TRABAJO FINAL INTEGRADOR

ESPECIALIZACIÓN EN INGENIERIA EN SISTEMAS DE INFORMACION

Título:

# “Marco de Trabajo para Problemas de Asignaciones basado en Algoritmos Evolutivos”

Autor: Ing. Brayan Julian Barbosa Sierra

Tutor: Mg. Cinthia S. Vegega y Ing. Exequiel Banga

Buenos Aires – Febrero/2022

# CONTENIDO

# RESUMEN

Los Algoritmos Evolutivos (AEs) brindan soluciones a diferentes tipos de problemas de manera eficiente y proporcionan un enfoque metaheurístico haciendo uso de técnicas basadas en la teoría de la evolución de las especies.

Este Trabajo Final Integrador presenta un marco de trabajo de software basado en Algoritmos Evolutivos (AEs) con el fin de establecer una base para la asignación de recursos en espacios establecidos. Inicialmente, este es diseñado utilizando diagramas de clases (UML) y después codificado en el lenguaje de programación Python para probar su rendimiento y eficacia.

Se realizan pruebas tomando como referencia la asignación de clases y creación de un horario académico para una institución educativa que contiene cinco (5) salones, e implementando las etapas de análisis del problema, diseño del algoritmo, codificación del programa, ejecución y verificación.

Finalmente, con base a la ejecución y resultados, se concluye que el marco de trabajo propuesto es una alternativa eficiente para modelar y solucionar problemas de asignación.

**Palabras claves**: algoritmos evolutivos, metaheurísticas, problemas de asignación.

# ABSTRACT

Evolutionary Algorithms (EAs) provide solutions to different types of problems efficiently and provide a metaheuristic approach making use of techniques based on the theory of the evolution of species.

This Final Integrative Paper presents a software framework based on Evolutionary Algorithms (EAs) in order to establish a basis for the allocation of resources in established spaces. Initially, it is designed in class diagrams (UML) and then coded in the Python programming language to test its performance and effectiveness.

This is tested taking as a reference the assignment of classes and the creation of an academic schedule for an educational institution that contains five (5) classrooms, and implementing the stages of problem analysis, algorithm design, program coding, execution and verification.

Finally, based on the execution and results, it is concluded that the proposed framework is an efficient alternative to model and solve allocation problems.

**Keywords**: evolutionary algorithms, metaheuristics, academic schedule.

# INDICE DE FIGURAS

# INDICE DE TABLAS

# NOMENCLATURA

| | |
|---|---|
| ACO | Optimización por Colonias de Hormigas |
| AEs | Algoritmos Evolutivos |
| IE | Institución Educativa |
| ILS | Búsqueda Local Iterativa |
| SS | Búsqueda Dispersa |
| UML | Unified Modeling Language |

# INTRODUCCION

Los procesos de distribución de recursos en espacios ya establecidos que se realizan de forma manual suelen ser complejos al proporcionar una posible solución. Esto, debido a la cantidad de datos que se deben tener en cuenta para evitar una distribución no adecuada. Es por ello, que el objetivo principal del presente trabajo final integrador es plantear un marco de trabajo que use algoritmos evolutivos para solucionar los problemas de asignación de recursos en espacios establecidos.

A los efectos de alcanzar el objetivo principal, presentado en el párrafo anterior, esta investigación posee siete objetivos específicos. En el primero y segundo, se efectúa una investigación bibliográfica y realiza un análisis teórico, ambos basándose en algoritmos evolutivos. En el tercero, se revisan investigaciones relacionadas con el problema de asignación de recursos en espacios establecidos. En el cuarto y quinto, se diseña un marco de trabajo y analiza el problema de asignación en una Institución Educativa, para poder así, en el séptimo extender el marco de trabajo con un enfoque de fases de resolución [Joyanes et al., 2005] a un caso específico. Finalmente se codifica y verifica que el algoritmo se ejecute satisfactoriamente.

Teniendo en cuenta los objetivos propuestos, el trabajo se estructura en cuatro capítulos, los cuales se describen a continuación:

En el capítulo 1 se presenta la naturaleza de la investigación. Dentro del mismo se detalla el planteamiento del problema (sección 1.1), el objetivo general y los objetivos específicos a conseguir para resolver dicho problema (sección 1.2). Finalmente se identifica la metodología de trabajo con el enforque y las herramientas a ser utilizadas (sección 1.3).

En el capítulo 2 se presenta el estado del arte. Con el fin de presentar la información necesaria para la comprensión de los temas desarrollados, se detalla el marco conceptual (sección 2.1), tecnológico (sección 2.2) y referencial (sección 2.3).

En el capítulo 3 se presenta el desarrollo del trabajo. Se describe inicialmente la propuesta de un marco de trabajo que permita la implementación de soluciones basadas en algoritmos evolutivos de manera estructurada y sencilla (sección 3.1). Después, se aplica el marco de trabajo propuesto para dar solución a un problema específico (sección 3.2). Para esto, se analiza el problema, se diseña el algoritmo y se codifica, ejecuta y valida el programa. Finalmente, se revisan y evalúan los resultados del marco de trabajo para validar su funcionamiento (sección 3.3).

Por último, el capítulo 4 presenta las conclusiones del trabajo realizado.

# CAPITULO 1. NATURALEZA DE LA INVESTIGACION

En este capítulo se exponen las generalidades del trabajo final integrador. En primer lugar, se realiza el planteamiento del problema (sección 1.1), en segundo lugar, se enuncian el objetivo general y los específicos (sección 1.2) finalmente, se detalla la metodología usada (sección 1.3).

## 1.1 PLANTEAMIENTO DEL PROBLEMA

Los procesos de distribución de recursos en espacios ya establecidos que se realizan de forma manual suelen ser complejos al proporcionar una posible solución. Esto, debido a la cantidad de datos que se deben tener en cuenta para evitar una distribución no adecuada y la asignación por parte de la persona encargada, ya que esta los estima como mejor le parezcan.

En este trabajo final integrador, se toma como ejemplo de validación el problema de asignación de recursos en una institución educativa, ya que al inicio de cada periodo escolar se presenta la necesidad de asignar los recursos humanos, es decir, determinar los horarios en que los docentes dictaran clases a los grupos de alumnos durante toda la semana.

Con el marco de trabajo propuesto se pretende dejar a nivel general una base para la solución de problemas de asignación en espacios, y a nivel especifico solucionar el problema de asignación de horarios en la institución educativa evitando la existencia de espacios vacíos entre las asignaturas y el cruce y exceso de carga académica de cada docente.

# 1.2 OBJETIVOS

En esta sección se detallan los objetivos del trabajo final integrador. Primero se plantea el objetivo general (sección 1.2.1) y se finaliza con los objetivos específicos (sección 1.2.2).

## 1.2.1 Objetivo General

Plantear un marco de trabajo que use algoritmos evolutivos para solucionar los problemas de asignación de recursos en espacios establecidos.

## 1.2.2 Objetivos Específicos

1. Efectuar una investigación bibliográfica de algoritmos evolutivos.
2. Realizar un análisis de la teoría en la que se asientan los algoritmos evolutivos.
3. Revisar investigaciones relacionadas con el problema de asignación de recursos en espacios establecidos.
4. Diseñar un marco de trabajo basado en algoritmos evolutivos.
5. Analizar el problema de asignación de recursos en una Institución Educativa.
6. Extender el marco de trabajo para dar solución al problema de la Institución Educativa.
7. Codificar y verificar que el algoritmo se ejecute satisfactoriamente.

# 1.3 METODOLOGÍA

En esta sección se detalla la metodología utilizada para la elaboración del trabajo final integrador. Se inicia indicando el tipo de investigación (sección 1.3.1), se continua con el enfoque (sección 1.3.2) y se finaliza con las herramientas usadas (sección 1.3.3).

## 1.3.1 Tipo de Investigación

Este trabajo final integrador tiene un enfoque de fases de resolución de problemas [Joyanes et al., 2005] aplicado a la propuesta de asignación de recursos en una institución educativa. Adicionalmente, se maneja un método experimental en la fase de codificación y ejecución del algoritmo para comprobar la utilidad de los resultados.

Dentro de las fases de resolución de problemas se encuentra:

- **Análisis del problema**: se define el problema y se especifica aquello necesario para su resolución.
- **Diseño de algoritmos**: se construye de forma detallada el algoritmo a codificar.
- **Codificación de un programa**: se implementa en un lenguaje de programación el algoritmo creado.
- **Ejecución y verificación: s**e comprueba que el programa se ejecute y arroje una solución correcta.

## 1.3.2 Enfoque

El trabajo final integrador presenta un enfoque predominantemente cualitativo al describir la propuesta del marco de trabajo que permite la implementación de soluciones basadas en algoritmos evolutivos, sin embargo, en la parte de verificación se usan datos cuantitativos para comprobar la efectividad del algoritmo.

### 1.3.3 Herramientas

Las herramientas usadas para realizar el trabajo son:

- Revisión documental: consiste en el análisis de la información obtenida para el marco y desarrollo del trabajo.

- Sistema operativo: el trabajo integrador se realiza sobre el sistema operativo Windows 10 Home, corriendo en una computadora personal con 8.00 Gb de RAM y un procesador Intel Core i7-7700HQ.

- Lenguaje de programación: el lenguaje seleccionado para realizar el trabajo final integrador es Python en la versión 3.8 [Python, 2021].

- Editor: el editor de código fuente usado es Visual Studio Code debido a su compatibilidad con varios lenguajes de programación, su fácil instalación y la cantidad de herramientas que este provee al desarrollador [Visual Studio Code, 2021].

- Modelado: las figuras y diagramas de clase se realizan con la herramienta de modelado StarUML [StarUML, 2021].

# CAPITULO 2. ESTADO DEL ARTE

En este capítulo se presenta la información necesaria para la comprensión de los temas desarrollados en el trabajo final integrador. Este se divide en el marco conceptual (sección 2.1), tecnológico (sección 2.2) y referencial (sección 2.3).

## 2.1 MARCO CONCEPTUAL

En esta sección se definen los temas principales que se mencionan a lo largo del trabajo y que son necesarios para lograr entender lo que se plasma en el mismo. Primero se define el marco de trabajo (sección 2.1.1), en segundo lugar, se describe la Optimización y Metaheurística (sección 2.1.2) y finalmente se presentan los Algoritmos Evolutivos (sección 2.1.3).

### 2.1.1 Marco de Trabajo

Es una estructura de software compuesta de componentes personalizables e intercambiables para el desarrollo de una aplicación. En otras palabras, un marco de trabajo se puede considerar como una aplicación genérica incompleta y configurable a la que podemos añadirle las últimas piezas para construir una aplicación concreta. Los objetivos principales que persigue un marco de trabajo son: acelerar el proceso de desarrollo, reutilizar código ya existente y promover buenas prácticas de desarrollo como el uso de patrones [Gutiérrez, 2014].

### 2.1.2 Optimización y la Metaheurística

La Optimización se concibe como el proceso de intentar encontrar la mejor solución posible a un problema. Estos tipos de problemas se dividen en dos categorías, aquellos en los que la solución esta codificada mediante valores reales y aquellos codificados por valores enteros. Dentro de la segunda categoría, se encuentran los problemas de

optimización combinatoria, los cuales consisten en encontrar un objeto entre un conjunto finito de posibilidades [Duarte Muñoz, 2007].

La Metaheurística se conoce como:

> una clase de métodos aproximados, que están diseñados para atacar problemas difíciles de optimización combinatoria donde la heurística clásica no ha logrado ser efectiva y eficiente. Además, proporcionan marcos generales que permiten crear nuevos híbridos combinando diferentes conceptos derivados de la Inteligencia Artificial, la Evolución Biológica y los Mecanismos Estadísticos [Osman & Kelly, 1996].

Adicionalmente, en la Metaheurística, se incluyen métodos como Optimización por colonias de hormigas (ACO), Algoritmos Evolutivos (EAs), Búsqueda local iterativa (ILS), Búsqueda dispersa (SS), entre otras [Duarte Muñoz, 2007].

### 2.1.3 Algoritmos Evolutivos

En [Petrowski & Ben-Hamida, 2017], se definen los Algoritmos Evolutivos (EAs) como algoritmos iterativos que hacen que los individuos de una población evolucionen a lo largo de varias generaciones, ya que estos contienen la información minima requerida para representar una solución más o menos eficiente al problema objetivo o una parte de una solución. Los operadores involucrados en este algoritmo son:

- **Generación de individuos**: paso de inicialización para constituir una población siendo la cantidad de individuos un parámetro libre del algoritmo, la población puede ser obtenida desde heurísticas dedicadas si alguna información sobre el problema a resolver es conocida, en caso contrario por defecto los individuos se general aleatoriamente.

- **Evaluación de la población**: paso en el cual se establecen los criterios que permiten decidir cuáles de las soluciones propuestas son mejores respecto del resto y cuáles no. Para ello, se asigna a cada individuo una medida de desempeño que permite distinguir las buenas soluciones de aquellas que no lo son.

- **Selección**: paso en el cual se selecciona los individuos más aptos y calificados y determina cuantas veces un individuo será reproducido en una generación dependiendo la medida de desempeño.

- **Cruzamiento**: paso en el cual se genera uno o más descendientes a partir de combinaciones de varios padres, a menudo dos de ellos, pero también posiblemente la combinación de toda la población de padres.

- **Mutación**: paso en el cual se busca brindar mayor diversidad en la población. El objetivo es alterar un individuo independientemente de los otros, esto ayuda a que se exploren regiones de dominio que probablemente no se han tenido en cuenta aún. Aleatoriamente se buscan posibles soluciones que quizá superen las encontradas hasta ese momento.

En [Yu & Gen, 2010], se definen estos como algoritmos que realizan tareas de optimización con capacidad de evolución y que, además, cuentan con tres características principales:

- **Basada en la población:** los Algoritmos Evolutivos (AEs) mantienen un grupo de solución llamado población con el objetivo de optimizar o conocer el problema de manera paralela.

- **Orientado al fitness:** cada solución en una población se denomina individuo, y cada uno de estos cuenta con una representación genética (código) y una evaluación de desempeño (valor de fitness). Los Algoritmos Evolutivos (AEs) conservan los individuos más adaptados como base de la optimización y convergencia de los algoritmos.

- **Variación impulsada:** los individuos se someten a una serie de operaciones de variación para imitar el cambio genético, y buscar el espacio de la solución.

En [Chiong, et al., 2012], se definen los Algoritmos Evolutivos (AEs) como algoritmos que simulan la teoria de la evolucion de Darwin [Darwin, 1870]. Estos incluyen, una o más poblaciones de individuos compitiendo por recursos limitados, poblaciones cambiando dinámicamente debido al nacimiento y muerte de individuos, una noción de aptitud que refleja la capacidad de un individuo para sobrevivir y reproducir, y una noción de reproducción variacional (los descendientes se parecen a sus padres, pero no son idénticos). Finalmente, estos listan a los Algoritmos Genéticos, la Programación Genética, Estrategias Evolutivas y Programacion Evolutiva, como los cuatro algoritmos evolutivos (AEs) mas usados a nivel mundial.

Finalmente, ejemplificando lo mencionado anteriormente, el procedimiento de algoritmos evolutivos comienza con un conjunto de padres seleccionados al azar. Si alguno de estos padres no cumple con todas las limitaciones físicas, se modifican hasta que lo hacen. En las generaciones siguientes, también se comprueba la viabilidad de cada descendencia. Además, se comparan los valores de aptitud de los padres y su descendencia y se rechazan los peores individuos, preservando a los restantes como padres de la nueva generación. Este procedimiento se repite hasta que se satisface un criterio de terminación dado [Davarynejad, et al., 2012].

## 2.2 MARCO TECNOLÓGICO

En esta sección se plasma la información acerca de las herramientas tecnológicas utilizadas dentro del trabajo final integrador con el fin de que el lector adquiera conocimiento y este al tanto de la tecnología usada en el desarrollo. Primero se enuncia a Python (sección 2.2.1), segundo a Visual Studio Code (sección 2.2.2), finalmente a starUML (sección 2.2.3).

### 2.2.1 Python 3.8

Python [Python, 2021] es un lenguaje de programación que tiene estructuras de datos de alto nivel eficientes y un simple sistema de programación orientado a objetos, permite escribir programas compactos y legibles, y los programas son típicamente más cortos que los programas equivalentes en C, C++ o Java por varios motivos:

- Los tipos de datos de alto nivel permiten expresar operaciones complejas en una sola instrucción;
- La agrupación de instrucciones se hace mediante indentación en vez de llaves de apertura y cierre;
- No es necesario declarar variables ni argumentos.

Python es un programa simple de utilizar, y se encuentra disponible en los sistemas operativos Windows, Mac OS X y Unix.

