

ExpositoTOP

1.0

Generado por Doxygen 1.14.0



---

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Lista de obsoletos</b>  | <b>1</b>  |
| <b>2</b> | <b>Índice de espacios de nombres</b>                                     | <b>3</b>  |
| 2.1      | Lista de espacios de nombres . . . . .                                   | 3         |
| <b>3</b> | <b>Índice jerárquico</b>   | <b>5</b>  |
| 3.1      | Jerarquía de clases . . . . .  | 5         |
| <b>4</b> | <b>Índice de clases</b>  | <b>7</b>  |
| 4.1      | Lista de clases . . . . .  | 7         |
| <b>5</b> | <b>Índice de archivos</b>  | <b>9</b>  |
| 5.1      | Lista de archivos . . . . .  | 9         |
| <b>6</b> | <b>Documentación de espacios de nombres</b>                              | <b>11</b> |
| 6.1      | Paquete es.ull.esit.utilities . . . . .                                  | 11        |
| 6.2      | Paquete es.ull.esit.utils . . . . .                                      | 11        |
| 6.3      | Paquete top . . . . .  | 11        |
| <b>7</b> | <b>Documentación de clases</b>   | <b>13</b> |
| 7.1      | Referencia de la clase es.ull.esit.utilities.BellmanFord . . . . .       | 13        |
| 7.1.1    | Descripción detallada . . . . .  | 14        |
| 7.1.2    | Documentación de constructores y destructores . . . . .                  | 14        |
| 7.1.2.1  | BellmanFord() . . . . .  | 14        |
| 7.1.3    | Documentación de funciones miembro . . . . .                             | 15        |
| 7.1.3.1  | calculateEdges() . . . . .   | 15        |
| 7.1.3.2  | getDistances() . . . . .   | 15        |
| 7.1.3.3  | getValue() . . . . .   | 16        |
| 7.1.3.4  | solve() . . . . .  | 16        |
| 7.1.4    | Documentación de datos miembro . . . . .                                 | 17        |
| 7.1.4.1  | distanceMatrix . . . . .   | 17        |
| 7.1.4.2  | distances . . . . .  | 17        |
| 7.1.4.3  | edges1 . . . . .   | 17        |
| 7.1.4.4  | edges2 . . . . .   | 17        |
| 7.1.4.5  | INFINITY . . . . .   | 17        |
| 7.1.4.6  | nodes . . . . .  | 17        |
| 7.1.4.7  | path . . . . .   | 18        |
| 7.1.4.8  | value . . . . .  | 18        |
| 7.2      | Referencia de la clase es.ull.esit.utilities.ExpositoUtilities . . . . . | 18        |
| 7.2.1    | Descripción detallada . . . . .  | 19        |
| 7.2.2    | Documentación de funciones miembro . . . . .                             | 19        |
| 7.2.2.1  | getFirstAppearance() . . . . .   | 19        |
| 7.2.2.2  | getFormat() [1/12] . . . . .   | 20        |
| 7.2.2.3  | getFormat() [2/12] . . . . .   | 20        |

---

|  |    |
|--|----|
| 7.2.2.4 getFormat() [3/12] . . . . .   | 21 |
| 7.2.2.5 getFormat() [4/12] . . . . .   | 22 |
| 7.2.2.6 getFormat() [5/12] . . . . .   | 23 |
| 7.2.2.7 getFormat() [6/12] . . . . .   | 24 |
| 7.2.2.8 getFormat() [7/12] . . . . .   | 25 |
| 7.2.2.9 getFormat() [8/12] . . . . .   | 25 |
| 7.2.2.10 getFormat() [9/12] . . . . .  | 26 |
| 7.2.2.11 getFormat() [10/12] . . . . .   | 26 |
| 7.2.2.12 getFormat() [11/12] . . . . .   | 27 |
| 7.2.2.13 getFormat() [12/12] . . . . .   | 28 |
| 7.2.2.14 isAcyclic() . . . . .   | 29 |
| 7.2.2.15 isDouble() . . . . .  | 29 |
| 7.2.2.16 isInteger() . . . . .   | 30 |
| 7.2.2.17 multiplyMatrices() . . . . .  | 31 |
| 7.2.2.18 printFile() . . . . .   | 32 |
| 7.2.2.19 simplifyString() . . . . .  | 32 |
| 7.2.2.20 thereIsPath() . . . . .   | 33 |
| 7.2.2.21 writeTextToFile() . . . . .   | 33 |
| 7.2.3 Documentación de datos miembro . . . . .   | 34 |
| 7.2.3.1 ALIGNMENT_LEFT . . . . .   | 34 |
| 7.2.3.2 ALIGNMENT_RIGHT . . . . .  | 34 |
| 7.2.3.3 DEFAULT_COLUMN_WIDTH . . . . .   | 34 |
| 7.3 Referencia de la clase top.mainTOPTW . . . . .                                       | 34 |
| 7.3.1 Descripción detallada . . . . .  | 35 |
| 7.3.2 Documentación de funciones miembro . . . . .                                       | 35 |
| 7.3.2.1 main() . . . . .   | 35 |
| 7.4 Referencia de la plantilla de la clase es.ull.esit.utils.Pair< F, S > . . . . .      | 36 |
| 7.4.1 Descripción detallada . . . . .  | 37 |
| 7.4.2 Documentación de constructores y destructores . . . . .                            | 37 |
| 7.4.2.1 Pair() . . . . .   | 37 |
| 7.4.3 Documentación de funciones miembro . . . . .                                       | 38 |
| 7.4.3.1 create() . . . . .   | 38 |
| 7.4.3.2 equals() . . . . .   | 39 |
| 7.4.3.3 hashCode() . . . . .   | 39 |
| 7.4.4 Documentación de datos miembro . . . . .   | 40 |
| 7.4.4.1 first . . . . .  | 40 |
| 7.4.4.2 second . . . . .   | 40 |
| 7.5 Referencia de la plantilla de la clase es.ull.esit.utilities.PowerSet< E > . . . . . | 40 |
| 7.5.1 Descripción detallada . . . . .  | 41 |
| 7.5.2 Documentación de constructores y destructores . . . . .                            | 41 |
| 7.5.2.1 PowerSet() . . . . .   | 41 |
| 7.5.3 Documentación de funciones miembro . . . . .                                       | 42 |

|   |    |
|---|----|
| 7.5.3.1 iterator() . . . . .  | 42 |
| 7.5.4 Documentación de datos miembro . . . . .  | 43 |
| 7.5.4.1 arr . . . . .   | 43 |
| 7.6 Referencia de la clase es.uji.esit.utilities.PowerSet< E >.PowerSetIterator . . . . . | 43 |
| 7.6.1 Descripción detallada . . . . .   | 44 |
| 7.6.2 Documentación de funciones miembro . . . . .  | 44 |
| 7.6.2.1 hasNext() . . . . .   | 44 |
| 7.6.2.2 next() . . . . .  | 45 |
| 7.6.2.3 remove() . . . . .  | 45 |
| 7.6.3 Documentación de datos miembro . . . . .  | 46 |
| 7.6.3.1 bset . . . . .  | 46 |
| 7.7 Referencia de la clase top.TOPTW . . . . .  | 46 |
| 7.7.1 Descripción detallada . . . . .   | 48 |
| 7.7.2 Documentación de constructores y destructores . . . . .                             | 48 |
| 7.7.2.1 TOPTW() . . . . .   | 48 |
| 7.7.3 Documentación de funciones miembro . . . . .  | 48 |
| 7.7.3.1 addNode() . . . . .   | 48 |
| 7.7.3.2 addNodeDepot() . . . . .  | 49 |
| 7.7.3.3 calculateDistanceMatrix() . . . . .   | 49 |
| 7.7.3.4 getDistance() [1/4] . . . . .   | 49 |
| 7.7.3.5 getDistance() [2/4] . . . . .   | 50 |
| 7.7.3.6 getDistance() [3/4] . . . . .   | 51 |
| 7.7.3.7 getDistance() [4/4] . . . . .   | 51 |
| 7.7.3.8 getDueTime() . . . . .  | 52 |
| 7.7.3.9 getMaxRoutes() . . . . .  | 53 |
| 7.7.3.10 getMaxTimePerRoute() . . . . .   | 54 |
| 7.7.3.11 getNodes() . . . . .   | 54 |
| 7.7.3.12 getPOIs() . . . . .  | 54 |
| 7.7.3.13 getReadyTime() . . . . .   | 55 |
| 7.7.3.14 getScore() [1/2] . . . . .   | 55 |
| 7.7.3.15 getScore() [2/2] . . . . .   | 56 |
| 7.7.3.16 getServiceTime() . . . . .   | 56 |
| 7.7.3.17 getTime() . . . . .  | 57 |
| 7.7.3.18 getVehicles() . . . . .  | 58 |
| 7.7.3.19 getX() . . . . .   | 58 |
| 7.7.3.20 getY() . . . . .   | 59 |
| 7.7.3.21 isDepot() . . . . .  | 60 |
| 7.7.3.22 setDueTime() . . . . .   | 61 |
| 7.7.3.23 setMaxRoutes() . . . . .   | 61 |
| 7.7.3.24 setMaxTimePerRoute() . . . . .   | 62 |
| 7.7.3.25 setNodes() . . . . .   | 62 |
| 7.7.3.26 setReadyTime() . . . . .   | 62 |

|   |    |
|---|----|
| 7.7.3.27 setScore()                                 | 63 |
| 7.7.3.28 setServiceTime()                           | 63 |
| 7.7.3.29 setX()                                     | 64 |
| 7.7.3.30 setY()                                     | 64 |
| 7.7.3.31 toString()                                 | 65 |
| 7.7.4 Documentación de datos miembro                | 66 |
| 7.7.4.1 depots                                      | 66 |
| 7.7.4.2 distanceMatrix                              | 66 |
| 7.7.4.3 dueTime                                     | 66 |
| 7.7.4.4 maxRoutes                                   | 66 |
| 7.7.4.5 maxTimePerRoute                             | 66 |
| 7.7.4.6 nodes                                       | 67 |
| 7.7.4.7 readyTime                                   | 67 |
| 7.7.4.8 score                                       | 67 |
| 7.7.4.9 serviceTime                                 | 67 |
| 7.7.4.10 vehicles                                   | 67 |
| 7.7.4.11 x  | 67 |
| 7.7.4.12 y  | 67 |
| 7.8 Referencia de la clase top.TOPTWEvaluator       | 68 |
| 7.8.1 Descripción detallada                         | 68 |
| 7.8.2 Documentación de funciones miembro            | 68 |
| 7.8.2.1 evaluate()                                  | 68 |
| 7.8.3 Documentación de datos miembro                | 69 |
| 7.8.3.1 NO_EVALUATED                                | 69 |
| 7.9 Referencia de la clase top.TOPTWGRASP           | 69 |
| 7.9.1 Descripción detallada                         | 70 |
| 7.9.2 Documentación de constructores y destructores | 70 |
| 7.9.2.1 TOPTWGRASP()                                | 70 |
| 7.9.3 Documentación de funciones miembro            | 71 |
| 7.9.3.1 aleatorySelectionRCL()                      | 71 |
| 7.9.3.2 comprehensiveEvaluation()                   | 71 |
| 7.9.3.3 computeGreedySolution()                     | 74 |
| 7.9.3.4 fuzzySelectionAlphaCutRCL()                 | 76 |
| 7.9.3.5 fuzzySelectionBestFDRCL()                   | 77 |
| 7.9.3.6 getMaxScore()                               | 77 |
| 7.9.3.7 getSolution()                               | 78 |
| 7.9.3.8 getSolutionTime()                           | 78 |
| 7.9.3.9 GRASP()                                     | 78 |
| 7.9.3.10 setSolution()                              | 80 |
| 7.9.3.11 setSolutionTime()                          | 81 |
| 7.9.3.12 updateSolution()                           | 81 |
| 7.9.4 Documentación de datos miembro                | 82 |

|  |     |
|--|-----|
| 7.9.4.1 NO_EVALUATED . . . . .                                 | 82  |
| 7.9.4.2 RANDOM . . . . .                                       | 83  |
| 7.9.4.3 solution . . . . .                                     | 83  |
| 7.9.4.4 solutionTime . . . . .                                 | 83  |
| 7.10 Referencia de la clase top.TOPTWReader . . . . .          | 83  |
| 7.10.1 Descripción detallada . . . . .                         | 83  |
| 7.10.2 Documentación de funciones miembro . . . . .            | 83  |
| 7.10.2.1 readProblem() . . . . .                               | 83  |
| 7.11 Referencia de la clase top.TOPTWRoute . . . . .           | 85  |
| 7.11.1 Descripción detallada . . . . .                         | 86  |
| 7.11.2 Documentación de funciones miembro . . . . .            | 86  |
| 7.11.2.1 getId() . . . . .                                     | 86  |
| 7.11.2.2 getPredecesor() . . . . .                             | 86  |
| 7.11.2.3 getSuccesor() . . . . .                               | 87  |
| 7.11.2.4 setId() . . . . .                                     | 87  |
| 7.11.2.5 setPredecesor() . . . . .                             | 87  |
| 7.11.2.6 setSuccesor() . . . . .                               | 87  |
| 7.12 Referencia de la clase top.TOPTWSolution . . . . .        | 88  |
| 7.12.1 Descripción detallada . . . . .                         | 90  |
| 7.12.2 Documentación de constructores y destructores . . . . . | 90  |
| 7.12.2.1 TOPTWSolution() . . . . .                             | 90  |
| 7.12.3 Documentación de funciones miembro . . . . .            | 91  |
| 7.12.3.1 addRoute() . . . . .                                  | 91  |
| 7.12.3.2 equals() [1/2] . . . . .                              | 92  |
| 7.12.3.3 equals() [2/2] . . . . .                              | 92  |
| 7.12.3.4 evaluateFitness() . . . . .                           | 93  |
| 7.12.3.5 getAvailableVehicles() . . . . .                      | 94  |
| 7.12.3.6 getCreatedRoutes() . . . . .                          | 94  |
| 7.12.3.7 getDistance() . . . . .                               | 95  |
| 7.12.3.8 getIndexRoute() . . . . .                             | 96  |
| 7.12.3.9 getInfoSolution() . . . . .                           | 96  |
| 7.12.3.10 getObjectiveFunctionValue() . . . . .                | 98  |
| 7.12.3.11 getPositionInRoute() . . . . .                       | 99  |
| 7.12.3.12 getPredecessor() . . . . .                           | 99  |
| 7.12.3.13 getPredecessors() . . . . .                          | 100 |
| 7.12.3.14 getProblem() . . . . .                               | 100 |
| 7.12.3.15 getSuccessor() . . . . .                             | 100 |
| 7.12.3.16 getSuccessors() . . . . .                            | 101 |
| 7.12.3.17 getWaitingTime() . . . . .                           | 101 |
| 7.12.3.18 hashCode() . . . . .                                 | 102 |
| 7.12.3.19 initSolution() . . . . .                             | 102 |
| 7.12.3.20 isDepot() . . . . .                                  | 102 |

