizó toda la experimentación con la configuración por defecto del software para todos los

algoritmos propuestos y del estado del arte, buscando que la comparación sólo dependiera de los algoritmos por sí mismos. En este sentido, solo los parámetros obtenidos por cada uno de los algoritmos determinan los resultados obtenidos o el valor de la variable dependiente, teniendo en cuenta que son las únicas variables que cambian cuando el simulador se ejecuta para generar sus datos.

Ahora bien, en términos de los parámetros obtenidos por cada algoritmo, para garantizar la repetitividad del experimento se aplicó el patrón singleton para tener una sola instancia del objeto Random (en Java), el cual genera mediante una semilla o valor numérico, los diferentes valores aleatorios utilizados en las operaciones internas de los diferentes algoritmos. Cada una de las 30 ejecuciones mencionadas anteriormente por cada algoritmo en cada red, fue realizada aplicando una semilla diferente (1 al 30). En este sentido, haciendo uso de la semilla, es posible obtener nuevamente los mismos resultados por cada algoritmo aplicado en la experimentación en términos del valor del NRMS y bajo la misma configuración de parámetros, obtenidos mediante el afinamiento de parámetros. El uso o no de la técnica de hilos de ejecución en paralelo y la cantidad de procesos externos que se estén ejecutando en la maquina donde se realizan las pruebas, determina el tiempo de ejecución de cada algoritmo, dado que el único criterio de parada utilizado en la investigación está definido por la cantidad de evaluaciones de la función objetivo que podrá realizar cada uno de ellos.

Lo anterior, permite afirmar que no existen amenazas provenientes de factores organísmicos ni situacionales, dado que todos los procesos anteriormente mencionados son repetibles bajo la misma configuración de parámetros y en el proceso no intervienen personas o sujetos que puedan alterar dichos resultados. Sin embargo, se considera una amenaza proveniente de la medida de respuesta, dado que en la captura de los datos reales o de campo, podría haber problemas relacionados con el uso de equipamiento o instrumentación utilizada para tal fin, lo cual indica que la variable dependiente podría ser medida de diversas maneras, y el proceso de calibración estaría basado en datos poco reales y al comparar los resultados obtenidos por la simulación y los capturados en campo, los resultados no serán los esperados, hecho que afecta a todos los algoritmos, propuestos y del estado del arte.

Dado los resultados obtenidos, se puede afirmar que la experimentación aplicada en las 3 redes provistas, no permite a los autores asegurar que el ranking de los algoritmos (basado en la prueba de Friedman) o la dominancia de los resultados (basado en la prueba de Wilcoxon) serán iguales en otros modelos de más o menos complejidad, ya que el proceso de afinamiento de los parámetros de los algoritmos incide en los resultados del proceso de calibración que cada algoritmo realiza. Sin embargo, es preciso comentar que el proceso de afinamiento realizado en esta investigación se realizó sobre la red de complejidad baja y los resultados sobre las otras redes fueron prometedores, lo que permite esperar que al realizar el

afinamiento sobre cada modelo antes de realizar la calibración, los resultados sean mejores a los presentados en este trabajo.

# Capítulo 5

## 5 CONCLUSIONES Y TRABAJOS FUTUROS

### 5.1 CONCLUSIONES

En este proyecto de investigación se adaptaron tres algoritmos meta-heurísticos, a saber: coevolución cooperativa basada en evolución diferencial (DECC-G), muestreo múltiple de descendientes (MOS) y algoritmo memético basado en encadenamiento de búsquedas locales (MA-SW-Chains), para la calibración de modelos de flujo de tráfico vehicular CORSIM. Estos algoritmos meta-heurísticos fueron evaluados sobre tres modelos CORSIM de diferente nivel de complejidad (baja, medio y alta) con los componentes de volumen, velocidad y longitud de colas. Los resultados de estas evaluaciones muestran que los algoritmos meta-heurísticos cumplen con el criterio de calibración (GEH) en los tres modelos. Los resultados obtenidos indican que el algoritmo meta-heurístico DECC-G obtiene mejores resultados con respecto a NRMS y menor tiempo de ejecución, lo cual es confirmado con los análisis estadísticos de Wilcoxon y Friedman, aunque en los modelos Reno e I-75, SPSA tiene un menor tiempo de ejecución que DECC-G.