### 2.2.2 Visual Studio Code

Visual Studio Code [Visual Studio Code, 2021] es un editor de código fuente ligero pero potente que se ejecuta en su escritorio y está disponible para Windows, macOS y Linux. Viene con soporte incorporado para JavaScript, TypeScript y Node.js y tiene un rico ecosistema de extensiones para otros lenguajes (como C ++, C #, Java, Python, PHP, Go) y tiempos de ejecución (como .NET y Unity).

### 2.2.3 StarUML

StarUML [StarUML, 2021] es un modelador de software sofisticado destinado a respaldar el modelado ágil y conciso. Algunas de las características clave son:

- Soporte multiplataforma (MacOS, Windows y Linux)
- Cumple con el estándar UML 2.x
- Diagrama entidad-relación (ERD), flujo de datos (DFD).
- UX moderno
- Soporte de pantalla Retina (High-DPI)
- Compatibilidad con la barra táctil de MacPro Pro
- Desarrollo impulsado por modelos
- API abiertas
- Varias extensiones de terceros
- Validación de modelo asincrónico
- Exportar a documentos HTML
- Actualizaciones automáticas

## 2.3 MARCO REFERENCIAL

En esta sección se enuncian algunos trabajos de autores que han estudiado la asignación de recursos con el método de algoritmos evolutivos, esto, con el fin de observar los resultados obtenidos en dichos estudios y conocer más el contexto en el que se encuentra la elaboración del trabajo final integrador.

Por un lado, algunos estudios que enuncian problemáticas de asignación de recursos y soluciones con Algoritmos Evolutivos en términos generales son:

- En [Pradenas Rojas & Matamala Vergara, 2012], cuyo estudio se titula *"Una formulación matemática y de solución para programar cirugías con restricciones de recursos humanos en el hospital público"*, se aborda la problemática de programación de cirugías de forma manual. Este problema se presenta al obtener una única solución y trayendo como consecuencia la no realización de cirugías de listas programadas o un mal uso de los recursos. Tras la implementación de los Algoritmos Genéticos (GAs), los autores concluyen que el algoritmo permite la programación semanal de intervenciones quirúrgicas, cumpliendo con los requerimientos de pabellones y personal especializado necesario para su realización.

- En [Minjares Lugo et al., 2008], cuyo artículo se titula *"Modelo hidrológico-agronómico-económico para la operación óptima del sistema de presas del río Yaqui*", se expresa la necesidad de desarrollar un modelo integral de optimización anual para definir la operación del sistema de presas del río Yaqui y la asignación del volumen mensual de agua para la irrigación de diferentes cultivos. Tras la implementación de Algoritmos Evolutivos, los autores demuestran que este modelo es una herramienta que puede ser utilizada para optimizar y analizar la operación del sistema de presa, así como para manejar los recursos hidráulicos, seleccionando el patrón de cultivos de acuerdo con sus máximos beneficios económicos y las extracciones óptimas mensuales del agua disponible del sistema de presas del río Yaqui.

- En [Bueno et al., 2018], cuyo artículo se titula *"Conocimiento en acción: Asignación de recursos a familias carentes mediante la aplicación de un algoritmo genético - Proyecto Koinonía"*, se plantea la problemática presente en las organizaciones que trabajan en la acción social, respecto a la distribución y escasez de recursos ante una demanda creciente. Los autores proponen un modelo de asignación de alimentos que permita a través de variables de análisis y contemplando la subjetividad del decisor, atender las necesidades específicas de los grupos familiares. Tras su implementación, los autores concluyen que, en definitiva, este modelo permite gestionar el conocimiento y optimizar la asignación de los recursos.

Por otro lado, algunos estudios que enuncian problemáticas de asignación de recursos y soluciones con Algoritmos Evolutivos en términos de asignación de recursos en instituciones educativas son:

- En [Solano Sabatier, Calvo Marín, & Trejos Picado, 2008], cuyo artículo se titula *"Implementación de un algoritmo genético para la asignación de aulas en un centro de estudio"*, se expresa el problema de asignación de aulas a los cursos que se imparten en cualquier centro educativo. Tras la implementación de un modelo de algoritmos genéticos, los autores concluyen que los resultados son satisfactorios ya que no sólo es posible reducir el enorme número de soluciones que otros métodos obligan a evaluar, sino que también permite encontrar con éxito resultados óptimos o muy buenos en un período corto.

- En [Viñas, Rodríguez, Corona, & Jiménez, 2019], cuyo trabajo se titula *"Software para la generación automática de horarios académicos"* se plantea la presentación de un software para resolver el problema de asignación. Tras la implementación de un modelo de Algoritmo Evolutivo con un operador de mutación adaptativa, los autores lograron minimizar el tiempo de diseño de horarios académicos y cumplir con todas las restricciones planteadas para el caso de estudio propuesto.

Con base a las referencias mencionadas anteriormente, se puede observar que:

- Los Algoritmos Evolutivos son métodos adaptativos que se usan para resolver problemas de asignación en diferentes casos, como lo fue la asignación de intervenciones quirúrgicas en un hospital, agua en un sistema de presas, alimentos a familias carentes, aulas y horarios en instituciones educativas.

- En las referencias se encuentra poco aporte técnico, debido a que se enuncia qué se realiza, pero no se detalla cómo se realiza.

Es por ello, que este trabajo final integrador como se menciona en los objetivos busca plantear y diseñar un marco de trabajo que use Algoritmos Evolutivos para solucionar los problemas de asignación de recursos en espacios establecidos.

# CAPITULO 3. DESARROLLO

Este capítulo describe inicialmente la propuesta de un marco de trabajo que permita la implementación de soluciones basadas en Algoritmos Evolutivos (AEs) de manera estructurada y sencilla (sección 3.1). Después, se aplica el marco de trabajo propuesto para dar solución a un problema específico y analizar su desarrollo y ejecución (sección 3.2). Finalmente, se revisan y evalúan los resultados del marco de trabajo para validar su funcionamiento (sección 3.3).

## 3.1 MARCO DE TRABAJO PARA ALGORITMOS EVOLUTIVOS

Esta sección describe los componentes que conforman el marco de trabajo y su importancia para el mismo. Para ello, los elementos son divididos en 2 etapas, la primera enfocada en mostrar la construcción del ambiente y estructura del posible individuo solución (sección 3.1.1), y la segunda enfocada al desarrollo del Algoritmo Evolutivo (AEs) y la forma en que se relaciona el individuo con las diferentes etapas que este contiene (sección 3.1.2).

### 3.1.1. Ambiente e Individuo Solución

Esta sección muestra las clases necesarias para describir el entorno del problema y el individuo solución con sus diferentes implementaciones. La misma está dividida en Individual (subsección 3.1.1.1), Resource (subsección 3.1.1.2), Assignment (subsección 3.1.1.3), TimeSlot (subsección 3.1.1.4), TimeTable (subsección 3.1.1.5), Schedule (subsección 3.1.1.6), TimeResource (subsección 3.1.1.7), IndividualTime (Subsección 3.1.1.8), SlotSpace (subsección 3.1.1.9), SpaceManager (subsección 3.1.1.10), SpaceResource (subsección 3.1.1.11), IndividuaSpace (subsección 3.1.1.12).

Una de las principales acciones que se efectúa cuando se realiza un Algoritmo Evolutivo es identificar las características tanto externas como internas que puedan afectar al individuo solución, es por ello, que la creación de un entorno o ambiente en el que podamos fácilmente describir la situación actual del problema se vuelve primordial en el marco de trabajo. Para esto, se establece una clase llamada "EnvironmentGeneral" (Figura 1) que permite agregar recursos que posean diferentes características (dependiendo del problema que se quiera resolver) de una forma dinámica.

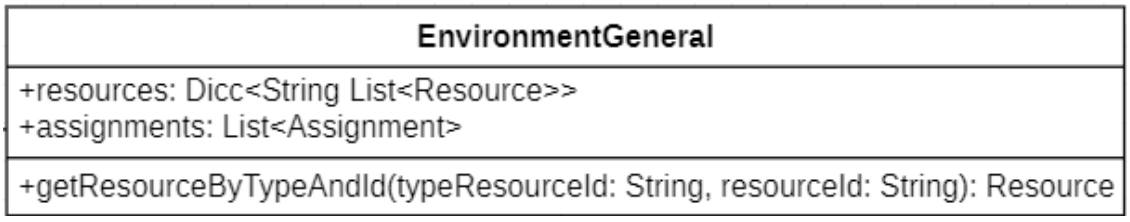


*Figura 1 Clase EnvironmentGeneral*

La clase **EnvironmentGeneral** permite agregar tanto recursos, los cuales pueden estar vinculados jerárquicamente para brindar conocimiento de las relaciones del problema, como asignaciones preestablecidas que se encuentran en el ambiente original de la solución, un ejemplo de esto puede ser el fijar a un trabajador de la salud en un horario especifico en la sala de urgencias.

Aun así, existen variables que no pueden ser generalizadas por una sola clase y que sirven para los diferentes problemas que se establecen como objetivo de solución, para esto se realiza una herencia de clases que permita extender los atributos y funcionalidades de la base. Las clases que fueron creadas con ese propósito son "**EnvironmentTime**" (Figura 2) el cual se encarga de establecer un horario que muestre la disponibilidad para asignar los diferentes recursos, y "**EnvironmentSpace**" (Figura 3) el cual por medio de su matriz de dimensiones permite conocer el espacio que pueden llegar a ocupar los recursos al ser asignados, con estas 2 extensiones del ambiente se consigue atender las necesidades básicas de los problemas de asignación de recurso y tiempo.

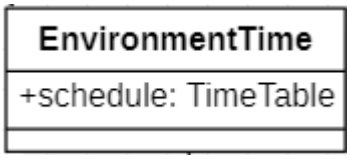


*Figura 2 Clase EnvironmentTime*


*Figura 3 Clase EnvironmentSpace*

A partir de la segmentación de los ambientes varios componentes del marco de trabajo se enfocan a la resolución de un tipo de problema en particular, para tener una mejor visualización de los componentes se realiza una división de este, en donde las clases para establecer problemas de asignación en el tiempo se visualizan en la Figura 4 y los de asignación en espacios en la Figura 5. En estos diagramas, se evidencian las clases compartidas por los diferentes problemas, las relaciones que tienen con las clases que componen al posible individuo solución y las funcionalidades que permiten interactuar con un Algoritmo Evolutivo.

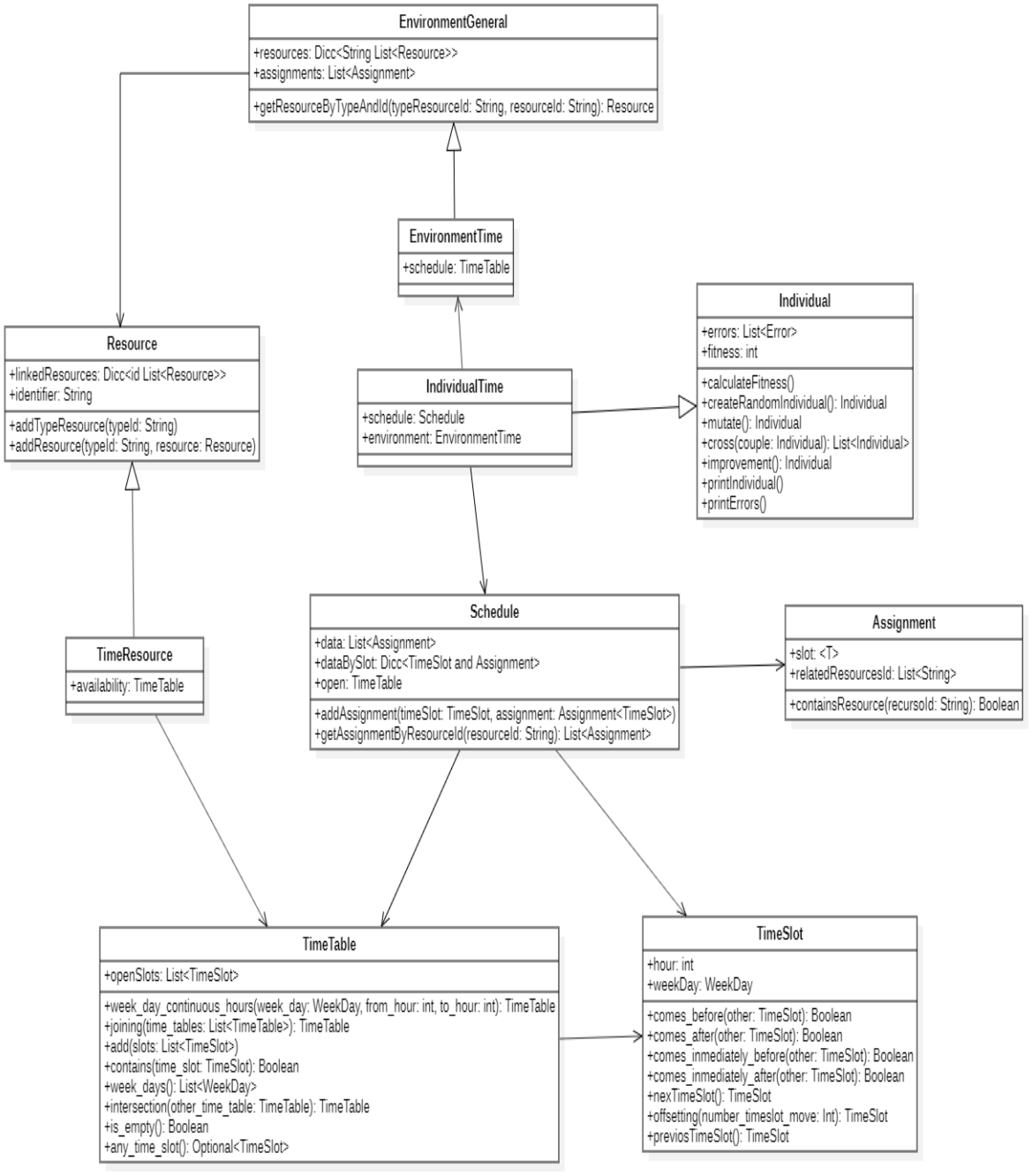


*Figura 4 Diagrama de clases de asignaciones en el tiempo*

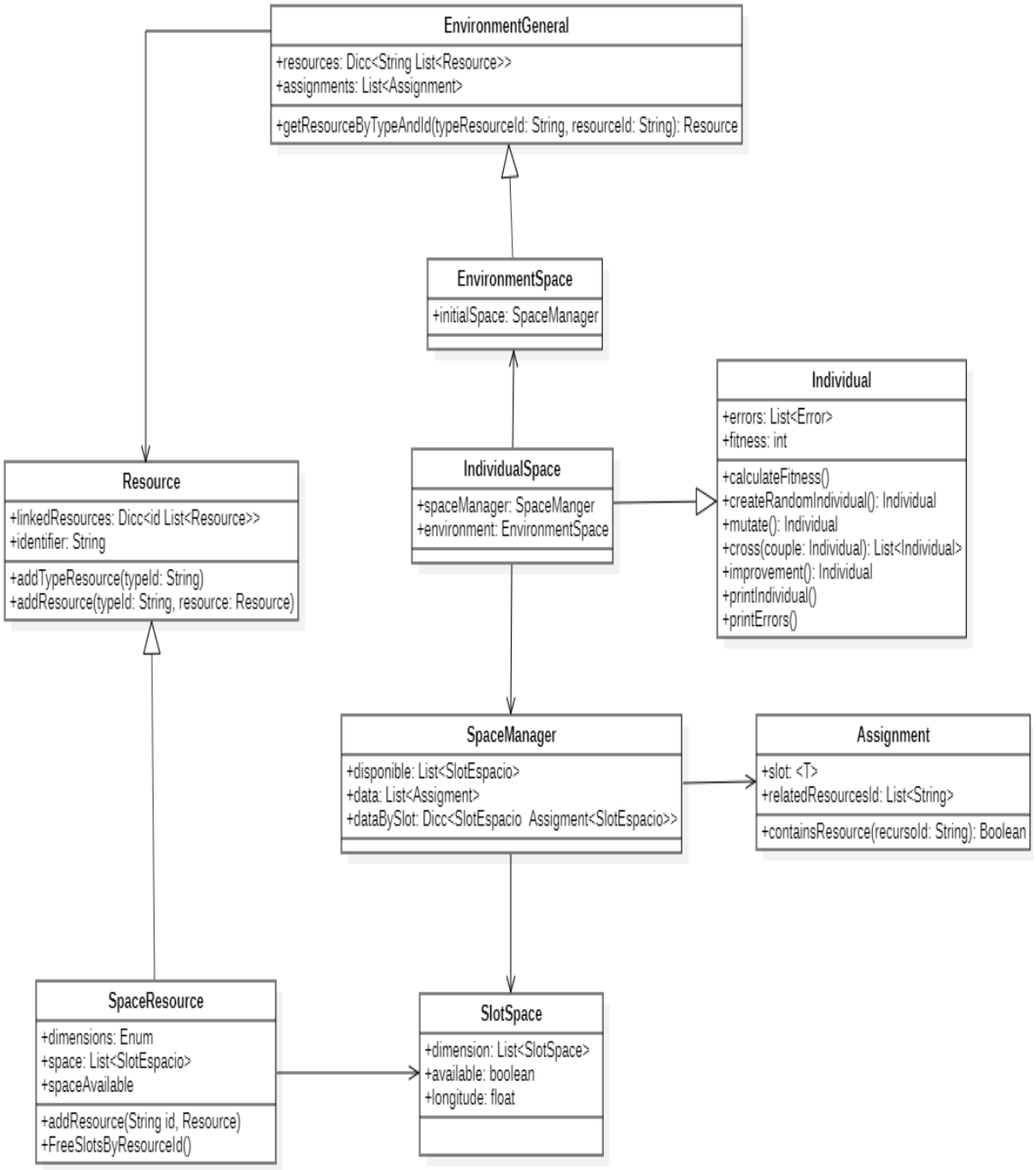


*Figura 5 Diagrama de clases asignación en espacios*

Con base a los diagramas que se muestran en las figuras 4 y 5, se continúa con definir los componentes que estos poseen en común y finalmente se exponen las clases que agregan al marco de trabajo las características necesarias para resolver los diferentes tipos de problemas.

