ALGORITMO MULTI-OBJETIVO BASADO EN LA MEJOR BÚSQUEDA ARMÓNICA GLOBAL Y SIMULACIÓN DE EVENTOS DISCRETOS PARA LA DEFINICIÓN DE RUTAS Y HORARIOS EN UN SISTEMA DE TRANSPORTE MASIVO DE PASAJEROS

Ing. EDGAR FABIAN RUANO DAZA

Director: PhD. CARLOS ALBERTO COBOS LOZADA

Asesor: PhD. JOSÉ TORRES JIMÉNEZ (CINVESTAV, Tamaulipas, México)

UNIVERSIDAD DEL CAUCA

FACULTAD DE INGENIERÍA ELECTRÓNICA Y TELECOMUNICACIONES

MAESTRÍA EN COMPUTACIÓN

DEPARTAMENTO DE SISTEMAS

GRUPO DE I+D EN TECNOLOGÍAS DE LA INFORMACIÓN (GTI)

SISTEMAS INTELIGENTES – MODELADO Y SIMULACIÓN

Popayán, Julio de 2016

Tabla de Contenido

Lista de Figuras

Lista de Tablas

1 Introducción

1.1 Definición del problema

En el mundo, los sistemas de transporte masivo de pasajeros (STMP) también conocidos como Bus Rapid Transit Systems (BRTS) han demostrado ser una solución viable para solventar las crecientes necesidades de transporte de la población, permitiendo mejorar la calidad de vida de las personas y reduciendo además el impacto ambiental negativo que actualmente se tiene con el uso del automóvil (gas, gasolina, diesel). Estos sistemas incluyen la eficiencia de los sistemas ferroviarios urbanos con un costo similar al de los sistemas de buses tradicionales [1, 2]. En Colombia, como en muchos otros países de la región, hoy en día se cuenta con varios STMP entre los que se encuentran: el metro y bus de Medellín, los articulados de Bogotá (Transmilenio), Cali (Mío), Pereira (MegaBus) y Bucaramanga (Metrolínea).

Al implementar los STMP surgen varios problemas (generales en procesos de planeación de transito): el diseño de la red, el diseño de las rutas, la definición de las frecuencias por ruta, la asignación de buses y la asignación de personal que conduce los buses. El creciente uso de estos sistemas y las diversas y cambiantes condiciones en las que se transportan los pasajeros hacen que el proceso de afinación de rutas y frecuencias de servicio se convierta en pieza clave para la aceptación y satisfacción del servicio por parte de los usuarios. Por otro lado, en la puesta a punto de los STMP también es necesario pensar en la contraparte, en los proveedores del servicio, los cuales buscan la mayor rentabilidad y la sostenibilidad del sistema a corto, mediano y largo plazo [3-5].

Frecuentemente los procesos de diseño y programación de STMP involucran fases de observación, análisis de datos, generación de soluciones y prueba. Dichas fases conforman un ciclo que se repite hasta que se logra el funcionamiento del sistema con resultados aceptables dependiendo de los objetivos propuestos por los diseñadores. La Figura 1 presenta una visión global de este proceso.

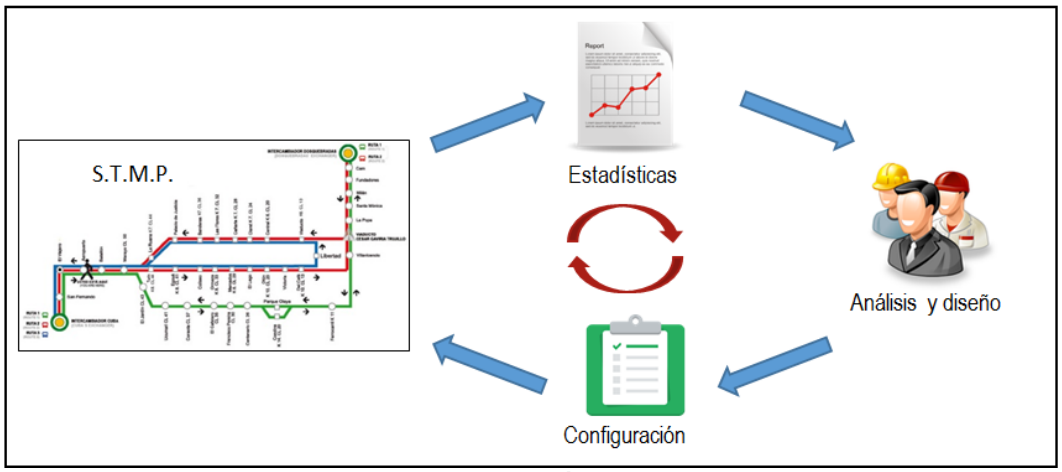

Figura 1 Proceso de configuración de STMP (fuente propia)

Entre los inconvenientes de este proceso se encuentra el costo por cada ciclo debido principalmente a que las pruebas se realizan sobre el sistema real en donde es necesario hacer que todos los involucrados en el funcionamiento del sistema (usuarios, operadores, etc.), se adapten a cada configuración implementada y donde además no se conoce la efectividad de la solución hasta que se analicen las estadísticas obtenidas a partir del funcionamiento del STMP.

Además de los costos que supone la afinación de las soluciones mediante el ensayo y error, se encuentra la dificultad del análisis y diseño en busca de una configuración para el STMP que permita alcanzar niveles de satisfacción de usuario, sostenibilidad y rentabilidad previamente definidos para el sistema, tanto así que en la literatura los problemas de "diseño de redes y rutas de transporte", "programación de frecuencias" y "programación de horarios" que se presentan en el diseño de STMP se agrupan dentro de un problema global denominado "Problema de Diseño y Programación de Redes de Transporte" o "Transit Network Desing and Scheduling Problem" (TNDSP) [4].

Dada la importancia de los sistemas de transporte en la vida cotidiana de las personas en las ciudades, estos problemas se han atacado con variados enfoques y desde hace ya mucho tiempo. Algunos de los enfoques más utilizados son: matemáticos (exactos), heurísticos (por ejemplo con algoritmos evolutivos) y con búsquedas por vecindad [4, 6]. Sin embargo, aunque se pueden encontrar muchos trabajos orientados a la solución de TNDSP, la mayoría de los trabajos previos reportados en el estado del arte, atacan sólo un sub-problema del conjunto definido en TNDSP, es decir, trabajan sobre el "diseño de redes de transporte", la "configuración de frecuencias en redes de transporte" o la "programación en redes de transporte", y son relativamente pocas las investigaciones enfocadas a resolver más de un sub-problema simultáneamente o la totalidad del TNDSP, esto sucede, entre otras cosas, porque los enfoques utilizados para la optimización sólo permiten manejar uno de los sub-problemas a la vez dado que cada uno de ellos tiene sus propias restricciones y objetivos, además cuando los sub-problemas se tratan por separado los objetivos a manejar no suelen ser conflictivos entre sí, lo que si sucede normalmente cuando se aborda más de un sub-problema con una misma estrategia de solución.

Por lo anterior, se necesitan estrategias de optimización multi-objetivo o "Multiobjective Optimization (MO)" [7]. La MO permite el manejo de múltiples objetivos a la vez, los cuales pueden ser o no conflictivos entre sí, con el fin de encontrar la mejor solución o un listado con las mejores soluciones, en este último caso, es el diseñador o quien esté a cargo, quien seleccione finalmente la solución que en su criterio sea la más conveniente [7-9]. En ese orden de ideas, este tipo de optimización aplicada al TNDSP podría permitir trabajar simultáneamente en dos o más sub-problemas a la vez. Dentro de la optimización MO se destacan las técnicas de optimización estocásticas (heurísticas) multi-objetivo. Dichas técnicas dependen de la retroalimentación obtenida de la evaluación de las soluciones propuestas durante la ejecución del algoritmo, lo que en el caso de los TNDSP, resulta en un gran reto debido a la cantidad de restricciones a tener en cuenta, además de los costos en tiempo y dinero que implica el valorar una solución particular. Para resolver este problema en el presente proyecto se propone el uso de herramientas de simulación de eventos discretos, que faciliten la generación de los datos necesarios para el cálculo de la aptitud o calidad de las soluciones encontradas, enfoque que ha reportado buenos resultados en el estado del arte [10].

Existen desde hace ya varias décadas, software y lenguajes de programación que proveen soporte para la simulación de diversos tipos de sistemas [11]. Se ha demostrado que el trabajo

sobre simulaciones y no sobre sistemas reales tiene ventajas tales como: reducción de costos, y esfuerzo necesarios, como también desventajas, entre ellas: la inexactitud debido a variables no consideradas al momento de crear los modelos (sobre simplificación del mundo real en el modelo) [12]. Como en muchas otras áreas, en el transporte es posible hacer uso de simulaciones para realizar cálculos y recopilar información. Usualmente los modelos de simulación se construyen y refinan tomando como base un caso de estudio real, esto permite asegurar que los datos obtenidos al ejecutar la simulación sean similares o muy parecidos a los que se conseguirían al hacer la observación sobre el sistema real [13]. La simulación entonces se convierte en esta propuesta en una parte importante de la solución al desafío que supone el cálculo de los valores de aptitud (fitness) en la optimización multi-objetivo. Arena [13-15] es una herramienta software para modelar y simular el funcionamiento de sistemas de eventos discretos, el funcionamiento de Arena se basa en SIMAN [16], un lenguaje de simulación de propósito general. Se seleccionó esta herramienta de simulación teniendo en cuenta varias razones: versatilidad para modelar diferentes dominios, baja curva de aprendizaje, alto nivel de interoperabilidad con otras aplicaciones usando el compilador SIMAN y velocidad de simulación.

Otro de los desafíos que presenta la MO es la selección del algoritmo base. Se sabe que los diferentes algoritmos existentes tienen fortalezas y debilidades que dependen no solo del problema que se está abordando (No free lunch theorem), sino de la configuración y afinación misma del algoritmo. Algunos algoritmos son tan complejos de configurar que se requiere también de un proceso complejo de afinación adicional solamente para definir cuáles de los valores de configuración son los mejores a usar para ejecutar el algoritmo en un problema determinado [8]. Teniendo en cuenta que el algoritmo de la búsqueda armónica (Harmony Search, HS) [17] y sus mejoras, Improved Harmony Search (IHS) [18] y Global-Best Harmony Search (GBHS) [19], han reportado excelentes resultados en la solución de problemas complejos en diferentes áreas de aplicación (entre ellas, el problema de generación de rutas vehiculares [20]), que el proceso de afinación de parámetros es más sencillo que en la mayoría de los algoritmos heurísticos, que tiene una rápida velocidad de convergencia evitando caer atrapado en óptimos locales, que GBHS supera la eficiencia de sus predecesores, que GBHS trabaja eficientemente tanto en problemas continuos como discretos [17, 18, 21], y que el Grupo de I+D en Tecnologías de la Información de la Universidad del Cauca (GTI) ha desarrollado a la fecha varias investigaciones exitosas con el mismo, en esta investigación se tomó la decisión de adoptar a GBHS como el algoritmo base del algoritmo multi-objetivo a proponer.

Por lo anterior, en esta investigación se buscó dar respuesta a la siguiente pregunta de investigación: ¿Es posible mejorar los procesos de configuración simultánea de rutas y horarios en un Sistema de Transporte Masivo de Pasajeros a través de un algoritmo multi-objetivo basado en la mejor búsqueda armónica global y una herramienta de simulación de eventos discretos (Arena) para soportar la evaluación de la función de aptitud?

1.2 Justificación

La importancia de la presente tesis de maestría se centró en la generación de un nuevo conocimiento útil para la comunidad científica y académica internacional dedicada a la resolución de los problemas asociados a la configuración óptima de sistemas de transporte masivo de pasajeros con el fin último de mejorar la calidad de vida de los usuarios mediante la

mejora del nivel de satisfacción de los mismos respecto a dichos sistemas proponiendo un nuevo algoritmo multi-objetivo basado en la mejor búsqueda armónica global que facilita la búsqueda de las mejores configuración de rutas y frecuencias para un STMP.

Desde una perspectiva práctica, este proyecto fue conveniente porque generó sus diferentes planteamientos teniendo en cuenta la información y restricciones de un STMP existente lo que incrementa la probabilidad de ser aplicado a un contexto real, aporta un conjunto de lineamientos para crear modelos de simulación que permiten medir el rendimiento de diferentes configuraciones de rutas, frecuencias y horarios en un STMP con el fin de buscar configuraciones óptimas que podrían ser reutilizados con diferentes técnicas de optimización. Dado que durante la ejecución del proyecto se consiguió trabajar en forma conjunta entre la Universidad del Cauca y Megabús S.A. de la ciudad de Pereira para obtener información precisa sobre el funcionamiento de dicho STMP, se logró generar un modelo de simulación de eventos discretos sobre Arena, ajustado (aunque no totalmente calibrado), con información real que permite la búsqueda de mejores configuraciones para dicho sistema.

Durante el desarrollo del proyecto se pusieron en práctica los conocimientos adquiridos en las asignaturas de la maestría en computación, así como también se debieron apropiar nuevos conocimientos y desarrollar nuevas habilidades de investigación en el área específica de optimización multi-objetivo.

Como producto principal de este proyecto se definió un algoritmo multi-objetivo basado en la mejor búsqueda armónica global que, soportado sobre modelos de simulación de eventos discretos, permite optimizar la configuración de rutas y frecuencias para un STMP teniendo como objetivos la minimización de costos de operación y la maximización de la satisfacción del usuario.

1.3 Objetivos

A continuación se presentan los objetivos definidos en el anteproyecto y que fueran aprobados por el Consejo de Facultad de la Facultad de Ingeniería Electrónica y Telecomunicaciones de la Universidad del Cauca.

1.3.1 Objetivo general

Proponer un algoritmo multi-objetivo basado en la mejor búsqueda armónica global y simulación de eventos discretos que permita configurar el diseño de rutas y horarios en un sistema de transporte masivo de pasajeros.

1.3.2 Objetivos específicos

- Modelar un sistema de transporte masivo de pasajeros (STMP), con sus correspondientes restricciones en una herramienta de simulación de eventos discreto (Arena), instanciar un caso específico de un sistema Colombiano y establecer los mecanismos de interoperabilidad con el algoritmo multi-objetivo a proponer.
- Diseñar e implementar un algoritmo multi-objetivo basado en la mejor búsqueda armónica global que permita generar un conjunto de soluciones optimas al problema de configuración de rutas y horarios para un sistema de transporte masivo de pasajeros (STMP),

minimizando el tiempo medio de uso y los costos de operación del sistema, teniendo en cuenta variables de: capacidad de buses, flota disponible, tiempo medio de recorrido entre estaciones y destinos de los pasajeros, y soportado por una herramienta de simulación de eventos discretos (Arena) para el cálculo de la función objetivo.

- Evaluar el comportamiento del algoritmo propuesto, mediante la comparación del rendimiento de sus soluciones óptimas encontradas contra el rendimiento de la actual configuración de rutas y horarios del sistema de transporte masivo de pasajeros (STMP) tomado como caso de estudio, midiendo los tiempos medios de uso y costos de operación del sistema, todo soportado en una simulación del sistema real.

1.4 Resultados obtenidos

Al culminar el presente trabajo de investigación se obtuvieron los siguientes productos:

- Modelo de simulación del sistema de transporte masivo de la ciudad de Pereira, Colombia llamado Megabús.
- Algoritmo multi-objetivo basado en la mejor búsqueda armónica global para la generación de configuraciones de rutas e intervalos de salida de buses (MOGBHS-TNDFSP)
- Prototipo software con implementación de MOGBHS para el problema específico de TNDFSP.
- Lineamientos para crear modelos de simulación de sistemas de transporte masivo de pasajeros sobre Arena para la generación de configuraciones de rutas e intervalos mediante MOGBHS-TNDFSP u algoritmos similares.
- Artículo 1: Ruano, E., Cobos, C., Torres, J.: Transit Network Frequencies-Setting Problem Solved Using a new Multi-Objective Global-Best Harmony Search Algorithm and Discrete Event Simulation. Aceptado para su publicación en “15th Mexican International Conference on Artificial Intelligence - MICAI 2016”. Ver anexo 8.
- Artículo 2: Ruano, E., Cobos, C., Torres, J., Mendoza, M: Definición de rutas y frecuencias óptimas para un sistema de transporte masivo de pasajeros usando búsqueda armónica global. Ver anexo 9.

2 Marco teórico

Las áreas relacionadas con esta investigación, las cuales han facilitado la base teórica y procedimental suficiente para su desarrollo, se describen brevemente a continuación, con el objetivo de dejar en claro conceptos necesarios para una buena lectura del presente documento.

2.1 Optimización estocástica

Se denomina optimización estocástica al proceso de minimización o maximización de una función objetivo compleja mediante la intervención de la aleatoriedad en dicho proceso. A diferencia de la optimización determinística, en la optimización estocástica no se tiene conocimiento previo de hacia dónde va a tender el proceso sobre el espacio de búsqueda. Estos algoritmos son usados cuando no se tienen indicios de una solución óptima, cuando no se tiene una ruta definida para encontrar la solución óptima, o cuando la búsqueda por fuerza bruta no es viable porque el espacio de búsqueda es demasiado grande [8, 22]. En los métodos de optimización estocástica se encuentran los basados en gradiente, los métodos de estado simple (subiendo la colina, re-cocido simulado, búsqueda tabú, etc.), los métodos basados en población (algoritmos genéticos, enjambres de partículas, etc.), entre otros.

2.1.1 Optimización multi-objetivo

En la optimización estocástica se han propuesto algoritmos que se enfocan en la optimización de más de una función objetivo a la vez, denominados algoritmos multi-objetivo. Estos métodos de optimización permiten un mejor acercamiento a los problemas reales en donde usualmente se tiene más de una función a optimizar. Es de resaltar que en este modelo de optimización el resultado no es una solución única sino un conjunto de las mejores soluciones encontradas [9]. Dentro de las estrategias multi-objetivo se encuentran tres enfoques principales, a saber: métodos ingenuos, métodos basados en ordenamiento de no-dominados y métodos basados en la fuerza de Pareto.

La principal diferencia entre los tres enfoques radica en el método usado para calcular el fitness (aptitud) de los individuos y su ingreso o no al conjunto de mejores soluciones. Dentro de los "métodos ingenuos" (menos efectivos) se pueden destacar técnicas que [8]:

- Crean una única función de fitness a partir de la ponderación de los objetivos basados en un peso asignado en el diseño del algoritmo.
- Realizan selección basado en competencia lexicográfica, es decir, que ordenan los objetivos de mayor a menor nivel de importancia en el dominio sobre el que se busca la solución para luego evaluar y filtrar las soluciones por cada uno de los objetivos previamente ordenados.
- Escogen las mejores soluciones tomando en consideración un único objetivo seleccionado aleatoriamente y que usualmente es diferente en cada ciclo o generación del algoritmo
- Seleccionan como las mejores soluciones aquellas que ganan en mayor número de objetivos.

- Selección por múltiple torneo, es decir, seleccionan la solución o el conjunto de ellas que hayan sido mejores en todos los objetivos del problema.

Para comprender los enfoques de ordenamiento de no-dominados y fuerza de Pareto es necesario comprender primero los conceptos de Dominio de Pareto, Soluciones No-dominadas y Frente de Pareto.

Dominio de Pareto: en un problema de maximización, siendo ***f*** y ***g*** soluciones con ***m*** dimensiones se puede decir que ***f domina a g*** sólo si para todo ***i*** que pertenece a (1,2,…, m) se cumple que: $f_i >= g_i$ y además existe al menos un ***j*** que pertenece a (1,2,…, m) tal que $f_j > g_j$. En la Figura 2 se muestra el concepto de Dominio de Pareto en un problema de maximización con dos (2) objetivos [8].

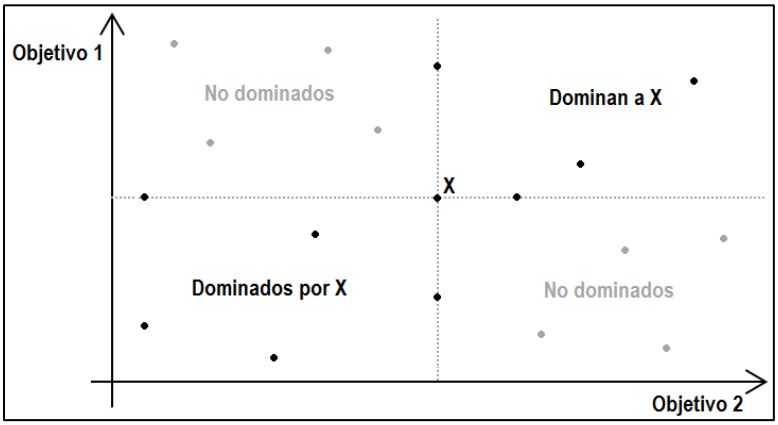

Figura 2 Descripción de Dominio de Pareto en Problema de Maximización (fuente propia)

Soluciones No-dominadas: como el nombre sugiere, una solución se considera no-dominada cuando en la población no existe una solución que la domine.

Frente de Pareto: conjunto de todas las soluciones no-dominadas de una población, la Figura 3 presenta un frente de Pareto para un problema donde se maximizan dos objetivos.

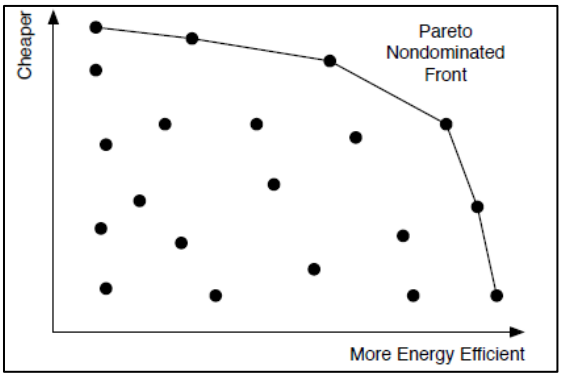

Figura 3 Descripción de Frente de Pareto (tomado de[8])

La estrategia de ordenamiento de no-dominados consiste básicamente en tomar el conjunto de soluciones que hacen parte del frente de Pareto, dichas soluciones se consideran las mejores y por tanto son el resultado del proceso de optimización. Para lograr esto, existen diferente aproximaciones, por ejemplo: el manejo de la dispersión entre los individuos que conforman el

frente. Uno de los mejores algoritmos basados en esta estrategia es el "Non-dominated Sorting Genetic Algorithm II" (NSGA II) presentado en [23]. En dicho trabajo se propone medir la distancia de dispersión entre los individuos del frente de Pareto, considerando mejores los elementos con mayor distancia respecto a las soluciones cercanas, en la Figura 4 se ilustra un mecanismo para medir la distancia de dispersión en un problema de minimización utilizando cuboides.

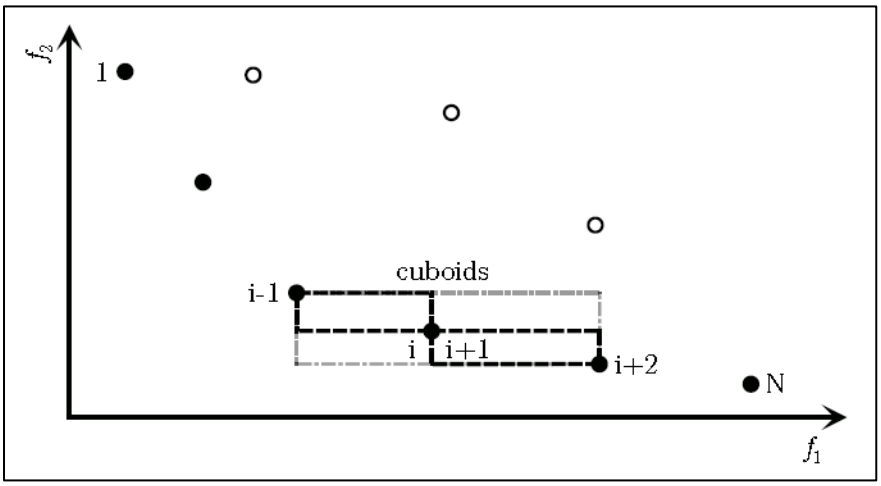

Figura 4 Descripción cálculo de distancia de dispersión (tomado de [24])

La distancia de dispersión para un elemento *i* del frente de Pareto se puede expresar de la forma:

$$d(i) = \sum_{k=0}^{M} |f_k(i) - f_k(i-1)| + |f_k(i) - f_k(i+1)|$$

Donde M representa el número de objetivos del problema tratado y $f_k(x)$ representa la evaluación de la solución x respecto al objetivo k.

Los métodos basados en fuerza de Pareto buscan los individuos que dominen la mayor cantidad de elementos de la población. Sin embargo la selección por fuerza de Pareto no permite obtener todos los individuos del frente de Pareto, para corregir esto se utiliza el concepto de debilidad de Pareto que corresponde al número de individuos que dominan a un elemento de la solución y se buscan los individuos con menor debilidad. Un enfoque más elaborado sugiere minimizar el *Wimpiness*, el cual se define como la suma de la fuerza de los individuos que dominan al elemento especifico. Uno de los mejores exponentes de esta estrategia es Strength Pareto Evolutionary Algorithm II (SPEA II) presentado en [25].

2.2 Diseño y programación de redes de transporte

Los problemas que supone el diseño y configuración de los diferentes componentes de las redes de transporte se engloban dentro de un área denominada "Transit Network Design and Scheduling Problems" (TNDSP). La misma incluye los sub-problemas de: Diseño de Redes (rutas) de Transporte, Configuración de frecuencias, Programación de Buses y Conductores.

Estos a su vez están agrupados en dos conjuntos: “Diseño de redes y programación de frecuencias” y “Programación de frecuencias y horarios”. De los sub-problemas mencionados, los dos primeros se consideran de mayor relevancia debido a su complejidad, impacto sobre el rendimiento de los sistemas de transporte y prevalencia en orden de atención cuando de diseñar y configurar un sistema de transporte se trata [4, 6]. En la Figura 5 se presenta un esquema general de TNDSP.

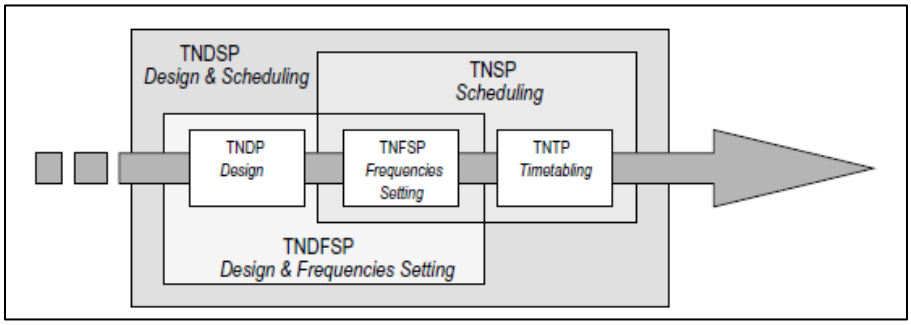

Figura 5 Transit Network Design and Scheduling Problem Structure (tomado de [4])

En los siguientes ítems se describen los sub-problemas previamente mencionados:

2.2.1 Diseño de redes

El problema de diseño de redes de tránsito (TNDP por sus siglas en inglés), consiste en determinar un conjunto de rutas cada una formada por dos terminales y una secuencia de paradas intermedias teniendo una distribución de demanda, una topología del sistema y un conjunto de objetivos y restricciones.

2.2.2 Programación de frecuencias

El problema de programación de frecuencias en redes de tránsito (TNFSP por sus siglas en inglés), consiste en determinar las frecuencias adecuadas para cada una de las rutas de un sistema de transporte para un periodo de tiempo, comúnmente los periodos se determinan por horas del día, día de la semana o temporada del año. El objetivo es establecer frecuencias para prestar un servicio regular que satisfaga a los usuarios con la menor cantidad de buses en la flota, y por tanto con el menor costo de operación.

2.2.3 Programación de horarios

El problema de programación de horarios en redes de tránsito (TNTP por sus siglas en inglés), consiste en determinar los horarios de inicio de operación para cada ruta de una red de transporte existente, con frecuencias pre-establecidas para cada ruta. El objetivo es mejorar las condiciones de transferencia entre rutas para los usuarios con el menor número de buses en la flota.

2.2.4 Programación de frecuencias y horarios

Teniendo una red de rutas y una demanda de servicio determinada, el problema de programación de frecuencias y horarios (TNSP por sus siglas en inglés), consiste en determinar

una configuración que incluya frecuencias y horarios de inicio de operación para cada ruta del sistema teniendo un conjunto de objetivos y restricciones.

