ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

MATEMÁTICAS PARA LA COMPUTACIÓN

GRADO EN INGENIERÍA INFORMÁTICA - TECNOLOGÍAS INFORMÁTICAS

OPTIMIZACIÓN MEDIANTE TABÚ SEARCH

Realizado por

Francisco Jesús Belmonte Pintre (frabelpin@alum.us.es)

Martín González López (margonlop19@alum.us.es)

Dirigido por

María Cruz López de los Mozos Martín

Departamento

Matemática Aplicada I

Sevilla, Enero de 2019

Índice general

1 CONTEXTO DEL PROBLEMA

El problema del emplazamiento de nodos router es un problema de optimización combinatoria. Los ingredientes del problema son:

- un plano cartesiano en 2 dimensiones,
- una serie de clientes desplegados en el plano y del cual conocemos su coordenada en el eje “X” y su coordenada en el eje “Y”,
- y una serie de nodos routers de los que disponemos, cada uno de ellos con su radio de cobertura, su coordenada en el eje “X” y su coordenada en el eje “Y”.

Queremos colocar los routers estratégicamente a lo largo del plano de tal manera que se cumplan 2 objetivos:

- que los routers estén interconectados entre sí formando lo que se conoce en matemáticas discretas como un árbol,
- que el número de clientes que se hayan bajo el área de cobertura de al menos un router sea el máximo posible.

2 TÉCNICA DE RESOLUCIÓN Y TECNOLOGÍA UTILIZADA

El algoritmo utilizado en el siguiente trabajo es un algoritmo de búsqueda Tabú, conocido en el mundo anglosajón como Tabú Search. La tecnología utilizada es Python Sage.

Este algoritmo hace uso de una memoria Tabú, esto es una estructura de datos que nos permita almacenar las soluciones para las que ya hemos pasado previamente, o bien características de las soluciones. De lo que se trata es de guiar la búsqueda de manera inteligente, ergo tendremos que evitar de alguna forma que el algoritmo vuelva a pasar por zonas previamente visitadas.

Se combina la prohibición de soluciones con la prohibición de características de soluciones. Esto se debe a que las características prohibidas dejan de serlo pasado un cierto número de iteraciones, eso significa que la búsqueda podría volver a pasar por una solución ya visitada una vez que la característica prohibida dejase de serlo.

El algoritmo también prohíbe soluciones muy parecidas a las soluciones ya encontradas, al prohibir las características de las soluciones por las que ya hemos pasado.

El algoritmo admite cierta flexibilidad a la hora de tratar con soluciones tabú, esto es, dejar pasar una solución considerada tabú al poseer alguna característica tabú si se cumple algún criterio de aspiración.

En el presente trabajo se han considerado dos criterios de aspiración:

- que la solución considerada tabú al poseer una característica tabú, se deja pasar siempre y cuando su valor de fitness mejore con respecto al de la mejor solución encontrada hasta el momento.
- que la solución considerada tabú al poseer una característica tabú, se deja pasar si han pasado un determinado número de iteraciones desde la vez que se declaró como tabú la característica.

Para el siguiente trabajo, las características de las soluciones que se declaran tabú será una determinada posición para un determinado router, esto es, antes de que a un router lo vayamos a cambiar de posición, declaramos como tabú su posición original previa al cambio, de esta manera se evita la posibilidad de que en las próximas iteraciones ese mismo router pueda volver a su posición original lo cual nos acercaría a una posición ya visitada previamente y por tanto no estaríamos consiguiendo nuestro objetivo de realizar una búsqueda inteligente.

A pesar de que dentro de un determinado número de iteraciones se pierde esa condición de tabú, seguiremos manteniendo una segunda memoria para evitar caer en soluciones ya visitadas, indexada mediante el cálculo de un valor hash para cada solución, con una probabilidad de colisionamiento de hashes muy baja.

Los movimientos aplicados para generar nuevas soluciones serán de dos tipos:

- intercambio de routers, esto es seleccionar dos routers e intercambiarlos de posición. Esto da lugar al vecindario de soluciones, esto es hacer el cálculo combinatorio de todos los posibles intercambios que se pueden llevar a cabo entre 2 routers.
- Mover un determinado router a una determinada posición. Ambos datos se elegirán al azar.

Emplearemos 2 variables para almacenar la solución al problema, una de ellas que nunca empeorará su valor de fitness y que será la mejor solución encontrada hasta el momento. La segunda recibirá en cada iteración la solución menos mala que haya encontrado en el vecindario de soluciones en caso de movimiento de intercambio de routers, o bien la nueva solución generada a partir de mover un determinado router a una determinada posición, independientemente de que mejore o empeore en fitness con respecto a la original.

En cuanto a fitness, daremos prioridad a encontrar un emplazamiento que nos permita tener el máximo número de nodos router conectados entre sí. Es por ello que la segunda función de fitness, esto es, el número de clientes cubiertos por al menos un router, se utilizará para desempatar en caso de que encontremos varias soluciones con igual valor de fitness de la primera función.

Tabú Search hace uso de las técnicas de intensificación y de diversificación.

Intensificación, esto es agregear características de las mejores soluciones encontradas hasta el momento a nuestra solución actual. Para ello, necesitaremos una memoria que recoja las mejores soluciones encontradas hasta el momento. Como característica a agregar a nuestra solución actual, optaremos por la posición que ocupan los routers en las mejores soluciones encontradas hasta el momento, y para cada router se elegirá la nueva posición a través de un método de selección por ruleta escogido de entre todas las posiciones que ocupaba el router i-ésima en las mejores soluciones encontradas hasta el momento.

Diversificación, esto es tratar de romper la localidad al pasar un determinado número de iteraciones sin que hayamos conseguido mejorar el fitness con respecto a la mejor solución encontrada hasta el momento. La idea consiste en modificar de manera destacable la solución actual, de forma que nos permite acceder a nuevas zonas del espacio de soluciones desconocidas hasta el momento.

Para nuestro trabajo, se ha optado como método de diversificación a mover de posición aquellos routers de la solución actual que menos veces se han movido de sitio a lo largo de la búsqueda Tabú. Para ello necesitaremos de una memoria que recoja la frecuencia de movimiento de cada routers. Los moveremos a aquellas posiciones en el plano más frecuentadas, y para ello también necesitaremos otra memoria que recorra la frecuencia para cada posición.

3 TIPOS IMPLEMENTADOS

Aprovechando que Python ofrece programación orientada a objetos, hemos diseñado 5 tipos para llevar a cabo el desarrollo del algoritmo:

1. Tipo **“Router”**.
2. Tipo **“Client”**.
3. Tipo **“Movement”**.
4. Tipo **“Solution”**.
5. Tipo **“TSMeshAlgorithmProblem”**.

Hemos implementado un método instanciador, “instantiateTSMeshAlgorithmProblem”, para crear un objeto de tipo “TSMeshAlgorithmProblem”. El propio tipo “TSMeshAlgorithmProblem” implementará el método Tabú Search, esto es, “executeTSMeshAlgorithm” que describirá los pasos necesarios para llevar a cabo la búsqueda Tabú. Desde el propio objeto instanciado se llamará a la función ejecuta.

3.1 TIPO “Router”

Emplea 3 atributos:

1. **“radius”:** el valor del radio.
2. **“posx”:** la posición en el eje “X”.
3. **“posy”:** la posición en el eje “Y”.

Tiene su método de inicialización, “__init__”, sus métodos get’s y set’s para los atributos, así como un método de representación por pantalla, “__repr__” y “str”.

```
class router(object):

    def __init__(self, posx, posy, radius=None):

        self.__radius = radius

        self.__posx = posx

        self.__posy = posy

    @property
    def radius(self):
        return self.__radius

    @property
    def posx(self):
        return self.__posx

    @property
    def posy(self):
        return self.__posy

    def setRadius(self, value):
        self.__radius = value

    def setPosx(self, value):
        self.__posx = value

    def setPosy(self, value):
        self.__posy = value

    @property
    def str(self):
        cad = "[X: "+str(self.posx)+
              ", Y: "+str(self.posy)+
              ", Radio: "+str(self.radius)+"]"
        return cad

    def __repr__(self):
        return self.str
```

3.2 TIPO "Client"

Emplea 2 atributos:

1. **"posx":** la posición en el eje "X".
2. **"posy":** la posición en el eje "Y".

Tiene su método de inicialización, "__init__", sus métodos get's y set's para los atributos, así como un método de representación por pantalla, "__repr__" y "str".

```
class client(object):

    def __init__(self, posx, posy):

        self.__posx = posx

        self.__posy = posy

    @property
    def posx(self):
        return self.__posx

    @property
    def posy(self):
        return self.__posy

    @property
    def str(self):
        cad = "[X: "+str(self.posx)+", Y: "+str(self.posy)+"]"
        return cad

    def __repr__(self):
        return self.str
```

3.3 TIPO: "Movement"

Emplea 7 atributos:

1. **"type":** el tipo que viene a determinar si es un movimiento "swap" de intercambio entre routers o un "moveToCell" para mover un determinado router a una determinada posición.
2. **"router_to_moveToCell":** el índice del router que va a ser movido de una posición inicial de la que parte, a nueva posición en el plano cartesiano 2D. Se emplea cuando el tipo de movimiento es "moveToCell".
3. **"posx_to_moveToCell":** la posición en el eje "X" a la que vamos a mover el router indicado en el atributo "router_to_moveToCell". Se emplea cuando el tipo de movimiento es "moveToCell".

4. **“posy_to_moveToCell”:** la posición en el eje “Y” a la que vamos a mover el router indicado en el atributo “router_to_moveToCell”. Se emplea cuando el tipo de movimiento es “moveToCell”.
5. **“router1_to_swap”:** índice del router que va a colocarse en la posición que ocupa el router indicado por el atributo “router2_to_swap”. Se emplea cuando el tipo de movimiento es “swap”.
6. **“router2_to_swap”:** índice del router que va a colocarse en la posición que ocupa el router indicado por el atributo “router1_to_swap”. Se emplea cuando el tipo de movimiento es “swap”.
7. **“lastIterationAsTabu”:** número que viene a indicar la última iteración en la cual este movimiento se estableció como movimiento tabú.

Tiene su método de inicialización, “__init__”, sus métodos get’s y set’s para los atributos, así como un método de representación por pantalla, “__repr__” y “str”.

```
class movement(object):

    def __init__(self, type, router_to_moveToCell=None,
                 posx_to_moveToCell=None, posy_to_moveToCell=None,
                 router1_to_swap=None, router2_to_swap=None):

        self.__type = type

        self.__router_to_moveToCell = router_to_moveToCell

        self.__posx_to_moveToCell = posx_to_moveToCell

        self.__posy_to_moveToCell = posy_to_moveToCell

        self.__router1_to_swap = router1_to_swap

        self.__router2_to_swap = router2_to_swap

        self.__LastIterationAsTabu = 0

    @property
    def type(self):
        return self.__type

    @property
    def router_to_moveToCell(self):
        return self.__router_to_moveToCell

    @property
    def posx_to_moveToCell(self):
        return self.__posx_to_moveToCell

    @property
    def posy_to_moveToCell(self):
        return self.__posy_to_moveToCell

    @property
    def router1_to_swap(self):
        return self.__router1_to_swap
```

```
@property
def router2_to_swap(self):
    return self.__router2_to_swap

@property
def LastIterationAsTabu(self):
    return self.__LastIterationAsTabu

def setLastIterationAsTabu(self, value):
    self.__LastIterationAsTabu = value

@property
def str(self):

    cad = ""
    if(self.type == "swap"):
        cad += "[type: swap, "+str(self.router1_to_swap)+
               "<->"+str(self.router2_to_swap)+"]"
    else:
        if(self.type == "moveToCell"):
            cad += "[type: moveToCell, "+
                   str(self.router_to_moveToCell)+
                   "->("+str(self.posx_to_moveToCell)+
                   ", "+str(self.posy_to_moveToCell)+")]"
    return cad

def __repr__(self):
    return self.str
```

3.4 TIPO: “Solution”

Emplea 6 atributos:

1. **“routers”:** contiene una lista de objetos de tipo “router”, que son los que conforman la solución.
2. **“fitnessValue”:** lengthGreatestConnectedSet + numberClientsCovered si lengthGreatestConnectedSet == len(solution.routers), sino lengthGreatestConnectedSet. En la práctica no va a ser utilizado.
3. **“lengthGreatestConnectedSet”:** indica el tamaño de la mayor componente conexa que se haya en la solución.
4. **“numberClientsCovered”:** indica el número de clientes que se haya cubierto por, al menos, 1 router de la solución.
5. **“precededSolution”:** un objeto de tipo “Solution” que identifica la solución original a partir de la cual aplicándole el movimiento “appliedMovement” se ha llegado a esta solución.
6. **“appliedMovement”:** un objeto de tipo “Movement” que identifica el movimiento que ha sido aplicado a la solución “precededSolution” para llegar a esta solución.

Tiene su método de inicialización, “__init__”, sus métodos get’s y set’s para los atributos, así como un método de representación por pantalla, “__repr__” y “str”.

Además, se ha implementado un método “Modify” el cual se empleará para construir las nuevas soluciones que se vayan encontrando durante la búsqueda. La idea es hacer una copia de la “current_solution” empleando el método “deepcopy”, y a partir de esa copia hacer las modificaciones oportunas con el método “Modify” para construir la nueva solución.

```
class solution(object):

    def __init__(self, routers, precededSolution=None,
                 appliedMovement=None):

        self.__routers = routers

        self.__fitnessValue = None

        self.__lengthGreatestConnectedSet = None

        self.__numberClientsCovered = None

        self.__precededSolution = precededSolution

        self.__appliedMovement = appliedMovement

    def modify(self, index, posx=None, posy=None,
               precededSolution=None, appliedMovement=None):

        if(not(isinstance(posx, type(None)))):
            self.__routers[index].setPosx(posx)

        if(not(isinstance(posy, type(None)))):
            self.__routers[index].setPosy(posy)

        if(not(isinstance(precededSolution, type(None)))):
            self.__precededSolution = precededSolution

        if(not(isinstance(appliedMovement, type(None)))):
            self.__appliedMovement = appliedMovement

    @property
    def routers(self):
        return self.__routers

    @property
    def fitnessValue(self):
        return self.__fitnessValue

    def setFitnessValue(self,value):
        self.__fitnessValue = value

    @property
    def numberClientsCovered(self):
        return self.__numberClientsCovered

    @property
    def lengthGreatestConnectedSet(self):
        return self.__lengthGreatestConnectedSet

    def setNumberClientsCovered(self,value):
        self.__numberClientsCovered = value
```

```
def setLengthGreatestConnectedSet(self,value):
    self.__lengthGreatestConnectedSet = value

@property
def appliedMovement(self):
    return self.__appliedMovement

@property
def precededSolution(self):
    return self.__precededSolution

@property
def str(self):
    cad = "##########################################################"
    cad += "\n[\n"
    routers = self.routers
    for i in range(0, len(routers)):
        if(not(i+1==len(routers))):
            cad += "Router "+str(i)+": "+routers[i].str+",\n"
        else:
            cad += "Router "+str(i)+": "+routers[i].str
    cad += "\n]\n"
    cad += "lengthGreatestConnectedSet: "+
           str(self.lengthGreatestConnectedSet)+"\n"
    cad += "numberClientsCovered: "+
           str(self.numberClientsCovered)+"\n"

    if(not(isinstance(self.__appliedMovement, type(None)))):
        cad += "appliedMovement: "+self.__appliedMovement.str+"\n"
    cad += "##########################################################"
    return cad

def __repr__(self):
    return self.str
```

3.5 TIPO: “TSMeshAlgorithmProblem”

```
class TSMeshAlgorithmProblem(object):
```

Implementa un simulador de algoritmo Tabú Search.