#### 3.1.1.1 Individual

Uno de los principales retos en los Algoritmos Evolutivos es identificar y crear el posible individuo solución, el cual es sometido a las diferentes etapas de este. Para ello, el marco de trabajo propone una clase abstracta "Individual" (Figura 6) que define los principales atributos y funciones que cada individuo debe implementar para poder ejecutarse en el algoritmo evolutivo.

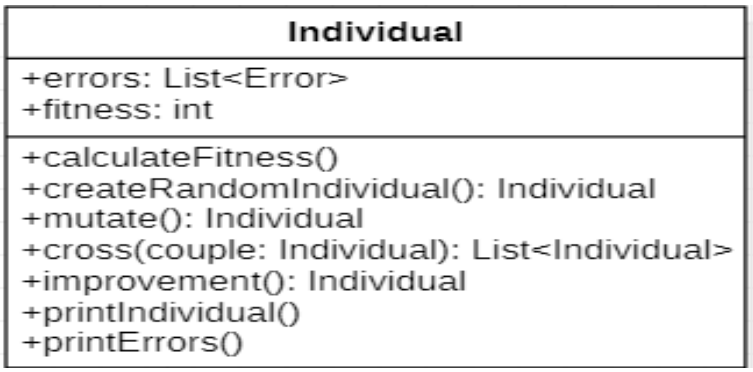


*Figura 5 Clase Individual*

De acuerdo con la Figura 6, el individuo está compuesto por una lista de errores (Errors) que son agregados al momento de evaluar la aptitud (Fitness) del mismo. Este proceso, es realizado por la función **"calculateFitness**", el cual da una valoración numérica que indica lo alejado que esta el individuo de una posible solución. Las otras funciones presentes en la clase sirven para:

- createRandomIndividual: generar un individuo con valores aleatorios que cumpla con las reglas principales del problema.
- mutate: realizar cambios en los atributos del individuo.

- cross: cruzar atributos de 2 individuos y crear otros individuos llamados comúnmente hijos.
- improvement: implementación opcional, la cual permite evaluar los errores presentes en el individuo y cambiar los valores de los atributos para mejorar el fitness a una posible solución.
- printIndividual: visualizar el individuo de una manera fácil y evaluar la posible solución.
- printErrors: visualizar los errores que el individuo posee actualmente y ver cómo afecta el error al fitness actual.

#### 3.1.1.2 Resource

Los recursos permiten describir las características que posee el ambiente inicial y da el punto de partida para establecer los requerimientos del individuo solución, además, estos recursos pueden estar relacionados entre sí y poseer características únicas para cada tipo de problema.

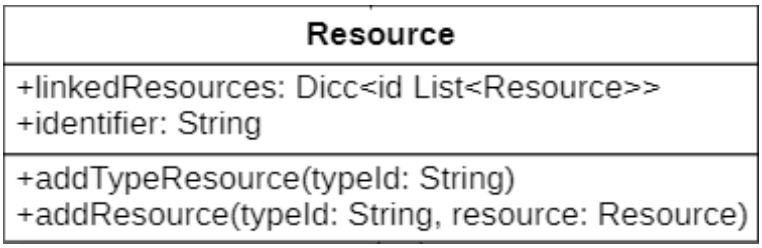


*Figura 6 Clase Resource*

Para poder cubrir las necesidades básicas, el marco de trabajo plantea una clase “Resource” (Figura 7) que permite por medio de una estructura de jerarquía representada por el atributo “linkedResources”, vincular los diferentes recursos que se necesitan para definir el problema. Además de esto, también posee un identificador que permite vincularlo con la posible solución y un par de métodos como lo son “addTypeResource” y “addResource” que permiten de manera dinámica ir agregando recursos a la jerarquía de una manera sencilla. Es importante recalcar que dependiendo la jerarquía que se establezca al momento

de iniciar con la resolución del problema, el Algoritmo Evolutivo puede encontrar las relaciones de una manera más cómoda, por lo que este proceso se considera fundamental al iniciar el proceso.

#### 3.1.1.3 Assignment

Al tratarse de problemas de asignación, la mejor representación de una solución está dada por una serie de objectos que permitan agregar todas las características que el individuo necesita para cumplir con los criterios y reglas que el problema requiere.

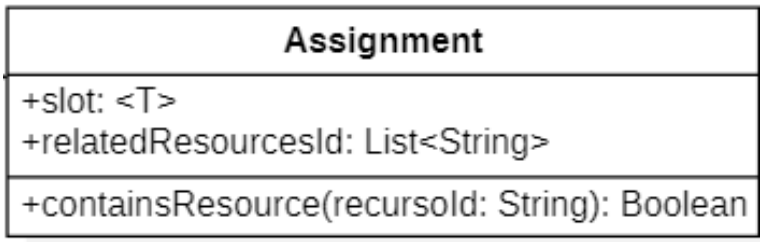


*Figura 7 Clase Assignment*

La figura 8 muestra la clase “Assignment”, la cual posee los atributos base necesarios para establecer la relación con los recursos que componen al individuo solución y estos son:

- slot: representa la zona donde el individuo asigna los recursos, este es de tipo genérico T, lo que permite que al momento de crear se pueda especificar el tipo de espacio disponible para adaptarse al problema que el usuario necesite.

- relatedResourcesId: almacena los identificadores de los recursos que están asignados en el espacio (Slot) actual, permitiendo revisar rápidamente los que ya han sido previamente asignados

- containsResource: función que permite conocer de una manera rápida si un recurso ya ha sido agregado en el objeto, esto es normalmente usado para conocer los que ya fueron utilizados o asignados.

#### 3.1.1.4 TimeSlot

La clase definida como “TimeSlot” (Figura 9) representa el espacio de asignación de un recurso en un determinado tiempo, este determina la hora (hour) y el día de la semana (weekDay) en el cual el recurso fue asignado. Para facilitar el uso del objeto, la clase posee una serie de métodos que ayudan al usuario a comparar, cambiar el horario, buscar el próximo o anterior espacio del objeto actual, entre otras.

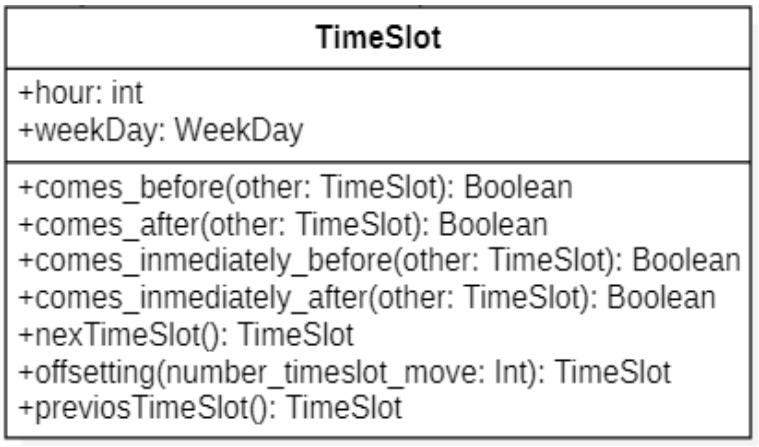


*Figura 8 Clase TimeSlot*

Los métodos usados en la clase TimeSlot son:

- comes_before: encargado de comparar el objeto de tipo TimeSlot que es enviado como parámetro con el actual para verificar si este situado en un espacio anterior en el tiempo.

- comes_after: al igual que el método anterior compara los dos objetos de tipo TimeSlot, pero en este caso se verifica si el actual está situado después en el tiempo al enviado por parámetro.
- comes_inmediately_before: método que posee el mismo comportamiento que comes_before pero este valida que el objeto actual este posicionado exactamente a 1 posición anterior al enviado por parámetro.

- comes_inmediately_after: método que posee el mismo comportamiento que comes_after pero este valida que el objeto actual este exactamente posicionado a 1 posición posterior al enviado como parámetro.

- nextTimeSlot: método que permite obtener el objeto de tipo TimeSlot siguiente en horario comparado con el actual.

- offsetting: método que permite al objeto actual sumarle una cantidad de horas para moverse a través de los días y así obtener uno nuevo ubicado en el momento deseado.

- previousTimeSlot: método que permite obtener el objeto anterior en horario, comparado con el actual

#### 3.1.1.5 TimeTable

Los problemas de asignación en el tiempo están caracterizados por querer buscar la asignación de recursos en un espacio limitado en el tiempo, pero además de esto, constantemente se evidencia que muchos de los recursos también pueden tener disponibilidades condicionadas que se deben tener en cuenta al momento de resolver el problema. Para esto, es necesario crear una clase que permita manejar, cruzar y unir las diferentes disponibilidades o restricciones para el uso del usuario, en este caso, se usa la clase TimeTable (Figura 10).

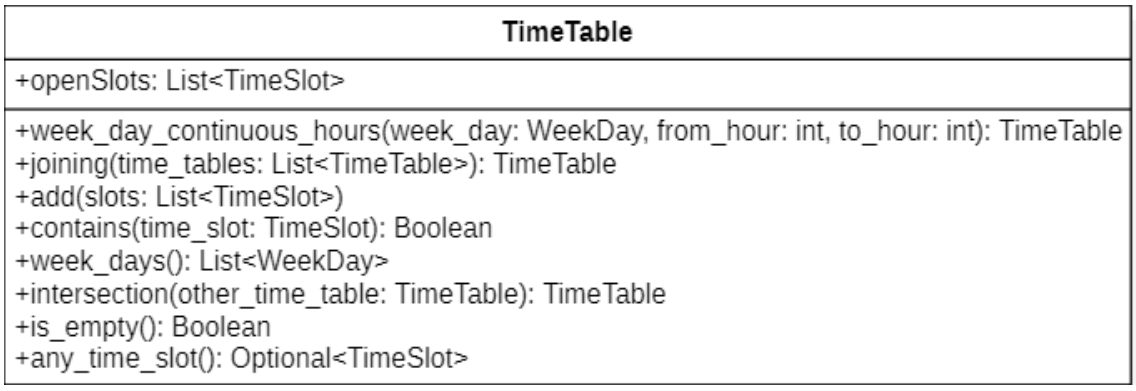


*Figura 9 Clase TimeTable*

La clase TimeTable provee las características necesarias para realizar de una manera controlada las diferentes combinaciones. Los diferentes atributos y métodos encontrados en esta clase son:

- openSlots: lista de objetos de tipo TimeSlot que ayuda a describir un horario, este es usualmente usado para construir la disponibilidad de un recurso en particular y definir en donde se debe llevar a cabo la solución y las restricciones que puede tener el problema.

- week_day_continuos_hours: función que permite por medio de sus parámetros definir el horario de un día de la semana el cual es retornado como un objeto de tipo TimeTable, este método permite al usuario crear los horarios día a día de manera sencilla.

- joining: método que permite unir dos objetos del mismo tipo de la clase, retornando un nuevo objeto del mismo tipo que posee la lista completa de todos los TimeSlot presentes en los dos objetos padres, este, es usado para facilitar la creación de horarios complejos.

- add: método que permite agregar una lista de espacios de tiempo (TimeSlot) al objeto de una forma más rápida.

- contains: método que verifica si un espacio de tiempo ya ha sido agregado al objeto.

- week_days: método que retorna la lista de días que están siendo utilizados en el objeto actual.

- intersection: al igual que una intersección de conjuntos, esta función permite de una manera sencilla para el usuario, obtener los TimeSlot presentes en los dos objetos que se están comparando, esto resulta de gran importancia ya que permite obtener los horarios libres de restricciones de los diferentes recursos que están presentes en el problema

- is_empty: función que permite conocer de forma rápida si el objeto no posee espacios de tiempo agregados en la lista.

- any_time_slot: método que permite obtener un espacio de tiempo seleccionado al azar de la lista que posee el atributo openSlots. Esto es de gran utilidad para realizar las diferentes asignaciones en el proceso evolutivo.

#### 3.1.1.6 Schedule

La clase nombrada como "Schedule" está directamente relacionada con el individuo solución ya que es la encargada de guardar y administrar las asignaciones y el tiempo disponible que este posee.

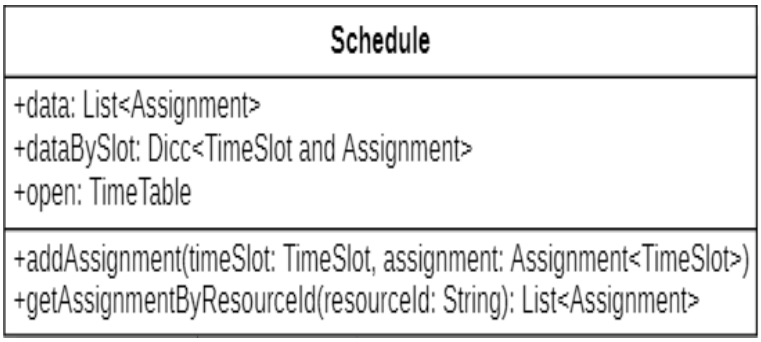


*Figura 10 Clase Schedule*

Los atributos y métodos que tiene esta clase permiten de una manera rápida conocer todos los recursos establecidos y los espacios disponibles para las próximas asignaciones.

- data: lista de asignaciones que posee el individuo, las mismas son generadas inicialmente de forma aleatoria, permitiendo tener una población diversa que por medio de AEs encuentre un individuo solución.

- dataBySlot: diccionario de datos que permite el acceso rápido a las asignaciones que posee el individuo por medio del TimeSlot, el cual identifica un momento específico dentro de la semana.

- open: objeto de tipo TimeTable que posee los espacios que no han sido asignados, esto permite conocer rápidamente la disponibilidad que aún posee el individuo solución, en caso de estar vacío se infiere que el individuo esta completo.

- addAssignment: función que permite agregar una asignación, esto lo añade en el atributo data dataBySlot y elimina el TimeSlot o espacio usado, de la tabla de disponibilidad "open".

- getAssignmentByResourceId: función que busca la lista de asignaciones en las cuales un recurso fue asignado. Esto permite saber de una manera sencilla si existe alguna sobrecarga o si el mismo aún puede ser asignado.

#### 3.1.1.7 TimeResource

Esta es una especificación de la clase Resource definida anteriormente, por lo cual cuenta con todos sus métodos y características base como lo es el atributo linkedResources, además de esto, para poder extender las habilidades de la clase y permitir que los recursos que se van a asignar tengan características enfocadas al problema se agrega el atributo "availability" de tipo TimeTable que agrega un horario de disponibilidad y el cual puede ser configurado por el usuario al momento de establecer el problema.


*Figura 11 Clase TimeResource*

#### 3.1.1.8 IndividualTime

La generalización del individuo posee métodos y características que permiten definirlo y usarlo en un Algoritmo Evolutivo, pero carece de características enfocadas a un tipo de problema específico y para poder abordar los problemas de asignación se realiza esta especificación.


*Figura 12 Clase IndividualTime*

La clase IndividualTime (Figura 13) permite por medio de sus atributos obtener las características base del problema (environment) y agregar un administrador de asignaciones y disponibilidades que el mismo individuo tendrá a lo largo de la solución (schedule).