|  |            |
|--|------------|
| 7.12.3.21 printSolution() . . . . .  | 103        |
| 7.12.3.22 setAvailableVehicles() . . . . .   | 104        |
| 7.12.3.23 setObjectiveFunctionValue() . . . . .  | 104        |
| 7.12.3.24 setPositionInRoute() . . . . .   | 104        |
| 7.12.3.25 setPredecessor() . . . . .   | 104        |
| 7.12.3.26 setSuccessor() . . . . .   | 105        |
| 7.12.3.27 setWaitingTime() . . . . .   | 105        |
| 7.12.4 Documentación de datos miembro . . . . .  | 106        |
| 7.12.4.1 availableVehicles . . . . .   | 106        |
| 7.12.4.2 NO_INITIALIZED . . . . .  | 106        |
| 7.12.4.3 objectiveFunctionValue . . . . .  | 106        |
| 7.12.4.4 positionInRoute . . . . .   | 106        |
| 7.12.4.5 predecessors . . . . .  | 106        |
| 7.12.4.6 problem . . . . .   | 107        |
| 7.12.4.7 routes . . . . .  | 107        |
| 7.12.4.8 successors . . . . .  | 107        |
| 7.12.4.9 waitingTime . . . . .   | 107        |
| <b>8 Documentación de archivos</b> . . . . .   | <b>109</b> |
| 8.1 Referencia del archivo C:/Users/Kefle/Desktop/ExpositoTOP/src/es/ull/esit/utilities/BellmanFord.java . . . . .       | 109        |
| 8.1.1 Descripción detallada . . . . .  | 110        |
| 8.2 BellmanFord.java . . . . .   | 110        |
| 8.3 Referencia del archivo C:/Users/Kefle/Desktop/ExpositoTOP/src/es/ull/esit/utilities/ExpositoUtilities.java . . . . . | 111        |
| 8.3.1 Descripción detallada . . . . .  | 111        |
| 8.4 ExpositoUtilities.java . . . . .   | 112        |
| 8.5 Referencia del archivo C:/Users/Kefle/Desktop/ExpositoTOP/src/es/ull/esit/utilities/PowerSet.java . . . . .          | 115        |
| 8.5.1 Descripción detallada . . . . .  | 115        |
| 8.6 PowerSet.java . . . . .  | 116        |
| 8.7 Referencia del archivo C:/Users/Kefle/Desktop/ExpositoTOP/src/es/ull/esit/utils/Pair.java . . . . .                  | 116        |
| 8.7.1 Descripción detallada . . . . .  | 117        |
| 8.8 Pair.java . . . . .  | 117        |
| 8.9 Referencia del archivo C:/Users/Kefle/Desktop/ExpositoTOP/src/top/mainTOPTW.java . . . . .                           | 118        |
| 8.9.1 Descripción detallada . . . . .  | 118        |
| 8.10 mainTOPTW.java . . . . .  | 118        |
| 8.11 Referencia del archivo C:/Users/Kefle/Desktop/ExpositoTOP/src/top/TOPTW.java . . . . .                              | 119        |
| 8.11.1 Descripción detallada . . . . .   | 119        |
| 8.12 TOPTW.java . . . . .  | 120        |
| 8.13 Referencia del archivo C:/Users/Kefle/Desktop/ExpositoTOP/src/top/TOPTWEvaluator.java . . . . .                     | 122        |
| 8.13.1 Descripción detallada . . . . .   | 123        |
| 8.14 TOPTWEvaluator.java . . . . .   | 123        |
| 8.15 Referencia del archivo C:/Users/Kefle/Desktop/ExpositoTOP/src/top/TOPTWGRASP.java . . . . .                         | 123        |
| 8.15.1 Descripción detallada . . . . .   | 124        |

|   |            |
|---|------------|
| 8.16 TOPTWGRASP.java . . . . .  | 124        |
| 8.17 Referencia del archivo C:/Users/Kefle/Desktop/ExpositoTOP/src/top/TOPTWReader.java . . . . .   | 128        |
| 8.17.1 Descripción detallada . . . . .  | 129        |
| 8.18 TOPTWReader.java . . . . .   | 129        |
| 8.19 Referencia del archivo C:/Users/Kefle/Desktop/ExpositoTOP/src/top/TOPTWRoute.java . . . . .    | 130        |
| 8.19.1 Descripción detallada . . . . .  | 130        |
| 8.20 TOPTWRoute.java . . . . .  | 131        |
| 8.21 Referencia del archivo C:/Users/Kefle/Desktop/ExpositoTOP/src/top/TOPTWSolution.java . . . . . | 131        |
| 8.21.1 Descripción detallada . . . . .  | 132        |
| 8.22 TOPTWSolution.java . . . . .   | 132        |
| <b>Índice alfabético</b>  | <b>137</b> |



# Capítulo 1

## Listado de obsoletos

### Miembro `top.TOPTW.addNode()`

Este método parece no ser utilizado de forma segura, ya que no redimensiona los arrays.

### Miembro `top.TOPTW.addNodeDepot()`

Este método parece no ser utilizado de forma segura.



## **Capítulo 2**

# **Índice de espacios de nombres**

### **2.1. Lista de espacios de nombres**

Lista de los espacios de nombres documentados, con breves descripciones:

|   |    |
|---|----|
| <a href="#">es.ull.esit.utilities</a> . . . . . | 11 |
| <a href="#">es.ull.esit.utils</a> . . . . .     | 11 |
| <a href="#">top</a> . . . . .                   | 11 |



# Capítulo 3

## Índice jerárquico

### 3.1. Jerarquía de clases

Este listado de herencia está ordenado de forma general pero no está en orden alfabético estricto:

|  |    |
|--|----|
| es.ull.esit.utilities.BellmanFord . . . . .                    | 13 |
| es.ull.esit.utilities.ExpositoUtilities . . . . .              | 18 |
| Iterable   |    |
| es.ull.esit.utilities.PowerSet< E > . . . . .                  | 40 |
| Iterator   |    |
| es.ull.esit.utilities.PowerSet< E >.PowerSetIterator . . . . . | 43 |
| top.mainTOPTW . . . . .  | 34 |
| es.ull.esit.utils.Pair< F, S > . . . . .                       | 36 |
| top.TOPTW . . . . .  | 46 |
| top.TOPTWEvaluator . . . . .                                   | 68 |
| top.TOPTWGRASP . . . . .                                       | 69 |
| top.TOPTWReader . . . . .                                      | 83 |
| top.TOPTWRoute . . . . .                                       | 85 |
| top.TOPTWSolution . . . . .                                    | 88 |



# Capítulo 4

## Índice de clases

### 4.1. Lista de clases

Lista de clases, estructuras, uniones e interfaces con breves descripciones:

|  |  |    |
|--|--|----|
| <a href="#">es.ull.esit.utilities.BellmanFord</a>                          | Implementa el algoritmo de Bellman-Ford . . . . .  | 13 |
| <a href="#">es.ull.esit.utilities.ExpositoUtilities</a>                    | Proporciona un conjunto de herramientas estáticas de propósito general . . . . .   | 18 |
| <a href="#">top.mainTOPTW</a>  | Clase principal que contiene el método <code>main</code> para ejecutar el experimento . . . . .                                | 34 |
| <a href="#">es.ull.esit.utils.Pair&lt; F, S &gt;</a>                       | Una clase genérica que almacena un par de objetos inmutables . . . . .   | 36 |
| <a href="#">es.ull.esit.utilities.PowerSet&lt; E &gt;</a>                  | Implementa un iterador para generar todos los subconjuntos (conjunto potencia) de un conjunto dado . . . . .                   | 40 |
| <a href="#">es.ull.esit.utilities.PowerSet&lt; E &gt;.PowerSetIterator</a> |  | 43 |
| <a href="#">top.TOPTW</a>  | Modela una instancia del problema <code>TOPTW</code> (Team Orienteering Problem with Time Windows)                             | 46 |
| <a href="#">top.TOPTWEvaluator</a>   | Evalúa la calidad de una solución para el problema <code>TOPTW</code> . . . . .  | 68 |
| <a href="#">top.TOPTWGRASP</a>   | Implementa la metaheurística GRASP (Greedy Randomized Adaptive Search Procedure) para resolver el <code>TOPTW</code> . . . . . | 69 |
| <a href="#">top.TOPTWReader</a>  | Proporciona funcionalidad para leer una instancia del problema <code>TOPTW</code> desde un archivo de texto . . . . .          | 83 |
| <a href="#">top.TOPTWRoute</a>   | Modela una ruta simple mediante la identificación de sus nodos predecesor y sucesor . . . . .                                  | 85 |
| <a href="#">top.TOPTWSolution</a>  | Representa una solución a una instancia del problema <code>TOPTW</code> . . . . .  | 88 |



# Capítulo 5

## Índice de archivos

### 5.1. Lista de archivos

Lista de todos los archivos con breves descripciones:

|  |     |
|--|-----|
| C:/Users/Kefle/Desktop/ExpositoTOP/src/es/ull/esit/utilities/ <a href="#">BellmanFord.java</a>                               |     |
| Contiene la implementación del algoritmo de Bellman-Ford para encontrar el camino más corto en un grafo con pesos . . . . .  | 109 |
| C:/Users/Kefle/Desktop/ExpositoTOP/src/es/ull/esit/utilities/ <a href="#">ExpositoUtilities.java</a>                         |     |
| Contiene una colección de métodos de utilidad estáticos para diversas tareas . . . . .                                       | 111 |
| C:/Users/Kefle/Desktop/ExpositoTOP/src/es/ull/esit/utilities/ <a href="#">PowerSet.java</a>                                  |     |
| Contiene la implementación de un iterador para generar el conjunto potencia de un conjunto dado . . . . .                    | 115 |
| C:/Users/Kefle/Desktop/ExpositoTOP/src/es/ull/esit/utils/ <a href="#">Pair.java</a>  |     |
| Contiene la definición de una clase genérica para almacenar un par de objetos . . . . .                                      | 116 |
| C:/Users/Kefle/Desktop/ExpositoTOP/src/top/ <a href="#">mainTOPTW.java</a>   |     |
| Punto de entrada principal para ejecutar el solver GRASP en un conjunto de instancias TOPTW                                  | 118 |
| C:/Users/Kefle/Desktop/ExpositoTOP/src/top/ <a href="#">TOPTW.java</a>   |     |
| Contiene la definición de la clase TOPTW, que modela una instancia del Team Orienteering Problem with Time Windows . . . . . | 119 |
| C:/Users/Kefle/Desktop/ExpositoTOP/src/top/ <a href="#">TOPTWEvaluator.java</a>  |     |
| Contiene la clase para evaluar la función objetivo de una solución TOPTW . . . . .   | 122 |
| C:/Users/Kefle/Desktop/ExpositoTOP/src/top/ <a href="#">TOPTWGRASP.java</a>  |     |
| Contiene la implementación de la metaheurística GRASP para el problema TOPTW . . . . .                                       | 123 |
| C:/Users/Kefle/Desktop/ExpositoTOP/src/top/ <a href="#">TOPTWReader.java</a>   |     |
| Contiene la clase TOPTWReader para leer instancias del problema TOPTW desde archivos . . . . .                               | 128 |
| C:/Users/Kefle/Desktop/ExpositoTOP/src/top/ <a href="#">TOPTWRoute.java</a>  |     |
| Contiene la definición de la clase TOPTWRoute, que representa una ruta simple . . . . .                                      | 130 |
| C:/Users/Kefle/Desktop/ExpositoTOP/src/top/ <a href="#">TOPTWSolution.java</a>   |     |
| Contiene la definición de la clase TOPTWSolution, que representa una solución para el problema TOPTW . . . . .               | 131 |



# Capítulo 6

## Documentación de espacios de nombres

### 6.1. Paquete es.ull.esit.utilities

#### Clases

- class [BellmanFord](#)  
*Implementa el algoritmo de Bellman-Ford.*
- class [ExpositoUtilities](#)  
*Proporciona un conjunto de herramientas estáticas de propósito general.*
- class [PowerSet](#)  
*Implementa un iterador para generar todos los subconjuntos (conjunto potencia) de un conjunto dado.*

### 6.2. Paquete es.ull.esit.utils

#### Clases

- class [Pair](#)  
*Una clase genérica que almacena un par de objetos inmutables.*

### 6.3. Paquete top

#### Clases

- class [mainTOPTW](#)  
*Clase principal que contiene el método `main` para ejecutar el experimento.*
- class [TOPTW](#)  
*Modela una instancia del problema [TOPTW](#) (Team Orienteering Problem with Time Windows).*
- class [TOPTWEvaluator](#)  
*Evaluá la calidad de una solución para el problema [TOPTW](#).*
- class [TOPTWGRASP](#)  
*Implementa la metaheurística GRASP (Greedy Randomized Adaptive Search Procedure) para resolver el [TOPTW](#).*
- class [TOPTWReader](#)  
*Proporciona funcionalidad para leer una instancia del problema [TOPTW](#) desde un archivo de texto.*
- class [TOPTWRoute](#)  
*Modela una ruta simple mediante la identificación de sus nodos predecesor y sucesor.*
- class [TOPTWSolution](#)  
*Representa una solución a una instancia del problema [TOPTW](#).*



# Capítulo 7

## Documentación de clases

### 7.1. Referencia de la clase es.uil.esit.utilities.BellmanFord

Implementa el algoritmo de Bellman-Ford.