Presenta los siguientes métodos:

3.5.1 MÉTODOS

3.5.1.1 MÉTODO: “__init__”

Instancia un objeto de tipo “TSMeshAlgorithmProblem” con los siguientes atributos:

```
def __init__(self, width, high, number_routers,
             number_clients,number_iterations, movement_type,
             tabu_size, elite_size,LARGE_HASH,
             max_number_iterations_without_improvement,
              probability_distribution,
             diversification_percentage, coverage_range):
```

3.5.1.1.1 ATRIBUTOS

- **“width”:** un número que indica el ancho del plano cartesiano 2D donde se van a desplegar los routers y los clientes.
- **“high”:** un número que indica el alto del plano cartesiano 2D donde se van a desplegar los routers y los clientes.
- **“number_routers”**: un número que indica el número de routers que se van a desplegar en el plano cartesiano 2D y que va a tener las soluciones que vayamos encontrando a lo largo de la búsqueda.
- **“number_clients”:** un número que indica el número de clientes que se van a desplegar en el plano cartesiano 2D y para los cuales queremos encontrar un emplazamiento para los routers que nos permita que el máximo número de clientes se vea cubierto por, al menos, un router.
- **“number_iterations”:** un número que se va a emplear en la condición de parada del algoritmo de búsqueda Tabú y que indica el número de iteraciones que va a llevar a cabo el bucle while del algoritmo Tabú Search.
- **“movement_type”:** tipo de movimiento que se va a emplear en la iteración i-ésima del algoritmo de búsqueda Tabú, “swap” o “moveToCell”.
- **“tabu_size”:** número que indica el tamaño de la lista tabú que contendrá el hash de las soluciones ya visitadas.

- **“elite_size”:** número que indica el tamaño de la lista de soluciones de élite, “best_sols”, encontradas a lo largo de la búsqueda.
- **“LARGE_HASH”:** número empleado para generar números aleatorios en el rango [1, LARGE_HASH], para generar los hashes de las soluciones.
- **“coverage_range”:** un rango de números empleado para generar los radios de los routers de la solución inicial.

```
self.__width = width
self.__high = high
self.__number_routers = number_routers
self.__number_clients = number_clients
self.__number_iterations = number_iterations
self.__movement_type = movement_type
self.__tabu_size = tabu_size
self.__elite_size = elite_size
self.__LARGE_HASH = LARGE_HASH
self.__coverage_range = coverage_range
```

- **“max_tabu_status”:** entero igual a number_routers/2.
- **“aspiration_value”:** número que indica el número de iteraciones que permanece en estado tabú un movimiento. Dicho valor será igual a (max_tabu_status/2) – log2(max_tabu_status).

```
############################################################
####### DERIVATIVES ########################################
############################################################

self.__max_tabu_status = number_routers/2
self.__aspiration_value =
            (self.__max_tabu_status/2) -
            log(self.__max_tabu_status, 2)
```

- **“current_solution”:** variable que almacena un objeto de tipo “Solution”, la solución actual. Puede empeorar o mejorar en su valor de fitness.
- **“previous_current_solution”:** copia de “current_solution” que se hace antes de asignar a “current_solution” la “best_admissible_solution”. Se utiliza para compararse con respecto a “current_solution” y verificar si la “current_solution” ha cambiado o no.
- **“best_solution_found”:** variable que almacena un objeto de tipo “Solution”, la mejor solución encontrada hasta el momento. Nunca empeora en su valor de fitness, siempre mejora o se queda igual.

```
############################################################
####### SOLUTIONS ##########################################
############################################################

self.__current_solution = None
self.__previous_current_solution = None
self.__best_solution_found = None
```

- **“clients”:** variable que almacena la lista de objetos de tipo “client”, los cuales son desplegados en el plano cartesiano 2D.

```
############################################################
####### CLIENTS ############################################
############################################################

                self.__clients = None
```

- **“TL”:** lista tridimensional de tamaño “number_routers” x ”width” x ”high”. Se indexa utilizando 3 índices:

  1. **r:** índice del router, de 0 a “number_routers” – 1.
  2. **posx:** posición en el eje “X”, de 0 a “width” – 1.
  3. **posy:** posición en el eje “Y”, de 0 a “high” – 1.

  El valor asociado a TL[r][posx][posy] es un número que identifica la última iteración en la cual se estableció como tabú que el router “r” estuviese en la posición (“posx”, ”posy”).

  La tabla se inicializa con todos los valores TL[r][posx][posy] a “-1”.

- **“hashing”:** lista tridimensional con las mismas características que “TL”, con la diferencia de que se inicializa con todos los valores hashing[r][posx][posy] a “random.int(1, “LARGE_HASH”)”. Esta tabla tridimensional compuesta por números aleatorios será empleada para generar el hash de una solución.
- **“TH”:** lista que contiene objetos de tipo “boolean”. El tamaño de la lista es de “tabu_size” elementos y se indexa con: TH[“solution.hash()” mod “tabu_size”]. Si el valor asociado es “True”, significa que la solución identificada con su hash, “solution.hash()”, ya ha sido visitada.

```
############################################################
####### SHORT TERM MEMORY ##################################
############################################################

                self.__TL = None
                self.__hashing = None
                self.__TH = None
```

- **“frecuency”:** tabla tridimensional que se indexa con el router “r”, su posición en el eje “X” “posx” y su posición en el eje “Y” “posy”. El valor asociado a frecuency[r][posx][posy] será el número de veces que el router “r” ha sido asignado a la posición (posx, posy). Se utilizará durante el proceso de intensificación, para generar la variable “suma” que después utilizaremos en el proceso de criterio de la ruleta, esto es el método “roulette_wheel_criteria”.
- **“most_frecuently_positions”:** lista bidimensional de tamaño “width” x “high” que almacena para cada posición (posx, posy) el número de veces. Se utiliza en el proceso de diversificación, para

mover los routers que menos se han movido a las posiciones donde más movimientos ha habido. Tiene menor complejidad computacional que "frecuency", del orden de una unidad menos, esto es, O(n^3) frente a O(n^2), de ahí que la usemos

- **"tfrecuency":** lista unidimensional de tamaño "number_routers" que lleva la cuenta del número de veces que ha cambiado de posición cada router. Se utiliza en el proceso de diversificación, para conocer que routers de la "current_solution" son los que menos se han movido durante la ejecución del algoritmo TS.
- **"best_sols"**: lista unidimensional que contiene objetos de tipo "Solution", concretamente las "elite_size" mejores soluciones encontradas hasta el momento. Se utiliza en el proceso de intensificación, para agregar características de las mejores soluciones a nuestra "current_solution".

```
############################################################
####### LONG TERM MEMORY ###################################
############################################################

                self.__frecuency = None
                self.__most_frecuently_positions = None
                self.__tfrecuency = None
                self.__best_sols = None
```

- **"actual_iteration":** número que indica la iteración actual en la que se encuentra el bucle while que forma parte de la ejecución del algoritmo Tabú Search. En cada iteracción del bucle while, este número se verá incrementado en una unidad.

```
############################################################
####### ACTUAL ITERATION ###################################
############################################################

                self.__actual_iteration = None
```

- **"diversification_percentage":** número real entre 0 y 1 que indica el porcentaje de routers, de la "current_solution", que menos se ha movido durante la ejecución del algoritmo TS y que van a ser movidos de posición durante la fase de diversificación. 0 significa ninguno de los routers y 1 significa los "number_routers" de la "current_solution", es decir todos.
- **"number_iterations_without_improvement":** número que lleva la cuenta del número de iteraciones que han pasado sin que haya habido mejoría del valor de fitness en la "best_solution_found". Una vez que este valor alcance la "max_number_iterations_without_improvement", se procederá a hacer diversificación.
- **"max_number_iterations_without_improvement":** número que indica el número de iteraciones necesarias que han de pasar sin que

haya una mejora del valor de fitness en la “best_solution_found” para que se lleve a cabo diversificación.

```
############################################################
####### DIVERSIFICATION ####################################
############################################################

                if(not(
                      diversification_percentage >= 0.0 and
                      diversification_percentage <= 1.0)):
                            raise Exception("Error")
                self.__diversification_percentage =
                      diversification_percentage
                self.__number_iterations_without_improvement =
                      None
                self.__max_number_iterations_without_improvement =
                      max_number_iterations_without_improvement
```

- **“probability_distribution”:** indica el tipo de distribución utilizada para generar las posiciones de los routers y de los clientes en la solución inicial de la que se parte, así como el radio de los routers.

```
############################################################
####### DISTRIBUTION POINTS ################################
############################################################

                self.__probability_distribution =
                                  probability_distribution
```

3.5.1.2 MÉTODO: “executeTSMeshAlgorithm”

Este es el método principal que implementa la resolución del problema a través de la heurística Tabú Search, esto es, un conjunto de pasos descritos y llamadas a otros métodos, describiendo así la búsqueda Tabú.

Desde el propio objeto instanciado se llamará a la función ejecuta:

```
def instantiateTSMeshAlgorithmProblem():

      instance = TSMeshAlgorithmProblem(16,16,4,12,256,"swap",51113,
                                        15,2^(32),10,"Uniform",0.25,
                                        [4, 1])

      instance.executeTSMeshAlgorithm
```

La llamada al método instanciador se haría de la siguiente forma:

```
      instantiateTSMeshAlgorithmProblem()
```

La secuencia de pasos que describe “executeTSMeshAlgorithm” es la siguiente:

3.5.1.2.1 PSEUDO CÓDIGO

Clientes <- Desplegar_clientes

Solución_actual <- Desplegar_solución_inicial

Dibujar_solución(Solución_actual)

Mejor_solución_encontrada <- Solución_actual

Resetear_listas

Iteración_actual <- 0

Mientras(Not (Condición_terminación))

    Admisible <- Generar_conjunto_admisible

    Solución_actual_previa <- Solución_actual

    Solución_actual <- Menos_mala(Admisible)

    Si (Condición_Intensificación):

        Intensificación

    Si (Condición_Diversificación)

        Diversificación

    Actualizar_listas_recencia

    Actualizar_listas_frecuencia

    Iteración_actual <- +1

Dibujar_solución(Mejor_solución_encontrada)

Devolver Mejor_solución_encontrada

3.5.1.2.2 CÓDIGO SAGE_PYTHON

```
@property
def executeTSMeshAlgorithm(self):
self.__clients = self.compute_clients
self.__current_solution = self.compute_initial_solution
self.__best_solution_found = deepcopy(self.__current_solution)
print(self.__current_solution)
self.plot_solution(self.__current_solution)
self.reset_TL
self.reset_hashing
self.reset_TH
self.reset_frecuency
self.reset_most_frecuently_positions
```

```
self.reset_tfrecuency
self.__best_sols = []
self.__number_iterations_without_improvement = 0
self.__actual_iteration = 0
while(self.__actual_iteration < self.__number_iterations):
      Admissible = self.Admissible(self.__current_solution)
      self.__previous_current_solution =
                                    deepcopy(self.__current_solution)
      self.__current_solution = Admissible.pop(0)
      print(self.__current_solution)
      for solution in Admissible:
            print(solution)
            if(self.is_better_than(solution, self.__current_solution)):
                  self.__current_solution = solution
      if(self.is_better_than(self.__current_solution,
                                          self.__best_solution_found)):
            self.__best_solution_found =
                                    deepcopy(self.__current_solution)
            self.__number_iterations_without_improvement = 0
      else:
            self.__number_iterations_without_improvement += 1
      if((len(self.__best_sols) == self.__elite_size) and
            (self.__actual_iteration %
            self.__max_number_iterations_without_improvement) == 0):
                  print "#Intensification"
                  self.Intensification(self.__current_solution)
                  print "#End_Intensification"
      if(self.__number_iterations_without_improvement ==
                  self.__max_number_iterations_without_improvement):
                  print "#Diversification"
                  self.Diversification(self.__current_solution)
                  self.__number_iterations_without_improvement = 0
                  print "#End_Diversification"
      self.update_recency(self.__current_solution)
      self.update_frecuency(self.__current_solution)
      self.__actual_iteration += 1
print(self.__best_solution_found)
self.plot_solution(self.__best_solution_found)
```

3.5.1.3 MÉTODO “plot_solution”

Este método tiene por objetivo representar gráficamente una solución, esto es, representar en un plano cartesiano 2D los puntos con sus coordenadas “X” e “Y” para los routers y para los clientes, y las circunferencias que definen los radios de cobertura de los routers trazadas con líneas discontinuas.

Requiere el uso de “SAGE” para su funcionamiento. En caso de Python puro harían falta el uso de bibliotecas matplotlibrary y configuraciones adicionales para su fucinoamiento.

3.5.1.3.1 PSEUDO CÓDIGO

Gráfico = Generar_gráfico_vacío

Para_todo_router_en_solución

    Gráfico <- Pintar_router

Para_todo_cliente_en_clientes

    Gráfico <- Pintar_cliente

Pintar_info_extra

Pintar_gráfico (Gráfico)

3.5.1.3.2 CÓDIGO SAGE_PYTHON

```
def plot_solution(self, solution):
        g = Graphics()
        for router in solution.routers:
            p = point((router.posx, router.posy), pointsize = 100)
            g += p.plot()
            c = circle((router.posx, router.posy),
                       router.radius,
                       rgbcolor = (1,0,0),
                       linestyle = '--')
            g += c.plot()
        for client in self.__clients:
            p = point((client.posx, client.posy),
                      pointsize = 50, rgbcolor = "green")
            g += p.plot()
        t = text(u""+solution.str+
                  "\nlengthGreatestConnectedSet: "+
                  str(solution.lengthGreatestConnectedSet)+
                  "/"+str(self.__number_routers)+
                 "\nnumberClientsCovered: "+
                  str(solution.numberClientsCovered)+
                  "/"+str(self.__number_clients),
                 (1,-4), bounding_box={'boxstyle':'round', 'fc':'w'},
                 horizontal_alignment="left"
                )
        g += t.plot()
        g.show(figsize=10)
```

3.5.1.4 MÉTODO: “compute_clients”

Este método se encarga de generar una lista de objetos de tipo “Client”.