#### 3.1.1.9 SlotSpace

La clase "SlotSpace" define el espacio que puede ser ocupado por un recurso en un momento determinado del Algoritmo Evolutivo, esta se puede ver como la representación de una caja en la cual se guardaran los elementos del problema.

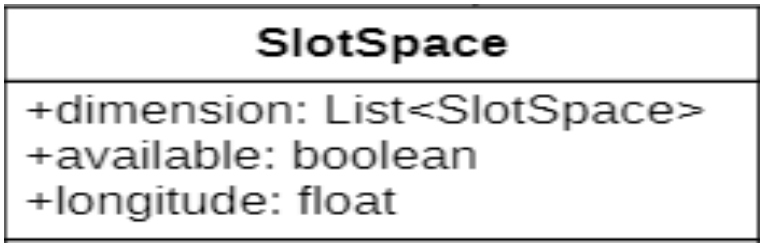


*Figura 13 Clase SlotSpace*

Los atributos que componen la clase SlotSpace son:

- dimensión: lista de objetos de tipo SlotSpace que permiten crear por medio de esta asociación espacios de 2 o 3 dimensiones según requiera el problema.

- available: permite conocer si el espacio está disponible o ya fue asignado a algún recurso.

- longitude: medida mínima en la cual un recurso puede ser asignado, el definir la longitud mínima ayuda a reducir la cantidad de objetos de tipo slotSpace que el problema tiene que usar. Es importante entender que un recurso puede usar varios objetos de este tipo para poder ser asignado al individuo solución con la única condición de que estos sean contiguos.

#### 3.1.1.10 SpaceManager

La clase "SpaceManager" permite al individuo administrar tanto los espacios que posee para determinar cuales están aún disponibles, como las asignaciones que se han realizado a lo largo del proceso y que posee el mismo individuo.

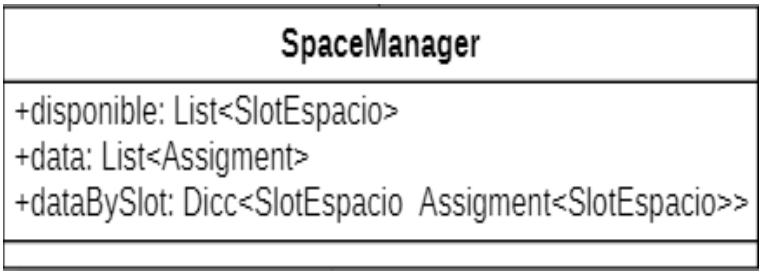


*Figura 14 Clase SpaceManager*

#### 3.1.1.11 SpaceResource

Al igual que la especificación anterior, TimeResource es usada para agregar atributos a los recursos que lo necesiten dependiendo el tipo de problema. En este caso, se vuelve necesario definir las características que permiten añadir el espacio disponible y el tamaño de recursos que se asignaran o usaran en el problema.

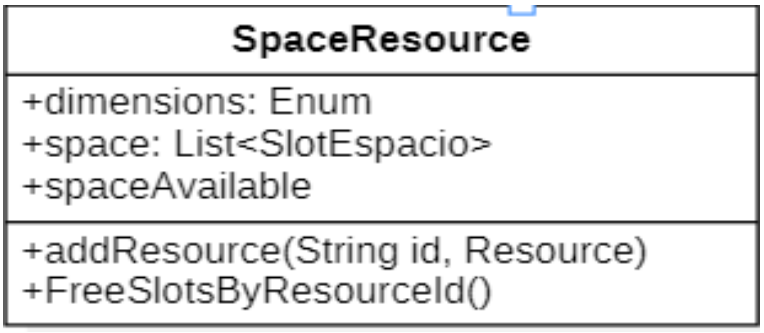


*Figura 15 Clase SpaceResource*

SpaceResource cuenta con atributos que permiten agregar las dimensiones y longitudes que tiene cada recurso en particular para poder establecer el área ocupada o disponible de cada uno.

#### 3.1.1.12 IndividualSpace

Al igual que en el caso IndividualTime, esta clase está enfocada a solucionar un tipo de problema en específico como lo es la asignación en un espacio limitado. La clase IndividualSpace (Figura 17) permite por medio de sus atributos obtener las características base del problema (environment) y agregar un administrador de las asignaciones y espacios disponibles que el mismo individuo tendrá a lo largo de la solución (spaceManager).


*Figura 16 Clase IndividualSpace*

### 3.1.2. Algoritmo Evolutivo

El marco de trabajo que se propone busca por medio de Algoritmos Evolutivos encontrar la solución a problemas de asignación. Para poder llevar a cabo este objetivo, posee clases base que permiten al usuario extender las funcionalidades para acoplarse a un tipo de problema en específico. Como se muestra en la figura 18, se crea un administrador (StandarSolver) encargado de recibir los parámetros necesarios para iniciar a ejecutar los N algoritmos evolutivos que permitan llegar a una solución. En esta sección, se explican a detalle los parámetros de las clases StandarSolver (subsección 3.1.2.1) y EvolutionaryAlgorithm (subsección 3.1.2.2) y como estos están relacionados directamente con el individuo.

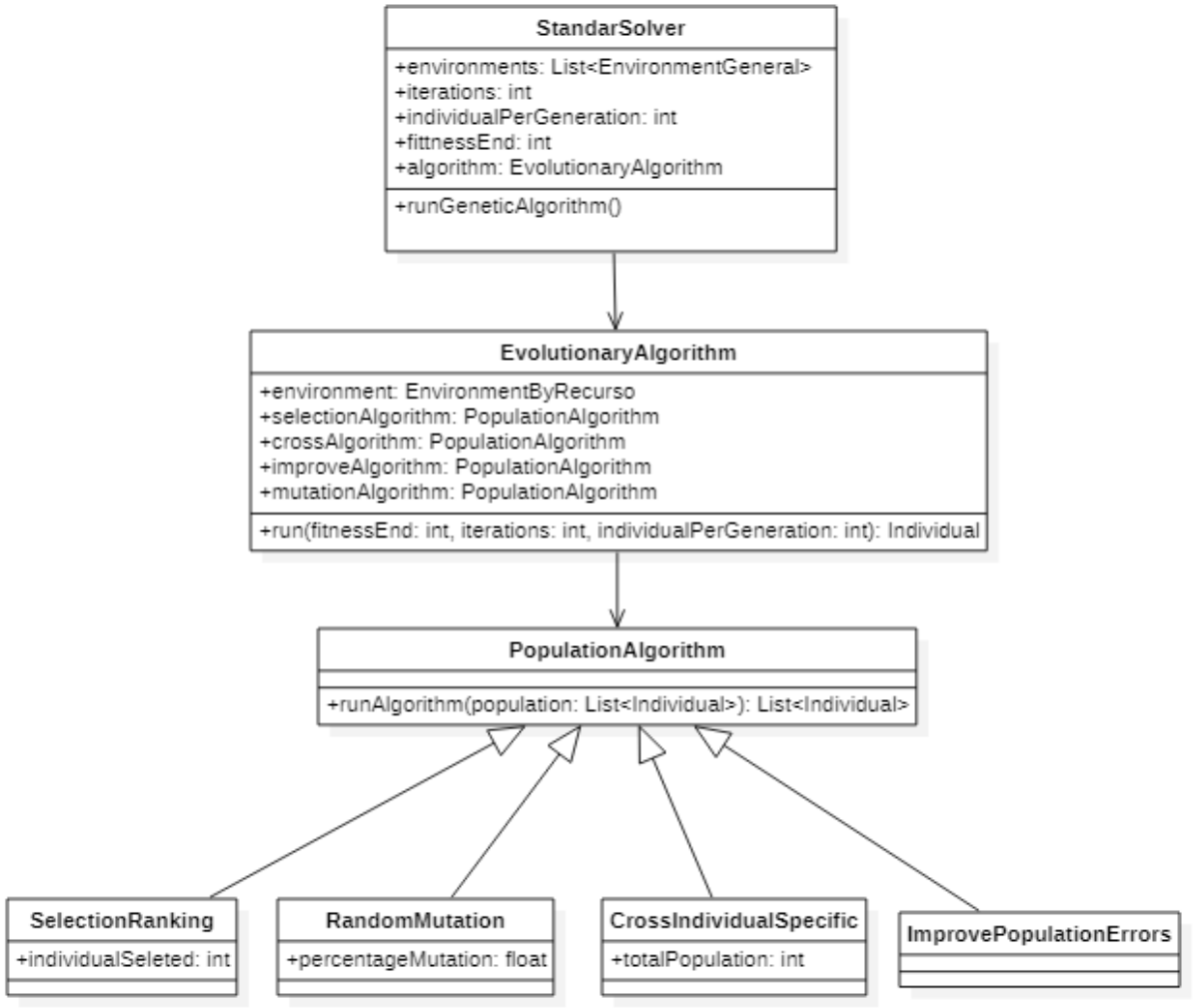


*Figura 17 Clases base algoritmo evolutivo*

#### 1.1.2.1 StandarSolver

Esta clase es la encargada de recopilar la información necesaria para la resolución del problema de asignación.

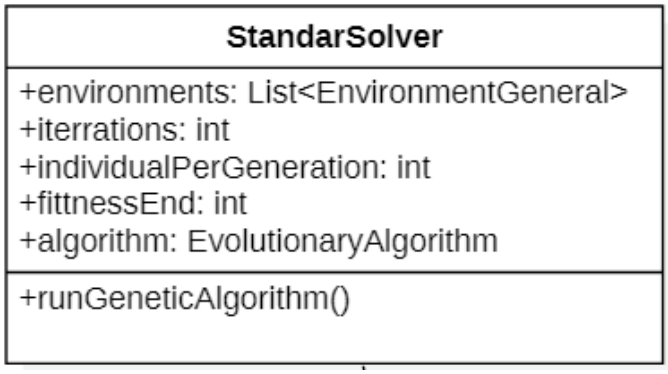


*Figura 18 Clase StandarSolver*

Los atributos que presenta esta clase son:

- environments: lista que contiene la información necesaria que describe el problema a resolver. Puede ser cualquier objeto que herede de la clase EnvironmenGenral.

- iterations: número de iteraciones máximo que realizara el algoritmo evolutivo.

- individualPerGeneration: cantidad de individuos que son evaluados en cada iteración del algoritmo evolutivo.

- fitnessEnd: valor numérico que indica la proximidad del individuo a la solución, usualmente se quiere obtener una solución sin errores, por lo cual un valor habitual puede ser 0, indicando que se requiere una solución completa sin errores.

- algorithm: objeto de tipo EvolutionaryAlgorithm (Figura 20) que es ejecutado por cada uno de los elementos de la lista de environments para obtener una solución completa del problema.

#### 1.1.2.2 EvolutionaryAlgorithm

Esta clase muestra las características base mencionadas en la definición de Algoritmos Evolutivos (AEs), en ella se definen los algoritmos que se usan en cada etapa. Adicionalmente, se agrega una etapa de mejora (improveAlgotithm) que permite corregir errores comunes encontrados en el problema específico que se esté resolviendo, lo que mejorara el desempeño del algoritmo impidiendo que pueda quedar en caminos sin solución o en errores difíciles de mejorar con el cruzamiento o la mutación.

Las etapas presentes en la clase y la forma en que serán ejecutadas para encontrar el individuo solución están representadas en la figura 20. Estas etapas tienen una fuerte vinculación con la clase Individual ya que en cada una se ejecuta el método correspondiente para que cada individuo se pueda evaluar (calculateFitness), realizar un cruzamiento (cross), mejorar (improvement) o mutar (mutate).

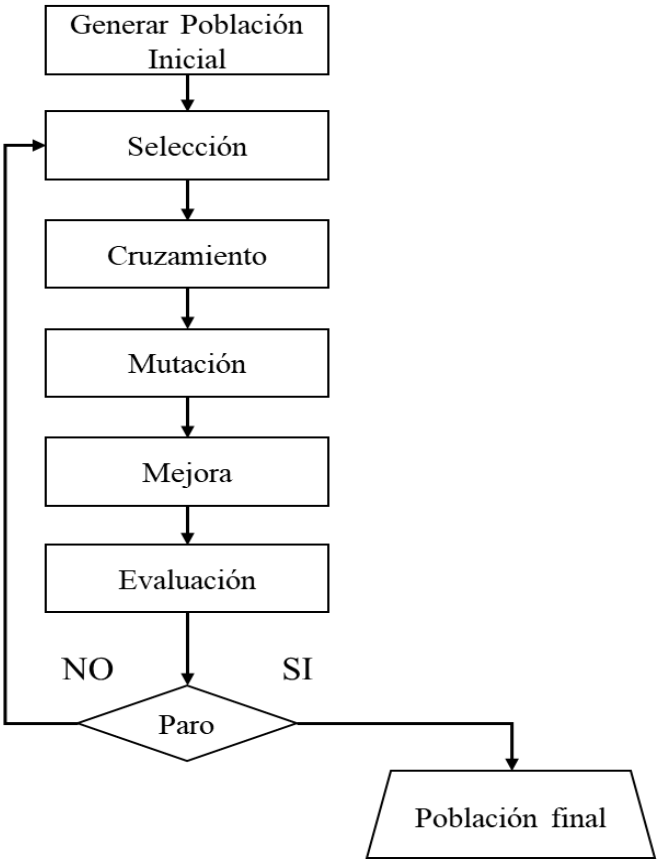


*Figura 19 Algoritmo Evolutivo Propuesto*

La figura anterior muestra como es ejecutado el algoritmo dentro de la clase EvolutionaryAlgorithm (Figura 21).

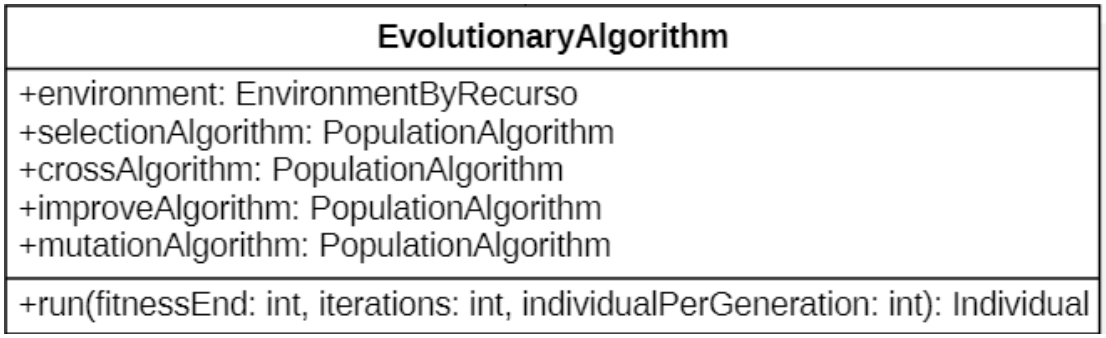


*Figura 20 Clase EvolutionaryAlgorithm*

Inicialmente, esta empieza generando la población inicial, la cual es realizada por la función initialPopulation y usa el individualReference para crear la cantidad de individuos previamente definidos por el usuario. Después, para las siguientes etapas, se crea la interfaz general llamada PopulationAlgorithm para que todos los algoritmos se implementen y ejecuten de una manera sencilla, asimismo esta provee el método runAlgorithm encargado de recibir la población y devolver una nueva con las modificaciones realizadas por cada etapa respectivamente.

En la figura 18, presentada anteriormente, se visualiza un ejemplo de implementación de cada algoritmo necesario en el Algoritmo Evolutivo, como lo es la selección mediante ranking (SelectionRanking) que requiere saber el porcentaje de individuos aceptados por cada generación, mutación aleatoria (RandomMutation) el cual permite por medio de un porcentaje y un valor aleatorio decidir si el individuo debe o no sufrir una mutación en la generación actual, el algoritmo de cruzamiento (CrossIndividualSpecific) el cual necesita conocer el tamaño de la población inicial para detenerse y una clase para mejora o corrección de errores (ImprovePopulationErrors) que es opcional y se ejecuta según lo requiera el problema.

Finalmente, para obtener el individuo solución debe ejecutar el método “run” al cual se le envían parámetros que previamente se configuran como lo son: la cantidad de iteraciones, aptitud del individuo (fitness) como condición de fin y la cantidad de individuos que se quieren por generación.

# 3.2 APLICACIÓN DEL MARCO DE TRABAJO PROPUESTO

En esta sección se aplica el marco de trabajo para dar solución al problema de asignación de recursos en el tiempo (asignaturas) dentro de una institución educativa, lo que permite realizar pruebas, ver la facilidad de uso y el ahorro de esfuerzo que puede aportar el mismo. Esta sección se divide en análisis del problema (sección 3.2.1), diseño de algoritmos (sección 3.2.2), codificación del programa (sección 3.2.3) y finalmente ejecución y verificación (sección 3.2.4).

