M.O.E.A. Software Documentation

Release 1.0

Aarón Martín Castillo Medina

Oct 12, 2016

CHAPTER 1

Indices and tables

- genindex
- modindex
- search

CHAPTER 2

Contents:

2.1 Begin (script)

Este archivo funge como un launcher **(disparador)** el cual simplemente crea y muestra la ventana principal **(véase la sección View)**.

2.2 Model (sección)

La sección Model **(ó Modelo)** contiene toda la base lógica del programa, más en específico, todas las características para poder ejecutar MOEA's apropiadamente alimentados con los datos obtenidos por la sección View **(ó Vista)** usando la sección Controller **(ó Controlador)** como intermediaria. Una vez que se obtenga algún resultado, éste será transmitido a la sección View a través del Controller.

2.2.1 ChromosomalRepresentation (módulo)

Ofrece elementos para elaborar una codificación adecuada.

Entiéndase por codificación a la forma de determinar el cromosoma y sus propiedades; cabe mencionar que el cromosoma será portado por los Individuals **(ó Individuos)**.

Es importante mencionar que cualquier codificación implementada debe ser sustentada en los métodos correspondientes al Crossover **(ó Cruza)** y Mutation **(ó Mutación)**, ésto porque dichas operaciones funcionan con cromosomas.

De esta manera, la idea es que el usuario pueda crear sus propias codificaciones, por lo que, además de agregar la descripción de la codificación a Controller/XML/Features.xml **(véase el archivo mencionado en la sección de código)**, deberá implementar por lo menos las siguientes funciones:

calculate_length_subchromosomes(vector_variables,number_of_decimals,representation_paramete

Por cada variable de decisión se crea una porción del cromosoma, entonces en esta función se calcula el tamaño de cada porción **(ó subcromosoma)**, ya que al final las operaciones de cruza y mutación se realizarán sobre el súper crosomoma, el cual es la concatenacion de todos los subcromosomas. Por eso es importante identificar el tamaño de cada subcromosoma, así como sus límites dentro del súper cromosoma.

Parameters

- **vector_variables** (*List*) – El vector de variables de decisión, donde cada variable trae consigo sus límites inferior y superior.
- **number_of_decimals** (*Integer*) – El número de decimales que deberá traer cada variable de decisión.

- **representation_parameters** (*Dictionary*) – Un diccionario que contiene todas las opciones adicionales para cada tipo de codificación.

Returns Una lista que contiene el tamaño del cromosoma por cada variable de decisión. Dado que el orden de las variables de decisión es inmutable, se preserva el mismo y por ello la lista contiene sólo los tamaños.

Return type List

create_chromosome(length_subchromosomes, vector_variables, number_of_decimals, representation_

Función que crea el cromosoma. Se usa la como apoyo el método **calculate_length_subchromosomes** descrito con anterioridad.

Parameters

- **length_subchromosomes** (*List*) – La lista que contiene los tamaños de cada variable de decisión.
- **vector_variables** (*List*) – La lista que contiene las variables de decisión, así como sus rangos.
- **number_of_decimals** (*Integer*) – El número de decimales que traerá cada variable de decisión.
- **representation_parameters** (*Dictionary*) – Un diccionario que contiene todas las opciones adicionales para cada tipo de codificación.

Returns El cromosoma devuelto en forma de lista.

Return type List

evaluate_subchromosomes(complete_chromosome, length_subchromosomes, vector_variables, number_

Tomando en cuenta que el cromosoma ya ha sido creado usando los tamaños de los subcromosomas, en esta función se procede a evaluar el súper cromosoma partiéndolo en los subcromosomas pertinentes y evaluando individualmente cada uno de éstos.

Parameters

- **complete_chromosome** (*List*) – El súper cromosoma a ser evaluado.
- **length_subchromosomes** (*List*) – La lista que contiene los tamaños de cada variable de decisión.
- **vector_variables** (*List*) – La lista que contiene las variables de decisión, así como sus rangos.
- **number_of_decimals** (*Integer*) – El número de decimales que traerá cada variable de decisión.
- **representation_parameters** (*Dictionary*) – Un diccionario que contiene todas las opciones adicionales para cada tipo de codificación.

Returns Un diccionario que contiene como llave la variable de decisión y como valor la evaluación del subcromosoma correspondiente.

Return type Dictionary

BinaryRepresentation (script)

Contiene todas las funcionalidades requeridas para que se pueda hacer uso de una codificación de tipo Binary **(ó Binaria)**; ésto significa que los alelos que conforman al cromosoma serán exclusivamente 0 ó 1.

binary_to_decimal(*chromosome*)

Método que convierte un número binario a decimal.

Este es un ejemplo de método que se puede agregar adicionalmente siempre y cuando se implementen las funciones que se han mencionado ya.

Parameters **chromosome** (*List*) – El cromosoma sobre el cual se hará la evaluación.

Returns La representación en decimal del número binario.

Return type Integer

calculate_length_subchromosomes(*vector_variables*, *number_of_decimals*, *representation_parameters*)

Esta es la implementación del método para la codificación en binario. A grandes rasgos primero se determina el número de bits que se deben tomar en cuenta para representar la magnitud de una determinada variable de decisión.

Haciendo esto para todas las variables de decisión se obtienen las longitudes de todos los subcromosomas.

Esta función se implementa obligatoriamente.

create_chromosome(*length_subchromosomes*, *vector_variables*, *number_of_decimals*, *representation_parameters*)

Crea un cromosoma binario completo con base en las longitudes de los subcromosmas.

Este método debe implementarse obligatoriamente.

evaluate_subchromosomes(*complete_chromosome*, *length_subchromosomes*, *vector_variables*, *number_of_decimals*, *representation_parameters*)

Realiza una evaluación de los subcromosomas para la codificación binaria **(ó Binary)**.

En términos generales se toma cada porción del subcrosomoma **(tomando en cuenta que previamente se calcularon sus longitudes)** y así se convierte a decimal, considerando la expansión numérica.

Posteriormente para obtener el número final se hace lo siguiente:

$$Conversión(subcromosoma) = A + DN(subcromosoma) \cdot \frac{B-A}{2^M-1}$$

Donde:

A es el límite inferior que toma la variable de decisión.

B es el límite superior que toma la variable de decisión.

M es la longitud del subcromosoma asociado a la variable de decisión.

DN (Decimal number) es el número en decimal del subcromosoma asociado a la variable de decisión.

FloatPointRepresentation (script)

Este script contiene todas las funcionalidades requeridas para que se pueda hacer uso de una codificación de tipo Float Point **(ó Punto Flotante)**; ésto significa que los alelos que conforman al cromosoma serán números de punto flotante.

Un número de punto flotante es aquél que tiene una parte entera y una decimal; cabe mencionar que si el número en cuestión no tiene expansión decimal, se le considera un número de representación Integer **(ó Entera)**; ésto porque en algunas fuentes se manejan la representación de Punto Flotante y Entera por separado.

calculate_length_subchromosomes(*vector_variables*, *number_of_decimals*, *representation_parameters*)

Realiza el cálculo de subcromosomas de acuerdo a la representación Float Point **(ó Punto Flotante)**.

Esta función es de aquéllas que se tienen que implementar obligatoriamente.

create_chromosome(*length_subchromosomes*, *vector_variables*, *number_of_decimals*, *representation_parameters*)

Crea un cromosoma con contenido de punto flotante.

Esta función es de aquéllas que se tienen que implementar obligatoriamente.

evaluate_subchromosomes(*complete_chromosome*, *length_subchromosomes*, *vector_variables*, *number_of_decimals*, *representation_parameters*)

Toma cada porción de cromosoma y la evalúa para luego ser asignada a la variable de decisión correspondiente. Este método es de los que se debe de implementar obligatoriamente.

2.2.2 Community (clase)

class **Community**(*vector_functions*, *vector_variables*, *available_expressions*, *number_of_decimals*, *representation_instance*, *representation_parameters*, *fitness_instance*, *fitness_parameters*, *sharing_function_instance*, *sharing_function_parameters*, *selection_instance*, *selection_parameters*, *crossover_instance*, *crossover_parameters*, *mutation_instance*, *mutation_parameters*)

Proporciona toda la infraestructura lógica para poder construir poblaciones y operar con éstas, además de transacciones relacionadas con sus elementos de manera individual.

Se le llama Community porque aludiendo a su significado una Community **(ó Comunidad)** consta de al menos una Population **(o Población)**. De esta manera se deduce que en algún momento habrán métodos que involucren a más de una población.

Parameters

- **vector_functions** (*List*) – Lista que contiene las funciones objetivo previamente saneadas por Controller/Controller.py.
- **vector_variables** (*List*) – Lista que contiene las variables de decisión previamente saneadas por Controller/Controller.py.
- **available_expressions** (*Dictionary*) – Diccionario que contiene algunas funciones escritas como azúcar sintáctica para que puedan ser utilizadas más fácilmente por el usuario y evaluadas más rápidamente en el programa **(véase Controller/XML/PythonExpressions.xml)**.
- **number_of_decimals** (*Integer*) – El número de decimales que tendrán las soluciones; con este número se determina en gran medida el tamaño del cromosoma.
- **representation_instance** (*Instance*) – Instancia de la técnica de representación que eligió el usuario **(véase Controller/Verifier.py)**.
- **representation_parameters** (*Dictionary*) – Diccionario que contiene todos los parámetros adicionales a la técnica de representación considerada por el usuario.
- **fitness_instance** (*Instance*) – Instancia de la técnica de Fitness que eligió el usuario **(véase Controller/Verifier.py)**.
- **fitness_parameters** (*Dictionary*) – Diccionario que contiene todos los parámetros adicionales a la técnica de Fitness seleccionada por el usuario.

- **sharing_function_instance** (*Instance*) – Instancia de la técnica de Sharing Function seleccionada por el usuario **(véase Controller/Verifier.py)**.
- **sharing_function_parameters** (*Dictionary*) – Diccionario que contiene todos los parámetros adicionales a la técnica de Fitness seleccionada por el usuario.
- **selection_instance** (*Instance*) – Instancia de la técnica de selección **(Selection)** elegida por el usuario **(véase Controller/Verifier.py)**.
- **selection_parameters** (*Dictionary*) – Diccionario que contiene todos los parámetros adicionales a la técnica de selección **(Selection)** usada por el usuario.
- **crossover_instance** (*Instance*) – Instancia de la técnica de cruza **(Crossover)** tomada por el usuario **(véase Controller/Verifier.py)**.
- **crossover_parameters** (*Dictionary*) – Diccionario que contiene todos los parámetros adicionales a la técnica de cruza **(Crossover)** manejada por el usuario.
- **mutation_instance** (*Instance*) – Instancia de la técnica de mutación **(Mutation)** tomada por el usuario **(véase Controller/Verifier.py)**.
- **mutation_parameters** (*Dictionary*) – Diccionario que contiene todos los parámetros adicionales a la técnica de mutación **(Mutation)** seleccionada por el usuario.

Returns Model.Community.Community.Community

Return type Instance

_Community__compare_dominance(*current*, *challenger*, *allowed_functions*)

Note: Este método es privado.

Permite realizar la comparación de las funciones objetivo de los individuos current y challenger tomadas una a una para indicar así quién es el dominado y quién es el que domina. Cabe mencionar que más apropiadamente se le conoce como dominancia fuerte de Pareto.

Parameters

- **current** (*Instance*) – El individuo inicial para comprobar dominancia.
- **challenger** (*Instance*) – El individuo que reta al inicial para comprobar dominancia.
- **allowed_functions** (*List*) – Lista que indica cuáles son las funciones objetivo que deben compararse.

Returns True si current domina a challenger, False en otro caso.

Return type Boolean

_Community__get_best_individual_results(*population*)

Note: Este método es privado.

Obtiene los valores de las variables de decisión y de las funciones objetivo por cada individuo.

Parameters **population** (*List*) – Una lista que contiene los mejores individuos por generación.

Returns Una lista que contiene por un lado la tupla (generacion, funciones) y por otro la tupla (generación, variables). Esto por cada generación.

Return type List

_Community__get_pareto_results(*population*)

Note: Este método es privado.

Obtiene el frente de Pareto, el complemento del frente de Pareto y el óptimo de Pareto.
Para una mejor orientación léase la parte escrita del proyecto.

Parameters **population** (*Instance*) – La población sobre la cual se obtendrán estos elementos.

Returns Una lista que contiene el frente de Pareto, su complemento y el óptimo de Pareto.

Return type List

_Community__using_sharing_function(*individual_i*, *individual_j*)

Note: Este método es privado.

Devuelve un valor que ayuda al cálculo del sharing function.
A grandes rasgos el sharing function sirve para hacer una selección más precisa de los mejores individuos cuando se da el caso de que tienen el mismo número de individuos dominados.

Parameters

- **individual_i** (*Instance*) – Individuo sobre el que se hará la operación.
- **individual_j** (*Instance*) – Individuo sobre el que se hará la operación.

Returns El resultado que contribuirá al sharing function.

Return type Float

assign_fonseca_and_flemming_pareto_rank(*population*, *allowed_functions='All'*)

Asigna una puntuación **(ó rank)** a cada uno de los individuos de una población con base en su dominancia de Pareto.
A grandes rasgos, el algoritmo asigna un rank que consiste en:

$$rank(Individuo) = número\_soluciones\_que\_dominan(Individuo) + 1$$

Esta técnica es usada principalmente por M.O.G.A.

Parameters

- **population** (*Instance*) – La población sobre la que se hará la operación.
- **allowed_functions** (*List*) – Lista que contiene las posiciones de las funciones que son admisibles para hacer comparaciones. Por defecto tiene el valor "All".

assign_goldberg_pareto_rank(*population*, *allowed_functions='All'*)

Asigna una puntuación **(ó rank)** a cada uno de los Individuos de una Población con base en su dominancia de Pareto.
En términos generales, el algoritmo trabaja con niveles, es decir, primero toma los Individuos no dominados y les asigna un valor 0, luego los elimina del conjunto y nuevamente aplica la operación sobre los no dominados del nuevo conjunto, a los que les asigna el valor 1, y así sucesivamente hasta no quedar Individuos.
Esta técnica es usada principalmente por N.S.G.A. II.

Parameters

- **population** (*Instance*) – La Población sobre la que se hará la operación.
- **allowed_functions** (*List*) – Lista que contiene las posiciones de las funciones que son admisibles para hacer comparaciones. Por defecto tiene el valor "All".

assign_population_fitness(*population*)
Aplica la asignación de Fitness para una población dada usando como base el Ranking de cada individuo **(véase Model/Fitness)**.

Parameters **population** (*Instance*) – La población sobre la que se hará la operación.

assign_zitzler_and_thiele_pareto_rank(*population*, *allowed_functions='All'*)

Asigna una puntuación (rank) a cada uno de los individuos de una población con base en su dominancia de Pareto.
A manera de esbozo, el algoritmo asigna un rank que consiste en una razón:

$$rank(Individuo) = \frac{número\_soluciones\_dominadas(Individuo)}{tamaño\_población} + 1$$

Esta técnica es usada principalmente por S.P.E.A. II

Parameters

- **population** (*Instance*) – La población sobre la que se hará la operación.
- **allowed_functions** (*List*) – Lista que contiene las posiciones de las funciones que son admisibles para hacer comparaciones. Por defecto tiene el valor "All".

calculate_population_niche_count(*population*)
Calcula el valor conocido como niche count que no es mas que la suma de los sharing function de todos los individuos j con el individuo i, con i != j.

Parameters **population** (*Instance*) – Conjunto sobre el que se hará la operación.

calculate_population_pareto_dominance(*population*, *allowed_functions*)
Realiza la comparación de dominancia entre todos los elementos de la población con base en la evaluación de sus funciones objetivo.

Parameters

- **population** (*Instance*) – La población sobre la que se hará la operación.
- **allowed_functions** (*List*) – Lista que indica las funciones objetivo permitidas para hacer la comparación.

calculate_population_sharing_function(*population*)
Calcula el sharing function de cada uno de los individuos de la población.

Parameters **population** (*Instance*) – Conjunto sobre el que se hará la operación.

create_population(*set_chromosomes*)
Crea una población usando un conjunto de cromosomas como base.

Parameters **set_chromosomes** (*List*) – Conjunto de cromosomas.

Returns Model.Community.Population

Return type Instance

evaluate_population_functions(*population*)
Evalúa cada uno de los subcromosomas de los individuos de la población **(Population)**.

Parameters **population** (*Instance*) – La población sobre la que se hará la operación.

execute_crossover_and_mutation(*selected_parents_chromosomes*)
Realiza la cruza y mutación de los individuos. Para el caso de la cruza ésta se lleva a cabo siempre entre dos individuos, mientras que la mutación es unaria.

Parameters **selected_parents_chromosomes** (*List*) – El conjunto de cromosomas sobre los cuales se aplicarán dichos operadores genéticos.

Returns Una instancia del tipo Model.Community.Population.

Return type Instance

execute_selection(*parents*)
Realiza la ejecución de la técnica de selección por medio de una instancia que se creó previamente **(véase Controller/Verifier.py)**.

Parameters **parents** (*Instance*) – El conjunto de individuos sobre el cual se aplicará la técnica

Returns Una lista con los cromosomas de aquellos individuos seleccionados.

Return type List

get_best_individual(*population*)
Obtiene el mejor individuo dentro de una población. Para estos fines el mejor individuo es aquél que tenga mejor dominancia.

Parameters **population** (*Instance*) – La población sobre la cual se hará la búsqueda.

Returns El individuo que cumple con la característica de la mayor dominancia.

Return type Instance

get_results(*best_individual_along_generations*, *external_set_population*)
Recolecta la información y la almacena en una estructura que contiene dos categorías principales: funciones objetivo y variables de decisión. Por cada una existen las subcategorías Pareto y mejor individuo, en referencia al óptimo o frente de Pareto **(según corresponda)** y a los valores del mejor individuo por generación **(véase View/Additional/ResultsGrapher/GraphFrame.py)**.

Parameters

- **best_individual_along_generations** (*List*) – Una lista que contiene los mejores individuos por generación.

- **external_set_population** (*Instance*) – La población sobre la cual se efectuarán las operaciones.

Returns Un diccionario con los elementos mostrados en la descripción.

Return type Dictionary

init_population(*population_size*)
Crea una población de manera aleatoria.

Parameters **population_size** (*Integer*) – El tamaño de la población.

Returns Model.Community.Community.Population

Return type Instance

Population (clase)

class Population(*population_size*, *vector_functions*, *vector_variables*, *available_expressions*, *number_of_decimals*)
Consiste en un conjunto de instancias de la clase Individual, proporcionando además métodos y atributos que se manifiestan tanto en grupo como de manera individual.

Parameters

- **population_size** (*Integer*) – El tamaño de la población.
- **vector_functions** (*List*) – Lista con las funciones objetivo.
- **vector_variables** (*List*) – Lista con las variables de decisión y sus rangos.
- **available_expressions** (*Dictionary*) – Diccionario que contiene algunas funciones escritas como azúcar sintáctica para que puedan ser utilizadas más fácilmente por el usuario y evaluadas más ŕapidamente en el programa **(véase Controller/XML/PythonExpressions.xml)**.
- **number_of_decimals** (*Integer*) – Número de decimales que tendrá cada solución **(tanto en variables de decisión como funciones objetivo)**.

Returns Model.Community.Population

Return type Instance

add_individual(*position*, *complete_chromosome*)
Añade un individuo a la población.