Se limita a hacer una llamada al método “random_2D_objects”, el cual se encargará de generar un total de “number_clients” objetos de tipo “Client” con posiciones en el eje “X” “posx” y posiciones en el eje “Y” “posy” que sigan la distribución de probabilidad “probability_distribution”.

3.5.1.4.1 PSEUDO CÓDIGO

Número <- Número_clientes

Distribución <- Distribución_probabilidad

Rango <- None

Clientes <- Generar_objetos_2D(Número, Distribución, Rango)

Devolver Clientes

3.5.1.4.2 CÓDIGO SAGE_PYTHON

```
@property
    def compute_clients(self):
        clients = self.random_2D_objects(
                                    self.__number_clients,
                                    self.__probability_distribution)
        return clients
```

3.5.1.5 MÉTODO: "random_2D_objects"

Este método se encarga de generar una lista de objetos de tipo "Client" o una lista de objetos de tipo "Router".

Es un método que es llamado desde "compute_clients" y "compute_initial_solution" respectivamente.

En el primero de los casos, se devuelve la lista de objetos de tipo "Client" y desde "compute_clients" es asignada al atributo "clients".

En el segundo caso, se devuelve la lista de objetos de tipo "Router" y desde "compute_initial_solution" se construye el objeto "Solution" mediante la llamada al constructor, pasándole por parámetro de entrada la lista.

Para generar las características de los objetos, esto es, "radius", "posx" y "posy", se sigue una distribución de probabilidad que es recibida por el método por parámetro de entrada.

3.5.1.5.1 PSEUDO CÓDIGO

Resultado <- [ ]

i <- 0

Mientras (i < Número_puntos_a_generar)

Si (Distribución_probabilidad = Uniformal)

Posx <- Uniformal[0, …, Ancho_cuadrícula – 1]

Posy <- Uniformal[0, …, Largo_cuadrícula – 1]

Si (NOT (rango_radio == None)):

Radio <- Uniformal[rango_radio]

Resultado <- +Router(Posx, Posy, Radio)

Sino

Resultado <- + Cliente(Posx, Posy)

Sino

#Implementar otras Distribuciones de probabilidad

Devolver Resultado

3.5.1.5.2 CÓDIGO SAGE_PYTHON

```
def random_2D_objects(self, number_points,
                      probability_distribution, radius_range=None):
        res = []
        i=0
        while(i < number_points):
            if(probability_distribution == "Uniform"):
                a = 0
                b = self.__width-1
                c = self.__high-1
                T = RealDistribution('uniform', [a, b])
                posx = round(T.get_random_element())
                T = RealDistribution('uniform', [a, c])
                posy = round(T.get_random_element())
                if(not(isinstance(radius_range, type(None)))):
                    a = radius_range[1]
                    b = radius_range[0]
                    T = RealDistribution('uniform', [a, b])
                    radius = T.get_random_element()
                    res.append(router(posx, posy, radius))
                else:
                    res.append(client(posx, posy))
            else:
                if(probability_distribution == "Normal"):
                    print "WIP"
                else:
                    if(probability_distribution == "Exponential"):
                        print "WIP"
                    else:
                        if(probability_distribution == "Weibull"):
                            print "WIP"
            i += 1
        return res
```

3.5.1.6 MÉTODO: “compute_initial_solution”

Este método se encarga de generar una solución inicial, esto es, una lista de objetos de tipo “Router”, como punto de partida del algoritmo Tabú Search.

Se limita a hacer una llamada al método “random_2D_objects”, el cual se encargará de generar un total de “number_routers” objetos de tipo “Router” con posiciones en el eje “X” “posx” y posiciones en el eje “Y” “posy” que sigan la distribución de probabilidad “probability_distribution”, así como valores para el atributo “radius” obtenidos de manera aleatoria bajo la distribución de probabilidad “probability_distribution” de entre un rango de valores definido por “coverage_range”.

Una vez generada la solución inicial, se hace una llamada al método “compute_fitness_value” para calcular y asignar a sus atributos “lengthGreatestConnectedSet” y “numberClientsCovered” el tamaño de la componente conexa más grande encontrada y el número de clientes que cubre la solución, respectivamente.

3.5.1.6.1 PSEUDO CÓDIGO

Número <- Número_routers_solución

Distribución <- Distribución_probabilidad

Rango <- rango_routers

Solución_actual <- Generar_objetos_2D(Número, Distribución, rango)

Calcular_fitness(Solucion_actual)

Devolver Solución_actual

3.5.1.6.2 CÓDIGO SAGE_PYTHON

```
@property
    def compute_initial_solution(self):
        routers = self.random_2D_objects(
                                self.__number_routers,
                                self.__probability_distribution,
                                self.__coverage_range)
        initial_solution = solution(routers)
        self.compute_fitness_value(initial_solution)
        return initial_solution
```

3.5.1.7 MÉTODO: “compute_fitness_value”

Este método recibe un objeto de tipo “Solution”, y su función es la de llamar al método encargado de encontrar el tamaño de la componente conexa más grande, este es el “compute_greatest_connectedSet()”, así como el método encargado de encontrar el número de clientes que está cubierto por, al menos, un router, este es el “compute_number_clients_covered()”.

Una vez se obtienen estos dos datos, se llama a los respectivos métodos set’s del objeto de tipo “Solution” para asignar a sus atributos “lengthGreatestConnectedSet” y “numberClientsCovered”, los dos datos anteriormente citados.

3.5.1.7.1 PSEUDO CÓDIGO

MayorComponenteConexa = Computar_mayor_componente_conexa(solución)

NúmeroClientesCubiertos = Computar_número_clientes_cubiertos(solución)

solución.MayorComponenteConexa <- MayorComponenteConexa

solución.NúmeroClientesCubiertos <- NúmeroClientesCubiertos

3.5.1.7.2 CÓDIGO SAGE_PYTHON

```
def compute_fitness_value(self, solution):
        length_greatest_connectedSet =
        self.compute_greatest_connectedSet(deepcopy(solution).routers)

        number_clients_covered =
        self.compute_number_clients_covered(solution.routers)

        solution.setLengthGreatestConnectedSet(
                                        length_greatest_connectedSet)

        solution.setNumberClientsCovered(number_clients_covered)
```

3.5.1.8 MÉTODO: “compute_greatest_connectedSet”

Este método tiene por objetivo obtener las componentes conexas que conforman la red de routers interconectados y obtener el tamaño de la más grande.

Este algoritmo trabaja con 2 listas, la primera se irá quedando a lo largo de la ejecución del algoritmo con todos aquellos routers que componen una componente conexa, mientras que la segunda se irá quedando con todos aquellos routers que no pertenecen a la componente conexa de la primera lista.

A la hora de instanciar el algoritmo, la primera lista contendrá los “n” routers de una posible solución.

La idea consiste en utilizar 2 índices para iterar, “i” y “j”. Con el primer índice “i” fijamos el primer router de la lista “l1”, y con el segundo índice “j” vamos seleccionando el resto de routers de la lista “l1”.

Determinar si el router “i” y el router “j” se hayan interconectados es tan sencillo como limitarse a comprobar si la suma de sus radios es mayor o igual que la distancia euclídea entre sus 2 posiciones en el plano cartesiano 2D.

Si “i” y “j” se hayan interconectados, entonces se mantiene el router “j” en la lista “l1”.

En caso contrario, eliminamos el router “j” de “l1” y lo introducimos en “l2”.

Una vez que con el índice “j” terminamos de recorrer “l1”, actualizamos ambos índices: i=i+1 y j=i+1, y repetimos el mismo proceso anteriormente descrito hasta que i=l1.size(), que marca la condición de parada y nos indica que en la lista l1 tenemos todos aquellos routers que conforman una componente conexa, mientras que en l2 tenemos todos aquellos routers que no forman parte de la componente conexa que conforman los routers de la lista l1 y que posiblemente pudieran conformar otras componentes conexas entre ellos.

En este momento se hace la función “Máx” entre l1.size() y lo que nos devuelva la llamada recursiva que debemos hacer al algoritmo empleando en dicha llamada recursiva como “l1” la “l2” que contiene aquellos routers que no formaban parte de la componente conexa encontrada.

El caso base del algoritmo es que la lista “l1” sea una lista con 0 o 1 elemento, en cuyo caso se retornaría uno de dichos valores en vez de llevar a cabo el proceso de bucle iterativo de búsqueda de la componente conexa empleando índices “i” y “j” que mencionamos anteriormente.

3.5.1.8.1 PSEUDO CÓDIGO 1

```
Conjunto_no_conexo <- Lista_entrada

Conjunto_conexo <- Conjunto_no_conexo[ 0 ]

Para_todo_router_en_Conjunto_conexo:

    Para_todo_router_en_Conjunto_no_conexo:

        Si ( Enlace( router_conexo, router_no_conexo ) )

            Quitar( router_no_conexo, Conjunto_no_conexo )

            Añadir_al_final( router_no_conexo, Conjunto_conexo )

Devolver Máximo( Conjunto_conexo, Recursiva( Conjunto_no_conexo ) )
```

3.5.1.8.2 PSEUDO CÓDIGO 2 #Enlace

Router_1 <- Párametro_entrada_1

Router_2 <- Párametro_entrada_2

Distancia <- Distancia_euclídea(Router_1, Router_2)

Suma_radios <- Radio(Router_1) + Radio(Router_2)

Si (Suma_radios >= Distancia)

    Devolver True

Sino

    Devolver False

3.5.1.8.3 CÓDIGO SAGE_PYTHON

```
def compute_greatest_connectedSet(self, nonConnectedSet):
        connectedSet = []
        new_nonConnectedSet = nonConnectedSet
        if(not(len(new_nonConnectedSet)==0)):
            connectedSet.append(new_nonConnectedSet.pop(0))
        if(nonConnectedSet==[]):
            return (len(connectedSet))
        else:
            i=0
            while(i<len(connectedSet)):
                j=0
                while(j<len(new_nonConnectedSet)):
                    distancia = sqrt(
                                    (connectedSet[i].posx -
                                     new_nonConnectedSet[j].posx)^(2)
                                         +
                                     (connectedSet[i].posy -
                                     new_nonConnectedSet[j].posy)^(2)
                                    )
                    if(connectedSet[i].radius +
                       new_nonConnectedSet[j].radius >=
                       distancia):
                         connectedSet.append(new_nonConnectedSet[j])
                         new_nonConnectedSet.pop(j)
                         j=j-1
                    j=j+1
                i=i+1
        return max(len(connectedSet),
self.compute_greatest_connectedSet(new_nonConnectedSet))
```

3.5.1.9 MÉTODO: “compute_number_clients_covered”

Este método genera el número de clientes que hay cubiertos por, al menos, un router. Esto es el valor de la segunda función de fitness utilizada y que funciona a modo de desempate con respecto a la primera función de fitness, la del tamaño de la máxima componente conexa, que es la que tiene prioridad.

La idea consiste en anidar 2 bucles, un primer bucle externo recorriendo los clientes y un segundo bucle interno recorriendo los routers.

Siendo “i” el cliente y “j” el router, determinar si “j” cubre a “i” consiste en verificar si su radio de convergencia es mayor o igual que la distancia euclídea entre la posición del cliente “i” y la posición del router “j” en el plano cartesiano 2D.

Si está cubierto, incrementamos en una unidad la variable suma, empleada para llevar la cuenta, y hacemos “break” para salir del bucle interno y pasar al siguiente cliente.

Recibe por parámetro de entrada una lista de objetos router.

3.5.1.9.1 PSEUDO CÓDIGO

Resultado <- 0

Distancia <- 0

Para_todo_cliente_en_clientes

    Para_todo_router_en_solucion

        Distancia <- Distancia_euclídea(router, cliente)

        Si (Distancia <= Radio(router))

            Resultado <- +1

            Break

Devolver Resultado

3.5.1.9.2 CÓDIGO SAGE_PYTHON

```
def compute_number_clients_covered(self, routers):
        res = 0
        for client in self.__clients:
            for router in routers:
                distancia = sqrt((router.posx-client.posx)^(2)+
                                 (router.posy-client.posy)^(2))
                if(distancia <= router.radius):
                    res = res + 1
                    break
        return res
```

3.5.1.10 MÉTODO: “reset_TL”

Este método genera una lista tridimensional, donde todos los valores están definidos inicialmente a “-1”.

El valor “-1” codificará que aún no se ha establecido como Tabú una última iteración del router “r” en la posición del eje “X“ “posx” y en la posición del eje “Y” “posy”.

Para generar la lista tridimensional, hace falta anidar 3 bucles: el primero de ellos con condición de parada el atributo “number_routers”, el segundo de ellos con condición de parada el atributo “width” y el tercero de ellos con condición de parada el atributo “high”.

Se actualizará en el método “update_recency”.

3.5.1.10.1 PSEUDO CÓDIGO

Resultado <- [ ]

Para_todo_i_en_rango(0, Número_routers_solución – 1)

    Auxiliar1 <- [ ]

    Para_todo_j_en_rango(0, Ancho_cuadrícula – 1)

        Auxiliar2 <- [ ]

        Para_todo_k_en_rango(0, Largo_cuadrícula – 1)

            Auxiliar2 <- +(-1)

        Auxiliar1 <- +(Auxiliar2)

    Resultado <- +(Auxiliar1)

Devolver Resultado

3.5.1.10.2 CÓDIGO SAGE_PYTHON

```
@property
def reset_TL(self):
        res = []
        for i in range(self.__number_routers):
            aux1 = []
            for j in range(self.__width):
                aux2 = []
                for k in range(self.__high):
                    aux2.append(-1)
                aux1.append(aux2)
            res.append(aux1)
        self.__TL = res
```

3.5.1.11 MÉTODO: “reset_hashing”

Este método genera una lista tridimensional donde cada uno de los valores está definido como un número aleatorio en el rango [1, LARGE_HASH].

Para generar la lista tridimensional, hace falta anidar 3 bucles: el primero de ellos con condición de parada el atributo “number_routers”, el segundo de ellos con condición de parada el atributo “width” y el tercero de ellos con condición de parada el atributo “high”.

Los números, generados aleatoriamente, que contiene esta tabla serán utilizados para generar los hashes de las soluciones que vayamos encontrando a lo largo del proceso de búsqueda, y poder identificar si una solución ha sido previamente visitada.