## 3.2.1 Análisis del Problema

La asignación de horarios es un problema que afecta a todas las instituciones educativas (IE), sin embargo, existen algunas alternativas que sirven para encontrar solución al problema de asignación. Es por ello, que, en esta sección, se establece la información mínima indispensable y las restricciones que el algoritmo debe cumplir (subsección 3.2.1.1) y los datos que se usan como escenario inicial para la resolución del problema (subsección 3.2.1.2).

### 3.2.1.1 Información y Restricciones

Para llevar a cabo la elaboración de los horarios académicos dentro de una institución educativa se debe contar con información mínima indispensable que consiste en saber los días laborables de la semana y de cada día las horas en que se dictan las clases, las asignaturas que se imparten, los profesores que trabajan en la institución y de cada profesor la carga académica (distribución de los docentes en las diferentes asignaturas que deben dictar). Además, se deben cumplir algunas restricciones obligatorias, como, por ejemplo: evitar el cruce de profesores.

En este ejemplo se toma como base una institución educativa que cuenta con 5 cursos de primaria, 8 profesores y 12 asignaturas que están presentes con diferentes cargas en cada curso y que deben ser impartidas de lunes a viernes de 7:00 a 13:00, adicionalmente, se definen una seria de restricciones que el algoritmo debe cumplir al brindar una posible solución y estas son:

- Un docente puede tener disponibilidad menor al horario académico de la institución (día de la semana, rangos horarios).
- Una materia debe definir la cantidad de horas semanales en un curso dado.
- Una materia debe definir la cantidad mínima y máxima de horas consecutivas para un curso dado.
- Se deben mantener horarios fijos para un docente en una materia/curso, esto quiere decir que se mantienen los mismos horarios todas las semanas.
- No generar dos rangos horarios distintos discontinuos en el mismo día de la semana para un curso/materia.
- La solución debe tener como resultado un calendario de cada uno de los cursos indicando docente, materia, día de la semana y rangos horarios.

#### 3.2.1.2 Conjunto de Datos del Problema

En esta subsección se especifican los datos que se usan como escenario inicial para la resolución del problema, se revisan las materias que corresponden a cada profesor y en que curso se deben impartir las mismas.

A continuación, se muestran los respectivos cursos, materias y profesores que son parte del problema:

- Cursos = {primero, segundo, tercero, cuarto, quinto}.

- Materias = {ingles, sistemas, lengua, química, ciencias, danza, dibujo, física, matemáticas, ed. Física, canto, música}.

- Profesores = {Juan, Pedro, Diana, Lucas, Martha, Luis, Nora, Paola}.

Estas materias son vistas en los diferentes cursos y son impartidas por diferentes profesores según el curso que lo requiera, en la siguiente tabla (Tabla 1) se evidencia la relación de las materias con el curso correspondiente y el profesor que puede ser asignado en cada caso.

*Tabla 1 Asignaturas por curso y profesor*

| | **Juan** | **Pedro** | **Diana** | **Lucas** | **Martha** | **Luis** | **Nora** | **Paola** |
|---|---|---|---|---|---|---|---|---|
| **Primero** | Ed. Física<br>Danza | Matemáticas | Dibujo | Lengua<br>Ingles | | Sistemas | Ciencias | Música<br>Canto |
| **Segundo** | Ed. Física<br>Danza | Matemáticas | Dibujo | Lengua<br>Ingles | | Sistemas | Ciencias | Música<br>Canto |
| **Tercero** | Ed. Física<br>Danza | Matemáticas<br>Física | Dibujo | Lengua<br>Ingles | | Sistemas | Ciencias<br>Química | Canto |
| **Cuarto** | Ed. Física<br>Danza | Matemáticas<br>Física | Dibujo | Lengua | Ingles | Sistemas | Ciencias<br>Química | |
| **Quinto** | Ed. Física<br>Danza | Matemáticas<br>Física | Dibujo | Lengua | Ingles | Sistemas | Ciencias<br>Química | |

Otro de los datos para tener en cuenta en el análisis, es que cada asignatura es dada con una diferente intensidad horaria en cada curso y se debe definir una cantidad mínima y máxima de horas que se puede tener diariamente, así como la cantidad semanal indicada. Estos datos están resumidos en las siguientes tablas (Tablas del 2 al 6).

*Tabla 2 Carga académica por materia y curso - Grado primero*

| Curso | Materia | Hs semanales | Hs Min diarias | Hs Max diarias |
|---|---|---|---|---|
| **Primero** | Ed. Física | 2 | 1 | 2 |
| | Danza | 4 | 1 | 2 |
| | Matemáticas | 5 | 1 | 2 |
| | Dibujo | 4 | 1 | 2 |
| | Lengua | 3 | 1 | 2 |
| | Ingles | 3 | 1 | 2 |
| | Sistemas | 1 | 1 | 1 |
| | Ciencias | 2 | 1 | 2 |
| | Música | 3 | 1 | 2 |
| | Canto | 3 | 1 | 2 |

*Tabla 3 Carga académica por materia y curso - Grado segundo*

| Curso | Materia | Hs semanales | Hs Min diarias | Hs Max diarias |
|---|---|---|---|---|
| **Segundo** | Ed. Física | 2 | 1 | 2 |
| | Danza | 3 | 1 | 2 |
| | Matemáticas | 4 | 1 | 2 |
| | Dibujo | 5 | 1 | 2 |
| | Lengua | 3 | 1 | 2 |
| | Ingles | 3 | 1 | 2 |
| | Sistemas | 2 | 1 | 2 |
| | Ciencias | 2 | 1 | 2 |
| | Música | 4 | 1 | 2 |
| | Canto | 2 | 1 | 2 |

*Tabla 4 Carga académica por materia y curso - Grado tercero*

| Curso | Materia | Hs semanales | Hs Min diarias | Hs Max diarias |
|---|---|---|---|---|
| **Tercero** | Ed. Física | 2 | 1 | 2 |
| | Danza | 2 | 1 | 2 |
| | Matemáticas | 5 | 1 | 2 |
| | Dibujo | 4 | 1 | 2 |
| | Lengua | 3 | 1 | 2 |
| | Ingles | 3 | 1 | 2 |
| | Sistemas | 3 | 1 | 2 |
| | Ciencias | 3 | 1 | 2 |
| | Canto | 3 | 1 | 2 |
| | Física | 1 | 1 | 1 |
| | Química | 1 | 1 | 1 |

*Tabla 5 Carga académica por materia y curso - Grado cuarto*

| Curso | Materia | Hs semanales | Hs Min diarias | Hs Max diarias |
|---|---|---|---|---|
| **Cuarto** | Ed. Física | 2 | 1 | 2 |
| | Danza | 2 | 1 | 2 |
| | Matemáticas | 5 | 1 | 2 |
| | Dibujo | 2 | 1 | 2 |
| | Lengua | 6 | 1 | 2 |
| | Ingles | 4 | 1 | 2 |
| | Sistemas | 3 | 1 | 2 |
| | Ciencias | 3 | 1 | 2 |
| | Física | 2 | 1 | 1 |
| | Química | 1 | 1 | 1 |

*Tabla 6 Carga académica por materia y curso – Grado quinto*

| Curso | Materia | Hs semanales | Hs Min diarias | Hs Max diarias |
|---|---|---|---|---|
| **Quinto** | Ed. Física | 2 | 1 | 2 |
| | Danza | 2 | 1 | 2 |
| | Matemáticas | 5 | 1 | 2 |
| | Dibujo | 2 | 1 | 2 |
| | Lengua | 5 | 1 | 2 |
| | Ingles | 4 | 1 | 2 |
| | Sistemas | 3 | 1 | 2 |
| | Ciencias | 3 | 1 | 2 |
| | Física | 3 | 1 | 2 |
| | Química | 1 | 1 | 1 |

Con las restricciones y los datos definidos para la institución educativa de ejemplo, se obtiene el punto de partida para la creación de un escenario inicial que permita encontrar la solución por medio de Algoritmos Evolutivos (AEs).

### 3.2.2 Diseño del Algoritmo

En esta sección se definen las características necesarias que complementan el marco de trabajo para solucionar el problema de asignación de horarios en una institución educativa, como los son: la especificación del individuo solución (subsección 3.2.2.1), los algoritmos de generación de la población inicial (subsección 3.2.2.2) y función de aptitud (subsección 3.2.2.3). Adicionalmente, se establecen los operadores de selección (subsección 3.2.2.4), cruce (subsección 3.2.2.5), mutación (subsección 3.2.2.6) y mejora (subsección 3.2.2.7) que usa el Algoritmo Evolutivo. Finalmente se presenta el diagrama de clases necesario para la implementación de la solución al problema de horarios (subsección 3.2.2.8).

El desarrollo de este ejemplo al ser un problema de asignación de recursos en el tiempo usa como referencia el diagrama de clases de la figura 4 que se encuentra en la sección 3.1.1 para establecer la relación con el marco de trabajo y usar las clases y funciones que este provee.

Con los datos provistos por el escenario inicial en la subsección 3.2.1.2 se diseña una jerarquía (Figura 22) que permite dividir el problema en pequeños escenarios que puedan ser solucionados de una manera más rápida por el algoritmo, con esto el problema inicial se divide para dar solución por curso sin perder la referencia de las variables que se comparten como son los profesores.

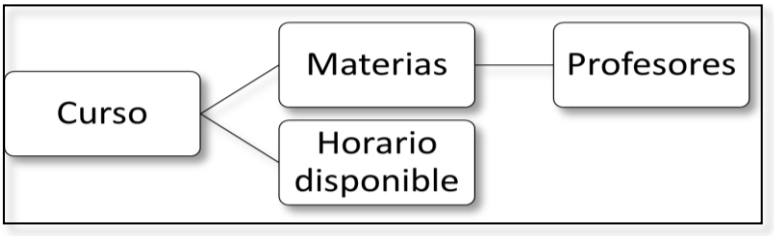


*Figura 21 Jerarquía de escenarios*

### 3.2.2.1 Individuo Solución

El individuo solución (SchoolWeek) se diseña como una especificación de la Clase IndivualTime (Figura 23) el cual permite asociar una matriz de asignaciones en su atributo Schedule y mantener una relación con el escenario inicial y la disponibilidad de los otros recursos en tiempo de ejecución.

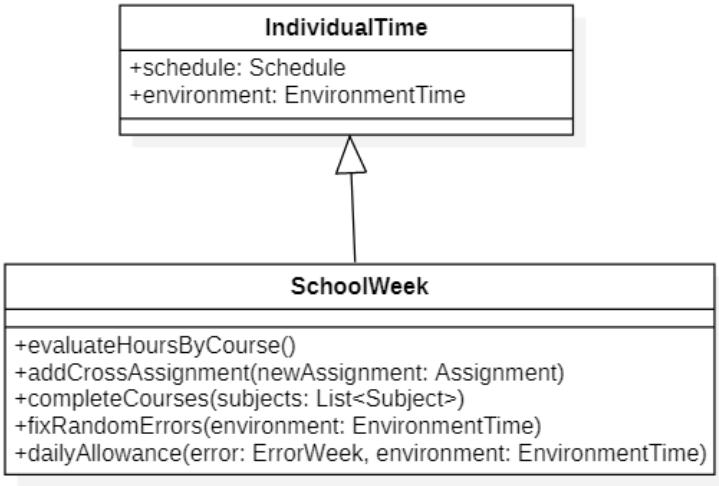


*Figura 22 Clase SchoolWeek*

#### 3.2.2.2 Generación de la Población Inicial

Para la creación de la población inicial se define un algoritmo que permite usar las funcionalidades que provee el marco de trabajo, para esto se seleccionan espacios disponibles al azar, una funcionalidad que facilite la clase TimeTable y se realiza una asignación de un recurso en particular, siendo en este caso la asignatura para el curso determinado.

#### 3.2.2.3 Función de Aptitud

El cálculo de la aptitud del individuo esta dado por la suma de errores presentes en el horario, para cada error se puede tener una penalización diferente y esta es sumada para evaluar la idoneidad del individuo. Para eso, se definen los siguientes errores:

- eMaxS = si la cantidad de horas diarias en el horario supera el máximo permitido para la asignatura se suma 1.

- eDifS = si cantidad de horas semanales es diferente al requerido por la asignatura, en este caso se suma el valor absoluto de la diferencia en la cantidad de horas.

- eCrP = si se presenta cruce de horarios por profesor se suma 1.

- eDisO = si la asignación esta fuera de disponibilidad del profesor se suma1.

Es por ello, que la función de aptitud está dada por la sumatoria de los errores presentes para cada individuo. La fórmula es**:**

**Aptitud= eMaxS + eDifS + eCrP + eDisO**

#### 3.2.2.4 Selección

El proceso de selección implementado para este trabajo es el algoritmo de selección por ranking lo que permite obtener los mejores individuos de cada iteración y generar de esta manera una mejor población, la cantidad de individuos que serán seleccionados en cada población será configurable al momento de ejecutar el algoritmo.

#### 3.2.2.5 Cruce

Para el proceso de cruzamiento se opta por una opción de múltiples cruces, en este se seleccionan dos individuos aleatoriamente que son los denominados padres de los nuevos individuos, después de esto al azar se seleccionan varios puntos de cruce y la información es intercambiada entre ellos para dar origen a un nuevo individuo, el proceso se repite hasta generar una población del mismo tamaño de la población original, este algoritmo da una diversidad más grande a la población y disminuye la probabilidad del estancamiento en un mínimo local.

#### 3.2.2.6 Mutación

Para la mutación se realiza una mutación aleatoria con probabilidad configurable que realiza solo un cambio en una de las asignaciones del individuo de tal forma que permita aumentar la variedad dentro de la población y a su vez promueva la creación de nuevos individuos que pueden o no mejorar el proceso evolutivo.

#### 3.2.2.7 Mejora

Se opta por agregar un último paso y es la opción de mejorar el individuo, para este caso se diseña un algoritmo que identifica los errores que posee el individuo en una iteración y por medio de una mutación dirigida a las asignaciones que presentan esos errores, mejorar el individuo y obtener de una manera más rápida una solución al problema.

### 3.2.2.8 Diagrama de Clases Horario Académico

Después de definir los algoritmos a utilizar, la estructura de los escenarios para dividir el problema inicial, las restricciones obligatorias, la forma de evaluar al posible individuo solución y el formato del horario solución, se procede con el diseño de la aplicación, que permite dar solución a la asignación de horarios en la institución académica. Para esto se diseña un diagrama de clases (Figura 24) que extienda las funcionalidades del marco de trabajo y permita dar solución al problema.

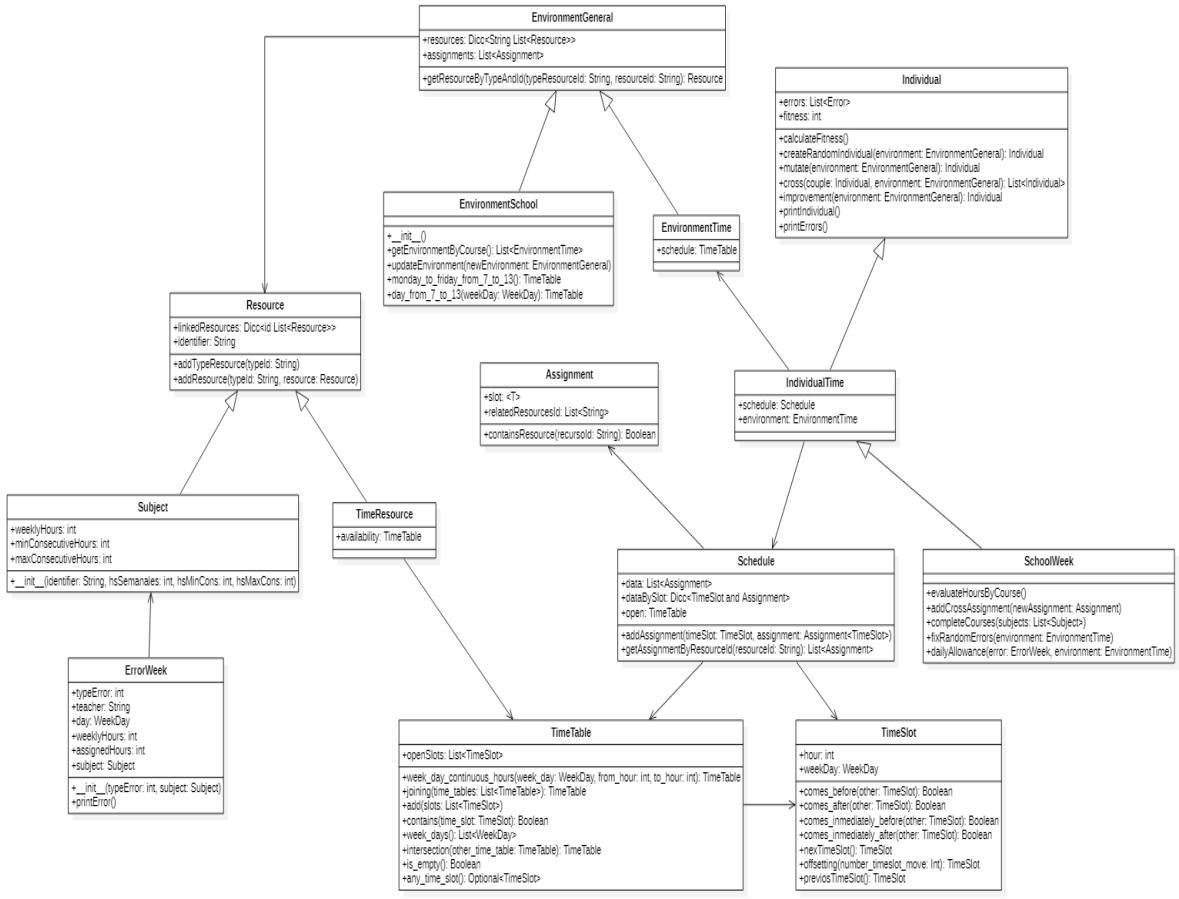


*Figura 23 Diagrama de clases Horario Escolar*

El diagrama anterior (Figura 24) utiliza las clases proveídas por el marco de trabajo para dar solución al problema, con lo cual las clases agregadas están encargadas del comportamiento específico necesario para crear el individuo solución.