Con respecto a los algoritmos del estado del arte GASA y SPSA (a estos algoritmos se les adaptó el uso de hilos en paralelo para hacer una comparación justa con los algoritmos meta-heurísticos propuestos), en los tres modelos de prueba los análisis estadísticos de Wilcoxon y Friedman muestran que DECC-G encuentra mejores soluciones que GASA y SPSA, además que MOS y MA-SW-Chains son mejores que SPSA. Para el modelo de complejidad mediana, los análisis estadísticos muestran que DECC-G y MOS son mejores que GASA, además que MA-SW-Chains es mejor que SPSA. Para el modelo de complejidad baja, los análisis estadísticos indican que los algoritmos meta-heurísticos propuestos encuentran mejores soluciones que GASA y SPSA. Con esto se demuestra que el mejor algoritmo es DECC-G, el cual es comparado en relación al tiempo de ejecución para realizar la calibración con los algoritmos del estado del arte GASA y SPSA, DECC-G realiza el proceso de calibración en menos tiempo que GASA en los tres modelos CORSIM, con respecto a SPSA, DECC-G es más demorado en el proceso de calibración en las redes de complejidad media y alta que SPSA, aunque DECC-G es mucho mejor que SPSA encontrado valores menores de NRMS. En la red pequeña, DECC-G realiza el proceso de calibración en menos tiempo que SPSA. Lo anterior es con base a los componentes de volumen y velocidad.

Con el propósito de representar de una mejor manera los modelos de prueba, en este proyecto de investigación se definió una nueva función objetivo que contempla la función propuesta en [12] e incluyó el componente de longitud de colas. Para evaluar esta nueva función objetivo con los algoritmos meta-heurísticos propuestos,

se utilizaron dos modelos CORSIM de complejidad baja y media. Los resultados de estas evaluaciones con la función objetivo incluyendo longitud de colas muestran que los algoritmos meta-heurísticos cumplen con el criterio de calibración (GEH) en los tres modelos. Además, los resultados indican que el algoritmo meta-heurístico DECC-G obtiene mejores resultados con respecto a NRMS y menor tiempo de ejecución, lo cual es confirmado con los análisis estadísticos de Wilcoxon y Friedman.

Con respecto a los algoritmos del estado del arte GASA y SPSA, en los tres modelos de prueba los análisis estadísticos de Wilcoxon y Friedman muestran que DECC-G encuentra mejores soluciones que GASA y SPSA, además que MOS y MA-SW-Chains son mejores que SPSA. Para el modelo de complejidad media, los análisis estadísticos muestran que DECC-G y MOS son mejores que GASA, además que MA-SW-Chains es mejor que SPSA. Para el modelo de complejidad baja, los análisis estadísticos indican que los algoritmos meta-heurísticos propuestos encuentran mejores soluciones que GASA y SPSA. Con esto se demuestra que el mejor algoritmo es DECC-G, bajo las condiciones presentadas en la experimentación, el cual es comparado en relación al tiempo de ejecución para realizar la calibración con los algoritmos del estado del arte GASA y SPSA, DECC-G realiza el proceso de calibración en menos tiempo que GASA en los tres modelos CORSIM, con respecto a SPSA, DECC-G es más demorado en el proceso de calibración en las redes de complejidad media y alta que SPSA, aunque DECC-G es mucho mejor que SPSA encontrado valores menores de NRMS. En la red pequeña, DECC-G realiza el proceso de calibración en menos tiempo que SPSA. Lo anterior es con base a los componentes de volumen, velocidad y longitud de colas.