2.2.5 Diseño y programación de frecuencias

El problema de diseño y programación de frecuencias en redes de tránsito (TNDFSP por sus siglas en inglés), consiste en determinar un conjunto de rutas asociadas con unas frecuencias en un área particular, teniendo una demanda de servicio determinada y un conjunto de objetivos y restricciones usualmente relacionadas a rutas directas, cobertura de servicio y costos de operación. En [26] se indica que el diseño de redes de transporte y la configuración de frecuencias no necesariamente deben calcularse simultáneamente argumentando que la red de rutas es un componente menos variable en la red de tránsito, por tanto esta construcción no necesita estar influenciada por parámetros flexibles como las frecuencias. Sin embargo, el diseño de rutas puede ser más eficiente cuando se involucra la programación de frecuencias simultáneamente.

2.3 Covering Arrays

Los Covering Arrays (CAs), son estructuras combinatoriales definidas como una matriz de *N* filas y *K* columnas sobre un alfabeto de *v* símbolos donde para cada conjunto de *t* columnas (denominada fuerza del arreglo), cada combinación de t-tuplas es cubierta al menos una vez y se denotan como CA(N; K, v, t) [27, 28]. Por ejemplo, para los parámetros K = 4, v = 2 y t = 2, el número mínimo de filas es N = 5 y el CA es el siguiente:

Filas\Columnas	1	2	3	4
1	0	0	0	0
2	0	1	1	1
3	1	0	1	1
4	1	1	0	1
5	1	1	1	0

Figura 6 Matriz de CA (5; 2, 4, 2) (fuente propia)

Dado el alfabeto de tamaño 2 (valores posibles de 0 o 1), y la fuerza 2 (interacción de las columnas), las posibles combinaciones serían “00”, “01”, “10” y “11”. En el CA se encuentran todas las posibles combinaciones para cada grupo de t columnas (los posibles grupos de columnas son 1-2, 1-3, 1-4, 2-3, 2-4 y 3-4). En la siguiente figura se presenta el CA agrupado por columnas para ver con claridad todas las combinaciones existentes.

Filas\Columnas	1	2	1	3	1	4	2	3	2	4	3	4
1	0	0	0	0	0	0	0	0	0	0	0	0
2	0	1	0	1	0	1	1	1	1	1	1	1
3	1	0	1	1	1	1	0	1	0	1	1	1
4	1	1	1	0	1	1	1	0	1	1	0	1
5	1	1	1	1	1	0	1	1	1	0	1	0

Figura 7 Pares de columnas del CA (5; 2, 4, 2) (fuente propia)

Recientemente los CAs han sido usados exitosamente para automatizar la generación de casos de pruebas software debido a que permiten realizar una gran cobertura de configuraciones con un mínimo número de pruebas, lo que disminuye el costo de aplicación las mismas.

La generación de CAs óptimos (con un mínimo número de filas), es un problema combinatorial complejo que ha sido estudiado previamente y atacado mediante métodos exactos, métodos Greddy, meta-heurísticas, métodos algebraicos, entre otros [28]. Resultado de estos trabajos de investigación se encuentran disponibles en el estado del arte un buen conjunto de CAs óptimos encontrados.

3 Estado del arte

El presente proyecto se enfocó en el problema de "Diseño y programación de frecuencias para una red de transporte" (Transport Network Design and Frequencies Settings Problem, TNDFSP) en el intento por facilitar la optimización de la configuración de los sistemas de transporte masivo de pasajeros. A continuación se hace un reporte detallado de las investigaciones previas relacionadas con estos dos problemas.

3.1 Optimización en Problemas de Diseño y Programación de Redes de Transporte

Las técnicas basadas en optimización estocástica (meta-heurísticas) son un ejemplo de las técnicas usadas para resolver el problema de diseño y programación de redes de transporte. Estas técnicas de optimización comprenden un grupo de algoritmos que emplean algún grado de aleatoriedad en la búsqueda de la solución óptima para problemas considerados difíciles (NP-hard) entre los que se encuentran los algoritmos evolutivos como por ejemplo los algoritmos genéticos y otros algoritmos bio-inspirados como las colonias de abejas [8]. A continuación se hace una breve descripción de las investigaciones más recientes relacionadas con el problema en cuestión y al final se presentan un resumen de las principales deficiencias de estos trabajos.

En 2007 [29], se utilizó una técnica hibrida que combina algoritmos genéticos con simulación para resolver el problema de programación de una red de transporte, el método fue probado con un caso de estudio logrando satisfacer tanto la demanda como la oferta de transporte, sin embargo, aunque las pruebas resultaron exitosas no se consideraron variables como por ejemplo los costos de operación y tiempos de viaje de los usuarios del sistema, además se deja parte del TNDSP sin resolver (entre otros, la programación de frecuencias de rutas). De otro lado este es un ejemplo claro de que el soporte de la simulación para la optimización en problemas relacionados con redes de transporte es una opción factible y recomendada.

En 2008 [30], haciendo uso de algoritmos genéticos y teoría de colas y de grafos, se busca la optimización de los tiempos utilizados por los usuarios para llegar a sus destinos mediante un sistema de transporte masivo particular. Debido a la simplificación realizada al modelar la red de transporte los resultados de su uso todavía dejan espacios de investigación que permiten encontrar mejores soluciones con nuevas alternativas para los procesos de optimización. Por ejemplo, falto incluir el manejo de restricciones de capacidad de buses del sistema y restricciones de capacidad de recepción simultánea de buses por estación. Un segundo ejemplo sería: teniendo en cuenta que se modela sólo una sección del sistema (no el sistema completo), cuando los buses ingresan a la sección pueden ya traer pasajeros consigo, luego eso restringiría la disponibilidad de cupos y por tanto cambiaría la dinámica de movimiento de pasajeros.

En 2012 [12] se presenta un modelo de "densidad de dirección de viajeros" y una extensión del mismo para diseñar redes de transporte considerando la densidad de demanda relativa a demandas directas, transferencias y longitud de rutas, todo esto para maximizar la densidad de demanda de las rutas. La estrategia de solución se basó en optimización por colonias de hormigas y los datos para el caso de estudio fueron tomados de la ciudad de Dalian, China. Este proyecto sirve como referente para el uso de modelos de simulación en los procesos de

optimización de diseños de rutas de transporte, sin embargo deja de lado el problema de programación de las mismas.

De igual forma en 2012 [10], se realizó una comparación entre el rendimiento de los algoritmos genéticos versus la optimización por colonias de hormigas al resolver un problema de diseño de programación en redes de transporte. Ambos algoritmos fueron afinados para obtener un calendario óptimo soportados en una micro-simulación del sistema de rutas de buses de Melbourne, Australia. En dicha comparación el algoritmo basado en optimización por colonias de hormigas demostró una mayor eficiencia al evaluar menos soluciones para llegar a una buena solución.

También en 2012 [20], se modificó el algoritmo de búsqueda armónica para la búsqueda de soluciones al problema de enrutamiento de vehículos de carga con restricciones de capacidad en aras de minimizar la distancia total recorrida y el número de vehículos usados. Este trabajo, aunque difiere del actual proyecto al considerar vehículos de carga y no de transporte de pasajeros, demuestra que la búsqueda armónica o algoritmos basados en ella pueden ser usados con resultados satisfactorios para resolver problemas de generación de rutas de transporte.

En 2013 [31] se presenta un modelo basado en colonias de abejas para optimizar el diseño de redes de transporte intentando maximizar el número de pasajeros satisfechos mediante la minimización del número de transferencias entre buses y del tiempo total de viaje de los usuarios en el sistema. El trabajo demostró dar resultados de calidad comparados con otros enfoques en la literatura, sin embargo, no consideró los costos de implementación de la solución dado que en un TNDSP usualmente cuando se minimizan tiempos de viaje de los usuarios es porque se incrementa el número de buses disponibles, lo que supone un incremento en costos de funcionamiento de los sistemas.

En 2014 [32] se propone la utilización de un algoritmo genético con elitismo para el diseño de redes de transporte, el cual busca minimizar el tiempo total de viaje de los pasajeros. La generación de soluciones se realiza utilizando la lógica de un algoritmo genético y la elección de los mejores individuos se realiza mediante “selección por torneo”. En el mismo año [33] propone un algoritmo para dar solución al problema de configuración de rutas y frecuencias para un STMP mediante la división de dicho problema en dos niveles con estrategias diferentes para cada nivel: el primer nivel, de configuración de rutas, se aborda mediante un algoritmo de “enjambres de abejas”, y el segundo nivel, de configuración de frecuencias, se aborda mediante búsqueda por descendencia directa. Esta última propuesta busca la minimización del número de transferencias de los pasajeros durante su recorrido en el sistema.

En 2015 [34] se propone un algoritmo genético con alternancia de objetivos para resolver el problema de diseño de redes de transporte y configuración de frecuencias. El algoritmo busca minimizar costos para los pasajeros y minimizar costos de operación alternando cíclicamente entre los dos objetivos durante su ejecución. Ese mismo año en [35] se propone una solución a TNDSP mediante un algoritmo de dos fases que combina la búsqueda por vecindad con el método de gradiente para determinar una configuración de rutas y frecuencias para un STMP minimizando el costo total de funcionamiento cuya evaluación se basa en un modelo matemático.

Los trabajos descritos anteriormente tienen en común que tratan de optimizar solamente un objetivo a la vez, o en otros casos, un conjunto de objetivos de un mismo sub-problema. Debido a las simplificaciones que se deben realizar para resolver el sub-problema específico, las soluciones resultantes son en su mayoría inutilizables en entornos reales debido a que en dichos entornos siempre hay más de un objetivo y más de un sub-problema a resolver al mismo tiempo.

3.2 Optimización multi-objetivo en problemas de diseño y programación de redes de transporte

En 2007 [36], se propone resolver el problema de TNDP haciendo uso de una adaptación de GRASP (Greddy Randomized Adaptive Search Procedure), una meta-heurística diseñada para resolver problemas de optimización combinatoria. El algoritmo propuesto fue denominado GRASP-TNDP, basa su funcionamiento en GRASP con la diferencia de que en lugar de generar una única solución, busca obtener un frente de Pareto aproximado. Las pruebas del trabajo se realizaron con información del sistema de transporte de la ciudad de Rivera, Uruguay, obteniendo buenos resultados. Sin embargo, cabe anotar que la información utilizada para la implementación y pruebas del algoritmo incluyó simplificaciones al problema que pueden llegar a hacer inutilizable las soluciones en un ambiente real, por ejemplo: el no considerar la capacidad de los automotores. Para solventar dicha debilidad, la presente propuesta hizo uso de modelos de simulación discretos que permitan incluir más variables a los ambientes de prueba y que dejen abierta la posibilidad de mejoras futuras a los mismos.

En 2009 [37], se planteó un conjunto de heurísticas que incluyen rutinas para generación de rutas y un algoritmo genético (GA) para buscar un conjunto óptimo de rutas con sus correspondientes frecuencias para la implementación en un sistema de transporte público con vehículos ZEV (Zero Emission Vehicles) buscando minimizar costos de operación, costos a usuarios y costos externos, teniendo en cuenta la elasticidad de la demanda y la cantidad de vehículos existentes. Este acercamiento, al generar las posibles rutas en un proceso desligado del algoritmo genético (el que al final hace la optimización y cálculo de frecuencias), está desaprovechando la información de fitness generada durante el proceso de corrida del GA que puede servir como retroalimentación para buscar mejores rutas. En este mismo año, en [3] se planteó una nueva meta-heurística multi-objetivo basada en GRASP para encontrar solución a TNDSP validando los resultados con grafos que hacían las veces de modelos a escala de ciudades, este trabajo mostró que el algoritmo propuesto produce más soluciones no-dominadas que el método de ponderación de pesos con el mismo esfuerzo computacional y comprobó que la optimización de rutas y frecuencias es un enfoque multi-objetivo viable. Además del algoritmo, la presente propuesta se diferencia de dicho trabajo porque hace uso de modelos de simulación basados en sistemas de transporte existentes para obtener la información necesaria para el cálculo de la aptitud de las soluciones, lo que permite considerar más variables y por tanto acercar el problema a un entorno real.

En 2012 [38], se formuló el diseño de redes de transporte de buses urbanos como un problema de optimización de recursos y costos abordándolo mediante “inteligencia de enjambres”, el objetivo final es el diseño de rutas incluyendo algunas restricciones de capacidad de buses y factibilidad de rutas, la estrategia puede catalogarse dentro de los métodos ingenuos en cuanto a optimización multi-objetivo se refiere, sin embargo, aunque este enfoque es multi-objetivo no cubre el problema del cálculo de frecuencias para las rutas diseñadas.

Calcular el fitness de cada una de las soluciones generadas respecto a los N objetivos evaluados supone un problema en el proceso de diseño de los algoritmos, para ello algunos autores han optado por usar herramientas de simulación. La estrategia consiste en aplicar las soluciones generadas por los algoritmos a un micro-modelo o un modelo a escala del sistema real en donde se implementaría la solución óptima y retroalimentar el algoritmo con los resultados de esas simulaciones. Un ejemplo de ello es [39] (2013), en donde se optimiza la estructura de tarifas y frecuencia del servicio para obtener un equilibrio entre oferta y demanda en un sistema de transporte a partir del uso de un algoritmo genético y un algoritmo de recocido simulado soportados sobre modelos de simulación matemáticos, al final las mejores soluciones son aplicadas en un sistema real en Guangzhou (China), para probar el modelo y los algoritmos sugeridos.

En 2014 [40] se propone una formulación bi-objetivo para la programación de horarios y vehículos que permita analizar la relación entre nivel de servicio y los costos de operación de redes de transporte usando un método $\in$-constraint. En el mismo año en [41] se propone un algoritmo memético para la configuración de rutas y frecuencias con el objetivo de minimizar el costo para los pasajeros y reducir la demanda insatisfecha, respecto a la investigación aquí descrita este enfoque busca sólo la satisfacción de una de las partes involucradas sin tener en cuenta los costos de operación y el algoritmo heurístico utilizado es diferente.

Recientemente, en 2015 [42] se aplica la optimización multi-objetivo para la programación de frecuencias en STMP, se propone una modificación al algoritmo de búsqueda tabú que en su versión original es mono-objetivo [43, 44], para atacar dos objetivos simultáneamente: minimizar el tiempo total de viaje de los pasajeros y minimizar el tamaño de flota utilizada, además del dominio del problema atacado el presente proyecto de investigación difiere en el algoritmo utilizado para la solución.

En el estado del arte se reportan investigaciones que hacen uso de la optimización multi-objetivo para resolver uno o más sub-problemas de TNDSP gran parte de ellas reportan buenos resultados, sin embargo pueden encontrarse debilidades en algunas de ellas como por ejemplo: se enfocan solamente en la configuración de frecuencias, los modelos para evaluar el fitness de las soluciones son complejos de construir, adaptar o extender, los objetivos a optimizar solo buscan la satisfacción del usuario. La presente investigación propone resolver dichas debilidades mediante un algoritmo multi-objetivo basado en GBHS para generar soluciones de configuración de rutas y frecuencias para un STMP, buscando la optimización de tiempo de uso y costos de operación, y utilizando modelos de simulación de eventos discretos para la evaluación de fitness de las soluciones en procura de facilitar su implementación en diferentes STMP, dicha combinación de elementos no se ha reportado a la fecha en el estado del arte.

3.3 Simulación de eventos discretos

Con el ánimo de proveer un mecanismo que permita al algoritmo que se propone en este proyecto de investigación, entender el problema, generar soluciones, medir la efectividad de las mismas, que dichos datos pudieran ser utilizados por el algoritmo para su evolución, teniendo en cuenta que un algoritmo heurístico requiere hacer dicha evaluación decenas, cientos o hasta miles de veces, y considerando las bondades de los modelos de simulación y sus casos exitosos, se decidió soportar el algoritmo propuesto en modelos de simulación de eventos

discretos. La selección de este tipo de simulación busca hacer que la solución propuesta requiera el menor esfuerzo posible para ser aplicada a un STMP y que solo se necesiten conocimientos relativos al funcionamiento del STMP para realizar el modelado y su calibración.

3.3.1 ARENA

Se realizó una búsqueda de lenguajes y herramientas de simulación de eventos discretos calificando tres características: 1) Aplicabilidad al área de estudio, 2) Integración con otros lenguajes y 3) Curva de aprendizaje.

La herramienta seleccionada fue Arena de “Rockwell Automation, Inc.”, esta permite construir modelos de simulación de eventos discretos basados en módulos, variables, entidades y atributos [14, 15, 45]. Entre sus cualidades están:

- Capacidad de incorporar nuevos módulos según la necesidad del usuario, lo que la hace muy extensible y útil para dominios variados, entre ellos el modelado de STMP.
- Cuenta con un entorno gráfico sencillo que facilita su aprendizaje.
- Permite al usuario seguir y depurar el comportamiento de los modelos de simulación de forma interactiva.
- Soporta funciones matemáticas y estadísticas comúnmente usadas en modelos de simulación.
- Permite el uso de expresiones lógicas y matemáticas para el cálculo de valores y toma de decisiones durante el proceso de simulación, entre otras características.
- Cuenta con un módulo que facilita la visualización del reporte de simulación

La principal desventaja de Arena es su alto costo de licencia que hace que su uso en campos académicos se vea reducido.

En la Figura 8 se presenta un captura de pantalla de la interfaz gráfica de Arena, en ella se puede apreciar el editor de expresiones, una interfaz de configuración de módulo, el lienzo donde se ubican y entrelazan los módulos que componen cada modelo de simulación, las barras de herramientas, la barra de depuración y los controles de ejecución de simulación, entre otros elementos que provee arena para la creación y refinación de los modelos.

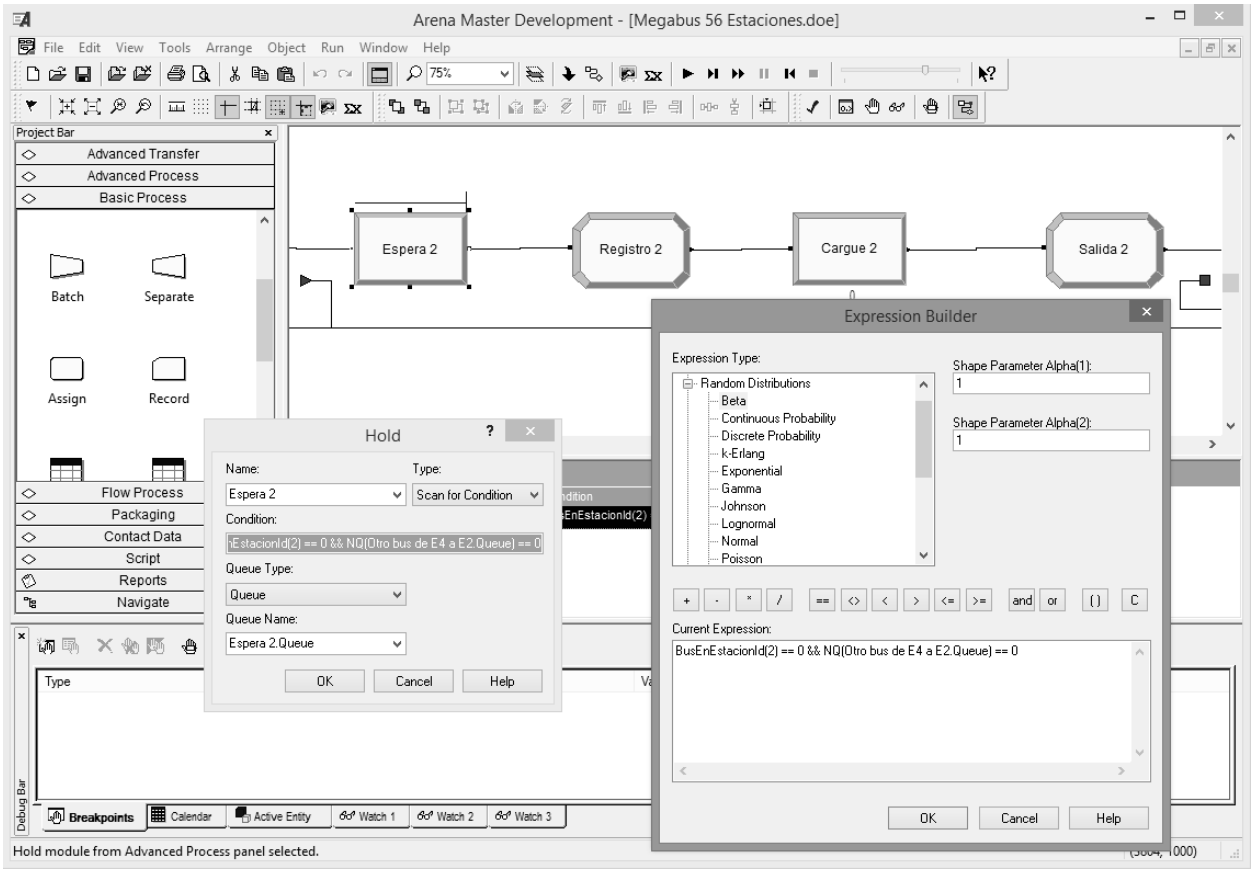

Figura 8 Captura de pantalla de la interfaz del software Arena (fuente propia)

A continuación se describen algunos de los elementos disponibles en Arena para la creación de modelos de simulación:

Módulos de datos: permiten definir estructuras de almacenamiento necesarias para la implementación de la simulación:

- Variable (variable): permite la definicion de nombre, tamaño, tipo de dato y valores iniciales de las variables globales. Estas variables pueden ser accesadas desde cualquier modulo de flujo de trabajo y son unicas durante la ejecución de toda la simulación.
- Entidad (entity): permite la descripcion de los diferentes elementos que se transfieren entre los modulos de flujo de trabajo. Por ejemplo: buses, productos, materiales, componentes, pasajeros, etc.
- Atributo (attribute): permite describir las propiedades de las entidades. Cada propiedad tiene: nombre, tipo de dato y valor inicial. Cada instancia de una entidad puede tener diferentes valores para cada uno de sus atributos. Si una instancia de una entidad ingresa a un modulo de flujo de trabajo en cualquier momento de la simulación, ese módulo puede acceder y/o modificar el valor de todos los atributos de esa instancia.

Módulos de flujo de trabajo: realizan modificaciones a los módulos de datos. Tienen acceso a las variables en cualquier momento de la simulación y a los atributos de una instancia de entidad solo si dicha instancia se encuentra en él. Algunos de los módulos disponibles son:

- Crear (create): provee puntos de partida para entidades. Las entidades son creadas con una programación o basadas en intervalos de tiempo. Las entidades abandonan el modulo para iniciar el recorrido por el sistema. La entidad a crear se especifica dentro del módulo.
- Eliminar (dispose): borra las entidades y guarda estadísticas de esta acción.
- Proceso (process): el principal mecanismo de proceso en la simulación. Provee opciones para bloquear y desbloquear recursos y diferentes procesos en las instancias de entidades.
- Asignar (assign): permite la asignación de nuevos valores a variables, atributos de entidades, tipos de entidad u otras variables de sistema. En cada módulo se pueden realizar una o más asignaciones.
- Decisión (decide): permite tomar decisiones durante la simulación basadas en condiciones o probabilidad. Condiciones pueden ser basadas en valores de atributos, valores de variables, tipos de entidades o expresiones. Las decisiones pueden tener dos o más vías dependiendo de la configuración del módulo.
- Guardar (record): permite guardar estadísticas en el modelo de simulación. Dispone de varios tipos de estadísticas, incluyendo tiempo de espera, tiempo de vida, estadísticas por entidad (p.e. costo), y estadísticas de tiempo.
- Espera (delay): como su nombre lo indica, este módulo detiene una instancia de una entidad por un determinado periodo de tiempo. Cuando una entidad ingresa a un módulo de espera, la expresión de tiempo es evaluada y la entidad se mantiene en el módulo durante el tiempo resultado de la evaluación de dicha expresión.
- Mantener (hold): a diferencia del módulo de espera, este mantiene una entidad a la espera de que se cumpla la condición de salida. Además las entidades que ingresan a este módulo se encolan para dar orden a la evaluación de la condición de salida.

En la Figura 9 se presentan las imágenes que representan cada uno de los módulos de tipo flujo de trabajo en el lienzo de Arena.

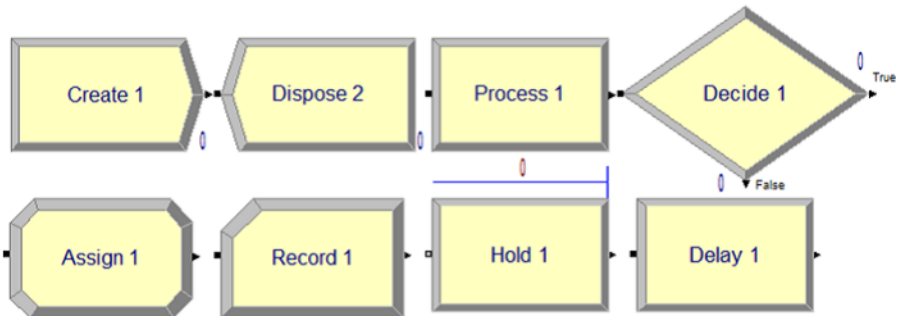

Figura 9 Módulos de tipo flujo de trabajo en el lienzo de Arena (fuente propia)

Sub-modelos: este componente permite agrupar módulos a voluntad del usuario para facilitar la gestión del modelo de simulación. Cada sub-modelo tiene un número determinado de puntos de entrada (cero o más), y un número determinado de puntos de salida (cero o más), dichos puntos sirven como puerto de comunicación con otros módulos o sub-módulos del modelo de simulación. Los elementos del STMP representados en el modelo de simulación (central de buses, estaciones de pasajeros, estaciones de buses, etc.), corresponden a un sub-modelo.

Arena basa su funcionamiento sobre el lenguaje de simulación SIMAN (Simulation and Analysis), y provee opciones para exportar los modelos de simulación a dicho lenguaje [16, 46].

3.3.2 SIMAN

SIMAN es un lenguaje de simulación multipropósito que permite la creación de modelos de simulación discretos usando la orientación al proceso; es decir, en un modelo de sistema particular, se estudian las entidades que se mueven a través del sistema. Una entidad para SIMAN es un cliente, un objeto que se mueve en la simulación y que posee características únicas conocidas como atributos. Los procesos denotan la secuencia de operaciones o actividades a través del que se mueven las entidades, siendo modeladas por un diagrama de bloques.

El código fuente de cualquier modelo de simulación en SIMAN se compone de dos archivos con extensiones .exp y .mod. El archivo .exp (Figura 10), contiene, entre otras, la definición de las Variables, Entidades y Atributos, mientras que en el archivo .mod (Figura 11), se registran los módulos que componen el modelo, el comportamiento de cada uno y las relaciones entre ellos.

Transporte.exp

```
1  PROJECT,       "Megabus","Unicauca",,,No,Yes,Yes,Yes,No,No,No,No,No,No;
2
3  ATTRIBUTES:    Pasajero.Ruta,DATATYPE(Real),0:
4                 Pasajero.Escala,DATATYPE(Real),0:
5                 Bus.Id,DATATYPE(Real),0;
6
7  VARIABLES:     Llego 13.NumberOut False,CLEAR(Statistics),CATEGORY("Exclude"):
8                 Mas Opciones 24.NumberOut False,CLEAR(Statistics),CATEGORY("Exclude"):
9                 Llego 32.NumberOut True,CLEAR(Statistics),CATEGORY("Exclude"):
10                Ingreso 14.NumberOut True,CLEAR(Statistics),CATEGORY("Exclude");
11
12 QUEUES:        Bus de E10 a E8.Queue,FIFO,,AUTOSTATS(Yes,,):
13                Otro bus de E45 a E44.Queue,FIFO,,AUTOSTATS(Yes,,):
14                Espera 3.Queue,FIFO,,AUTOSTATS(Yes,,);
15
16 PICTURES:      Picture.Airplane:
17                Picture.Green Ball:
18                Picture.Blue Page:
19                Picture.Red Ball;
20
21 TALLIES:       Guardar,,DATABASE(,"Expression","User Specified","Guardar");
22
23 REPLICATE,     1,,,Yes,Yes,,,,24,Minutes,No,No,,,Yes,No;
24
25 ENTITIES:      Pasajero,Picture.Person,0.0,0.0,0.0,0.0,0.0,0.0,AUTOSTATS(Yes,,):
26                Bus,Picture.Van,0.0,0.0,0.0,0.0,0.0,0.0,AUTOSTATS(Yes,,);
27
```

Figura 10 Ejemplo de archivo .exp con código SIMAN (fuente propia)

Transporte.mod

```
;
;     Model statements for module:  BasicProcess.Decide 543 (Bus Vacio)
;
5$            BRANCH,        1:
                             If,Bus.Capacidad==40,1589$,Yes:
                             Else,1590$,Yes;
1589$         ASSIGN:        Bus Vacio.NumberOut True=Bus Vacio.NumberOut True + 1:NEXT(4$);

1590$         ASSIGN:        Bus Vacio.NumberOut False=Bus Vacio.NumberOut False + 1:NEXT(6$);

;
;     Model statements for module:  AdvancedProcess.Hold 408 (Bus Con Pasajeros)
;
6$            QUEUE,         Bus Con Pasajeros.Queue;
              SCAN:          0:NEXT(4$);

;
;     Model statements for module:  AdvancedProcess.Delay 6 (Recorrido CB a E1)
;
22$           DELAY:         MinutesToBaseTime(NORM( 2, 0.2)),,Other:NEXT(8$);

```

Figura 11 Ejemplo de archivo .mod de código SIMAN (fuente propia)

Para realizar una ejecución de la simulación es necesario compilar los archivos fuente, hacer un proceso de "link" entre los archivos resultado de la compilación y finalmente ejecutar el resultado de todo ese proceso, en la Figura 12 se presenta un ejemplo de archivo .bat para ejecución de una simulación partiendo del código fuente SIMAN.