Con las definiciones dadas en las anteriores subsecciones se crea el entorno solución EnvironmentSchool que esta creado como una extensión de la clase EnvironmentGeneral y que permite agregar los atributos de curso, profesores, materias y disponibilidades requeridas, para cada recurso mencionado se emplean las siguientes características del marco de trabajo:

- Cursos: para la creación de los cursos se necesita que ellos puedan agregar otra seria de recursos vinculados, por lo cual se usa la clase Resource para crear y vincular las materias que tiene.

- Profesores: para la creación de los profesores se identifica la necesidad de poseer disponibilidades que sean fácilmente configurables, para esto se usa la clase TimeResource que permite con su atributo availability configurar el tiempo que el recurso tiene disponible.

- Materias: para la creación de las materias se define la clase Subject, la cual hereda el comportamiento de Resource, lo que permite que sean agregadas por el ambiente y que así mismo cada materia puede agregar una lista de Resource, en este caso los profesores que dictan la materia en específico. Las materias se generan por curso, lo cual permite que se agreguen los atributos que pide el problema, como lo son las horas semanales, horas máximas consecutivas diarias y horas mínimas consecutivas diarias.

Finalmente, para la creación del individuo solución se realiza una especificación de la clase IndividualTime que realiza las implementaciones de los métodos abstractos basado en las definiciones de la función de aptitud, selección, cruce, mutación y mejora que se aplican en cada iteración del algoritmo evolutivo; Para este último, se crea una clase ErrorWeek, la cual ayuda a identificar los errores y permite al algoritmo de mejora tomar una decisión según el tipo de error.

### 3.2.3 Codificación del Programa

En esta sección se realiza la codificación del marco de trabajo (subsección 3.2.3.1) y las extensiones necesarias para la solución del problema del horario académico (subsección 3.2.3.2), para realizar este desarrollo como se menciona anteriormente (sección 1.3.3) el lenguaje de programación es Python el cual posee estructuras de datos de alto nivel eficientes y un simple sistema de programación, unido a esto se usa Visual Studio Code como editor de texto que proporciona una fácil interfaz de usuario y fácil instalación.

#### 3.2.3.1 Codificación del Marco de Trabajo

En esta sección se muestran las principales clases que provee el marco de trabajo y que ayudan a realizar la solución de un problema de asignación de un horario académico con algoritmos evolutivos. Estas clases permiten al desarrollador tener una estructura definida y resolver el problema. El código disponible en github [Barbosa Sierra, 2021].

Tomando como referencia las figuras 4 y 18 las cuales comprenden el marco de trabajo junto con las clases que permiten implementar los Algoritmos Evolutivos se dispone a la creación de estas.

- **EnvironmentGeneral y EnvironmentTime:** Estas clases son las necesarias para realizar la implementación del problema de los horarios académicos, estas permiten agregar la información inicial para el desarrollo (el código está basado en la Figura 1 y la Figura 2).

```
class GeneralEnvironment():
    resources: Dict[str, List[Resource]]
    assignments: List[Assignment]

    def __init__(self):
        self.assignments = list()
        self.resources = {}

    def getResourceByTypeAndID(self, typeResourceId: str,
               resourceId: str) -> Resource:
        for typeResource in self.resources:
            if(typeResource == typeResourceId):
                resourcesList: List[Resource] =
                       self.resources[typeResource]
                for resource in resourcesList:
                    if resource.identifier == resourceId:
                        return resource

    #Metodo que implementaran las especificaciones
    def updateEnvironment(self,newEnvironment):
        pass
```

El EnvironemtTime hereda las características de GeneralEnvironment y se encarga de añadir el atributo adicional timetable, que permitirá indicar la franja horaria en el cual se debe llevar a cabo el problema de asignación.

```
#Clase que extiende de GeneralEnvironment
class EnvironmentTime(GeneralEnvironment):
    timetable: TimeTable

    def __init__(self):
        super(EnvironmentTime,self).__init__()
```

- **Individual**: clase basada en la Figura 6, se realiza la definición de los 7 métodos presentes, los mismos son métodos abstractos por lo cual no estarán implementados ya que las especificaciones que se creen estarán obligadas a implementar su propio comportamiento. El siguiente código muestra el ejemplo de la clase con sus correspondientes atributos y la definición de uno de los métodos.

```
class Individual(ABC):
    mistakes = []
    fitness: int = 0

    def calculateFitness(self):
        raise NotImplementedError
```

- **IndividualTime**: clase basada en la figura 13 para la resolución de los horarios académicos, el mismo hereda de la clase Individual sus atributos y métodos abstractos, pero no implementa los mismos, solo agrega atributos que permiten el manejo del ambiente especifico y de una mánager que se encarga del manejo de asignaciones.

```
class IndividualTime(Individual):
    timetable:Schedule=None
    environment:EnvironmentTime=None

    #constructor
    def __init__(self, environment:EnvironmentTime):
        self.timetable:Schedule = Schedule(environment.timetable)
```

- **Resource y TimeResource**: la clase resource permite tener una jerarquía de elementos que conforman el individuo solución, a continuación, se evidencian los atributos que permiten esto y las funciones que ayudan a agregar un recurso.

```
class Resource():
    #jerarquia de recursos asociados a la solucion
    linkedResources:Dict[str,List[Resource]]
    identifier:str

    def __init__(self, identifier:str):
        self.identifier=identifier
        self.linkedResources = {}

    def addTypeResourcesLinked(self, typeResource:str):
        if typeResource not in self.linkedResources:
            self.linkedResources[typeResource]=list()

    def addResourceLinked(self,typeResource:str, resource: Resource):
        self.addTypeResourcesLinked(typeResource)
        self.linkedResources[typeResource].append(resource)
```

El resourceTime hereda el comportamiento y atributos de Resource, el cual permite vincularlo y asociarlo a otros recursos según lo requiera para crear jerarquías que faciliten la resolución del problema específico, además de esto agrega el atributo de disponibilidad (availability) que permite agregar una franja horaria para el recurso.

```
#Clase que hereda de Resource
class ResourceTime(Resource):
    availability:TimeTable
```

- **Assignment**: esta clase es la medida más pequeña de la solución, agrupa un espacio de tiempo o espacio específico y los recursos asignados a este, el siguiente código muestra sus atributos y su función principal que ayuda a conocer si un elemento fue asignado previamente.

```
class Assignment:
    timeSlot: TimeSlot
    listResourceId: List[str]

    def __init__(self):
        self.listResourceId= list()

    def containResource(self, resourceId:str)-> bool:
        return True if resourceId in self.listResourceId else False
```

- **Schedule**: esta clase es el administrador de recursos y asignaciones que posee el individuo solución, es la encargada de conocer la disponibilidad de cada recurso y de asignarlo a un espacio disponible. Con su diccionario dataByTimeSlo y su lista de data, permite de manera rápida obtener los elementos necesarios para poder asignar los recursos en cada iteración del algoritmo evolutivo.

```
class Schedule:
    data: List[Assignment]
    dataByTimeSlot: Dict[TimeSlot, Assignment]
    timetable: TimeTable

    def __init__(self, timetable:TimeTable):
        self.data= list()
        self.dataByTimeSlot={}
        self.timetable = TimeTable(timetable._open_slots)

    def addAssignment(self, timeSlot:TimeSlot,assignment: Assignment):
        self.data.append(assignment)
        self.dataByTimeSlot[timeSlot]=assignment
        self.timetable._open_slots.remove(timeSlot)

    def getAssignmentsByResource(self,resourceId)->List[Assignment]:
        return list(filter(lambda assignment:
                 assignment.containResource(resourceId),self.data))
```

- **TimeSlot y TimeTable**: las dos clases a continuación son muy importantes ya que son las encargadas de establecer el arreglo de espacios (TimeSlot) que serán vinculados a una asignación para completar el individuo solución.

  A continuación, se presentan las principales características y funciones de la clase TimeSlot, la cual tiene como principal objetivo la comparación rápida y el conocimiento del arreglo de espacios que tiene el individuo solución. Con estas funciones podremos movernos fácilmente hacia adelante o atrás en horas, días según lo necesita el problema.

```
class TimeSlot:
    week_day: WeekDay
    hour: int

    def __lt__(self, other: TimeSlot):
        return self.comes_before(other)

    def comes_before(self, other_time_slot: TimeSlot) -> bool:
        if self.week_day < other_time_slot.week_day:
            return True
        elif self.week_day == other_time_slot.week_day:
            return self.hour < other_time_slot.hour
        return False

    def comes_after(self, other_time_slot: TimeSlot) -> bool:
        return other_time_slot.comes_before(self)

    # number_of_timeslots_to_move número de casillas que se correrá
     el                  actual TimeSlot
    def offsetting(self, number_of_timeslots_to_move: int)
      -> TimeSlot:
        moved_slot = self
        for _ in range(abs(number_of_timeslots_to_move)):
            if number_of_timeslots_to_move > 0:
                moved_slot = moved_slot.nextTimeSlot()
            else:
                moved_slot = moved_slot.previousTimeSlot()
        return moved_slot

    def previousTimeSlot(self) -> TimeSlot:
        if self.hour == 0:
            return TimeSlot(self.week_day.previous(), 23)
        return TimeSlot(self.week_day, self.hour - 1)

    def __eq__(self, other):
        return self.week_day == other.week_day and self.hour ==
             other.hour
```

Por su parte la clase TimeTable esta creada para poder comparar, unir, y realizar intersecciones de los diferentes recursos que tenga el problema a solucionar, esto da al desarrollador una herramienta para poder gestionar las disponibilidades de los mismo y asignar la solución en una disponibilidad conjunta de todos, entre otras herramientas que pueden ser útiles dependiendo el problema a solucionar.

A continuación, se muestra la codificación de la clase TimeTable con sus principales funciones y características.

```
class TimeTable:
    _open_slots: List[TimeSlot]

    #Metodo que ayuda a la creación inicial de los horarios
    @staticmethod
    def week_day_continuous_hours(week_day: WeekDay, from_hour: int,

      to_hour: int) -> TimeTable:
        slots = list(map(lambda hour: TimeSlot(week_day, hour),
                         list(range(from_hour, to_hour))))
        return TimeTable(slots)

     #Metodo que ayuda unir los espacios y así crear un horario
    @staticmethod
    def joining(time_tables: List[TimeTable]) -> TimeTable:
        return TimeTable(sorted(reduce(operator.concat, list(map(
             lambda tt: tt.slots(), time_tables)))))

    def _add(self, slots: List[TimeSlot]):
        list_without_duplicates = list(set(slots))
        self._open_slots = sorted(self._open_slots +
             list_without_duplicates)

    def slots(self) -> List[TimeSlot]:
        return self._open_slots

    def contains(self, time_slot: TimeSlot) -> bool:
        return self._open_slots.__contains__(time_slot)

    def week_days(self) -> List[WeekDay]:
        return sorted(list(set(map(lambda x: x.week_day,
             self._open_slots))))

    def intersection(self, other_time_table: TimeTable) -
> TimeTable:
        return TimeTable(list(set(self._open_slots).intersection(
             other_time_table._open_slots)))

    def is_empty(self) -> bool:
        return len(self._open_slots) == 0

    def any_time_slot(self) -> Optional[TimeSlot]:
        if len(self._open_slots) == 0:
            return None
```

- **StandardSolver**: esta clase es la encargada de ejecutar el Algoritmo Evolutivo y sub- dividirlo en las jerarquías necesarias que el desarrollador quiera, estas jerarquías están agrupadas en el atributo de la clase "environments", el cual se usa para recorrer estas divisiones e ir actualizando los recursos cuando se completa la solución de cada etapa, esto garantizara que los que ya fueron asignados no se repitan o usen espacios que ya no están disponibles.

  Para el caso de los horarios estudiantiles se generan divisiones por curso, que al completar los 5 cursos se tendrá un individuo solución. El siguiente código muestra la creación de la clase

```
class StandardSolver():
    environments: List[GeneralEnvironment]
    iterations: int
    finalFitness: int
    quantityIndividuals: int
    evolutionaryAlgorithm: EvolutionaryAlgorithmManager
    classReference:Type[_T]

    def runAlgorithm(self):
        #Se soluciona por sub ambientes que puedan generar el
         problema
        for environment in self.environments:
            self.evolutionaryAlgorithm.cleanSolution()
            self.evolutionaryAlgorithm.environment = environment
            #Se crea una referencia de la clase del individuo
             solucion
            self.evolutionaryAlgorithm.individualReference =
               self.classReference(environment)
            self.evolutionaryAlgorithm.run(self.quantityIndividuals,
                self.iterations)
          #se actualiza el ambiente con los recursos ya asignados
            environment.updateEnvironment(
               self.evolutionaryAlgorithm.bestIndividual.
               environment)
```

- **EvolutionaryAlgorithm**: como lo muestra la figura 21, esta clase define los atributos necesarios para la ejecución de las diferentes etapas de un algoritmo evolutivo como los son, la selección, cruzamiento, mutación y en el caso de este desarrollo la mejora del individuo, las variables están inicializadas con valores nulos exceptuando el umbral de aptitud y el valor de la mejor aptitud inicial, los cuales tienen valores iniciales de 0 y 1000 respectivamente, estos pueden ser cambiados si el problema lo requiere.

```
class EvolutionaryAlgorithmManager():
    selectionAlgorithm: SelectionAlgorithm = None
    mutationAlgorithm: MutationAlgorithm = None
    crossAlgorithm: CrossAlgorithm = None
    aptitudeThreshold = 0
    finishActions:FinishAction = None
    individualReference: Individual=None
    environment =None
    bestFitness =1000
    bestIndividual:Individual=None
```

La función “run” de la clase EvolutionaryAlgorithm es la encargada de encontrar una solución al problema tomando los atributos de la clase y los parámetros enviados al método, a continuación, se muestra un fragmento de código que muestra el funcionamiento y la forma en que va iterando en cada generación, se usan los comentarios de Python, que pueden ser identificados por el símbolo “#” para separar y explicar el funcionamiento de este.

```
def run(self, populationQuantity: int, generations: int):
        #Se crea la poblacion inicial, con la referencia del
         individuo y la cantidad especificada
        population = self.createStartingPopulation(
             populationQuantity, self.individualReference)

        for generationIndex in range(0, generations):
            #Se calcula el fitness de la nueva generacion
            self.calculateFitnessPopulation(population)

            #Se realiza la selección
            population=self.selectionAlgorithm.select(population).
                   copy()

            #Se crea una copia del mejor individuo solución
            if self.bestFitness>population[0].fitness:

                self.bestFitness= population[0].fitness
                self.bestIndividual = copy.deepcopy(population[0])

            #Se realiza el cruzamiento
            population=self.crossAlgorithm.crossPopulation(population
                 , populationQuantity,self.environment).copy()

            #Se realiza la mutación
            population=self.mutationAlgorithm.mutationPopulation(
                 population,self.environment).copy()
            self.calculateFitnessPopulation(population)

          #Se verifica si el individuo satisface la condicion de fin
            if self.bestFitness==self.aptitudeThreshold:
                self.bestIndividual.printIndividual()
                 break
            else:
                #Se realiza la mejora de los individuos
                self.improvementPopulation(population,
                   self.environment)

            #Método de implementación opcional, se ejecuta al
            terminar         una generación
            self.finishActions.runFinishGenerationBlock(population,
                 self.bestFitness,self.bestIndividual)

        #Metodo que se ejecuta al encontrar la solución o terminar
        las generaciones indicadas
        self.finishActions.runFinishRunBlock(population,
             self.bestFitness,self.bestIndividual)
```

#### 3.2.3.2 Codificación de la Solución para Horarios Académicos

En esta sección se muestran fragmentos de código de las clases complementarias que fueron necesarias para crear la solución con algoritmos evolutivos de un horario académico, se toma como referencia el diagrama de clases de la Figura 24 para la creación de estas.