3.5.1.11.1 PSEUDO CÓDIGO

Resultado <- [ ]

Para_todo_i_en_rango(0, Número_routers_solución – 1)

    Auxiliar1 <- [ ]

    Para_todo_j_en_rango(0, Ancho_cuadrícula – 1)

        Auxiliar2 <- [ ]

        Para_todo_k_en_rango(0, Largo_cuadrícula – 1)

            Número <- Generar_aleatorio(1, …, Large_hash)

            Auxiliar2 <- +(Número)

        Auxiliar1 <- +(Auxiliar2)

    Resultado <- +(Auxiliar1)

Devolver Resultado

3.5.1.11.2 CÓDIGO SAGE_PYTHON

```
@property
def reset_hashing(self):
        res = []
        for i in range(self.__number_routers):
            aux1 = []
            for j in range(self.__width):
                aux2 = []
                for k in range(self.__high):
                    aux2.append(randint(1,self.__LARGE_HASH))
                aux1.append(aux2)
            res.append(aux1)
        self.__hashing = res
```

3.5.1.12 MÉTODO: “reset_TH”

Este método genera una lista de “tabu_size” elementos “boolean” con valor “False”.

Dicha lista la utilizaremos para verificar si una solución ya ha sido previamente visitada.

La lista se indexa de la siguiente forma: TH[“solution.hash()” mod “tabu_size”].

Si el valor asociado es “True”, significa que la solución identificada con su hash, “solution.hash()”, ya ha sido visitada.

Se actualizará en el método “update_recency”.

3.5.1.12.1 PSEUDO CÓDIGO

Resultado <- [ ]

Para_todo_i_en_rango(0, Tamaño_lista_tabú – 1)

    Resultado <- +False

Devolver Resultado

3.5.1.12.2 CÓDIGO SAGE_PYTHON

```
@property
def reset_TH(self):
        res = [False]*self.__tabu_size
        self.__TH = res
```

3.5.1.13 MÉTODO: “reset_frecuency”

Genera una tabla tridimensional que se indexa con el router “r”, su posición en el eje “X” “posx” y su posición en el eje “Y” “posy”.

El valor asociado a frecuency[r][posx][posy] será el número de veces que el router “r” ha sido asignado a la posición (posx, posy).

Se utilizará durante el proceso de intensificación, para generar la variable “suma” que después utilizaremos en el proceso de criterio de la ruleta, esto es el método “roulette_wheel_criteria”.

Se actualizará en el método de “update_frecuency”.

3.5.1.13.1 PSEUDO CÓDIGO

Resultado <- [ ]

Para_todo_i_en_rango(0, Número_routers_solución – 1)

Auxiliar1 <- [ ]

Para_todo_j_en_rango(0, Ancho_cuadrícula – 1)

Auxiliar2 <- [ ]

Para_todo_k_en_rango(0, Largo_cuadrícula – 1)

Auxiliar2 <- +(0)

Auxiliar1 <- +(Auxiliar2)

Resultado <- +(Auxiliar1)

Devolver Resultado

3.5.1.13.2 CÓDIGO SAGE_PYTHON

```
@property
def reset_frecuency(self):
        res = []
        for i in range(self.__number_routers):
            aux1 = []
            for j in range(self.__width):
                aux2 = []
                for k in range(self.__high):
                    aux2.append(0)
                aux1.append(aux2)
            res.append(aux1)
        self.__frecuency = res
```

3.5.1.14 MÉTODO: “reset_most_frecuently_positions”

Genera una tabla bidimensional que se indexa por posición en el eje “X” “posx” y posición en el eje “Y” “posy”, esto es, most_frecuently_positions[posx][posy] y cuyo valor asignado es el número de veces que ha sido asignado a dicha posición un router.

El objetivo de esta tabla es el mismo que el de la tabla “frecuency”, pero con la salvedad de que esta tabla requiere de menos complejidad computacional para encontrar las posiciones más frecuentes a donde han sido establecidos los routers, las cuales necesitaremos para llevar a cabo el proceso de diversificación, en el método “diversification”, por el cual moveremos los

routers que menos veces se han movido de sitio a las posiciones más frecuentes.

Si bien “frecuency” es un triple bucle anidado, esto es, complejidad O(n^3), está es un doble bucle anidado con complejidad O(n^2).

La tabla “frecuency” se actualizará en el método de “update_frecuency”, tras llevar a cabo el proceso de intensificación y diversificación.

3.5.1.14.1 PSEUDO CÓDIGO

Resultado <- [ ]

Para_todo_j_en_rango(0, Ancho_cuadrícula – 1)

    Auxiliar1 <- [ ]

    Para_todo_k_en_rango(0, Largo_cuadrícula – 1)

        Auxiliar1 <- +(0)

    Resultado <- +(Auxiliar1)

Devolver Resultado

3.5.1.14.2 CÓDIGO SAGE_PYTHON

```
@property
def reset_most_frecuently_positions(self):
        res = []
        for j in range(self.__width):
            aux1 = []
            for k in range(self.__high):
                aux1.append(0)
            res.append(aux1)
        self.__most_frecuently_positions = res
```

3.5.1.15 MÉTODO: “reset_tfrecuency”

Este método genera una lista con “number_routers” elementos, todos ellos ceros.

Servirá para llevar la cuenta de cuantas veces se ha cambiado de posición cada uno de los routers.

Se utilizará durante el proceso de diversificación, esto es, en el método “Diversification”, ordenando esta lista de forma decreciente y quedándonos con el “diversification_porcentage” de los routers de la lista que menos veces se han movido de sitio, para efectuar un movimiento de “swap” o “moveToCell”

para cada uno de estos routers, moviéndolos a las posiciones donde más movimientos ha habido, esto son, las (posx, posy) que más veces se han visto ocupadas por entradas y salidas de routers.

Se actualizará en el método de “update_frecuency”.

3.5.1.15.1 PSEUDO CÓDIGO

Resultado <- [ ]

Para_todo_i_en_rango(0, Número_routers_solución – 1)

    Resultado <- +0

Devolver Resultado

3.5.1.15.2 CÓDIGO SAGE_PYTHON

```
@property
def reset_tfrecuency(self):
        res = [0]*self.__number_routers
        self.__tfrecuency = res
```

3.5.1.16 MÉTODO: “Admissible”

Este método genera una lista de solución/soluciones admisibles a partir de una solución inicial de la que partimos, “current_solution”, y una lista de movimiento/movimientos a aplicar sobre “current_solution”, generados a partir del método “Movements()”.

Consideramos que una solución es admisible bajo 2 condiciones:

1. Que la solución no sea una solución ya visitada con anterioridad, esto es, que TH[solution.hash() mod “tabu_size”] == False. Si es “True”, la solución queda automáticamente descartada.

2. Si (1.) es “False”, habrá que verificar que el movimiento utilizado para generar la solución, no se trata de un movimiento con estado Tabú. La comprobación de que un movimiento tenga el estado de tabú es definido en una primera comprobación por el método “is_tabu()”, el cual verifica si la tabla tridimensional “TL” tiene un valor asociado distinto de “-1” para TL[r][posx][posy]. Puesto que “-1” codifica que el router “r” en la posición del eje “X” “posx” y la posición del eje “Y” “posy” aún no sido establecido como tabú en ninguna ocasión, lo contrario significará que si ha habido al menos una iteración del algoritmo donde se ha establecido como Tabú, ergo tendremos que pasar a una segunda comprobación que determine si finalmente ha dejado de ser Tabú. Así mismo, el método “is_tabu()” aprovecha para asignar al atributo

"lastIterationAsTabu" del objeto de tipo "Movement" el valor asociado a TL[r][posx][posy], que nos hará falta en la segunda comprobación.

La segunda comprobación es llevada a cabo por el método "aspirates()", el cual deja pasar la solución en uno de estos dos casos:

1. Que la solución tenga un valor de fitness mejor que el valor de fitness que tiene "best_solution_found". Para ello se hace uso del método "is_better_than()".

2. Que la actual iteración del bucle while, "actual_iteration" sea mayor o igual que la suma del atributo "aspiration_value" del actual objeto "TSMeshAlgorithmProblem" más el atributo "lastIterationAsTabu" del objeto de tipo "Movement" que estamos tratando de determinar si aún sigue siendo Tabú o ha dejado de serlo.

La única labor del método "admissible()" es la de filtrar las soluciones, esto es quedarse con unas y descartar otras, en el caso de que sea un vecindario de soluciones por haber aplicado el movimiento de tipo "swap" y se hayan generado todas las combinaciones de intercambios. En caso de ser un tipo de movimiento "moveToCell", sólo se generará una posible nueva solución.

Generaremos 4 listas:

1. **Admissible:** La lista final que se devuelve. Ad(s) = (N(s) – T(s)) U As(s).
2. **Neighbourhood:** Lista que contiene todas las soluciones generadas a partir de la "current_solution" y los diferentes movimientos generados, independientemente de que la nueva solución sea una solución repetida o que la nueva solución se haya generado a partir de un movimiento que es tabú.
3. **Tabú:** Lista que contiene todas aquellas soluciones que son repetidas, y todas aquellas soluciones que si bien no son repetidas, son soluciones generadas a partir de un movimiento que es tabú y que no ha conseguido superar algunos de los criterios de aspiración.
4. **Aspiration:** Lista que contiene todas aquellas soluciones consideradas tabú al haberse generado a partir de un movimiento tabú, pero que sí han conseguido superar alguno de los criterios de aspiración.

Se ha considerado esta diferenciación de soluciones habidas teniendo en mente almacenar la mayor información posible generada por el algoritmo en cada una de las iteraciones, para hacer más fácil el poder seguir la traza del algoritmo. Se tuvo en mente diseñar un tipo que almacenara toda esta información en cada iteración, pero finalmente fue descartado.

Los pasos a seguir por el algoritmo son:

1. Se generan las 4 listas vacías.
2. Se genera la lista de movimientos.
3. Para todo movimiento:
    1. Hacemos una copia de la "current_solution".

2. Generamos la nueva solución a partir de la copia y el movimiento.
3. Introducimos la nueva solución en el conjunto “N”.
4. Si la nueva solución no es una solución visitada:
    1. Si la nueva solución es tabú:
        1. Introducimos la nueva solución en “T”.
        2. Si la nueva solución cumple algún criterio de aspiración:
            1. Introducimos la nueva solución en “As”.
            2. Retiramos de “T” la nueva solución.
    2. Sino, introducimos la nueva solución en “Ad”
5. Sino, intoducimos la nueva solución visitada en “T”.

4. Extendemos “Ad” con “As”. Retornamos “Ad”.

3.5.1.16.1 PSEUDO CÓDIGO

```
Admisible <- [ ]
Vecindario <- [ ]
Tabu <- [ ]
Aspiración <- [ ]
Movimientos <- Generar_movimientos( solución )
Para_todo_movimiento_en_Movimientos
        Copia <- Generar_copia( solución )
        Aplicar_movimiento( Copia, Movimiento )
        Vecindario <- +Copia
        Si ( Not ( Es_visitada ( Copia ) ) )
                Si ( Es_tabu( Movimiento ) )
                        Tabu <- +Copia
                        Si ( Criterio_aspiración ( Copia, Movimiento ) )
                                Aspiración <- + Copia
                                Quitar ( Copia, Tabu )
                Sino
                        Admisible <- +Copia
        Sino
                Tabu <- +Copia
Extender ( Admisible, Aspiración )
Devolver Admisible
```

3.5.1.16.2 CÓDIGO SAGE_PYTHON

```
def Admissible(self, solution):
        Admissible = []
        Neighbourhood = []
        Tabu = []
        Aspiration = []
        for m in self.Movements(solution):
            newSolution = deepcopy(solution)
            self.apply_movement(newSolution, m)
            Neighbourhood.append(newSolution)
            if(not(self.__TH[self.hashing(newSolution) %
                   self.__tabu_size] == True)):
                if(self.is_tabu(newSolution, m)):
                    Tabu.append(newSolution)
                    if(self.aspirates(newSolution, m)):
                        Aspiration.append(newSolution)
                        Tabu.remove(newSolution)
                else:
                    Admissible.append(newSolution)
            else:
                Tabu.append(newSolution)
        Admissible.extend(Aspiration)
        return Admissible
```

3.5.1.17 MÉTODO: “Movements”

Este método es el encargado de generar todos los posibles movimientos, en caso de que el tipo de movimiento sea “swap” o bien un movimiento si se trata de “moveToCell”.

Lo/los mete en una lista y los retorna como lista de objetos de tipo “Movement”.

Previo a crear el objeto de tipo “Movement”, se determina si se va a llevar a cabo un movimiento de tipo “swap” o bien un “moveToCell”, para ello se genera un número aleatorio de entre 2 posibles números, el 0 o el 1, y se asigna el resultado a la variable “movement_type”.

Si finalmente el tipo de movimiento es “swap”, habrá que anidar un bucle doble para generar todas las posibles combinaciones de intercambios entre routers de la solución actual, “current_solution”.

Si por el contrario se trata de un movimiento “moveToCell”, se genera un número aleatorio de entre los “number_routers” – 1, esto es, [0,1,2,3,…,”number_routers” – 1], el cual nos indicará el índice del router que va a ser movido de posición, esto es, “router_to_moveToCell”.

Para la posición en el eje “X” a la cual se va a mover el router, se hace lo mismo, esto es generar otro número aleatorio de entre los “width” – 1, esto es, [0,1,2,3,…,”width” – 1]. Este valor es el que recibe el atributo “posx_to_moveToCell” del nuevo objeto tipo “Movement” a crear.

Para la posición en el eje “Y” a la cual se va a mover el router, se hace lo mismo, esto es generar otro número aleatorio de entre los “high” – 1, esto es, [0,1,2,3,…,”high” – 1]. Este valor es el que recibe el atributo “posy_to_moveToCell” del nuevo objeto tipo “Movement” a crear.

Con toda esta información, se genera el objeto de tipo “Movement”, y se mete dentro de la lista a retornar.

3.5.1.17.1 PSEUDO CÓDIGO

Movimientos <- [ ]

Número <- Generar_aleatorio(0, 1)

Si(Número = 0)

    Tipo_movimiento <- swap

Sino

    Tipo_movimiento <- moveToCell

Si (Tipo_movimiento = swap)

    Para_todo_i_en_rango(0, Numero_routers_solucion – 1)

        Para_todo_j_en_rango(i+1, Numero_routers_solucion – 1)

            Movimientos <- +Generar_movimiento(swap, i, j)

Sino

    i <- Generar_aleatorio(0, …, Numero_routers_solucion – 1)

    Posx <- Generar_aleatorio(0, …, Ancho_cuadrícula – 1)

    Posy <- Generar_aleatorio(0, …, Largo_cuadrícula – 1)

    Movimientos <- +Generar_movimiento(moveToCell, i, Posx, Posy)

Devolver Movimientos

3.5.1.17.2 CÓDIGO SAGE_PYTHON

```
def Movements(self, solution):
        Movements = []
        n = randint(0, 1)
        if(n==0):
            self.__movement_type = "swap"
        else:
            self.__movement_type = "moveToCell"
        if(self.__movement_type == "swap"):
            for i in range(0, len(solution.routers)):
                for j in range(i+1, len(solution.routers)):
```

```
                Movements.append(
                        movement("swap",None,None,None,i,j)
                                )
        else:
            if(self.__movement_type == "moveToCell"):
                i = randint(0, self.__number_routers-1)
                posx = randint(0, self.__width-1)
                posy = randint(0, self.__high-1)
                Movements.append(
                        movement("moveToCell",i,posx,posy,None,None))
        return Movements
```

3.5.1.18 MÉTODO: "apply_movement"

Este método recibe por entrada una copia de la solución actual, "current_solution", y tiene por finalidad modificar dicha copia a través del método "modify" que tiene el tipo "Solution" para obtener la nueva solución.