Transporte.bat

```
"C:\...\Rockwell Software\Arena\model.exe" "Transporte.mod" "Transporte.m"
"C:\...\Rockwell Software\Arena\expmt.exe" "Transporte.exp" "Transporte.e"
"C:\...\Rockwell Software\Arena\linker.exe" "Transporte.m" "Transporte.e" "Transporte.p"
"C:\...\Rockwell Software\Arena\siman.exe" "Transporte.p" "Transporte.out" -wn 2
```

Figura 12 Ejemplo de archivo .bat para ejecución de simulación en SIMAN (fuente propia)

Al final del proceso de simulación se obtiene un archivo con extensión .out con los resultados, en este archivo se encuentra la siguiente información: cantidad de entidades involucradas en el modelo, tiempos de espera, variables de sistema, variables de usuario, entre otra. Para cada uno de los ítems incluidos en este archivo se almacena: valor mínimo, valor máximo, valor promedio y número de observaciones realizadas durante la simulación para calcular dichos valores. En la Figura 13 se presenta un ejemplo del archivo resultado de la ejecución de una simulación en código SIMAN.

Transporte.out

```
                                    ARENA Simulation Results
                                  eruano - License: 7328734345
                                 Summary for Replication 1 of 1

Project: Megabús                                                Run execution date : 8/ 7/2015
Analyst: Unicauca                                               Model revision date: 8/ 7/2015
Replication ended at time      : 180.0 Minutes
Base Time Units: Minutes

                                          TALLY VARIABLES
Identifier                                Average   Half Width  Minimum    Maximum   Observations

Guardar                                   .36900    (Corr)      .00000     1.0000       2524
Pasajero.VATime                           .00000    (Insuf)     .00000     .00000        239
Pasajero.NVATime                          .00000    (Insuf)     .00000     .00000        239
Pasajero.WaitTime                         24.295    (Insuf)     2.2914     140.67        239
Pasajero.TranTime                         .00000    (Insuf)     .00000     .00000        239
Pasajero.OtherTime                        .00000    (Insuf)     .00000     .00000        239
Pasajero.TotalTime                        24.295    (Insuf)     2.2914     140.67        239

                                       DISCRETE-CHANGE VARIABLES
Identifier                                Average   Half Width  Minimum    Maximum   Final Value

Pasajero.WIP                              4743.4    (Corr)      .00000     9155.0     9155.0
Bus.WIP                                   32.420    (Insuf)     .00000     44.000     42.000
Bus de E10 a E8.Queue.NumberInQueue       3.5895    (Insuf)     .00000     40.000     .00000
Otro bus de E45 a E44.Queue.NumberInQueue .00000    (Insuf)     .00000     .00000     .00000
Bus de E3 a E5.Queue.NumberInQueue        .18291    (Insuf)     .00000     5.0000     .00000

                                           OUTPUTS
Identifier                                 Value

Pasajero.NumberIn                         9394.0
Pasajero.NumberOut                        239.00
Bus.NumberIn                              91.000
Bus.NumberOut                             49.000
System.NumberOut                          288.00

Simulation run time: 0.07 minutes.
Simulation run complete.
```

Figura 13 Ejemplo de archivo .out resultado de simulación SIMAN (fuente propia)

La conveniencia de SIMAN para el proyecto radica en la simplicidad tanto de su estructura de código como del mecanismo de compilación y ejecución, estas características facilitan la integración con lenguajes de programación que posean la capacidad de ejecutar aplicaciones del sistema operativo Windows vía batch (archivos por lotes) y leer/escribir archivos de texto.

El proceso entonces consiste en crear un modelo de simulación sobre Arena, con todas las facilidades de usabilidad y depuración que ello supone, para luego exportar dicho modelo a código SIMAN. Posteriormente, durante la ejecución del algoritmo, este código se lee, modifica y ejecuta para generar soluciones y evaluar el rendimiento de las mismas por parte del algoritmo multi-objetivo.

3.3.3 ModeLAB

La cantidad de componentes y comportamientos a tener en cuenta para crear un modelo de simulación de eventos discretos de un STMP en Arena que pueda utilizarse como insumo para la ejecución del algoritmo propuesto son muchos e incrementan la dificultad del proceso de modelado y el tiempo necesario para implementar dicho modelo. Para reducir la complejidad del proceso de modelado y evitar los gastos relacionados con la licencia de Arena se creó ModeLAB [47], una herramienta web para la construcción de modelos de simulación de STMP orientados a generar el código SIMAN necesario para la evaluación de la aptitud (fitness) de las soluciones generadas por el algoritmo propuesto.

El desarrollo de la primera versión de ModeLAB fue realizado en paralelo con este proyecto de investigación por los estudiantes Juan Pablo Salazar Restrepo y Carlos Robinson Campo Zambrano como parte de su trabajo de grado para optar al título de ingenieros de sistemas bajo la dirección del PhD. Carlos Alberto Cobos Lozada y la co-dirección del autor de esta tesis de maestría (MSc.(c) Edgar Fabian Ruano Daza).

ModeLAB reduce la complejidad de los módulos de Arena a sólo cuatro (4) de ellos: Central de buses, Estación simple, Estación doble y Cruces, los cuales pueden ser arrastrados y posicionados en un lienzo o área de modelado que ofrece todas las posibilidades de edición y configuración. Estos módulos pueden ser conectados entre sí y configurados a través de ventanas modales o de tablas (se presentan ambas opciones), pueden ser cortados, copiados, pegados o borrados del modelo así como también hay opciones para alinearlos, alejarlos o acercarlos. Los modelos pueden ser guardados para su posterior edición o pueden ser exportados a sentencias SIMAN.

En la Figura 14 se presenta una captura de pantalla de ModeLAB desplegado sobre el navegador Google Chrome.

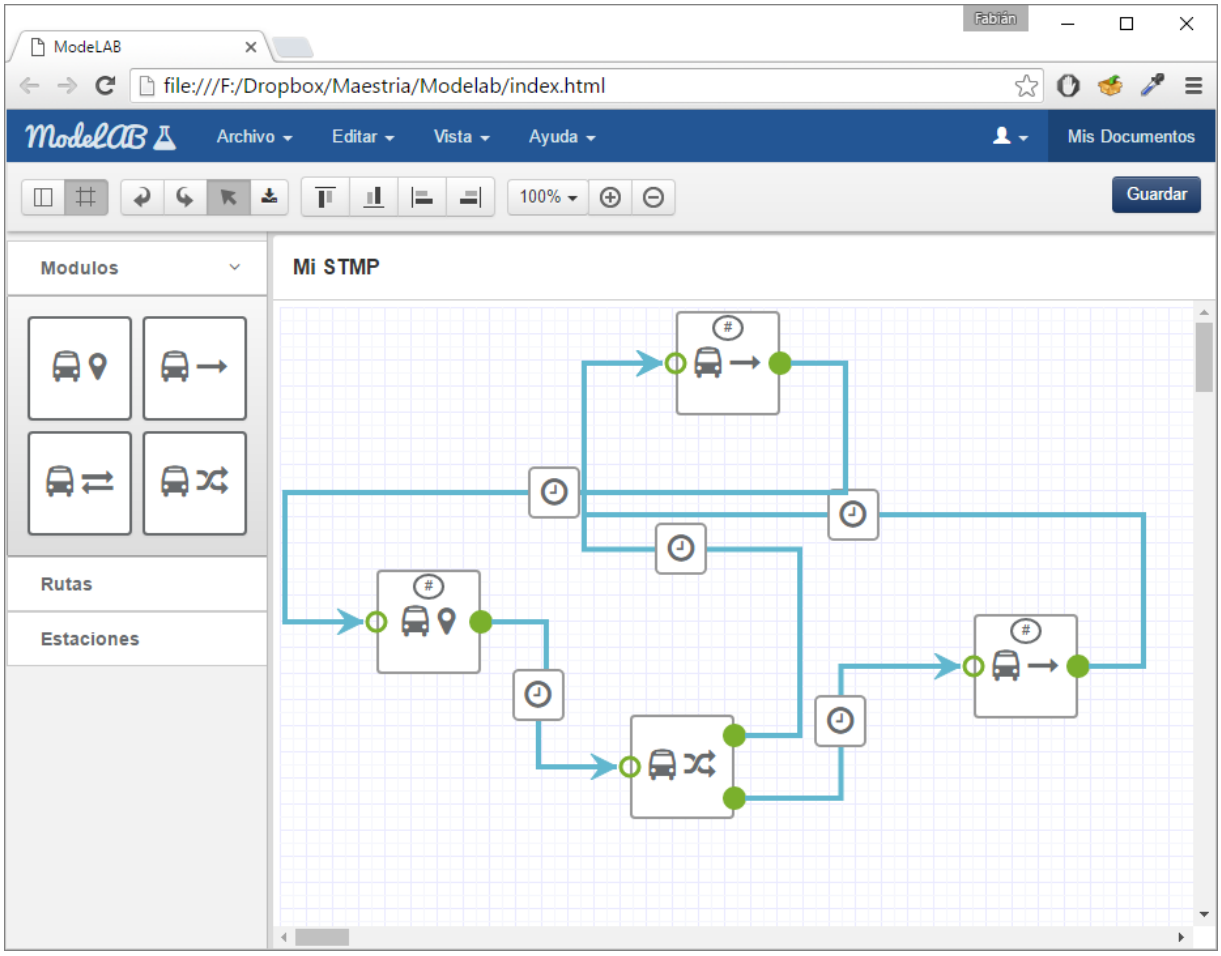

Figura 14 Captura de pantalla de ModeLAB desplegada en Google Chrome (fuente propia)

Aunque ModeLAB facilita el proceso de modelado y reduce costos, no se utiliza en el presente trabajo de investigación porque aún se encuentra en fase de pruebas y los límites de tiempo no permitieron esperar hasta que se finalizará la fase de puesta en producción.

4 Propuesta

Como resultado de la investigación se desarrolló un algoritmo multi-objetivo basado en la mejor búsqueda armónica global (MOGBHS, por sus siglas en inglés), soportado por un modelo de simulación de eventos discreto para resolver el problema de "diseño de rutas y programación de frecuencias en redes de transporte". En esa sección se describe la estructura del proceso de configuración de STMP incluyendo el algoritmo propuesto, las consideraciones de diseño del algoritmo y el funcionamiento de cada uno de los componentes del mismo.

4.1 Proceso de configuración de STMP usando MOGBHS y MOEAs en general

Para aplicar el algoritmo propuesto en la resolución del problema de configuración de rutas y frecuencias de un STMP, es necesario realizar un proceso de abstracción sobre el STMP para crear un modelo que permita al algoritmo identificar los elementos relevantes para generar soluciones para el problema en cuestión.

Retomando lo presentado previamente en el marco teórico, los algoritmos multi-objetivo no dan como resultado una única solución sino un conjunto de "mejores soluciones" que posteriormente serán analizadas por expertos en el dominio del problema para seleccionar, basados en la experiencia, la mejor solución a ser aplicada al sistema en cuestión.

De acuerdo con lo anterior el proceso de configuración de un STMP haciendo uso de un algoritmo multi-objetivo será como se presenta en la Figura 15. El primer paso es la evaluación por parte de expertos del STMP en cuestión para generar un modelo que sea entendible para el algoritmo, dicho modelo será tomado como insumo para ejecutar el proceso evolutivo dando como resultado un conjunto de "mejores soluciones" para la configuración del STMP, este conjunto debe ser evaluado por personal calificado para seleccionar la configuración final que posteriormente se aplicará al STMP real.

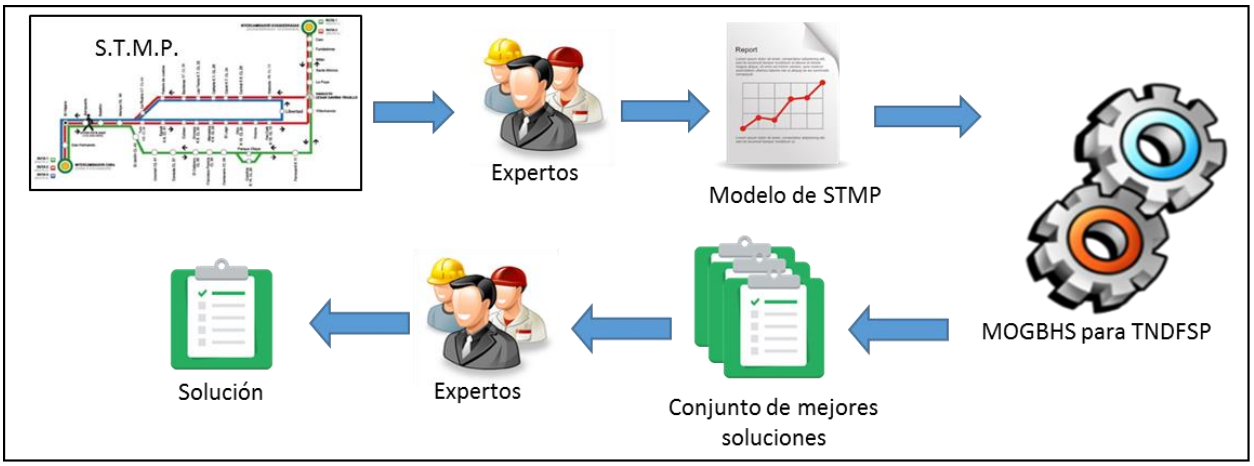

Figura 15 Proceso de configuración de STMP usando MOEAs (fuente propia)

A continuación se describen brevemente las etapas del proceso de configuración de STMP utilizando el algoritmo MOGBHS propuesto.

Etapa 1: Modelado del STMP

La fase inicial del proceso tiene como objetivo generar un modelo de simulación de eventos discretos del STMP en cuestión, que sea manipulable por el algoritmo propuesto. En esta fase es necesario personal con conocimientos específicos en el software de simulación seleccionado y sobre el funcionamiento del STMP a modelar, dicho personal será encargado de tomar información sobre el STMP, abstraerla para posteriormente crear el modelo en el software teniendo en cuenta los lineamientos propuestos en el presente documento. Una vez generado y afinado el modelo de simulación, es necesario generar archivos de código fuente y realizarles algunas modificaciones para permitir que el algoritmo propuesto pueda utilizarlos como insumo para entender el sistema, generar y evaluar las soluciones.

Etapa 2. Ejecución de MOGBHS

La segunda fase del proceso tiene como insumos los archivos resultados de la primera fase y el motor de compilación y ejecución de SIMAN. El algoritmo propuesto genera una lista con las mejores soluciones encontradas, donde cada una de ellas incluye la definición del conjunto de rutas para el STMP y las frecuencias para dichas rutas.

Etapa 3. Selección de la solución e implementación de la misma sobre el STMP

La última fase requiere nuevamente intervención humana, se analiza el conjunto de soluciones generadas por el algoritmo para seleccionar la que se considere más adecuada dependiendo de las políticas de administración del STMP y la experiencia del personal. Una vez seleccionada la configuración a implementar se realiza su implantación en el sistema real.

4.2 Modelo de simulación STMP

En esta sección se presentan lineamientos para la creación del modelo de simulación del STMP y las acciones a realizar posterior a la creación del modelo, orientadas hacia el uso de dicho modelo por parte del algoritmo propuesto. Como se mencionó en el estado del arte (sección 3.3), el software seleccionado para la creación de modelos de simulación de eventos discretos es Arena, por tanto los lineamientos y acciones descritas en esta sección están basados en los componentes disponibles en Arena y los archivos de código SIMAN. Una vez se refine el funcionamiento de ModeLAB se planea mover toda la sección de creación del modelo de simulación a dicha herramienta para facilitar el proceso y reducir costos de implementación.

4.2.1 Lineamientos para crear modelo de simulación de STMP sobre Arena

Con el objetivo de garantizar la integración del algoritmo propuesto con el código fuente SIMAN resultado del modelado del STMP en Arena, se definieron unos lineamientos a seguir para la creación del modelo de simulación. En esta sección se describe como se deben usar y relacionar los componentes de Arena para crear un modelo de simulación de un STMP que sirva como herramienta de evaluación de fitness para las soluciones generadas por el MOGBHS.

Entidades y Atributos:

- *Pasajero*: Entidad que como su nombre lo indica representa un usuario del sistema. Se sugiere establecerle un icono que simbolice una persona para facilitar su seguimiento en el modelo de simulación. Los siguientes son los atributos de la entidad Pasajero, es necesario que los nombres iguales a los que se listan:
    - Pasajero.Tiempo: Tipo de dato: Real.
    - Pasajero.Escala: Tipo de dato: Real
    - Pasajero.Bus: Tipo de dato: Real.
    - Pasajero.Ruta: Tipo de dato: Real
    - Pasajero.Recorrido: Tipo de dato: String
    - Pasajero.Destino: Tipo de dato: Real
    - Pasajero.Origen: Tipo de dato: Real
    - Pasajero.RecorridoTotal: Tipo de dato: String
- *Bus*: Entidad que como su nombre lo indica representa un bus en transito dentro del sistema. Se sugere establecerle un icono que simbolice un bus para facilitar su seguimiento en el modelo de simulación. Los siguientes son los atributos de la entidad Bus, es necesario que los nombres sean iguales a los que se listan:
    - Bus.Ruta: Tipo de dato: Real
    - Bus.Id: Tipo de dato: Real
    - Bus.Capacidad: Tipo de dato: Real
    - Bus.CapacidadTotal: Tipo de dato: Real

Variables:

- *RutaIntervalos*: Arreglo de tipo Real, de tamaño correspondiente a la cantidad de rutas en el sistema y que contiene la frecuencia de salida de los buses de cada una de las rutas que operan en el sistema.
- *RutaMaxBuses*: Arreglo de tipo Real, de tamaño correspondiente a la cantidad de rutas en el sistema y que define la cantidad maxima de instancias tipo “Bus” que se crearan para cada una de las rutas en el sistema.
- *PasajeroIntervalos*: Arreglo de tipo Real, de tamaño correspondiente a la cantidad de estaciones del sistema y que determina la frecuencia de llegada de los arribos de pasajeros a cada una de las estaciones.
- *PasajeroMaximo*: Arreglo de tipo Real, de tamaño correspondiente a la cantidad de estaciones del sistema y que determina la cantidad maxima de instancias de tipo “Pasajero” que se crearán para ingresar a las estaciones.
- *PasajeroNoPorArribo*: Arreglo de tipo Real, de tamaño correspondiente a la cantidad de estaciones del sistema y que determina el numero de entidades tipo “Pasajero” que se crean simultaneamente para ingresar a las estaciones

- *RutasCortas*: Matriz bidimensional de tamaño “Cantidad de estaciones” x “Cantidad de estaciones” de tipo “String” en la cual, de acuerdo a la configuracion de rutas, se informa cual o cuales son las rutas mas rapidas para llegar desde una estación a otra en el sistema. Permite que las entidades tipo pasajero se comporten deacuerdo a la disponibilidad de rutas actual.
- *Rutas*: Matriz bidimensional de tamaño “Cantidad de estaciones” x “Cantidad de rutas” de tipo Real mediante la cual se determina el recorrido de las rutas en el sistema y las estaciones de parada de las mismas (una ruta puede pasar por una estación sin detenerse en ella).
- *BusSiguienteId*: contador de tipo Real que permite establecer un identificador unico a cada una de las entidades tipo “bus” creadas en el sistema, esta es utilizada para sincronizar el movimiento de las entidades tipo “bus” con las entidades tipo “pasajero”.
- *ContadorPasajerosDestino*: Array de tamaño correspondiente a la cantidad de estaciones de tipo Real en el cual se almacena la cantidad de entidades tipo “Pasajero” que tienen como destino cada una de las estaciones del sistema.
- *CantidadBuses*: Array de tamaño equivalente a la cantidad de rutas de tipo Real en el cual se almacena la cantidad de entidades tipo “Buse” creadas para cada una de las rutas configuradas en el sistema.
- *BusEnEstacionId*: Array de tamaño correpondiente a la cantidad de estaciones de tipo Real en el cual se almacena el identificador unico de la entidad tipo “Bus” que esta detenida en cada una de las estaciones del sistema.
- *ProbabilidadDestinos:* Array de tipo String, de tamaño correspondiente a la cantidad de estaciones en el sistema en el cual se almacenan definiciones de funciones de probabilidad acumulada que permiten determinar la probabilidad de que un pasajero que ingresa al sistema vaya a una determinada estación.

Sub-Modelos y Módulos:

- *Central de buses*: este sub-modelo, como su nombre lo indica, representa una central desde donde parten buses para cumplir con determinadas rutas y a donde llegan los mismos luego de finalizar su recorrido por el sistema. Debe contener los siguientes modulos:
    - Ruta [# ruta]: Tipo Crear, encargado de la creación de entidades tipo Bus que seguiran una ruta específica. Se debe construir un modulo de este tipo por cada ruta a configurar en el sistema, los módulos deben ser únicos por ruta y pueden estar distribuidos entre las diferentes centrales de buses del modelo. El módulo debe configurarse como sigue:
        - Nombre: Ruta [# ruta]
        - TipoEntidad: Bus
        - Tipo: Expression
        - Expresión: RutaIntervalos([# ruta])
        - Unidades: Minutes

- Entidades por arribo: 1
- Máximo llegadas: RutaMaxBuses([# ruta])
- Primera cración: 0

- Asignar Ruta [# ruta]: Tipo Asignar, este módulo se encarga de establecer el numero de ruta a las entidades tipo bus creadas por su correspondiente módulo "Ruta [# ruta]". El módulo debe realizar la siguiente asignación:
  - Atributo, "Bus.Ruta" = [# ruta]
- Inicializar Ruta [# central]: Tipo Asignar, este módulo se encarga de establecer el indicador único para cada bus que recorre el sistema, incrementa dicho contador y establece la capacidad disponible de transporte de los buses. Realiza las siguientes asignaciones:
  - Variable, "BusSiguienteId" = "BusSiguienteId" + 1
  - Atributo, "Bus.Id" = "BusSiguienteId"
  - Atributo, "Bus.CapacidadTotal" = <Capacidad de buses del sistema>
  - Atributo, "Bus.Capacidad" =<Capacidad de buses del sistema>
- Salida buses [# central]: Tipo Eliminar, se encarga de recibir los buses que completan su recorrido por el sistema para eliminar dichas entidades. Solo necesita configuración del nombre.

En cuanto a puntos de entrada/salida este sub-modelo debe contener:

- Un (1) punto de entrada proveniente de uno o mas submodelos de tipo "recorrido de buses", "cruce de buses" y/o "estacion de buses.
- Un (1) punto de salida con destino a sub-modelos de tipo "recorrido de buses", "cruce de buses" o "estación de buses".

Las conexiones entre los módulos que componen el sub-modelo de central de buses se debe realizar como se ilustra en la Figura 16

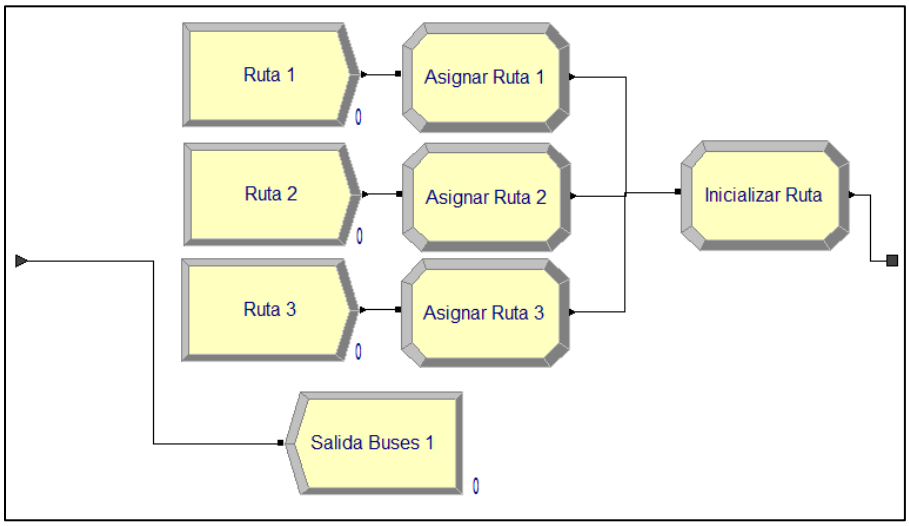

Figura 16 Sub-modelo de central de buses (fuente propia)

- *Estacion de buses:* este sub-modelo representa el comportamiento de las entidades tipo Bus en una estación del sistema, esto es: la llegada a la estación y espera de turno si otro bus esta estacionado en ella, la espera para embarque y desembarque de pasajeros y la salida de la estación. Debe contener los siguientes módulos:
  - Ingreso [# estación]: Tipo Decisión, encargado de evaluar si una entidad tipo bus debe detenerse en la estación dependiendo de la ruta que tenga definida dicha entidad. El módulo debera configurarse como sigue:
    - Nombre: Ingreso [# estación]
    - Tipo: 2 way by condition
    - Si: Variable Array (2D)
    - Nombrado: Rutas
    - Fila: Bus.Ruta
    - Columna: [# estación]
    - Es: "=="
    - Valor: 2

    Es de anotar que el valor dos (2) indica que el bus pasa y se detiene en la estación, el valor cero (0) indica que el bus no pasa por la estación y el uno (1) indica que pasa pero no se detiene en la estación.
  - Espera [# estación]: Tipo Espera, mantiene en espera a las entidades tipo Bus que intenten acceder a la estación hasta que exista espacio para su estacionamiento, se asume que en la estación solamente puede detenerse un bus a la vez. La configuración para este módulo sera la siguiente:
    - Nombre: Espera [# estación]
    - Tipo: Scan for condition
    - Condición: BusEnEstacionId([# estación]) == 0 && NQ(Otro bus de E[# estación] a E[# estación anexa].Queue) == 0
    - Tipo de cola: Queue
    - Nombre cola: "Espera [# estación].Queue"
  - Registro [# estación]: Tipo Asignación, se encarga de registrar el ingreso del bus a la estación e inicializar las variables que permiten simular el comportamiento de cargue y descargue de pasajeros. Las siguientes asignaciones se deben realizar en este módulo:
    - Variable Array (1D), BusEnEstacionRuta([# estación]) = Bus.Ruta
    - Variable Array (1D), BusEnEstacionCapacidad([# estación]) = Bus.Capacidad
    - Variable Array (1D), BusEnEstacionId([# estación]) = Bus.Id
  - Cargue [# estación]: Tipo Espera, simula el tiempo que tarda el bus en la estación a la espera de que embarquen o desembarquen pasajeros. La configuración de este módulo debe ser la siguiente:

- Nombre: Cargue [# estación]
- Tipo: Standard
- Acción: Delay
- Tipo espera: Normal
- Unidades: minutos
- Ubicación: Value added
- Valor: <tiempo medio de espera de buses en la estación>
- Desviación estandar: 0.1

- Salida [# estación]: Tipo Asignación, se encarga de registrar la salida de la entidad tipo bus de la estación y actualizar los datos de capacidad de la entidad saliente. Realiza las siguientes asignaciones:
  - Atributo, Bus.Capacidad = BueEnEstacionCapacidad([# estación])
  - Variable Array (1D), CantidadBuses([# estación]) = CantidadBuses([# estación]) + 1
  - Variable Array (1D), BusEnEstacionId([# estación]) = 0
- Guardar [# estación]: Tipo Guardar, permite guardar el registro del porcentaje de ocupación de un bus al momento que sale de la estación se detenga o no en la misma. Esto permitira obtener la estadística de capacidad desperdiciada al final de la simulación. La configuración del módulo debe ser la siguiente
  - Nombre: Guardar [# estación]
  - Tipo: Expresión
  - Valor: Bus.Capacidad / Bus.CapacidadTotal
  - Tally name: Guardar

En cuanto a puntos de entrada/salida este sub-modelo debe contener:

- Un (1) punto de entrada proveniente de uno o mas submodelos de tipo “recorrido de buses”, “cruce de buses” y/o “estacion de buses”.
- Un (1) punto de salida con destino a sub-modelos de tipo “recorrido de buses”, “cruce de buses” o “estación de buses”.