- **ErrorWeek**: esta clase ayuda a recopilar los errores presentados de una manera más ordenada y comprensible tanto para el usuario si se quiere visualizar con el método printError, como para el desarrollador al momento de corregirlos con algún algoritmo automático.

```
class ErrorWeek():
    typeError:int
    teacher :str
    day:WeekDay
    weeklyHours:int
    assignedHours:int
    subject:Subject

    def printError(self):
        errorCompleto=" typeError:"+str(self.typeError)
        errorCompleto+=" subject:"+self.subject.identifier
        errorCompleto+=" teacher :"+self.teacher
        errorCompleto+=" day:"+self.day.name
        errorCompleto+=" Maximum hours:"+str(self.weeklyHours)
        errorCompleto+=" assignedHours:"+str(self.assignedHours)
        print(errorCompleto)
```

- **Subject**: esta clase hace referencia a la asignatura de un curso en específico, siguiendo el análisis que se realiza previamente del requerimiento se entiende que se deben agregar los parámetros horas mínimas consecutivas, horas máximas consecutivas y horas semanales que posee la misma. Además de esto, la clase por medio de la generalización hereda los atributos y comportamientos de Resource lo cual facilita al desarrollador la asignación o vinculación de estas con otro recurso como lo es el profesor.

```
class Subject(Resource):
    weeklyHours = 0
    horasMinimasCons = 0
    maxConsecutiveHours = 0

    def __init__(self, identifier: str, hsSemanales: int,
      hsMinimasCons: int, hsMaximasCons: int):
        super(Subject,self).__init__(identifier)
        self.weeklyHours = hsSemanales
        self.horasMinimasCons = hsMinimasCons
        self.maxConsecutiveHours = hsMaximasCons
```

- **EnvironmentSchool**: en esta clase es donde se definen las variables que se conocen desde el inicio y que conforman el escenario inicial del problema, para este caso se separa en varios fragmentos de código que muestran todo el manejo de recursos que se debe crear y actualizar a lo largo del proceso.

  A continuación, muestra la creación de los profesores usando la clase ResourceTime y su respectiva asignación de disponibilidad, esta se realiza con una función utilitaria que permite la creación de un objeto TimeTable que crea un horario de lunes a viernes de 7 a 13 el cual es el utilizado para las clases.

```
class EnvironmentSchool(GeneralEnvironment):
    def __init__(self):
        super(EnvironmentSchool,self).__init__()
        juan = ResourceTime("Juan")
        juan.availability = monday_to_friday_from_7_to_13()
        pedro = ResourceTime("Pedro")
        pedro.availability = monday_to_friday_from_7_to_13()
        diana = ResourceTime("Diana")
        diana.availability = monday_to_friday_from_7_to_13()
        lucas = ResourceTime("Lucas")
        lucas.availability = monday_to_friday_from_7_to_13()
        martha = ResourceTime("Martha")
        martha.availability = monday_to_friday_from_7_to_13()
        luis = ResourceTime("Luis")
        luis.availability = monday_to_friday_from_7_to_13()
        nora = ResourceTime("Nora")
        nora.availability = monday_to_friday_from_7_to_13()
        paola = ResourceTime("Paola")
        paola.availability = monday_to_friday_from_7_to_13()
```

Seguido de esto se presenta la creación de las asignaturas y su vinculación al profesor y al curso respectivo, mediante la clase Subject (Subsección 3.2.2.8). Para poder evidenciar esto, a continuación, se muestra el fragmento de código para el curso primero.

```
class EnvironmentSchool(GeneralEnvironment):

    def __init__(self):
        # MateriasCursoPorCurso
        # Primero
        edFisica1 = Subject("Ed Fisica1", 2, 1, 2)
        edFisica1.addResourceLinked("Profesor", juan)
        mate1 = Subject("Matematicas1", 4, 1, 2)
        mate1.addResourceLinked("Profesor", pedro)
        dibujo1 = Subject("Dibujo1", 5, 1, 2)
        dibujo1.addResourceLinked("Profesor", diana)
        danza1 = Subject("Danza1", 4, 1, 2)
        danza1.addResourceLinked("Profesor", juan)
        ciencias1 = Subject("Ciencias1", 2, 1, 2)
        ciencias1.addResourceLinked("Profesor", nora)
        lengua1 = Subject("Lengua1", 3, 1, 2)
        lengua1.addResourceLinked("Profesor", lucas)
        sis1 = Subject("Sistemas1", 1, 1, 1)
        sis1.addResourceLinked("Profesor", luis)
        musica1 = Subject("Musica1", 3, 1, 2)
        musica1.addResourceLinked("Profesor", paola)
        canto1 = Subject("Canto1", 3, 1, 2)
        canto1.addResourceLinked("Profesor", paola)
        ingles1 = Subject("Ingles1", 3, 1, 2)
        ingles1.addResourceLinked("Profesor", lucas)
        primero = Resource("1")
        primero.addResourceLinked("Subject", edFisica1)
        primero.addResourceLinked("Subject", mate1)
        primero.addResourceLinked("Subject", dibujo1)
        primero.addResourceLinked("Subject", danza1)
        primero.addResourceLinked("Subject", ciencias1)
        primero.addResourceLinked("Subject", lengua1)
        primero.addResourceLinked("Subject", sis1)
        primero.addResourceLinked("Subject", musica1)
        primero.addResourceLinked("Subject", canto1)
        primero.addResourceLinked("Subject", ingles1)
```

Finalmente, la clase EnvironmetSchool implementa el método abstracto updateEnvironment que actualiza los recursos que se van asignando cada que se termina una subsección del problema y que lo utiliza la clase StandardSolver. Además de esto, se crea el método getEnvironmentByCourse que permite crear estas subsecciones o divisiones de escenarios (para este caso están planteados por curso), y se encarga de asignar una disponibilidad para el curso, en este caso se usa la misma función utilitaria que crea el TimeTable de la semana escolar de 7 a 13.

```
 def getEnvironmentByCourse(self) -> List[EnvironmentTime]:
        cursos: List[Resource] = self.resources["cursos"]
        ambientes: List[EnvironmentTime] = list()
        for curso in cursos:
            ambienteCurso = EnvironmentTime()
            ambienteCurso.resources = curso.linkedResources
            ambienteCurso.timetable = monday_to_friday_from_7_to_13()
            ambientes.append(ambienteCurso)
        return ambientes

def updateEnvironment(self, newEnvironment:GeneralEnvironment):
   for subject in newEnvironment.resources["Subject"]:
       profesorNuevo: ResourceTime =
            subject.linkedResources["Profesor"][0]
       materiaOld = self.getResourceByTypeAndID("Subject",
            subject.identifier)
       profesorViejo: ResourceTime = materiaOld.
             linkedResources["Profesor"][0]
       timeTableNueva = profesorViejo.availability.intersection(
             profesorNuevo.availability)
       profesorViejo.availability = TimeTable(
             timeTableNueva._open_slots)
```

- **SchoolWeek**: esta es una de las clases más importantes ya que se trata de la representación del individuo solución, se visualizan los fragmentos de código separados por funcionalidades que permiten entender el cómo se vincula con las iteraciones que lleva a cabo el algoritmo evolutivo. Esta primera parte del código muestra la herencia creada a la clase IndividualTime y la inicialización de los atributos, es importante notar que el environment ayuda a que el individuo posea una copia de las variables del escenario en todo momento.

```
# Clase que hereda comportamiento de IndividualTime
class SchoolWeek(IndividualTime):

    def __init__(self, environment: EnvironmentTime):
        self.environment: EnvironmentTime =
            copy.deepcopy(environment)
        self.fitness = 0
        self.mistakes = []
        super(SchoolWeek, self).__init__(self.environment)
```

El primer paso en los algoritmos evolutivos es crear una población inicial, y en este caso usaremos la función createRandomIndividual para ese propósito, esta función recibe el ambiente o escenario que contiene todos los atributos iniciales y crea los individuos con asignaciones al azar con la ayuda de la función completeCourses.

```
def createRamdomIndividual(self, ambienteNuevo: EnvironmentTime)
   -> Individual:
   newAux = SchoolWeek(ambienteNuevo)
   newAux.completeCourses(newAux.environment.resources["Subject"])
   return newAux
def completeCourses(self, subjects: List[Subject]):
    subjectCounter = 0
    while(subjectCounter < len(subjects)):
       subject: Subject = subjects[subjectCounter]
       subjectCounter += 1
       weeklyHours = subject.weeklyHours
       assignedHours = 0
       while weeklyHours > assignedHours:
           teacher : ResourceTime = subject.
               linkedResources["Profesor"][0]
           scheduleAvailable: TimeTable = teacher.availability.
              intersection(self.timetable.timetable)
           time: Optional[TimeSlot] = scheduleAvailable.
             any_time_slot()
           if time is not None:
               newAssignment: Assignment = Assignment()
               newAssignment.timeSlot = time
               newAssignment.listResourceId.append(subject.
                   identifier)
               newAssignment.listResourceId.append(teacher.
                   identifier)
               self.timetable.addAssignment(time, newAssignment)
               teacher .availability._open_slots.remove(time)
               assignedHours += 1
           else:
               break
```

Cada individuo que es creado debe ser posteriormente evaluado para conocer el nivel de aptitud que tiene para ser una posible solución, para esto se implementa la función calculateFitness la cual inicializara los errores en cada iteración, y con la ayuda de la función evaluateHourByCourse recolecta la información necesaria para identificar los errores del individuo, para este caso en particular se evidencia errores de horas consecutivas superiores para una materia o horas semanales diferentes a las estipuladas en el escenario inicial, las mismas son identificadas y agregadas al arreglo de errores.

```
def calculateFitness(self):
    self.fitness = 0
    self.mistakes.clear()
    self.evaluateHoursByCourse()

def evaluateHoursByCourse(self):
    for subject in self.environment.resources["Subject"]:
        assignments: List[Assignment] = self.timetable.
            getAssignmentsByResource(subject.identifier)
        for day in self.environment.timetable.week_days():
            assignmentByDay = list(filter(lambda assignment:
                 assignment.timeSlot.week_day == day, assignments))
            hoursDay = len(assignmentByDay)
            if hoursDay > subject.maxConsecutiveHours:
                self.fitness += 1
                error = ErrorWeek(1, subject)
                error.day = day
                error.assignedHours = hoursDay
                error.weeklyHours = subject.maxConsecutiveHours
                self.mistakes.append(error)
        if subject.weeklyHours != len(assignments):
            error = ErrorWeek(0, subject)
            error.assignedHours = len(assignments)
            error.weeklyHours = subject.weeklyHours
            self.fitness += abs(subject.weeklyHours-
                    len(assignments))
            self.mistakes.append(error)
```

Seguido de esto los individuos deben pasar por las etapas del algoritmo evolutivo, para el horario académico se usa la selección de ranking, la cual toma los mejores individuos de cada iteración por medio del conteo de errores que posee el individuo y el valor de fitness más cercano a 0, este código no hace parte del individuo ya que se usa un código genérico que provee el marco de trabajo.

Para el cruce de individuos se realiza una implementación específica a los horarios académicos que permite actualizar las diferentes disponibilidades de profesores, asignaturas y realizar nuevamente el cálculo de la función de aptitud, para eso se muestra a continuación el fragmento de código desarrollado que realiza este trabajo.

```
def cross(self, couple: IndividualTime,
    ambienteNuevo: EnvironmentTime) -> List[Individual]:
    newWeek1: SchoolWeek = SchoolWeek(ambienteNuevo)
    newWeek2: SchoolWeek = SchoolWeek(ambienteNuevo)

    for time in self.timetable.dataByTimeSlot:
        if random.uniform(0.0, 1.0) <= 0.5:
           if time in self.timetable.dataByTimeSlot:
              newAssignment1: Assignment = self.timetable.
                  dataByTimeSlot[time]
              newWeek1.addCrossAssignment(newAssignment1)
           if time in couple.timetable.dataByTimeSlot:
              newAssignment2: Assignment = couple.timetable.
                  dataByTimeSlot[time]
              newWeek2.addCrossAssignment(newAssignment2)
         else:
           if time in self.timetable.dataByTimeSlot:
              newAssignment2: Assignment = self.timetable.
                  dataByTimeSlot[time]
              newWeek2.addCrossAssignment(newAssignment2)
           if time in couple.timetable.dataByTimeSlot:
              newAssignment1: Assignment = couple.timetable.
                  dataByTimeSlot[time]
              newWeek1.addCrossAssignment(newAssignment1)

    weeks: list = []
    weeks.append(newWeek1)
    weeks.append(newWeek2)
    return weeks

def addCrossAssignment(self, newAssignment: Assignment):
    self.timetable.addAssignment(
        newAssignment.timeSlot, copy.deepcopy(newAssignment))
    subject: Subject = self.environment.getResourceByTypeAndID(
        "Subject", newAssignment.listResourceId[0])
    teacher: ResourceTime = subject.linkedResources["Profesor"][0]
    teacher.availability._open_slots.remove(newAssignment.timeSlot)
```

La mutación del individuo se realiza usando un concepto genérico el cual, al azar selecciona una de las asignaciones y la reemplaza por otra seleccionada aleatoriamente, en este caso se toma una clase en específico y se cambia por otra que puede ser impartida el mismo día.

Finalmente, para poder generar mejores resultados en menores generaciones y evitar posibles bloqueos y mínimos locales, se realiza la implementación de la función improvement, esta función evalúa los errores, selecciona 1 error al azar y trata de repararlo a través de mutación selectiva, cambiando la asignación errónea por otra que puede ser dictada el mismo día, esto ayuda a reducir la cantidad de errores en cada iteración y permite visualizar los errores que se presentan con mayor frecuencia. Para esto se muestra en la parte inferior un fragmento de código que permite este cambio en el individuo.

```
def improvement(self, environment) -> Individual:
    self.fixRandomErrors(environment)

def fixRandomErrors(self, environment):
    if len(self.mistakes) > 0:
       # Selección del error al azar
       error: ErrorWeek = random.choice(self.mistakes)
       if isinstance(error, ErrorWeek):
           if error.typeError == 0:
              if error.weeklyHours > error.assignedHours:
                  subject: Subject = error.subject
              else:
                  if error.weeklyHours < error.assignedHours:
                      subject = error.subject
           elif error.typeError == 1:
              self.dailyAllowance(error, environment)
           elif error.typeError == 2:
               dayTemp = error.day
```

```
def dailyAllowance(self, error: ErrorWeek, environment):
    subject = error.subject
    teacher = environment.getProfesoresByCursada(subject)[0]
    # Evaluando los errores mas frecuentes y cambiando la asignacion
    if error.weeklyHours > error.assignedHours:
       day = self.getDia(error.day)
       day.replaceAleatoriaCursada(subject, environment)
       error.assignedHours += 1
    else:
       if error.weeklyHours < error.assignedHours:
           day = self.getDia(error.day)
           day.removerCursada(subject, environment)
           error.assignedHours -= 1
```

- **Main**: Función principal que se ejecuta cuando inicia el programa, este archivo es normalmente llamado principal “main” por su traducción en inglés, en esta se definen una función llamada “run” que contiene la creación y asignación de los atributos que tendrá el algoritmo evolutivo y es el encargado de iniciar la ejecución del StandarSolver.

```
def run():
    #ranking del 30% de la poblacion
    selection: SelectionAlgorithm = SelectionRanking(0.3)
    cross: CrossAlgorithm = CrossSpecificIndividual()
    mutation: MutationAlgorithm = MutateRandomPoints()
    finishAction: FinishAction = FinishBestIndividualImpl()
    environment: EnvironmentSchool = EnvironmentSchool()
    #Se generan los ambientes separados por curso
    ambientesPorCurso = environment.getEnvironmentByCourse()
    EA = EvolutionaryAlgorithmManager()
    EA.selectionAlgorithm = selection
    EA.crossAlgorithm = cross
    EA.mutationAlgorithm = mutation
    EA.finishActions = finishAction
    admin: StandardSolver = StandardSolver(
             environments=ambientesPorCurso,
            iterations=1000,
            finalFitness=0,
            quantityIndividuals=20,
            evolutionaryAlgorithm=EA,
            classReference=SchoolWeek)
    admin.runAlgorithm()
```

### 3.2.4 Ejecución y Resultados

En esta sección se muestra la ejecución y resultado del código (subsección 3.2.4.1) y un ejemplo de horario solución (subsección 3.2.4.1).