Parameters

- **position** (*Integer*) – La posición dentro del arreglo de individuos donde se colocará el nuevo elemento.
- **complete_chromosome** (*Array*) – El cromosoma del individuo.

calculate_population_properties()
Calcula atributos individuales con base en los valores de toda la población.

get_individuals()
Regresa los individuos de la población.

Returns Estructura que contiene a los individuos de la población.

Return type Array

get_length_vector_functions()
Regresa el número de elementos del vector de funciones objetivo.

Returns Número de funciones objetivo.

Return type Integer

get_size()
Otorga el tamaño de la población.

Returns El tamaño de la población.

Return type Integer

get_total_expected_value()
Regresa el valor esperado de la población.

Returns El valor esperado.

Return type Float

get_total_fitness()
Captura el Fitness total de la población.

Returns El valor del Fitness poblacional.

Return type Float

print_info()
Imprime en texto las características de los individuos de la población, tanto grupales como individuales **(en consola)**.

set_total_fitness(*value*)
Actualiza el Fitness total de la población.

Parameters **value** (*Float*) – El valor a actualizar.

shuffle_individuals()
Desordena los elementos de la población.

sort_individuals(*method*, *is_descendent*)
Ordena los individuos de acuerdo a algún criterio dado.

Parameters

- **method** (*String*) – El método o atributo sobre el cual se hará la comparación.
- **is_descendent** (*Boolean*) – Indica si el ordenamiento es ascendente o descendente.

Individual (clase)

class Individual(*complete_chromosome*, *vector_functions*, *available_expressions*, *number_of_decimals*)

La base de toda operación lógica.
Consiste en una abstracción de un elemento simple en función de un ecosistema. Si bien la parte esencial es el cromosoma, en esta implementación se añaden algunos elementos extra con la finalidad de facilitar ciertas operaciones.

Parameters

- **complete_chromosome** (*Array*) – El cromosoma que conformará al individuo.
- **vector_functions** (*List*) – Lista que contiene a las funciones objetivo.

- **available_expressions** (*Dictionary*) – Diccionario que contiene algunas funciones escritas como azúcar sintáctica para que puedan ser utilizadas más fácilmente por el usuario y evaluadas más rapidamente en el programa **(véase Controller/XML/PythonExpressions.xml)**.
- **number_of_decimals** (*Integer*) – El número de decimales que deberá tener cada solución, influye en el comportamiento del cromosoma.

Returns Model.Community.Population.Individual

Return type Instance

_Individual__evaluate_single_function(*function*, *expressions*)

Note: Este método es privado.

Evalúa una función objetivo.

Parameters

- **function** (*String*) – La función que será evaluada.
- **expressions** (*Dictionary*) – El diccionario que ayuda a evaluar la función. Expressions = variables + constantes + funciones built-in.

Returns La función evaluada.

Return type Float

evaluate_functions(*decision_variables*)
Evalúa todas las funciones objetivo.

Parameters **decision_variables** (*List*) – El vector de variables de decisión.

get_complete_chromosome()
Regresa el cromosoma del individuo.

Returns El cromosoma.

Return type Array

get_decision_variables()
Da el vector de variables de decisión.

Returns El vector de variables de decisión.

Return type List

get_evaluated_functions()
Regresa las funciones objetivo evaluadas.

Returns Las funciones objetivo evaluadas.

Return type List

get_evaluated_functions_dict()
Regresa las funciones objetivo evaluadas.

Returns Las funciones objetivo evaluadas en forma de diccionario.

Return type Dictionary

get_expected_value()

Se obtiene el valor esperado del individuo.

Por definición, el valor esperado es el número de posibles hijos que puede tener un individuo. Mientras más apto, más hijos.

Returns El valor esperado.

Return type Float

get_fitness()

Regresa el Fitness del individuo.

Returns El Fitness.

Return type Float

get_niche_count()

Regresa el valor niche para el individuo.

Returns El tamaño niche.

Return type Float

get_pareto_dominated()

Regresa el número de soluciones que dominan al individuo actual.

Returns El número de soluciones que dominan a la actual.

Return type Integer

get_pareto_dominates()

Regresa el número de soluciones que son dominadas por el actual individuo.

Returns El número de soluciones dominadas.

Return type Integer

get_rank()

Regresa la puntuación **(rank)** que se le designó al individuo **(véase Model/Community/Community.py)**.

Returns El rango.

Return type Float

get_vector_functions()

Obtiene el vector de funciones objetivo.

Returns El vector de funciones objetivo.

Return type List

print_info()

Imprime las características básicas del Individuo **(en consola)**.

set_expected_value(*value*)

Actualiza el valor esperado del individuo.

Parameters **value** (*Float*) – El valor a actualizar.

set_fitness(*value*)

Actualiza el valor del Fitness.

Parameters **value** (*Float*) – El valor a actualizar.

set_niche_count(*value*)
Actualiza el valor niche.

Parameters **value** (*Float*) – El valor a actualizar.

set_pareto_dominated(*value*)
Actualiza el número de soluciones que dominan a la solución actual.

Parameters **value** (*Integer*) – El valor a actualizar.

set_pareto_dominates(*value*)
Actualiza el número de soluciones dominadas por el individuo actual.

Parameters **value** (*Integer*) – El valor a actualizar.

set_rank(*rank*)
Actualiza el rango del individuo.

Parameters **rank** (*Float*) – El valor a actualizar.

2.2.3 Fitness (módulo)

Este módulo provee técnicas que calculan el Fitness **(ó Aptitud)** de los Individuals **(ó Individuos)** de una Population **(ó Población)**.

Se entiende por Fitness a un número que indica la calidad del Individuo **(en particular de sus variables de decisión)** frente a las funciones objetivo al momento de ser evaluadas, esto es, a mayor Fitness, mayor es la probalidad de que las variables de decisión del Individuo sean la solución óptima para las funciones objetivo.

La asignación del Fitness depende en gran medida del ranking que se les haya otorgado a los Individuos previamente **(véase Model/Community/Community.py)**.

Indirectamente, esto nos indica que un Individuo con un Fitness alto tiene más probabilidades de ser elegido en los métodos de Selection **(ó Selección)** y propagar su carga genética; así en las funciones de dicha sección **(Model/Operator/Selection)** el criterio para escoger a un Individuo está basado comúnmente en su Fitness.

Al final la meta es que el usuario cree sus propias versiones de asignación de Fitness, para lo cual es imperativo que, además de agregar la descripción de la codificación a Controller/XML/Features.xml **(véase el archivo mencionado en la sección de código)**, se implemente la siguiente función:

assign_fitness(population, fitness_parameters):

Realiza la asignación de Fitness de los individuos.
Dentro de esta se suelen usar métodos de la clase Population **(véase Model/Community/Population/Population.py)** y de la clase Individual **(véase Model/Community/Population/Individual/Individual.py)**, por lo que es muy recomendable que el usuario verifique las funciones disponibles. Algunas de las que se ocupan más frecuentemente son:

get_rank (Individual)
set_fitness (Individual).
set_total_fitness (Population)
calculate_population_properties (Population)

Parameters

- **population** (*Instance*) – La Población sobre la cual se hará el cálculo de Fitness por cada Individuo.
- **fitness_parameters** (*Dictionary*) – Un diccionario que puede contener opciones adicionales para el cálculo de Fitness.

LinearRankingFitness (script)

Se implementa la asignación de Fitness conocida como Linear Ranking **(ó Ranking Lineal)**.
Es denominada así porque el Fitness se asigna con una función lineal que tiene como fundamento la posición que ocupa el Individuo dentro de la Población.
El procedimiento es: tomando en cuenta el ranking asignado a los Individuals **(ó Individuos)** por la clase Community **(véase Model/Community/Community.py)** se ordenan de acuerdo a este valor y entonces el Fitness se basa en la posición que cada uno de los Individuos ocupa. Más en específico, el Fitness está proporcionado por la siguiente fórmula:

$$Fitness(Individuo) = 2 - SP + \frac{2 \cdot (SP-1) \cdot posición(Individuo)}{tamaño\_población - 1}$$

Donde:

SP (Selective Pressure ó Presión Selectiva) es un valor que oscila entre 1 y 2.

Posición(Individuo) es la que ocupa el Individuo de acuerdo al rank.

Haciendo un análisis somero en la fórmula, se puede apreciar que los Individuos con mejor Fitness serán aquéllos que se encuentren en las últimas posiciones, sin embargo los rankings que se manejan en este proyecto son inversamente proporcionales a la calidad de los Individuos **(véase Model/Community/Community.py)**; por ello es importante ordenar a los Individuos de manera descendente para que la operación tenga sentido.
La función encargada de esto se llama sort_individuals y está en **Model/Community/Population/Population.py**.

assign_fitness(*population*, *fitness_parameters*)

Se realiza la implementación del Fitness de tipo Linear Ranking **(ó Ranking Lineal)** tomando en cuenta los datos proporcionados en la parte superior.

LinearScalingFitness (script)

Es creada la asignación de Fitness conocida como Linear Scaling **(ó Escalamiento Lineal)**.
Se le llama así porque el Fitness de los Individuos será regido por una función lineal.
Más en concreto, la función **(también denominada fórmula)** es:

$$Fitness(Individuo) = \alpha \cdot F_0(Individuo) + \beta$$

Donde:

α toma los valores -1 y 1.

β es 0 ó un número positivo.

F_0 es conocido como el valor de la función objetivo del Individuo. Nótese que F_0 debe ser directamente proporcional al Fitness del Individuo.

Con respecto de F_0 es importante considerar lo siguiente: dado que se está manejando un sistema multiobjetivo puede haber más de un valor en existencia, por ello se necesita una cantidad que conjunte estas evaluaciones el cual es el rank, sin embargo el rank es inversamente proporcional a la calidad de un Individuo.
De esta manera se debe hacer una modificación para garantizar que exista un valor proporcional al Fitness del Individuo, por lo cual F_0 se reescribe así:

$$F_0(Individuo) = tamaño\_población - rank(Individuo)$$

Reescribiendo la fórmula inicial se tiene lo siguiente:

$$Fitness(Individuo) = \alpha \cdot [tamaño\_población - rank(Individuo)] + \beta$$

Con esta actualización ya es posible obtener un Fitness acorde al rank del Individuo sin alterar la esencia de la técnica.

assign_fitness (*population*, *fitness_parameters*)
Se construye la asignación del Fitness de tipo Linear Scaling **(ó Escalamiento Lineal)** tomando como referencia toda la información provista anteriormente.

NonLinearRankingFitness (script)

Se implementa la asignación de Fitness conocida como Non-Linear Ranking **(ó Ranking No Lineal)** que, a diferencia de los demás métodos, la aplica usando como base la posición del Individual **(ó Individuo)** en la Population **(ó Población)** como resultado de las operaciones de ranking **(véase Model/Community/Community.py)**.
Posteriormente el Fitness se constituye tomando la posición del Individuo y una función polinomial **(la cual es una función no lineal, de ahí el nombre)**.
La fórmula es la siguiente:

$$Fitness(Individuo) = \frac{TP \cdot X^{posición(Individuo)}}{\sum_{i=1}^{TP} X^{i-1}}$$

Donde:

TP es el tamaño de la Población.

Posición(Individuo) es la que ocupa éste de acuerdo al ranking previo.

X es la solución al polinomio: $(SP - TP) \cdot X^{TP-1} + SP \cdot X^{TP-2} + ... + SP \cdot X + SP = 0$

SP (Selective Pressure ó Presión Selectiva) varía entre 1 y 2.

Haciendo un análisis somero en la fórmula, se puede apreciar que los Individuos con mejor Fitness serán aquéllos que se encuentren en las últimas posiciones, sin embargo los rankings que se manejan en este proyecto son inversamente proporcionales a la calidad de los Individuos **(véase Model/Community/Community.py)**; por ello es importante ordenar a los Individuos de manera descendente para que la operación tenga sentido.
La función encargada de esto se llama sort_individuals y está en **Model/Community/Population/Population.py**.

assign_fitness (*population*, *fitness_parameters*)
Utilizando la explicación concretada al principio, se realiza la implementación de la asignación de Non-Linear Ranking Fitness **(ó Fitness de Ranking No Lineal)**.

calculate_root (*polynome*, *x_0*, *epsilon*)

Calcula la solución de un polinomio usando el método Newton-Raphson. A grandes rasgos el funcionamiento es el siguiente:

Tomando como base el punto x_0 se obtiene x_1 así:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

Una vez obtenido x_1 se calcula x_2 de la misma manera:

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}$$

El proceso se repite para 'n' iteraciones hasta que el valor alcance la precisión de epsilon ó el polinomio ya no tenga más derivadas. Concretando lo anterior:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Cuando x_{n+1} se acerque a epsilon ó cuando el polinomio no sea más derivable el método se detendrá.

Parameters

- **polynome** (*List*) – El polinomio en el que se buscará la solución.
- **x_0** (*Float*) – el punto sobre el que se hará la evaluación del polinomio.
- **epsilon** (*Float*) – La precisión decimal que se necesita para poder devolver el resultado.

Returns La solución al polinomio.

Return type Float

derivate(*polynome*)
Método que calcula la derivada de un polinomio, modificando directamente éste sin regresar nada.

Parameters **polynome** (*List*) – El polinomio inicial.

evaluate_polynome(*polynome*, *x*)
Evalúa un polinomio en un cierto valor.

Parameters

- **polynome** (*List*) – El polinomio a evaluar.
- **x** (*Float*) – El valor sobre el que se evaluará el polinomio.

Returns La evaluación del polinomio.

Return type Float

ProportionalFitness (script)

Se desarrolla la asignación de Fitness conocida como Proportional **(ó Proporcional)**.
La función **(ó fórmula)** utilizada es la siguiente:

$$Fitness(Individuo) = \frac{F_0(Individuo)}{\sum_{i=1}^{tamaño\_población} F_0(Individuo_i)}$$

Donde:

F_0 es conocido como el valor de la función objetivo del Individuo. Nótese

que F_0 debe ser proporcional al Fitness del Individuo.

De acuerdo a la información provista anteriormente, la asignación es llamada así porque, como dice el nombre, el Fitness de un Individuo corresponde a la parte proporcional de la cantidad total de F_0 de la Population **(ó Población)**.
De esta manera es posible ajustar los valores para que no existan Fitness dispares.
Con respecto de F_0 es importante considerar que, dado que se está manejando un sistema multi objetivo puede haber más de un valor en existencia, por ello se necesita una cantidad que conjunte estas evaluaciones el cual es el rank, sin embargo el rank es inversamente proporcional a la calidad de un Individuo.
Entonces se debe hacer una modificación para garantizar que exista un valor proporcional al Fitness del Individuo, por lo cual F_0 se reescribe así:

$$F_0(Individuo) = tamaño\_población - rank(Individuo)$$

Reescribiendo la fórmula inicial se tiene lo siguiente:

$$Fitness(Individuo) = \frac{tamaño\_población - rank(Individuo)}{\sum_{i=1}^{tamaño\_población}[tamaño\_población - rank(Individuo_i)]}$$

Con esta actualización ya es posible obtener un Fitness acorde al rank del Individuo sin alterar la esencia de la técnica.

assign_fitness (*population*, *fitness_parameters*)
Se implementa la asignación de Proportional Fitness **(ó Fitness Proporcional)** con base en la información especificada con anterioridad.

2.2.4 Operator (módulo)

En éste se encuentran implementadas todas aquellas funcionalidades que intervengan en el proceso de la creación de una nueva Population **(ó Población)** hija.
La finalidad de ésto es propagar y realizar combinaciones de la carga genética de los Individuals **(ó Individuos)** más aptos mediante el cromosoma **(véase Model/ChromosomalRepresentation)** para obtener soluciones con una mejor calidad que sus predecesoras.
Para este punto es importante mencionar que la calidad de un Individuo es directamente proporcional a su Fitness **(véase Model/Fitness)**
En términos generales, la manera de construir una Población hija es la siguiente:

- De la Población actual y tomando como base el Fitness de cada Individuo se seleccionan aquéllos que serán los elegidos para reproducirse. Nótese que un Individuo puede ser tomado en cuenta más de una vez si se da el caso.
- Con base en los elegidos, se toman sus respectivos cromosomas y se realiza la operación de Crossover **(ó Cruza)**. Ésta es una simulación de una reproducción de tipo sexual donde se toman dos padres para "procrear" dos hijos. Las características de los hijos dependerán de las técnicas usadas **(véase Model/Operator/Crossover)**.
- Se toman los hijos y uno a uno se les aplica la operación de mutación.

Al final Población hija constará de los hijos "mutados".

A continuación se muestran las siguientes subcategorías correspondientes a los pasos descritos anteriormente, cada una con sus respectivas técnicas desarrolladas:

Selection (módulo)

En esta sección se encuentran implementadas todas las técnicas relacionadas con la selección de Individuos.
Como se ha mencionado antes, durante dicha operación la importancia de la elección radica en el Fitness de cada Individuo, además un Individuo puede ser seleccionado más de una vez si la causa lo amerita.
Así, se elegirán tantos Individuos como elementos haya en la Población.
El objetivo radica en mantener el equilibrio entre una "selección justa" y la oportunidad de permitir a los Individuos con una calidad media o baja la propagación de su carga genética.

Al final se busca que el usuario desarrolle sus propias técnicas de selección, por lo cual, además de añadir el método en el listado localizado en **Controller/XML/Features.xml**, deberá implementar la siguiente función:

`execute_selection_technique(population,selection_parameters):`

Lleva a cabo la selección de Individuos de una Población. Es importante recalcar que, la función que más se ocupa es:

get_fitness (Model/Community/Population/Individual.py)

Aunque existen otras que pueden tener relevancia para el usuario **(véase Model/Community/Population.py)**.
Como medida adicional, para los eventos de Crossover **(ó Cruza)** y Mutation **(ó Mutación)** se recomienda ampliamente que este método regrese únicamente los cromosomas asociados a los Individuos, ya que ésto facilita sobremanera las operaciones mencionadas.

Parameters

- **`population`** (*`Instance`*) – La Población sobre la cual se se seleccionarán los Individuos.
- **`selection_parameters`** (*`Dictionary`*) – Un diccionario que puede contener opciones adicionales para la selección de Individuos.

Returns Una lista que contiene los cromosomas de los Individuos seleccionados.

Return type List

Roulette (script)

Se implementa el método de selección conocido como Roulette **(ó Ruleta)**. También es llamado Proportional Selection **(ó Selección Proporcional)**.
En la función se distinguen dos etapas principales: construir la ruleta y "ponerla a girar" para que se elija el elemento.

- Para la primera etapa se toma como base el Valor Esperado **(ó Expected Value)** de cada Individuo **(véase Model/Community/Population/Individual.py)**.

  El Valor Esperado para fines de este proyecto es el número de "hijos" que un Individuo puede ofrecer.
  Éste se calcula de la siguiente forma:

$$Valor\_Esperado(Individuo) = \frac{tamaño\_población \cdot Fitness(Individuo)}{\sum_{i=1}^{tamaño\_población} Fitness(Individuo_i)}$$

Al final aquéllos con Valores Esperados altos tendrán lugar a mayores espacios en la ruleta y por ende su probabilidad de elección aumenta.

- Para recorrer la ruleta en realidad se toma un valor aleatorio entre 0 y la suma de los Valores Esperados. Entonces se van sumando los Valores Esperados de los Individuos hasta que se exceda el valor aleatorio mencionado antes. Aquel elemento cuyo Valor Esperado haya excedido la suma se considera el elegido y es seleccionado para la etapa de cruza.