#### Métodos públicos

- `BellmanFord (int[][] distanceMatrix, int nodes, ArrayList< Integer > path)`  
*Constructor de la clase `BellmanFord`.*
- `int[] getDistances ()`  
*Obtiene el array de distancias calculadas desde el nodo origen.*
- `int getValue ()`  
*Obtiene el valor (coste) del camino más corto calculado.*
- `void solve ()`  
*Ejecuta el algoritmo de Bellman-Ford.*

#### Métodos privados

- `void calculateEdges ()`  
*Construye las listas de aristas a partir de la matriz de adyacencia.*

#### Atributos privados

- `final int[][] distanceMatrix`  
*Matriz de adyacencia que representa las distancias (pesos) entre nodos.*
- `ArrayList< Integer > edges1 = null`  
*Lista de nodos de origen para cada arista del grafo.*
- `ArrayList< Integer > edges2 = null`  
*Lista de nodos de destino para cada arista del grafo.*
- `final int nodes`  
*Número total de nodos en el grafo.*
- `final ArrayList< Integer > path`  
*Lista para almacenar el camino más corto encontrado.*
- `int[] distances = null`  
*Array para almacenar las distancias más cortas desde el origen a cada nodo.*
- `int value`  
*El valor (coste) del camino más corto encontrado.*

## Atributos estáticos privados

- static final int **INFINITY** = 999999

*Valor que representa el infinito, usado para inicializar distancias.*

### 7.1.1. Descripción detallada

Implementa el algoritmo de Bellman-Ford.

Se utiliza para encontrar los caminos más cortos desde un único nodo origen a todos los demás nodos en un grafo dirigido y ponderado. Es capaz de manejar aristas con pesos negativos.

Definición en la línea 15 del archivo [BellmanFord.java](#).

### 7.1.2. Documentación de constructores y destructores

#### 7.1.2.1. BellmanFord()

```
es.ull.esit.utilities.BellmanFord.BellmanFord (
    int distanceMatrix[][],
    int nodes,
    ArrayList< Integer > path) [inline]
```

Constructor de la clase [BellmanFord](#).

Parámetros

|                             |   |
|-----------------------------|---|
| <code>distanceMatrix</code> | La matriz de adyacencia del grafo.                                      |
| <code>nodes</code>          | El número de nodos en el grafo.   |
| <code>path</code>           | Una lista (generalmente vacía) que se llenará con el camino encontrado. |

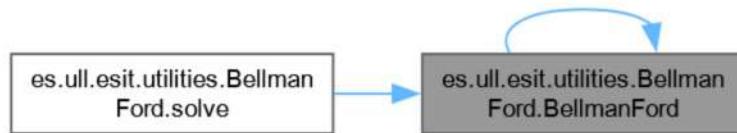
Definición en la línea 56 del archivo [BellmanFord.java](#).

```
00056
00057     this.distanceMatrix = distanceMatrix;
00058     this.nodes = nodes;
00059     this.path = path;
00060     this.calculateEdges();
00061     this.value = BellmanFord.INFINITY;
00062 }
```

Gráfico de llamadas de esta función:



Gráfico de llamadas a esta función:



### 7.1.3. Documentación de funciones miembro

#### 7.1.3.1. calculateEdges()

```
void es.ull.esit.utilities.BellmanFord.calculateEdges () [inline], [private]
```

Construye las listas de aristas a partir de la matriz de adyacencia.

Recorre la matriz de distancias y, si existe una arista (distancia no es infinita), la añade a las listas edges1 (origen) y edges2 (destino).

Definición en la línea 69 del archivo [BellmanFord.java](#).

```

00069      {
00070          this.edges1 = new ArrayList<>();
00071          this.edges2 = new ArrayList<>();
00072          for (int i = 0; i < this.nodes; i++) {
00073              for (int j = 0; j < this.nodes; j++) {
00074                  if (this.distanceMatrix[i][j] != Integer.MAX_VALUE) {
00075                      this.edges1.add(i);
00076                      this.edges2.add(j);
00077                  }
00078              }
00079          }
00080      }
  
```

Gráfico de llamadas a esta función:



#### 7.1.3.2. getDistances()

```
int[] es.ull.esit.utilities.BellmanFord.getDistances () [inline]
```

Obtiene el array de distancias calculadas desde el nodo origen.

Devuelve

Un array de enteros con las distancias.

Definición en la línea 86 del archivo [BellmanFord.java](#).

```

00086      {
00087          return this.distances;
00088      }
  
```

### 7.1.3.3. `getValue()`

```
int es.ull.esit.utilities.BellmanFord.getValue () [inline]
```

Obtiene el valor (coste) del camino más corto calculado.

Devuelve

El coste del camino.

Definición en la línea 94 del archivo [BellmanFord.java](#).

```
00094         {
00095             return this.value;
00096         }
```

### 7.1.3.4. `solve()`

```
void es.ull.esit.utilities.BellmanFord.solve () [inline]
```

Ejecuta el algoritmo de Bellman-Ford.

Inicializa las distancias y predecesores, y luego relaja las aristas repetidamente para encontrar los caminos más cortos. Finalmente, reconstruye el camino desde el destino hasta el origen utilizando el array de predecesores.

Definición en la línea 104 del archivo [BellmanFord.java](#).

```
00104         {
00105             int numEdges = this.edges1.size();
00106             int[] predecessor = new int[this.nodes];
00107             this.distances = new int[this.nodes];
00108             for (int i = 0; i < this.nodes; i++) {
00109                 this.distances[i] = BellmanFord.INFINITY;
00110                 predecessor[i] = -1;
00111             }
00112             this.distances[0] = 0;
00113             for (int i = 0; i < (this.nodes - 1); i++) {
00114                 for (int j = 0; j < numEdges; j++) {
00115                     int u = this.edges1.get(j);
00116                     int v = this.edges2.get(j);
00117                     if (this.distances[v] > this.distances[u] + this.distanceMatrix[u][v]) {
00118                         this.distances[v] = this.distances[u] + this.distanceMatrix[u][v];
00119                         predecessor[v] = u;
00120                     }
00121                 }
00122             }
00123             this.path.add(this.nodes - 1);
00124             int pred = predecessor[this.nodes - 1];
00125             while (pred != -1) {
00126                 this.path.add(pred);
00127                 pred = predecessor[pred];
00128             }
00129             this.value = -this.distances[this.nodes - 1];
00130         }
```

Gráfico de llamadas de esta función:



### 7.1.4. Documentación de datos miembro

#### 7.1.4.1. distanceMatrix

```
final int [][] es.ull.esit.utilities.BellmanFord.distanceMatrix [private]
```

Matriz de adyacencia que representa las distancias (pesos) entre nodos.

Definición en la línea [24](#) del archivo [BellmanFord.java](#).

#### 7.1.4.2. distances

```
int [] es.ull.esit.utilities.BellmanFord.distances = null [private]
```

Array para almacenar las distancias más cortas desde el origen a cada nodo.

Definición en la línea [44](#) del archivo [BellmanFord.java](#).

#### 7.1.4.3. edges1

```
ArrayList<Integer> es.ull.esit.utilities.BellmanFord.edges1 = null [private]
```

Lista de nodos de origen para cada arista del grafo.

Definición en la línea [28](#) del archivo [BellmanFord.java](#).

#### 7.1.4.4. edges2

```
ArrayList<Integer> es.ull.esit.utilities.BellmanFord.edges2 = null [private]
```

Lista de nodos de destino para cada arista del grafo.

Definición en la línea [32](#) del archivo [BellmanFord.java](#).

#### 7.1.4.5. INFINITY

```
final int es.ull.esit.utilities.BellmanFord.INFINITY = 999999 [static], [private]
```

Valor que representa el infinito, usado para inicializar distancias.

Definición en la línea [20](#) del archivo [BellmanFord.java](#).

#### 7.1.4.6. nodes

```
final int es.ull.esit.utilities.BellmanFord.nodes [private]
```

Número total de nodos en el grafo.

Definición en la línea [36](#) del archivo [BellmanFord.java](#).

### 7.1.4.7. path

```
final ArrayList<Integer> es.ull.esit.utilities.BellmanFord.path [private]
```

Lista para almacenar el camino más corto encontrado.

Definición en la línea 40 del archivo [BellmanFord.java](#).

### 7.1.4.8. value

```
int es.ull.esit.utilities.BellmanFord.value [private]
```

El valor (coste) del camino más corto encontrado.

Definición en la línea 48 del archivo [BellmanFord.java](#).

La documentación de esta clase está generada del siguiente archivo:

- C:/Users/Kefle/Desktop/ExpositoTOP/src/es/ull/esit/utilities/[BellmanFord.java](#)

## 7.2. Referencia de la clase es.ull.esit.utilities.ExpositoUtilities

Proporciona un conjunto de herramientas estáticas de propósito general.

### Métodos públicos estáticos

- static void [printFile](#) (String file)
 

*Imprime el contenido de un archivo en la consola.*
- static String [simplifyString](#) (String string)
 

*Simplifica una cadena de texto.*
- static double[ ][ ] [multiplyMatrices](#) (double a[ ][ ], double b[ ][ ])
 

*Multiplica dos matrices de dobles.*
- static void [writeTextToFile](#) (String file, String text) throws IOException
 

*Escribe un texto en un archivo.*
- static String [getFormat](#) (String string)
 

*Formatea una cadena que representa un número.*
- static String [getFormat](#) (double value)
 

*Formatea un valor doble a una cadena con tres decimales.*
- static String [getFormat](#) (double value, int zeros)
 

*Formatea un valor doble a una cadena con un número específico de decimales.*
- static String [getFormat](#) (String string, int width)
 

*Formatea una cadena para que ocupe un ancho fijo, con alineación a la derecha.*
- static String [getFormat](#) (String string, int width, int alignment)
 

*Formatea una cadena para que ocupe un ancho fijo con una alineación específica.*
- static String [getFormat](#) (ArrayList< String > strings, int width)
 

*Formatea una lista de cadenas, cada una con un ancho fijo.*
- static String [getFormat](#) (ArrayList< Integer > strings)
 

*Formatea una lista de enteros con el ancho de columna por defecto.*

- static String [getFormat](#) (String[] strings, int width)
 

*Formatea un array de cadenas, cada una con un ancho fijo.*
- static String [getFormat](#) (String[][] matrixStrings, int width)
 

*Formatea una matriz de cadenas en una tabla.*
- static String [getFormat](#) (String[] strings)
 

*Formatea un array de cadenas con el ancho de columna por defecto.*
- static String [getFormat](#) (String[] strings, int[] width)
 

*Formatea un array de cadenas con anchos de columna individuales.*
- static String [getFormat](#) (String[] strings, int[] width, int[] alignment)
 

*Formatea un array de cadenas con anchos y alineaciones individuales.*
- static boolean [isInteger](#) (String str)
 

*Comprueba si una cadena puede ser parseada como un entero.*
- static boolean [isDouble](#) (String str)
 

*Comprueba si una cadena puede ser parseada como un doble.*
- static boolean [isAcyclic](#) (int[][] distanceMatrix)
 

*Comprueba si un grafo representado por una matriz de adyacencia es acíclico.*
- static boolean [thereIsPath](#) (int[][] distanceMatrix, int node)
 

*Comprueba si existe un camino desde un nodo de vuelta a sí mismo (un ciclo).*

### Atributos públicos estáticos

- static final int [DEFAULT\\_COLUMN\\_WIDTH](#) = 10
 

*Ancho de columna por defecto para el formateo de texto.*
- static final int [ALIGNMENT\\_LEFT](#) = 1
 

*Constante para alineación a la izquierda.*
- static final int [ALIGNMENT\\_RIGHT](#) = 2
 

*Constante para alineación a la derecha.*

### Métodos privados estáticos

- static int [getFirstAppearance](#) (int[] vector, int element)
 

*Encuentra la primera aparición de un elemento en un vector.*

## 7.2.1. Descripción detallada

Proporciona un conjunto de herramientas estáticas de propósito general.

Incluye métodos para formateo de cadenas, operaciones con archivos, validación de tipos de datos y manipulación de matrices.

Definición en la línea [27](#) del archivo [ExpositoUtilities.java](#).

## 7.2.2. Documentación de funciones miembro

### 7.2.2.1. [getFirstAppearance\(\)](#)

```
int es.ull.esit.utilities.ExpositoUtilities.getFirstAppearance (
    int[] vector,
    int element) [inline], [static], [private]
```

Encuentra la primera aparición de un elemento en un vector.

**Parámetros**

|                          |                |                                   |
|--------------------------|----------------|-----------------------------------|
| <input type="checkbox"/> | <i>vector</i>  | El array de enteros donde buscar. |
| <input type="checkbox"/> | <i>element</i> | El elemento a buscar.             |

**Devuelve**

El índice de la primera aparición, o -1 si no se encuentra.

Definición en la línea 48 del archivo [ExpositoUtilities.java](#).

```
00048
00049     for (int i = 0; i < vector.length; i++) {
00050         if (vector[i] == element) {
00051             return i;
00052         }
00053     }
00054     return -1;
00055 }
```

**7.2.2.2. `getFormat()` [1/12]**

```
String es.ull.esit.utilities.ExpositoUtilities.getFormat (
    ArrayList< Integer > strings) [inline], [static]
```

Formatea una lista de enteros con el ancho de columna por defecto.

**Parámetros**

|                          |                |                      |
|--------------------------|----------------|----------------------|
| <input type="checkbox"/> | <i>strings</i> | La lista de enteros. |
|--------------------------|----------------|----------------------|

**Devuelve**

Una única cadena con todos los elementos formateados.

Definición en la línea 227 del archivo [ExpositoUtilities.java](#).

```
00227
00228     String format = "";
00229     for (int i = 0; i < strings.size(); i++) {
00230         format += "%" + (i + 1) + "$" + DEFAULT_COLUMN_WIDTH + "s";
00231     }
00232     Integer[] data = new Integer[strings.size()];
00233     for (int t = 0; t < strings.size(); t++) {
00234         data[t] = strings.get(t);
00235     }
00236     return String.format(format, (Object[]) data);
00237 }
```

**7.2.2.3. `getFormat()` [2/12]**

```
String es.ull.esit.utilities.ExpositoUtilities.getFormat (
    ArrayList< String > strings,
    int width) [inline], [static]
```

Formatea una lista de cadenas, cada una con un ancho fijo.