En algunos puntos del sistema se forman grupos de sub-módulos de tipo estación de buses para crear las estaciones de buses compuestas, sin embargo entre dichos sub-módulos puede que no exista ninguna conexión mediante sus puntos de entrada/salida.

Las conexiones entre los módulos que componen el sub-modelo de estación de buses se deben enlazar como se ilustra en la Figura 17.

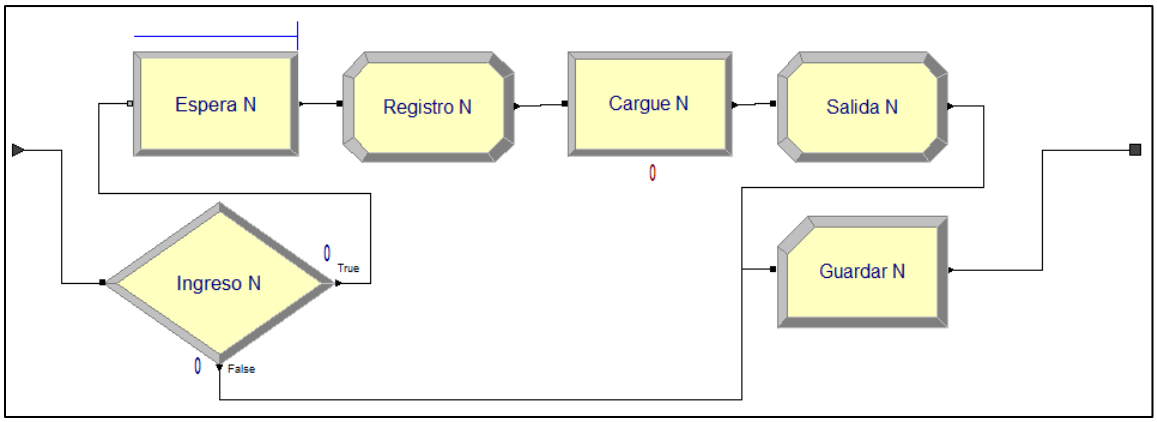

Figura 17 Sub-modelo de estación de buses (fuente propia)

- *Estación de pasajeros:* este sub-modelo representa el comportamiento de las entidades tipo Pasajero en una estación del sistema, los comportamientos representados son: Llegada de pasajeros a la estación, selección del destino y ruta a tomar, espera de un bus con la ruta deseada y con capacidad disponible, embarque y desembarque, conexión entre rutas y salida de la estación. Debe contener los siguientes módulos:
  - Pasajeros [# estación]: Tipo Crear, representa la afluencia de pasajeros a la estación, se encarga de crear pasajeros según las variables globales que definen cantidad e intervalos de llegada de pasajeros. El módulo debe configurarse como sigue:
    - Nombre: Pasajeros [# estación]
    - Tipo entidad: Pasajero
    - Tipo: Expresión
    - Expresión: POIS(PasajeroIntervalos([# estación]))
    - Unidades: minutos
    - Entidades por arribo: PasajeroNoPorArribo([# estación])
    - Máximo llegadas: PasajeroMaximo([# estación])
    - Primera cración: PasajeroTiempoInicio([# estación])
  - Destino [# estación]: Tipo Asignación, realiza la selección de un destino para las entidades tipo pasajero creadas en el sub-modelo dependiendo de las probabilidades configuradas en las variables globales. Además establece variables para identificar el origen del pasajero y para registrar la hora de ingreso al sistema. Las asignaciones a realizar son:
    - Atributo, Pasajero.Destino = EVAL(ProbabilidadDestinos([# estación], 1) + ProbabilidadDestinos([# estación], 2) + ProbabilidadDestinos([# estación], 3) + ProbabilidadDestinos([# estación], 4))
    - Atributo, Pasajero.Tiempo = TNOW
    - Atributo, Pasajero.Origen = [# estacion]
    - Variable Array (1D), ContadorPasajerosDestino(Pasajero.Destino) = ContadorPasajerosDestino(Pasajero.Destino) + 1

- Escala [# estación]: Tipo Asignación, establece al pasajero la ruta a seguir dependiendo del destino del mismo y los valores almacenados en la variable RutasCortas. La asignacion a realizar es:
  - Atributo, Pasajero.Recorrido = RutasCortas([# estación], Pasajero.Destino)
- Punto reingreso [# estación], Tipo Decisión, en el caso que la siguiente estación en la ruta corta del pasajero sea una estación anexa a la estación actual, lo envía hacia dicha estación sin que la entidad pasajero tenga que interactuar con entidades tipo bus. La configuración del módulo es la siguiente:
  - Nombre: Punto Reingreso [# estacion]
  - Tipo: 2 way by condition
  - Si: expression
  - Valor: Val(Mid(Pasajero.Recorrido, 4, 3)) == [# estación anexa]

  En este caso los valores 4 y 3 definen la ubicación del numero de estación siguiente dentro del atributo Pasajero.Recorrido que es de tipo String.
- Esperar bus [# estación], Tipo Espera, representa la espera que hace el pasajero hasta que un bus se detenga en la estación donde el se encuentra. La configuración del módulo es la siguiente:
  - Nombre: Esperar Bus [# estación]
  - Tipo: Scan for condition
  - Condición: BusEnEstacionId([# estación]) <> 0 && NQ(Bus de E3 a E5.Queue) == 0
  - Tipo cola: Queue
  - Nombre cola: Esperar Bus [# estación].Queue
- Recorrido a E[# estación anexa], Tipo Asignación, se encarga de registrar el paso por la estación actual en el historial de recorrido de la entidad Pasajero el cual se almacena en la variable Pasajero.RecorridoTotal. La asignación a realizar es:
  - Atributo, Pasajero.RecorridoTotal = Pasajero.RecorridoTotal + Mid(Pasajero.Recorrido, 1, 7)

  En este caso los valores 1 y 7 definen la ubicación de la ruta y numero de estación visitada dentro del atributo Pasajero.Recorrido que es de tipo String.
- Embarcar [# estación], Tipo Decisión, este módulo representa la decisión que debe tomar el pasajero una vez haya llegado un bus a la estación donde el se encuentra, dicha decisión se toma con base en la ruta del bus, la capacidad disponible y el destino del pasajero. La configuración del módulo es:
  - Nombre: Embarcar [# estación]
  - Tipo: 2 way by condition
  - If: Expression
  - Valor: BusEnEstacionCapacidad([# estación]) > 0 && BusEnEstacionRuta([# estación]) == Val(Mid(Pasajero.Recorrido, 1, 3))

En este caso los valores 1 y 3 definen la ubicación del numero de ruta a tomar dentro del atributo Pasajero.Recorrido que es de tipo String.

- Embarque [# estación]: Tipo Asignación, este módulo se encarga de representar el ingreso de un pasajero a un bus, registra la nueva ubicación del pasajero actualizando su historial de recorrido, el bus en donde se encuentra y restando capacidad al mismo. Las asignaciones a realizar son:
  - Atributo, Pasajero.Ruta = BusEnEstacionRuta([# estación])
  - Atrubuto, Pasajero.Bus = BusEnEstacionId([# estación])
  - Variable array (1D), BusEnEstacionCapacidad([# estación]) = BusEnEstacionCapacidad([# estación]) – 1
  - Atributo, Pasajero.Escala = Val(Mid(Pasajero.Recorrido, 4, 3))
  - Atributo, Pasajero.RecorridoTotal = Pasajero.RecorridoTotal + Mid(Pasajero.Recorrido, 1, 7)

  En este caso los valores 1 y 7 definen la ubicación de la ruta y numero de stación visitada dentro del atributo Pasajero.Recorrido que es de tipo String, mientras que los valores 4 y 3 definen unicamente la ubicación del numero de estación siguiente dentro del mismo atributo.
- Mas opciones [# estación]: Tipo Decisión, en el caso que el bus detenido en la estación no corresponda a la primera opción de viaje del pasajero se evalua si se tienen mas opciones disponibles, este módulo representa esta evaluación. La configuración debe ser de la siguente forma:
  - Nombre: Mas Opciones [# estación]
  - Tipo: 2 way by condition
  - Si: expression
  - Valor: Len(Pasajero.Recorrido) > 7

  Si la longitud de la variable Pasajero.Recorrido es mayor que siete (7) es porque existe al menos una opción que aun no ha sido evaluada.
- Siguiente opción [# estación]: Tipo Asignación, en este módulo se ubica la siguiente opción de viaje del pasajero en la primera posición para ser evaluada de nuevo en el módulo “Embarcar [# estación]”. La asignación a realizar es:
  - Atributo, Pasajero.Recorrido = Mid(Pasajero.Recorrido, 8)

  En este caso el valor 8 indica la ubicación de la seguna opción de viaje disponible en la variable Pasajero.Recorrido de tipo String.
- Espera otro bus [# estación]: Tipo Espera, representa el tiempo de espera entra la evaluación de la ultima opcion de viaje y la salida del bus de la estación. Luego de esta espera se reinicia el proceso de espera de un bus. La configuración del módulo es como sigue:
  - Nombre: Espera Otro Bus [# estación]
  - Tipo: Scan for condition

- Condition: BusEnEstacionId([# estación]) == 0
- Tipo de cola: Queue
- Nombre de cola: Espera Otro Bus [# estación].Queue

- Llegada [# estación]: Tipo Decisión, representa la acción que debe tomar un pasajero cuando el bus en el que esta viajando llega a una estación basado en el identificador de la estación y su estación objetivo. La siguiente es la configuración del módulo:
  - Nombre: Llegada [# estación]
  - Tipo: 2 way by condition
  - Si: Expression
  - Valor: Pasajero.Escala == [# estación]
- Desembarco [# estación]: Tipo Asignación, se encargar de registrar el desembarco de un pasajero en la estación mediante el incremento de la capacidad disponible del bus en el que viajaba. La asignación a realizar es:
  - Variable array (1D), BusEnEstacionCapacidad([# estación]) = BusEnEstacionCapacidad([# estación]) + 1
- Llego [# estación]: Tipo Decisión, representa la evaluación que realiza un pasajero al llegar a una estación: ¿este es mi destino final?. En caso de que lo sea se encamina a salir del sistema, en otro caso inicia la busqueda de su siguiente estación objetivo. La configuración del módulo es como sigue:
  - Nombre: Llego [# estación]
  - Tipo: 2 way by condition
  - Si: Attribute
  - Nombre atributo: Pasajero.Destino
  - Es: “==”
  - Valor: [# estación]
- Salir [# estación]: Tipo Eliminar, este módulo se encarga de recibir las entidades tipo Pasajero que han llegado a su destino. Representa la salida de un pasajero del sistema. Solo requiere la configuración del nombre.

En cuanto a puntos de entrada/salida este sub-modelo debe contener:

- Un (1) punto de entrada proveniente de uno o mas submodelos de tipo “recorrido de pasajeros” y/o “cruce de pasajeros”.
- Cero o un (0-1) puntos de entrada para “trasbordo de pasajeros”, este punto esta conectado a los puntos de salida para “trasbordo de pasajeros” de los sub-modelos de tipo “estación de pasajeros” con los cuales se agrupa el actual sub-modelo para crear las “estaciones de pasajeros compuestas”. En caso de tratarse de una estación simple este punto no es necesario.
- Un (1) punto de salida con destino a sub-modelos de tipo “recorrido de pasajeros” o “cruce de pasajeros”.

- Cero a muchos (0-N) puntos de salida para “trasbordo de pasajeros”, estos deben conectarse a los puntos de entrada para “trasbordo de pasajeros” de los sub-modelos de tipo “estación de pasajeros” con los cuales se agrupa el actual sub-modelo para crear las “estaciones de pasajeros compuestas”. En caso de tratarse de una estación simple, estos puntos de salida no son necesarios.

Las conexiones entre los módulos que componen el sub-modelo de estación de pasajeros se deben realizar como se ilustra en la Figura 18 para una estación compuesta y como se muestra en la Figura 19 para una estación simple.

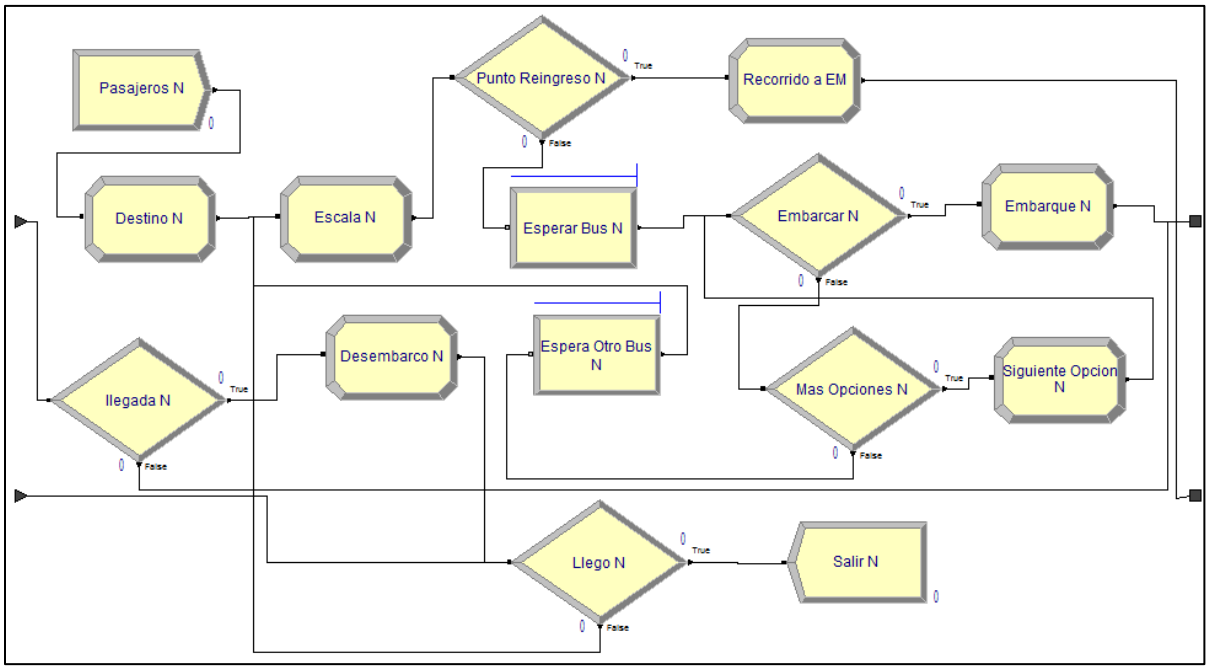

Figura 18 Sub-modelo de estación de pasajeros compuesta (fuente propia)

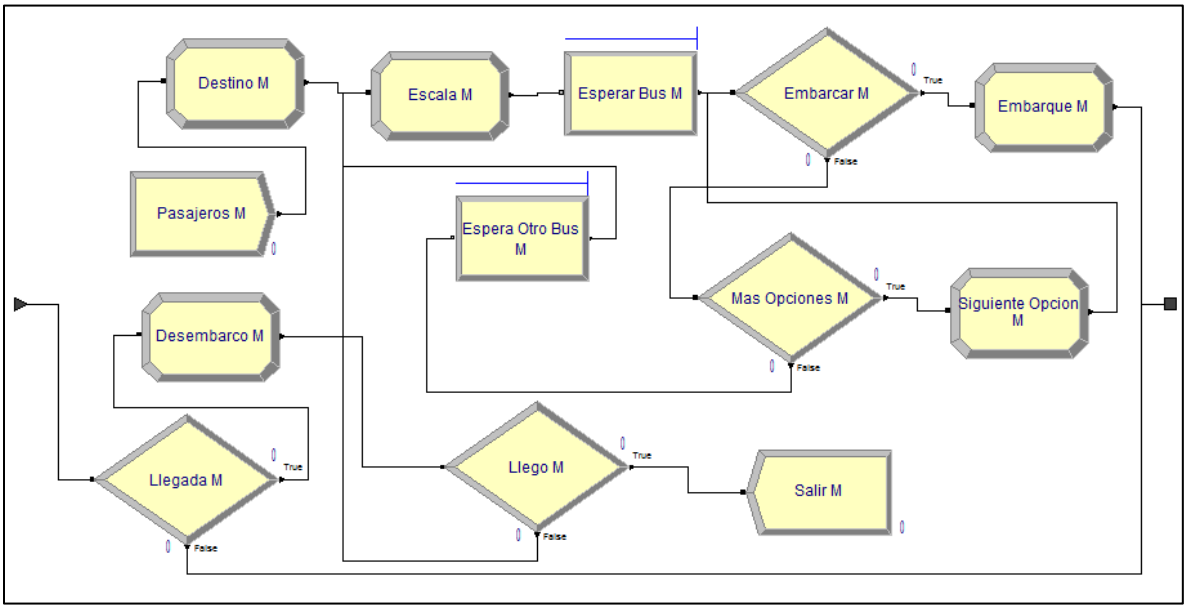

Figura 19 Sub-modelo de estación de pasajeros simple (fuente propia)

- *Recorrido de buses*: este sub-modelo representa el comportamiento de los buses durante los recorridos que realizan entre las diferentes estaciones del modelo en procura de cumplir con la ruta que les fue asignada. Debe contener un único módulo:
  - Recorrido [# estacion origen] a [# estación destino]: Tipo Espera, encargado de simular el tiempo que tarda un bus en realizar el recorrido entre dos estaciones. El tiempo de recorrido tendrá una media y cierta probabilidad de variación. El módulo se debe configurar como sigue:
    - Nombre: Recorrido E[# estacion origen] a E[# estacion destino]
    - Ubicación: Other
    - Tiempo espera: NORM([Media de tiempo de recorrido], [Varianza])
    - Unidades: minutos

    Los valores para “media de tiempo de recorrido” y “varianza” dependen de las estadisticas de tiempo que se tengan en el sistema real para cada uno de los recorridos entre las estaciones del modelo. Además de la Normal se pueden utilizar otras funciones para representar el calculo del tiempo de recorrido si se considera que existe otra formula que se asemeja mas al comportamiento del sistema real.

  En cuanto a puntos de entrada/salida este sub-modelo debe contener:

  - Un (1) punto de entrada proveniente de uno o mas submodelos de tipo “central de buses”, “estación de buses” y/o “cruce de buses”.
  - Un (1) punto de salida con destino a sub-modelos de tipo “central de buses”, “estación de buses” o “cruce de buses”.

  En la Figura 20 se representan las conexiones del sub-modelo de recorrido de buses.

Figura 20 Sub-modelo de recorrido de buses (fuente propia)

- *Cruce de buses*: este sub-modelo representa el comportamiento de los buses cuando llegan a una intersección de dos o mas vías en el sistema y deben decidir que vía tomar para cumplir con la ruta asignada. Debe contener un único módulo:
  - Cruce B[# de cruce]: Tipo Decisión, encargado de evaluar cual debe ser la ruta a seguir por una entidad tipo bus dependiendo de la ruta que le fue asignada. La configuración del módulo debe establecerse como sigue:
    - Nombre: Cruce B[# cruce]
    - Tipo: 2 way by condition
    - Si: Variable array (2D)
    - Nombrado: Rutas
    - Fila: Bus.Ruta
    - Es: >

- Columna: [# primera estación destino]
- Valor: 0

En caso que sea una decisión sobre mas de dos caminos debe utilizarse el tipo “N way by condition” y configurar las condiciones dependiendo de la cantidad de estaciones destino conectadas al cruce.

En cuanto a puntos de entrada/salida este sub-modelo debe contener:

- Un (1) punto de entrada proveniente de uno o mas submodelos de tipo “estación de buses” y/o “recorrido de buses”.
- Dos o más (2-N) puntos de salida con destino a sub-modelos de tipo “estación de buses” o “recorridos de buses”.

Las conexiones a realizar en el sub-módulo cruce de buses se deben realizar como se ilustra en la Figura 21.

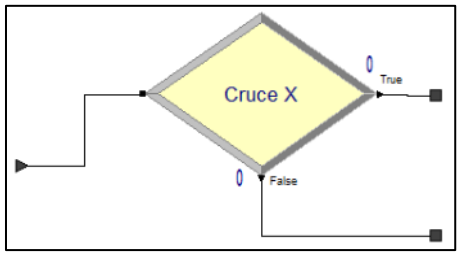

Figura 21 Sub-modelo de cruce de buses (fuente propia)

- *Recorrido pasajeros*: este sub-modelo representa el comportamiento de los pasajeros durante los recorridos entre estaciones a bordo de un bus del sistema. Los movimientos realizados en este sub-modelo por las entidades tipo Pasajero dependen de los movimientos realizados por las entidades tipo bus en los sub-modelos “estacion de buses” y “recorrido de buses”. Debe contener los siguientes módulos:
  - Bus de E[# estacion origen] a E[#estacion destino]: Tipo Espera, este módulo mantiene en espera a los pasajeros que estan realizando un recorrido entre dos estaciones de pasajeros, hasta que un bus arribe a la estación de buses equivalente a la estacion de pasajeros destino de su recorrido. La configuración del módulo es la siguiente:
    - Nombre: Bus de E[# estacion origen] a E[#estacion destino]
    - Tipo: Scan for condition
    - Condición: BusEnEstacionId([# estacion destino]) <> 0
    - Tipo de lista: Queue
    - Nombre de lista: Bus de E[# estacion origen] a E[#estacion destino].Queue
  - Mi bus de E[# estacion origen] a E[#estacion destino]: Tipo Decisión, en este módulo se verifica si el bus que llegó a la estación destino corresponde al bus en el cual esta embarcado el pasajero, en ese caso sale del sub-modelo, en caso contrario continúa la espera. La configuración del módulo debera ser la siguiente:
    - Nombre: Mi bus de EX a EY

- Tipo: 2 way by condition
- Si: Attribute
- Nombrado: Pasajero.Bus
- Es: ==
- Value: BusEnEstacionId([#estacion destino])

- Otro bus de E[# estacion origen] a E[#estacion destino]: Tipo Espera, este módulo mantiene en espera a los pasajeros que deben continuar en el módulo mientras que su bus llegue a la equivalente estación destino. Se debe configurar como sigue:
  - Nombre: Otro bus de E[# estacion origen] a E[#estacion destino]
  - Tipo: Scan for condition
  - Condición: BusEnEstacionId([# estacion destino]) == 0
  - Tipo de lista: Queue
  - Nombre de lista: Otro bus de E[# estacion origen] a E[#estacion destino].Queue

En cuanto a puntos de entrada/salida este sub-modelo debe contener:

- Un (1) punto de entrada proveniente de uno o mas submodelos de tipo “estación de pasajeros” y/o “cruce de pasajeros”.
- Un (1) punto de salida con destino a sub-modelos de tipo “estación de pasajeros” o “cruce de pasajeros”.

Las conexiones entre los módulos que componen el sub-modelo recorrido de pasajeros deben establecerse como se ilustra en la Figura 22

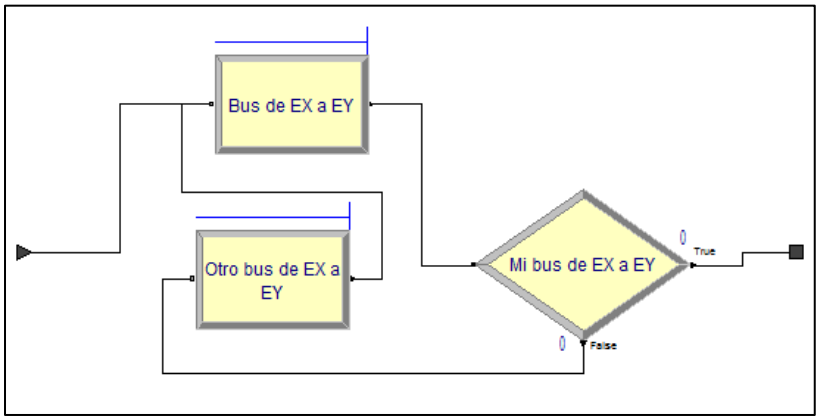

Figura 22 Sub-modelo de recorrido de pasajeros (fuente propia)

- *Cruce de pasajeros*: similar al cruce de buses, este módulo representa el comportamiento de los pasajeros cuando llegan a una intersección de dos o mas vías en el sistema y deben decidir que via tomar para cumplir la ruta seleccionada para su recorrido. Debe contener un único módulo:
  - Cruce P[# de cruce]: Tipo decisión, encargado de evaluar cual debe ser la ruta a seguir por una entidad tipo pasajero dependiendo de la ruta seleccionada para su recorrido. La configuración del módulo debe ser como sigue:

- Nombre: Cruce P[# de cruce]
- Tipo: 2 way by condition
- Si: Variable Array (2D)
- Nombrado: Rutas
- Fila: Pasajero.Ruta
- Columna: [# primera estación destino]
- Es: >
- Valor: 0

En caso que sea una decisión sobre mas de dos caminos debe utilizarse el tipo “N way by condition” y configurar las condiciones dependiendo de la cantidad de estaciones destino conectadas al cruce.

En cuanto a puntos de entrada/salida este sub-modelo debe contener:

- Un (1) punto de entrada proveniente de uno o mas submodelos de tipo “estación de pasajeros” y/o “recorrido de pasajeros”.
- Dos o más (2-N) puntos de salida con destino a sub-modelos de tipo “estación de pasajeros” o “recorrido de pasajeros”.

Las conexiones a realizar en el sub-módulo cruce de pasajeros se deben establecer como se muestra en la Figura 23.

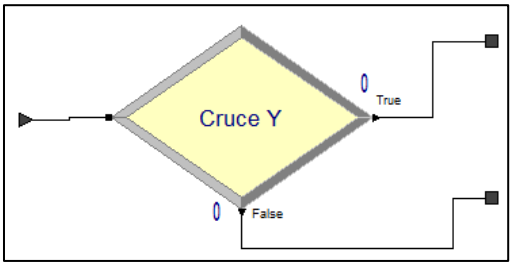

Figura 23 Sub-módulo de cruce de pasajeros (fuente propia)

Teniendo en cuenta los anteriores lineamientos para la creación del modelo de simulación en procura de dar soporte a la ejecución del algoritmo propuesto, se requiere la siguiente información del STMP:

- Mapa de rutas
- Mapa de conexiones físicas que pueden ser recorridas por los buses entre estaciones
- Información estadística sobre afluencia de pasajeros en cada una de las estaciones del sistema
- Información estadística sobre la tendencia de viaje de los pasajeros en el sistema
- Plan de operación
- Cantidad de buses disponibles y la capacidad de los mismos.

4.2.2 Modificaciones a los archivos de código fuente en SIMAN

Además de los lineamientos previamente descritos para la construcción del modelo de simulación y posterior al proceso de exportación de dicho modelo a archivos SIMAN, es necesario realizar ajustes al código fuente para garantizar que se puedan insertar las configuraciones de rutas y frecuencias determinadas por el algoritmo.