Para la selección de Individuos se efectúa la segunda operación tantas veces como el tamaño de la Población.
Cabe mencionar que el Valor Esperado ya se calcula de manera automática en este proyecto **(véase Model/Community/Population/Population.py)**.

execute_selection_technique (*population*, *selection_parameters*)
De acuerdo al proceso descrito anteriormente, se implementa la técnica conocida como Roulette **(ó Ruleta)**.

ProbabilisticTournament (script)

Se desarrolla la técnica conocida como Torneo Probabilístico **(ó Probabilistic Tournament)**.
Tal como lo sugiere el nombre, la selección será llevada a cabo en forma de competencia directa entre los Individuos.
Tradicionalmente se comparan sus Fitness y de esta manera el Individuo ganador es aquél con la cantidad mayor de Fitness, pero dado que se maneja un esquema probabilístico la decisión no depende totalmente del factor antes mencionado.

De esta manera se pueden recapitular los siguientes pasos:

- Tomar k $(2 \leqslant k \leqslant tamaño\_población)$ Individuos de la Población.
- Realizar el torneo de manera secuencial entre los elementos seleccionados anteriormente, esto es, tomar el elemento A y enfrentarlo con B, al resultado de la batalla anterior enfrentarlo con C y así sucesivamente.

  Para ello por cada encuentro se crea un número aleatorio entre 0 y 1, si el número es menor a 0.5 se toma al elemento con menor Fitness, de lo contrario se elige al de mayor Fitness. La operación se lleva a cabo hasta que se tenga un ganador de los k Individuos.

Los dos pasos anteriores se repiten hasta que se hayan obtenido tantos Individuos como el tamaño de la Población.

execute_selection_technique (*population*, *selection_parameters*)
Tomando en cuenta las bases descritas previamente, se implementa el método conocido como Probabilistic Tournament **(ó Torneo Probabilístico)**.

StochasticUniversalSampling (script)

Se determina la técnica conocida como Stochastic Universal Sampling **(ó Muestreo Estocástico Universal)**.
Primero que nada es menester mencionar que es necesario el uso del Expected Value **(ó Valor Esperado)** de cada Individuo.
Para fines concernientes a este proyecto, se trata del número de "hijos" que un Individuo puede ofrecer. Éste se calcula de la siguiente forma:

$$Valor\_Esperado(Individuo) = \frac{tamaño\_población \cdot Fitness(Individuo)}{\sum_{i=1}^{tamaño\_población} Fitness(Individuo_i)}$$

Con base a lo anterior, el método consiste en lo siguiente:

- Se selecciona un valor aleatorio entre 0 y 1, a éste se le llamará Pointer **(ó Puntero)**
- De manera secuencial se seleccionarán tantos Individuos como el tamaño de la población, los cuales deben estar igualmente espaciados en su Valor Esperado tomando como referencia el valor de Pointer.

Es importante aclarar el segundo punto, así que se abordará desde una perspectiva computacional:

- Se deben tener variables adicionales que indiquen la acumulación tanto del Pointer **(CP, Cumulative Pointers)** como de los Valores Esperados **(CEV, Cumulative Expected Value)** así como al Individuo actual que está siendo seleccionado **(I)**.
- Para averiguar si un Individuo está igualmente espaciado en su Valor Esperado con respecto de los demás basándose en Pointer, basta con corroborar que:

$$CP + Pointer > CEV + EV$$

- Si la condición descrita es verdadera los valores EV e I deben actualizarse **(I se ajusta al siguiente Individuo)** ya que esto indica que se buscará al siguiente Individuo espaciado equitativamente con el valor Pointer. No se hace nada si la condición es falsa.
- Independientemente del valor de la condición anterior, CP y CEV deben actualizarse durante todo el ciclo.

Cabe mencionar que si la lista de Individuos se agota, se puede volver a iterar desde el inicio teniendo cautela en conservar CEV y CP.

`execute_selection_technique`(*population*, *selection_parameters*)
De acuerdo a la información provista anteriormente, se implementa el método conocido como Stochastic Universal Sampling **(ó Muestreo Estocástico Universal)**.

Crossover (módulo)

Aquí se desarrollan las técnicas de Crossover **(ó Cruza)**.
Prosiguiendo con el ciclo de creación de una nueva Población, es en este apartado donde se lleva a cabo la concepción de nuevos Individuos.

Debido a esto se busca crear "hijos" más aptos que respondan mejor ante la problemática fundamentada, es decir, concebir soluciones que se adapten mejor a los criterios establecidos por el usuario desde un inicio basándose en las soluciones predecesoras.

Es menester mencionar que esta función es meramente binaria, lo cual significa que siempre deben haber dos padres, además se debe hacer hincapié en que la Cruza se ejecuta a nivel cromosómico **(véase Model/ChromosomalRepresentation)**,por lo que se debe tener mesura con el tratamiento de los métodos, dicho de otra manera, cada Representación Cromosómica debe ir acompañada de al menos una función de Cruza.

Como dato para posteriores referencias, un gen hace referencia a una casilla del cromosoma, mientras que un alelo es el valor que puede existir en un gen.

Entonces se persigue que el usuario construya sus propias funciones de Cruza, para lo cual, además de añadir el método en el listado localizado en **Controller/XML/Features.xml**, deberá implementar la siguiente función:

execute_crossover_technique(chromosome_a, chromosome_b, crossover_parameters):

Lleva a cabo la cruza de dos Individuos a nivel cromosómico.

Además esta función debe retornar siempre dos hijos los cuales serán la cruza de A con B y la cruza de B con A, esto nos indica que, con el objetivo de incrementar la calidad de los Individuos sin perder la carga genética ganada o introducir elementos riesgosos, la cruza consiste en generar un nuevo Individuo y su recíproco; así se garantiza una adecuada y controlada descendencia.

Finalmente, esta función debe contar con la probabilidad de Cruza, la cual indica si se debe o no hacer la operación cromosómica; en caso de ser la respuesta negativa los hijos resultan en copias idénticas de los padres.

Parameters

- **chromosome_a** (*List*) – El cromosoma del Individuo A.
- **chromosome_b** (*List*) – El cromosoma del Individuo B.
- **crossover_parameters** (*Dictionary*) – Un diccionario que puede contener opciones adicionales para la cruza de Individuos.

Returns Un arreglo con dos cromososomas, el primero es la cruza de A con B, mientras que el segundo es la cruza de B con A.

Return type Array

NPointsCrossover (script)

Se implementa el método que lleva por nombre N-Points Crossover **(ó Cruza en N-Puntos)**. Para comenzar, esta técnica está elaborada para usarse tanto por Representación Cromosómica **(véase Model/ChromosomalRepresentation)** de tipo FloatPoint **(ó de Punto Flotante)** como Binary **(ó Binaria)**.
Su funcionamiento consiste en construir a los descendientes usando sub-bloques de cromosomas de cada uno de los padres, determinados éstos por una cierta cantidad de puntos de corte, de ahí el nombre.
Aterrizando lo anterior de una manera concisa se tiene lo siguiente:

- Consideremos a los cromosomas de los padres Padre I: $I_1 I_2 ... I_n$

  y Padre J: $J_1 J_2 .... J_n$

- Posteriormente se determinan aleatoriamente los puntos de corte, cabe mencionar que si los cromosomas son de tamaño n, pueden existir máximo n - 1 puntos. Supongamos que se crean k puntos $(1 \leqslant k \leqslant n-1)$ y por lo tanto cada cromosoma queda separado en k + 1 bloques.

  De esta manera obtenemos: Padre I en bloques **(BI)**: $BI_1 BI_2 ... BI_{k+1}$; Padre J en bloques **(BJ)**: $BJ_1 BJ_2 ... BJ_{k+1}$.

- Finalmente cada hijo constará de la alternancia de bloques de manera secuencial comenzando por el bloque inicial de un padre determinado, dicho de otra forma, los hijos estarán constituidos de la siguiente manera:

- Para el hijo H_1: $BI_1 BJ_2 ... BI_{k+1}$
- Para el hijo H_2: $BJ_1 BI_2 ... BJ_{k+1}$

Sólo queda mencionar que hasta el cierre de este proyecto no existe una manera transparente desde el View **(ó Vista)** de conocer, dada una representación Binaria y un conjunto de variables de decisión y funciones objetivo, el número máximo de puntos de corte permitidos para este procedimiento, sin embargo, una manera de mitigar esta situación fue contemplar algún posible caso de error en esta sección y mandar un mensaje de error a la Vista por si llegase a suceder algún desperfecto durante el proceso.

execute_crossover_technique(*chromosome_a*, *chromosome_b*, *crossover_parameters*)
Usando como base la información proporcionada anteriormente, se implementa el método conocido como N-Points Crossover **(ó Cruza en 'N' Puntos)**.

UniformCrossover (script)

Se lleva a cabo la implementación de la técnica conocida como Uniform Crossover **(ó Cruza Uniforme)**.
Primero que nada esta operación está fabricada para usarse tanto con la Representación Cromosómica **(véase Model/ChromosomalRepresentation)** de tipo FloatPoint **(ó Punto Flotante)** como Binary **(ó Binaria)**.

La característica de este procedimiento es crear nuevos Individuos intercambiando secuencialmente los genes de sus padres; visto de una manera más estructurada consiste en lo siguiente:

- Tenemos a los cromosomas de los padres Padre A: $A_1 A_2 ... A_n$

  y Padre B: $B_1 B_2 ... B_n$
- Ahora, cada hijo será construido con genes de uno y sólo uno de los padres a menos que se indique lo contrario; este movimiento será posible con una variable denominada Pmask **(Pm)** que toma valores de 0 a 1 y una probabilidad de Pmask **(Pp)** que también toma valores de 0 a 1. Entonces lo anterior se puede declarar así:
  - Para el hijo (H_1) que tomará sus genes del padre A **(PA)**:

    Si $Pm \leqslant Pp \, entonces \, H_1(i) = A_i, \, en \, otro \, caso \, H_1(i) = B_i; 1 \leqslant i \leqslant n$
  - Para el hijo (H_2) que tomará sus genes del padre B **(PB)**:

    Si $Pm \leqslant Pp \, entonces \, H_2(i) = B_i, \, en \, otro \, caso \, H_1(i) = A_i; 1 \leqslant i \leqslant n$

execute_crossover_technique(*chromosome_a*, *chromosome_b*, *crossover_parameters*)
Tomando en cuenta la información proporcionada con anterioridad, se implementa el método conocido como Uniform Crossover **(ó Cruza Uniforme)**.

Mutation (módulo)

En esta parte se encuentran detalladas las técnicas relacionadas con Mutation **(ó Mutación)**.
Retomando el proceso de creación de una nueva Población, es aquí donde una vez obtenidos los hijos, se modifican pequeñas porciones **(genes)** de sus cromosomas de manera individual.
Con ésto se persigue principalmente que estas ínfimas alteraciones permitan incrementar la exploración del material genético y por ende otorgar Individuos aún más aptos sin caer en el peligro de perder características valiosas en la Población.

Considerando lo anterior, lo primero que hay que tomar en cuenta es que la operación de Mutación es unaria, esto significa que sólo se puede mutar el cromosoma de un Individuo a la vez.
También y reiterando la información pasada, la Mutación es una operación que se lleva a cabo a nivel cromosómico **(véase Model/ChromosomalRepresentation)**, por lo que se debe tener mesura con el tratamiento de los métodos, dicho de otra manera, cada Representación Cromosómica debe ir acompañada de al menos una función de Mutación.

Como dato para posteriores referencias, un gen hace referencia a una casilla del cromosoma, mientras que un alelo es el valor que puede existir en un gen.

Así, se invita a que el usuario construya sus propias versiones de Mutación, por lo cual, además de añadir el método en el listado localizado en **Controller/XML/Features.xml**, deberá implementar la siguiente función:

execute_mutation_technique(chromosome,mutation_parameters):

Lleva a cabo mutación del Individuo a nivel cromosómico.
A grandes rasgos, modifica los alelos de los genes tomando en cuenta la gama de valores a los que se pueden transformar **(por ejemplo, una mutación de representación Binaria puede transformarse sólo en valores 0 ó 1)**.
El método debe retornar siempre el cromosoma mutado.
Finalmente, esta función debe contar con la probabilidad de Mutación, la cual indica si se debe o no hacer la operación cromosómica por cada gen; en caso de ser la respuesta negativa el Individuo no experimenta modificación alguna en el gen y se pasa al siguiente y así sucesivamente.

Parameters

- **chromosome** (*List*) – El cromosoma para ser mutado.
- **mutation_parameters** (*Dictionary*) – Un diccionario que puede contener opciones adicionales para la mutación del cromosoma del Individuo.

Returns El cromosoma modificado.

Return type List

BinaryMutation (script)

Se implementa el método conocido como Binary Mutation **(ó Mutación Binaria)**.
El procedimiento es el siguiente:

- Se trata cada gen individualmente y se modifica de acuerdo a una probabilidad de Mutación asignada, si ésta es suficiente se procede a hacer el cambio, en otro caso se deja el alelo asociado al gen intacto.
- Retomando el caso en que se puede modificar el alelo del gen se verifica su valor actual y ya que se maneja una representación Binaria su transformación es muy simple: si se encuentra un 0, el alelo toma el valor 1 y viceversa.

execute_mutation_technique(*chromosome*, *mutation_parameters*)
Usando la información mostrada anteriormente, se desarrolla la función conocida como Binary Mutation **(ó Mutación Binaria)**.

FloatPointMutation (script)

Se concreta el método conocido como Float Point Mutation **(ó Mutación de Punto Flotante)**.
El procedimiento es el siguiente:

- Se trata cada gen individualmente y se modifica de acuerdo a una probabilidad de Mutación asignada, si ésta es suficiente se procede a hacer el cambio, en otro caso se deja el alelo asociado al gen intacto.
- Retomando el caso en que se puede modificar el alelo del gen se verifica los límites de la variable de decisión que está ligada a éste, así como la precisión decimal. Entonces se crea el nuevo número con la precisión decimal requerida y se sustituye por el anterior.

`execute_mutation_technique`(*chromosome*, *mutation_parameters*)
Utilizando los datos de la parte superior, se desarrolla la función conocida como Float Point Mutation **(ó Mutación de Punto Flotante)**.

2.2.5 SharingFunction (módulo)

En esta sección se almacenan las técnicas relativas al Sharing Function **(ó Función de Compartición)**.
El objetivo de estas técnicas se delega a un rol secundario pero aún así muy importante y consiste en realizar un filtrado más minucioso de los mejores Individuos y así tomar a los candidatos elegidos para dejar descendencia.

La operación es útil en casos en el que la calidad de los Individuos es muy similar y entonces se desea seleccionar a los que son superiores, sin embargo, es menester mencionar que, en exceso, dicha selección parsimoniosa puede dar lugar a un efecto negativo del Selective Pressure **(ó Presión Selectiva, véase Model/MOEA)**.
Esto provoca que, lejos de dar una Población de elementos óptimos, los Indviduos se queden estancados puesto que al tener todos cargas genéticas muy similares, existe una pobre exploración genética en sus cromosomas y entonces no se llegará a una optimización de funciones objetivo adecuada.

Es por ello que no todos los MOEAS **(véase Model/MOEA)** lo utilizan, sin embargo se decidió implementar esta sección ya que extrapolando las circunstancias, en cualquier momento se puede hacer uso de técnicas de esta índole.

Haciendo énfasis en la parte matemática, el Sharing Function funciona así:

Cada Individual **(ó Individuo)** tendrá asociado un Shared Fitness **(ó Fitness Compartido)** que fungirá como el Fitness original asignado a cada Individo y el cual será obtenido de la siguiente manera:

$$SharedFitness(Individuo) = \frac{Fitness(Individuo)}{NicheCount(Individuo)}$$

Para fines de implementación el Shared Fitness será colocado en la misma variable utilizada para almacenar el Fitness original, esto por cada Individuo.
El Niche Count es un valor que indica qué tan cercano en calidad se encuentra un Individuo con respecto de los demás. La forma de calcularlo es la siguiente:

$$NicheCount(Individuo) = \sum_{j=1}^{tamaño\_población} SF(D(Individuo, Individuo_j))$$

Donde $D(Individuo_i, Individuo_j)$ es la distancia que existe entre el Individuo i y el Individuo j; mientras que el SF es el Sharing Function.
Entonces el SF se define como:

$$SF(D(Individuo_i, Individuo_j)) = \begin{cases} 1 - (\frac{D(Individuo_i, Individuo_j)}{\sigma_s})^{\alpha}, & si \;\; D < \sigma_s. \\ \\ 0, & en\ cualquier\ otro\ caso. \end{cases}$$

Donde α es una variable que casi siempre se asigna a 1 y σ_s marca el límite en el cual dos Individuos se consideran cercanos en calidad, es decir, viven en el mismo Niche Count.

Llegados a este punto, si bien la parte que se utilizará finalmente es el Shared Fitness, sólo las técnicas concernientes a $D(Individuo_i, Individuo_j)$ serán las que se implementen en esta sección, pues lo demás siempre se mantendrá estático.

Siendo más específicos con base en lo anterior, existen dos tipos de funciones de Distancia:

- De Similaridad Genotípica **(ó Genotypic Similarity)**.
- De Similaridad Fenotípica **(ó Phenotypic Similarity)**.

La primera indica en pocas palabras que la comparación se hará usando únicamente características relacionadas con el cromosoma, mientras que la segunda implicará la comparación usando las funciones objetivo evaluadas con las variables de decisión de cada Individuo.

Eventualmente se desea que el usuario implemente sus propias funciones, por ello es que, además de añadir el método en el listado localizado en **Controller/XML/Features.xml**, deberá implementar la siguiente función:

`calculate_distance(individual_i,individual_j,distance_parameters):`

Calcula la distancia de calidad que existe entre dos Individuos cualesquiera.
Dada la simpleza del método, se puede usar independientemente de las categorías antes especificadas.
Es importante resaltar que la función debe regresar un escalar que aluda a la distancia entre los Individuos.

Parameters

- **`individual_i`** (*`Instance`*) – El Individuo para calcular distancia.
- **`individual_j`** (*`Instance`*) – El Individuo para calcular distancia.
- **`distance_parameters`** (*`Dictionary`*) – Un diccionario que puede contener opciones adicionales para el cálculo de la distancia entre Individuos.

Returns Un valor escalar que indica la distancia entre los Individuos.

Return type Float

A continuación se muestran las subcategorías correspondientes:

GenotypicSimilarity (módulo)

La similaridad Genotípica **(ó Genotypic Similarity)**, es una subcategoría que calcula las distancias entre dos Individuos cualesquiera usando para ello características Genotípicas de éstos, lo cual quiere decir que se emplearán rasgos meramente internos endémicos de los Individuos.

Para fines del proyecto típicamente se utiliza el cromosoma y/o sus características asociadas, no obstante siendo sensatos con el término, el cromosoma no es la única herramienta que se puede usar sino cualquier rasgo interno.

HammingDistance (script)

La Distancia de Hamming **(ó Hamming Distance)** es una implementación perteneciente a la subcategoría Genotypic Similarity **(ó Similaridad Genotípica)**.
Esta consiste en comparar los alelos entre los cromosomas de los Individuos y devolver un valor numérico que indica en cuántos alelos los cromosomas de los Individuos resultaron tener valores diferentes.
Como consecuencia lógica, la magnitud de la Distancia de Hamming es inversamente proporcional a la calidad de los Individuos.

Es ampliamente usada para la Representación Cromosómica **(véase Model/ChromosomalRepresentation)** de tipo Binario **(ó Binary)**, aunque su uso no se limita sólo a esta codificación.