Al incluir el componente de longitud de colas, los algoritmos meta-heurísticos propuestos logran mayores porcentajes de mejora en su valor de NRMS, aunque empiecen con valor NRMS inicial mayor, lo cual es esperado por que se incluye un componente adicional que acumula error.

El éxito por el que DECC-G encuentra mejores soluciones en comparación a los otros dos algoritmos meta-heurísticos propuestos (MOS y MA-SW-Chains) y algoritmos del estado del arte (GASA y SPSA) se debe a su enfoque de divide y vencerás, es decir que DECC-G no toma el problema de calibración como un todo, sino que divide en varios subcomponentes todos los parámetros que se deben calibrar y va optimizando cada uno de ellos como un problema aparte, además de optimizar algunas soluciones con vectores de pesos, factor que es relevante para salir de estancamientos de óptimos locales. Cabe resaltar que DECC-G muestra su mejor desempeño en funciones totalmente separables [19] [16], por esto se considera que el problema de calibración de flujo de tráfico vehicular a pesar de usar simulación es un problema separable en relación con los parámetros que se calibran. Esto tiene mucha concordancia con la experiencia de los expertos, ya que en este problema el usuario selecciona los parámetros que desea calibrar, lo que

ocasiona variabilidad en el tamaño del individuo que va dar solución al problema de calibración, es decir que independientemente el número de parámetros que se van a calibrar, la representación de la solución objetivo puede ser evaluada en la misma función objetivo.

## 5.2 TRABAJOS FUTUROS

En este proyecto de investigación se utilizaron tres modelos CORSIM de prueba para evaluar los algoritmos meta-heurísticos propuestos y dos modelos para evaluar la nueva función objetivo con el componente de longitud de colas, por esta razón es conveniente realizar pruebas en modelos mucho más grandes, es decir que tengan más enlaces, datos reales para el componente de longitud de colas y realizar experimentos con estos nuevos modelos, para observar y analizar el comportamiento de DECC-G, MOS y MA-SW-Chains con la nueva función objetivo.

Por restricciones de tiempo para el desarrollo del proyecto, el afinamiento de parámetros de los algoritmos meta-heurísticos propuestos se realizó sólo con el modelo de complejidad baja y el mejor conjunto de parámetros de cada algoritmo en este modelo, se utilizó para las demás redes. Para lograr mejores resultados, es necesario afinar los parámetros de las meta-heurísticas por cada modelo ya que el número de parámetros que se van a calibrar dependen del modelo y los valores son muy variantes.

Como trabajo futuro, el grupo de investigación pretende adaptar un nuevo algoritmo que combinen las principales características del mejor algoritmo meta-heurístico propuesto (DECC-G) y del algoritmo Mejor Búsqueda Global Armónica Multi-Objetivo (MOGBHS), ya que este último algoritmo presentó muy buenos resultados en el problema de calibración de modelos de flujo de tráfico vehicular CORSIM, proyecto desarrollado en paralelo a esta investigación.

# Capítulo 6

## 6 BIBLIOGRAFÍA


[1] S. Robinson, *Simulation: the practice of model development and use*. Palgrave Macmillan, 2014.

[2] E. J. Miller, J. D. Hunt, J. E. Abraham, and P. A. Salvini, "Microsimulating urban systems," *Computers, environment and urban systems,* vol. 28, no. 1, pp. 9-44, 2004.

[3] S. Kim, W. Suh, and J. Kim, "Traffic Simulation Software: Traffic Flow Characteristics in CORSIM," in *Information Science and Applications (ICISA), 2014 International Conference on*, 2014, pp. 1-3: IEEE.

[4] Y. Hollander and R. Liu, "The principles of calibrating traffic microsimulation models," *Transportation,* vol. 35, no. 3, pp. 347-362, 2008.