También recibe por entrada el objeto de tipo "Movement" con la información necesaria para llevar a cabo los cambios pertienentes que nos permita generar la nueva solución a partir de la modificación de la copia de la "current_solution".

Si el movimiento a aplicar es un movimiento de tipo "swap", tendremos que ayudarnos de una copia de la copia de la solución, puesto que si vamos a modificar los atributos "posx" y "posy" del "router1" dándole los valores "posx" y "posy" del router2, posteriormente cuando le queramos dar a "router2" los valores "posx" y "posy" de router1 no lo podremos hacer porque ya los habremos modificado previamente, de ahí el uso de una copia de la copia.

A través del método "modify" del tipo "Solution", también asignaremos a la nueva solución su "precededSolution", que será la copia de la copia, y el "appliedMovement", que será el objeto de tipo "Movement" que recibe el método por entrada.

Una vez modificada la copia, se llamará al método encargado de computar su valor de fitness, puesto que estamos ante una nueva solución que difiere de su predecesora, y ello implica que su valor de fitness ha cambiado.

Tras esto, se retornará la copia modificada y debidamente evaluada en su función de fitness.

3.5.1.18.1 PSEUDO CÓDIGO

Copia <- Generar_copia(Solucion)

Si(Tipo_movimiento = swap)

    i <- Movement.index_router_1

    j <- Movement.index_router_2

Solucion.router[i].Posx <- Copia.router[j].Posx

Solucion.router[i].Posy <- Copia.router[j].Posy

Solucion.router[j].Posx <- Copia.router[i].Posx

Solucion.router[j].Posx <- Copia.router[i].Posy

Solucion.precededSolution <- Copia

Solucion.appliedMovement <- Movement

Sino

i <- Movement.index_router

Solucion.router[i].Posx <- Movement.Posx

Solucion.router[i].Posy <- Movement.Posy

Solucion.precededSolution <- Copia

Solucion.appliedMovement <- Movement

Calcular_fitness(Solucion)

3.5.1.18.2 CÓDIGO SAGE_PYTHON

```
def apply_movement(self, solution, movement):

        copy_solution = deepcopy(solution)

        if(movement.type == "swap"):

            solution.modify(
            movement.router1_to_swap,
            copy_solution.routers[movement.router2_to_swap].posx,
            copy_solution.routers[movement.router2_to_swap].posy)

            solution.modify(
            movement.router2_to_swap,
            copy_solution.routers[movement.router1_to_swap].posx,
            copy_solution.routers[movement.router1_to_swap].posy,
            copy_solution,
            movement)

        else:

            if(movement.type == "moveToCell"):

                solution.modify(movement.router_to_moveToCell,
                                movement.posx_to_moveToCell,
                                movement.posy_to_moveToCell,
                                copy_solution,
                                movement)

        self.compute_fitness_value(solution)
```

3.5.1.19 MÉTODO: “hashing”

Este método es el encargado de generar el hash de una solución. Para ello empleará los números aleatorios contenidos en la tabla tridimensional “hashing”.

El método recibe por entrada un objeto de tipo “Solution”, esto es la solución para la cual queremos generar su hash correspondiente.

Su funcionamiento es simple, para cada uno de los routers de la solución que recibe el método por parámetro de entrada, se hace uso del índice del router, de su posición en el eje “X” y de su posición en el eje “Y” para indexar en la tabla “hashing” y obtener el número aleatorio que se haya en dicha posición de la tabla.

Todos esos números se suman y se genera un nuevo número, el “hash”, que es el valor que finalmente retorna el método y que trata de identificar unívocamente a una solución dada.

3.5.1.19.1 PSEUDO CÓDIGO

Hash <- 0

Para_todo_i_en_rango(0, Número_routers_solucion – 1)

j <- solución.router[i].Posx

k <- solución.router[i].Posy

Hash <- +Tabla_hash[i][j][k]

Devolver Hash

3.5.1.19.2 CÓDIGO SAGE_PYTHON

```
def hashing(self, solution):
        hash = 0
        for i in range(0, len(solution.routers)):
            hash += self.__hashing[i]
                                  [solution.routers[i].posx]
                                  [solution.routers[i].posy]
        return hash
```

3.5.1.20 MÉTODO: “is_tabu”

Recibe por parámetro de entrada un objeto de tipo “Solution” y un objeto de tipo “Movement”.

Este método lleva a cabo la primera comprobación del estado Tabú de la nueva solución a través del estudio de las características que presenta el movimiento a partir del cual se ha generado la nueva solución que tratamos de determinar si es o no admisible.

Esta primera comprobación se limita a indexar la tabla tridimensional “TL” para el router “r”, la posición en el eje “X” “posx” y la posición en el eje “Y” “posy” que indica el objeto de tipo “Movement”. Esto es, TL[r][posx][posy] y determinar si dicho valor es igual a “-1” o distinto de “-1”.

El valor “-1” codifica que el router “r” en la posición del eje “X” “posx” y la posición del eje “Y” “posy” aún no sido establecido como tabú en ninguna ocasión. De darse dicho caso, se retornaría Falso y por tanto no hará falta llevar a cabo la segunda comprobación a través del método “aspirates()”.

Un valor distinto de “-1” significa que si ha habido al menos una iteración del algoritmo donde se ha establecido como Tabú. De darse dicho caso, se retornaría True y por tanto hará falta llevar a cabo la segunda comprobación a través del método “aspirates()” para determinar si finalmente se deja pasar la solución como una solución admisible.

Aprovecharemos la indexación en la tabla “TL” para asignar el valor TL[r][posx][posy] al atributo “lastIterationAsTabu” del objeto de tipo “Movement”, ya que dicha información nos hará falta en caso de que el valor sea distinto de “-1” y por tanto tengamos que llevar a cabo la segunda comprobación.

3.5.1.20.1 PSEUDO CÓDIGO

```
Resultado <- False
Si( Tipo_movimiento = swap)
        i <- Movement.index_router_1
        Posx <- solución.router[ i ].Posx
        Posy <- solución.router[ i ].Posy
        UltimaIteraciónTabu[ 0 ] <- TL[ i ][ Posx ][ Posy ]

        j <- Movement.index_router_2
        Posx <- solución.router[ j ].Posx
        Posy <- solución.router[ j ].Posy
        UltimaIteraciónTabu[ 1 ] <- TL[ j ][ Posx ][ Posy ]
        Movement.UltimaIteraciónTabu <- UltimaIteraciónTabu
        Resultado <- UltimaIteraciónTabu[0] > -1  |  UltimaIteraciónTabu[1] > -1
```

Sino

i <- Movement.index_router

Posx <- Movement.Posx

Posy <- Movement.Posy

UltimaIteraciónTabu <- TL[i][Posx][Posy]

Movement.UltimaIteraciónTabu <- UltimaIteraciónTabu

Resultado <- (UltimaIteraciónTabu > -1)

Devolver Resultado

3.5.1.20.2 CÓDIGO SAGE_PYTHON

```
def is_tabu(self, solution, movement):
        res = False
        if(movement.type == "swap"):

            LastIterationAsTabu =

            [self.__TL[movement.router1_to_swap]
                      [solution.routers[movement.router1_to_swap].posx]
                      [solution.routers[movement.router1_to_swap].posy]
            ,
             self.__TL[movement.router2_to_swap]
                      [solution.routers[movement.router2_to_swap].posx]
                      [solution.routers[movement.router2_to_swap].posy]
            ]

            movement.setLastIterationAsTabu(LastIterationAsTabu)
            res = LastIterationAsTabu[0] > (-1)
                  or
                  LastIterationAsTabu[1] > (-1)
        else:
            if(movement.type == "moveToCell"):

                LastIterationAsTabu =

                (self.__TL[movement.router_to_moveToCell]
                          [movement.posx_to_moveToCell]
                          [movement.posy_to_moveToCell])

                movement.setLastIterationAsTabu(LastIterationAsTabu)
                res = LastIterationAsTabu > (-1)
        return res
```

3.5.1.21 MÉTODO: “aspirates”

Recibe por parámetro de entrada un objeto de tipo “Solution” y un objeto de tipo “Movement”.

Este método precede a la primera comprobación llevada a cabo a través del método “is_tabu()” que determinó si existía en la tabla tridimensional “TL” alguna iteración por la cual se establecía la condición Tabú del movimiento aplicado para generar la nueva solución que tratamos de determinar si está o no en estado Tabú.

Lleva a cabo la segunda comprobación por la cual se determina si pasa finalmente a ser una solución admisible, o bien se descarta por ser Tabú.

La nueva solución será finalmente admisible si bien mejora en valor de Fitness a “best_solution_found”, o bien que la actual iteración del bucle while, “actual_iteration” sea mayor o igual que la suma del atributo “aspiration_value” del actual objeto “TSMeshAlgorithmProblem” más el atributo “lastIterationAsTabu” del objeto de tipo “Movement”.

3.5.1.21.1 PSEUDO CÓDIGO

```
Resultado <- False
Si( Es_Mejor( solución, Mejor_solución_encontrada) )
        Resultado <- True
Sino
        Si( Tipo_movimiento == swap)
                a1 <- Movimiento.UltimaIteraciónTabu[ 0 ]
                a2 <- Movimiento.UltimaIteraciónTabu[ 1 ]
                b <- Valor_aspiración
                c <- Iteración_actual
                Resultado <- ( Máximo( a1, a2 ) + b <= c )
        Sino
                a <- Movimiento.UltimaIteraciónTabu
                b <- Valor_aspiración
                c <- Iteración_actual
                Resultado <- ( a + b <= c )
Devolver Resultado
```

3.5.1.21.2 CÓDIGO SAGE_PYTHON

```
def aspirates(self, solution, movement):
        res = False
        if(self.is_better_than(solution, self.__best_solution_found)):
            res = True
        else:
            if(movement.type == "swap"):
                res = max(movement.LastIterationAsTabu[0],
                          movement.LastIterationAsTabu[1]) +
                      self.__aspiration_value <=
                      self.__actual_iteration
            else:
                if(movement.type == "moveToCell"):
                    res = movement.LastIterationAsTabu +
                          self.__aspiration_value <=
                          self.__actual_iteration
        return res
```

3.5.1.22 MÉTODO: "is_better_than"

Recibe por parámetro de entrada dos objetos de tipo "Solution": "solution1" y "solution2", y trata de determinar si "solution1" tiene mejor valor de fitness que "solution2".

Para ello, compara en primer lugar el atributo del tamaño de la máxima componente conexa y desempata, en caso de iguales, por el atributo del número de clientes cubiertos. Se da máxima prioridad a fitness 1.

3.5.1.22.1 PSEUDO CÓDIGO

Resultado <- False

Sol_1_fitness_1 <- solución_1.MayorComponenteConexa

Sol_2_fitness_1 <- solución_2.MayorComponenteConexa

Si(Sol_1_fitness_1 > Sol_2_fitness_1)

    Resultado <- True

Sino

    Si(Sol_1_fitness_1 == Sol_2_fitness_1)

        Sol_1_fitness_2 <- solución_1.NúmeroClientesCubiertos

        Sol_2_fitness_2 <- solución_2.NúmeroClientesCubiertos

        Si(Sol_1_fitness_2 > Sol_2_fitness_2)

            Resultado <- True

Devolver Resultado

3.5.1.22.2 CÓDIGO SAGE_PYTHON

```
def is_better_than(self, solution1, solution2):
        res = False
        if(solution1.lengthGreatestConnectedSet >
           solution2.lengthGreatestConnectedSet):
            res = True
        else:
            if(solution1.lengthGreatestConnectedSet ==
               solution2.lengthGreatestConnectedSet):
                if(solution1.numberClientsCovered >
                   solution2.numberClientsCovered):
                    res = True
        return res
```

3.5.1.23 MÉTODO: “update_recency”

Este método actualiza la memoria basada en la recencia, esto es, actualiza la tabla tridimensional “TL”, así como la tabla unidimensional “hashing”.

Si la “current_solution” ha cambiado, esto es, el hash de la “current_solution” es distinto del hash de la “previous_current_solution”, significa que definitivamente se ha aplicado un movimiento que ha dado lugar a una solución diferente, ergo habrá que actualizar la tabla tridimensional “TL” haciendo Tabú el movimiento aplicado, esto es, asignarle el valor de la actual iteración.

La nueva solución habrá que marcarla como visitada, esto es, indexar en la tabla unidimensional “hashing” con el hash de la solución en módulo “tabu_size”, y asignar el valor booleano “True”.

3.5.1.23.1 PSEUDO CÓDIGO

Solución_hash <- hashing(solución)

Tabú_hash[Solución_hash % Tamaño_lista_tabú] <- True

Si (Tipo_movimiento == swap)

    i <- solución.MovimientoAplicado.router_1

    solución_precedida <- solución.SoluciónPrecedida

    j <- solución_precedida.router[i].posx

    k <- solución_precedida.router[i].posy

    Lista_tabú[i][j][k] <- Iteración_actual

i <- solución.MovimientoAplicado.router_2

solución_precedida <- solución.SoluciónPrecedida

j <- solución_precedida.router[i].posx

k <- solución_precedida.router[i].posy

Lista_tabú[i][j][k] <- Iteración_actual

Sino

i <- solución.MovimientoAplicado.router

solución_precedida <- solución.SoluciónPrecedida

j <- solución_precedida.router[i].posx

k <- solución_precedida.router[i].posy

Lista_tabú[i][j][k] <- Iteración_actual

3.5.1.23.2 CÓDIGO SAGE_PYTHON

```
def update_recency(self, solution):

self.__TH[self.hashing(solution) % self.__tabu_size] == True

if(self.hashing(self.__previous_current_solution) !=
                                          self.hashing(solution)):

            if(movement.type == "swap"):

                  (self.__TL[solution.appliedMovement.router1_to_swap]
                            [solution.precededSolution.routers[
                              solution.appliedMovement.router1_to_swap
                                                            ].posx
                            ]
                            [solution.precededSolution.routers[
                              solution.appliedMovement.router1_to_swap
                                                            ].posy
                            ]) = self.__actual_iteration

                  (self.__TL[solution.appliedMovement.router2_to_swap]
                            [solution.precededSolution.routers[
                              solution.appliedMovement.router2_to_swap
                                                            ].posx
                            ]
                            [solution.precededSolution.routers[
                              solution.appliedMovement.router2_to_swap
                                                            ].posy
                            ]) = self.__actual_iteration
```

```
        else:

            (self.__TL
                  [solution.appliedMovement.router_to_moveToCell]
                  [solution.precededSolution.routers[
                    solution.appliedMovement.router_to_moveToCell
                                                    ].posx
                  ]
                  [solution.precededSolution.routers[
                    solution.appliedMovement.router_to_moveToCell
                                                    ].posy
                  ]) = self.__actual_iteration
```

3.5.1.24 MÉTODO: “update_frecuency”

Recibe por parámetro de entrada un objeto de tipo “Solution”, que es la nueva “current_solution” con toda la información necesaria para actualizar las tablas de frecuencia, esto es, la solución predecesora y el movimiento aplicado sobre ésta para generar la solución actual.