`calculate_distance`(*individual_i*, *individual_j*, *distance_parameters*)
Con base en la información proporcionada anteriormente, se implementa el cálculo de la distancia entre dos Individuos apoyándose de la técnica conocida como Distancia de Hamming **(ó Hamming Distance)**.

PhenotypicSimilarity (módulo)

La Similaridad Fenotípica **(ó Phenotypic Similarity)** es una subcategoría que calcula las distancias entre cualesquiera dos Individuos usando características concernientes al Fenotipo, es decir, rasgos exteriores de los Individuos.
Para fines relativos al proyecto, dichos atributos tradicionalmente no son otra cosa que las funciones objetivo evaluadas de cada Individuo, usando para ello las variables de decisión que cada uno lleva consigo.

Aún considerando lo anterior, siendo más generales, cualquier característica externa que se relacione con el Individuo puede ser utilizada.

EuclideanDistance (script)

La Distancia Euclidiana **(ó Euclidean Distance)** es una implementación de cálculo de distancia entre dos Individuos que pertenece a la subcategoría Phenotypic Similarity **(ó Similaridad Fenotípica)**

La forma de hacer el cálculo es la siguiente:
Supongamos que tenemos los vectores $U = (u_1, u_2, ..., u_n)$ y $V = (v_1, v_2, ..., v_n)$. Entonces la Distancia Euclidiana se define como:

$$d_E(U, V) = \sqrt{(v_1 - u_1)^2 + (v_2 - u_2)^2 + ... + (v_n - u_n)^2}$$

Para los fines que nos conciernen, los vectores $U\ y\ V$ serán las evaluaciones en las funciones objetivos de cada Individuo participante.

Finalmente es menester mencionar que, aunque tradicionalmente esta técnica se usa para Representaciones Cromosómicas **(véase Model/ChromosomalRepresentation)** de tipo FloatPoint **(ó Punto Flotante)**, en sentido estricto no se encuentra limitada sólo a este tipo de codificación.

calculate_distance(*individual_i*, *individual_j*, *distance_parameters*)
Apoyándose de la técnica conocida como Distancia Euclidiana **(ó Euclidean Distance)** se implementa el cálculo de la distancia para dos Individuos cualesquiera.

2.2.6 MOEA (módulo)

En esta parte se encuentran desarrolladas todas las técnicas concernientes al uso de M.O.E.A.'s **(Multi Objective Evolutionary Algorithms ó Algoritmos Evolutivos Multi Objetivo)**.

Un M.O.E.A. es la convergencia y culminación de todas las técnicas que se han implementado en la sección Model **(ó Modelo)** con la finalidad de ofrecer una solución óptima ante un problema multiobjetivo mediante el uso de Algoritmos Genéticos.

Primero, solucionar un problema multiobjetivo aterrizado en un lenguaje matemático consiste en lo siguiente:
Tenemos un vector de funciones objetivo:

$$F(\vec{x}) = [f_1(\vec{x}), f_2(\vec{x}), ..., f_n(\vec{x})]^T;\ con\ n \geqslant 1.$$

Donde:

$$\vec{x} = [x_1, x_2, ..., x_k]^T;\ k \geqslant 1.$$

Representa al vector de variables de decisión que "alimenta" a cada función objetivo.
La meta es encontrar un vector especial de variables de decisión, llamémosle:

$$\vec{x*} = [x_1*, x_2*, ..., x_k*]^T;\ k \geqslant 1.$$

Tal que:

$$f_i(\vec{x*}) \leqslant f_i(\vec{x});\ 1 \leqslant i \leqslant n;\ \forall f \in F.$$

Dicho de otra forma, se debe encontrar el vector de variables de decisión que minimize todas y cada una de las funciones objetivo en existencia.
Adicionalmente, todo vector de variables de decisión debe estar sujeto a las restricciones:

$$h_i(\vec{x}) = 0;\ 1 \leqslant i \leqslant p\ \ (restricciones\ de\ igualdad).$$
$$g_i(\vec{x}) \leqslant 0;\ 1 \leqslant i \leqslant m\ \ (restricciones\ de\ desigualdad).$$

Las cuales para fines de este proyecto son aquéllas a las que se encuentran afianzadas las variables de decisión **(véase View/Main/DecisionVariable/VariableFrame.py)**

Una definición adicional que sin lugar a dudas se verá utilizada es la de *dominancia* entre vectores de variables de decisión, para ello tomemos dos vectores $U = (u_1, u_2, ..., u_k)$ y $V = (v_1, v_2, ..., v_k)$, se dice que **U domina a V ó V es dominada por U** si:

$$\forall i \in \{1, ..., k\} \;\; u_i \leqslant v_i \wedge \exists i \in \{1, ..., k\}; \;\; u_i < v_i.$$

Lo anterior significa que U debe ser mejor que V en cada uno de sus componentes para garantizar la dominancia.
La simbología que se suele usar para identificar este hecho es $u \succ v$.

Algo importante a mencionar es que en las definiciones se trata únicamente la minimización de funciones objetivo porque, en caso de querer la maximización, simplemente se realiza la sustitución:

$$f_i'(\vec{x}) = -f_i(\vec{x}); \; 1 \leqslant i \leqslant n, \; para \; alguna \; f \in F.$$

Es decir, minimizando la función negativa se obtiene el máximo. El proyecto ya contempla este tipo de casos **(véase View/Main/ObjectiveFunction/FunctionFrame)**.
Como dato adicional, es menester añadir que, en un escenario típico muchas de las funciones objetivo entrarán en conflicto, esto quiere decir que en algunas se buscará el mínimo mientras que en otras, el máximo.

Con base en lo anterior, el funcionamiento de un M.O.E.A. **(resolver un problema de optimización multiobjetivo usando algoritmos genéticos)** generalmente se lleva a cabo de la siguiente manera:

1.- Usando una Representación Cromosómica **(véase Model/ChromosomalRepresentation)**, crear la Población Padre y evaluar cada uno de los Individuos respecto a las funciones objetivo.

2.- Asignar un Ranking a los Individuos de la Población Padre **(véase Model/Community/Community.py)**.

3.- Con base en el Ranking, asignar el Fitness a cada uno de los Individuos **(véase Model/Fitness)**.

4.- Tomando en cuenta el Fitness, aplicar las operaciones de Selección, Cruza y Mutación con la finalidad de crear una Población Hija **(véase Model/GeneticOperator)**. Todos los métodos empleados en este punto deben funcionar acorde a la Representación Cromosómica del punto 1.

5- **(Opcional)** Utilizar el Fitness Compartido para aplicar una elección más minuciosa de los mejores Individuos en la Población Hija **(véase Model/SharingFunction)**.

6.- Designar a la población Hija como la nueva población Padre.

7.- Repetir los pasos 2 a 6 hasta haber alcanzado un número límite de generaciones **(iteraciones)**.

A grandes rasgos la diferencia entre un M.O.E.A. y otro es la Presión Selectiva **(ó Selective Pressure)** que se aplica durante el procedimiento, para fines de este proyecto se trata de la tolerancia para seleccionar a los Individuos de calidad media o baja frente a los mejores. Una baja Presión Selectiva permite elegir Individuos no tan aptos; el caso es análogo para una alta Presión Selectiva.
Es por eso que se han tomado los M.O.E.A.'s más representativos, pues se desea ilustrar la consistencia y eficacia de dichos métodos en general a través de variadas circunstancias.

Tomando en cuenta lo anterior, la finalidad es que el usuario desarrolle sus propios M.O.E.A.'s, por ello es que, además de además de añadir el método en el listado localizado en **Controller/XML/Features.xml**, deberá implementar la siguiente función:

execute_moea(execution_task_count,generations_queue,generations,population_size,vector_func
community_instance,algorithm_parameters,representation_instance,representation_parameters,
sharing_function_instance,sharing_function_parameters,selection_instance,selection_paramete
mutation_instance,mutation_parameters):

Devuelve la solución óptima para un conjunto de funciones objetivo **vector_functions** ligadas a un conjunto de restricciones **vector_variables** tomando como fundamento el uso de algoritmos genéticos.
El método se apoya de las características subyacentes; en lo concerniente a la devolución de resultados se recomienda ver el método **get_results** localizado en **Model/Community/Community.py**.

Parameters

- **execute_task_count** (*Integer*) – El identificador que se utiliza para orquestar el orden en que el método será ejecutado con respecto de los demás **(véase View/Additional/ResultsGrapher/ResultsGrapherTopLevel.py)**.
- **generations_queue** (*Instance*) – Una estructura auxiliar **(Queue o Cola)** que es necesaria para indicar a la interfaz gráfica el progreso del método **(véase Controller/Controller.py, View/MainWindow.py, View/Additional/ResultsGrapher/ResultsGrapherTopLevel.py)** .
- **generations** (*Integer*) – El número de generaciones **(iteraciones)** que se emplearán para la ejecución del método.
- **population_size** (*Integer*) – El tamaño de la Población **(número de Individuos)**.
- **vector_functions** (*List*) – El vector con las funciones objetivo insertadas por el usuario.
- **vector_variables** (*List*) – El vector con las variables de decisión ingresadas por el usuario.
- **available_expressions** (*Dictionary*) – Un diccionario con expresiones creadas para que la evaluación de funciones objetivo sea mucho más sencilla **(véase Controller/Verifier.py, Controller/XML/PythonExpressions.xml, View/Additional/MenuInternalOption/InternalOptionTab/PythonExpressionFrame.py)**.
- **number_of_decimals** (*Integer*) – La precisión decimal **(número de decimales)** que tendrán las soluciones inherentes a los Individuos.
- **community_instance** (*Instance*) – Una instancia de la clase Community **(véase Controller/Verifier.py, Model/Community/Community.py)**.
- **algorithm_parameters** (*Instance*) – Un diccionario para añadir opciones adicionales para los M.O.E.A.'s.

- **representation_instance** (*Instance*) – Una instancia de la técnica de Representación Cromosómica **(ó Chromosomal Representation)** usada por el usuario **(véase Controller/Verifier.py, Model/ChromosomalRepresentation)**.
- **representation_parameters** (*Dictionary*) – Un diccionario con opciones adicionales a la técnica de Representación Cromosómica usada.
- **fitness_instance** (*Instance*) – Una instancia de la técnica de Fitness seleccionada por el usuario **(véase Controller/Verifier.py, Model/Fitness)**.
- **fitness_parameters** (*Dictionary*) – Un diccionario con parámetros adicionales para la técnica de Fitness utilizada.
- **sharing_function_instance** (*Instance*) – Una instancia de la técnica de Sharing Function **(ó Función de Compartición)** usada por el usuario **(véase Controller/Verifier.py, Model/SharingFunction)**.
- **sharing_function_parameters** (*Dictionary*) – Un diccionario con opciones adicionales para la técnica de Sharing Function seleccionada.
- **selection_instance** (*Instance*) – Una instancia de la técnica de Selection **(ó Selección)** seleccionada por el usuario **(véase Controller/Verifier.py, Model/Operator/Selection)**.
- **selection_parameters** (*Dictionary*) – Un diccionario con opciones adicionales para la técnica de Selection empleada.
- **crossover_instance** (*Instance*) – Una instancia de la técnica de Crossover **(ó Cruza)** seleccionada por el usuario **(véase Controller/Verifier.py, Model/Operator/Crossover)**.
- **crossover_parameters** (*Dictionary*) – Un diccionario con parámetros adicionales para la técnica de Cruza solicitada.
- **mutation_instance** (*Instance*) – Una instancia de la técnica de Mutation **(ó Mutación)** empleada por el usuario **(véase Controller/Verifier.py, Model/Operator/Mutation)**.
- **mutation_parameters** – Un diccionario con parámetros adicionales para la técnica de Mutación usada.

Returns El diccionario que resulta de aplicar el método **get_results** que se encuentra en **Model/Community/Community.py**.

Return type Dictionary

A continuación se muestra la lista de los M.O.E.A.'s implementados:

VEGA (script)

Se implementa la técnica M.O.E.A conocida como V.E.G.A. **(Vector Evaluated Genetic Algorithm ó Algoritmo Genético de Vectores Evaluados)**.
La forma de proceder del algoritmo es la siguiente:

1.- Se crea la Población Padre (de tamaño n).

2.- Tomando en cuenta las k funciones objetivo y la Población Padre, se crean k subpoblaciones de tamaño n/k cada una, si este número llega a ser irracional se pueden hacer ajustes con respecto de la distribución de los Individuos.

3.- Por cada subpoblación, se aplica la técnica de Selección y obtienen los n/k Individuos, terminado esto se deben unificar todos los seleccionados de nuevo en una súper Población.

4.- Con la súper Población del paso 3, se crea a la población Hija, la cual pasará a convertirse en la la nueva Población Padre.

5.- Se repiten los pasos 2 a 4 hasta haber alcanzado el número de generaciones **(iteraciones)** límite.

Como se puede apreciar es una implementación muy sencilla de optimización multiobjetivo, sin embargo el inconveniente que tiene es la fácil pérdida de material genético valioso.
Lo anterior significa que un Individuo que en una generación previa era el mejor para una función objetivo i al momento de ser separado y seleccionado en una subpoblación j (y por ende analizado bajo la función objetivo j) puede ser muy malo en calidad y por tanto no ser seleccionado; perdiéndose la ganancia genética hasta el momento obtenida para la función objetivo i; $i \neq j$.

Por ello es que se puede decir que V.E.G.A. genera soluciones promedio que destacan con una calidad media para todas las funciones objetivo.

Finalmente hay que comentar que para este algoritmo no se requiere aplicar un Ranking específico, no obstante, se ha decidido utilizar el de Fonseca & Flemming **(véase Model/Community/Community.py)** pues es el más sencillo de implementar.

create_subpopulations(*comunidad*, *main_population*)
Método que divide a la Población principal en subpoblaciones de acuerdo al número de funciones objetivo.

Parameters

- **comunidad** (*Instance*) – Una instancia de Community para poder crear poblaciones..
- **main_population** (*Instance*) – La Población que será dividida.

Returns Una lista con las subpoblaciones **(de tipo Population)**.

Return type List

execute_moea(*execution_task_count*, *generations_queue*, *generations*, *population_size*, *vector_functions*, *vector_variables*, *available_expressions*, *number_of_decimals*, *community_instance*, *algorithm_parameters*, *representation_instance*, *representation_parameters*, *fitness_instance*, *fitness_parameters*, *sharing_function_instance*, *sharing_function_parameters*, *selection_instance*, *selection_parameters*, *crossover_instance*, *crossover_parameters*, *mutation_instance*, *mutation_parameters*)
De acuerdo a la información proporcionada con anterioridad, se implementa el método que representa a la técnica M.O.E.A. conocida como V.E.G.A. **(Vector Evaluated Genetic Algorithm ó Algoritmo Genético de Vectores Evaluados)**.

SPEAII (script)

Se desarrolla la implementación de la técnica M.O.E.A. conocida como S.P.E.A. II **(Strength Pareto Evolutionary Algorithm ó Algoritmo Evolutivo de Fuerza de Pareto)**.

El funcionamiento del algoritmo es el siguiente:

1.- Se inicializa una población llamada *P* y un conjunto inicialmente vacío llamado *E* **(E albergará Individuos también)**; ambos son de tamaño n.

2.- Se asigna el Fitness a los Individuos de *P* y *E* **(para ello se evalúan las funciones objetivo de los Individuos de ambos conjuntos y se asigna el Ranking Zitzler & Thiele)**.

3.- A continuación se funden *P* y *E* en una súper Población **(llamémosle S también señalado en el algoritmo como Mating Pool, de tamaño n)**.Para ello primero se añaden los Individuos *NO DOMINADOS* de *P* en *S* y posteriormente los *NO DOMINADOS* de *E* en *S*.
Aquí se distinguen dos casos:

- Si llegasen a faltar Individuos se añaden al azar Individuos *DOMINADOS* de *P* en *S* hasta completar la demanda.
- Si después de la fusión el número de Individuos supera a n, entonces se hace un truncamiento en *S* hasta ajustar su tamaño a n.

4.- *S* será la nueva *E*, además se crea la población Hija de la recién creada *E* **(E-Child)**.

5.- E-Child será la nueva P.

6.- Se repiten los pasos 2 a 5 hasta que se haya alcanzado el límite de generaciones **(iteraciones)**.

Finalmente lo que se regresa es *E*, ya que ahí es donde se han almacenado los mejores Individuos de todas las generaciones.

La característica de este algoritmo es que tiene una Presión Selectiva alta ya que se da prioridad a los Individuos no dominados **(de ahí el nombre de Fuerza de Pareto ó los más fuertes con respecto al principio de Pareto)**, y el hecho de mezclar a *E* y *P* en una súper Población garantiza la conservación de los mejores Individuos sin importar el transcurso de las generaciones **(a eso se le conoce como Elitismo)**, pero también da una tolerancia, aunque mínima, a los Individuos de menor calidad como en el punto 3.
Además al momento de actualizar *S* a *E* y E-Child a *P* se tiene una especie de seguro de vida, es decir, si en algún momento la población E-Child llegara a tener una calidad baja se tiene el respaldo de *E* para una generación posterior para formar *S*.

Se debe tener en cuenta que el algoritmo originalmente no contempla ni una súper Población *S* ni E-Child sino que en los pasos 3 y 4 se utiliza solamente *E* para referirse tanto a E-child como a *S*, sin embargo para no confundir al usuario en la funcionalidad del método se decidió colocar contenedores extra para poder diferenciar más precisamente a los elementos involucrados.

Algo muy importante a mencionar es que en el paso 1 y al momento de crear la población E-Child es necesario evaluar las funciones objetivo, asignar un Ranking y posteriormente un Fitness para que se puedan aplicar los operadores geneticos **(véase Model/GeneticOperator)**, para este caso el Ranking es estrictamente el de Zitzler & Thiele; la descripción completa de éste se encuentra en **Model/Community/Community.py**.

execute_moea(*execution_task_count*, *generations_queue*, *generations*, *population_size*, *vector_functions*, *vector_variables*, *available_expressions*, *number_of_decimals*, *community_instance*, *algorithm_parameters*, *representation_instance*, *representation_parameters*, *fitness_instance*, *fitness_parameters*, *sharing_function_instance*, *sharing_function_parameters*, *selection_instance*, *selection_parameters*, *crossover_instance*, *crossover_parameters*, *mutation_instance*, *mutation_parameters*)

Con base en la información señalada se lleva a cabo la implementación del M.O.E.A. conocido como S.P.E.A. II **(Strength Pareto Evolutionary Algorithm ó Algoritmo Evolutivo de Fuerza de Pareto)**.

MOGA (script)

Se desarrolla la técnica M.O.E.A. que lleva por nombre M.O.G.A. **(Multi Objective Genetic Algorithm ó Algoritmo Genético Multi Objetivo)**.
Su funcionamiento es el siguiente:

1.- Se crea la Población Padre, se evalúan las funciones objetivo de sus correspondientes Individuos.

2.- Se asigna a los Individuos un Ranking **(Fonseca & Flemming)** y posteriormente se calcula el Niche Count de la Población Padre.

3.- Tomando en cuenta los valores del punto 2 se obtiene el Fitness para cada Individuo y posteriormente su Shared Fitness.

4.- Se aplica el operador de selección sobre la Población Padre para determinar los elegidos para dejar descendencia.

5.- Se crea la Población Hija, se evalúan las funciones objetivo de sus correspondientes Individuos.

6.- Se asigna a los Individuos un Ranking **(Fonseca & Flemming)** y posteriormente se calcula el Niche Count de la Población Hija.

7.- Tomando en cuenta los valores del punto 6 se obtiene el Fitness para cada Individuo y posteriormente su Shared Fitness.