- Modificaciones al archivo .mod
  - Eliminar el código asociado a los modulos de Crear Ruta
  - Eliminar el código asociado a los modulos de Asignar Ruta
  - Adicionar tag `$DefinicionRutas` al final del archivo
- Modificaciones al archivo .exp
  - En la definicion de la variable RutaIntervalos es necesario incluir el tag `$intervalos` de forma que la sección de código tenga la siguiente estructura:

```
RutaIntervalos([No_rutas_modelo]),CLEAR(System),CATEGORY("User Specified-
User Specified"),DATATYPE(Real),$intervalos:
```

  - En la definicion de la variable RutasCortas es necesario establecer el tag `$RutasCortas` de forma que la sección de código tenga la siguiente estructura:

```
RutasCortas([No_Estaciones],[No_Estaciones]),CLEAR(System),DATATYPE(String
)$RutasCortas
```

  - En la definición de la variable RutaTiempoInicio es necesario incluir el tag `$CantidadRutas` de forma que la sección de código tenga la siguiente estructura:

```
RutaTiempoInicio($CantidadRutas)
```

  - En la definición de la variable RutaIntervalos es necesario incluir los tags `$CantidadRutas` e `$intervalos` de forma que la sección de código tenga la siguiente estructura:

```
RutaIntervalos($CantidadRutas),CLEAR(System),CATEGORY("User Specified-User
Specified"),DATATYPE(Real),$intervalos:
```

  - En la definición de la variable RutaMaxBuses es necesario incluir el tag `$CantidadRutas` de forma que la sección de código tenga la siguiente estructura:

```
RutaMaxBuses($CantidadRutas)
```

  - En la definición de la variable Rutas es necesario incluir los tags `$CantidadRutas` y `$DetalleRutas` de forma que la sección de código tenga la siguiente estructura:

```
Rutas($CantidadRutas, [No_Estaciones]),CLEAR(System),CATEGORY("User
Specified-User Specified"),DATATYPE(Real),$DetalleRutas:
```

Para la integración con el prototipo implementado es necesario modificar los nombres de los archivos codigo fuente SIMAN de forma que queden: Transporte.mod y Transporte.exp

Incluyendo los elementos propios de Arena y SIMAN, el proceso de configuración de STMP usando el algoritmo propuesto queda como se ilustra en la Figura 24

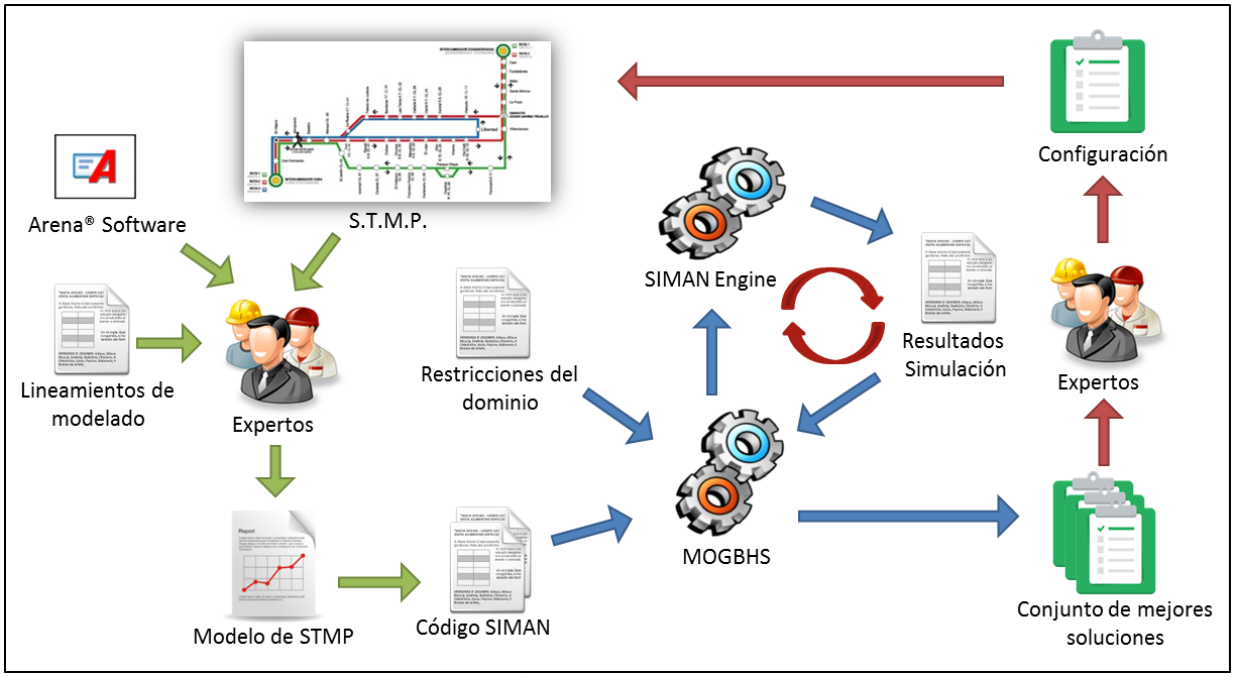

Figura 24 Proceso de optimización definido para utilizar MOGBHS (fuente propia)

4.3 Consideraciones de Diseño

En esta sección se describen las consideraciones y los elementos tenidos en cuenta durante el diseño del algoritmo propuesto para la búsqueda de las mejores configuraciones de rutas e intervalos de salida de buses para un STMP.

4.3.1 Selección de enfoque multi-objetivo

Como se vio previamente en el estado del arte (sección 2.1.1), la mayoría de los MOEAs reportados usan alguno de los siguientes tres enfoques para manejar los múltiples objetivos: Métodos ingenuos, métodos basados en ordenamiento de no-dominados y métodos basados en la fuerza de Pareto

Teniendo en cuenta los buenos resultados en problemas de optimización multi-objetivo utilizando el enfoque basado en ordenamiento de no-dominados [3, 23, 25, 37-39], dicha estrategia fue seleccionada para el algoritmo propuesto. Se decidió, además, incluir la Distancia de Dispersión (Crowding distance) como mecanismo para garantizar mayor exploración de las soluciones durante la evolución del algoritmo. En consecuencia, el algoritmo propuesto considera como mejores soluciones aquellas que se encuentren en el primer frente de Pareto (o frentes inferiores) y que tengan un mayor valor en su evaluación de distancia de dispersión.

4.3.2 Selección del algoritmo base

El algoritmo de optimización Harmony Search (HS) fue originalmente propuesto en [21], el cual se basa en el proceso de improvisación utilizado por los músicos de jazz en la búsqueda de la armonía perfecta, a grandes rasgos: HS genera aleatoriamente, evalúa y ordena la población

en un lugar llamado Memoria Armónica (HM por sus siglas en inglés). Luego, por un determinado número de improvisaciones genera una nueva armonía cada vez, donde cada variable puede ser completamente aleatoria o tomada desde la memoria armónica (esta probabilidad es definida por el parámetro Harmony Memory Consideration Rate – HMCR), en caso de tomar desde la memoria armónica, puede o no (dependiendo del parámetro Pitch Adjustment Rate – PAR), ser alterada mediante la adición o sustracción del valor definido en el parámetro Bandwith (BW). En la Figura 25 se presenta el pseudo-código de HS.

```
01  HM.Initialize(VariableLimits)
02  for i = 1 to NIterations do
03    NewHarmony = New Harmony()
04    for j = 1 to NVariables do
05      if Random(0,1) < HMCR then
06        NewHarmony.Variables[j] = HM.Harmonies[Random(0,HMS)].Variables[j]
07        if Random(0,1) < PAR then
08          if Random(0,1) < 0.5 then
09            NewHarmony.Variables[j] += BW
10          else
11            NewHarmony.Variables[j] -= BW
12          end if
13        end if
14      else
15        NewHarmony.Variables[j] = Random(0, 1)
16      end if
17    end for
18    if HM.BetterThanOne(NewHarmony) then
19      HM.Add(NewHarmony)
20      HM.Sort()
21      HM.RemoveTheWorst()
22    end if
23  end for
```

Figura 25 Pseudo-código de Harmony Search (basado en [21]).

Posterior a su publicación se han generado variantes o mejoras para el algoritmo HS, algunas de las más relevantes son:

- Improved Harmony Search (IHS), hace que los valores para PAR y BW dependan del numero de iteración en que se encuentre lo que garantiza una mayor exploración en las primeras iteraciónes y mayor explotación en las ultimas [18].
- Global Best Harmony Search (GBHS), reemplaza el ajuste por adición/sustraccion del valor de la variable BW por un una extracción de un valor desde la mejor armonía almacenada en la Memoria Armonica [19].
- Self-adaptive Harmony Search, reemplaza el ajuste fijo dado por el parametro BW por un calculo aleatorio entre el minimo y maximo valor de la HM para la variable a ajustar. Ademas reemplaza la generación aleatoria por secuencias de baja discrepancia para el calculo de la memoria inicial [48].
- Self-adaptive Global Best Harmony Search (SGHS), modifica los parámetros HMCR y PAR para que se adapten dinamicamente mediante mecanismos de aprendizaje y el parametro

BW se modifica para incentivar la exploración en etapas iniciales y la explotación en finales [17]

- Global Best Harmony Search +LEM, introduce modelos evolutivos de aprendizaje para mejorar la convergencia y efectividad del algoritmo [49]
- Global Dynamic Harmony Search (GDHS), hace que todos los parametros del algoritmo sean dinámicos, por tanto no hay que hacer configuracion del algoritmo. El parámetro PAR se incrementa linealmente con el numero de generacion, mientras que BW se decrementa exponencialmente con el numero de generacion actual, HMCR se mantiene en valores altos hasta la mitad del numero de generaciones y luego comienza a decrementar su valor para evitar caer en optimos locales [50].
- Improved Global Best Harmony Search (IGBHS), propone una mejora al rendimiento de HS haciendo que se explore mas al inicio de la ejecucion del algoritmo mediante la improvisación de armonias tomando aleatoriamente elementos de la memoria armónica y haciendo grandes modificaciones, y se explote mas al final del proceso mediante pequeños ajustes a los mejores elementos de la HM [51]

Teniendo en cuenta que: 1) HS, IHS y GBHS tienen en común que necesitan un pequeño número de parámetros para calibrar el rendimiento del algoritmo (7 parámetros para IHS y 5 para los otros dos); 2) HS, IHS y GBHS tienen rápida convergencia, requieren recursos de computo modestos, y tienen baja probabilidad de quedar atrapados en óptimos locales [17, 21]; 3) GBHS mejora los resultados de IHS (que a su vez mejora la precisión y convergencia de HS); y 4) GBHS trabaja eficientemente tanto en problemas discretos como continuos; en esta investigación se decidió usar GBHS como la base del algoritmo multi-objetivo propuesto. En la Figura 26 se presenta el pseudo-código de GBHS.

```
01 HM.Initialize(VariableLimits)
02 HM.Sort()
03 for i = 1 to NIterations do
04   NewHarmony = New Harmony()
05   for j = 1 to NVariables do
06     if Random(0, 1) < HMCR then
07      NewHarmony.Variables[j] = HM.Harmonies[Random(0,HMS)].Variables[j]
08      PAR = PARMin + (((PARMax - PARMin) / NIterations) * i)
09      if Random(0, 1) < PAR then
10        NewHarmony.Variables[j] = HM.FromBestHarmony(Random(0, NVariables))
11      end if
12     else
13       NewHarmony.Variables[j] = Random(0, 1)
14     end if
15   end for
16   if HM.BetterThanOne(NewHarmony) then
17     HM.Add(NewHarmony)
18     HM.Sort()
19     HM.RemoveTheWorst()
20   end if
21 end for
```

Figura 26 Pseudo-código de Global Best Harmony Search (fuente propia)

4.3.3 Selección de objetivos a optimizar

Entendiendo que el problema objetivo de este proyecto de investigación se enmarca dentro del grupo TNDFSP (Transit Network Desing and Frequencies Settings Problems), y considerando los trabajos previamente realizados en esta área mediante diferentes enfoques [4, 26, 52-58], en los cuales se considera prioritaria la satisfacción del usuario y los costos operacionales del sistema, se identificaron como objetivos de optimización:

- Minimizar el tiempo de uso del sistema por parte de los pasajeros, entendiendo que la satisfacción de los usuarios depende casi en su totalidad de la minimización de dicho tiempo.
- Minimizar los costos de operación, para ello se considera que los desperdicios de capacidad de transporte del sistema influyen negativamente en los costos de operación del mismo, por tanto al minimizar la capacidad desperdiciada de los buses en tránsito se minimizaran los costos de operación.

4.3.4 Evaluación de fitness

Teniendo en cuenta que el algoritmo propuesto es multi-objetivo, y que utiliza ordenamiento de no-dominados y distancia de dispersión para la clasificación y selección de las mejores soluciones encontradas, los objetivos a minimizar son evaluados de forma independiente.

El tiempo de uso del sistema por parte de los pasajeros corresponde al promedio de tiempo contado desde que el pasajero ingresa a una estación para esperar la ruta deseado hasta que sale de la estación en su destino final. Haciendo uso del modelo de simulación en SIMAN, es posible obtener esta medida directamente desde el archivo de resultado de ejecución de la simulación (*.out).

La capacidad desperdiciada de los buses corresponde al promedio de porcentaje de ocupación de los buses desde que salen de la central de buses hasta que finalizan su recorrido en la misma o en otra central. El porcentaje de capacidad desperdiciada de un bus se define como: *[Capacidad Disponible del Bus] / [Capacidad Total del Bus].* Haciendo uso del modelo de simulación en SIMAN y los módulos tipo “Registro” que este lenguaje soporta, es posible obtener esta medida directamente desde el archivo de resultado de la ejecución de la simulación (*.out).

4.4 Multi-Objective Global Best Harmony Search (MOGBHS)

HS y sus mejoras (incluyendo a GBHS), fueron originalmente diseñados para trabajar con un único objetivo. Para crear el algoritmo propuesto se modificó GBHS de forma que permitiera la optimización simultánea de varios objetivos haciendo uso de ordenamiento de no-dominados y distancia de dispersión (una modificación similar realizada a HS se describe en [59]), convirtiendo al algoritmo propuesto en la primera versión multi-objetivo de GBHS.

El algoritmo propuesto, llamado Multi-Objective Global Best Harmony Search (MOGBHS), genera aleatoriamente un conjunto de armonías y las almacena en la Memoria Armónica (HM), evalúa todos los objetivos para cada elemento en la HM, para luego ordenar por frente de Pareto y distancia de dispersión teniendo en cuenta esas evaluaciones. Posteriormente se

realizan un número determinado de improvisaciones (iteraciones evolutivas), en cada una de las cuales: a) se genera una nueva armonía aplicando la lógica del algoritmo GBHS; b) se evalúa la nueva armonía frente a todos los objetivos a optimizar; c) se adiciona la nueva armonía a la HM existente; d) se ordena la HM por frentes de Pareto y distancia de dispersión; y e) se eliminan todos los elementos que hagan que la HM exceda su tamaño máximo definido por el parámetro HMS (Harmony Memory Size) del algoritmo, siendo estos los peores elementos contenidos en la HM.

Las variables del algoritmo MOGBHS son:

- HM: Arreglo que almacena todas las soluciones propuestas por el algoritmo, su tamaño máximo es determinado por el parámetro HMS.
- PAR: Pitch Adjustment Rate, determina la probabilidad de que un valor previamente tomado desde la memoria armónica sea reemplazado por un valor tomado desde la mejor armonía existente en HM. Teniendo PARmin, PARmax y NIter como parámetros del algoritmo, e I como una variable, el valor de PAR está determinado por la fórmula:

  PAR = PARmin + (((PARmax - PARmin) / NIter) * I)
- I: Contador de improvisaciones realizadas por el algoritmo.
- J: Contador de las variables de una solución (armonía) en cada improvisación.

Los parámetros necesarios para la ejecución de MOGBHS son:

- HMS: Harmony Memory Size, tamaño máximo que puede alcanzar la memoria armónica.
- HMCR: Harmony Memory Consideration Rate, determina la probabilidad de seleccionar valores para las nuevas armonías desde la memoria armónica.
- NVariables: Numero de variables que involucran cada armonía, depende de la configuración utilizada para representar las soluciones al problema.
- NIter: Número de iteraciones/improvisaciones a realizar como parte de la ejecución del algoritmo.
- NObjectives: número de objetivos que se van a optimizar.

En la Figura 27 se presenta el pseudocódigo de MOGBHS y posteriormente se describen las funciones involucradas en el funcionamiento del mismo.

```
01 HM.PopulationRandomInitialize()
02 HM.NonDominatedOrderCalculate()
03 HM.CrowdingDistanceCalculate()
04 HM.Sort()
05 for I = 1 to NIter do
06   NewHarmony = New Harmony()
07    for J = 0 to NVariables - 1 do
08      if Random(0,1) < HMCR then
09        NewHarmony.Variables[J] = HM.Harmonies[Random(0,HMS)].Variables[J]
10        PAR = PARMin + (((PARMax - PARMin) / NIterations) * I)
11        if Random(0,1) < PAR then
12         NewHarmony.Variables[J] = HM.FromBestHarmony(Random(0, NVariables))
13       end if
14      else
15        NewHarmony.Variables[J] = Random(MinDomainValue, MaxDomainValue)
16      end if
17    end for
18    if HM.InPopulation(NewHarmony) = false AND NewHarmony.IsViable() then
19      NewHarmony.Evaluate()
20      HM.Add(NewHarmony)
21      HM.NonDominatedOrderCalculate()
22      HM.CrowdingDistanceCalculate()
23      HM.Sort()
24      HM.RemoveTheWorst()
25    end if
26 end for
```

Figura 27 Pseudo-código Multi-Objective Global Best Harmony Search (fuente propia)

La función *PopulationRandomInitialize* es responsable de la generación de una población inicial de armonías de tamaño HMS. En la Figura 28 se presenta el pseudocódigo de esta función.

```
01 count = 1
02 while count < HMS
03    for k = 0 to NVariables - 1 do
04      NewHarmony.Variables[k] = Random(MinDomainValue, MaxDomainValue)
05    end for
06    if NewHarmony.IsViable() then
07      NewHarmony.Evaluate()
08      HM.Add(NewHarmony)
09      count = count + 1
10    end if
11 end while
```

Figura 28 Función PopulationRandomInitialize (fuente propia)

La función *Evaluate,* establece los valores de fitness que la armonía obtiene al ser evaluada contra todos los objetivos del problema. Estos valores son almacenados en el vector Evaluations de tamaño NObjectives que cada armonía debe tener. El comportamiento de esta función depende del dominio sobre el cual se esté aplicando el algoritmo (en este caso particular el dominio corresponde al TNDSP), y de los objetivos que se están optimizando (Ver sección 4.3.3). La lógica de esta función se presenta en las secciones 4.5 y 4.5.3 de este documento.

La función *NonDominatedOrderCalculate* establece a cada elemento de la memoria armónica el número de frente de pareto al que pertenece. Para esta función se asume que cada solución (armonía) tiene los atributos:

- Front: determina el numero de frente de pareto al que pertenece.
- BossCount: almacena el numero de soluciones de la población que la dominan.
- Dominated: lista que permite almacenar los elementos de la solución a los cuales domina.

En la Figura 29 se presenta el pseudocódigo de la funcion NonDominateOrderCalculate.

```
01 for m = 0 to HM.Length() do
02   for n = 0 to HM.Length() do
03     if m != n then
04       if Dominates(HM[m], HM[n]) = true then
05         HM[m].Dominated.Add(n)
06       else if Dominates(HM[n], HM[m]) = true then
07         HM[m].BossCount = HM[m].BossCount + 1
08       end if
09     end if
10   end for
11   if HM[m].BossCount = 0 then
12     Front0.Add(m)
13     HM[m].Front = 0
14   end if
15 end for
16 FrontNumber = 1
17 while Front0.Length() != 0 do
18   for each p in Front0 do
19     for each q in HM[p].Dominated do
20       HM[q].BossCount = HM[q].BossCount -1
21       if HM[q].BossCount = 0 then
22         NewFront.Add(q)
23         HM[q].Front = FrontNumber
24       end if
25     end for
26   end for
27   Front0 = NewFront
28   FrontNumber = FrontNumber + 1
29 end while
```

Figura 29 Procedimiento NonDominatedOrderCalculate (Adaptado desde [23])

La función Dominates se usa para calcular el ordenamiento de no-dominados, retorna verdadero o falso indicando cuando una armonía domina a otra considerando todos los objetivos. La Figura 30 presenta el pseudocódigo de esta función.

```
01 for t = 0 to NObjectives do
02   if HarmonyA.Evaluations[t] > HarmonyB.Evaluations[t] then
03     return false
04   end if
05   if HarmonyA.Evaluations[t] < HarmonyB.Evaluations[t] then
```

```
06      return true
07    end if
08  end for
09  return false
```

Figura 30 Función Dominates (Adaptado desde [8])

La función *CrowdingDistanceCalculate* toma la población existente y establece en cada solución la distancia de dispersión a las armonías ubicadas en su mismo frente de Pareto. Esta función requiere de una estructura auxiliar llamada "PositionValue" con dos atributos: un entero llamado "Position" y un decimal llamado "Value"; además requiere de un atributo llamado CrowdingDistance para cada armonía en donde almacena el valor individual de la distancia de dispersión, muy similar a lo propuesto en [23]. En la Figura 31 se presenta el pseudocódigo de esta función.

```
01  for each S in HM do
02    S.CrowdingDistance = 0
03  end for
04  HM.SortByFront()
05  lstGroups = new Array()
06  for position = 0 to HM.Length() do
07    lstGroups[HM[position].Front].Add(position)
08  end for
09  for each oGroup in lstGroups do
10    for obj = 0 to NObjectives do
11      lstPos = new Array<PositionValue>()
12      for each iPosition in oGroup do
13        lstPos.Add(new PositionValue(iPosition, HM[iPosition].Evaluations[obj]))
14      end for
15      lstPos.SortByValue()
16      HM[lstPos[0].position].CrowdingDistance = DoubleType.GetMaxValue()
17      HM[lstPos[lstPos.Length()-1].position].CrowdingDistance =
    DoubleType.GetMaxValue()
18      if AbsoluteValue(lstPos[0].value - lstPos[lstPos.Length() -1].value) >
    0.000001 then
19        for pos = 1 to (lstPos.Length() - 1) do
20          HM[lstPos[pos].position].CrowdingDistance =
    HM[lstPos[pos].position].CrowdingDistance + lstPos[pos + 1] - lstPos[pos - 1]
21        end for
22      end if
23    end for
24  end for
```

Figura 31 Procedimiento CrowdingDistanceCalculate (Adaptado desde [23])

Además de las funciones previamente ilustradas, MOGBHS utiliza las siguientes funciones:

- Sort: Ordena la memoria armónica por número de frente de Pareto (orden ascendente) y, en caso de que existan varios elementos con el mismo número de frente, estos se ordenan teniendo en cuenta la distancia de dispersión (orden descendente).
- SortByFront: ordena la memoria armónica teniendo en cuenta únicamente el número de frente de Pareto (orden ascendente).

- InPopulation: permite determina si una armonía ya existe dentro de la HM.
- IsViable: dependiendo de las reglas y restricciones del dominio donde se esté implementando MOGBHS, determina si una solución es viable o no.
- Add: adiciona un ítem a un arreglo o lista.
- Length: permite obtener la longitud actual de un arreglo o lista.
- RemoveTheWorst: elimina los peores elementos de la HM, que son los que se encuentren al final de la memoria después de aplicar la función Sort y que hagan que se exceda el tamaño definido por HMS.
- GetMaxValue: permite obtener el mayor valor de un tipo de dato, que para el caso del algoritmo representa un valor infinito.
- AbsoluteValue: permite obtener el valor absoluto de un número recibido como parámetro.

4.5 MOGBHS-TNDFSP

En esta sección se describe la adaptación realizada al algoritmo MOGBHS para resolver Transporte Network Desing and Frequency Settings Problem (MOGBHS-TNDFS), por tanto el algoritmo descrito en esta sección tiene como objetivo: encontrar los mejores conjuntos de rutas de buses y frecuencias de salida de cada una de las rutas seleccionadas como parte de cada solución para un sistema de transporte masivo de pasajeros con una estructura de estaciones y corredores viales previamente definidos.

Para la aplicación de MOGBHS al problema de TNDFSP es necesario, además de los parámetros propios del algoritmo, descritos en la sección inmediatamente anterior, los siguientes elementos:

- Archivos de código fuente SIMAN generados a partir del modelo de simulación creado en Arena que simboliza el STMP evaluado.
- Base de datos con el conjunto de posibles rutas válidas aplicables al STMP. Cada registro en la base de datos incluye como mínimo la siguiente información:
    - Estaciones incluidas en el recorrido de la ruta.
    - Definición de estaciones donde se detienen los buses para cada ruta.
    - Identificador único.

  Esta base de datos de rutas válidas posibles en el STMP puede ser construida utilizando cualquier enfoque cuyo resultado final garantice que las rutas cumplan con la información descrita previamente. En el presente proyecto se utilizaron Covering Arrays (CAs) [27, 28], el proceso de creación de la base de datos de rutas haciendo uso de CAs se describe en la sección 4.6.1 del presente documento. Una vez definido el total de posibles rutas para el STMP es necesario asignar un identificador entero único y consecutivo a todas ellas, para posteriormente incluir un par de parámetros a MOGBHS que determinen el mínimo y máximo identificador de las rutas en la base de datos para así facilitar su selección al generar las armonías.

Parámetros mínimo y máximo para definir la cantidad de rutas que puede tener una armonía generada por MOGBHS.

Considerando: 1) que la cantidad de rutas viables a implementar en los STMP, la cantidad de posibles combinaciones de configuracion de rutas y la cantidad de posibilidades para la configuracion de frecuencias para cada una de dichas rutas determinan un enorme espacio de búsqueda, y 2) que en el estado del arte se han encontrado casos exitosos donde se utilizan algoritmos multi-nivel en donde cada nivel se encarga de resolver un problema particular del conjunto de TNDSP, se plantea una implementacion de dos niveles del algoritmo propuesto para resolver el problema de configuracion de rutas y frecuencias para un STMP. MOGBHS-TNDFSP que corresponde al algoritmo del nivel exterior se encarga de la busqueda de los mejores conjuntos de rutas, y se apoya en MOGBHS-TNFSP que constituye el algoritmo de nivel interior para encontrar las mejores frecuencias de salida de buses para las rutas seleccionadas y para realizar la evaluación del fitness de la solucion respecto a los objetivos a optimizar (MOGBHS -TNFSTP se describe en la sección 4.5.3 de este documento).

Una armonía generada por MOGBHS - TNDFSP contiene como primer elemento la cantidad de rutas que conforman la solución, seguido del listado de identificadores de las rutas seleccionadas y finalmente las frecuencias de salida de buses para cada una de las rutas. La estructura de una armonía se presenta en la Figura 32, donde N corresponde a la cantidad de rutas que componen la armonía, R_1 a R_N corresponden a los identificadores de las rutas seleccionadas y F_1 a F_N son las frecuencias de salida de buses para las correspondientes rutas.

0	1	2	…	N – 1	N	N+1	N+2	…	2N-1	2N
N	R_1	R_2	…	R_{N-1}	R_N	F_1	F_2	…	F_{N-1}	F_N

Figura 32 Representación de solución de MOGBHS – TNDFSP (fuente propia)

Como se puede deducir la longitud de cada armonía depende de la cantidad de rutas que ella incluye, y corresponde a 2N + 1, siendo N la cantidad de rutas.

El proceso de construcción de cada armonía involucra dos fases, ambas utilizando la lógica de improvisación de GBHS: 1) selección de cantidad de rutas que compondrán la armonía, y 2) selección de las rutas desde la base de datos teniendo en cuenta la cantidad previamente seleccionada. La generación de las frecuencias para las rutas escogidas y la evaluación de fitness de la solución respecto a los objetivos a optimizar se realizan mediante una sub-ejecución de MOGBHS-TNFSP.

MOGBHS - TNDFSP genera una memoria armónica inicial y la ordena por frente de Pareto y distancia de dispersión, posteriormente realiza un número predeterminado de iteraciones en cada una de las cuales se genera una nueva armonía y se adiciona a la memoria armónica existente si se considera viable teniendo en cuenta las restricciones de diseño (entre otras, la flota disponible). Al final de cada iteración, si se ha encontrado una solución viable y se ha adicionado a la memoria armónica, se ordena la memoria armónica existente por frente de Pareto y distancia de dispersión para posteriormente remover la peor armonía, la cual corresponde a aquella que tenga mayor número de frente de Pareto y menor distancia de dispersión. A continuación, en la Figura 33, se presenta el pseudocódigo de MOGBHS para TNDFSP.

```
01 HM.PopulationRandomInitialize()
02 HM.NonDominatedOrderCalculate()
03 HM.CrowdingDistanceCalculate()
04 HM.Sort()
05 for I = 1 to NIter do
06   NewHarmony = New Harmony()
07   if Random(0,1) < HMCR then
08       NewHarmony.Variables[0] = HM.Harmonies[Random(0,HMS)].Variables[0]
09       PAR = PARMin + (((PARMax - PARMin) / NIterations) * I)
10       if Random(0,1) < PAR then
11        NewHarmony.Variables[0] = HM.FromBestHarmony(0)
12       end if
13   else
14     NewHarmony.Variables[0] = Random(MinRoutesNumber,MaxRoutesNumber)
15   end if
16   for J = 1 to NewHarmony.Variables[0] do
17     if Random(0,1) < HMCR then
18       NewHarmony.Variables[J] = HM.Harmonies[Random(0,HMS)].Variables[J]
19       PAR = PARMin + (((PARMax - PARMin) / NIterations) * I)
20       if Random(0,1) < PAR then
21        NewHarmony.Variables[J] = HM.FromBestHarmony(Random(0, NVariables))
22       end if
23     else
24       NewHarmony.Variables[J] = Random(MinRouteId, MaxRouteId)
25     end if
26   end for
27   NewHarmony.SolutionComplete()
28   if HM.InPopulation(NewHarmony) = false AND NewHarmony.IsViable() = true then
29     NewHarmony.Evaluate()
30     HM.Add(NewHarmony)
31     HM.NonDominatedOrderCalculate()
32     HM.CrowdingDistanceCalculate()
33     HM.Sort()
34     HM.RemoveTheWorst()
35   end if
36 end for
```

Figura 33 Pseudo-código MOGBHS adaptado para solucionar TNDFSP (fuente propia)

Una vez generada una nueva armonía, ya sea en la inicialización de la memoria armónica o durante las improvisaciones del algoritmo, es necesario validar la viabilidad de la misma. Una armonía se considera viable cuando la configuración de rutas seleccionadas aplicada al STMP permite a un usuario de dicho sistema ir desde cualquier estación hasta cualquier otra incluso haciendo uno o más trasbordos para ello. En caso de que una armonía no sea viable se le aplica un proceso denominado *Completar Solución* que consiste en:

1. Identificar las estaciones aisladas o que tienen mayores problemas de conexión con la configuración actual de las rutas definida en la armonía.
2. Identificar la ruta “fácil” (ruta que se detiene en todas las estaciones de su recorrido), que cubre la mayor cantidad de estaciones identificadas en el punto 1.