[5] A. Zamuda, J. Brest, and B. Bošković, "Large scale global optimization using differential evolution with self-adaptation and cooperative co-evolution," in *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*, 2008, pp. 3718-3725: IEEE.

[6] B. K. Abdalhaq and M. I. A. Baker, "Using meta heuristic algorithms to improve traffic simulation," *Journal of Algorithms,* vol. 2, no. 4, pp. 110-128, 2014.

[7] J.-B. Lee and K. Ozbay, "Calibration of a macroscopic traffic simulation model using enhanced simultaneous perturbation stochastic approximation methodology," in *Transportation Research Board 87th Annual Meeting*, 2008, no. 08-2964.

[8] S. Chiappone, O. Giuffrè, A. Granà, R. Mauro, and A. Sferlazza, "Traffic simulation models calibration using speed–density relationship: An automated procedure based on genetic algorithm," *Expert Systems with Applications,* vol. 44, pp. 147-155, 2016.

[9] K. Ozbay, "Calibration of a Macroscopic Traffic Simulation Model using Bayesian Simultaneous Perturbation Stochastic Approximation Methodology."

[10] J. Ma, H. Dong, and H. Zhang, "Calibration of microsimulation with heuristic optimization methods," *Transportation Research Record: Journal of the Transportation Research Board,* 2007.

[11] A. Paz, V. Molano, and C. Gaviria, "Calibration of corsim models considering all model parameters simultaneously," in *Intelligent Transportation Systems (ITSC), 2012 15th International IEEE Conference on*, 2012, pp. 1417-1422: IEEE.

[12] A. Paz, V. Molano, E. Martinez, C. Gaviria, and C. Arteaga, "Calibration of traffic flow models using a memetic algorithm," *Transportation Research Part C: Emerging Technologies,* vol. 55, pp. 432-443, 2015.

[13] R. Fernández, M. F. Lecaros, and A. Valencia, "Calibración del microsimulador de tráfico TSIS-CORSIM en Chile," *Ingeniería de Transporte,* vol. 17, no. 1, 2013.

[14] B. Park, J. Won, and I. Yun, "Application of microscopic simulation model calibration and validation procedure: Case study of coordinated actuated signal system," *Transportation Research Record: Journal of the Transportation Research Board,* no. 1978, pp. 113-122, 2006.

[15] R. Dowling, A. Skabardonis, and V. Alexiadis, "Traffic analysis toolbox volume III: guidelines for applying traffic microsimulation modeling software," 2004.

[16] "Competition on Large Scale Global Optimization, IEEE CEC 2015," 2015 Retrieved 28 January 2016, from http://goanna.cs.rmit.edu.au/~xiaodong/lsgo-competition- cec15.

[17] A. Alabert, A. Berti, R. Caballero, and M. Ferrante, "No-Free-Lunch theorems in the continuum," *Theoretical Computer Science,* vol. 600, pp. 98-106, 2015.

[18] A. LaTorre, S. Muelas, and J.-M. Peña, "Multiple offspring sampling in large scale global optimization," in *Evolutionary Computation (CEC), 2012 IEEE Congress on*, 2012, pp. 1-8: IEEE.

[19] Z. Yang, K. Tang, and X. Yao, "Large scale evolutionary optimization using cooperative coevolution," *Information Sciences,* vol. 178, no. 15, pp. 2985-2999, 2008.

[20] D. Molina, M. Lozano, and F. Herrera, "MA-SW-Chains: Memetic algorithm based on local search chains for large scale continuous global optimization," in *Evolutionary Computation (CEC), 2010 IEEE Congress on*, 2010, pp. 1-8: IEEE.

[21] Federal Highway Administration, “CORSIM User’s Guide,” no. December. 2006.

[22] C. Cobos *et al.*, "Calibration of Microscopic Traffic Flow Simulation Models Using a Memetic Algorithm with Solis and Wets Local Search Chaining (MA-SW-Chains)," in *Ibero-American Conference on Artificial Intelligence*, 2016, pp. 365-375: Springer.