## Parámetros

|                                  |                            |
|----------------------------------|----------------------------|
| <input type="checkbox"/> strings | La lista de cadenas.       |
| <input type="checkbox"/> width   | El ancho para cada cadena. |

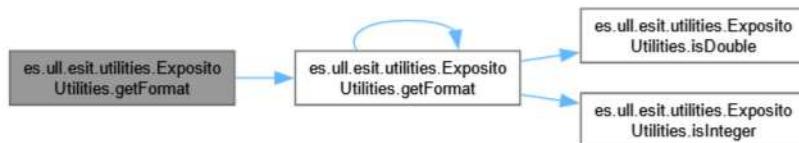
## Devuelve

Una única cadena con todos los elementos formateados y concatenados.

Definición en la línea 210 del archivo [ExpositoUtilities.java](#).

```
00210     String format = "";
00211     for (int i = 0; i < strings.size(); i++) {
00212         format += "%" + (i + 1) + "$" + width + "s";
00213     }
00214     String[] data = new String[strings.size()];
00215     for (int t = 0; t < strings.size(); t++) {
00216         data[t] = "" + ExpositoUtilities.getFormat(strings.get(t));
00217     }
00218     return String.format(format, (Object[]) data);
00219 }
00220 }
```

Gráfico de llamadas de esta función:

7.2.2.4. `getFormat()` [3/12]

```
String es.ull.esit.utilities.ExpositoUtilities.getFormat (
    double value) [inline], [static]
```

Formatea un valor doble a una cadena con tres decimales.

## Parámetros

|                                |                       |
|--------------------------------|-----------------------|
| <input type="checkbox"/> value | El valor a formatear. |
|--------------------------------|-----------------------|

## Devuelve

La cadena formateada.

Definición en la línea 148 del archivo [ExpositoUtilities.java](#).

```
00148     {
00149         DecimalFormat decimalFormatter = new DecimalFormat("0.000");
00150         DecimalFormatSymbols symbols = new DecimalFormatSymbols();
00151         symbols.setDecimalSeparator('.');
00152         decimalFormatter.setDecimalFormatSymbols(symbols);
00153         return decimalFormatter.format(value);
00154     }
```

### 7.2.2.5. `getFormat()` [4/12]

```
String es.ull.esit.utilities.ExpositoUtilities.getFormat (
    double value,
    int zeros)  [inline], [static]
```

Formatea un valor doble a una cadena con un número específico de decimales.

## Parámetros

|                          |              |                                 |
|--------------------------|--------------|---------------------------------|
| <input type="checkbox"/> | <b>value</b> | El valor a formatear.           |
| <input type="checkbox"/> | <b>zeros</b> | El número de dígitos decimales. |

## Devuelve

La cadena formateada.

Definición en la línea 162 del archivo [ExpositoUtilities.java](#).

```
00162
00163     String format = "0.";
00164     for (int i = 0; i < zeros; i++) {
00165         format += "0";
00166     }
00167     DecimalFormat decimalFormatter = new DecimalFormat(format);
00168     DecimalFormatSymbols symbols = new DecimalFormatSymbols();
00169     symbols.setDecimalSeparator('.');
00170     decimalFormatter.setDecimalFormatSymbols(symbols);
00171     return decimalFormatter.format(value);
00172 }
```

**7.2.2.6. getFormat() [5/12]**

```
String es.ull.esit.utilities.ExpositoUtilities.getFormat (
    String string) [inline], [static]
```

Formatea una cadena que representa un número.

Si la cadena es un número doble, lo formatea con tres decimales.

## Parámetros

|                          |               |                        |
|--------------------------|---------------|------------------------|
| <input type="checkbox"/> | <b>string</b> | La cadena a formatear. |
|--------------------------|---------------|------------------------|

## Devuelve

La cadena formateada.

Definición en la línea 133 del archivo [ExpositoUtilities.java](#).

```
00133
00134     if (!ExpositoUtilities.isInteger(string)) {
00135         if (ExpositoUtilities.isDouble(string)) {
00136             double value = Double.parseDouble(string);
00137             string = ExpositoUtilities.getFormat(value);
00138         }
00139     }
00140     return string;
00141 }
```

Gráfico de llamadas de esta función:

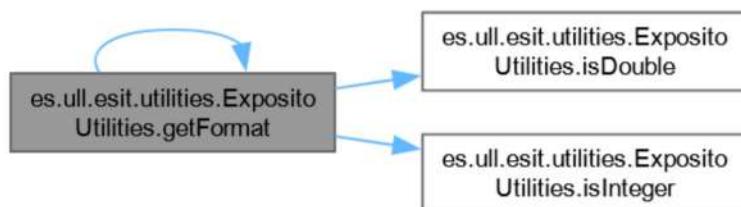
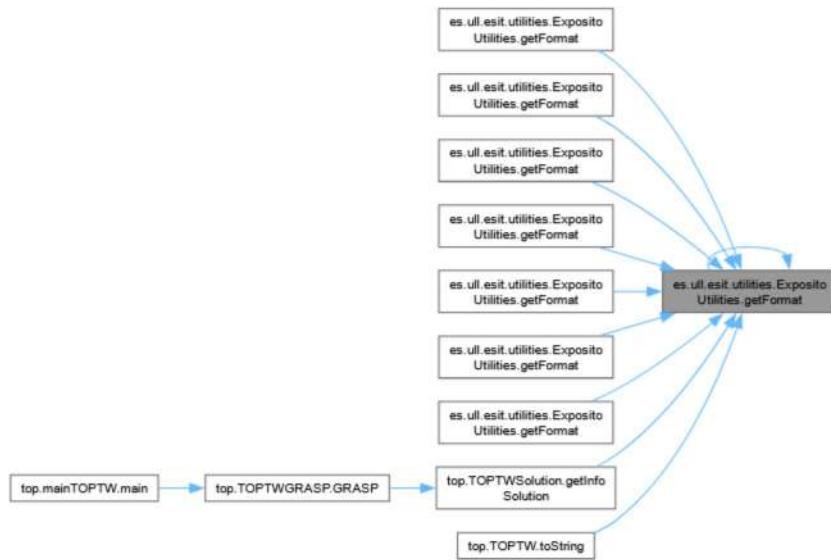


Gráfico de llamadas a esta función:



### 7.2.2.7. `getFormat()` [6/12]

```
String es.ull.esit.utilities.ExpositoUtilities.getFormat (
    String string,
    int width) [inline], [static]
```

Formatea una cadena para que ocupe un ancho fijo, con alineación a la derecha.

#### Parámetros

|                     |   |
|---------------------|---|
| <code>string</code> | La cadena a formatear.                  |
| <code>width</code>  | El ancho total de la cadena resultante. |

#### Devuelve

La cadena formateada.

Definición en la línea 180 del archivo [ExpositoUtilities.java](#).

```
00180
00181     return ExpositoUtilities.getFormat(string, width, ExpositoUtilities.ALIGNMENT_RIGHT);
00182 }
```

Gráfico de llamadas de esta función:



### 7.2.2.8. `getFormat()` [7/12]

```
String es.ull.esit.utilities.ExpositoUtilities.getFormat (
    String string,
    int width,
    int alignment) [inline], [static]
```

Formatea una cadena para que ocupe un ancho fijo con una alineación específica.

#### Parámetros

|                        |   |
|------------------------|---|
| <code>string</code>    | La cadena a formatear.                            |
| <code>width</code>     | El ancho total.                                   |
| <code>alignment</code> | La alineación (ALIGNMENT_LEFT o ALIGNMENT_RIGHT). |

#### Devuelve

La cadena formateada.

Definición en la línea 191 del archivo [ExpositoUtilities.java](#).

```
00191
00192     String format = "";
00193     if (alignment == ExpositoUtilities.ALIGNMENT_LEFT) {
00194         format = "%" + width + "s";
00195     } else {
00196         format = "%" + 1 + "$" + width + "s";
00197     }
00198     DecimalFormatSymbols symbols = new DecimalFormatSymbols();
00199     symbols.setDecimalSeparator('.');
00200     String[] data = new String[]{string};
00201     return String.format(format, (Object[]) data);
00202 }
```

### 7.2.2.9. `getFormat()` [8/12]

```
String es.ull.esit.utilities.ExpositoUtilities.getFormat (
    String[] strings) [inline], [static]
```

Formatea un array de cadenas con el ancho de columna por defecto.

#### Parámetros

|                      |                      |
|----------------------|----------------------|
| <code>strings</code> | El array de cadenas. |
|----------------------|----------------------|

#### Devuelve

Una única cadena con todos los elementos formateados.

Definición en la línea 280 del archivo [ExpositoUtilities.java](#).

```
00280
00281     int[] alignment = new int[strings.length];
00282     Arrays.fill(alignment, ExpositoUtilities.ALIGNMENT_RIGHT);
00283     int[] widths = new int[strings.length];
00284     Arrays.fill(widths, ExpositoUtilities.DEFAULT_COLUMN_WIDTH);
00285     return ExpositoUtilities.getFormat(strings, widths, alignment);
00286 }
```

Gráfico de llamadas de esta función:



#### 7.2.2.10. `getFormat()` [9/12]

```
String es.ull.esit.utilities.ExpositoUtilities.getFormat (
    String[] strings,
    int width)  [inline], [static]
```

Formatea un array de cadenas, cada una con un ancho fijo.

Parámetros

|                      |                            |
|----------------------|----------------------------|
| <code>strings</code> | El array de cadenas.       |
| <code>width</code>   | El ancho para cada cadena. |

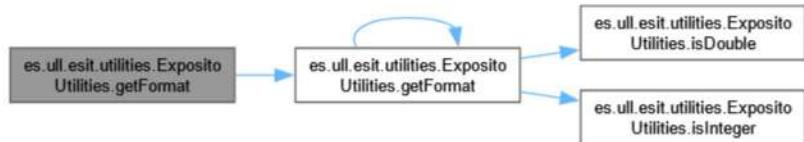
Devuelve

Una única cadena con todos los elementos formateados.

Definición en la línea 245 del archivo [ExpositoUtilities.java](#).

```
00245     int[] alignment = new int[strings.length];
00246     Arrays.fill(alignment, ExpositoUtilities.ALIGNMENT_RIGHT);
00247     int[] widths = new int[strings.length];
00248     Arrays.fill(widths, width);
00249     return ExpositoUtilities.getFormat(strings, widths, alignment);
00250   }
```

Gráfico de llamadas de esta función:



#### 7.2.2.11. `getFormat()` [10/12]

```
String es.ull.esit.utilities.ExpositoUtilities.getFormat (
    String[] strings,
    int[] width)  [inline], [static]
```

Formatea un array de cadenas con anchos de columna individuales.

## Parámetros

|                      |  |
|----------------------|--|
| <code>strings</code> | El array de cadenas.                                     |
| <code>width</code>   | Un array con el ancho para cada columna correspondiente. |

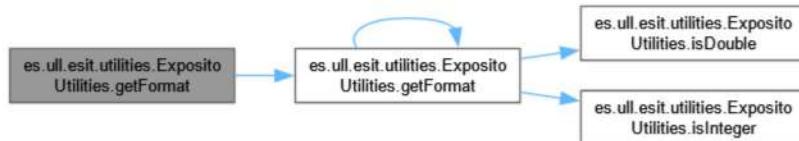
## Devuelve

Una única cadena con todos los elementos formateados.

Definición en la línea 294 del archivo [ExpositoUtilities.java](#).

```
00294     int[] alignment = new int[strings.length];
00295     Arrays.fill(alignment, ExpositoUtilities.ALIGNMENT_RIGHT);
00296     return ExpositoUtilities.getFormat(strings, width, alignment);
00298 }
```

Gráfico de llamadas de esta función:

7.2.2.12. `getFormat()` [11/12]

```
String es.ull.esit.utilities.ExpositoUtilities.getFormat (
    String[] strings,
    int[] width,
    int[] alignment) [inline], [static]
```

Formatea un array de cadenas con anchos y alineaciones individuales.

## Parámetros

|                        |   |
|------------------------|---|
| <code>strings</code>   | El array de cadenas.                          |
| <code>width</code>     | Un array con el ancho para cada columna.      |
| <code>alignment</code> | Un array con la alineación para cada columna. |

## Devuelve

Una única cadena con todos los elementos formateados.

Definición en la línea 307 del archivo [ExpositoUtilities.java](#).

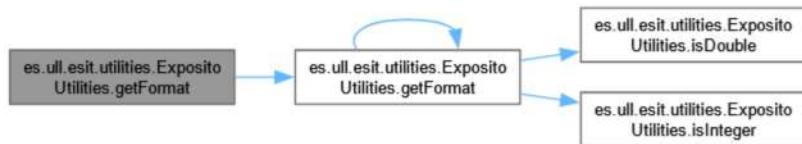
```
00307     String format = "";
00308     for (int i = 0; i < strings.length; i++) {
00309         if (alignment[i] == ExpositoUtilities.ALIGNMENT_LEFT) {
00310             format += "%" + (i + 1) + "$-" + width[i] + "s";
00311         } else {
00312             format += "%" + (i + 1) + "$" + width[i] + "s";
```

```

00314         }
00315     }
00316     String[] data = new String[strings.length];
00317     for (int t = 0; t < strings.length; t++) {
00318         data[t] = "" + ExpositoUtilities.getFormat(strings[t]);
00319     }
00320     return String.format(format, (Object[]) data);
00321 }

```

Gráfico de llamadas de esta función:



### 7.2.2.13. `getFormat()` [12/12]

```

String es.ull.esit.utilities.ExpositoUtilities.getFormat (
    String matrixStrings[][],
    int width) [inline], [static]

```

Formatea una matriz de cadenas en una tabla.

Parámetros

|                            |                             |
|----------------------------|-----------------------------|
| <code>matrixStrings</code> | La matriz de cadenas.       |
| <code>width</code>         | El ancho para cada columna. |

Devuelve

Una cadena de texto que representa la tabla formateada.