8.- La Población Hija pasará a ser la nueva Población Padre.

9.- Se repiten los pasos 4 a 8 hasta que se haya alcanzado el número límite de generaciones **(iteraciones)**.

Como se puede apreciar, la implementación de este algoritmo es muy sencilla, además se rige casi en su totalidad por el Shared Fitness **(ó Fitness Compartido)**, por lo que la Presión Selectiva **(ó Selective Pressure)** incluida dependerá en gran medida de la función de Distancia que se utilice, así como de la magnitud indicada por el usuario.

Finalmente es menester mencionar que para esta implementación el Ranking utilizado debe ser estrictamente el de Fonseca & Flemming **(véase Model/Community/Community.py)**.

execute_moea(*execution_task_count*, *generations_queue*, *generations*, *population_size*, *vector_functions*, *vector_variables*, *available_expressions*, *number_of_decimals*, *community_instance*, *algorithm_parameters*, *representation_instance*, *representation_parameters*, *fitness_instance*, *fitness_parameters*, *sharing_function_instance*, *sharing_function_parameters*, *selection_instance*, *selection_parameters*, *crossover_instance*, *crossover_parameters*, *mutation_instance*, *mutation_parameters*)

Tomando como referencia el pseudocódigo antes citado, se elabora la implementación de M.O.G.A. **(Multi Objective Genetic Algorithm ó Algoritmo Genético Multi Objetivo)**.

NSGAII (script)

En esta parte se lleva a cabo la implementación del M.O.E.A. denominado N.S.G.A. II **(Non-dominated Sorting Genetic Algorithm ó Algoritmo Genético de Ordenamiento No Dominado)**.

La forma de proceder del método es la siguiente:

1.- Se crea una Población Padre **(de tamaño n)**, a la cual se le evalúan las funciones objetivo de sus Individuos, se les asigna un Ranking **(Goldberg)** y posteriormente se les otorga un Fitness.

2.- Con base en la Población Padre se aplica el operador de Selección para elegir a los Individuos que serán aptos para reproducirse.

3.- Usando a los elementos del punto 2, se crea una Población Hija **(de tamaño n)**.

4.- Se crea una súper Población **(llamémosle S, de tamaño 2n)** que albergará todos los Individuos tanto de la Población Padre como Hija; a *S* se le evalúan las funciones objetivo de sus Individuos, se les asigna un Ranking **(Goldberg)** y posteriormente se les otorga un Fitness.

5.- La súper Población *S* se divide en subcategorías de acuerdo a los niveles de dominancia que existan, es decir, existirá la categoría de dominancia 0, la cual almacena Individuos que tengan una dominancia de 0 Individuos **(ningún Individuo los domina)**, existirá la categoría de dominancia 1 con el significado análogo y así sucesivamente hasta haber cubierto todos los niveles de dominancia existentes.

6.- Se construye la nueva Población Padre, pare ello constará de los Individuos de *S* donde la prioridad será el nivel de dominancia, es decir, primero se añaden los elementos del nivel 0,luego los del nivel 1 y así en lo sucesivo hasta haber adquirido n elementos. Se debe aclarar que la adquisición de Individuos por nivel debe ser total, esto significa que no se pueden dejar Individuos sueltos para el mismo nivel de dominancia.

Supongamos que a un nivel k existen tantos Individuos que su presunta adquisición supera el tamaño n, en este caso se debe hacer lo siguiente:

> 6.1.- Se crea una Población provisional **(Prov)** con los Individuos del nivel k, se evalúan las funciones objetivo a cada uno de sus Individuos, se les asigna un Ranking **(Goldberg)** y posteriormente se les asigna el Fitness.
>
> Con los valores anteriores se calcula el Niche Count **(véase Model/SharingFunction)** de los Individuos; una vez hecho ésto se seleccionan desde Prov los Individuos faltantes con los mayores Niche Count, esto hasta completar el tamaño n de la nueva Población Padre.

7.- Al haber conformado la nueva Población Padre, se evalúan las funciones objetivo de sus Individus, se les asigna el Ranking correspondiente **(Goldberg)** y se les atribuye su Fitness.

8.- Se repiten los pasos 2 a 7 hasta haber alcanzado el límite de generaciones **(iteraciones)**.

Como su nombre lo indica, la característica de este algoritmo es la clasificación de los Individuos en niveles para su posterior selección.

Esto al principio propicia una Presión Selectiva moderada por la ausencia de elementos con dominancia baja que suele existir en las primeras generaciones, sin embargo en iteraciones posteriores se agudiza la Presión Selectiva ya que eventualmente la mayoría de los Individuos serán alojados en las primeras categorías de dominancia, cubriendo casi instantáneamente la demanda de Individuos necesaria en el paso 6, por lo que las categorías posteriores serán cada vez menos necesarias con el paso de los ciclos.

Por otra parte la fusión de las Poblaciones en *S* garantiza que siempre se conserven a los mejores Individuos independientemente de la generación transcurrida, a eso se le llama Elitismo.
Por cierto que en el algoritmo original no existe un nombre oficial para *S* sino más bien se señala como una estructura genérica, sin embargo se le ha formalizado con un identificador para guiar apropiadamente al usuario en el flujo del algoritmo.

Para finalizar se señala que el uso del ranking de Goldberg **(véase Model/Community/Community.py)** es indispensable.

execute_moea(*execution_task_count*, *generations_queue*, *generations*, *population_size*, *vector_functions*, *vector_variables*, *available_expressions*, *number_of_decimals*, *community_instance*, *algorithm_parameters*, *representation_instance*, *representation_parameters*, *fitness_instance*, *fitness_parameters*, *sharing_function_instance*, *sharing_function_parameters*, *selection_instance*, *selection_parameters*, *crossover_instance*, *crossover_parameters*, *mutation_instance*, *mutation_parameters*)

Con base en los datos recabados se desarrolla la técnica M.O.E.A. que lleva por nombre N.S.G.A. II **(Non-dominated Sorting Genetic Algorithm ó Algoritmo Genético de Ordenamiento No Dominado)**-

2.3 View (sección)

La capa View **(ó Vista)** contiene todos los elementos que serán alusivos a la interfaz gráfica. De acuerdo al modelo MVC **(Model-View-Controller)**, opera exclusivamente con la capa Controller **(ó Controlador)**.

2.3.1 MainWindow (clase)

class MainWindow

Aquí es donde se mezclan todas las estructuras gráficas que forman parte de la sección View **(ó vista)**.
Se trata de una Ventana que contendrá todas las opciones necesarias para que el usuario pueda ejecutar a voluntad M.O.E.A.'s **(Multi Objective Evolutionary Algorithm ó Algoritmo Evolutivo Multi Objetivo)**

El flujo que se suele seguir es el siguiente:

- El usuario ingresa las características que desea que contenga el M.O.E.A. que será ejecutado.
- Posteriormente el Controller **(ó Controlador, véase Controller/Controller.py)** verifica la consistencia de los datos anteriores para que no haya conflicto en el lado del Model **(ó Modelo)**.
- Si no existe problema alguno se prosigue con el proceso, en otro caso se arroja un mensaje de error.
- Siguiendo con el flujo normal se ejecutará una instancia del M.O.E.A. solicitado en la capa de Model **(ó Modelo)**, la cual tendrá una ventana asociada en View **(ó Vista)** que indicará el progreso del primero.

- Cuando una instancia termine de ejecutarse, la ventana del progreso desaparece y en su lugar se muestra otra conteniendo los resultados del M.O.E.A. **(véase View/Additional/ResultsGrapher/ResultsGrapherToplevel.py)**.

Es importante mencionar que esta clase y el proyecto en general están diseñados para que se puedan crear varias instancias simultáneamente, con ello se espera aprovechar al máximo los recursos computacionales en los que el proyecto fuera a ejecutarse.

Returns Tkinter.Frame

Return type Instance

_MainWindow__change_frame(*current_frame_name*)

Note: Este método es privado.

Hace el cambio en la Ventana Principal ocultando un Frame y mostrando otro.

Parameters **current_frame_name** (*Tkinter.Frame*) – El Frame que se va a mostrar en la Ventana Principal.

_MainWindow__check_queues()

Note: Este método es privado.

Una vez iniciado un proceso que ejecuta un M.O.E.A., este método revisa periódicamente las colas **(Queues)** sobre las cuales los procesos escribirán todo tipo de información pertinente.

_MainWindow__get_information()

Note: Este método es privado.

Método que obtiene los datos ingresados por el usuario de cada uno de los Frames asociados a esta Ventana Principal.

Returns Un diccionario con toda las preferencias del usuario recolectadas para cada uno de los Frames disponibles.

Return type Dictionary

_MainWindow__init_procedure(*event*)

Note: Este método es privado.

Inicia el procedimiento para ejecutar un M.O.E.A.

Los pasos que se realizan son:

- Recolecta las preferencias ingresadas por el usuario en los Frames que conforman la Ventana Principal.
- Se sanitizan dichos datos con ayuda del Controller.
- En caso de no haber problemas con la sanitización, se ejecuta el proceso alojándolo en un hilo **(Thread)** para que permita seguir teniendo acceso a la Ventana Principal; por el contrario si hubo alguna falla regresa un mensaje de error.

Gracias a este método el proyecto entero tiene la característica de ser Multi-Hilo **(ó Multi-Threading)**, es decir, se pueden ejecutar varios procedimientos de manera independiente.

Parameters **event** (*String*) – El evento del elemento gráfico que activa esta función.

_MainWindow__initialize_frames()

Note: Este método es privado.

Método que inicializa los Frames que se colocarán en la Ventana Principal como opciones.

_MainWindow__load_images()

Note: Este método es privado.

Carga las imágenes que se encuentran en el directorio View/Images para que puedan ser usadas por los Frames.

Returns Un diccionario con todas las imágenes cargadas.

Return type Dictionary

_MainWindow__obtain_results(*execution_task_count*, *generations_queue*, *gathered_information*, *sanitized_information*)

Note: Este método es privado.

Ejecuta un M.O.E.A.. Esta es la función que se coloca en un hilo para ser llevada a cabo de manera independiente con la finalidad de dejar libre la Ventana Principal y de manera secundaria ejecutar varios procedimientos simultáneamente.

Parameters

- **execution_task_count** (*Integer*) – El número de proceso actual.
- **generations_queue** (*Instance*) – Una referencia a una cola **(Queue)** donde los procesos escribirán su avance en cuanto a las generaciones transcurridas.
- **gathered_information** (*Dictionary*) – La información que el usuario ingresó al momento de iniciar el proceso actual.
- **sanitized_information** (*Dictionary*) – La información anterior sanitizada.

_MainWindow__restore_settings(*event*)

> **Note:** Este método es privado.

Limpia y deja por defecto los valores estándar del Frame mostrado actualmente en la Ventana Principal. El método no aplica para regresar a M.O.P.'s **(Multi Objective Problems)** cargados anteriormente.

Parameters **event** (*String*) – El evento del elemento gráfico que acciona esta función.

_MainWindow__update_frame(*event*)

> **Note:** Este método es privado.

Muestra en la Ventana Principal el Frame actual.

Parameters **event** (*String*) – El evento del elemento gráfico que activa la función.

load_mop_example(*elements*)
Carga el M.O.P **(Multi Objective Problem)** seleccionado a los Frames correspondientes **(Objective Functions y Decision Variables)**.

Parameters **elements** (*Array*) – Un arreglo que contiene dos elementos, el primero son las funciones objetivo precargadas mientras que el segundo son las variables de decisión también precargadas. Ambas provienen del menú secundario **(véase View/Additional/MenuInternalOption/InternalOptionTab/MOPExampleFrame.py, Controller/XML/MOPExamples.xml)**.

resource_path(*relative_path*)
Esta función se utiliza para poder crear ejecutables apropiadamente. A grandes rasgos el ejecutable se empaqueta en un directorio llamado _MEIPASS, entonces aquí se implementa la búsqueda de dicho archivo devolviendo un path **(ruta)**.

Returns La ruta del directorio _MEIPASS.

Return type String

run()
Lanza la Ventana Principal.

2.3.2 Main (módulo)

Contiene todos los elementos gráficos para que el usuario pueda configurar los atributos que intervienen en la ejecución de un M.O.E.A.

Home (módulo)

Contiene toda la información posible para poder describir tanto los elementos que conforman el programa como su correcto uso.

IntroductionFrame (clase)

class **IntroductionFrame** (*parent*, *canvas_function*, *features*)
Bases: Tkinter.Frame

Contiene información básica y concisa sobre el producto de software, la cual es organizada y mostrada de acuerdo al número de secciones existentes en éste.
De manera secundaria proporciona la infraestructura para poder darle al usuario un desplazamiento más rápido entre dichas secciones.

Parameters

- **parent** (*Tkinter.Frame*) – Frame padre al que pertenece.
- **canvas_function** (*Instance*) – Una función alusiva al funcionamiento del Canvas.
- **features** (*Dictionary*) – Conjunto de técnicas con sus respectivos parámetros para que se puedan cargar automáticamente en este Frame.

Returns Tkinter.Frame

Return type Instance

_IntroductionFrame__go_to_selected_section (*event*)

Note: Este método es privado.

Con base en la liga elegida por el usuario, realiza el desplazamiento hacia la sección correspondiente.

Parameters **event** (*String*) – Elemento que ejecutó esta función.

HomeFrame (clase)

class **HomeFrame** (*parent*, *features*)
Bases: Tkinter.Frame

Unifica dos elementos: Canvas e IntroductionFrame.
La razón de haber hecho esto es que, cuando se agregan muchas funciones al FunctionFrame, se tiene que agregar una barra de desplazamiento para poder acceder a los que se encuentran hasta abajo. Dentro del ambiente de Tkinter, el elemento más sencillo para lograr esto es un Canvas, por ello se anida el IntroductionFrame al Canvas.

Parameters

- **parent** (*Tkinter.Frame*) – Frame padre al que pertenece.
- **features** (*Dictionary*) – Conjunto de técnicas con sus respectivos parámetros para que se puedan cargar automáticamente en este Frame **(véase Controller/XMLParser.py)**.

Returns Tkinter.Frame

Return type Instance

_HomeFrame__update_scrollbar(*event=None*)

Note: Este método es privado.

Actualiza la barra de desplazamiento de acuerdo al número de elementos existentes en el Frame, esto para poder hacer un recorrido apropiado de la barra.

Parameters **event** (*String*) – Elemento que ejecutó esta función.

move_to_section(*y_coordinate*)
Mueve la barra de desplazamiento (y por ende el contenido) con base en la coordenada (en Y) que se le pase como parámetro.

Parameters **y_coordinate** – Coordenada que se necesita para hace el desplazamiento. Oscila entre 0 y 1.

restore_settings()
Restaura la configuración del Frame a la que tenía por defecto.

DecisionVariable (módulo)

Proporciona los elementos gráficos para que el usuario pueda insertar, modificar y eliminar variables de decisión con sus respectivos rangos.

DecisionVariableFrame (clase)

class DecisionVariableFrame(*parent*, *features*)
Bases: Tkinter.Frame

Realiza la fusión de un Canvas y VariableFrame, debido a que, cuando se agregan numerosas variables al VariableFrame, se debe insertar una barra de desplazamiento para poder acceder a aquéllos que se encuentren hasta abajo. Dentro del ambiente de Tkinter, el elemento más sencillo para lograr este efecto es un Canvas, por ello se anida el VariableFrame al Canvas.

Parameters

- **parent** (*Tkinter.Frame*) – Frame padre al que pertenece.
- **features** (*Dictionary*) – Conjunto de técnicas con sus respectivos parámetros para que se puedan cargar automáticamente en este Frame **(véase Controller/XMLParser.py)**.

Returns Tkinter.Frame

Return type Instance

_DecisionVariableFrame__activate_scroll(*event*)

Note: Este método es privado.

Actualiza la barra de desplazamiento y con base en esta acción la activa o desactiva.

Parameters **event** (*String*) – Elemento que ejecutó esta función.

_DecisionVariableFrame__update_scrollbar(*event=None*)

Note: Este método es privado.

Actualiza la barra de desplazamiento de acuerdo al número de elementos existentes en el Frame, esto para poder hacer un recorrido apropiado de la barra.

Parameters **event** (*String*) – Elemento que ejecutó esta función.

get_information()
Regresa la información recabada en el Frame.

Returns Un diccionario que contiene una lista con las variables ingresadas.

Return type Dictionary

insert_mop_example(*variables*)

Inserta un M.O.P **(Multi Objective Problem ó Problema Multi Objetivo)**.
En este caso significa que se insertarán las variables con sus respectivos rangos en el Frame para poder hacer pruebas rápidas en el programa, habiendo antes limpiado por completo el contenido del Frame.
(véase Controller/XML/MOPExample.xml)
(véase View/Additional/MenuInternalOption/InternalOptionFrame.py).

Parameters **functions** (*List*) – Lista de variables para ser insertadas en el Frame.

restore_settings()
Restaura el contenido del Frame, en este caso significa que se eliminará todo lo que esté en éste y se dejará una casilla vacía libre.

VariableFrame (clase)

class VariableFrame(*parent*, *features*)
Bases: Tkinter.Frame

Proporciona bases gráficas para que el usuario pueda insertar variables de decisión, así como información relativa a éstas.
En términos generales, el usuario insertará casillas para ingresar variables de decisión, indicando también el valor mínimo y máximo que podrán tener.
Es importante comentar que todas las variables de decisión deben contener rangos finitos, es decir, no se contemplan valores infinitos, aunque algunos M.O.P.'s **(Multi Objective Problems ó Problemas Multi Objetivo)** manejan este tipo de rangos.

Parameters

- **parent** (*Tkinter.Frame*) – Frame padre al que pertenece.
- **features** (*Dictionary*) – Conjunto de técnicas con sus respectivos parámetros para que se puedan cargar automáticamente en este Frame **(véase Controller/XMLParser.py)**.

Returns Tkinter.Frame

Return type Instance

_VariableFrame__add_variable(*event*)

Note: Este método es privado.

Agrega una casilla al Frame. Esta función se usa si fue ejecutada por un evento.

Parameters **event** (*String*) – Identificador del elemento gráfico que activó la función.

_VariableFrame__delete_single_variable(*event*)

Note: Este método es privado.

Elimina una casilla y todos los elementos gráficos que la acompañan. También elimina todo rastro que se encuentre en las estructuras lógicas.

Parameters **event** (*String*) – Identificador del elemento gráfico que activó la función.

_VariableFrame__grid_widgets()

Note: Este método es privado.

Coloca elementos en el Frame.

get_current_elements()

Regresa el número actual de casillas en el Frame.

Returns Cantidad de elementos en la estructura rows, donde se guardan las casillas (Entries).

Return type Int

get_information()

Toma la información del Frame y regresa las variables con sus rangos que el usuario ingresó.

Returns Un diccionario que contiene una lista con las variables (y rangos) escritas.

Return type Dictionary

insert_mop_example(*variables*)

Inserta un M.O.P (Multi Objective Problem) que no es más que un conjunto de variables con sus rangos para que se pueda hacer más rápidamente una prueba.

Previo a ésto se limpia el Frame para insertar únicamente el M.O.P.

(véase Controller/XML/MOPExample.xml)

(véase View/Additional/MenuInternalOption/InternalOptionFrame.py).

Parameters **functions** (*List*) – Conjunto de variables para insertar en el Frame.

insert_variable(*variable=None*)

Coloca en el Frame una colección de elementos:

[casilla para insertar variable ,casilla de rango minimo, casilla de rango máximo, botón para eliminar]
Si el parámetro function es **None**, se añade la casilla vacía, de lo contrario se agrega ésta con la variable y sus rangos.