3. Incluir la ruta identificada en la armonía y evaluar si con la nueva ruta cumple la condición de validez (que se pueda alcanzar cualquier estación desde cualquier otra incluso haciendo uno o más trasbordos), en caso de no ser válida se repite desde el punto 1 de lo contrario se termina el proceso.

En la Figura 34 se presenta el pseudocódigo de la función *SolutionComplete* utilizada para aplicar el proceso descrito previamente y posteriormente en la Figura 35, se presenta el pseudocódigo de la función *IsViable* la cual es una auxiliar para *SolutionComplete* y se encarga de retornar un listado con los identificadores de las estaciones problema.

```
01 lstEasyRoutes = DBManager.LoadEasyRoutes()
02 lstTroubleStations = Harmony.IsViable()
03 While lstTroubleStations.Count > 0 Then
04   lstCountStations = New Dictionary()
05   for I = 0 to lstEasyRoutes.Count – 1 do
06      lstCountStations.Add(lstEasyRoutes[I].ID, 0)
07      for J = 0 to lstTroubleStations.Count – 1 do
08         if lstEasyRoutes[I].Stations.Contains(lstTroubleStations[J]) then
09            lstCountStations[lstEasyRoutes[I].ID]++
10         end if
11      end for
12   end for
13   lstCountStations.SortByValue()
14   Harmony.Routes.Add(lstCountStations.first().ID)
15   lstTroubleStations = Harmony.IsViable()
16 end while
```

Figura 34 Pseudo-código de la función SolutionComplete (fuente propia)

```
01  lstTroubleStations = New Dictionary()
02  for I = 0 to NStations -1 do
03     lstDestines = FindDestinesFrom(I)
04     for J = 0 to NStations – 1 do
05       if I <> J and Not lstDestines.Contains(J) then
06          if lstTroubleStations.ContainsKey(J) then
07             lstTroubleStations.Add(J, 1)
08          else
09             lstTroubleStations[J] ++
10          end if
11       end if
12     end for
13  end for
14  lstTroubleStations.SortByValueDesc()
15  return lstTroubleStations.getFirstKeys()
```

Figura 35 Pseudo-código de función IsViable (fuente propia)

En el modelo de simulación propuesto, como en los STMP reales, el comportamiento de los pasajeros depende de las rutas disponibles, el usuario usualmente tiende a utilizar los caminos mas cortos (en tiempo) para llegar a su destino, por lo anterior y teniendo en cuenta que cada armonía define un conjunto diferente de rutas disponibles, se hace necesario actualizar la variable RutasCortas en el modelo de simulación para que el comportamiento de las entidades tipo Pasajero sea consecuente con las rutas disponibles. Este calculo de rutas cortas para cada una de las armonías generadas en el transcurso de la ejecución de MOGBHS - TNDFSP se

realiza mediante una variacion del algoritmo de Dijkstra [60]. Una vez generada la matriz de rutas cortas, se actualizan los archivos fuentes del modelo de simulación. El detalle de la generación de rutas cortas mediante Dijkstra se describe en la sección 4.5.2 de este documento.

La función *Evaluate* de MOGBHS - TNDFSP se encarga de:

- Invocar el cálculo de rutas cortas entre todas las estaciones del sistema teniendo en cuenta el conjunto de rutas seleccionadas para conformar la armonía que se está evaluando.
- Insertar en los archivos de código fuente SIMAN la configuración de rutas y rutas cortas correspondientes a la solución que se está evaluando.
- Invocar la ejecución de MOGBHS para encontrar las mejores combinaciones de intervalos de salida de buses para el modelo de simulación configurado con las rutas y rutas cortas correspondientes a la armonía que se está evaluando. Esta adaptación de MOGBHS para Transit Network Frequency Settings Problem (TNFSP) se describe en la sección 4.5.3 de este documento.

A continuación, en Figura 36, se presenta el pseudocódigo de la función *Evaluate* de MOGBHS para TNDFSP.

```
01 RoutesDescription = Array[Harmony.Variables[0]][ConstTotalStations]
02 for k = 1 to Harmony.Variables[0] do
03     RoutesDescription[k-1] = DBManager.ExtractRouteString(Harmony.Variables[k])
04 end for
05 ReplaceInFile(“Transporte.exp”, “$CantidadRutas”, Harmony.Variables[0])
06 ReplaceInFile(“Transporte.exp”, “$DetalleRutas”, RoutesDescription.ToString())
07 ShortRoutes = DijkstraShortRoutes()
08 ReplaceInFile(“Transporte.exp”, “$RutasCortas”, ShortRoutes)
09 BestHarmony = MOGBHSforTNFSP.Execute()
10 Harmony.Intervals = BestHarmony.Intervals
11 Harmony.Evaluations = BestHarmony.Evaluations
```

Figura 36 Pseudo-código función Evaluate de MOGBHS para TNDFSP (fuente propia)

La función *ReplaceInFile* recibe 3 parámetros: el nombre del archivo sobre el cual se debe realizar la búsqueda, el texto buscado que se reemplaza y el nuevo valor a establecer en el archivo de código SIMAN.

Al final de la ejecución de MOGBHS para TNDFSP se cuenta con un conjunto de armonías que representan las mejores configuraciones de rutas e intervalos para el STMP evaluado para que el usuario final seleccione a su criterio la que se implementará en el sistema real.

4.5.1 Covering Arrays para crear base de rutas

Dada la necesidad de generar un conjunto con todas las posibles rutas viables para el STMP evaluado, siendo este un problema combinatorial altamente complejo (NP-hard) y conociendo las capacidades de los Covering Arrays (CAs) para la resolución de este tipo de problemas, se utilizaron estas estructuras para generar un conjunto representativo de las rutas viables para el STMP. Cabe anotar que esta base de datos de rutas puede generarse con cualquier otro mecanismo, no necesariamente con CAs.

La idea consiste en identificar los posibles recorridos en el sistema dada la topografía del mismo y para cada uno de ellos, generar un covering array con fuerza 3, alfabeto 2 (0 y 1, donde 0 indica que no se detiene en una estación y 1 a que se detiene en la estación), numero de columnas igual a la cantidad de estaciones y la menor cantidad de líneas posible. Debido a que previo a este trabajo se han realizado investigaciones para encontrar CAs con las características que mencionamos, el trabajo es relativamente sencillo [27, 28].

Se define 3 como fuerza de los CAs utilizados porque luego de un análisis de los recorridos de pasajeros en un STMP se encontró que la gran mayoría de los usuarios hacen un máximo de 3 paradas para llegar a su destino (incluyendo inicio y fin). Se considera apropiado como trabajo futuro realizar una evaluación más detallada de este parámetro, usando fuerzas 4, 5 y 6.

Cada una de las líneas de los CA representa una ruta aplicable a un recorrido específico, para cada una de ellas se debe realizar una validación de viabilidad en donde se descartan aquellas rutas que incluyan menos de dos estaciones físicas en donde el bus se detiene. Al final del proceso de generación y depuración se tiene una base de datos de rutas para el STMP conformada por la unión de todas las líneas de los CAs.

Finalmente, para facilitar el proceso de *Completar Solución* previamente descrito, se incluye en la base de rutas una ruta "fácil" por cada recorrido posible. Entendiéndose por ruta "fácil" aquella en la cual el vehículo se detiene en todas las estaciones que conforman el recorrido de la ruta.

Por ejemplo, teniendo un recorrido que contiene un total de 6 estaciones a su paso es necesario implementarlas con un CA de 6 columnas, alfabeto 2 y fuerza 3, que tendrá un total de 12 filas. Haciendo la correspondencia entre filas – rutas y columnas – estaciones, la configuración inicial de posibles rutas para el recorrido es:

Rutas\Estaciones	E1	E2	E3	E4	E5	E6
1	1	1	0	1	1	0
2	0	0	0	1	1	1
3	0	1	1	1	0	1
4	1	0	1	0	1	0
5	1	1	1	0	0	1
6	0	0	1	1	0	0
7	0	0	0	0	0	0
8	1	0	1	1	1	1
9	0	1	0	0	1	1
10	1	1	0	1	0	0
11	1	0	0	0	0	1
12	0	1	1	0	1	0

Figura 37 CA (12; 6, 2, 3) aplicado a la generación de rutas por recorrido (fuente propia)

Dado que no es lógico que un bus haga un recorrido sin detenerse en ninguna estación, la ruta número siete (7) no es viable y se debe remover del listado, en cambio se debe incluir una ruta fácil para el recorrido (que incluiría paradas en todas las estaciones).

4.5.2 Dijsktra para rutas cortas

Teniendo en cuenta que el comportamiento de los usuarios dentro de un STMP depende de las rutas disponibles debido a que los pasajeros siempre tienden a encontrar la ruta más corta hacia su destino, y que cada armonía generada por MOGBHS - TNDFSP representa una nueva configuración de rutas, es necesario que se recalcule el mapa de rutas cortas (que los usuarios emplean para calcular su recorrido y movimientos en el sistema) cada vez que se genera o modifica una armonía. Debido a la simplicidad de implementación y facilidad para hacer seguimiento durante su proceso de búsqueda, se seleccionó al algoritmo de Dijkstra como algoritmo base para implementar la búsqueda de rutas cortas.

Para el proceso de búsqueda de rutas cortas se necesita como insumo los archivos fuente en SIMAN que representan el modelo de simulación creado en Arena y que incluyen la configuración de rutas dada por una armonía. Desde estos archivos se toma información de tiempos de desplazamiento y disponibilidad de rutas y recorridos desde cada una de las estaciones.

Para cada una de las estaciones en el sistema se ejecuta una búsqueda de caminos cortos hacia todas las demás estaciones del sistema utilizando la lógica del algoritmo Dijkstra, que consiste en ir explorando todos los caminos a partir de un origen (en este caso la estación seleccionada), e ir marcando las demás estaciones del sistema con los caminos mínimos encontrados para llegar a cada una de ellas. Además de los costos de desplazamiento entre estaciones se tiene en cuenta el costo adicional que supone hacer el trasbordo entre rutas. Posterior a la búsqueda de los caminos más cortos a partir de un origen se realiza la actualización de una matriz de rutas temporal.

Finalmente cuando se tiene la matriz de rutas cortas completa, se genera el código SIMAN que la representa para luego actualizar los archivos de simulación. Esto permite que en las ejecuciones de la simulación las entidades tipo Pasajero tengan en cuenta el mapa de rutas cortas que corresponde a la configuración de rutas actual.

4.5.3 MOGBHS - TNFSP

Una vez el algoritmo MOGBHS - TNDFSP genere una armonía con una configuración de rutas para el STMP, se actualicen los archivos de código SIMAN con la descripción de las rutas seleccionadas, y se calcule la matriz de rutas cortas, entra en juego una adaptación del MOGBHS que se encarga de encontrar la mejor configuración de frecuencias de salida de buses para las rutas seleccionadas en la armonía que se está evaluando, a esta adaptación, por el objetivo que persigue se le denomina MOGBHS - TNFSP (por las siglas de Transport Network Frecuency Settings Problem).

Para MOGBHS - TNFSP una armonía se define como un conjunto de N intervalos de salida, donde N corresponde al número de rutas seleccionadas en la armonía de MOGBHS - TNDFSP. En la Figura 38 se representa una armonía del algoritmo MOGBHS-TNFSP.

0	1	2	…	N – 2	N-1
I_0	I_1	I_2	…	I_{N-2}	I_{N-1}

Figura 38 Solución para TNFSP generada por MOGBHS (fuente propia)

A diferencia del MOGBHS, en esta versión la generación aleatoria de valores para las variables que componen las armonías (tanto para la generación de nuevas armonías en el ciclo principal del algoritmo como en la generación de población inicial), se hace entre dos límites dados por el dominio del problema, en este caso el mínimo intervalo entre buses de una misma ruta y el máximo intervalo entre buses que cumplen el mismo recorrido.

Haciendo una correspondencia con el algoritmo MOGBHS (definido en la sección 4.4 de este documento), el valor para el parámetro NVariables de MOGBHS corresponde al número de rutas definidas en la armonía, de manera similar, una armonía de MOGBHS equivale a una configuración de frecuencias de salida de buses para el STMP es decir un arreglo de tamaño NVariables. A continuación, en la Figura 39, se presenta el pseudo-código de MOGBHS-TNFSP.

```
01 HM.PopulationRandomInitialize()
02 HM.NonDominatedOrderCalculate()
03 HM.CrowdingDistanceCalculate()
04 HM.Sort()
05 for I = 1 to NIterations do
06   NewHarmony = New Harmony()
07   for J = 1 to NVariables do
08     if Random(0,1) < HMCR then
09       NewHarmony.Variables[J] = HM.Harmonies[Random(0,HMS)].Variables[J]
10       PAR = PARMin + (((PARMax - PARMin) / NIterations) * I)
11       if Random(0,1) < PAR then
12        NewHarmony.Variables[J] = HM.FromBestHarmony(Random(0, NVariables))
13       end if
14     else
15       NewHarmony.Variables[J] = Random(MinIntervalTime, MaxIntervalTime)
16     end if
17   end for
18   if HM.InPopulation(NewHarmony) = false AND NewHarmony.IsViable() then
19     NewHarmony.Evaluate()
20     HM.Add(NewHarmony)
21     HM.NonDominatedOrderCalculate()
22     HM.CrowdingDistanceCalculate()
23     HM.Sort()
24     HM.RemoveTheWorst()
25   end if
26 end for
```

Figura 39 Pseudo-código de MOGBHS – TNFSP (fuente propia)

La lógica de la función *Evaluate* de MOGBHS - TNFSP se ilustra en la Figura 40.

```
01 IntervalString = “”
02 for k = 0 to NVariables do
03     IntervalString = IntervalString + “,” + Harmony.Variables[k]
04 end for
05 ReplaceInFile(“Transport.mod”, “$intervalos”, IntervalString)
06 ExecuteSimulation()
07 ReadFromOut(Harmony.Evaluations)
```

Figura 40 Función Evaluate para MOGBHS – TNFSP (fuente propia)

La función *ExecuteSimulation* se encarga de ejecutar la simulación haciendo uso del motor SIMAN. Finalmente la función *ReadFromOut* recorre el archivo que contiene los resultados de la simulación para extraer los valores que corresponden a la evaluación de los objetivos seleccionados, para el caso que nos concierne: el promedio de capacidad desperdiciada y el promedio de tiempo total de uso del sistema por parte de los pasajeros.

Para hacer que MOGBHS - TNFSP pueda ser utilizado por MOGBHS - TNDFSP para encontrar las mejores configuraciones de frecuencias, es necesario que el primero retorne solo una mejor solución, sin embargo dada la naturaleza multi-objetivo del algoritmo, y como ya se había mencionado antes, el resultado de la ejecución del mismo es un conjunto de mejores armonías seleccionadas con base en el rendimiento respecto a dos objetivos de optimización, por tanto fue necesario incluir una función automática (sin intervención humana) de selección de la mejor solución basada en la distancia euclidiana, esto implica los siguientes tres (3) pasos:

1. Teniendo un conjunto de armonías generadas por MOGBHS - TNFSP y evaluadas frente a los objetivos a minimizar, se identifican las soluciones que corresponden al primer frente de Pareto como se muestra en la Figura 41.

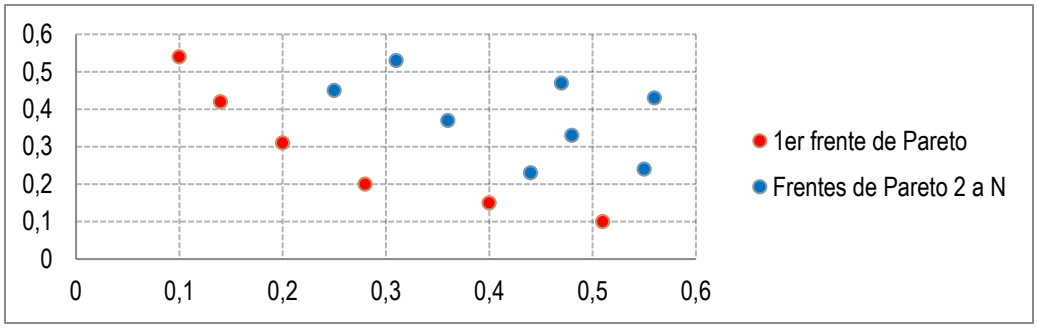

Figura 41 Primer frente de Pareto en problema de minimización de dos objetivos (fuente propia)

2. Se calcula la distancia euclidiana de cada una de las soluciones que conforman el primer frente de Pareto hasta el origen, en la Figura 42 se representa el cálculo de dicha distancia para una de las soluciones del primer frente de Pareto

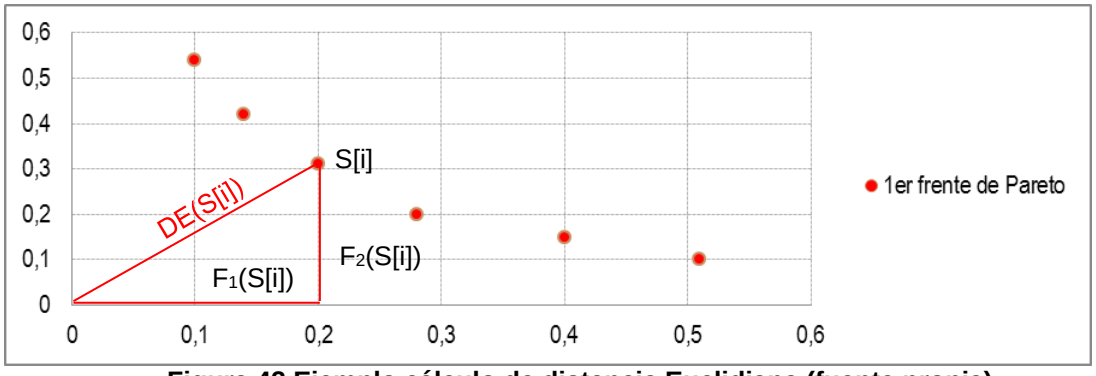

Figura 42 Ejemplo cálculo de distancia Euclidiana (fuente propia)

3. Se selecciona la solución con la menor distancia euclidiana al eje, esto es, la solución que tenga mejor rendimiento en todos los objetivos a minimizar.

Con la adición de esta función de selección de la mejor solución, el pseudo-código de MOGBHS - TNFSP queda como se muestra en la Figura 43.

```
01 HM.PopulationRandomInitialize()
02 HM.NonDominatedOrderCalculate()
03 HM.CrowdingDistanceCalculate()
04 HM.Sort()
05 for I = 1 to NIterations do
06   NewHarmony = New Harmony()
07    for J = 1 to NVariables do
08      if Random(0,1) < HMCR then
09        NewHarmony.Variables[J] = HM.Harmonies[Random(0,HMS)].Variables[J]
10        PAR = PARMin + (((PARMax - PARMin) / NIterations) * I)
11        if Random(0,1) < PAR then
12         NewHarmony.Variables[J] = HM.FromBestHarmony(Random(0, NVariables))
13       end if
14      else
15        NewHarmony.Variables[J] = Random(MinIntervalTime, MaxIntervalTime)
16      end if
17    end for
18    if HM.InPopulation(NewHarmony) = false AND NewHarmony.IsViable() then
19      NewHarmony.Evaluate()
20      HM.Add(NewHarmony)
21      HM.NonDominatedOrderCalculate()
22      HM.CrowdingDistanceCalculate()
23      HM.Sort()
24      HM.RemoveTheWorst()
25    end if
26 end for
27 for J = 1 to HMS do
28   if HM[J].Front == 0 then
29      HM[J].euclidianDistance = CalculateEuclideanDistance(HM[J].Evaluations)
30   else
31      HM[J].euclidianDistance = Float.maxNumber()
32   end if
33 end for
34 SortByEuclideanDistance(HM)
35 Return HM[1]
```

Figura 43 Pseudo-código de MOGBHS – TNFSP con selección por distancia euclidiana (fuente propia)

Como se describió a lo largo de este capítulo, en este proyecto de investigación se generó un nuevo algoritmo denominado Multi-Objective Global Best Harmony Search (MOGBHS) y se realizó una adaptación del mismo para resolver el problema de diseño y programación de frecuencias de rutas de buses (TNDFSP), para un sistema de transporte masivo de pasajeros (STMP), el cual hace uso de simulación de eventos discretos para la evaluación de los objetivos que se buscan optimizar, Covering Arrays para generar una base de posibles rutas en el STMP, una adaptación del algoritmo de Dijkstra para ajustar el comportamiento del modelo de simulación de acuerdo a las rutas seleccionadas en las armonías de MOGBHS - TNDFSP, y una segunda adaptación de MOGBHS para encontrar las frecuencias de salidas de buses (MOGBHS - TNFSP). En la siguiente sección se describe el proceso de experimentación y los resultados del algoritmo propuesto.

5 Experimentación

Para la experimentación se creó un modelo de simulación de eventos discretos de un STMP existente, se implementó un prototipo software que incluye el algoritmo propuesto, se diseñó y ejecuto una prueba de concepto, se realizó un proceso de calibración de parámetros del algoritmo, se ejecutó un conjunto de pruebas con los mejores parámetros encontrados, y finalmente se compararon resultados de MOGBHS contra los resultados de uno de los mejores algoritmos para el campo de la optimización multi-objetivo encontrados en la literatura (NSGA – II).

5.1 Selección de STMP para aplicar MOGBHS

Para realizar la selección de un Sistema de Transporte Masivo de Pasajeros existente que serviría como base para las pruebas de MOGBHS se tuvieron en cuenta dos aspectos:

- Tamaño del sistema: cantidad de estaciones, recorridos y rutas existentes. Este aspecto se tuvo en cuenta por la naturaleza del proyecto y los límites tanto de tiempo como de mano de obra para el desarrollo.
- Disponibilidad de información: facilidad de acceso a descripción de recorridos del sistema, información de rutas, flota, capacidad, frecuencias de buses, afluencia de pasajeros, plan de operaciones, etc.

Se realizó una evaluación entre los STMP existentes en Colombia:

- TransMilenio de Bogotá [61]
- Transmetro de Barranquilla [62]
- Transcaribe de Cartagena de Indias [63]
- MIO de Cali [64]
- Megabús de Pereira [65]
- Metrolínea de Bucaramanga [66]
- Metroplus de Medellín [67]

Por su tamaño y alta complejidad fueron descartados los STMP de las ciudades principales, es decir: Bogotá, Medellín, Cali, Buraramanga y Barranquilla. Luego de realizar gestiones con Megabús S.A. se logró adelantar un convenio entre Unicauca y Megabús S.A. para el suministro de la información disponible para realizar la implementación del modelo de simulación necesario para las pruebas de MOGBHS, por lo cual se seleccionó este STMP como base para generar el modelo de simulación y la realización de pruebas.

Megabús de Pereira cuenta con 37 estaciones, de las cuales 16 son dobles (en donde se puede hacer intercambio entre buses que van en sentidos opuestos), 18 son estaciones sin intercambio y 3 intercambiadores (estaciones donde se hace intercambio de pasajeros con otros sistemas de transporte). El sistema de troncales alcanza una longitud aproximada de 27 kms. Se tienen 3 rutas cuya frecuencia de salida de buses varía entre los 4 y los 17 minutos

dependiendo de la hora del día y del día de la semana. Presta servicio desde las 5:30 hasta las 20:30 todos los días, y mueve cerca de 200.000 pasajeros diarios. En la Figura 44 se presenta un mapa de rutas y estaciones de Megabús.

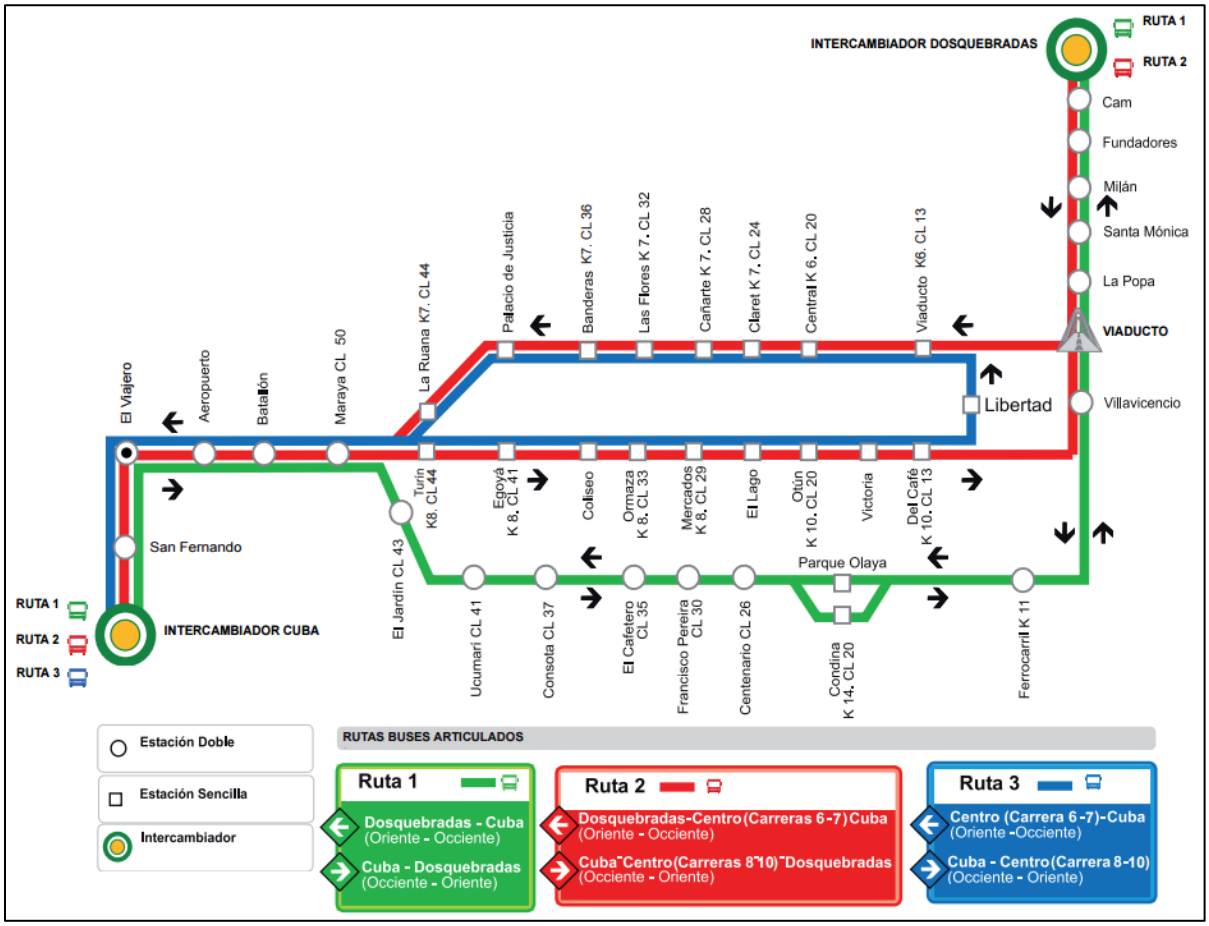

Figura 44 Mapa de rutas y estaciones Megabús – Pereira (Tomado de [68])

5.2 Modelo de simulación de STMP y base de datos de rutas

Posterior a la selección del STMP a tomar como base para el modelado se inició el proceso de implementación del modelo de simulación de eventos discretos sobre Arena [14, 45], teniendo en cuenta los lineamientos descritos previamente en la sección 4.2.1 de este documento.