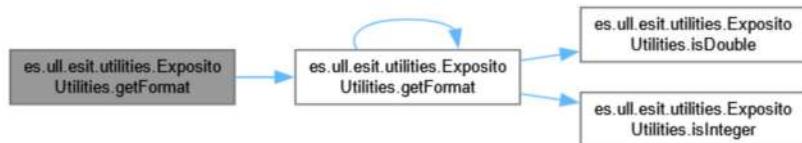
Definición en la línea 259 del archivo [ExpositoUtilities.java](#).

```

00259
00260     String result = "";
00261     for (int i = 0; i < matrixStrings.length; i++) {
00262         String[] strings = matrixStrings[i];
00263         int[] alignment = new int[strings.length];
00264         Arrays.fill(alignment, ExpositoUtilities.ALIGNMENT_RIGHT);
00265         int[] widths = new int[strings.length];
00266         Arrays.fill(widths, width);
00267         result += ExpositoUtilities.getFormat(strings, widths, alignment);
00268         if (i < (matrixStrings.length - 1)) {
00269             result += "\n";
00270         }
00271     }
00272     return result;
00273 }

```

Gráfico de llamadas de esta función:



### 7.2.2.14. isAcyclic()

```
boolean es.ull.esit.utilities.ExpositoUtilities.isAcyclic (
    int distanceMatrix[][][])
```

Comprueba si un grafo representado por una matriz de adyacencia es acíclico.

#### Parámetros

|                             |                          |
|-----------------------------|--------------------------|
| <code>distanceMatrix</code> | La matriz de adyacencia. |
|-----------------------------|--------------------------|

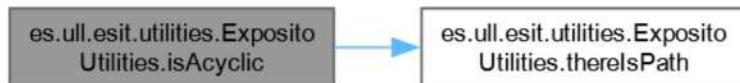
#### Devuelve

`true` si es acíclico, `false` si se detecta un ciclo.

Definición en la línea 356 del archivo [ExpositoUtilities.java](#).

```
00356
00357     int numRealTasks = distanceMatrix.length - 2;
00358     int node = 1;
00359     boolean acyclic = true;
00360     while (acyclic && node <= numRealTasks) {
00361         if (ExpositoUtilities.thereIsPath(distanceMatrix, node)) {
00362             return false;
00363         }
00364         node++;
00365     }
00366     return true;
00367 }
```

Gráfico de llamadas de esta función:



### 7.2.2.15. isDouble()

```
boolean es.ull.esit.utilities.ExpositoUtilities.isDouble (
    String str)
```

Comprueba si una cadena puede ser parseada como un doble.

#### Parámetros

|                  |                        |
|------------------|------------------------|
| <code>str</code> | La cadena a comprobar. |
|------------------|------------------------|

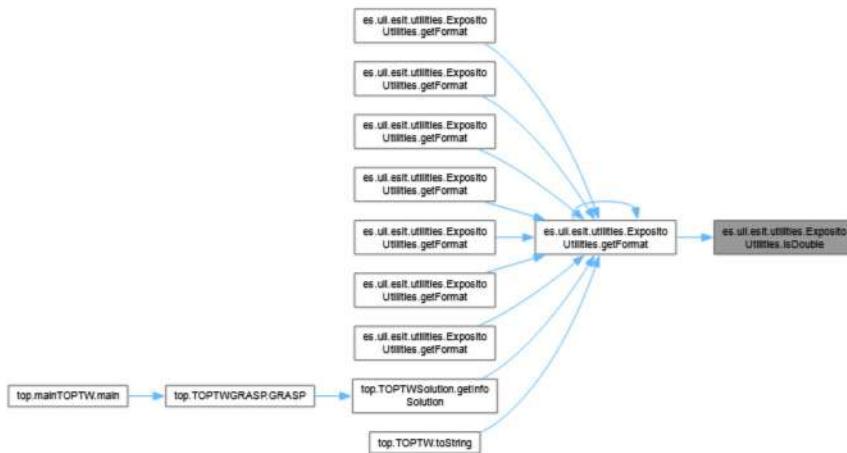
**Devuelve**

`true` si es un doble, `false` en caso contrario.

Definición en la línea 342 del archivo [ExpositoUtilities.java](#).

```
00342     {
00343         try {
00344             Double.parseDouble(str);
00345             return true;
00346         } catch (Exception e) {
00347         }
00348         return false;
00349     }
```

Gráfico de llamadas a esta función:

**7.2.2.16. isInteger()**

```
boolean es.ul.esit.utilities.ExpositoUtilities.isInteger (
    String str) [inline], [static]
```

Comprueba si una cadena puede ser parseada como un entero.

**Parámetros**

|                          |                  |                        |
|--------------------------|------------------|------------------------|
| <input type="checkbox"/> | <code>str</code> | La cadena a comprobar. |
|--------------------------|------------------|------------------------|

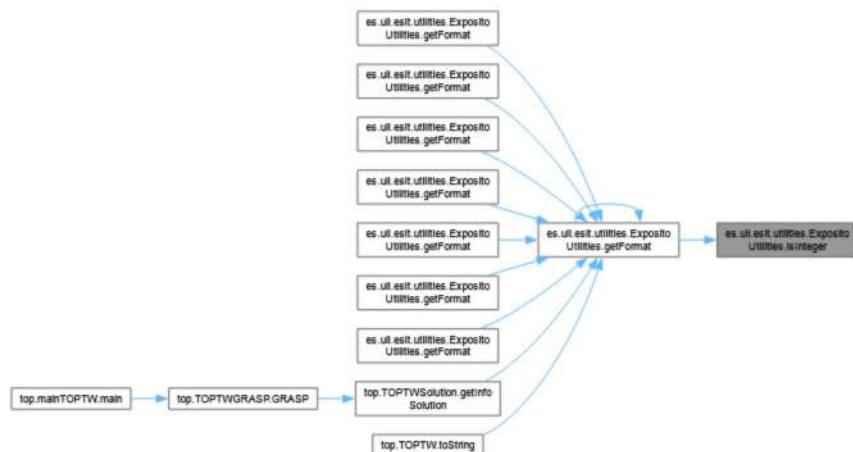
**Devuelve**

`true` si es un entero, `false` en caso contrario.

Definición en la línea 328 del archivo [ExpositoUtilities.java](#).

```
00328     {
00329         try {
00330             Integer.parseInt(str);
00331             return true;
00332         } catch (Exception e) {
00333         }
00334         return false;
00335     }
```

Gráfico de llamadas a esta función:



#### 7.2.2.17. multiplyMatrices()

```
double[][][] es.ul.evit.utilities.ExpositoUtilities.multiplyMatrices (
```

|        |                            |
|--------|----------------------------|
| double | a[][][],                   |
| double | b[][][] [inline], [static] |

Multiplica dos matrices de dobles.

## Parámetros

|          |                                |
|----------|--------------------------------|
| <i>a</i> | La primera matriz (izquierda). |
| <i>b</i> | La segunda matriz (derecha).   |

## Devuelve

La matriz resultante de la multiplicación, o `null` si las dimensiones son incompatibles.

Definición en la línea 94 del archivo [ExpositoUtilities.java](#).

```
00094
00095     if (a.length == 0) {
00096         return new double[0][0];
00097     }
00098     if (a[0].length != b.length) {
00099         return null;
00100     }
00101     int n = a[0].length;
00102     int m = a.length;
00103     int p = b[0].length;
00104     double ans[][] = new double[m][p];
00105     for (int i = 0; i < m; i++) {
00106         for (int j = 0; j < p; j++) {
00107             for (int k = 0; k < n; k++) {
00108                 ans[i][j] += a[i][k] * b[k][j];
00109             }
00110         }
00111     }
00112     return ans;
00113 }
```

### 7.2.2.18. `printFile()`

```
void es.uill.esit.utilities.ExpositoUtilities.printFile (
    String file) [inline], [static]
```

Imprime el contenido de un archivo en la consola.

#### Parámetros

|                          |                   |                                |
|--------------------------|-------------------|--------------------------------|
| <input type="checkbox"/> | <code>file</code> | La ruta al archivo a imprimir. |
|--------------------------|-------------------|--------------------------------|

Definición en la línea 61 del archivo [ExpositoUtilities.java](#).

```
00061      {
00062          try (BufferedReader reader = new BufferedReader(new FileReader(file))) {
00063              String line = reader.readLine();
00064              while (line != null) {
00065                  System.out.println(line);
00066                  line = reader.readLine();
00067              }
00068          } catch (IOException ex) {
00069              Logger.getLogger(ExpositoUtilities.class.getName()).log(Level.SEVERE, null, ex);
00070          }
00071      }
```

### 7.2.2.19. `simplifyString()`

```
String es.uill.esit.utilities.ExpositoUtilities.simplifyString (
    String string) [inline], [static]
```

Simplifica una cadena de texto.

Reemplaza tabulaciones y múltiples espacios por un único espacio, y elimina espacios al inicio y al final.

#### Parámetros

|                          |                     |                          |
|--------------------------|---------------------|--------------------------|
| <input type="checkbox"/> | <code>string</code> | La cadena a simplificar. |
|--------------------------|---------------------|--------------------------|

#### Devuelve

La cadena simplificada.

Definición en la línea 79 del archivo [ExpositoUtilities.java](#).

```
00079      {
00080          string = string.replaceAll("\t", " ");
00081          for (int i = 0; i < 50; i++) {
00082              string = string.replaceAll("  ", " ");
00083          }
00084          string = string.trim();
00085      }
00086      return string;
00087  }
```

Gráfico de llamadas a esta función:



### 7.2.2.20. therelsPath()

```
boolean es.ull.esit.utilities.ExpositoUtilities.therelsPath (
    int distanceMatrix[][],
    int node) [inline], [static]
```

Comprueba si existe un camino desde un nodo de vuelta a sí mismo (un ciclo).

#### Parámetros

|                             |  |
|-----------------------------|--|
| <code>distanceMatrix</code> | La matriz de adyacencia.                       |
| <code>node</code>           | El nodo de inicio y fin del ciclo a comprobar. |

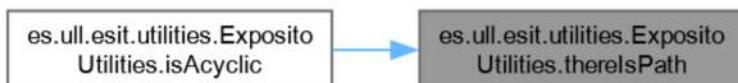
#### Devuelve

`true` si se encuentra un ciclo, `false` en caso contrario.

Definición en la línea 375 del archivo [ExpositoUtilities.java](#).

```
00375     HashSet<Integer> visits = new HashSet<>();
00376     HashSet<Integer> noVisits = new HashSet<>();
00377     for (int i = 0; i < distanceMatrix.length; i++) {
00378         if (i != node) {
00379             noVisits.add(i);
00380         }
00381     }
00382     visits.add(node);
00383     while (!visits.isEmpty()) {
00384         Iterator<Integer> it = visits.iterator();
00385         int toCheck = it.next();
00386         visits.remove(toCheck);
00387         for (int i = 0; i < distanceMatrix.length; i++) {
00388             if (toCheck != i && distanceMatrix[toCheck][i] != Integer.MAX_VALUE) {
00389                 if (i == node) {
00390                     return true;
00391                 }
00392                 if (noVisits.contains(i)) {
00393                     noVisits.remove(i);
00394                     visits.add(i);
00395                 }
00396             }
00397         }
00398     }
00399     return false;
00400 }
```

Gráfico de llamadas a esta función:



### 7.2.2.21. writeTextToFile()

```
void es.ull.esit.utilities.ExpositoUtilities.writeTextToFile (
    String file,
    String text) throws IOException [inline], [static]
```

Escribe un texto en un archivo.

### Parámetros

|                   |                          |
|-------------------|--------------------------|
| <code>file</code> | La ruta al archivo.      |
| <code>text</code> | El contenido a escribir. |

### Excepciones

|                          |                            |
|--------------------------|----------------------------|
| <code>IOException</code> | Si ocurre un error de E/S. |
|--------------------------|----------------------------|

Definición en la línea 121 del archivo [ExpositoUtilities.java](#).

```
00121
00122     try (BufferedWriter writer = new BufferedWriter(new FileWriter(file))) {
00123         writer.write(text);
00124     }
00125 }
```

## 7.2.3. Documentación de datos miembro

### 7.2.3.1. ALIGNMENT\_LEFT

`final int es.ull.esit.utilities.ExpositoUtilities.ALIGNMENT_LEFT = 1 [static]`

Constante para alineación a la izquierda.

Definición en la línea 36 del archivo [ExpositoUtilities.java](#).

### 7.2.3.2. ALIGNMENT\_RIGHT

`final int es.ull.esit.utilities.ExpositoUtilities.ALIGNMENT_RIGHT = 2 [static]`

Constante para alineación a la derecha.

Definición en la línea 40 del archivo [ExpositoUtilities.java](#).

### 7.2.3.3. DEFAULT\_COLUMN\_WIDTH

`final int es.ull.esit.utilities.ExpositoUtilities.DEFAULT_COLUMN_WIDTH = 10 [static]`

Ancho de columna por defecto para el formateo de texto.

Definición en la línea 32 del archivo [ExpositoUtilities.java](#).

La documentación de esta clase está generada del siguiente archivo:

- C:/Users/Kefle/Desktop/ExpositoTOP/src/es/ull/esit/utilities/[ExpositoUtilities.java](#)

## 7.3. Referencia de la clase top.mainTOPTW

Clase principal que contiene el método `main` para ejecutar el experimento.

## Métodos públicos estáticos

- static void **main** (String[ ] args)

*Método principal que ejecuta el solver TOPTW GRASP.*

### 7.3.1. Descripción detallada

Clase principal que contiene el método `main` para ejecutar el experimento.

Esta clase define un conjunto de instancias de prueba, las lee una por una, y ejecuta el algoritmo GRASP con diferentes tamaños de RCL para cada una.

Definición en la línea 13 del archivo [mainTOPTW.java](#).

### 7.3.2. Documentación de funciones miembro

#### 7.3.2.1. main()

```
void top.mainTOPTW.main (
    String[] args) [inline], [static]
```

Método principal que ejecuta el solver **TOPTW** GRASP.

Itera sobre una lista predefinida de archivos de instancia, carga cada problema, y ejecuta el algoritmo GRASP varias veces con diferentes tamaños de RCL (3, 5, 7) para cada instancia, imprimiendo los resultados en la consola.