Este método actualiza la memoria basada en la frecuencia, esto es, actualiza la tabla unidimensional “tfrecuency”, la tridimensional “frecuency” y la bidimensional “most_frecuently_positions”. Así mismo, actualiza la lista unidimensional “best_sols” que mantiene las “elite_size” mejores soluciones encontradas hasta el momento.

Si la “current_solution” ha cambiado, esto es, el hash de la “current_solution” es distinto del hash de la “previous_current_solution”, significa que definitivamente se ha aplicado un movimiento que ha dado lugar a una solución diferente, ergo habrá que actualizar la tabla unidimensional “tfrecuency” incrementando en una unidad para aquel router que haya cambiado de posición debido al movimiento aplicado sobre la solución original, y que ha dado lugar a esta nueva solución.

También incrementamos en una unidad frecuency[r][posx][posy], esto es, el router “r” que ha cambiado de posición, estableciéndose ahora en la (posx, posy).

Lo mismo para “most_frecuently_positions”, incrementando en una unidad most_frecuently_positions[posx][posy]

Si la lista unidimensional “best_sols” no está llena, esto es, len(best_sols) < “elite_size”, introducimos la nueva “current_solution”.

Si la lista unidimensional “best_sols” está llena, entonces tendremos que recorrerla hasta encontrar la solución menos buena, y compararla con la nueva “current_solution”. Si la nueva “current_solution” tiene mejor valor de fitness que la menos buena, se sustituye.

3.5.1.24.1 PSEUDO CÓDIGO

```
SI ( Tipo_movimiento == swap )
        i <- solución.MovimientoAplicado.router_1
        j <- solución.MovimientoAplicado.router_2
        Posx <- solución.soluciónPrecedida.router[ j ].Posx
        Posy <- solución.soluciónPrecedida.router[ j ].Posy

        T_frecuencia[ i ] <- +1
        Frecuencia[ i ][ Posx ][ Posy ] <- +1
        Posiciones_más_frecuentes[ Posx ][ Posy ] <- +1

        i <- solución.MovimientoAplicado.router_2
        j <- solución.MovimientoAplicado.router_1
        Posx <- solución.soluciónPrecedida.router[ j ].Posx
        Posy <- solución.soluciónPrecedida.router[ j ].Posy

        T_frecuencia[ i ] <- +1
        Frecuencia[ i ][ Posx ][ Posy ] <- +1
        Posiciones_más_frecuentes[ Posx ][ Posy ] <- +1

Sino
        i <- solución.MovimientoAplicado.router
        Posx <- solución.MovimientoAplicado.Posx
        Posy <- solución.MovimientoAplicado.Posy

        T_frecuencia[ i ] <- +1
        Frecuencia[ i ][ Posx ][ Posy ] <- +1
        Posiciones_más_frecuentes[ Posx ][ Posy ] <- +1
```

Si (Longitud_lista_mejores_soluciones < Longitud_elite)

Lista_mejores_soluciones <- +(solución)

Sino

Peor_solución <- Lista_mejores_soluciones[0]

Para_toda_solución_en_Lista_mejores_soluciones

Si(Not (Es_mejor(solución, Peor_solución)))

Peor_solución <- solución

Si(Es_mejor(solución_actual, Peor_solución))

Quitar(Peor_solución, Lista_mejores_soluciones)

Añadir(solución_actual, Lista_mejores_soluciones)

3.5.1.24.2 CÓDIGO SAGE_PYTHON

```
def update_frecuency(self, solution):

if(self.hashing(self.__previous_current_solution) !=
                                        self.hashing(solution)):
      if(solution.appliedMovement.type == "swap"):

            self.__tfrecuency
                  [solution.appliedMovement.router1_to_swap] += 1

            self.__tfrecuency
                  [solution.appliedMovement.router2_to_swap] += 1

            (self.__frecuency
                  [solution.appliedMovement.router1_to_swap]
                  [solution.precededSolution.routers[
                              solution.appliedMovement.router2_to_swap
                                                    ].posx
                  ]
                  [solution.precededSolution.routers[
                              solution.appliedMovement.router2_to_swap
                                                    ].posy
                  ]) += 1

            (self.__frecuency
                  [solution.appliedMovement.router2_to_swap]
                  [solution.precededSolution.routers[
                              solution.appliedMovement.router1_to_swap
                                                    ].posx
                  ]
                  [solution.precededSolution.routers[
                              solution.appliedMovement.router1_to_swap
                                                    ].posy
                  ]) += 1
```

```
            (self.__most_frecuently_positions
                  [solution.precededSolution.routers[
                              solution.appliedMovement.router2_to_swap
                                                ].posx
                  ]
                  [solution.precededSolution.routers[
                              solution.appliedMovement.router2_to_swap
                                                ].posy
                  ]) += 1

            (self.__most_frecuently_positions
                  [solution.precededSolution.routers[
                              solution.appliedMovement.router1_to_swap
                                                ].posx
                  ]
                  [solution.precededSolution.routers[
                              solution.appliedMovement.router1_to_swap
                                                ].posy
                  ]) += 1

      else:

            self.__tfrecuency[
                        solution.appliedMovement.router_to_moveToCell
                              ] += 1

            (self.__frecuency[
                        solution.appliedMovement.router_to_moveToCell
                              ]
                              [
                        solution.appliedMovement.posx_to_moveToCell
                              ]
                              [
                        solution.appliedMovement.posy_to_moveToCell
                              ]) += 1

            (self.__most_frecuently_positions[
                        solution.appliedMovement.posx_to_moveToCell
                                          ]
                                          [
                        solution.appliedMovement.posy_to_moveToCell
                                          ]) += 1

if(len(self.__best_sols) < self.__elite_size):
      self.__best_sols.append(solution)
else:
      min = self.__best_sols[0]
      for i in range(0, len(self.__best_sols)):
            if(not(self.is_better_than(self.__best_sols[i], min))):
                  min = self.__best_sols[i]
      if(self.is_better_than(solution, min)):
            self.__best_sols.remove(min)
            self.__best_sols.append(solution)
```

3.5.1.25 MÉTODO: “Intensification”

Este método recibe por parámetro de entrada un objeto de tipo “Solution”, esto es, la “current_solution”. Implementa un doble bucle anidado, con el externo se fija el índice del router, y con el interno nos vamos moviendo a través de las diferentes soluciones de la lista unidimensional “best_sols”, que contiene las mejores soluciones encontradas hasta el momento.

Fijado el router i-ésimo, en cada solución de la lista “best_sols” presumiblemente ocupará una posición distinta, ergo para cada una de estas posiciones calculamos el número de veces que el router i-ésimo se ha establecido en dichas posiciones a través de la indexación de “frecuency”, y todos estos números los sumamos en un sumatorio, esto es, “sum”.

Al mismo tiempo que vamos calculando “sum” iremos rellenando una lista de listas llamada “ri”, que contendrá para el router i-ésimo una lista por cada solución de las “best_sols”, esto es, una lista de “elite_size” listas donde cada una de esas listas contiene 5 parámetros: índice del router i-ésimo, índice de la solución j-ésima, posición que ocupa el router “i” en el eje “X” para la solución “j”, posición que ocupa el router “i” en el eje “Y” para la solución “j” y el número de veces que el router “i” se ha establecido en la posición que ocupa el router “i” en la solución “j”.

Una vez hallamos recorrido todas las soluciones, esto es, hayamos salido del bucle interno, ordenamos la lista de listas “ri” estableciendo como orden el 5 parámetro de forma inversa, de manera que al principio de la lista nos queden aquellas listas donde su quinto parámetro es más grande y llamamos al método de la ruleta, esto es, “roulette_wheel_criteria” el cual seleccionará una nueva posición para el router i-ésimo de la solución, basándose en las posiciones que ha ocupado el router i-ésimo en las mejores soluciones encontradas hasta el momento.

3.5.1.25.1 PSEUDO CÓDIGO

```
i <- 0
Mientras( i < Número_routers_solución – 1 )
        j <- 0
        Suma <- 0
        Res <- [ ]
        Mientras( j < Longitud_lista_mejores_soluciones – 1 )
                Freq <- frecuencia_router_i_en_Posx_Posy_de_solución_j
                Suma <- +Freq
                Posx <- Posx_router_i_en_solución_j
                Posy <- Posy_router_i_en_solución_j
```

Res <- [i, j, Posx, Posy, Freq]

j <- +1

Ordenar.(Res, Parámetro[4], Al_reves)

Si(Not(Vacío(Res) | Suma == 0))

Método_ruleta(solución, Res, Suma)

i <- +1

3.5.1.25.2 CÓDIGO SAGE_PYTHON

```
def Intensification(self, solution):
        i = 0
        while(i < self.__number_routers):
            sum = 0
            ri = []
            for j in range(0, len(self.__best_sols)):
                frecuencia =
                self.__frecuency[i]
                                [self.__best_sols[j].routers[i].posx]
                                [self.__best_sols[j].routers[i].posy]
                sum += frecuencia
                ri.append([i,
                           j,
                           self.__best_sols[j].routers[i].posx,
                           self.__best_sols[j].routers[i].posy,
                           frecuencia])
            ri.sort(key=lambda x: x[4], reverse=True)
            if(not(len(ri)==0 or sum==0)):
                self.roulette_wheel_criteria(solution, ri, sum)
            i += 1
```

3.5.1.26 MÉTODO: “roulette_wheel_criteria”

Recibe por parámetro de entrada un objeto de tipo “Solution”, una lista de listas con un máximo de “elite_size” listas y un entero producto de un sumatorio calculado en el método “Intensification”.

Este método implementa la selección por ruleta. De lo que se trata es de seleccionar una nueva posición para el router i-ésimo, y la nueva posición vendrá dada de entre las posiciones que ocupó el router i-ésimo en las mejores soluciones encontradas hasta el momento, esto es, “best_sols”.

Una vez tengamos la nueva posición, crearemos el objeto de tipo “Movement” que contendrá el índice del router a cambiar de posición y la posición a la cual lo vamos a mover, y llamamos al método “apply_movement” para que modifique la solución cambiando dicho router de sitio.

El funcionamiento es el siguiente, se genera un número aleatorio entre el “0” y el “1”, y se declara una variable que llevará la acumulación de la suma a “0”.

Recorremos un bucle bajo un índice que va desde 0 hasta la longitud de la lista de listas, que tendrá una longitud máxima de “elite_size” listas. En cada iteración del bucle actualizamos la variable que lleva la acumulación de la suma hasta el momento, sumándole a lo que ya tiene el quinto parámetro de ri[j], esto es, ri[j][4] el cual contiene el número de veces que el router “i” se ha establecido en la posición que ocupa el router “i” en la solución “j”. Teniendo en cuenta que “ri” se pasó por parámetro de entrada ya ordenada utilizando por orden este quinto parámetro de forma inversa, los primeros valores que encontremos serán los más grandes.

Una vez hemos actualizado la variable acumulativa de la suma, comparamos ésta con el número generado aleatoriamente.

Si es mayor, hemos terminado. Generamos el objeto de tipo “Movement” y llamamos al método “apply_movement”. Si no es mayor, volvemos a iterar.

3.5.1.26.1 PSEUDO CÓDIGO

Número <- Generar_número_aleatorio(0, 1)

Suma_acumulativa <- 0

Para_todo_elemento_en_Res

    Suma_acumulativa <- +(elemento[4])/Suma #Proporción

    Si (Suma_acumulativa > Número)

        Router <- elemento[0]

        Posx <- elemento[2]

        Posy <- elemento[3]

        move <- Movimiento(moveToCell, Router, Posx, Posy)

        Break

Aplicar_movimiento(solución, move)

3.5.1.26.2 CÓDIGO SAGE_PYTHON

```
def roulette_wheel_criteria(self, solution, ri, sum):
        random = uniform(0, 1)
        acumulative_sum = 0
        for i in range(0, len(ri)):
            acumulative_sum += ri[i][4]/sum
            if(acumulative_sum > random):
                move = movement("moveToCell", ri[i][0], ri[i][2],
                                ri[i][3], None, None)
                break
        sys.setrecursionlimit(10000)
        self.apply_movement(solution, move)
```

3.5.1.27 MÉTODO: “Diversification”

Este método recibe por parámetro de entrada un objeto de tipo “Solution”, esto es, la “current_solution”.

El primer paso que lleva a cabo es ordenar la lista unidimensional “tfrecuency”, la cual lleva un recuento del número de veces que cada router ha cambiado de posición.

De esta manera, los objetos tipo “router” del principio de la lista serán aquellos que menos se han movido de sitio durante la ejecución del algoritmo TS.

El segundo paso es llevar a cabo un “aplana” sobre la lista bidimensional “most_frecuently_positions”, a través de un doble bucle anidado, quedándonos con una lista de listas, donde cada lista interna tiene 3 parámetros: [número de veces que un router i-ésimo se ha establecido en (posx, posy), posx, posy].

Posteriormente, se ordena de manera inversa esta lista unidimensional, tomando como orden el primer parámetro de la lista, de tal forma que nos quede al principio de la lista aquellas posiciones (posx, posy) donde más movimiento de routers ha habido.

El siguiente paso es un bucle de “diversification_percentage” x ”number_routers” iteraciones, utilizando como índices en el bucle:

[0, 1, …, “diversification_percentage” x ”number_routers”]

A continuación se determina si algún router de la “current_solution” ocupa una posición del subconjunto de posiciones del conjunto total de posiciones de “most_frecuently_positions” que va a ser utilizado para establecer los routers que menos se han movido a las posiciones de dicho subconjunto.

En caso afirmativo, lo que llevaríamos a cabo sería un movimiento de tipo “swap”.

En caso negativo, lo que llevaríamos a cabo sería un movimiento de tipo “moveToCell”.

Por último, se construye el objeto de tipo “Movement” y se llama al método “apply_movement” para que modifique la “current_solution”, llevando acabo la diversificación.