Parameters **function** (*String*) – Una terna (nombre de la variable, rango máximo, rango mínimo) para ser insertada en las casillas correspondientes.

restore_settings()
Restaura el contenido del Frame a sus valores por defecto. Esto significa que borrará cualquier contenido que se encuentre en existencia y dejará una casilla vacía.

ObjectiveFunction (módulo)

Proporciona los elementos gráficos para que el usuario pueda insertar, modificar y eliminar funciones objetivo.

ObjectiveFunctionFrame (clase)

class **ObjectiveFunctionFrame**(*parent*, *features*)
Bases: Tkinter.Frame

Unifica dos elementos: Canvas y FunctionFrame.
La razón de haber hecho esto es que, cuando se agregan muchas funciones al FunctionFrame, se tiene que agregar una barra de desplazamiento para poder acceder a los que se encuentran hasta abajo. Dentro del ambiente de Tkinter, el elemento más sencillo para lograr esto es un Canvas, por ello se anida el FunctionFrame al Canvas.

Parameters

- **parent** (*Tkinter.Frame*) – Frame padre al que pertenece.
- **features** (*Dictionary*) – Conjunto de técnicas con sus respectivos parámetros para que se puedan cargar automáticamente en este Frame **(véase Controller/XMLParser.py)**.

Returns Tkinter.Frame

Return type Instance

_ObjectiveFunctionFrame__activate_scroll(*event*)

Note: Este método es privado.

Actualiza la barra de desplazamiento y con base en esta acción la activa o desactiva.

Parameters **event** (*String*) – Elemento que ejecutó esta función.

_ObjectiveFunctionFrame__update_scrollbar(*event=None*)

Note: Este método es privado.

Actualiza la barra de desplazamiento de acuerdo al número de elementos existentes en el Frame, esto para poder hacer un recorrido apropiado de la barra.

Parameters **event** (*String*) – Elemento que ejecutó esta función.

get_information()
Regresa la información recabada en el Frame.

Returns Un diccionario que contiene una lista con las funciones escritas.

Return type Dictionary

insert_mop_example(*functions*)

Inserta un M.O.P (Multi Objective Problem).
En este caso significa que se insertarán funciones para poder hacer pruebas rápidas en el programa.
(véase Controller/XML/MOPExample.xml)
(véase View/Additional/MenuInternalOption/InternalOptionFrame.py).

Parameters **functions** (*List*) – Lista de funciones para ser insertadas en el Frame.

restore_settings()
Restaura el contenido del Frame, en este caso significa que se eliminará todo lo que esté en éste y se dejará una casilla vacía libre.

FunctionFrame (clase)

class FunctionFrame(*parent*, *features*)
Bases: Tkinter.Frame

Esta clase proporciona una base gráfica para que el usuario pueda agregar tantas functiones objetivo como desee.
A grandes rasgos el usuario podrá agregar casillas donde se colocarán las funciones objetivo, esto utilizando un botón. De igual manera, las casillas pueden ser eliminadas usando un ícono que estará cerca de cada una de éstas.
Importante es mencionar que el formato de las funciones deben estar escritas en sintaxis de Python.

Parameters

- **parent** (*Tkinter.Frame*) – Frame padre al que pertenece.
- **features** (*Dictionary*) – Conjunto de técnicas con sus respectivos parámetros para que se puedan cargar automáticamente en este Frame **(véase Controller/XMLParser.py)**.

Returns Tkinter.Frame

Return type Instance

_FunctionFrame__add_function(*event*)

Note: Este método es privado.

Agrega una casilla al Frame. Esta función se usa si fue ejecutada por un evento.

Parameters **event** (*String*) – Identificador del elemento gráfico que activó la función.

_FunctionFrame__delete_single_function(*event*)

Note: Este método es privado.

Elimina una casilla y todos los elementos gráficos que la acompañan. También elimina todo rastro que se encuentre en las estructuras lógicas.

Parameters **event** (*String*) – Identificador del elemento gráfico que activó la función.

_FunctionFrame__grid_widgets()

Note: Este método es privado.

Coloca elementos en el Frame.

get_current_elements()
Regresa el número actual de casillas en el Frame.

Returns Cantidad de elementos en la estructura rows, donde se guardan las casillas (Entries).

Return type Int

get_information()
Toma la información del Frame y regresa las funciones objectivo que el usuario insertó.

Returns Un diccionario que contiene una lista con las funciones escritas.

Return type Dictionary

insert_function(*function=None*)

Coloca en el Frame una colección de elementos:
[casilla para insertar funcion, opción de maximizar, opción de minimizar, botón para eliminar]
Si el parámetro function es **None**, se agrega la casilla vacía, de lo contrario se añade ésta con la función.

Parameters **function** (*String*) – Una función para ser insertada en el primer elemento de la colección.

insert_mop_example(*functions*)

Inserta un M.O.P (Multi Objective Problem) que no es más que un conjunto de funciones para que se pueda hacer más rápidamente una prueba.
Previo a ésto se limpia el Frame para insertar únicamente el M.O.P.
(véase Controller/XML/MOPExample.xml)
(véase View/Additional/MenuInternalOption/InternalOptionFrame.py).

Parameters **functions** (*List*) – Conjunto de funciones para insertar en el Frame.

restore_settings()
Restaura el contenido del Frame a sus valores por defecto. Esto significa que borrará cualquier contenido que se encuentre en existencia y dejará una casilla vacía.

Population (módulo)

Proporciona las estructuras gráficas para que el usuario pueda configurar atributos de la población.

PopulationFrame (clase)

class **PopulationFrame**(*parent*, *features*)

Bases: Tkinter.Frame

Unifica y mantiene un control sobre las clases PopulaceFrame y FitnessFrame, esto con el fin de poder colocar los elementos apropiadamente y agilizar el intercambio de información con el usuario.

Parameters

- **parent** (*Tkinter.Frame*) – Frame padre al que pertenece.
- **features** (*Dictionary*) – Conjunto de técnicas con sus respectivos parámetros para que se puedan cargar automáticamente en este Frame **(véase Controller/XMLParser.py)**.

Returns Tkinter.Frame

Return type Instance

get_information()

Toma la información propiciada en cada Frame y después la unifica para regresar un sólo conjunto de información.

Returns Un diccionario con la información de PopulaceFrame y FitnessFrame.

Return type Dictionary

restore_settings()

Restaura los valores por defecto en ambos Frames.

PopulaceFrame (clase)

class **PopulaceFrame**(*parent*, *name*, *features*)

Bases: *View.Main.Population.TemplatePopulation.TemplatePopulationFrame.TemplatePopulatio*

Esta clase proporciona la infraestructura gráfica para que el usuario pueda elegir métodos y características concernientes a la conformación de la población.

También hereda atributos de la clase TemplatePopulationFrame con el fin de establecer una forma más rápida y ordenada de colocar componentes y recolectar la información de éstos.

Parameters

- **parent** (*Tkinter.Frame*) – Frame padre al que pertenece.
- **name** (*String*) – Identificador **(único)** que tendrá el Frame.
- **features** (*Dictionary*) – Conjunto de técnicas con sus respectivos parámetros para que se puedan cargar automáticamente en este Frame **(véase Controller/XMLParser.py)**.

Returns Tkinter.Frame

Return type Instance

get_information()
Recolecta la información genérica **(usando el método de la clase Padre)**, y también se le añade aquélla recolectada exclusivamente en esta clase.

Returns

Un diccionario que contiene:
Métodos genéricos
Número de Generaciones,
Tamaño de la Población,
Número de Decimales

Return type Dictionary

restore_settings()
Por un lado, restaura el contenido de los elementos pertenecientes sólo a esta clase, y por el otro, activa el método de la clase Padre que realiza una restauración de los elementos genéricos.

FitnessFrame (clase)

class FitnessFrame(*parent*, *name*, *features*)
Bases: *View.Main.Population.TemplatePopulation.TemplatePopulationFrame.TemplatePopulatio*

Esta clase proporciona la infraestructura gráfica para que el usuario pueda elegir métodos concernientes a la asignación del Fitness para la población.
Además hereda atributos de la clase TemplatePopulationFrame para facilitar la colocacion y extracción de información pertinente para el usuario.

Parameters

- **parent** (*Tkinter.Frame*) – Frame padre al que pertenece.
- **name** (*String*) – Identificador **(único)** que tendrá el Frame.
- **features** (*Dictionary*) – Conjunto de técnicas con sus respectivos parámetros para que se puedan cargar automáticamente en este Frame **(véase Controller/XMLParser.py)**.

Returns Tkinter.Frame

Return type Instance

get_information()
Llama al método de la clase Padre, el cual recopila toda la información elegida por el usuario y la regresa en forma de diccionario.

Returns Diccionario con información de los métodos genéricos.

Return type Dictionary

restore_settings()
Llamar al método de la clase Padre, el cual restaura los valores por defecto de los elementos dinámicos y estáticos del Frame.

TemplatePopulationFrame (clase)

class **TemplatePopulationFrame**(*parent*, *name*, *features*)
Bases: `Tkinter.Frame`

Esta clase proporciona la infraestructura gráfica para que el usuario pueda elegir técnicas y configurar atributos concernientes al Fitness de una población y a la población en general.
A grandes rasgos se trata de una plantilla que deberán implementar las clases FitnessFrame y PopulaceFrame.
La clase permite la selección de cada posible técnica disponible y automáticamente se muestran los parámetros necesarios **(si los hay)** para cada una de éstas.

Parameters

- **parent** (*Tkinter.Frame*) – Frame padre al que pertenece.
- **name** (*String*) – Identificador (único) que tendrá el Frame.
- **features** (*Dictionary*) – Conjunto de técnicas con sus respectivos parámetros para que se puedan cargar automáticamente en este frame **(véase Controller/XMLParser.py)**.

Returns Tkinter.Frame

Return type Instance

_TemplatePopulationFrame__create_dynamic_widgets()

Note: Este método es privado.

Inicializa los elementos dinámicos del Frame, esto es, de acuerdo al tipo que lleva cada parámetro se creará un widget diferente.

_TemplatePopulationFrame__update_widgets(*event=None*)

Note: Este método es privado.

Realiza solamente la actualización y colocación de elementos dinámicos en el Frame.
Si el parámetro event es distinto de **None**, significa que se lanzó un evento que provocará que se actualicen los parámetros de acuerdo con la técnica seleccionada.

Parameters **event** (*String*) – Contiene el valor del elemento que ejecutó esta función.

get_information()
Recolecta la información que ha seleccionado e introducido el usuario, también la organiza para que se pueda utilizar apropiadamente.

Returns

Un diccionario que contiene:

Clase,
Técnica,
Parametros

Return type Dictionary

grid_widgets()
Permite la colocación adecuada de elementos estáticos y dinámicos, considerando además el espacio o características necesarias de redimensionamiento para éstos últimos.

restore_settings()
Asigna los valores por defecto tanto de las técnicas como de sus respectivos parámetros, también limpia aquéllos en donde se hayan insertado valores.

GeneticOperator (módulo)

Proporciona los elementos gráficos para que el usuario pueda realizar operaciones relacionadas con la Selección, Cruza y Mutación de individuos de una población.

SelectionFrame (clase)

class SelectionFrame(*parent*, *name*, *features*)
Bases: *View.Main.GeneticOperator.TemplateGeneticOperator.TemplateGeneticOperatorFrame.Te*

Esta clase proporciona la infraestructura gráfica para que el usuario pueda elegir métodos y características relacionadas con la selección de individuos.
También hereda atributos de la clase TemplateGeneticOperatorFrame para facilitar la carga de elementos en el Frame y su correspondiente recolección de información.

Parameters

- **parent** (*Tkinter.Frame*) – Frame padre al que pertenece.
- **name** (*String*) – Identificador **(único)** que tendrá el Frame.
- **features** (*Dictionary*) – Conjunto de técnicas con sus respectivos parámetros para que se puedan cargar automáticamente en este Frame **(véase Controller/XMLParser.py)**.

Returns Tkinter.Frame

Return type Instance

get_information()
Recolecta la información relativa a esta clase haciendo uso del método de la clase Padre.

Returns Diccionario con información de los métodos genéricos.

Return type Dictionary

restore_settings()
Ejecuta el método de la clase Padre, el cual restaura los valores por defecto de los elementos dinámicos y estáticos del Frame.

CrossoverFrame (clase)

class **CrossoverFrame**(*parent*, *name*, *features*)
Bases: *View.Main.GeneticOperator.TemplateGeneticOperator.TemplateGeneticOperatorFrame.Ter*

Esta clase proporciona la infraestructura gráfica para que el usuario pueda elegir técnicas y características concernientes a la cruza entre individuos.
También hereda atributos de la clase TemplateGeneticOperatorFrame para facilitar la carga de elementos en el Frame y su correspondiente recolección de información.

Parameters

- **parent** (*Tkinter.Frame*) – Frame padre al que pertenece.
- **name** (*String*) – Identificador **(único)** que tendrá el Frame.
- **features** (*Dictionary*) – Conjunto de técnicas con sus respectivos parámetros para que se puedan cargar automáticamente en este Frame **(véase Controller/XMLParser.py)**.

Returns Tkinter.Frame

Return type Instance

get_information()
Recolecta la información genérica **(usando el método de la clase Padre)**, y también se le añade aquélla recolectada exclusivamente en esta clase.

Returns

Un diccionario que contiene:
Métodos genéricos
Probabilidad de cruza

Return type Dictionary

restore_settings()
Ejecuta el método de la clase Padre, el cual restaura los valores por defecto de los elementos dinámicos y estáticos del Frame.

MutationFrame (clase)

class **MutationFrame**(*parent*, *name*, *features*)
Bases: *View.Main.GeneticOperator.TemplateGeneticOperator.TemplateGeneticOperatorFrame.Ter*

Esta clase proporciona la infraestructura gráfica para que el usuario pueda elegir técnicas y características relativas a la mutación de individuos.
También hereda atributos de la clase TemplateGeneticOperatorFrame para facilitar la carga automática de elementos en el Frame y su consecuente recolección de información.

Parameters

- **parent** (*Tkinter.Frame*) – Frame padre al que pertenece.
- **name** (*String*) – Identificador **(único)** que tendrá el Frame.
- **features** (*Dictionary*) – Conjunto de técnicas con sus respectivos parámetros para que se puedan cargar automáticamente en este Frame **(véase Controller/XMLParser.py)**.

Returns Tkinter.Frame

Return type Instance

get_information()
Recolecta la información genérica **(usando el método de la clase Padre)**, y también se le añade aquélla recolectada exclusivamente en esta clase.

Returns

Un diccionario que contiene:
Métodos genéricos
Probabilidad de mutación

Return type Dictionary

restore_settings()
Ejecuta el método de la clase Padre, el cual restaura los valores por defecto de los elementos dinámicos y estáticos del Frame.

TemplateGeneticOperatorFrame (clase)

class TemplateGeneticOperatorFrame(*parent*, *name*, *features*)
Bases: Tkinter.Frame

Esta clase proporciona la infraestructura gráfica para que el usuario pueda elegir técnicas y configurar atributos concernientes a la Selección, Cruza y Mutación de individuos de una población.
A grandes rasgos se trata de una plantilla que deberán implementar las clases SelectionFrame, CrossoverFrame y MutationFrame.
La clase permite la selección de cada posible técnica disponible y automáticamente se muestran los parámetros necesarios **(si los hay)** para cada una de éstas.

Parameters

- **parent** (*Tkinter.Frame*) – Frame padre al que pertenece.
- **name** (*String*) – Identificador **(único)** que tendrá el Frame.
- **features** (*Dictionary*) – Conjunto de técnicas con sus respectivos parámetros para que se puedan cargar automáticamente en este Frame **(véase Controller/XMLParser.py)**.

Returns Tkinter.Frame

Return type Instance

_TemplateGeneticOperatorFrame__create_dynamic_widgets()

Note: Este método es privado.

Inicializa los elementos dinámicos del Frame, esto es, de acuerdo al tipo que lleva cada parámetro se creará un widget diferente.

_TemplateGeneticOperatorFrame__update_widgets(*event=None*)

Note: Este método es privado.

Realiza solamente la actualización y colocación de elementos dinámicos en el Frame.
Si el parámetro event es distinto de **None**, significa que se lanzó un evento que provocará que se actualicen los parámetros de acuerdo con la técnica seleccionada.

Parameters **event** (*String*) – Contiene el valor del elemento que ejecutó esta función.

get_information()
Recolecta la información que ha seleccionado e introducido el usuario, también la organiza para que se pueda utilizar apropiadamente.

Returns

Un diccionario que contiene:
Clase,
Técnica,
Parámetros

Return type Dictionary

grid_widgets()
Permite la colocación adecuada de elementos estáticos y dinámicos, considerando además el espacio o características necesarias de redimensionamiento para éstos últimos.

restore_settings()
Asigna los valores por defecto tanto de las técnicas como de sus respectivos parámetros, también limpia aquéllos en donde se hayan insertado valores.

MOEA (módulo)

Proporciona los elementos gráficos para que el usuario realice configuraciones concernientes a los M.O.E.A.s (Multi Objective Evolutionary Algorithms) y sus atributos relacionados.

MOEAFrame (clase)

class **MOEAFrame** (*parent*, *features*)
Bases: Tkinter.Frame

Esta clase une los Frames AlgorithmFrame y SharingFunctionFrame, la razón de ésto es para facilitar el acomodo de componentes de manera individual, para así garantizar un acceso asequible a la información.

Parameters

- **parent** (*Tkinter.Frame*) – Frame padre al que pertenece.
- **features** (*Dictionary*) – Conjunto de técnicas con sus respectivos parámetros para que se puedan cargar automáticamente en este Frame **(véase Controller/XMLParser.py)**.

Returns Tkinter.Frame

Return type Instance

get_information ()
Toma la información solicitada en cada Frame y después la unifica para regresar un sólo conjunto de información.

Returns Un diccionario con la información de AlgorithmFrame y SharingFunctionFrame.

Return type Dictionary

restore_settings ()
Restaura los valores por defecto en cada Frame.

AlgorithmFrame (clase)

class **AlgorithmFrame** (*parent*, *name*, *features*)
Bases: Tkinter.Frame

Esta clase proporciona una base gráfica para que el usuario pueda seleccionar técnicas con sus parámetros correspondientes (si es que tienen) referentes a los MOEAs **(Multi Objective Evolutionary Algorithms ó Algoritmo Evolutivo Multi Objetivo)**.

Parameters

- **parent** (*Tkinter.Frame*) – Frame padre al que pertenece.
- **name** (*String*) – Identificador **(único)** que tendrá el Frame.
- **features** (*Dictionary*) – Conjunto de técnicas con sus respectivos parámetros para que se puedan cargar automáticamente en este Frame **(véase Controller/XMLParser.py)**.

Returns Tkinter.Frame

Return type Instance

_AlgorithmFrame__create_dynamic_widgets ()

Note: Este método es privado.

Inicializa los elementos dinámicos del Frame, esto es, de acuerdo al tipo que lleva cada parámetro se creará un widget diferente.

_AlgorithmFrame__grid_widgets()

Note: Este método es privado.

Coloca elementos en el Frame, tanto estáticos como dinámicos.

_AlgorithmFrame__update_widgets(*event=None*)

Note: Este método es privado.

Realiza solamente la actualización y colocación de elementos dinámicos en el Frame.
Si el parámetro event es distinto de **None**, significa que se lanzó un evento que provocará que se actualicen los parámetros de acuerdo con la técnica seleccionada.