Para el modelado y calibración del modelo de simulación se tomó la configuración de las horas valle de la mañana entre semana (días hábiles), es decir de 08:30 a 11:30. En este rango de tiempo la configuración del sistema está así:

Tabla 1 Configuración rutas Megabús horas valle – mañana – días hábiles. Fuente [69]

Ruta	Longitud (Km)	Tiempo Ciclo (min)	Flota	Frecuencia (min)
R1	22,72	68	8	8,5
R2	22,32	75	10	7,5
R3	14,74	51	6	8,5

En el marco del trabajo Megabús S.A. - Unicauca se obtuvo, entre otros, la siguiente información que se utilizó para la generación y calibración del modelo de simulación:

- Configuración de rutas incluyendo estaciones, distancias entre estaciones y tiempos de recorrido.
- Flota disponible incluyendo capacidad por bus y cantidad de buses.
- Promedio de ingreso diario de pasajeros por estación.

En principio se crearon los sub-modelos de estaciones de buses, pasajeros, recorridos de pasajeros, recorridos de buses, centrales de buses y cruces. Se enlazaron siguiendo la topografía del STMP real, y se realizaron las configuraciones de todos los módulos del sistema basadas en los lineamientos definidos previamente en este documento.

Teniendo la configuración de rutas del STMP se determinó el valor de la variable *Rutas* y se calcularon las rutas cortas entre todas las estaciones del sistema haciendo uso del algoritmo de Dijkstra modificado para facilitar la toma de decisiones de las entidades tipo usuario durante su recorrido en el sistema, y se incluyeron en la variable *RutasCortas* de la simulación. Para suplir la falta de información acerca de las tendencias de viaje de los pasajeros entre las diferentes estaciones, se definió una probabilidad uniforme para la selección de estaciones destino, la cual se estableció en la variable *ProbabilidadDestinos*. Se estableció ciento cuarenta (140) pasajeros como la capacidad máxima de transporte de un bus del sistema y se define que todos los buses tienen la misma capacidad. Además se definió un tiempo de espera de cada bus en las estaciones para embarque y desembarque de pasajeros de treinta (30) segundos (0,5 minutos). Asumir esta información implica dificultades en la aplicación de la solución, por ello, se hace necesario en un trabajo futuro no sólo tomar estos datos de la realidad sino también desarrollar un proceso formal de calibración del modelo.

El resultado de este proceso fue un modelo de simulación de eventos discretos implementado en Arena con un total de 37 estaciones (entre dobles, sin intercambio e intercambiadores), que permite medir el tiempo medio de uso del sistema por parte de los pasajeros y el porcentaje de capacidad desperdiciada de los buses dependiendo de la configuración de rutas y frecuencias de salida de las mismas. En la Figura 45 se observa una captura de pantalla del modelo de simulación implementado. Este proceso de implementación del modelo de simulación sobre Arena se realizó como parte del trabajo de experimentación y pruebas del proyecto de construcción de ModeLAB [47].

De la ejecución del modelo de simulación configurado con el plan de operaciones que utiliza el STMP real para horas valle de la mañana y días hábiles se obtuvieron los siguientes resultados:

- Promedio de tiempo de uso del sistema: 26.26 minutos
- Promedio de capacidad desperdiciada: 91.7 sillas

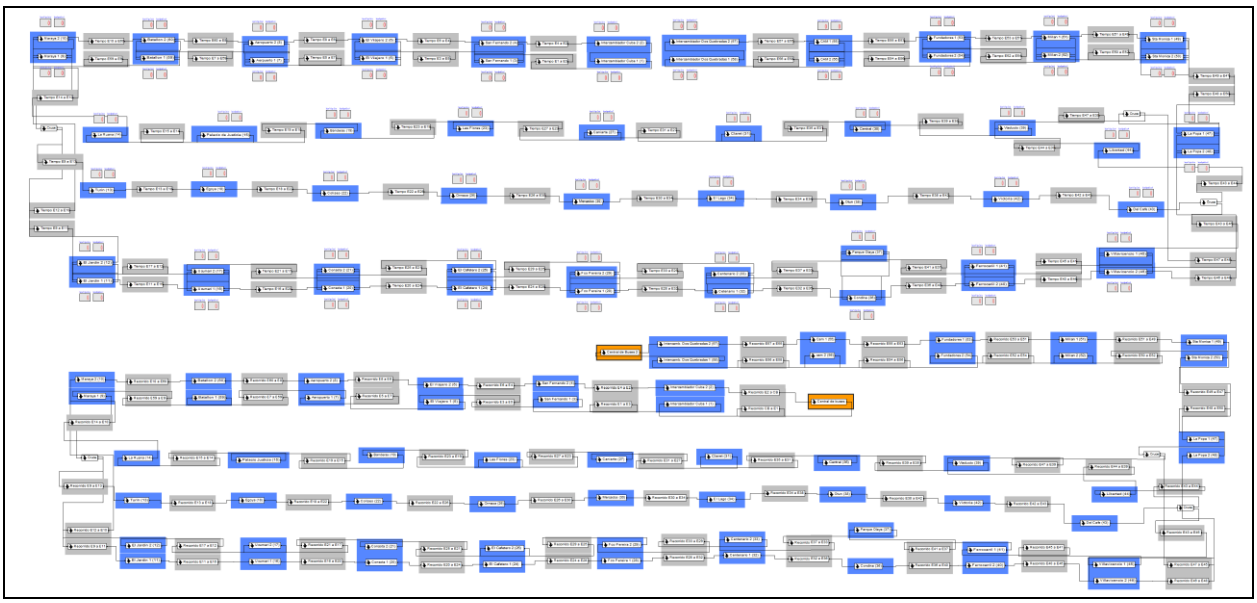

Figura 45 Modelo de simulación de Megabús implementado en Arena (fuente propia)

Finalizado el proceso de modelado, se generaron los archivos en código SIMAN y se realizaron las modificaciones descritas en la sección 4.2.2 de este documento para que sirvieran como insumo de MOGBHS.

Además de los archivos SIMAN, el algoritmo MOGBHS-TNDFSP requiere que se cree una base de datos de posibles rutas aplicables al STMP, para ello se utilizaron CAs de la siguiente forma:

- Se identificaron cinco (5) posibles recorridos dada la topografía del STMP.
- Teniendo en cuenta el número de estaciones de cada recorrido se seleccionó un CA para generar las posibles rutas para dichos recorridos, estos fueron:

  CA(24; 30, 2, 3)

  CA(26; 41, 2, 3)

  CA(26; 42, 2, 3)

  CA(26; 43, 2, 3)

  El CA de 42 columnas se utilizó en dos recorridos dado que coincidían en número de estaciones involucradas.
- Se hizo limpieza de las rutas que no fueran lógicas o inviables, entre otras aquellas con menos de dos estaciones reales en su recorrido.
- Se agruparon las rutas encontradas para cada recorrido y se insertaron en una base de datos, se obtuvieron un total de 119 rutas.
- Se adicionaron 5 rutas fáciles (una por recorrido), para facilitar el proceso de *Completar Solución* en caso de soluciones inválidas, para un total de 124 posibles rutas.

- Se establecieron identificadores únicos para el total de rutas en la base de datos y una marca para las rutas fáciles para facilitar su búsqueda en el proceso de completar solución.

Detalles del modelo de simulación, archivos SIMAN y las rutas identificadas se encuentran en los Anexos 1 y 2.

5.3 Prototipo MOGBHS

Para realizar las pruebas del algoritmo propuesto se creó un prototipo software con las siguientes características funcionales:

- Toma como insumo los archivos SIMAN resultado del proceso de modelado descrito en el punto anterior y la base de datos de posibles rutas generadas mediante CAs para el STMP en estudio
- Permite parametrizar y ejecutar MOGBHS - TNDFSP para encontrar las mejores configuraciones de rutas y frecuencias de buses para un STMP teniendo como resultado un archivo de tipo .CSV donde se describen las soluciones encontradas

El prototipo fue implementado en C#, con Visual Studio versión 2010 y .NET framework versión 4.0. A continuación, en la Figura 46 se presenta una imagen con la interfaz del prototipo software que permite configurar los parámetros y realizar la ejecución de MOGBHS - TNDFSP.

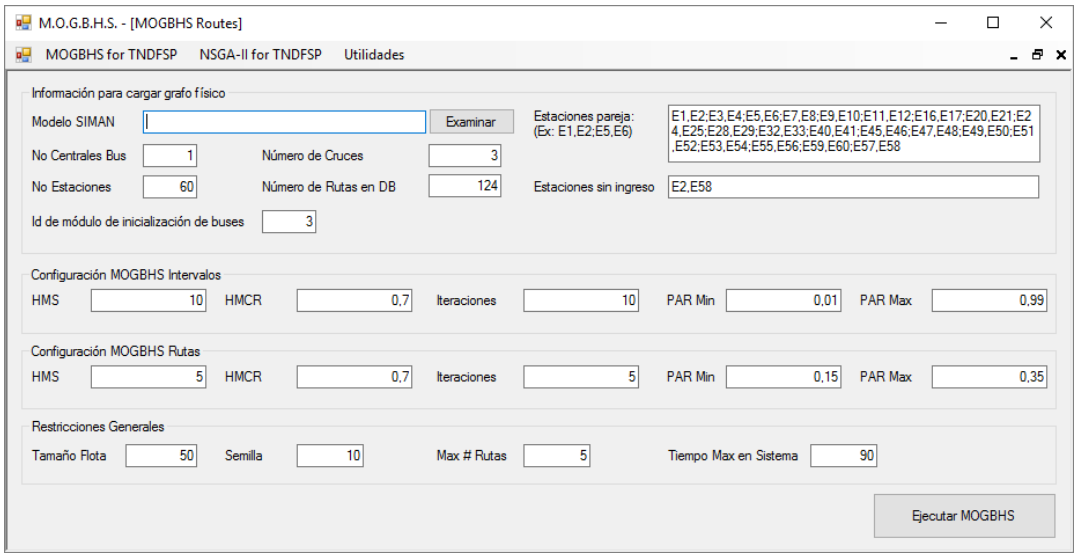

Figura 46 Captura de pantalla de la opción para ejecución de MOGBHS-TNDFSP del prototipo software (fuente propia)

Una vez realizada la ejecución del algoritmo con los parámetros establecidos en la interfaz del prototipo, se genera un archivo *.csv con la última memoria armónica del algoritmo, es decir las mejores armonías encontradas por el algoritmo, cada una de ellas incluye los identificadores de las rutas seleccionadas y las frecuencias de salida de buses.

El diseño técnico, detalles de la implementación del prototipo software y de la entrada y salida de datos se encuentran en el Anexo 3.

5.4 NSGA-II

Teniendo en cuenta que NSGA-II se reporta en el estado del arte como uno de los mejores algoritmos en el área de la optimización multi-objetivo y teniendo en cuenta su reducido número de parámetros (lo que significa mayor facilidad de afinamiento) [23, 24, 70], se realizó una implementación de dicho algoritmo para resolver el problema de TNDFSP de manera similar a como se implementó MOGBHS, es decir, atacando el problema de configuración de rutas y configuración de frecuencias en dos niveles diferentes, y haciendo uso del modelo de simulación de eventos discretos del STMP para la evaluación del fitness de las soluciones generadas. Lo anterior con el objetivo de comparar el rendimiento de los dos algoritmos.

5.5 Prueba de concepto

Con el fin de corroborar la viabilidad de utilizar el algoritmo propuesto para búsqueda de soluciones del grupo de problemas de TNDFSP, y la viabilidad de soportar la evaluación del fitness de las soluciones generadas por el algoritmo multi-objetivo propuesto sobre un modelo de simulación de eventos discretos y el uso de dichas evaluaciones como retroalimentación del algoritmo, se implementó una prueba de concepto que consiste en resolver únicamente el problema de configuración de frecuencias para un STMP (TNFSP) mediante MOGBHS y soportado sobre el modelo de simulación previamente construido. El algoritmo ejecutado corresponde a MOGBHS-TNFSP sin la selección por distancia euclidiana (descrito en el numeral 4.5.3 de este documento).

Dado que en esta prueba de concepto no se generaron configuraciones de rutas, se utilizaron las tres (3) rutas definidas actualmente en el sistema real de Megabús, se determinó además que la frecuencia para cada una de las rutas puede oscilar entre uno (1) y treinta (30) minutos, por tanto para el caso específico se tienen 27000 opciones de configuración de frecuencias.

Como primera acción de esta prueba se ejecutó una búsqueda exhaustiva de las configuraciones de frecuencias con mejores evaluaciones de fitness frente a los dos objetivos a minimizar, se encontraron un total de 99 soluciones en el primer frente de Pareto consideradas las mejores soluciones existentes para la minimización de ambos objetivos.

Posteriormente se realizaron 30 ejecuciones de MOGBHS-TNFSP con la parametrización recomendada en la literatura para la ejecución de GBHS:

- HMS: 100
- HMCR: 0.7
- PAR mínimo: 0.1
- PAR máximo: 0.9
- Número de iteraciones: 27000
- Número de variables: 3

En un tercer paso se realizaron 30 ejecuciones de NSGA-II adaptado para buscar soluciones de TNFSP con la siguiente parametrización tomada del estado del arte:

- Probabilidad de mutación: 1/3
- Probabilidad de cruce: 0,9
- Tamaño de población: 100

- Numero de variables: 3
- Número de iteraciones: 27000

Tanto en la búsqueda exhaustiva como en las búsquedas heurísticas se utilizó el modelo de simulación de eventos discretos como soporte para la evaluación del fitness frente a los dos objetivos evaluados.

Tras un análisis de las soluciones generadas por los algoritmos heurísticos se encontró que MOGBHS encontró más soluciones óptimas que NSGA-II luego de las 3000 evaluaciones de las funciones objetivo (EFO). La efectividad fue mejor desde un 175.12% en 3000 EFOs hasta un 488.68% con 27000 EFOs. En la Tabla 2 se presenta la comparación de efectividad de los dos algoritmos y en la Figura 47 se presenta dicha información de forma gráfica.

Tabla 2 Comparación de efectividad MOGBHS-TNFSP vs NSGAII-TNFSP (fuente propia)

	MOGBHS - TNFSP		NSGA-II - TNFSP		
EFO	Soluciones optimas encontradas (Promedio)	Efectividad (%)	Soluciones optimas encontradas (Promedio)	Efectividad (%)	Mejora de Efectividad (%)
3000	19,53	19,73	7,10	7,17	175,12
6000	33,93	34,28	8,73	8,82	288,55
9000	42,73	43,16	9,87	9,97	333,11
12000	50,40	50,91	10,80	10,91	366,67
15000	55,40	55,96	10,97	11,08	405,17
18000	59,60	60,20	10,80	10,91	451,85
21000	63,47	64,11	11,13	11,25	470,06
24000	66,30	66,97	11,60	11,72	471,55
27000	68,48	69,17	11,63	11,75	488,68

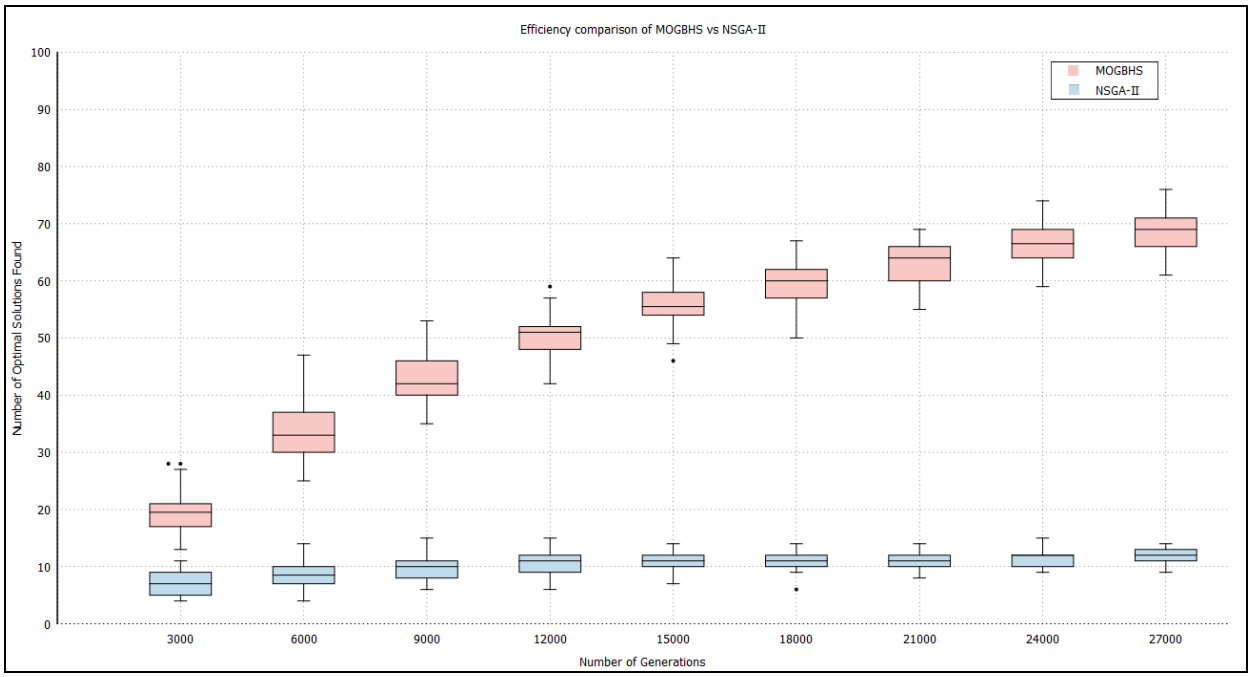

Figura 47 Comparación de efectividad MOGBHS-TNFSP vs NSGAII-TNFSP (fuente propia)

5.6 Calibración de parámetros

Previo a la ejecución de las pruebas se realizó un proceso de afinamiento para algunos de los parámetros del algoritmo. Teniendo en cuenta recomendaciones de configuración en trabajos previos sobre HS y GBHS se definieron valores fijos para:

- PAR Mínimo = 0,01
- Número de improvisaciones = 100

Y se definieron 27 combinaciones con valores recomendados para HMS, HMCR y PAR Máximo para hacer el afinamiento de parámetros, en la Tabla 3 se muestra la configuración de las pruebas para el afinamiento.

Tabla 3 Conjunto de pruebas para el afinamiento de parámetros (fuente propia)

No Prueba	HMS	HMCR	PARmax
1	10	0,7	0,3
2	10	0,7	0,35
3	10	0,7	0,4
4	10	0,8	0,3
5	10	0,8	0,35
6	10	0,8	0,4
7	10	0,9	0,3
8	10	0,9	0,35
9	10	0,9	0,4
10	15	0,7	0,3
11	15	0,7	0,35
12	15	0,7	0,4
13	15	0,8	0,3
14	15	0,8	0,35
15	15	0,8	0,4
16	15	0,9	0,3
17	15	0,9	0,35
18	15	0,9	0,4
19	20	0,7	0,3
20	20	0,7	0,35
21	20	0,7	0,4
22	20	0,8	0,3
23	20	0,8	0,35
24	20	0,8	0,4
25	20	0,9	0,3
26	20	0,9	0,35
27	20	0,9	0,4

Para la selección del conjunto de parámetros a utilizar en las pruebas comparativas del algoritmo se efectuó el siguiente procedimiento:

- Se marcaron todas las armonías con un identificador de Número de la Prueba que las había generado.
- Se unieron todas las armonías resultado de las ejecuciones para calibración.
- Se realizó un ordenamiento por frentes de Pareto y distancia de dispersión de todas las soluciones.
- Se eliminó del conjunto de soluciones todas aquellas que no pertenecieran a los tres (3) primeros frentes de Pareto.
- En el conjunto resultante se contaron las armonías existentes por número de prueba.

- Se seleccionó la prueba con mayor número de armonías en el conjunto de los tres (3) primeros frentes de Pareto.

A continuación se muestra la Tabla 4 con número de ocurrencias por prueba en el conjunto de armonías que conforman los primeros tres frentes de Pareto:

Tabla 4 Número de armonías ubicadas en los primeros frentes de Pareto por número de prueba (fuente propia)

No Prueba	No Armonías en Frentes 1 a 3	No Prueba	No Armonías en Frentes 1 a 3	No Prueba	No Armonías en Frentes 1 a 3
1	2	10	0	19	1
2	5	11	1	20	0
3	0	12	1	21	3
4	0	13	0	22	4
5	2	14	1	23	0
6	2	15	1	24	0
7	1	16	0	25	2
8	1	17	1	26	1
9	3	18	2	27	0

En consecuencia la configuración utilizada para la experimentación fue la prueba numero dos (2) con los siguientes parámetros:

- PAR Mínimo = 0,01
- PAR Máximo = 0,35
- HMCR = 0,7
- HMS = 10
- Número de improvisaciones = 100

El detalle del proceso de afinación de parámetros para MOGBHS se encuentra en el Anexo 4.

En un proceso de afinación de parámetros similar al realizado para MOGBHS se determinaron los parámetros de NSGA-II, la configuración seleccionada para la ejecución de dicho algoritmo incluye los siguientes parámetros:

- Probabilidad de mutación = 0,333
- Probabilidad de cruce = 0,7
- Tamaño población = 10
- Número de improvisaciones = 100

El detalle de la implementación de NSGA-II y del proceso de afinación de parámetros para dar soluciones a TNDFSP se encuentra en el Anexo 5.

5.7 Ejecución y resultados

Se realizaron treinta (30) ejecuciones tanto de MOGBHS como de NSGA-II sobre idénticos modelos de simulación, con los parámetros resultados de la afinación de cada uno de los algoritmos, y con los mismos parámetros de evaluación de rendimiento frente a los objetivos seleccionados, con el fin de encontrar las fortalezas y debilidades del algoritmo propuesto.

Para lograr la independencia de las diferentes ejecuciones de ambos algoritmos se utilizaron diferentes semillas para la función de aleatoriedad en cada ejecución.

Para lograr la normalización de los valores de "Tiempo de uso del sistema", para las pruebas se estableció un máximo valor de tiempo de uso de noventa (90) minutos y un valor mínimo de un (1) minuto en el sistema, tiempo suficiente para hacer dos recorridos completos (ida y regreso) sobre el recorrido de mayor longitud del sistema en una ruta fácil. Dado que la capacidad máxima de pasajeros por bus se estableció en ciento cuarenta (140), dicho valor corresponde al máximo valor de capacidad desperdiciada y cero (0) al valor mínimo.

5.7.1 Comparación de rendimiento frente a los objetivos a optimizar

Se agruparon las 30 memorias armónicas resultado de las ejecuciones de MOGBHS, cada una con 10 armonías y sus correspondientes evaluaciones de rendimiento frente a los objetivos seleccionados (capacidad de transporte desperdiciada y tiempo de uso del sistema). De otro lado se agruparon las 30 poblaciones resultado de las ejecuciones de NSGA-II, cada una con 10 soluciones y sus correspondientes evaluaciones de rendimiento. Posteriormente se generó el grafico presentado en la Figura 48 que muestra la distribución de las soluciones de ambos algoritmos frente a los objetivos evaluados.

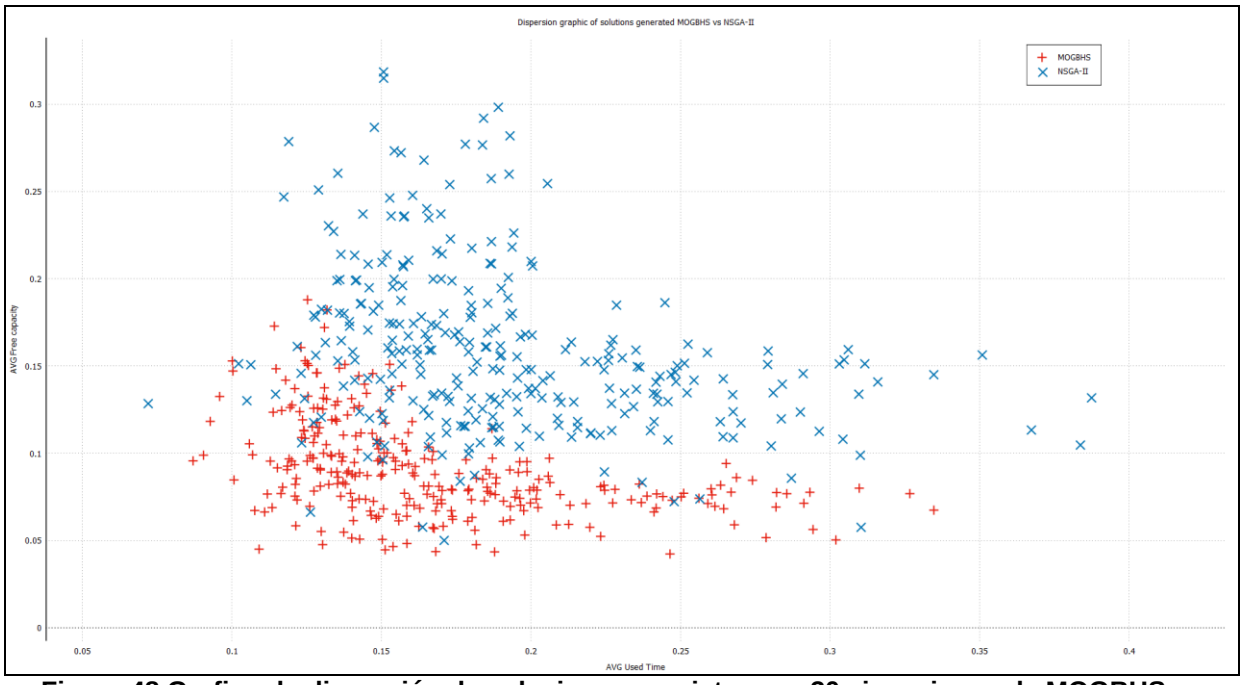

Figura 48 Grafico de dispersión de soluciones provistas por 30 ejecuciones de MOGBHS vs NSGA-II (fuente propia)

Se puede observar que las configuraciones provistas por el algoritmo propuesto en este proyecto (MOGBHS – TNDFSP) obtienen mejores resultados en la evaluación de rendimiento frente a los objetivos seleccionados. A continuación se presentan comparaciones de promedio y desviación estándar de las evaluaciones de las armonías/soluciones dadas por los algoritmos frente a los dos objetivos.

Tabla 5 Promedio de evaluación por objetivo de las armonías/soluciones de MOGBHS vs NSGA-II (fuente propia)

Algoritmo\ Objetivo	Tiempo medio de uso	Tiempo medio de uso (mins)	Capacidad media desperdiciada	Capacidad media desperdiciada (# cupos)
MOGBHS	0,165404028	15,89	0,0915333	12,81
NSGA-II	0,190890808	18,18	0,159106	22,27

En la Tabla 5 se puede apreciar la ventaja en la minimización de ambos objetivos que tiene MOGBHS frente a NSGA-II. A continuación, en la Tabla 6 se presenta la desviación estándar de las evaluaciones de las armonías dadas por MOGBHS y las soluciones generadas por NSGA-II, se aprecia que las ejecuciones de MOGBHS son más compactas o menos dispersas que las encontradas por NSGA-II, otro punto a favor del algoritmo propuesto.

Tabla 6 Desviación estándar de evaluación por objetivo de las armonías/soluciones dadas por MOGBHS vs NSGA-II (fuente propia)

Algoritmo\Objetivo	Tiempo medio de uso	Capacidad media desperdiciada
MOGBHS	0,047403769	0,027997591
NSGA-II	0,050960594	0,047636891

Comparando el tiempo medio de uso y la capacidad media desperdiciada de las armonías generadas por MOGBHS en las treinta (30) pruebas frente a las mismas medidas tomadas de la ejecución del modelo de simulación configurado con el plan de operaciones del STMP real para las horas valle de la mañana en días hábiles (numeral 5.2), se encuentra una mejora considerable tanto en la minimización del tiempo medio de uso como de la capacidad media desperdiciada. En la Tabla 7 se presenta la comparación de dichas medidas.