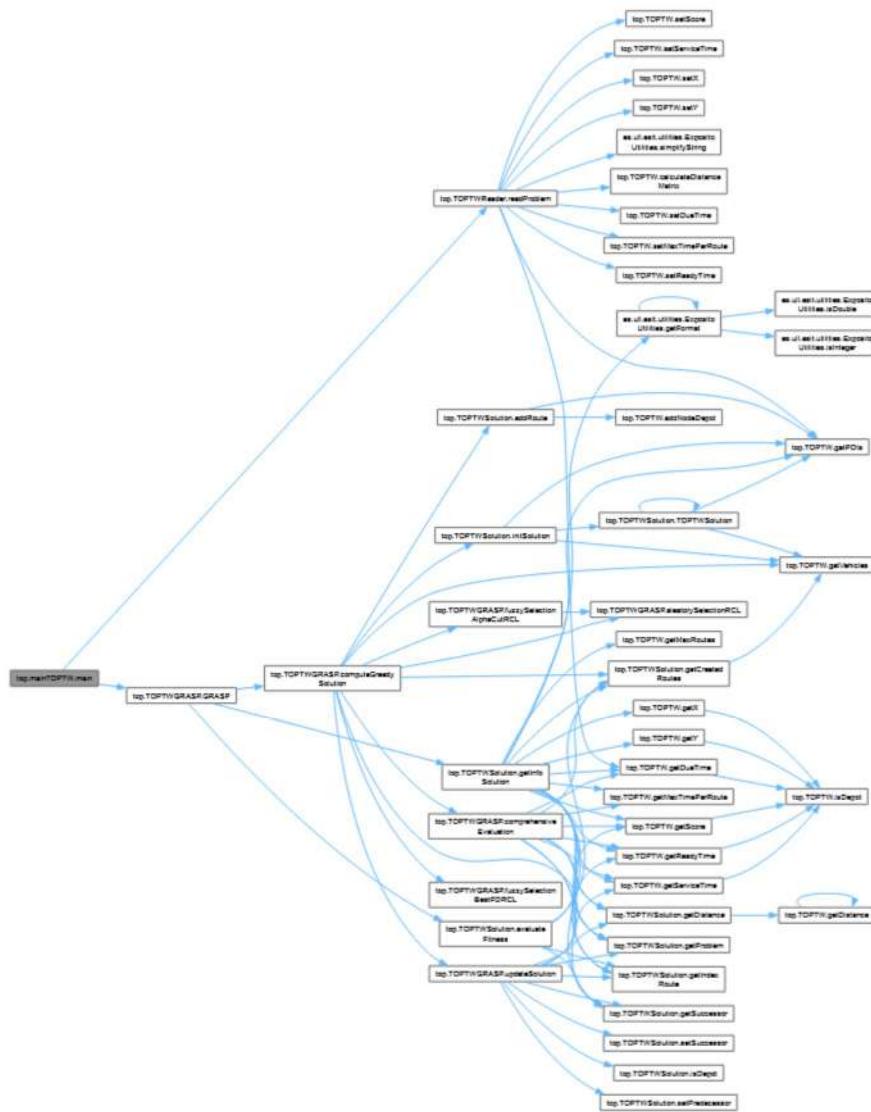
#### Parámetros

|                                      |  |
|--------------------------------------|--|
| <input type="checkbox"/> <b>args</b> | Argumentos de la línea de comandos (no se utilizan). |
|--------------------------------------|--|

Definición en la línea 22 del archivo [mainTOPTW.java](#).

```
00022
00023
00024     String[] instances = new String[29];
00025
00026     instances[0] = "c101.txt"; instances[3] = "c104.txt"; instances[6] = "c107.txt";
00027     instances[1] = "c102.txt"; instances[4] = "c105.txt"; instances[7] = "c108.txt";
00028     instances[2] = "c103.txt"; instances[5] = "c106.txt"; instances[8] = "c109.txt";
00029
00030     instances[9] = "r101.txt"; instances[12] = "r104.txt"; instances[15] = "r107.txt";
00031     instances[10] = "r102.txt"; instances[13] = "r105.txt"; instances[16] = "r108.txt";
00032     instances[11] = "r103.txt"; instances[14] = "r106.txt"; instances[17] = "r109.txt";
00033     instances[18] = "r110.txt"; instances[19] = "r111.txt"; instances[20] = "r112.txt";
00034
00035     instances[21] = "rc101.txt"; instances[24] = "rc104.txt"; instances[27] = "rc107.txt";
00036     instances[22] = "rc102.txt"; instances[25] = "rc105.txt"; instances[28] = "rc108.txt";
00037     instances[23] = "rc103.txt"; instances[26] = "rc106.txt";
00038
00039     for(int i = 0; i < instances.length; i++) {
00040         String INSTANCE = "Instances/TOPTW/" + instances[i];
00041         TOPTW problem = TOPTWReader.readProblem(INSTANCE);
00042         TOPTWSolution solution = new TOPTWSolution(problem);
00043         TOPTWGRASP grasp = new TOPTWGRASP(solution);
00044
00045         System.out.println(" --> Instance: " + instances[i]);
00046         grasp.GRASP(10000, 3);
00047         grasp.GRASP(10000, 5);
00048         grasp.GRASP(10000, 7);
00049         System.out.println("");
00050     }
00051 }
```

Gráfico de llamadas de esta función:



La documentación de esta clase está generada del siguiente archivo:

- [C:/Users/Kefle/Desktop/ExpositoTOP/src/top/mainTOPTW.java](#)

## 7.4. Referencia de la plantilla de la clase es.ull.esit.utils.Pair< F, S >

Una clase genérica que almacena un par de objetos inmutables.

### Métodos públicos

- **Pair (F first, S second)**  
Constructor para crear un par.
- **boolean equals (Object o)**  
Compara este par con otro objeto para ver si son iguales.
- **int hashCode ()**  
Calcula el código hash para el par.

## Métodos públicos estáticos

- static< A, B > [Pair< A, B >.create \(A a, B b\)](#)

*Método de fábrica estático para crear una instancia de [Pair](#).*

## Atributos públicos

- final F [first](#)  
*El primer elemento del par.*
- final S [second](#)  
*El segundo elemento del par.*

### 7.4.1. Descripción detallada

Una clase genérica que almacena un par de objetos inmutables.

Proporciona una forma conveniente de tratar dos objetos como una única unidad. Los objetos `first` y `second` son públicos y finales.

#### Parámetros

|                        |                               |
|------------------------|-------------------------------|
| <code>&lt;F&gt;</code> | El tipo del primer elemento.  |
| <code>&lt;S&gt;</code> | El tipo del segundo elemento. |

Definición en la línea 16 del archivo [Pair.java](#).

### 7.4.2. Documentación de constructores y destructores

#### 7.4.2.1. Pair()

```
es.ull.esit.utils.Pair< F, S >.Pair (
    F first,
    S second) [inline]
```

Constructor para crear un par.

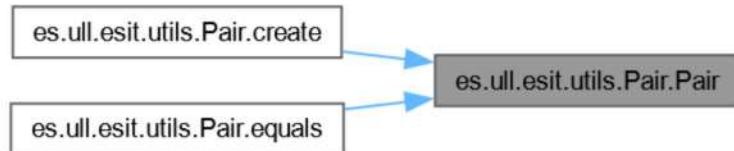
#### Parámetros

|                     |                    |
|---------------------|--------------------|
| <code>first</code>  | El primer objeto.  |
| <code>second</code> | El segundo objeto. |

Definición en la línea 31 del archivo [Pair.java](#).

```
00031           {
00032     this.first = first;
00033     this.second = second;
00034 }
```

Gráfico de llamadas a esta función:



### 7.4.3. Documentación de funciones miembro

#### 7.4.3.1. create()

```
static< A, B > Pair< A, B > es.ul.esit.utils.Pair< F, S >.create (
    A a,
    B b) [inline], [static]
```

Método de fábrica estático para crear una instancia de `Pair`.

Permite la creación de un par con inferencia de tipos.

##### Parámetros

|     |                               |
|-----|-------------------------------|
| <A> | El tipo del primer elemento.  |
| <B> | El tipo del segundo elemento. |
| a   | El primer objeto.             |
| b   | El segundo objeto.            |

##### Devuelve

Una nueva instancia de `Pair` conteniendo a y b.

Definición en la línea 68 del archivo `Pair.java`.

```
00068
00069     return new Pair<A, B>(a, b);
00070 }
```

Gráfico de llamadas de esta función:



#### 7.4.3.2. equals()

```
boolean es.ull.esit.utils.Pair< F, S >.equals ( Object o) [inline]
```

Compara este par con otro objeto para ver si son iguales.

Parámetros

|                                   |                       |
|-----------------------------------|-----------------------|
| <input type="checkbox"/> <i>o</i> | El objeto a comparar. |
|-----------------------------------|-----------------------|

Devuelve

true si el objeto es un [Pair](#) y ambos elementos son iguales, false en caso contrario.

Definición en la línea 42 del archivo [Pair.java](#).

```
00042         {  
00043             if (! (o instanceof Pair)) {  
00044                 return false;  
00045             }  
00046             Pair<?, ?> p = (Pair<?, ?>) o;  
00047             return Objects.equals(p.first, first) && Objects.equals(p.second, second);  
00048     }
```

Gráfico de llamadas de esta función:



#### 7.4.3.3. hashCode()

```
int es.ull.esit.utils.Pair< F, S >.hashCode () [inline]
```

Calcula el código hash para el par.

Devuelve

El código hash, basado en los códigos hash de los dos elementos.

Definición en la línea 55 del archivo [Pair.java](#).

```
00055         {  
00056             return (first == null ? 0 : first.hashCode()) ^ (second == null ? 0 : second.hashCode());  
00057     }
```

#### 7.4.4. Documentación de datos miembro

##### 7.4.4.1. first

```
final F es.ull.esit.utils.Pair< F, S >.first
```

El primer elemento del par.

Definición en la línea [20](#) del archivo [Pair.java](#).

##### 7.4.4.2. second

```
final S es.ull.esit.utils.Pair< F, S >.second
```

El segundo elemento del par.

Definición en la línea [24](#) del archivo [Pair.java](#).

La documentación de esta clase está generada del siguiente archivo:

- C:/Users/Kefle/Desktop/ExpositoTOP/src/es/ull/esit/utils/[Pair.java](#)

### 7.5. Referencia de la plantilla de la clase es.ull.esit.utilities.PowerSet< E >

Implementa un iterador para generar todos los subconjuntos (conjunto potencia) de un conjunto dado.

Diagrama de herencia de es.ull.esit.utilities.PowerSet< E >

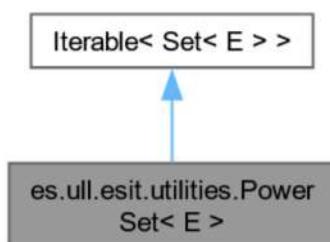
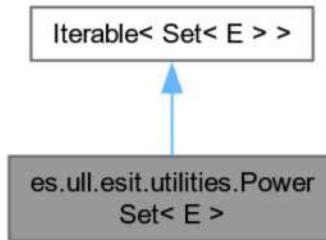


Diagrama de colaboración de es.ull.esit.utilities.PowerSet< E >:



## Clases

- class [PowerSetIterator](#)

## Métodos públicos

- [PowerSet](#) (`Set< E > set`)  
*Constructor de la clase PowerSet.*
- [Iterator< Set< E > > iterator](#) ()  
*Devuelve un iterador sobre los subconjuntos.*

## Atributos privados

- `E[] arr = null`

### 7.5.1. Descripción detallada

Implementa un iterador para generar todos los subconjuntos (conjunto potencia) de un conjunto dado.

Utiliza un `BitSet` para generar de forma eficiente todas las combinaciones de elementos del conjunto original.

#### Parámetros

<*E*> El tipo de los elementos en el conjunto.

Definición en la línea [20](#) del archivo [PowerSet.java](#).

### 7.5.2. Documentación de constructores y destructores

#### 7.5.2.1. PowerSet()

```
es.ull.esit.utilities.PowerSet< E >.PowerSet (
    Set< E > set) [inline]
```

Constructor de la clase [PowerSet](#).

### Parámetros

|                          |                  |   |
|--------------------------|------------------|---|
| <input type="checkbox"/> | <code>set</code> | El conjunto original del cual se generará el conjunto potencia. |
|--------------------------|------------------|---|

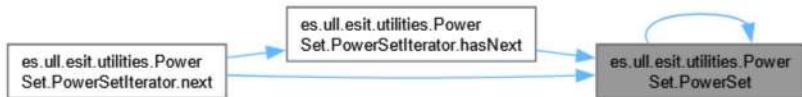
Definición en la línea 29 del archivo [PowerSet.java](#).

```
00029           {
00030     this.arr = (E[]) set.toArray();
00031 }
```

Gráfico de llamadas de esta función:



Gráfico de llamadas a esta función:



### 7.5.3. Documentación de funciones miembro

#### 7.5.3.1. iterator()

```
Iterator< Set< E > > es.ul.esit.utilities.PowerSet< E >.iterator () [inline]
```

Devuelve un iterador sobre los subconjuntos.

Devuelve

una nueva instancia del iterador del conjunto potencia.

Definición en la línea 38 del archivo [PowerSet.java](#).

```
00038   {
00039     return new PowerSetIterator();
00040 }
```

### 7.5.4. Documentación de datos miembro

#### 7.5.4.1. arr

```
E [ ] es.ull.esit.utilities.PowerSet< E >.arr = null [private]
```

Definición en la línea 22 del archivo [PowerSet.java](#).

La documentación de esta clase está generada del siguiente archivo:

- C:/Users/Kefle/Desktop/ExpositoTOP/src/es/ull/esit/utilities/[PowerSet.java](#)

## 7.6. Referencia de la clase es.ull.esit.utilities.PowerSet< E >.PowerSetIterator

Diagrama de herencia de es.ull.esit.utilities.PowerSet< E >.PowerSetIterator

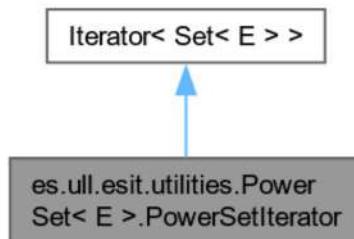
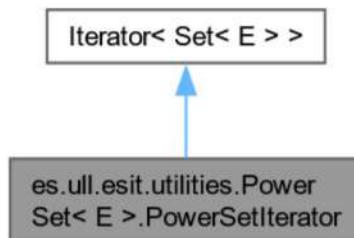


Diagrama de colaboración de es.ull.esit.utilities.PowerSet< E >.PowerSetIterator:



## Métodos públicos

- boolean `hasNext ()`  
*Comprueba si hay más subconjuntos para generar.*
- Set< E > `next ()`  
*Devuelve el siguiente subconjunto en la secuencia.*
- void `remove ()`  
*Operación no soportada.*

## Atributos privados

- BitSet `bset`

### 7.6.1. Descripción detallada

Definición en la línea 42 del archivo [PowerSet.java](#).