3.5.1.27.1 PSEUDO CÓDIGO

```
Ordenada_tabla_tiempo_frecuencia <-
        <- Ordenar.( [i, Tabla_tiempo_frecuencia[ i ] ], Parámetro[ 1 ] )
Aplana <- [ ]
Para_todo_Posx_en_Tabla_posiciones_más_visitadas
    Para_todo_Posy_en_Tabla_posiciones_más_visitadas
        Frequencia <- Tabla_posiciones_más_visitadas[ Posx ][ Posy ]
        Aplana <- +[ Frequencia, Posx, Posy ]
Ordenada_tabla_Pos_más_visitadas <-
            <- Ordenar.( Aplana, Parametro[ 0 ], Al_Reves )
Número <- Porcentaje_diversificación * Número_routers_solución
Para_todo_i_en_rango( 0, Número – 1 )
    Router_2_swap <- 0 #Para swap
    Posición_está_ocupada <- False
    Para_todo_router_en_solución_actual:
        Posx<-Posx_en_entrada_i_de_Ordenada_Tabla_Pos_más_visitadas
        Posy<-Posy_en_entrada_i_de_Ordenada_Tabla_Pos_más_visitadas
        Si ( router.Posx == Posx  && router.Posy == Posy)
            Posición_está_ocupada <- True
            Break
        Sino
            Router_2_swap <- +1
    Si( Posición_está_ocupada == True )
        Router_1_swap <- Ordenada_tabla_tiempo_frecuencia[ i ][ 0]
        move <- Movimiento( swap, Router_1_swap, Router_2_swap )
    Sino
        Router <- Ordenada_tabla_tiempo_frecuencia[ i ][ 0]
        Posx<-Posx_en_entrada_i_de_Ordenada_Tabla_Pos_más_visitadas
        Posy<-Posy_en_entrada_i_de_Ordenada_Tabla_Pos_más_visitadas
        move <- Movimiento( moveToCell, Router, Posx, Posy )
    Aplicar_movimiento( solución, move )
```

3.5.1.27.2 CÓDIGO SAGE_PYTHON

```
def Diversification(self, solution):

sorted_tfrecuency =
      sorted([[i, self.__tfrecuency[i]] for i in range(
                                          0, len(self.__tfrecuency))],
                                          key=lambda x: x[1])

most_frecuently_positions = []

for i in range(0, len(self.__most_frecuently_positions)):
      for j in range(0, len(self.__most_frecuently_positions[i])):
            most_frecuently_positions.append(
                        [self.__most_frecuently_positions[i][j],i,j])

sorted_most_frecuently_positions =
      sorted(most_frecuently_positions,key=lambda x: x[0],
                                                      reverse=True)

n = int(self.__diversification_percentage * self.__number_routers)

for i in range(0, n):
      res = False
      j = 0
      for router in solution.routers:
            if(router.posx == sorted_most_frecuently_positions[i][1]
               and
               router.posy == sorted_most_frecuently_positions[i][2]
              ):
                  res = True
                  break
            else:
                  j += 1
      if(res == True):
            move = movement("swap",None,None,None,
                            sorted_tfrecuency[i][0],j)
      else:
            move = movement("moveToCell",sorted_tfrecuency[i][0],
                            sorted_most_frecuently_positions[i][1],
                            sorted_most_frecuently_positions[i][2],
                            None,None)
      self.apply_movement(solution, move)
```

4 EJEMPLOS

4.1 EJEMPLO 1

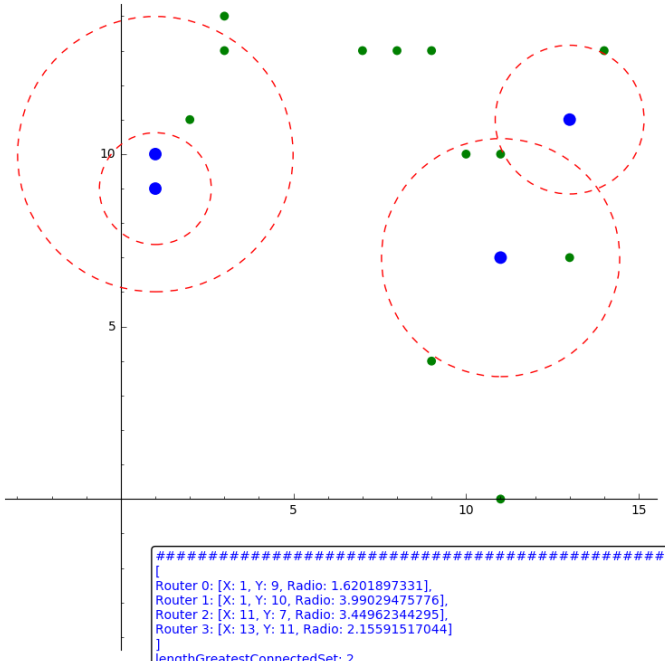
```
def instantiateTSMeshAlgorithmProblem():

      instance = TSMeshAlgorithmProblem(16,16,4,12,64 ,"swap",51113,
                                        15,2^(32),10,"Uniform",0.25,
                                        [4, 1])

      instance.executeTSMeshAlgorithm
```

4.1.1 SOLUCIÓN INICIAL

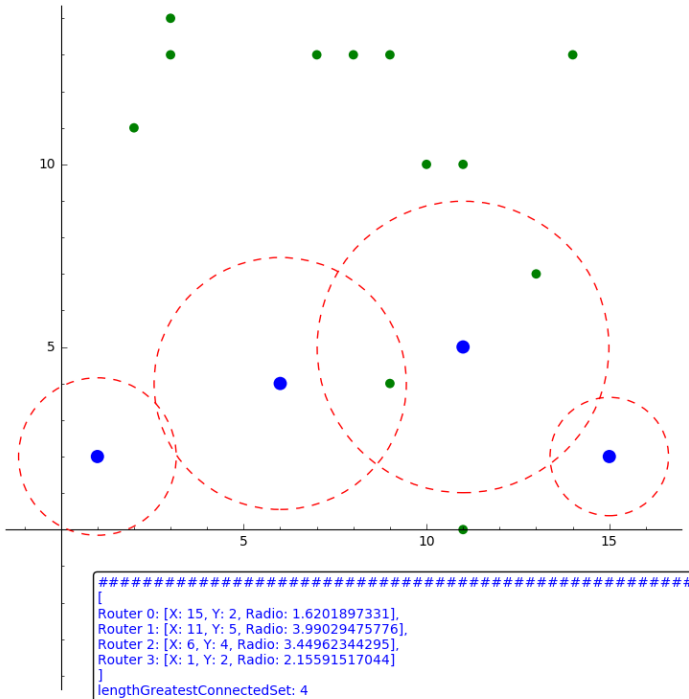
```
################################################################################
######################
[
Router 0: [X: 1, Y: 9, Radio: 1.6201897331],
Router 1: [X: 1, Y: 10, Radio: 3.99029475776],
Router 2: [X: 11, Y: 7, Radio: 3.44962344295],
Router 3: [X: 13, Y: 11, Radio: 2.15591517044]
]
lengthGreatestConnectedSet: 2
numberClientsCovered: 5
################################################################################
######################
```




```
################################################################################
[
Router 0: [X: 1, Y: 9, Radio: 1.6201897331],
Router 1: [X: 1, Y: 10, Radio: 3.99029475776],
Router 2: [X: 11, Y: 7, Radio: 3.44962344295],
Router 3: [X: 13, Y: 11, Radio: 2.15591517044]
]
lengthGreatestConnectedSet: 2
numberClientsCovered: 5
################################################################################
lengthGreatestConnectedSet: 2/4
numberClientsCovered: 5/12
```

4.1.2 SOLUCIÓN FINAL

```
##########################################################################
######################
[
Router 0: [X: 15, Y: 2, Radio: 1.6201897331],
Router 1: [X: 11, Y: 5, Radio: 3.99029475776],
Router 2: [X: 6, Y: 4, Radio: 3.44962344295],
Router 3: [X: 1, Y: 2, Radio: 2.15591517044]
]
lengthGreatestConnectedSet: 4
numberClientsCovered: 2
appliedMovement: [type: swap, 0<->2]
##########################################################################
######################
```




```
##########################################################################################
[
Router 0: [X: 15, Y: 2, Radio: 1.6201897331],
Router 1: [X: 11, Y: 5, Radio: 3.99029475776],
Router 2: [X: 6, Y: 4, Radio: 3.44962344295],
Router 3: [X: 1, Y: 2, Radio: 2.15591517044]
]
lengthGreatestConnectedSet: 4
numberClientsCovered: 2
appliedMovement: [type: swap, 0<->2]
##########################################################################################
lengthGreatestConnectedSet: 4/4
numberClientsCovered: 2/12
```

4.2 EJEMPLO 2

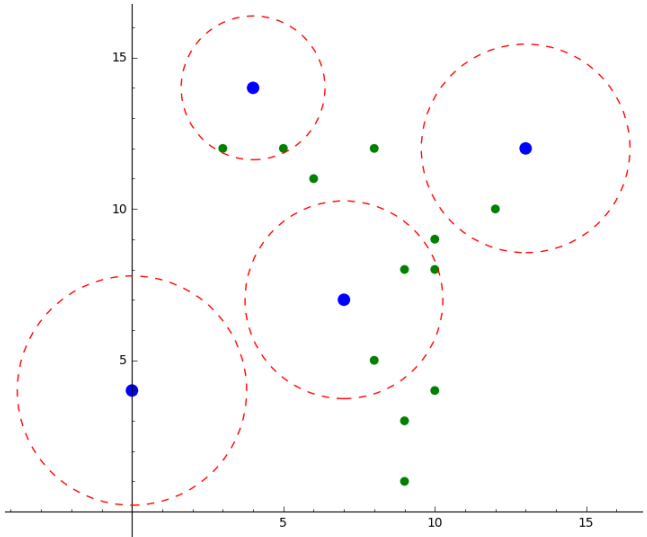
```
def instantiateTSMeshAlgorithmProblem():

      instance = TSMeshAlgorithmProblem(16,16,4,12,128,"swap",51113,
                                        15,2^(32),10,"Uniform",0.25,
                                        [4, 1])

      instance.executeTSMeshAlgorithm
```

4.2.1 SOLUCIÓN INICIAL

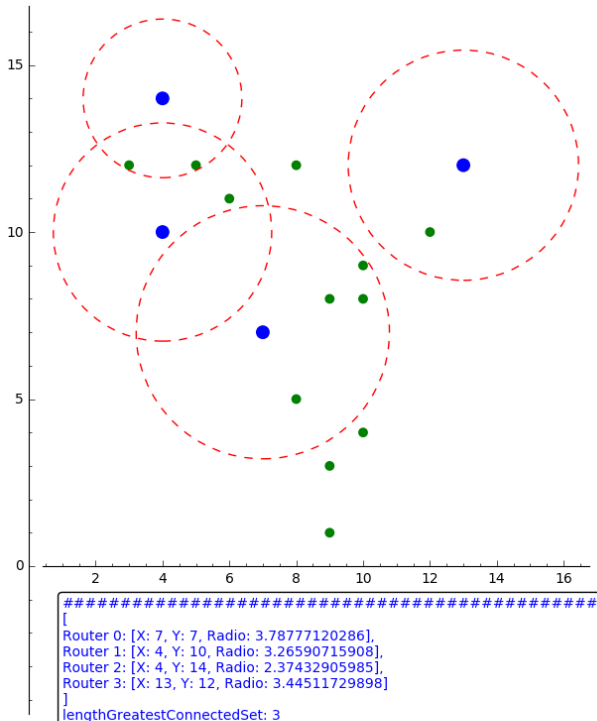
```
##########################################################################
######################
[
Router 0: [X: 0, Y: 4, Radio: 3.78777120286],
Router 1: [X: 7, Y: 7, Radio: 3.26590715908],
Router 2: [X: 4, Y: 14, Radio: 2.37432905985],
Router 3: [X: 13, Y: 12, Radio: 3.44511729898]
]
lengthGreatestConnectedSet: 1
numberClientsCovered: 6
##########################################################################
######################
```




```
##########################################################################################
[
Router 0: [X: 0, Y: 4, Radio: 3.78777120286],
Router 1: [X: 7, Y: 7, Radio: 3.26590715908],
Router 2: [X: 4, Y: 14, Radio: 2.37432905985],
Router 3: [X: 13, Y: 12, Radio: 3.44511729898]
]
lengthGreatestConnectedSet: 1
numberClientsCovered: 6
##########################################################################################
lengthGreatestConnectedSet: 1/4
numberClientsCovered: 6/12
```

4.2.2 SOLUCIÓN FINAL

```
##########################################################################
######################
[
Router 0: [X: 7, Y: 7, Radio: 3.78777120286],
Router 1: [X: 4, Y: 10, Radio: 3.26590715908],
Router 2: [X: 4, Y: 14, Radio: 2.37432905985],
Router 3: [X: 13, Y: 12, Radio: 3.44511729898]
]
lengthGreatestConnectedSet: 3
numberClientsCovered: 8
appliedMovement: [type: swap, 0<->1]
##########################################################################
######################
```




```
##########################################################################################
[
Router 0: [X: 7, Y: 7, Radio: 3.78777120286],
Router 1: [X: 4, Y: 10, Radio: 3.26590715908],
Router 2: [X: 4, Y: 14, Radio: 2.37432905985],
Router 3: [X: 13, Y: 12, Radio: 3.44511729898]
]
lengthGreatestConnectedSet: 3
numberClientsCovered: 8
appliedMovement: [type: swap, 0<->1]
##########################################################################################
lengthGreatestConnectedSet: 3/4
numberClientsCovered: 8/12
```

4.3 EJEMPLO 3

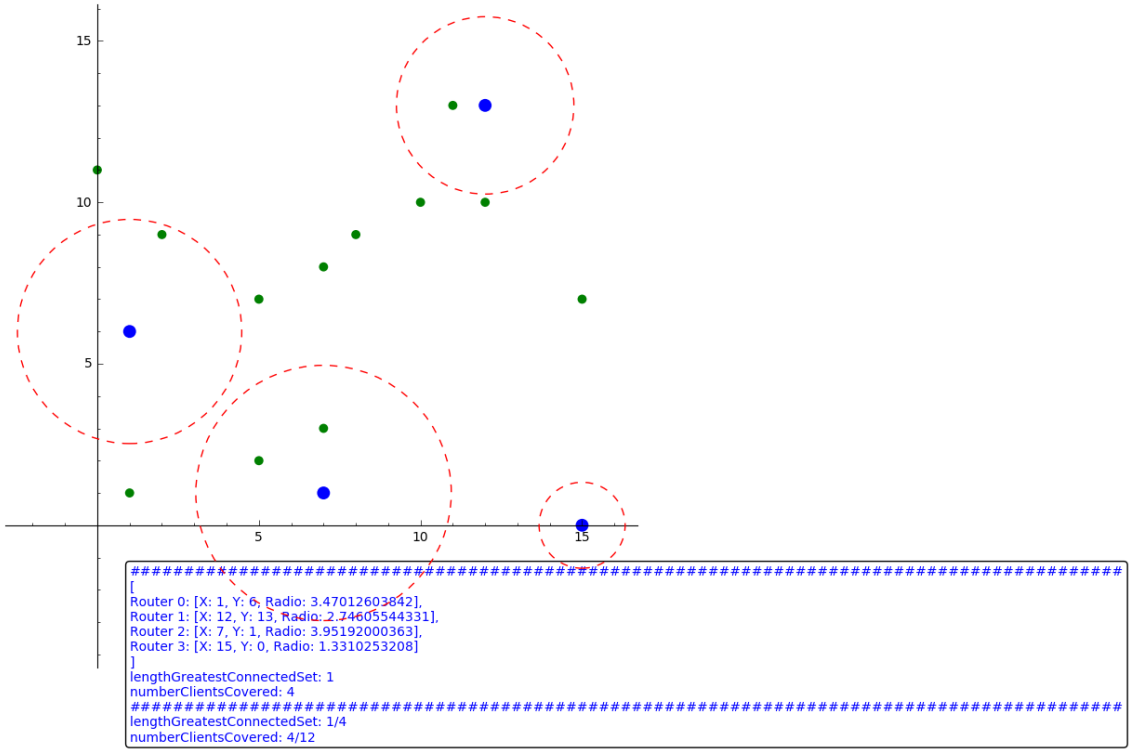
```
def instantiateTSMeshAlgorithmProblem():

      instance = TSMeshAlgorithmProblem(16,16,4,12,256,"swap",51113,
                                        15,2^(32),10,"Uniform",0.25,
                                        [4, 1])

      instance.executeTSMeshAlgorithm
```