Parameters **event** (*String*) – Contiene el valor del elemento que ejecutó esta función.

get_information()
Recolecta la información que ha seleccionado e introducido el usuario, también la organiza para que se pueda utilizar apropiadamente.

Returns

Un diccionario que contiene:
Clase,
Técnica,
Parámetros

Return type Dictionary

restore_settings()
Asigna los valores por defecto tanto de las técnicas como de sus respectivos parámetros, también limpia aquéllos en donde se hayan insertado valores.

SharingFunctionFrame (clase)

class SharingFunctionFrame(*parent*, *name*, *features*)
Bases: Tkinter.Frame

Esta clase proporciona una base gráfica para que el usuario pueda seleccionar métosos con sus respectivos parámetros **(si es que tienen)** referentes a los métodos de Sharing Function.
Un método de Sharing Function sirve para aplicar una selección más intensiva de Individuos en caso de haber un "empate" entre éstos.

Parameters

- **parent** (*Tkinter.Frame*) – Frame padre al que pertenece.
- **name** (*String*) – Identificador **(único)** que tendrá el Frame.
- **features** (*Dictionary*) – Conjunto de técnicas con sus respectivos parámetros para que se puedan cargar automáticamente en este Frame **(véase Controller/XMLParser.py)**.

Returns Tkinter.Frame

Return type Instance

_SharingFunctionFrame__create_dynamic_widgets()

Note: Este método es privado.

Inicializa los elementos dinámicos del Frame, esto es, de acuerdo al tipo que lleva cada parámetro se creará un widget diferente.

_SharingFunctionFrame__grid_widgets()

Note: Este método es privado.

Coloca elementos en el Frame, tanto estáticos como dinámicos.

_SharingFunctionFrame__update_widgets(*event=None*)

Note: Este método es privado.

Realiza solamente la actualización y colocación de elementos dinámicos en el Frame.
Si el parámetro event es distinto de **None**, significa que se lanzó un evento que provocará que se actualicen los parámetros de acuerdo con la técnica seleccionada.

Parameters **event** (*String*) – Contiene el valor del elemento que ejecutó esta función.

get_information()
Recolecta la información que ha seleccionado e introducido el usuario, también la organiza para que se pueda utilizar apropiadamente.

Returns

Un diccionario que contiene:
Clase,
Técnica,
Parámetros

Return type Dictionary

restore_settings()
Asigna los valores por defecto tanto de las técnicas como de sus respectivos parámetros, también limpia aquéllos en donde se hayan insertado valores.

2.3.3 Additional (módulo)

Proporciona elementos gráficos que, aunque no tienen cabilda en la ventana principal, sí contienen herramientas auxiliares de importancia.

GenerationSignalToplevel (clase)

class GenerationSignalToplevel(*parent*, *execution_task_number*)
Bases: Tkinter.Toplevel

Se trata de un Toplevel **(ventana independiente)** que muestra el progreso de las generaciones al momento de ejecutar un Task.
Esta ventana aunque es creada y mostrada en los procesos de la capa View, será en Model/MOEA en donde se utilice y actualice, ya que la idea es crear una "señal" que indique al usuario el progreso del MOEA en ejecución para que se dé una idea del desempeño del algoritmo.

Parameters

- **parent** (*Tkinter.Frame*) – Frame padre al que pertenece.
- **execution_task_number** (*Integer*) – Número que indica el actual Task en ejecución **(véase View/Additional/ResultsGrapher/ResultsGrapherToplevel.py)**.

Returns Tkinter.Toplevel

Return type Instance

_GenerationSignalToplevel__center()

Note: Este método es privado.

Centra la ventana independiente con respecto de la ventana principal. En otras palabras, la ventana independiente será colocada en el centro de la ventana principal.

_GenerationSignalToplevel__do_nothing()

Note: Este método es privado.

Simplemente es una función "dummy" que no realiza nada. Es utilizada como sustituto de la función del ícono "Cerrar" y así evitar que el usario intencionadamente intente ocluir la ventana del número de generaciones.

close()
Oculta y elimina toda referencia gráfica y lógica de la ventana independiente, indicando así que el número de generaciones ha alcanzado su límite.

hide()
Oculta la ventana independiente de la pantalla pero no la elimina de los registros gráficos.

show()
Reactiva la ventana independiente, realizando además durante esta ejecución un par de consignas más para dar una experiencia de usuario suficiente y concisa.

update_current_generation(*current_generation*)

Actualiza la generación actual en la ventana independiente.
Típicamente esta función será usada en todos los algoritmos de la capa Model/MOEA, pues es allí donde se designará el progreso del algoritmo que a su vez se verá reflejado en la capa de View.

Parameters **current_generation** (*Integer*) – La generación que está corriendo actualmente en el MOEA.s

Returns 1 si se ha alcanzado la generación límite, 0 en otro caso.

Return type Integer

update_number_of_generations(*number_of_generations*)
Actualiza el número total de generaciones. Generalmente esta función será llamada desde Controller/Controller.py ya que ahí es donde se decide si las configuraciones iniciales son adecuadas para poder ejecutar el algoritmo.

Parameters **number_of_generations** (*Integer.*) – El número de generaciones total que tendrá el MOEA.

MenuInternalOption (módulo)

Contiene elementos gráficos que permiten acceder a configuraciones internas del programa y también a M.O.P.s (Multi Objective Problems) previamente cargados para hacer uso fácil de ellos.

class MenuInternalOption(*parent*, *features*)
Bases: Tkinter.Menu

Se crea el Menú de Opciones Internas o Menú Secundario.
Básicamente se trata de una serie de características que, aunque no forman parte esencial del programa, sí ofrecen alternativas que pueden facilitar la experiencia de usuario.
Este menú será atado al Frame Principal y desde allí el usuario podrá tener acceso a las opciones que aquí se describen.

Parameters

- **parent** (*Tkinter.Frame*) – El Frame Padre al que pertenece esta implementación.
- **features** (*Dictionary*) – Un diccionario con las características que deberá tener cada una de las opciones listadas.

Returns Tkinter.Menu

Return type Instance

_MenuInternalOption__launch_about_toplevel()

Note: Este método es privado.

Abre la ventana independiente **(Toplevel)** de tipo About. También verifica que se abra una y sólo una instancia de dicha ventana.

_MenuInternalOption__launch_internal_option_toplevel()

Note: Este método es privado.

Abre la ventana independiente **(Toplevel)** de tipo Internal Options **(o simplemente Options)**. También verifica que se abra una y sólo una instancia de dicha ventana.

about_toplevel_custom_close()
Indica que la única instancia que debe crearse para la opción About está disponible.

internal_option_toplevel_custom_close()
Indica que la única instancia que debe crearse para la opción Options está disponible.

InternalOptionToplevel (clase)

class InternalOptionToplevel(*parent*, *features*, *custom_function*)
Bases: Tkinter.Toplevel

Contiene un Menú pequeño con pestañas que indican las características internas del sistema a las que puede tener acceso el usuario.
En su mayoría se trata de características que muestran los métodos, técnicas y sistemas auxiliares que garantizan un manejo más armonioso del programa y si así lo desea el usuario, modificarlos para ajustar su desempeño.

Parameters

- **parent** (*Tkinter.Menu*) – El elemento Padre al que pertenece la actual ventana independiente **(Toplevel)**.
- **features** (*Dictionary*) – Un diccionario que contiene las características necesarias que serán mostradas en esta ventana independiente.
- **custom_function** (*Instance*) – Una variable que contiene una función, la cual redefinirá más apropiadamente el comportamiento de la actual ventana principal con respecto de su Frame Padre.

Returns La ventana independiente que contiene la información señalada.

Return type Tkinter.Toplevel

_InternalOptionToplevel__center()

Note: Este método es privado.

Centra la ventana independiente con respecto de la ventana principal. En otras palabras, la ventana independiente será colocada en el centro de la ventana principal.

close()

Note: Este método es privado.

Cierra y elimina todo rastro de esta ventana independiente.

InternalOptionTab (módulo)

Contiene las partes gráficas que conformarán cada una de las pestañas concernientes al Toplevel **(ventana independiente)** de opciones internas **(InternalOptionToplevel)**.

MOPExampleFrame (clase)

class MOPExampleFrame(*parent*, *features*)

Bases: Tkinter.Frame

Muestra la información relativa a los M.O.P.'s y provee de métodos que facilitan la carga de éstos en la Ventana Principal. Un M.O.P. **(Multi Objective Problem)** es un conjunto de funciones y variables bien definidas que ya han sido previamente estudiadas, así como su comportamiento en conjunto; la idea es proporcionarle al usuario un ambiente de carga fácil de datos para que pueda probar los ejemplos ya tratados por muchos autores en los libros que se citarán en el trabajo escrito.

Parameters

- **parent** (*Tkinter.Toplevel*) – El elemento Padre al que pertenece el actual Frame.
- **features** (*Dictionary*) – Un diccionario que contiene las características necesarias que serán mostradas en este Frame.

Returns El Frame que contiene la información señalada.

Return type Tkinter.Frame

_MOPExampleFrame__get_mop_example(*event*)

Note: Este método es privado.

Con base en la selección de M.O.P. hecha por el usuario, se carga éste en la Ventana Principal.

Parameters **event** (*String*) – El evento del elemento gráfico que activa esta función.

_MOPExampleFrame__update_current_mop(*event=None*)

Note: Este método es privado.

Despliega la información relacionada con el M.O.P. seleccionado.

Parameters **event** (*String*) – El evento del elemento gráfico que activa esta función.

FeatureFrame (clase)

class **FeatureFrame**(*parent*)
Bases: Tkinter.Frame

ExpressionFrame (clase)

class **ExpressionFrame**(*parent*, *features*)
Bases: Tkinter.Frame

Al momento de crear y evaluar funciones objetivo hay algunas palabras reservadas que no pueden ser usadas en Python directamente si no se hace un renombramiento apropiado.
En este Frame se ofrecen opciones simples para mostrar y añadir expresiones de Python.
Dicha información se encuentra en
Controller/XML/PythonExpressions.xml

Parameters

- **parent** (*Tkinter.Toplevel*) – El elemento Padre al que pertenece el actual Frame.
- **features** (*Dictionary*) – Un diccionario que contiene las características necesarias que serán mostradas en este Frame.

Returns El Frame que contiene la información señalada.

Return type Tkinter.Frame

_ExpressionFrame__add_expression(*event*)

Note: Este método es privado.

Inserta una casilla que conforma una expresión dentro del Frame.

Parameters **event** (*String*) – Identificador del elemento gráfico que activó la función.

_ExpressionFrame__delete_single_expression(*event*)

Note: Este método es privado.

Elimina una expresión y todos los elementos gráficos que la acompañan. También elimina todo rastro que se encuentre en las estructuras lógicas.

Parameters **event** (*String*) – Identificador del elemento gráfico que activó la función.

_ExpressionFrame__get_information()

Note: Este método es privado.

Toma la información del Frame **(en específico de las casillas)** y regresa las expresiones con sus respectivos equivalentes en Python.

Returns Una lista que contiene arreglos de dos elementos donde el primero es la expresión normal mientras que el segundo es la expresión equivalente en Python.

Return type List

_ExpressionFrame__insert_expression(*expression=None*)

Note: Este método es privado.

Coloca en el Frame una colección de elementos:

[etiqueta para expresión normal, expresión normal, etiqueta para expresión de Pyrhon, expresión de Python, botón para eliminar]

Si el parámetro expression es **None**, se añade la casilla vacía, de lo contrario se agrega ésta con la información pertinente.

Parameters **expression** (*Array*) – Un arreglo con dos elementos, el primero contiene la expresión normal mientras que el segundo maneja la información de la expresión equivalente en Python.

_ExpressionFrame__load_expressions()

Note: Este método es privado.

Carga las expresiones a manera de contenido gráfico en el Frame.

Dichas expresiones son tomadas del archivo Controller/XML/PythonExpressions.xml.

_ExpressionFrame__save_changes(*event*)

Note: Este método es privado.

Toma la información existente en las casillas y procede a sobreescribir el archivo Controller/XML/PythonExpressions.xml con la información recién recabada.

Parameters **event** (*String*) – Identificador del elemento gráfico que activó la función.

get_current_elements()

Regresa el número actual de casillas en el Frame.

Returns Cantidad de elementos en la estructura rows, donde se guardan las casillas (Entry's).

Return type Int

PythonExpressionFrame (clase)

class **PythonExpressionFrame**(*parent*, *features*)
Bases: Tkinter.Frame

Realiza la fusión de un Canvas y ExpressionFrame, debido a que, cuando se agregan numerosas variables al ExpressionFrame, se debe insertar una barra de desplazamiento para poder acceder a aquéllos que se encuentren hasta abajo. Dentro del ambiente de Tkinter, el elemento más sencillo para lograr este efecto es un Canvas, por ello se anida el ExpressionFrame al Canvas.

Parameters

- **parent** (*Tkinter.Frame*) – Frame padre al que pertenece.
- **features** (*Dictionary*) – Conjunto de técnicas con sus respectivos parámetros para que se puedan cargar automáticamente en este Frame **(véase Controller/XML/PythonExpressions.xml)**.

Returns Tkinter.Frame

Return type Instance

_PythonExpressionFrame__activate_scroll(*event=None*)

Note: Este método es privado.

Actualiza la barra de desplazamiento y con base en esta acción la activa o desactiva.

Parameters **event** (*String*) – Elemento que ejecutó esta función.

_PythonExpressionFrame__update_scrollbar(*event=None*)

Note: Este método es privado.

Actualiza la barra de desplazamiento de acuerdo al número de elementos existentes en el Frame, esto para poder hacer un recorrido apropiado de la barra.

Parameters **event** (*String*) – Elemento que ejecutó esta función.

AboutToplevel (clase)

class **AboutToplevel**(*parent*, *custom_function*)
Bases: Tkinter.Toplevel

Esta ventana independiente **(Toplevel)** proporciona información básica del programa así como de sus desarrolladores.

Parameters

- **parent** (*Tkinter.Menu*) – El elemento Padre al que pertenece la actual ventana independiente **(Toplevel)**.

- **custom_function** (*Instance*) – Una variable que contiene una función, la cual redefinirá más apropiadamente el comportamiento de la actual ventana principal con respecto de su Frame Padre.

Returns Tkinter.Toplevel

Return type Instance

_AboutToplevel__center()

Note: Este método es privado.

Centra la ventana independiente con respecto de la ventana principal. En otras palabras, la ventana independiente será colocada en el centro de la ventana principal.

_AboutToplevel__close(*custom_function*)

Note: Este método es privado.

Cierra y elimina todo rastro de esta ventana independiente.

Parameters **custom_function** (*Instance*) – Una variable que contiene una función que ha de ejecutarse dentro de este método.

ResultsGrapher (módulo)

Proporcional los elementos gráficos para poder presentar las gráficas de los resultados que ha arrojado la ejecución de algún M.O.E.A.

ResultsGrapherToplevel (clase)

class ResultsGrapherToplevel(*parent*, *execution_task_count*, *main_features*, *gathered_information*, *final_results*)

Bases: Tkinter.Toplevel

Esta clase lanza una ventana independiente que muestra los resultados arrojados por una configuración previa del usuario.

Primero que nada es menester mencionar que una ventana independiente es un Toplevel en Tkinter, la cual es casi ajena a la ventana principal **(véase View/Main/MainWindow.py)**, pero si ésta última es cerrada, se eliminarán también las ventanas independientes creadas.

Cada ventana independiente mostrará el número de Task, es decir, el orden en el que fue procesada la información con respecto de otros Tasks.

Entiéndase por Task a una ejecución de algún algoritmo MOEA bajo un cierto conjunto de configuraciones iniciales. Así, los Tasks serán mostrados en una ventana independiente. La numeración de los Tasks irá siempre en orden progresivo, lo que significa que el número será reinicializado sólamente volviendo a ejecutar el programa principal.

De esta manera es posible tener varias ventanas independientes abiertas y en cuestiones más generales, es posible ejecutar varios Tasks simultáneamente, ya que el programa es multi-threading en ese sentido.

Finalmente, la información será mostrada en dos pestañas: en una **(SummaryFrame)** se otorga un resumen de todas las funciones objetivo, variables de decisión, MOEA usado y configuraciones adicionales en el Task.

En la otra **(GraphFrame)** se colocan todas las gráficas pertinentes producto de la ejecución del MOEA con las funciones objectivo, variables de decisión y configuraciones ingresadas **(véase Model/Community/Community.py) (véase View/Additional/ResultsGrapher/GraphFrame.py)**.

Si por cualquier circunstancia llega a haber una falla interna durante la ejecución del proceso, ninguna de las dos pestañas será mostrada y en su lugar aparecerá una de error **(ErrorFrame)**, especificando además el tipo de error y en qué parte de Model **(ó Modelo)** ocurrió.

Parameters

- **parent** (*Tkinter.Frame*) – Frame padre al que pertenece.
- **execution_task_count** (*Integer*) – Número que indica el actual Task en ejecución.
- **main_features** (*Dictionary*) – Diccionario que contiene, entre otras cosas, los nombres de los parámetros asociados a cada técnica.
- **gathered_information** (*Dictionary*) – Diccionario que contiene todas las configuraciones recabadas ingresadas por el usuario **(véase View/Main/MainWindow.py)**.
- **final_results** (*Dictionary*) – Diccionario que contiene la información procesada lista para graficar **(véase View/Additional/ResultsGrapher/GraphFrame.py)**.

Returns Tkinter.Toplevel

Return type Instance

_ResultsGrapherToplevel__center()

Note: Este método es privado.

Centra la ventana independiente con respecto de la ventana principal. En otras palabras, la ventana independiente será colocada en el centro de la ventana principal.

_ResultsGrapherToplevel__create_renamed_settings()

Note: Este método es privado.

Tal como su nombre lo dice, renombra las funciones objetivo y variables de decisión para posteriormente almacenarlas en una estructura por cada tipo.

Renombrar una función o variable de decisión es hacer un mapeo que consista en:

Elemento_renombrado -> elemento original.

Para el caso de la función objetivo, el renombramiento se da anteponiendo la letra F seguido de la posición en la que fue insertada originalmente por el usuario.

El caso es análogo para la variable de decisión, sólo que la letra es V.

La idea de renombrar las funciones y variables surge como alternativa al momento de graficar los datos **(véase View/Additional/ResultsGrapher/GraphFrame.py)**, ya que el usuario puede ingresar funciones muy largas o variables con identificadores muy complejos y esto en la parte gráfica se vería muy amontonado; por ello fue preferible mostrar la parte renombrada en la sección de GraphFrame y colocar la muestra original en el SummaryFrame.

GraphFrame (clase)

class **GraphFrame**(*parent*, *execution_task_count*, *objective_functions*, *decision_variables*, *final_results*)
Bases: `Tkinter.Frame`

Proporciona un Frame que contiene gráficas alimentadas por los resultados obtenidos al ejecutar algún MOEA, el cual ha sido refinado por las configuraciones recabadas de la ventana principal **(véase Model/MOEA)**.

Parameters

- **parent** (*Tkinter.Frame*) – Frame padre al que pertenece.
- **execution_task_count** (*Integer*) – Número que indica el actual Task en ejecución.
- **objective_functions** (*List*) – Lista que contiene las funciones objetivo renombradas.
- **decision_variables** (*List*) – Lista que contiene las variables de decisión renombradas.
- **final_results** (*Dictionary*) – Diccionario que contiene la información para graficar. Se divide en dos categorías principales: Frente de Pareto y Mejor Individuo por Generación.