### 7.6.2. Documentación de funciones miembro

#### 7.6.2.1. `hasNext()`

```
boolean es.ull.esit.utilities.PowerSet< E >.PowerSetIterator.hasNext () [inline]
```

Comprueba si hay más subconjuntos para generar.

Devuelve

`true` si hay más subconjuntos, `false` en caso contrario.

Definición en la línea 54 del archivo [PowerSet.java](#).

```
00054         {
00055             return !this.bset.get(PowerSet.this.arr.length);
00056         }
```

Gráfico de llamadas de esta función:

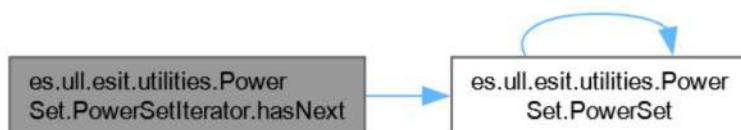
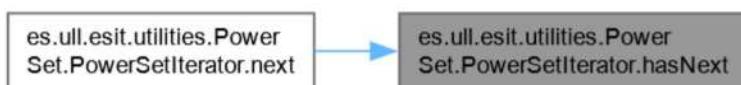


Gráfico de llamadas a esta función:



### 7.6.2.2. next()

`Set< E > es.ull.esit.utilities.PowerSet< E >.PowerSetIterator.next () [inline]`

Devuelve el siguiente subconjunto en la secuencia.

Devuelve

Un Set<E> que representa el siguiente subconjunto.

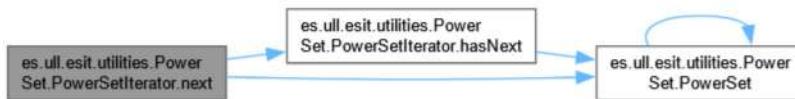
Excepciones

|                          |                               |                          |
|--------------------------|-------------------------------|--------------------------|
| <input type="checkbox"/> | <i>NoSuchElementException</i> | si no hay más elementos. |
|--------------------------|-------------------------------|--------------------------|

Definición en la línea 64 del archivo PowerSet.java.

```
00064      {
00065          if (!hasNext()) {
00066              throw new NoSuchElementException();
00067          }
00068          Set<E> returnSet = new TreeSet<E>();
00069          for (int i = 0; i < PowerSet.this.arr.length; i++) {
00070              if (this.bset.get(i)) {
00071                  returnSet.add(PowerSet.this.arr[i]);
00072              }
00073          }
00074          // Incrementa el bitset para la siguiente combinación
00075          for (int i = 0; i < this.bset.size(); i++) {
00076              if (!this.bset.get(i)) {
00077                  this.bset.set(i);
00078                  break;
00079              } else {
00080                  this.bset.clear(i);
00081              }
00082          }
00083          return returnSet;
00084      }
```

Gráfico de llamadas de esta función:



### 7.6.2.3. remove()

`void es.ull.esit.utilities.PowerSet< E >.PowerSetIterator.remove () [inline]`

Operación no soportada.

Excepciones

|                          |                                      |          |
|--------------------------|--------------------------------------|----------|
| <input type="checkbox"/> | <i>UnsupportedOperationException</i> | Siempre. |
|--------------------------|--------------------------------------|----------|

Definición en la línea 91 del archivo PowerSet.java.

```
00091      {
00092          throw new UnsupportedOperationException("Not Supported!");
00093      }
```

### 7.6.3. Documentación de datos miembro

#### 7.6.3.1. bset

```
BitSet es.ull.esit.utilities.PowerSet< E >.PowerSetIterator.bset [private]
```

Definición en la línea 43 del archivo [PowerSet.java](#).

La documentación de esta clase está generada del siguiente archivo:

- C:/Users/Kefle/Desktop/ExpositoTOP/src/es/ull/esit/utilities/[PowerSet.java](#)

## 7.7. Referencia de la clase top.TOPTW

Modela una instancia del problema [TOPTW](#) (Team Orienteering Problem with Time Windows).

### Métodos públicos

- [TOPTW](#) (int nodes, int routes)
 

*Constructor de la clase TOPTW.*
- boolean [isDepot](#) (int a)
 

*Comprueba si un índice de nodo corresponde a un depósito.*
- double [getDistance](#) (int[] route)
 

*Calcula la distancia total de una ruta representada por un array de enteros.*
- double [getDistance](#) (ArrayList< Integer > route)
 

*Calcula la distancia total de una ruta representada por un ArrayList de enteros.*
- double [getDistance](#) (ArrayList< Integer >[] routes)
 

*Calcula la distancia total sumando las distancias de múltiples rutas.*
- void [calculateDistanceMatrix](#) ()
 

*Calcula y almacena la matriz de distancias euclidianas entre todos los nodos.*
- double [getMaxTimePerRoute](#) ()
 

*Obtiene el tiempo máximo permitido por ruta.*
- void [setMaxTimePerRoute](#) (double maxTimePerRoute)
 

*Establece el tiempo máximo permitido por ruta.*
- double [getMaxRoutes](#) ()
 

*Obtiene el número máximo de rutas (vehículos) permitidas.*
- void [setMaxRoutes](#) (double maxRoutes)
 

*Establece el número máximo de rutas.*
- int [getPOIs](#) ()
 

*Obtiene el número de Puntos de Interés (POIs), sin contar el depósito.*
- double [getDistance](#) (int i, int j)
 

*Obtiene la distancia entre dos nodos.*
- double [getTime](#) (int i, int j)
 

*Obtiene el tiempo de viaje entre dos nodos, que es igual a la distancia.*
- int [getNodes](#) ()
 

*Obtiene el número total de nodos (clientes).*
- void [setNodes](#) (int nodes)
 

*Establece el número de nodos.*

- **double getX (int index)**  
*Obtiene la coordenada X de un nodo.*
- **void setX (int index, double x)**  
*Establece la coordenada X de un nodo.*
- **double getY (int index)**  
*Obtiene la coordenada Y de un nodo.*
- **void setY (int index, double y)**  
*Establece la coordenada Y de un nodo.*
- **double getScore (int index)**  
*Obtiene la puntuación (score) de un nodo.*
- **double[] getScore ()**  
*Obtiene el array completo de puntuaciones de todos los nodos.*
- **void setScore (int index, double score)**  
*Establece la puntuación de un nodo.*
- **double getReadyTime (int index)**  
*Obtiene el tiempo de inicio de la ventana de tiempo (ready time) de un nodo.*
- **void setReadyTime (int index, double readyTime)**  
*Establece el tiempo de inicio de la ventana de tiempo (ready time) de un nodo.*
- **double getDueTime (int index)**  
*Obtiene el tiempo de fin de la ventana de tiempo (due time) de un nodo.*
- **void setDueTime (int index, double dueTime)**  
*Establece el tiempo de fin de la ventana de tiempo (due time) de un nodo.*
- **double getServiceTime (int index)**  
*Obtiene el tiempo de servicio requerido en un nodo.*
- **void setServiceTime (int index, double serviceTime)**  
*Establece el tiempo de servicio de un nodo.*
- **int getVehicles ()**  
*Obtiene el número de vehículos disponibles.*
- **String toString ()**  
*Genera una representación en formato de cadena de la instancia del problema.*
- **int addNode ()**  
*Incrementa el contador de nodos.*
- **int addNodeDepot ()**  
*Incrementa el contador de depósitos.*

### Atributos privados

- **int nodes**
- **double[] x**
- **double[] y**
- **double[] score**
- **double[] readyTime**
- **double[] dueTime**
- **double[] serviceTime**
- **int vehicles**
- **int depots**
- **double maxTimePerRoute**
- **double maxRoutes**
- **double[][] distanceMatrix**

## 7.7.1. Descripción detallada

Modela una instancia del problema [TOPTW](#) (Team Orienteering Problem with Time Windows).

Esta clase almacena todos los datos de una instancia del problema, incluyendo la información de los nodos (coordenadas, puntuaciones, ventanas de tiempo), el número de vehículos, y las restricciones generales como el tiempo máximo por ruta. También proporciona métodos para calcular distancias y acceder a los datos.

Definición en la línea 19 del archivo [TOPTW.java](#).

## 7.7.2. Documentación de constructores y destructores

### 7.7.2.1. TOPTW()

```
top.TOPTW.TOPTW (
    int nodes,
    int routes) [inline]
```

Constructor de la clase [TOPTW](#).

Parámetros

|                     |  |
|---------------------|--|
| <code>nodes</code>  | El número de nodos (clientes/POIs) en el problema, sin contar el depósito. |
| <code>routes</code> | El número de rutas (vehículos) disponibles.                                |

Definición en la línea 38 del archivo [TOPTW.java](#).

```
00038     this.nodes = nodes;
00039     this.depots = 0;
00040     this.x = new double[this.nodes + 1];
00041     this.y = new double[this.nodes + 1];
00042     this.score = new double[this.nodes + 1];
00043     this.readyTime = new double[this.nodes + 1];
00044     this.dueTime = new double[this.nodes + 1];
00045     this.serviceTime = new double[this.nodes + 1];
00046     this.distanceMatrix = new double[this.nodes + 1][this.nodes + 1];
00047     for (int i = 0; i < this.nodes + 1; i++) {
00048         for (int j = 0; j < this.nodes + 1; j++) {
00049             this.distanceMatrix[i][j] = 0.0;
00050         }
00051     }
00052     this.maxRoutes = routes;
00053     this.vehicles = routes;
00054 }
```

## 7.7.3. Documentación de funciones miembro

### 7.7.3.1. addNode()

```
int top.TOPTW.addNode () [inline]
```

Incrementa el contador de nodos.

**Obsoleto** Este método parece no ser utilizado de forma segura, ya que no redimensiona los arrays.

Devuelve

El nuevo número de nodos.

Definición en la línea 382 del archivo [TOPTW.java](#).

```
00382     this.nodes++;
00383     return this.nodes;
00384 }
```

### 7.7.3.2. addNodeDepot()

```
int top.TOPTW.addNodeDepot () [inline]
```

Incrementa el contador de depósitos.

**Obsoleto** Este método parece no ser utilizado de forma segura.

Devuelve

El nuevo número de depósitos.

Definición en la línea 392 del archivo [TOPTW.java](#).

```
00392     this.depots++;
00393     return this.depots;
00394 }
00395 }
```

Gráfico de llamadas a esta función:



### 7.7.3.3. calculateDistanceMatrix()

```
void top.TOPTW.calculateDistanceMatrix () [inline]
```

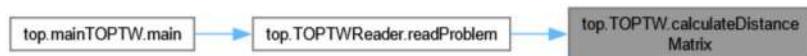
Calcula y almacena la matriz de distancias euclidianas entre todos los nodos.

Utiliza las coordenadas X e Y de los nodos para calcular la distancia. La matriz es simétrica.

Definición en la línea 117 del archivo [TOPTW.java](#).

```
00117     for (int i = 0; i < this.nodes + 1; i++) {
00118         for (int j = 0; j < this.nodes + 1; j++) {
00119             if (i != j) {
00120                 double diffXs = this.x[i] - this.x[j];
00121                 double diffYs = this.y[i] - this.y[j];
00122                 this.distanceMatrix[i][j] = Math.sqrt(diffXs * diffXs + diffYs * diffYs);
00123                 this.distanceMatrix[j][i] = this.distanceMatrix[i][j];
00124             } else {
00125                 this.distanceMatrix[i][j] = 0.0;
00126             }
00127         }
00128     }
00129 }
00130 }
```

Gráfico de llamadas a esta función:



### 7.7.3.4. getDistance() [1/4]

```
double top.TOPTW.getDistance (
    ArrayList< Integer > route) [inline]
```

Calcula la distancia total de una ruta representada por un ArrayList de enteros.

## Parámetros

|                          |       |  |
|--------------------------|-------|--|
| <input type="checkbox"/> | route | La ruta como un ArrayList de índices de nodos. |
|--------------------------|-------|--|

## Devuelve

La distancia total de la ruta.

Definición en la línea 89 del archivo [TOPTW.java](#).

```
00089         double distance = 0.0;
00090         for (int i = 0; i < route.size() - 1; i++) {
00091             int node1 = route.get(i);
00092             int node2 = route.get(i + 1);
00093             distance += this.getDistance(node1, node2);
00094         }
00095     }
00096     return distance;
00097 }
```

Gráfico de llamadas de esta función:



### 7.7.3.5. `getDistance()` [2/4]

```
double top.TOPTW.getDistance (
    ArrayList< Integer >[] routes) [inline]
```

Calcula la distancia total sumando las distancias de múltiples rutas.

## Parámetros

|                          |        |   |
|--------------------------|--------|---|
| <input type="checkbox"/> | routes | Un array de ArrayLists, donde cada ArrayList es una ruta. |
|--------------------------|--------|---|

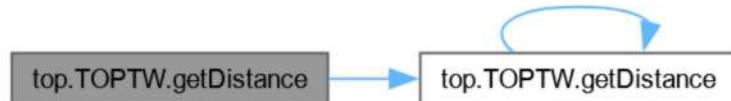
## Devuelve

La distancia total acumulada de todas las rutas.

Definición en la línea 104 del archivo [TOPTW.java](#).

```
00104         double distance = 0.0;
00105         for (ArrayList<Integer> route : routes) {
00106             distance += this.getDistance(route);
00107         }
00108     }
00109     return distance;
00110 }
```

Gráfico de llamadas de esta función:



#### 7.7.3.6. `getDistance()` [3/4]

```
double top.TOPTW.getDistance (
    int i,
    int j) [inline]
```

Obtiene la distancia entre dos nodos.