Tabla 7 Evaluación de objetivos en soluciones dadas por MOGBHS vs evaluación del modelo de simulación (fuente propia)

Fuente\Objetivo	Tiempo medio de uso (mins)	Capacidad media desperdiciada (# cupos)
MOGBHS	15,89	12,81
Modelo simulación	26,26	91,27

5.7.2 Mejores soluciones encontradas

Posteriormente se realizó un ordenamiento por frentes de Pareto tanto del grupo completo de armonías generadas por MOBGHS, como del grupo completo de soluciones generadas por NSGA-II. Con ello se obtuvieron las mejores soluciones de las 30 ejecuciones de ambos algoritmos, estas se presentan en las siguientes tablas:

Tabla 8 Mejores armonías generadas en 30 ejecuciones de MOGBHS (fuente propia)

Cantidad de rutas	Ids de rutas	Frecuencias (min)
5	64,60,95,110,23(f)	25,16,10,28,2
4	123(f),11,20,23(f)	27,18,3,2
4	100,2,103,23(f)	25,2,29,3
4	21,38,103,23(f)	20,11,9,2
4	92,64,108,23(f)	14,25,14,2
4	7,103,23(f),106	26,23,3,3
3	23(f),123(f),19	14,4,2
4	123(f),7,95,23(f)	24,16,4,3
Promedio: 4.0		**Promedio: 12.78**

Tabla 9 Mejores soluciones generadas en 30 ejecuciones de NSGA-II (fuente propia)

Cantidad de rutas	Ids de rutas	Frecuencias (min)
5	47, 8, 103, 52, 26	25, 14, 23, 19, 5
5	50, 0, 111, 113, 23(f)	25, 19, 16, 16, 6
6	114, 9, 80, 64, 81, 7	21, 18, 24, 16, 10, 8
6	101, 9, 80, 64, 81, 7	25, 15, 24, 27, 12, 5
4	53, 96, 109, 23 (f)	13, 27, 25, 4
Promedio: 5.2		**Promedio: 17.0**

En las anteriores tablas, los Ids de rutas corresponden a los identificadores únicos de las rutas en la base de datos de rutas (ver Anexo 2), se marcan las rutas fáciles con una (f), y las frecuencias corresponden a valores en minutos para cada una de las rutas.

Sobre las mejores rutas generadas por los algoritmos se puede apreciar que el promedio en la cantidad de rutas seleccionadas por MOGBHS fue de 4.0, en su lugar NSGA-II seleccionó un promedio de 5.2 rutas, se observa además que la frecuencia promedio configurada por MOGBHS es de 12.78 mins mientras que el mismo promedio en las soluciones de NSGA-II fue de 17.0 mins, lo que justifica en parte el incremento de tiempo de uso en las soluciones dadas por este último.

El detalle de las memorias armónicas resultados de las ejecuciones de MOGBHS y las poblaciones resultado de las ejecuciones de NSGA-II se presentan en los Anexos 6 y 7.

5.7.3 Comparación de tiempos de ejecución

Dado que cada ejecución del modelo de simulación puede tardar entre 1 y 5 segundos, los tiempos de ejecución de ambos algoritmos dependen en gran medida de la cantidad de ejecuciones de dicha simulación para la evaluación de las armonías/soluciones. Aunque ambos algoritmos se ejecutaron exactamente con el mismo tamaño de población/memoria, y realizan la misma cantidad de improvisos/generaciones tanto en el primer nivel (generación de rutas), como en el segundo nivel (generación de frecuencias), varían bastante en sus tiempos de ejecución porque solo las configuraciones con frecuencias que puedan ser cubiertas por la flota

disponible se evalúan mediante la ejecución de la simulación en caso contrario se les establecen valores de evaluación muy altos para que sean descartados por la lógica de los algoritmos.

En la Tabla 10 se presenta un consolidado de los tiempos de ejecución en minutos de las ejecuciones de ambos algoritmos.

Tabla 10 Consolidado de tiempos de ejecución en minutos de MOGBHS y NSGA-II (fuente propia)

Ejecución	MOGBHS	NSGA-II
1	1381	799
2	1405	797
3	1464	718
4	1235	761
5	1492	707
6	1366	988
7	1727	959
8	1689	966
9	1612	973
10	1504	923
11	1482	773
12	1446	801
13	1330	687
14	1382	843
15	1439	812

Ejecución	MOGBHS	NSGA-II
16	1545	762
17	1366	763
18	1310	757
19	1283	711
20	1499	743
21	1226	947
22	1763	987
23	1669	966
24	1774	970
25	1382	983
26	1281	780
27	1574	783
28	1371	788
29	1576	778
30	1081	810
Promedio	**1455**	**835**

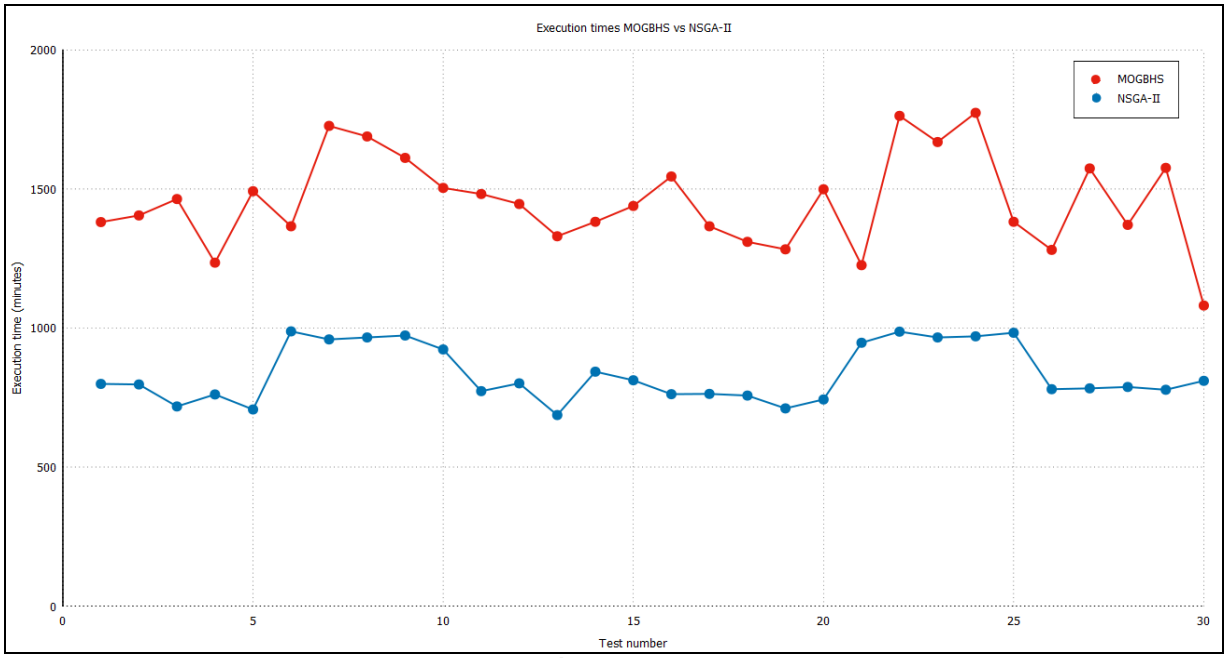

Figura 49 Tiempos de ejecución MOGBHS vs NSGA-II (fuente propia)

Tanto en la Tabla 10 de consolidados como en la Figura 49 de dispersión, se puede apreciar que el algoritmo propuesto tarda mucho más tiempo en ejecutarse que el algoritmo de referencia, sin embargo si se tiene en cuenta que, previa a la ejecución de la simulación, los algoritmos realizan una validación de viabilidad de la solución teniendo en cuenta la flota disponible, los tiempos totales de recorrido de las rutas seleccionadas y las frecuencias de las mismas, se puede concluir entonces que MOGBHS ejecuta más simulaciones que NSGA-II y por tanto genera más configuraciones que pueden ser aplicables con la flota disponible en el STMP.

6 Conclusiones, recomendaciones y trabajo futuro

6.1 Conclusiones

Con el objetivo de proveer un mecanismo que facilite la configuración de rutas y frecuencias en sistemas de transporte masivo de pasajeros (Transport Network Design and Frequencies Settings Problem – TNDFSP), en la presente tesis de maestría se propone una adaptación de la mejor búsqueda armónica global (Global best harmony search – GBHS), para convertirla en un algoritmo de optimización multi-objetivo haciendo uso del ordenamiento por frentes de Pareto y distancia de dispersión. El algoritmo propuesto llamado Multi-Objective Global Best Harmony Search (MOGBHS) genera una memoria armónica inicial aleatoria, evalúa el rendimiento de cada una de las armonías generadas respecto a los objetivos a optimizar, ordena dicha memoria por frentes de Pareto y distancia de dispersión para luego realizar un número determinado de iteraciones en donde, en cada iteración, se genera una nueva armonía la cual es evaluada frente a los objetivos a optimizar, esta nueva armonía se incluye en la actual memoria armónica, se ordena la memoria armónica resultante por frentes de Pareto y distancia de dispersión, y se elimina la peor solución basada en ese ordenamiento. Al final del proceso iterativo se obtiene el conjunto de las armonías que mejor rendimiento tienen respecto a los objetivos a optimizar. Con la intención de manejar el gran espacio de búsqueda y la complejidad que supone la representación de las soluciones para resolver TNDFSP, se realizó una implementación multi-nivel de MOGBHS: un nivel externo que se encarga del problema de selección de las mejores configuraciones de rutas (Transport Network Design Settings Problem - TNDSP), y un nivel interno encargado de seleccionar las mejores configuraciones de frecuencias para el conjunto de rutas seleccionadas por el primer nivel (Transport Network Frequency Settings Problem - TNFSP).

Para evitar los costos que supone la evaluación del rendimiento de las configuraciones de rutas y frecuencias generadas por MOGBHS sobre un sistema de transporte masivo de pasajeros real, en el presente proyecto de investigación se propone el uso de simulaciones de eventos discretos. Se seleccionó SIMAN como lenguaje de simulación y Arena de Rockwell Software como la herramienta para realizar la creación de los modelos de simulación de STMP y su posterior exportación a código SIMAN. Para garantizar tanto el correcto modelado de los elementos y comportamientos del STMP relevantes para la configuración de rutas y frecuencias como la interoperabilidad entre los modelos de simulación de STMP y MOGBHS se definieron un conjunto de lineamientos para la creación de dichos modelos sobre Arena. Estos lineamientos incluyen, entre otros, la definición de una matriz de rutas cortas utilizada para simular el hecho de que los pasajeros cambian su comportamiento en el sistema dependiendo de las rutas disponibles para su transporte. Durante la ejecución de MOGBHS, al momento de evaluar una armonía específica, se modifica el código fuente SIMAN para insertar las configuraciones de rutas y frecuencias definidas en la armonía, y la matriz de rutas cortas resultante de dicha configuración de rutas, posteriormente se ejecuta el modelo de simulación haciendo uso del motor SIMAN, y finalmente se extrae la información del resultado de dicha ejecución para ser utilizada en la evaluación de rendimiento de la armonía. En el prototipo presentado en este proyecto de investigación se utiliza una adaptación del algoritmo de Dijsktra para la generación de la matriz de rutas cortas sin embargo, dada la independencia de este componente del resto de elementos de MOGBHS, puede ser reemplazado por cualquier otro algoritmo que cumpla el mismo objetivo.

Con el ánimo de dar soporte a la evaluación de MOGBHS en el proceso de configuración de rutas y frecuencias, se implementó un modelo de simulación de eventos discretos para Megabús de la ciudad de Pereira, Colombia. Con el fin de obtener la información necesaria para realizar dicho modelado se realizó una interacción entre la Universidad del Cauca y la entidad encargada de la gestión de dicho sistema: Megabús S.A. El resultado fue un modelo de simulación de eventos discretos implementado en Arena con un total de 37 estaciones (entre dobles, sin intercambio e intercambiadores), que permite medir el tiempo medio de uso del sistema por parte de los pasajeros y el porcentaje de capacidad desperdiciada de los buses dependiendo de la configuración de rutas y frecuencias de salida de las mismas. Posterior a este proceso, dada la necesidad de MOGBHS-TNDFSP de tener un conjunto de rutas válidas aplicables al STMP, y aprovechando las bondades de los Covering Arrays en problemas combinatoriales, se utilizó un conjunto de estos para definir las rutas viables teniendo en cuenta todos los posibles recorridos sobre el STMP que a su vez dependen de la distribución física de las estaciones y vías.

Un primer conjunto de pruebas realizadas a un prototipo software con la implementación de MOGBHS para resolver el problema de configuración de frecuencias (TNFSP) para la actual configuración de rutas de Megabús, buscando minimizar tanto el tiempo total de los pasajeros en el sistema como la capacidad desperdiciada de la flota, y soportado en el modelo de simulación de eventos discretos implementado para la evaluación del fitness de las soluciones, demostró que el algoritmo propuesto encontró más soluciones óptimas que NSGA-II (uno de los mejores algoritmos del estado del arte). La efectividad fue mejorada desde un 175.12% en las 3000 evaluaciones de la función objetivo hasta un 488.68% en las 27000 evaluaciones de la función objetivo.

Un segundo conjunto de pruebas que incluyeron una fase de calibración de parámetros para MOSGBHS y NSGA-II ambos enfocados en resolver el problema de configuración de rutas y frecuencias (TNDFSP) para la actual configuración de estaciones y recorridos de Megabús, buscando minimizar tanto el tiempo total de los pasajeros en el sistema como la capacidad desperdiciada de la flota, y soportado en el modelo de simulación de eventos discretos implementado para la evaluación del fitness de las soluciones, demostró que las soluciones generadas por MOGBHS obtienen mejores resultados en la evaluación de rendimiento frente a los objetivos seleccionados. Mientras que las soluciones generadas por NSGA-II reportan un promedio en el tiempo total de uso de 18.18 minutos y un promedio en capacidad desperdiciada por bus de 22.27 pasajeros, las armonías generadas por MOGBHS reportan un promedio de tiempo de uso de los pasajeros de 15.89 minutos y un promedio de capacidad desperdiciada por bus de 12.81 pasajeros. Resultados que son promisorios y muy superiores frente a NSGA-II.

6.2 Recomendaciones y Trabajo Futuro

En este proyecto se presenta el nuevo algoritmo MOGBHS como una solución al problema de configuración de rutas y frecuencias para un STMP, sin embargo queda un problema por resolver del conjunto de “problemas de diseño y programación de redes de transporte” y es el de configuración de horarios. Como trabajo futuro se planea implementar una versión del algoritmo MOGBHS que resuelva por completo el problema de diseño y programación de redes de transporte.

Dado que el algoritmo propuesto permite la optimización simultánea de múltiples objetivos que pueden ser incluso conflictivos entre sí lo que facilita la aplicación en gran variedad de dominios, se propone evaluar MOGBHS en problemas genéricos multi-objetivo y compararlo con otros algoritmos del estado del arte, como NSGA-II, SPEA2 entre otros.

Para el presente proyecto de investigación se tuvieron algunas restricciones de disponibilidad de información para realizar la calibración del modelo de simulación, se planea realizar un trabajo conjunto con Megabus S.A. para mejorar el modelo de simulación e incluir variables que no pudieron ser consideradas en este trabajo como por ejemplo: la capacidad variable de la flota, la inyección de buses con rutas especiales para suplir demandas particulares, entre otras.

En procura de reducir la complejidad en el proceso de modelado de STMP sobre Arena y evitar los altos costos de licenciamiento se desarrolló una primera versión de ModeLAB, una herramienta web para la construcción de modelos de simulación de STMP orientados a generar el código SIMAN necesario para la evaluación de fitness de las soluciones generadas por MOGBHS. Este proyecto ya obtiene en la simulación los mismos resultados que en Arena, pero a futuro se espera que se pueda integrar con el algoritmo para obtener una solución más económica. Además, expandir ModeLAB para que permita el modelado de los nuevos elementos comentados en el párrafo anterior y su completa inclusión en el proceso de búsqueda de las mejores configuraciones para STMP haciendo uso de MOGBHS.

Dada la debilidad encontrada en el tiempo de ejecución del algoritmo se planea hacer uso de bases de datos para almacenar las evaluaciones de soluciones y reducir así el número de ejecuciones de la simulación, se sugiere implementar el prototipo sobre un lenguaje de programación de menor nivel como por ejemplo C o C++ para mejorar el rendimiento de la aplicación y replantear el mecanismo de búsqueda de rutas cortas para mejorar su eficiencia y minimizar el tiempo de ejecución de MOGBHS.

Actualmente para que un modelo de simulación en código SIMAN pueda ser utilizado por MOGBHS debe ser modificado de forma manual, se planea automatizar este proceso. De esta forma las modificaciones a los modelos de simulación no requerirán trabajo adicional sobre código fuente SIMAN por parte de los modeladores del STMP.

Aprovechando las bondades de los Covering Arrays para apoyar la solución de problemas combinatoriales se planea hacer uso de estos para mejorar el proceso de afinación de parámetros del algoritmo y además estudiar a fondo el uso de los mismos para la generación de las rutas aplicables a los STMP.

7 Bibliografía

1. Cervero, R., *Bus Rapid Transit (BRT): An Efficient and Competitive Mode of Public Transport.* IURD Working Paper 2013-01, 2013.

2. Suzuki, H., R. Cervero, and K. Iuchi, *Transforming cities with transit: Transit and land-use integration for sustainable urban development*. 2013: World Bank Publications.

3. Mauttone, A. and M. Urquhart, *A multi-objective metaheuristic approach for the Transit Network Design Problem.* Public Transport, 2009. **1**(4): p. 253-273.

4. Guihaire, V. and J.-K. Hao, *Transit network design and scheduling: A global review.* Transportation Research Part A: Policy and Practice, 2008. **42**(10): p. 1251-1273.

5. Barnhart, C. and G. Laporte, *Handbooks in Operations Research & Management Science: Transportation: Transportation*. Vol. 14. 2006: Elsevier.

6. Farahani, R.Z., et al., *A review of urban transportation network design problems.* European Journal of Operational Research, 2013. **229**(2): p. 281-302.

7. Zhou, A., et al., *Multiobjective evolutionary algorithms: A survey of the state of the art.* Swarm and Evolutionary Computation, 2011. **1**(1): p. 32-49.

8. Luke, S., *Essentials of Metahuristics.* 2010.

9. Abdullah Konak, D.W.C., Alice E. Smith, *Multi-objective optimization using genetic algorithms: A tutorial.* Reliability Engineering and System Safety 2006. **91**.

10. Mazloumi, E., et al., *Efficient Transit Schedule Design of timing points: A comparison of Ant Colony and Genetic Algorithms.* Transportation Research Part B: Methodological, 2012. **46**(1): p. 217-234.

11. Robinson, S., *Discrete-event simulation: from the pioneers to the present, what next?* Journal of the Operational Research Society, 2005. **56**(6): p. 619-629.

12. Yu, B., et al., *Transit route network design-maximizing direct and transfer demand density.* Transportation Research Part C: Emerging Technologies, 2012. **22**(0): p. 58-75.

13. Allen, T.T., *Introduction to Discrete Event Simulation and Agent-Based Modeling.* Columbus: Springer, 2011.

14. Automation, R., *Arena simulation software.* Accessed November, 2013. **24**.

15. Bapat, V. and D.T. Sturrock. *The arena product family: enterprise modeling solutions: the arena product family: enterprise modeling solutions*. in *Proceedings of the 35th conference on Winter simulation: driving innovation*. 2003: Winter Simulation Conference.

16. Pegden, C.D., R.P. Sadowski, and R.E. Shannon, *Introduction to simulation using SIMAN*. 1995: McGraw-Hill, Inc.

17. Pan, Q.-K., et al., *A self-adaptive global best harmony search algorithm for continuous optimization problems.* Applied Mathematics and Computation, 2010. **216**(3): p. 830-848.

18. Mahdavi, M., M. Fesanghary, and E. Damangir, *An improved harmony search algorithm for solving optimization problems.* Applied Mathematics and Computation, 2007. **188**(2): p. 1567-1579.

19. Omran, M.G. and M. Mahdavi, *Global-best harmony search.* Applied Mathematics and Computation, 2008. **198**(2): p. 643-656.

20. Pichpibul, T. and R. Kawtummachai. *Modified Harmony Search Algorithm for the Capacitated Vehicle Routing Problem*. in *Proceedings of the International MultiConference of Engineers and Computer Scientists*. 2012.

21. Lee, K.S. and Z.W. Geem, *A new meta-heuristic algorithm for continuous engineering optimization: harmony search theory and practice.* Computer Methods in Applied Mechanics and Engineering, 2005. **194**(36â€“38): p. 3902-3933.

22. Spall, J.C. *Stochastic Optimization*. in *Handbook of Computational Statistics*. 2004: Citeseer.

23. Deb, K., et al., *A fast and elitist multiobjective genetic algorithm: NSGA-II.* Evolutionary Computation, IEEE Transactions, 2002. **6**(2): p. 17.

24. Fortin, F.-A. and M. Parizeau. *Revisiting the NSGA-II crowding-distance computation*. in *Proceedings of the 15th annual conference on Genetic and evolutionary computation*. 2013: ACM.

25. Eckart Zitzler, Marco Laumanns, and L. Thiele, *SPEA2: Improving the Strength Pareto Evolutionary Algorithm.* TIK-Report, 2001. **103**.

26. Yu, B., et al. *Optimizing bus transit network with parallel ant colony algorithm*. in *Proceedings of the Eastern Asia Society for Transportation Studies*. 2005.

27. Torres-Jimenez, J. and E. Rodriguez-Tello, *New bounds for binary covering arrays using simulated annealing.* Information Sciences, 2012. **185**(1): p. 137-152.

28. Torres-Jimenez, J. and I. Izquierdo-Marquez. *Survey of covering arrays*. in *Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2013 15th International Symposium on*. 2013: IEEE.

29. Jose L. Walteros, Germán Riano, and A. Medaglia, *Algoritmo Genético Híbrido para el Diseno de Rutas de un Sistema de Transporte Masivo Urbano.* 2007.

30. Duarte, S., D. Becerra, and L. Niño, *Un Modelo de Asignación de Recursos a Rutas en el Sistema de Transporte Masivo Transmilenio.* Revista Avances en Sistemas e Informática (RASI), 2008. **5**(1): p. 163-171.

31. Nikolić, M. and D. Teodorović, *Transit network design by Bee Colony Optimization.* Expert Systems with Applications, 2013. **In press**(0).

32. Nayeem, M.A., M.K. Rahman, and M.S. Rahman, *Transit network design by genetic algorithm with elitism.* Transportation Research Part C: Emerging Technologies, 2014. **46**: p. 30-45.

33. Szeto, W. and Y. Jiang, *Transit route and frequency design: Bi-level modeling and hybrid artificial bee colony algorithm approach.* Transportation Research Part B: Methodological, 2014. **67**: p. 235-263.

34. Arbex, R.O. and C.B. da Cunha, *Efficient transit network design and frequencies setting multi-objective optimization by alternating objective genetic algorithm.* Transportation Research Part B: Methodological, 2015. **81**: p. 355-376.

35. An, K. and H.K. Lo, *Two-phase stochastic program for transit network design under demand uncertainty.* Transportation Research Part B: Methodological, 2016. **84**: p. 157-181.

36. Mauttone, A. and M.E. Urquhart, *Diseño óptimo de recorridos y frecuencias para transporte público.* 2007.

37. Beltran, B., et al., *Transit network design with allocation of green vehicles: A genetic algorithm approach.* Transportation Research Part C: Emerging Technologies, 2009. **17**(5): p. 475-483.

38. Cipriani, E., S. Gori, and M. Petrelli, *Transit network design: A procedure and an application to a large urban area.* Transportation Research Part C: Emerging Technologies, 2012. **20**(1): p. 3-14.

39. Wang, J., G. Sun, and X. Hu, *Optimization of transit operation strategies: A Case Study of Guangzhou, China* Annual Meeting of the Transportation Research Board, 2013.

40. Ibarra-Rojas, O.J., R. Giesen, and Y.A. Rios-Solis, *An integrated approach for timetabling and vehicle scheduling problems to analyze the trade-off between level of service and operating costs of transit networks.* Transportation Research Part B: Methodological, 2014. **70**: p. 35-46.

41. Zhao, H. and R. Jiang, *The Memetic algorithm for the optimization of urban transit network.* Expert Systems with Applications, 2015. **42**(7): p. 3760-3773.

42. Giesen, R., et al. *Multi-Objective Transit Frequency Optimization: Solution Method and Its Application to a Medium-Sized City*. in *Transportation Research Board 94th Annual Meeting*. 2015.

43. Glover, F., *Tabu search-part I.* ORSA Journal on computing, 1989. **1**(3): p. 190-206.

44. Glover, F., *Tabu search—part II.* ORSA Journal on computing, 1990. **2**(1): p. 4-32.

45. Allen, T.T., *Introduction to ARENA Software*, in *Introduction to Discrete Event Simulation and Agent-based Modeling*. 2011, Springer. p. 145-160.

46. Pegden, C.D. *Introduction to SIMAN*. in *Proceedings of the 15th conference on Winter simulation-Volume 1*. 1983: IEEE Press.

47. Salazar Restrepo, J.P. and C.R. Campo Zambrano, *Herramienta web para el Modelado de Sistemas de Transporte Masivo de Pasajeros soportada en simulación discreta usando SIMAN*, in *Facultad de Ingeniería Electrónica y Telecomunicaciones*. 2016, Universidad del Cauca: Popayán, Colombia.

48. Wang, C.-M. and Y.-F. Huang, *Self-adaptive harmony search algorithm for optimization.* Expert Systems with Applications, 2010. **37**(4): p. 2826-2837.

49. Cobos, C., D. Estupiñán, and J. Pérez, *GHS+ LEM: Global-best Harmony Search using learnable evolution models.* Applied Mathematics and Computation, 2011. **218**(6): p. 2558-2578.

50. Khalili, M., et al., *Global Dynamic Harmony Search algorithm: GDHS.* Applied Mathematics and Computation, 2014. **228**: p. 195-219.

51. Kumar, V., J.K. Chhabra, and D. Kumar, *Parameter adaptive harmony search algorithm for unimodal and multimodal optimization problems.* Journal of Computational Science, 2013.

52. Bussieck, M., *Optimal lines in public rail transport*. 1998: Citeseer.

53. Borndörfer, R., M. Grötschel, and M.E. Pfetsch, *A path-based model for line planning in public transport*. 2005: Konrad-Zuse-Zentrum für Informationstechnik Berlin.

54. Lampkin, W. and P. Saalmans, *The design of routes, service frequencies, and schedules for a municipal bus undertaking: A case study.* OR, 1967: p. 375-397.

55. Carrese, S. and S. Gori, *An urban bus network design procedure.* Applied Optimization, 2002. **64**: p. 177-196.

56. Lee, Y.-J. and V.R. Vuchic, *Transit network design with variable demand.* Journal of Transportation Engineering, 2005. **131**(1): p. 1-10.

57. Pattnaik, S., S. Mohan, and V. Tom, *Urban bus transit route network design using genetic algorithm.* Journal of Transportation Engineering, 1998. **124**(4): p. 368-375.

58. Tom, V. and S. Mohan, *Transit route network design using frequency coded genetic algorithm.* Journal of Transportation Engineering, 2003. **129**(2): p. 186-195.

59. Sivasubramani, S. and K. Swarup, *Multi-objective harmony search algorithm for optimal power flow problem.* International Journal of Electrical Power & Energy Systems, 2011. **33**(3): p. 745-752.

60. Dijkstra, E.W., *A note on two problems in connexion with graphs.* Numerische mathematik, 1959. **1**(1): p. 269-271.

61. Transmilenio S.A., a.m.d.B. *Transmilenio*. 2016 [cited 2016 12 Mayo 2016]; Available from: http://www.transmilenio.gov.co/.

62. Transmetro. *Transmetro*. 2016 [cited 2016 12 de Mayo de 2016]; Available from: http://www.transmetro.gov.co/.

63. Transcaribe, a.d.C.d.i. *Transcaribe*. 2016 [cited 2016 12 de Mayo de 2016]; Available from: http://www.transcaribe.gov.co/.

64. Metrocali, a.d.C. *MIO - Masivo integrado de occidente*. 2016 [cited 2016 12 de Mayo de 2016]; Available from: http://www.mio.com.co/.

65. Megabús S.A, A.P. *Megabus*. 2016 [cited 2016 12 de Mayo de 2016]; Available from: www.megabus.gov.co.

66. Metrolínea S.A, a.d.B. *Metrolínea*. 2016 [cited 2016 12 de Mayo de 2016]; Available from: http://www.metrolinea.gov.co/.

67. Metroplús S.A., a.d.M. *Metroplús*. 2016 [cited 2016 12 de Mayo de 2016]; Available from: http://www.metroplus.gov.co/.

68. Megabús S.A, A.P. *Rutas Megabús*. 2016 [cited 2016 12 de Mayo de 2016]; Available from: http://www.megabus.gov.co/rutas.html.

69. Megabús S.A, A.P., *Información operativa Megabús*. 2016. p. 4.

70. Castro-Gutierrez, J., D. Landa-silva, and J. Moreno Perez. *Nature of real-world multi-objective vehicle routing with evolutionary algorithms*. in *Systems, Man, and Cybernetics (SMC), 2011 IEEE International Conference on*. 2011.