[23] R. E. Anderson and C. Hicks, "Highlights of contemporary microsimulation," *Social Science Computer Review,* vol. 29, no. 1, pp. 3-8, 2011.

[24] L. E. Owen, Y. Zhang, L. Rao, and G. McHale, "Street and traffic simulation: traffic flow simulation using CORSIM," in *Proceedings of the 32nd conference on Winter simulation*, 2000, pp. 1143-1147: Society for Computer Simulation International.

[25] Mctrans, “Reference Manual,” 2010.

[26] A. H. Gandomi, X.-S. Yang, S. Talatahari, and A. H. Alavi, *Metaheuristic applications in structures and infrastructures*. Newnes, 2013.

[27] M. A. Potter and K. A. De Jong, "A cooperative coevolutionary approach to function optimization," in *International Conference on Parallel Problem Solving from Nature*, 1994, pp. 249-257: Springer.

[28] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of global optimization,* vol. 11, no. 4, pp. 341-359, 1997.

[29] Z. Yang, K. Tang, and X. Yao, "Self-adaptive differential evolution with neighborhood search," in *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, 2008, pp. 1110-1116: IEEE.

[30] F. J. Solis and R. J.-B. Wets, "Minimization by random search techniques," *Mathematics of operations research,* vol. 6, no. 1, pp. 19-30, 1981.

[31] L.-Y. Tseng and C. Chen, "Multiple trajectory search for large scale global optimization," in *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, 2008, pp. 3052-3059: IEEE.

[32] A. LaTorre de la Fuente, "A framework for hybrid dynamic evolutionary algorithms: multiple offspring sampling (MOS)," Informatica, 2009.

[33] D. Molina, M. Lozano, A. M. Sánchez, and F. Herrera, "Memetic algorithms based on local search chains for large scale continuous optimisation problems: MA-SSW-Chains," *Soft Computing,* vol. 15, no. 11, pp. 2201-2220, 2011.

[34] F. Altiparmak, M. Gen, L. Lin, and I. Karaoglan, "A steady-state genetic algorithm for multi-product supply chain network design," *Computers & Industrial Engineering,* vol. 56, no. 2, pp. 521-537, 2009.

[35] D. Molina, M. Lozano, C. García-Martínez, and F. Herrera, "Memetic algorithms for continuous optimisation based on local search chains," *Evolutionary computation,* vol. 18, no. 1, pp. 27-63, 2010.

[36] K.-O. Kim and L. Rilett, "Simplex-based calibration of traffic microsimulation models with intelligent transportation systems data," *Transportation Research Record: Journal of the Transportation Research Board,* no. 1855, pp. 80-89, 2003.

[37] B. Park and A. Kamarajugadda, "Development and evaluation of a stochastic traffic signal optimization method," *International journal of sustainable transportation,* vol. 1, no. 3, pp. 193-207, 2007.

[38] B. “Brian” Park, I. Yun, and K. Ahn, "Stochastic optimization for sustainable traffic signal control," *International journal of sustainable transportation,* vol. 3, no. 4, pp. 263-284, 2009.

[39] A. L. Cunha, J. E. Bessa Jr, and J. R. Setti, "Genetic algorithm for the calibration of vehicle performance models of microscopic traffic simulators," in *Progress in Artificial Intelligence*: Springer, 2009, pp. 3-14.

[40] S. Hoogendoorn and R. Hoogendoorn, "Calibration of microscopic traffic-flow models using multiple data sources," *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences,* vol. 368, no. 1928, pp. 4497-4517, 2010.

[41] P. Korcek, L. Sekanina, and O. Fucik, "Calibration of traffic simulation models using vehicle travel times," in *Cellular Automata*: Springer, 2012, pp. 807-816.