#### 3.2.4.1 Ejecución y Resultados

El código se elabora en una computadora con las especificaciones mencionadas en la sección 1.3.3. Se realiza una toma de 40 ejecuciones de las cuales 20 usan una disponibilidad completa para los profesores (Tabla 7) y 20 restantes agregan restricciones que dificultan el encontrar la solución (Tabla 8).

De las ejecuciones del programa se capturan los siguientes datos: velocidad en encontrar la solución (medida en segundos), validación de la solución (Si, cuando encuentra y No, cuando no la solución), cantidad de errores que obtuvo cuando no encontró una solución y una descripción del error si se presenta.

*Tabla 7 Validación sin restricciones horarias*

| No | Velocidad | Solución | N° Errores | Descripción del error |
|---|---|---|---|---|
| **1** | 4.7s | Si | 0 | |
| **2** | 5.02s | Si | 0 | |
| **3** | 4.07s | Si | 0 | |
| **4** | 4.92s | Si | 0 | |
| **5** | 4.66s | Si | 0 | |
| **6** | 4.69s | Si | 0 | |
| **7** | 4.91s | Si | 0 | |
| **8** | 5.49s | Si | 0 | |
| **9** | 6.02s | Si | 0 | |
| **10** | 4.89s | Si | 0 | |
| **11** | 4.07s | Si | 0 | |
| **12** | 5.10s | Si | 0 | |
| **13** | 6.03s | Si | 0 | |
| **14** | 4.95s | Si | 0 | |
| **15** | 4.83s | Si | 0 | |
| **16** | 5.23s | Si | 0 | |
| **17** | 5.37s | Si | 0 | |
| **18** | 6.24s | Si | 0 | |
| **19** | 5.75s | Si | 0 | |
| **20** | 4.67s | Si | 0 | |

La anterior tabla (Tabla 7) muestra los resultados obtenidos al ejecutar el código manteniendo restricciones de disponibilidad mínimas para los profesores, en ella se puede observar que el algoritmo evolutivo está respondiendo de manera eficiente, encontrando la solución en un tiempo alrededor de los 5 segundos y cumpliendo todas las restricciones del problema original.

*Tabla 8 Validación con restricciones horarias*

| No | Velocidad | Solución | N° Errores | Descripción del error |
|---|---|---|---|---|
| **1** | 193.92s | Si | 0 | |
| **2** | 211.57s | No | 1 | Se asignaron 3 horas de Ingles para el curso cuarto, y se piden 4 |
| **3** | 4.15s | Si | 0 | |
| **4** | 214.39s | Si | 0 | |
| **5** | 4.75s | Si | 0 | |
| **6** | 4.17s | Si | 0 | |
| **7** | 211.96s | Si | 0 | |
| **8** | 4.48s | Si | 0 | |
| **9** | 4.39s | Si | 0 | |
| **10** | 210.59 | No | 1 | Se asignaron 3 horas consecutivas para Lengua del curso quinto y el máximo es 2 |
| **11** | 618.83s | Si | 0 | |
| **12** | 210.07s | Si | 0 | |
| **13** | 4.39s | Si | 0 | |
| **14** | 208.27s | No | 1 | Se asignaron 3 horas de Ingles para el curso cuarto, y se piden 4 |
| **15** | 252.02s | Si | 0 | |
| **16** | 421.99s | Si | 0 | |
| **17** | 212.21s | No | 1 | Se asignaron 3 horas de Ingles para el curso cuarto, y se piden 4 |
| **18** | 144.52s | Si | 0 | |
| **19** | 205.93 | Si | 0 | |
| **20** | 192.42s | No | 1 | Se asignaron 3 horas de Ingles para el curso cuarto, y se piden 4 |

La anterior tabla (Tabla 8) muestra la ejecución del horario académico añadiendo restricciones de disponibilidad para profesores con poca carga académica, lo cual agrega complejidad al algoritmo lo cual se puede evidenciar en los resultados obtenido:

- El tiempo de ejecución fue altamente incrementado llegando a un máximo de 618 segundos alrededor de 10 minutos para obtener una solución al problema.
- En el 25% de los casos el algoritmo no está llegando a una solución que cumpla con todos los criterios de aceptación pedidos por el escenario inicial.
- Se evidencia que la asignación al azar genera que algunas ejecuciones lleguen a una solución rápidamente (5 segundos) y que existen caminos críticos en donde los profesores con menos disponibilidad son asignados al final sin poder corregirlos en las iteraciones programadas.

#### 3.2.4.2 Ejemplo Horario Solución

En esta subsección se muestra un ejemplo de horario solución para el problema planteado y generado por el código. Este horario esta agrupado por curso y muestra las asignaturas que son impartidas en cada una de las horas indicada.

*Tabla 9 Horario Primero*

| **Horario grado 1** | | | | | |
|---|---|---|---|---|---|
| | **Lunes** | **Martes** | **Miércoles** | **Jueves** | **Viernes** |
| **7 a 8** | Ed. Física | Canto | Danza | Dibujo | Ingles |
| **8 a 9** | Ed. Física | Matemáticas | Ingles | Dibujo | Matemáticas |
| **9 a 10** | Matemáticas | Sistemas | Música | Lengua | Música |
| **10 a 11** | Dibujo | Música | Dibujo | Matemáticas | Danza |
| **11 a 12** | Danza | Danza | Dibujo | Ingles | Canto |
| **12 a 1** | C. Naturales | Lengua | Lengua | C. Naturales | Canto |

*Tabla 10 Horario Segundo*

| Horario grado 2 | | | | | |
|---|---|---|---|---|---|
| | **Lunes** | **Martes** | **Miércoles** | **Jueves** | **Viernes** |
| **7 a 8** | Lengua | Lengua | Ingles | Sistemas | Danza |
| **8 a 9** | Matemáticas | Música | Dibujo | Sistemas | Danza |
| **9 a 10** | Ed. Física | Música | Dibujo | Matemáticas | Matemáticas |
| **10 a 11** | Ed. Física | Matemáticas | Lengua | Dibujo | Música |
| **11 a 12** | Dibujo | Canto | Danza | C. Naturales | C. Naturales |
| **12 a 1** | Dibujo | Canto | Música | Ingles | Ingles |

*Tabla 11 Horario Tercero*

| Horario grado 3 | | | | | |
|---|---|---|---|---|---|
| | **Lunes** | **Martes** | **Miércoles** | **Jueves** | **Viernes** |
| **7 a 8** | Dibujo | Ed. Física | Matemáticas | C. Naturales | Canto |
| **8 a 9** | Dibujo | Ed. Física | Danza | C. Naturales | Canto |
| **9 a 10** | C. Naturales | Ingles | Ingles | Sistemas | Lengua |
| **10 a 11** | Lengua | Sistemas | Canto | Ingles | Matemáticas |
| **11 a 12** | Matemáticas | Sistemas | Lengua | Dibujo | Danza |
| **12 a 1** | Matemáticas | Matemáticas | Dibujo | Física | Química |

*Tabla 12 Horario Cuarto*

| Horario grado 4 | | | | | |
|---|---|---|---|---|---|
| | **Lunes** | **Martes** | **Miércoles** | **Jueves** | **Viernes** |
| **7 a 8** | Matemáticas | Matemáticas | Dibujo | Lengua | Química |
| **8 a 9** | Lengua | Lengua | Física | Lengua | Lengua |
| **9 a 10** | Lengua | Ed. Física | Matemáticas | Dibujo | Ingles |
| **10 a 11** | C. Naturales | Ed. Física | Matemáticas | C. Naturales | Ingles |
| **11 a 12** | C. Naturales | Física | Ingles | Sistemas | Matemáticas |
| **12 a 1** | Danza | Sistemas | Ingles | Sistemas | Danza |

*Tabla 13 Horario Quinto*

| Horario grado 5 | | | | | |
|---|---|---|---|---|---|
| | **Lunes** | **Martes** | **Miércoles** | **Jueves** | **Viernes** |
| **7 a 8** | C. Naturales | Sistemas | Ingles | Matemáticas | Ingles |
| **8 a 9** | C. Naturales | Sistemas | Ingles | Matemáticas | Ingles |
| **9 a 10** | Dibujo | Matemáticas | Ed. Física | C. Naturales | Danza |
| **10 a 11** | Matemáticas | Lengua | Ed. Física | Sistemas | Química |
| **11 a 12** | Lengua | Lengua | Matemáticas | Física | Lengua |
| **12 a 1** | Lengua | Danza | Física | Dibujo | Física |

## 3.3 VALIDACION DE MARCO DE TRABAJO

En esta sección se realiza una validación cualitativa del marco de trabajo, tomando el desarrollo de la asignación de horarios como ejemplo de evaluación. Los ítems que son evaluados son: curva de aprendizaje, complejidad de desarrollo, utilidad del código provisto, resultados de la solución del problema y los mismos son evaluados con Muy Bajo, Bajo, Medio, Alto, Muy Alto.

- **La curva de aprendizaje:**

  Este ítem tiene una dificultad media (Medio) debido al tiempo que se emplea para comprender las funcionalidades, la jerarquización y las utilidades que sirven para generar los escenarios y el individuo solución necesario para el problema específico.

- **Complejidad de desarrollo:**

  Después de comprender el marco de trabajo, el desarrollo es creado de una manera más fácil, aprovechando las utilidades y generalizaciones que proveen una estructura para la solución. La dificultad de este es de un nivel medio (Media) ya que si bien el marco provee utilidades que facilitan el planteamiento del problema, cada problema específico debe plantear el posible individuo solución y dependiendo la dificultad codificar cruces o mutaciones que le permitan de una mejor manera llegar a la solución.

- **Utilidad del código provisto:**

  La utilidad del código provisto es de nivel alto (Alto), esto debido a que la jerarquía de los recursos, la estructuración del código, las generalizaciones y las utilidades del manejo del tiempo o espacio, permiten de una manera más sencilla buscar espacios disponibles para asignar, cambiar los parámetros del algoritmo evolutivo de una manera sencilla, agregar nuevos algoritmos de selección, mutación o cruce si el problema lo requiere, etc.

- **Resultados de la solución del problema:**

  En los resultados de la solución del problema se obtiene una calificación alta (Alto), ya que se logra encontrar la solución al problema de ejemplo usado, pero al agregar una complejidad mucho mayor se encuentra que un gran porcentaje de las ejecuciones no llegan a una solución completa, se plantea en trabajos futuros realizar una mejora de este que ayude a mantener el rango de ejecuciones incompletas por debajo del 10%.

# CAPITULO 4. CONCLUSIONES

Los problemas de asignación de recursos en espacios establecidos poseen una alta complejidad al generar un modelo con una solución simple que pueda ser implementada mediante un sistema de software tradicional. Es por ello, que para este tipo de problemas puede ser usada una de las principales técnicas de la metaheurística como lo son los algoritmos evolutivos.

El presente trabajo diseña, implementa y verifica un marco de trabajo que disminuye el tiempo invertido por el desarrollador para plantear, diseñar y ejecutar un problema con algoritmos evolutivos. Para esto, como primer paso, se analizan investigaciones similares que dan solución a problemas específicos usando este tipo de algoritmos, lo cual permite identificar los objetos generales que interactúan en este y la gran cantidad de código que debe ser realizado para cada solución.

El marco de trabajo contiene las clases generales que describen a un algoritmo evolutivo y otras que estructuran al usuario para generar el modelo de solución. Esto permite disminuir el tiempo de codificación y plantear los problemas de una forma estructurada, además de esto las utilidades presentadas en las clases ayudan a tener un mejor control de las disponibilidades de los recursos. Al extender el marco de trabajo codificado en Python para dar solución al problema de horarios de una institución académica, se observa que el uso del marco posee una curva de aprendizaje con un punto alto al inicio que puede retrasar el desarrollo de la solución pero que al entender sus funcionalidades permite de una manera sencilla extender las funcionalidades básicas y adaptarse al problema.

Al validar el marco de trabajo con los resultados obtenidos del problema de horarios académicos, se concluye que:

- El código extendido es capaz de encontrar la solución a un problema complejo de asignación, sacando el mejor provecho del marco de trabajo.

- Al agregar restricciones que limiten los recursos en asignaciones específicas, el algoritmo tiene grandes probabilidades de fallo al realizar por defecto una asignación aleatoria.

- El marco de trabajo logra cumplir el objetivo de disminuir el tiempo de planeación y codificación encontrando una solución a un problema específico.

Finalmente, como futuras líneas de trabajo se prevé agregar funcionalidades que permitan crear una priorización a las asignaciones, para evitar en mayor medida que el individuo solución se guie hacia un mínimo o máximo local sin lograr cumplir en totalidad la solución. Por otro lado, se pretende evaluar el algoritmo en problemas de otros dominios que puedan aportar más información y generalidades al marco de trabajo.

# BIBLIOGRAFIA

Barbosa Sierra, J. (2021). *Asignación Algoritmos Evolutivos*. Obtenido de https://github.com/julbar22/asignacionAE.

Bueno, M. E., Dos Reis, M., Illescas, G., Tripodi, G., Vallejos, I., & Casariego, I. (2018). Conocimiento en acción: Asignación de recursos a familias carentes mediante la aplicación de un algoritmo genético - Proyecto Koinonía. *Revista De La Escuela De Perfeccionamiento En Investigación Operativa, 19*(32), 183–205.

Chiong, R., Weise, T., Michalewicz, Z., Blum, C., Clerc, M., De Jong, K., & Neri, F. (2012). Evolutionary Optimization. *Variants of Evolutionary Algorithms for Real-World Applications*, 1-29.

Davarynejad, M., Vrancken, J., Berg, J., & Coello Coello, C. (2012). A Fitness Granulation Approach for Large-Scale Structural Design Optimization. *Variants of Evolutionary Algorithms for Real-World Applications*, 245-280.

Duarte Muñoz, A. (2007). *Metaheurísticas.* Madrid: DYKINSON S.L.

Gutiérrez, J. (2014). *¿ Qué es un framework web?* Obtenido de http://www.lsi.us.es/~javierj/investigacion_ficheros/Framework.pdf

Joyanes Aguilar , L., Castilo Sanz, A., Sánchez García, L., & Zahonero Martínez, I. (2005). *C Algoritmos, programación y estructuras de datos.* España: Algoritmos, programación y estructuras de datos.

Kuri Morales, A., & Galaviz Casas, J. (2002). *Algoritmos Geneticos.* México: Talleres Gráficos de la Dirección de Publicaciones del Inst. Politécnico Nacional.

Minjares Lugo, J. L., Salmón Castelo, R. F., Oroz Ramos, L. A., & Cruz Medina, I. R. (2008). Modelo hidrológico-agronómico-económico para la operación óptima del sistema de presas del río Yaqui, usando algoritmos genéticos. *Revista interdisciplinaria de ciencia y tecnologiia del agua, 23*(3).

Osman, I., & Kelly, J. (1996). *Meta-Heuristics: Theory and Applications.* Norwer, Massachusetts: Kluwer Academic Publishers Group.

Petrowski, A., & Ben-Hamida, S. (2017). *Evolutionary Algorithms.* London, UK: ISTE Ltd.

Pradenas Rojas, L., & Matamala Vergara, E. (2012). Una formulación matemática y de solución para programar cirugías con restricciones de recursos humanos en el hospital público. *Revista chilena de ingeniería, 20*(2), 230-241.

Python. (2021). *1. Abriendo el apetito — documentación de Python - 3.8.8*. Obtenido de Docs.python.org: https://docs.python.org/es/3.8/tutorial/appetite.html

Sivanandam, S., & Deepa, S. (2008). *Introduction to Genetic Algorithms.* New York: Springer.

Solano Sabatier, Y., Calvo Marín, M., & Trejos Picado, L. (2008). Implementación de un algoritmo genético para la asignación de aulas en un centro de estudio. *Uniciencia 22*, 115-121.

Staruml. (2021). *Introduction - staruml*. Obtenido de https://docs.staruml.io/

Viñas, S. E., Rodríguez, N. E., Corona, E., & Jiménez, A. J. (2019). *Software para la generación automática de horarios académicos.* México: Avances en Ciencias e Ingeniería.

Visual Studio Code. (2021). *Getting Started - Visual Studio Code*. Obtenido de https://code.visualstudio.com/docs

Yu, X., & Gen, M. (2010). *Introduction to oEvolutionary Algorithms.* Springer.