Returns Tkinter.Frame

Return type Instance

_GraphFrame__change_canvas_category()

Note: Este método es privado.

Realiza el cambio de Canvas de la categoría de funciones objetivo a la de variables y decision y viceversa, tomando en cuenta factores como por ejemplo si alguna de las dos categorías tiene un OptionMenu asociado (para entonces colocarlo apropiadamente) e identificando siempre el último Canvas seleccionado de la categoría anterior para que cuando sea oportuno se vuelva a colocar.

_GraphFrame__change_inner_canvas(*event*)

Note: Este método es privado.

Realiza el cambio de Canvas dentro de una misma categoría, esto en caso en que los datos hayan arrojado más de una gráfica. El cambio se hace con ayuda de su OptionMenu asociado.

Parameters **event** (*String*) – Elemento que ejecutó esta función.

_GraphFrame__create_2d_canvas(*x_label*, *x_index*, *y_label*, *y_index*, *collection_points*)

Note: Este método es privado.

Crea una gráfica en 2 dimensiones que es envuelta en un Canvas.

Parameters

- **x_label** (*String*) – Nombre para el eje X de la gráfica.

- **x_index** (*Integer*) – Posición dentro de collection_points para los datos del eje X.
- **y_label** (*String*) – Nombre para el eje Y de la gráfica.
- **y_index** (*Integer*) – Posición dentro de collection_points para los datos del eje Y.
- **collection_points** (*Dictionary*) – Diccionario que contiene los puntos a graficar.

Returns Canvas

Return type Matplotlib.FigureCanvasTkAgg

_GraphFrame__create_3d_canvas(*x_label*, *x_index*, *y_label*, *y_index*, *z_label*, *z_index*, *collection_points*)

Note: Este método es privado.

Crea una gráfica en 3 dimensiones que es envuelta en un Canvas.

Parameters

- **x_label** (*String*) – Nombre para el eje X de la gráfica.
- **x_index** (*Integer*) – Posición dentro de collection_points para los datos del eje X.
- **y_label** (*String*) – Nombre para el eje Y de la gráfica.
- **y_index** (*Integer*) – Posición dentro de collection_points para los datos del eje Y.
- **z_label** (*String*) – Nombre para el eje Z de la gráfica.
- **z_index** (*Integer*) – Posición dentro de collection_points para los datos del eje Z.
- **collection_points** (*Dictionary*) – Diccionario que contiene los puntos a graficar.

Returns Canvas

Return type Matplotlib.FigureCanvasTkAgg

_GraphFrame__create_decision_variables_canvas(*decision_variables*, *collection_points*)

Note: Este método es privado

Crea los Canvas para las variables de decisión.

Parameters

- **decision_variables** (*List*) – Lista que contiene las variables de decisión renombradas.
- **collection_points** (*Dictionary*) – Diccionario que contiene los valores de las funciones objetivo de todos los individuos en la población final.

_GraphFrame__create_objective_functions_canvas(*objective_functions*, *collection_points*)

Note: Este método es privado.

Crea los Canvas para las funciones objetivo.

Parameters

- **objective_functions** (*List*) – Lista que contiene las funciones objetivo renombradas.
- **collection_points** (*Dictionary*) – Diccionario que contiene los valores de las funciones objetivo de todos los individuos en la población final.

_GraphFrame__custom_save_figure(*current_canvas*)

Note: Este método es privado.

Arroja una ventana emergente modificada para guardar archivos, en este caso las gráficas.
Las modificaciones con respecto de la original consisten en agregar un título para tener conocimiento de las imágenes del Task que se van a guardar.
Además se modifica el comportamiento de la ventana para adherirlo a la ventana del Task y no a la ventana principal.

Parameters **current_canvas** (*Matplotlib.FigureCanvasTkAgg*) – El Canvas asociado a la barra de navegación sobre la cual actuará la función.

ErrorFrame (clase)

class ErrorFrame(*parent*, *final_results*)
Bases: Tkinter.Frame

Este Frame surge si durante el proceso interno en el Modelo **(véase Model/MOEA)** se suscita algún error del cual el método no se pueda recuperar.
Entonces aquí se desplegará toda la información relativa a la falla, asímismo funciona como medida de contingencia para darle una salida al método y evitar que se quede atorado.

Parameters

- **parent** (*Tkinter.Frame*) – Frame padre al que pertenece.
- **final_results** (*Dictionary*) – Diccionario que contiene en este caso las características alusivas a la falla **(véase Model/MOEA)**.

Returns Tkinter.Frame

Return type Instance

SummaryFrame (clase)

class SummaryFrame(*parent*, *renamed_objective_functions*, *renamed_decision_variables*, *main_features*, *gathered_information*)
Bases: Tkinter.Frame

Unifica dos elementos: Canvas y ContentFrame. La razón de esto es que, en promedio la información mostrada por ContentFrame rebasará el tamaño de la ventana de la información final **(véase View/Additional/ResultsGrapher/ResultsGrapherTopLevel.py)**, es entonces que se deben agregar barras de desplazamiento para poder acceder al contenido que quedaría oculto.

Uno de los elementos en Tkinter más sencillos que cumplen con este cometido es un Canvas. Luego entonces esa es la razón de tal fusión.

Parameters

- **parent** (*Tkinter.Frame*) – Frame padre al que pertenece.
- **renamed_objective_functions** – Diccionario de funciones objetivo renombradas **(véase View/Additional/ResultsGrapher/ResultsGrapherToplevel.py)**.
- **renamed_decision_variables** (*Dictionary*) – Diccionario de variables de decisión renombradas **(véase View/Additional/ResultsGrapher/ResultsGrapherToplevel.py)**.
- **main_features** (*Dictionary*) – Diccionario que contiene, entre otras cosas, los nombres de los parámetros asociados a cada técnica.
- **gathered_information** (*Dictionary*) – Diccionario que contiene todas las configuraciones recabadas ingresadas por el usuario **(véase View/Main/MainWindow.py)**.

Returns Tkinter.Frame

Return type Instance

_SummaryFrame__update_scrollbar(*event*)

Note: Este método es privado.

Actualiza la barra de desplazamiento de acuerdo al número de elementos existentes en el Frame, esto para poder hacer un recorrido apropiado de la barra.

Parameters **event** (*String*) – Elemento que ejecutó esta función.

ContentFrame (clase)

class ContentFrame(*parent*, *renamed_objective_functions*, *renamed_decision_variables*, *main_features*, *gathered_information*)

Bases: Tkinter.Frame

Recaba el contenido de todas las funciones objetivo, variables de decisión y demás parámetros que el usuario ingresó para poder ejecutar un Task determinado. Es entonces que plasma toda esta información en un Frame para que el usuario pueda cotejar los datos ingresados con los resultados obtenidos **(véase View/Additional/ResultsGrapher/GraphFrame.py)**.

Parameters

- **parent** (*Tkinter.Frame*) – Frame padre al que pertenece.
- **renamed_objective_functions** – Diccionario de funciones objetivo renombradas **(véase View/Additional/ResultsGrapher/ResultsGrapherToplevel.py)**.

- **renamed_decision_variables** (*Dictionary*) – Diccionario de variables de decisión renombradas **(véase View/Additional/ResultsGrapher/ResultsGrapherToplevel.py)**.
- **main_features** (*Dictionary*) – Diccionario que contiene, entre otras cosas, los nombres de los parámetros asociados a cada técnica.
- **gathered_information** (*Dictionary*) – Diccionario que contiene todas las configuraciones recabadas ingresadas por el usuario **(véase View/Main/MainWindow.py)**.

Returns Tkinter.Frame

Return type Instance

2.4 Controller (módulo)

class Controller

execute_procedure(*execution_task_count*, *generations_queue*, *sanitized_information*)

Realiza la ejecución de algún algoritmo M.O.E.A. **(Multi-Objective Evolutionary Algorithm)** y se encarga de obtener los resultados apropiadamente.

Parameters

- **execution_task_count** (*Integer*) – Una característica numérica que identifica inequívocamente a esta función que será ejecutada de las demás, ya que el objetivo del proyecto es poder ejecutar varios de estos métodos de manera concurrente **(véase View/Additional/ResultsGrapher/ResultsGrapherToplevel.py)**.
- **generations_queue** (*Instance*) – Una instancia a una cola **(Queue)**, la cual servirá para escribir a esa estructura el número actual de generación por el que cursa el algoritmo. Esta acción es para fines de concurrencia **(véase View/MainWindow.py)**.
- **sanitized_information** (*Dictionary*) – Los parámetros que ingresó el usuario debidamente verificados y saneados.

Returns Un diccionario con información de los resultados de haber ejecutado el M.O.E.A. seleccionado por el usuario, la estructura del mismo puede verse en **Model/Community/Community.py**.

Return type Dictionary

load_features()

load_mop_examples()

load_python_expressions()

sanitize_settings(*general_information*, *features*)

Lleva a cabo la verificación y saneamiento de todos los datos que ha ingresado el usuario en la sección View **(véase View/MainWindow)**.

Parameters

- **general_information** (*Dictionary*) – El conjunto de datos que el usuario ha ingresado o seleccionado.
- **features** (*Dictionary*) – Una colección de todos los elementos con sus características disponibles para el usuario.

Returns

Return type Dictionary

save_python_expressions (*features*)

2.4.1 XMLParser (clase)

class XMLParser

Permite leer y escribir a archivos .xml **(los que se localizan en Controller/XML)**, los cuales tienen almacenados:

- Los nombres de las técnicas con sus parámetros que se encuentran disponibles en la sección Model **(Features.xml)**.
- La colección de palabras reservadas para poder emplear funciones y constantes auxiliares en las funciones objetivo **(PythonExpressions.xml)**.
- El conjunto de M.O.P.'s **(Multi-Objective Problems, localizados en MOPExamples.xml)**.

indent (*element*, *level=0*)

Indenta **(coloca espacios)** apropiadamente en un documento .xml para poder distinguir más rápidamente los distintos niveles que existen en éste.

Parameters

- **element** (*String*) – Una línea del archivo .xml
- **level** (*Integer*) – El nivel en el que se está haciendo el proceso de identado.

load_xml_features (*features_filename*)

Lee el archivo que contenga el listado de técnicas y sus parámetros disponibles **(véase Model)** y carga todos los elementos que se encuentran en éste.

Parameters **features_filename** (*String*) – Nombre del archivo en cuestión.

Returns Un diccionario que contiene todos los elementos del archivo.

Return type Dictionary

load_xml_mop_examples (*features_filename*)

Lee el archivo que contenga el listado de M.O.P.'s **(Multi-Objective Problems)** y carga todos los elementos que se encuentran en éste.

Un M.O.P es una mezcla de variables de decisión y funciones objetivo ya estudiadas, se utilizan para reproducir su comportamiento y así garantizar, además de un correcto funcionamiento del programa, una opción rápida para probar las técnicas que se ofrecen.

Parameters **features_filename** (*String*) – Nombre del archivo en cuestión.

Returns Un diccionario que contiene todos los elementos del archivo.

Return type Dictionary

load_xml_python_expressions (*features_filename*)

Lee el archivo que contenga el listado de expresiones en Python y carga todos los elementos que se encuentran en éste.

La idea detrás de esto es que, al momento de crear y/o evaluar funciones objetivo existen algunas palabras reservadas que no pueden ser usadas directamente como son las funciones trigonométricas, por eso es que estas expresiones sirven como intermediarias entre el usuario y el intérprete de Python.

En ocasiones a este tipo de expresiones, no sólo en el ámbito actual sino en general, se les conoce como azúcar sintáctica.

Parameters **features_filename** (*String*) – Nombre del archivo en cuestión.

Returns Un diccionario que contiene todos los elementos del archivo.

Return type Dictionary

write_xml_python_expressions(*features_filename*, *features*)

Sobreescribe el archivo donde se encuentra el listado de expresiones en Python.
El objetivo es que, una vez ejecutándose el programa y a través del menú pertinente **(véase View/Additional/MenuInternalOption/InternalOptionTab/PythonExpressionFrame.py)**, el usuario pueda añadir o eliminar las expresiones de Python que desee.

En ocasiones a este tipo de expresiones, no sólo en el ámbito actual sino en general, se les conoce como azúcar sintáctica.

Parameters

- **features_filename** (*String*) – Nombre del archivo en cuestión.
- **features** (*List*) – La estructura que contiene las expresiones para ser guardadas en el archivo .xml.

2.4.2 Verifier (clase)

class Verifier

Realiza principalmente la verificación y transformación adecuada de los datos que el usuario introduce en View/MainWindow.py para alimentar a los algoritmos que se encuentran en la sección Model.
De manera secundaria también ofrece métodos de verificación para la extracción y colocación de datos en los archivos .xml **(véase XMLParser y el directorio Controller/XML)**.

_Verifier__cast_parameter(*parameter_value*, *parameter_settings*)

Note: Este método es privado.

Verifica un parámetro asociado a alguna técnica. Primero asegura que el parámetro se pueda evaluar correctamente, posteriormente convierte apropiadamente el tipo de dato pasando de String a Boolean, Integer ó Float según corresponda.

Parameters

- **parameter_value** (*Float*) – El valor actual del parámetro.

- **parameter_settings** (*Dictionary*) – Un diccionario que contiene el tiop del parámetro (bool, integer ó float) y el rango que debe tomar tanto inferior como superior.

Returns El valor saneado del parámetro si no hay fallas, pero si se encuentra algún desperfecto entonces se regresa un diccionario con la información detallada del desperfecto.

Return type (Boolean, Integer, Float)/Dictionary

_Verifier__verify_instance(*name_class*)

Note: Este método es privado.

Devuelve una instancia del nombre de la clase que se le pase como parámetro.

Esta funcionalidad es útil sobre todo para la sección Model ya que uno de los objetivos es proporcionar al usuario de una infraestructura rápida con técnicas fácilmente intercambiables sin necesidad de estar importando explícitamente cada una de éstas.

De esta forma con base en una instancia se puede ejecutar cualquier método de manera dinámica.

Parameters **name_class** (*String*) – el nombre de la clase **(con su ruta)** de la cual se desea obtener una instancia.

Returns Una instancia de la clase solicitada si el proceso es exitoso, en otro caso se obtiene un diccionario con los detalles de la falla.

Return type Instance/Dictionary

get_dynamic_function(*complete_function*)

Obtiene una instancia de una función en un String de la forma **biblioteca.función**. Este método se usa para convertir las expresiones de Python en instancias que serán utilizadas al momento de evaluar funciones objetivo **(véase View/Additional/MenuInternalOption/InternalOptionTab/PythonExpressionFrame.py, Controller/XML/PythonExpressions.xml)**.

Parameters **complete_function** (*String*) – un String preferentemente de la forma **biblioteca.función (el punto debe ir incluido)**.

Returns Una instancia de la función asociada a la biblioteca.

Return type Instance

sanitize_decision_variables(*vector_variables*)

Verifica el conjunto de elementos de la categoría "Decision Variables" **(véase View/Main/DecisionVariable/DecisionVariableFrame.py)**, los cuales son precisamente las variables de decisión.

Primero se asegura que cada variable de decisión se pueda evaluar correctamente, posteriormente convierte apropiadamente el tipo de dato de sus respectivos rangos, pasando de String a Float.

Parameters **vector_variables** (*Dictionary*) – El vector que contiene las variables de decisión con sus correspondientes rangos.

Returns Un diccionario con las variables de decisión y sus rangos debidamente saneados.

Return type Dictionary

sanitize_genetic_operators_settings(*genetic_operators_settings*, *features*, *vector_variables*, *number_of_decimals*)

Revisa la integridad y sanea los datos que ingresó el usuario concernientes a la sección "Genetic Operators Settings" **(véase View/Main/GeneticOperator/GeneticOperatorFrame.py)**.

Parameters

- **genetic_operators_settings** (*Dictionary*) – El listado de técnicas y sus parámetros que el usuario eligió en la sección correspondiente.
- **features** (*Dictionary*) – El conjunto de las opciones disponibles para esta sección, así como sus características.
- **vector_variables** (*List*) – El vector de variables de decisión.
- **number_of_decimals** (*Integer*) – El número de decimales que llevará cada solución en Population.

Returns Un diccionario que, dependiendo de los resultados, puede contener o información del error encontrado durante el procedimiento o todos los datos debidamente verificados y transformados.

Return type Dictionary

sanitize_moeas_settings(*moeas_settings*, *features*)

Verifica integridad y lleva a cabo el saneamiento de los datos que ingresó el usuario concernientes a la sección "MOEAs Settings" **(véase View/Main/MOEA/MOEAFrame.py)**.

Parameters

- **moeas_settings** (*Dictionary*) – El listado de técnicas y sus parámetros que el usuario eligió en la sección correspondiente.
- **features** (*Dictionary*) – El conjunto de las opciones disponibles para esta sección, así como sus características.

Returns Un diccionario que, dependiendo de los resultados, puede contener o información del error encontrado durante el procedimiento o todos los datos debidamente verificados y transformados.

Return type Dictionary

sanitize_objective_functions(*vector_variables*, *available_expressions*, *vector_functions*)

Lleva a cabo el saneamiento de los elementos correspondientes a la categoría "Objective Functions" **(véase View/Main/ObjectiveFuncion/ObjectiveFunctionFrame.py)**, los cuales son de hecho sólo las funciones objetivo.

Parameters

- **vector_variables** (*Dictionary*) – El vector de variables de decisión que el usuario ha ingresado.
- **available_expressions** (*Dictionary*) – Un listado con las expresiones de Python disponibles **(véase Controller/XML/PythonExpressions.xml, View/Additional/MenuInternalOption/InternalOptionTab/PythonExpressionFrame.py)**.
- **vector_functions** (*Dictionary*) – El vector de funciones objetivo ingresados por el usuario.

Returns Si el proceso fue exitoso, se obtiene el mismo vector_functions, en otro caso se regresa un diccionario con información detallada sobre el errror encontrado.

Return type List/Dictionary

sanitize_population_settings(*population_settings*, *features*)

Verifica la consistencia y realiza el saneamiento de los datos que ingresó el usuario concernientes a la sección "Population Settings" **(véase View/Main/Population/PopulationFrame.py)**.

Parameters

- **population_settings** (*Dictionary*) – El listado de técnicas y sus parámetros que el usuario eligió en la sección correspondiente.
- **features** (*Dictionary*) – El conjunto de las opciones disponibles para esta sección, así como sus características.

Returns Un diccionario que, dependiendo de los resultados, puede contener o información del error encontrado durante el procedimiento o todos los datos debidamente verificados y transformados.

Return type Dictionary

sanitize_techniques(*general_information*, *features*)

Realiza una verificación adicional concerniente al tipo de representación de todas las técnicas seleccionadas.

Lo anterior significa que, usando la Representación Cromosómica **(ó Chromosomal Representation, véase Model/ChromosomalRepresentation, View/Main/Population/PopulationFrame.py)**, todas las técnicas deben concordar con el mismo tipo de representación cromosómica que se haya seleccionado.

Para esta versión sólo están disponibles las representaciones binaria y de punto flotante.

Parameters

- **general_information** (*Dictionary*) – El listado de características disponibles **(véase XMLParser.py)**.
- **features** (*Dictionary*) – La colección de datos que seleccionó el usuario en la sección View.

Returns Un diccionario el cual, si la verificación es exitosa, es el mismo general_informacion, si por el contrario falla, entonces es un diccionario que contiene detalles del error.

Return type Dictionary

verify_load_mop_examples(*data*)

verify_load_python_expressions(*data*)

verify_load_xml_features(*data*)

verify_write_xml_python_expressions(*data*)

b

c

m

v