[42] O. B. Espinosa, "A Genetic Algorithm for the Calibration of a Micro-Simulation Model," *arXiv preprint arXiv:1201.3456,* 2012.

[43] L. Jie, H. Van Zuylen, Y. Chen, F. Viti, and I. Wilmink, "Calibration of a microscopic simulation model for emission calculation," *Transportation Research Part C: Emerging Technologies,* vol. 31, pp. 172-184, 2013.

[44] R. Omrani and L. Kattan, "Simultaneous calibration of microscopic traffic simulation model and estimation of origin/destination (od) flows based on genetic algorithms in a high-performance computer," in *Intelligent Transportation Systems-(ITSC), 2013 16th International IEEE Conference on*, 2013, pp. 2316-2321: IEEE.

[45] J. Yuan, S. H. Ng, and K. L. Tsui, "Calibration of stochastic computer models using stochastic approximation methods," *Automation Science and Engineering, IEEE Transactions on,* vol. 10, no. 1, pp. 171-186, 2013.

[46] D. K. Hale, C. Antoniou, M. Brackstone, D. Michalaka, A. T. Moreno, and K. Parikh, "Optimization-based assisted calibration of traffic simulation models," *Transportation Research Part C: Emerging Technologies,* vol. 55, pp. 100-115, 2015.

[47] R. Balakrishna, C. Antoniou, M. Ben-Akiva, H. Koutsopoulos, and Y. Wen, "Calibration of microscopic traffic simulation models: Methods and application," *Transportation Research Record: Journal of the Transportation Research Board,* 2007.

[48] C. Antoniou, C. L. Azevedo, L. Lu, F. Pereira, and M. Ben-Akiva, "W-SPSA in practice: Approximation of weight matrices and calibration of traffic simulation models," *Transportation Research Part C: Emerging Technologies,* vol. 59, pp. 129-146, 2015.

[49] W. Veerbeek, A. Pathirana, R. Ashley, and C. Zevenbergen, "Enhancing the calibration of an urban growth model using a memetic algorithm," *Computers, Environment and Urban Systems,* vol. 50, pp. 53-65, 2015.

[50] M. Maciejewski and J. Bischoff, "Large-scale microscopic simulation of taxi services," *Procedia Computer Science,* vol. 52, pp. 358-364, 2015.

[51] R. Zhong, K. Fu, A. Sumalee, D. Ngoduy, and W. Lam, "A cross-entropy method and probabilistic sensitivity analysis framework for calibrating microscopic traffic models," *Transportation Research Part C: Emerging Technologies,* vol. 63, pp. 147-169, 2016.

[52] C. Cobos *et al.*, "Multi-objective Memetic Algorithm Based on NSGA-II and Simulated Annealing for Calibrating CORSIM Micro-Simulation Models of Vehicular Traffic Flow," in *Conference of the Spanish Association for Artificial Intelligence*, 2016, pp. 468-476: Springer.

[53] V. Papathanasopoulou, I. Markou, and C. Antoniou, "Online calibration for microscopic traffic simulation and dynamic multi-step prediction of traffic speed," *Transportation Research Part C: Emerging Technologies,* vol. 68, pp. 144-159, 2016.

[54] J. Torres-Jimenez and E. Rodriguez-Tello, "New bounds for binary covering arrays using simulated annealing," *Information Sciences,* vol. 185, no. 1, pp. 137-152, 2012.

[55] A. Araujo and E. Giné, *The central limit theorem for real and Banach valued random variables*. Wiley New York, 1980.

[56] J. Alcalá-Fdez *et al.*, "KEEL: a software tool to assess evolutionary algorithms for data mining problems," *Soft Computing,* vol. 13, no. 3, pp. 307-318, 2009.

[57] S. Abraham, S. Sanyal, and M. Sanglikar, "Finding numerical solutions of diophantine equations using ant colony optimization," *Applied Mathematics and Computation,* vol. 219, no. 24, pp. 11376-11387, 2013.