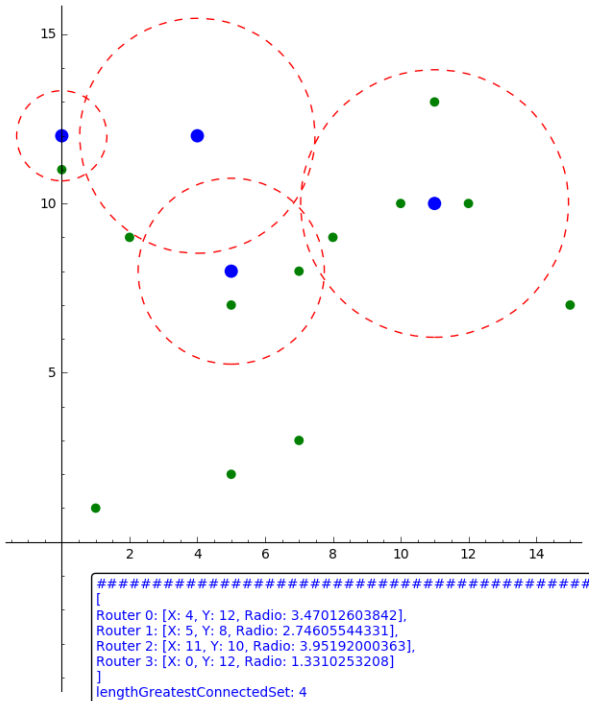
4.3.1 SOLUCIÓN INICIAL

```
##########################################################################
######################
[
Router 0: [X: 1, Y: 6, Radio: 3.47012603842],
Router 1: [X: 12, Y: 13, Radio: 2.74605544331],
Router 2: [X: 7, Y: 1, Radio: 3.95192000363],
Router 3: [X: 15, Y: 0, Radio: 1.3310253208]
]
lengthGreatestConnectedSet: 1
numberClientsCovered: 4
##########################################################################
######################
```



4.3.2 SOLUCIÓN FINAL

```
##########################################################################
######################
[
Router 0: [X: 4, Y: 12, Radio: 3.47012603842],
Router 1: [X: 5, Y: 8, Radio: 2.74605544331],
Router 2: [X: 11, Y: 10, Radio: 3.95192000363],
Router 3: [X: 0, Y: 12, Radio: 1.3310253208]
]
lengthGreatestConnectedSet: 4
numberClientsCovered: 7
appliedMovement: [type: moveToCell, 1->(5, 8)]
##########################################################################
######################
```




```
##########################################################################################
[
Router 0: [X: 4, Y: 12, Radio: 3.47012603842],
Router 1: [X: 5, Y: 8, Radio: 2.74605544331],
Router 2: [X: 11, Y: 10, Radio: 3.95192000363],
Router 3: [X: 0, Y: 12, Radio: 1.3310253208]
]
lengthGreatestConnectedSet: 4
numberClientsCovered: 7
appliedMovement: [type: moveToCell, 1->(5, 8)]
##########################################################################################
lengthGreatestConnectedSet: 4/4
numberClientsCovered: 7/12
```

4.4 EJEMPLO 4

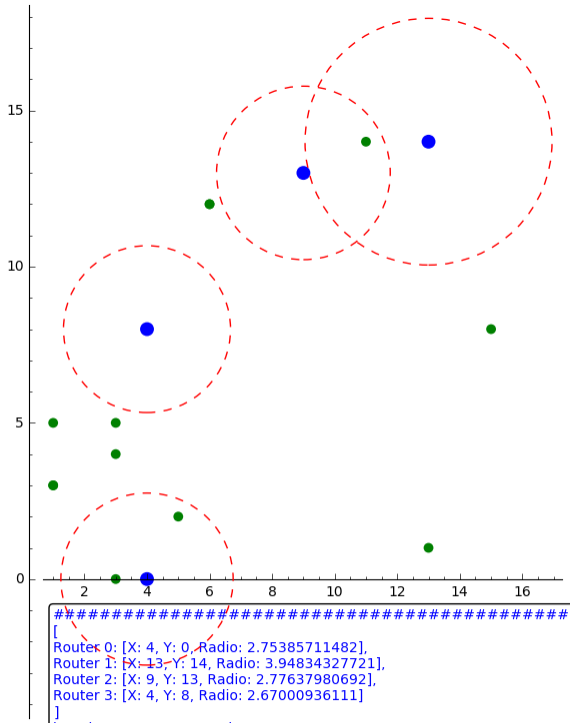
```
def instantiateTSMeshAlgorithmProblem():

      instance = TSMeshAlgorithmProblem(16,16,4,12,512,"swap",51113,
                                        15,2^(32),10,"Uniform",0.25,
                                        [4, 1])

      instance.executeTSMeshAlgorithm
```

4.4.1 SOLUCIÓN INICIAL

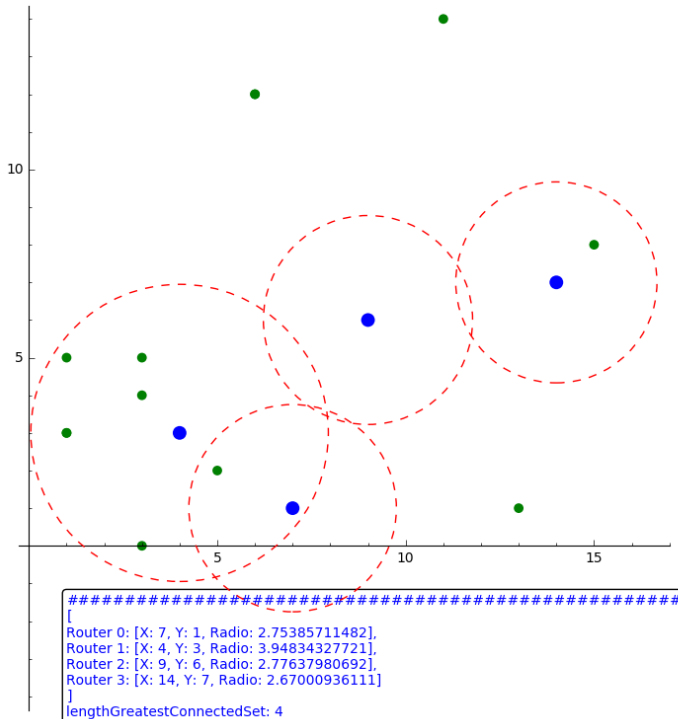
```
##########################################################################
######################
[
Router 0: [X: 4, Y: 0, Radio: 2.75385711482],
Router 1: [X: 13, Y: 14, Radio: 3.94834327721],
Router 2: [X: 9, Y: 13, Radio: 2.77637980692],
Router 3: [X: 4, Y: 8, Radio: 2.67000936111]
]
lengthGreatestConnectedSet: 2
numberClientsCovered: 3
##########################################################################
######################
```




```
##########################################################################################################
[
Router 0: [X: 4, Y: 0, Radio: 2.75385711482],
Router 1: [X: 13, Y: 14, Radio: 3.94834327721],
Router 2: [X: 9, Y: 13, Radio: 2.77637980692],
Router 3: [X: 4, Y: 8, Radio: 2.67000936111]
]
lengthGreatestConnectedSet: 2
numberClientsCovered: 3
##########################################################################################################
lengthGreatestConnectedSet: 2/4
numberClientsCovered: 3/12
```

4.4.2 SOLUCIÓN FINAL

```
##########################################################################
######################
[
Router 0: [X: 7, Y: 1, Radio: 2.75385711482],
Router 1: [X: 4, Y: 3, Radio: 3.94834327721],
Router 2: [X: 9, Y: 6, Radio: 2.77637980692],
Router 3: [X: 14, Y: 7, Radio: 2.67000936111]
]
lengthGreatestConnectedSet: 4
numberClientsCovered: 8
appliedMovement: [type: moveToCell, 3->(14, 7)]
##########################################################################
######################
```




```
##########################################################################################
[
Router 0: [X: 7, Y: 1, Radio: 2.75385711482],
Router 1: [X: 4, Y: 3, Radio: 3.94834327721],
Router 2: [X: 9, Y: 6, Radio: 2.77637980692],
Router 3: [X: 14, Y: 7, Radio: 2.67000936111]
]
lengthGreatestConnectedSet: 4
numberClientsCovered: 8
appliedMovement: [type: moveToCell, 3->(14, 7)]
##########################################################################################
lengthGreatestConnectedSet: 4/4
numberClientsCovered: 8/12
```

4.5 EJEMPLO 5

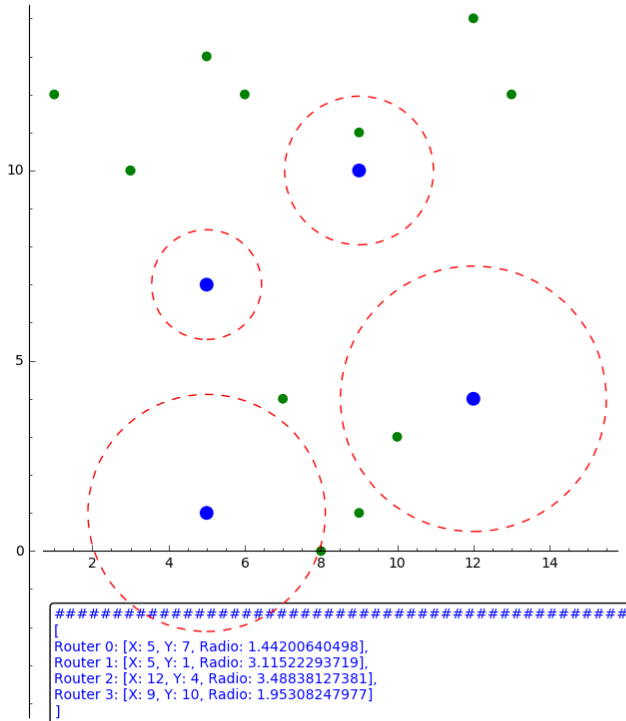
```
def instantiateTSMeshAlgorithmProblem():

      instance = TSMeshAlgorithmProblem(16,16,4,12,1024,"swap",51113,
                                        15,2^(32),10,"Uniform",0.25,
                                        [4, 1])

      instance.executeTSMeshAlgorithm
```

4.5.1 SOLUCIÓN INICIAL

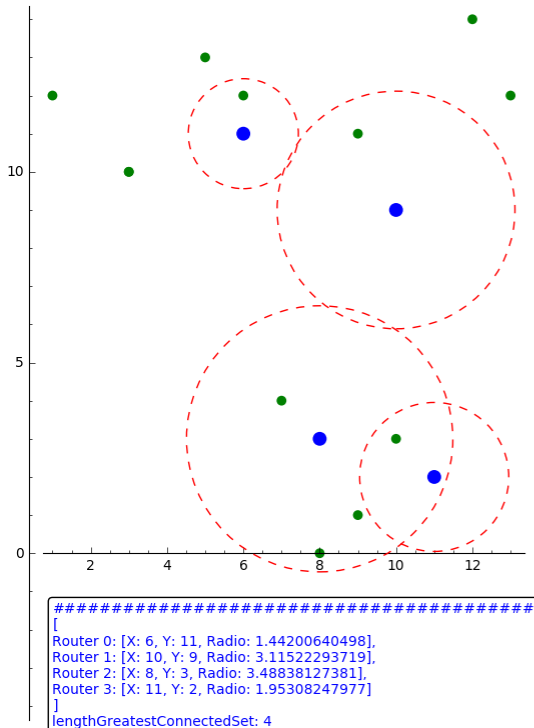
```
##########################################################################
######################
[
Router 0: [X: 5, Y: 7, Radio: 1.44200640498],
Router 1: [X: 5, Y: 1, Radio: 3.11522293719],
Router 2: [X: 12, Y: 4, Radio: 3.48838127381],
Router 3: [X: 9, Y: 10, Radio: 1.95308247977]
]
lengthGreatestConnectedSet: 1
numberClientsCovered: 2
##########################################################################
######################
```




```
##########################################################################################################
[
Router 0: [X: 5, Y: 7, Radio: 1.44200640498],
Router 1: [X: 5, Y: 1, Radio: 3.11522293719],
Router 2: [X: 12, Y: 4, Radio: 3.48838127381],
Router 3: [X: 9, Y: 10, Radio: 1.95308247977]
]
lengthGreatestConnectedSet: 1
numberClientsCovered: 2
##########################################################################################################
lengthGreatestConnectedSet: 1/4
numberClientsCovered: 2/12
```

4.5.2 SOLUCIÓN FINAL

```
##########################################################################
######################
[
Router 0: [X: 6, Y: 11, Radio: 1.44200640498],
Router 1: [X: 10, Y: 9, Radio: 3.11522293719],
Router 2: [X: 8, Y: 3, Radio: 3.48838127381],
Router 3: [X: 11, Y: 2, Radio: 1.95308247977]
]
lengthGreatestConnectedSet: 4
numberClientsCovered: 6
appliedMovement: [type: swap, 2<->3]
##########################################################################
######################
```




```
##########################################################################################
[
Router 0: [X: 6, Y: 11, Radio: 1.44200640498],
Router 1: [X: 10, Y: 9, Radio: 3.11522293719],
Router 2: [X: 8, Y: 3, Radio: 3.48838127381],
Router 3: [X: 11, Y: 2, Radio: 1.95308247977]
]
lengthGreatestConnectedSet: 4
numberClientsCovered: 6
appliedMovement: [type: swap, 2<->3]
##########################################################################################
lengthGreatestConnectedSet: 4/4
numberClientsCovered: 6/12
```