Si alguno de los nodos es un depósito, se mapea al índice 0.

##### Parámetros

|                |                          |
|----------------|--------------------------|
| <code>i</code> | Índice del primer nodo.  |
| <code>j</code> | Índice del segundo nodo. |

##### Devuelve

La distancia entre los nodos `i` y `j`.

Definición en la línea 179 del archivo [TOPTW.java](#).

```
00179
00180      if(this.isDepot(i)) { i=0; }
00181      if(this.isDepot(j)) { j=0; }
00182      return this.distanceMatrix[i][j];
00183 }
```

Gráfico de llamadas de esta función:



#### 7.7.3.7. `getDistance()` [4/4]

```
double top.TOPTW.getDistance (
    int[] route) [inline]
```

Calcula la distancia total de una ruta representada por un array de enteros.

**Parámetros**

|                                       |  |
|---------------------------------------|--|
| <input type="checkbox"/> <i>route</i> | La ruta como un array de índices de nodos. |
|---------------------------------------|--|

**Devuelve**

La distancia total de la ruta.

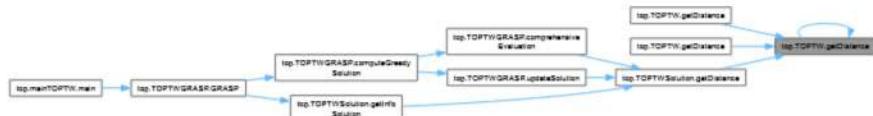
Definición en la línea 74 del archivo [TOPTW.java](#).

```
00074         {
00075             double distance = 0.0;
00076             for (int i = 0; i < route.length - 1; i++) {
00077                 int node1 = route[i];
00078                 int node2 = route[i + 1];
00079                 distance += this.getDistance(node1, node2);
00080             }
00081             return distance;
00082         }
```

Gráfico de llamadas de esta función:



Gráfico de llamadas a esta función:

**7.7.3.8. `getDueTime()`**

```
double top.TOPTW.getDueTime (
        int index) [inline]
```

Obtiene el tiempo de fin de la ventana de tiempo (due time) de un nodo.

**Parámetros**

|                                       |                  |
|---------------------------------------|------------------|
| <input type="checkbox"/> <i>index</i> | Índice del nodo. |
|---------------------------------------|------------------|

Devuelve

El due time del nodo.

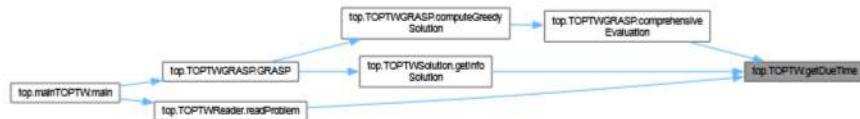
Definición en la línea 302 del archivo [TOPTW.java](#).

```
00302         {
00303     if(this.isDepot(index)) { index=0; }
00304     return this.dueTime[index];
00305 }
```

Gráfico de llamadas de esta función:



Gráfico de llamadas a esta función:



### 7.7.3.9. getMaxRoutes()

```
double top.TOPTW.getMaxRoutes () [inline]
```

Obtiene el número máximo de rutas (vehículos) permitidas.

Devuelve

El número máximo de rutas.

Definición en la línea 152 del archivo [TOPTW.java](#).

```
00152         {
00153     return maxRoutes;
00154 }
```

Gráfico de llamadas a esta función:



### 7.7.3.10. getMaxTimePerRoute()

```
double top.TOPTW.getMaxTimePerRoute () [inline]
```

Obtiene el tiempo máximo permitido por ruta.

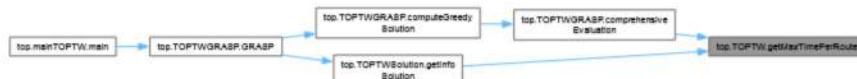
Devuelve

El tiempo máximo por ruta.

Definición en la línea 136 del archivo [TOPTW.java](#).

```
00136
00137     return maxTimePerRoute;
00138 }
```

Gráfico de llamadas a esta función:



### 7.7.3.11. getNodes()

```
int top.TOPTW.getNodes () [inline]
```

Obtiene el número total de nodos (clientes).

Devuelve

El número de nodos.

Definición en la línea 201 del archivo [TOPTW.java](#).

```
00201
00202     return this.nodes;
00203 }
```

### 7.7.3.12. getPOIs()

```
int top.TOPTW.getPOIs () [inline]
```

Obtiene el número de Puntos de Interés (POIs), sin contar el depósito.

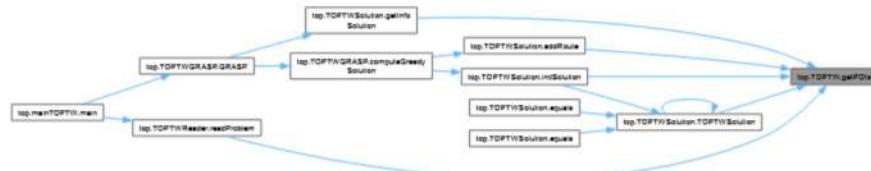
Devuelve

El número de nodos cliente.

Definición en la línea 168 del archivo [TOPTW.java](#).

```
00168
00169     return this.nodes;
00170 }
```

Gráfico de llamadas a esta función:



### 7.7.3.13. `getReadyTime()`

```
double top.TOPTW.getReadyTime (
    int index) [inline]
```

Obtiene el tiempo de inicio de la ventana de tiempo (ready time) de un nodo.

#### Parámetros

|                          |                    |                  |
|--------------------------|--------------------|------------------|
| <input type="checkbox"/> | <code>index</code> | Índice del nodo. |
|--------------------------|--------------------|------------------|

#### Devuelve

El ready time del nodo.

Definición en la línea 283 del archivo [TOPTW.java](#).

```
00283         if(this.isDepot(index)) { index=0; }
00284         return this.readyTime[index];
00285     }
```

Gráfico de llamadas de esta función:



Gráfico de llamadas a esta función:



### 7.7.3.14. `getScore() [1/2]`

```
double[] top.TOPTW.getScore () [inline]
```

Obtiene el array completo de puntuaciones de todos los nodos.

#### Devuelve

Un array de doubles con las puntuaciones.

Definición en la línea 265 del archivo [TOPTW.java](#).

```
00265         {
00266             return this.score;
00267         }
```

### 7.7.3.15. `getScore()` [2/2]

```
double top.TOPTW.getScore (
    int index) [inline]
```

Obtiene la puntuación (score) de un nodo.

Parámetros

|                    |                  |
|--------------------|------------------|
| <code>index</code> | Índice del nodo. |
|--------------------|------------------|

Devuelve

La puntuación del nodo.

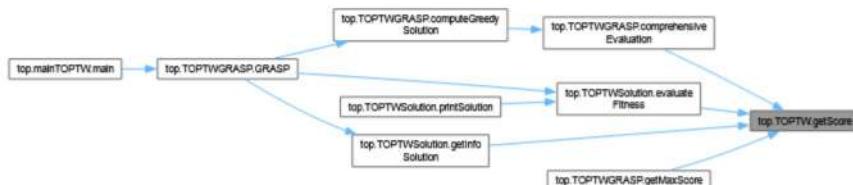
Definición en la línea 256 del archivo [TOPTW.java](#).

```
00256      {
00257          if(this.isDepot(index)) { index=0; }
00258          return this.score[index];
00259      }
```

Gráfico de llamadas de esta función:



Gráfico de llamadas a esta función:



### 7.7.3.16. `getServiceTime()`

```
double top.TOPTW.getServiceTime (
    int index) [inline]
```

Obtiene el tiempo de servicio requerido en un nodo.

## Parámetros

|                                       |                  |
|---------------------------------------|------------------|
| <input type="checkbox"/> <i>index</i> | Índice del nodo. |
|---------------------------------------|------------------|

## Devuelve

El tiempo de servicio.

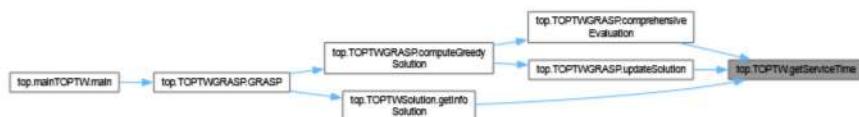
Definición en la línea 321 del archivo [TOPTW.java](#).

```
00321         if(this.isDepot(index)) { index=0; }
00322         return this.serviceTime[index];
00323     }
```

Gráfico de llamadas de esta función:



Gráfico de llamadas a esta función:



### 7.7.3.17. `getTime()`

```
double top.TOPTW.getTime (
    int i,
    int j) [inline]
```

Obtiene el tiempo de viaje entre dos nodos, que es igual a la distancia.

## Parámetros

|                                   |                          |
|-----------------------------------|--------------------------|
| <input type="checkbox"/> <i>i</i> | Índice del primer nodo.  |
| <input type="checkbox"/> <i>j</i> | Índice del segundo nodo. |

Devuelve

El tiempo de viaje entre i y j.

Definición en la línea 191 del archivo [TOPTW.java](#).

```
00191         if(this.isDepot(i)) { i=0; }
00192         if(this.isDepot(j)) { j=0; }
00193         return this.distanceMatrix[i][j];
00194     }
```

Gráfico de llamadas de esta función:



### 7.7.3.18. `getVehicles()`

```
int top.TOPTW.getVehicles () [inline]
```

Obtiene el número de vehículos disponibles.

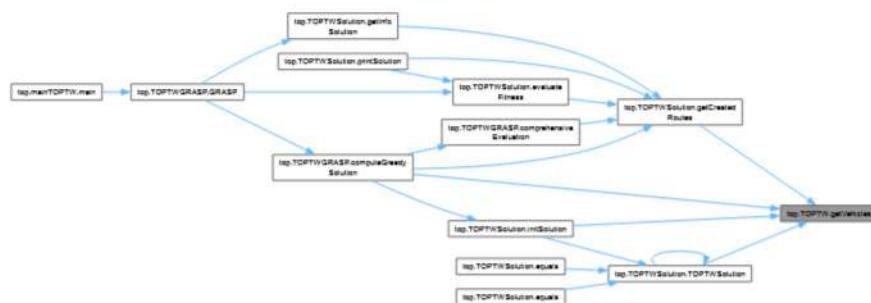
Devuelve

El número de vehículos.

Definición en la línea 339 del archivo [TOPTW.java](#).

```
00339         {
00340             return this.vehicles;
00341         }
```

Gráfico de llamadas a esta función:



### 7.7.3.19. `getX()`

```
double top.TOPTW.getX (
    int index) [inline]
```

Obtiene la coordenada X de un nodo.

## Parámetros

|                          |                    |                  |
|--------------------------|--------------------|------------------|
| <input type="checkbox"/> | <code>index</code> | Índice del nodo. |
|--------------------------|--------------------|------------------|

## Devuelve

La coordenada X.

Definición en la línea 218 del archivo [TOPTW.java](#).

```
00218     {
00219         if(this.isDepot(index)) { index=0; }
00220         return this.x[index];
00221     }
```

Gráfico de llamadas de esta función:



Gráfico de llamadas a esta función:



### 7.7.3.20. `getY()`

```
double top.TOPTW.getY (
    int index) [inline]
```

Obtiene la coordenada Y de un nodo.

## Parámetros

|                          |                    |                  |
|--------------------------|--------------------|------------------|
| <input type="checkbox"/> | <code>index</code> | Índice del nodo. |
|--------------------------|--------------------|------------------|

Devuelve

La coordenada Y.

Definición en la línea 237 del archivo [TOPTW.java](#).

```
00237     {
00238         if(this.isDepot(index)) { index=0; }
00239         return this.y[index];
00240     }
```

Gráfico de llamadas de esta función:



Gráfico de llamadas a esta función:



### 7.7.3.21. `isDepot()`

```
boolean top.TOPTW.isDepot (
    int a) [inline]
```

Comprueba si un índice de nodo corresponde a un depósito.

Parámetros

|                |  |
|----------------|--|
| <code>a</code> | El índice del nodo a comprobar. Los depósitos tienen índices mayores que el número de nodos. |
|----------------|--|

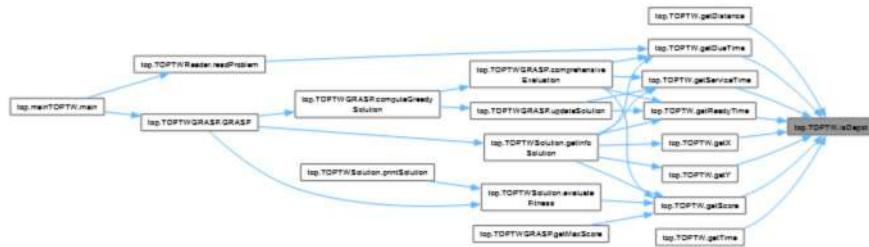
Devuelve

`true` si el nodo es un depósito, `false` en caso contrario.

Definición en la línea 62 del archivo [TOPTW.java](#).

```
00062
00063     if(a > this.nodes) {
00064         return true;
00065     }
00066     return false;
00067 }
```

Gráfico de llamadas a esta función:



### 7.7.3.22. setDueTime()

```
void top.TOPTW.setDueTime (
    int index,
    double dueTime) [inline]
```

Establece el tiempo de fin de la ventana de tiempo (due time) de un nodo.

Parámetros

|                                  |                        |
|----------------------------------|------------------------|
| <input type="checkbox"/> index   | Índice del nodo.       |
| <input type="checkbox"/> dueTime | El valor del due time. |

Definición en la línea 312 del archivo [TOPTW.java](#).

```
00312
00313     this.dueTime[index] = dueTime;
00314 }
```

Gráfico de llamadas a esta función:



### 7.7.3.23. setMaxRoutes()

```
void top.TOPTW.setMaxRoutes (
    double maxRoutes) [inline]
```

Establece el número máximo de rutas.

Parámetros

|                                    |                                  |
|------------------------------------|----------------------------------|
| <input type="checkbox"/> maxRoutes | El nuevo número máximo de rutas. |
|------------------------------------|----------------------------------|

Definición en la línea 160 del archivo [TOPTW.java](#).

```
00160
00161     this.maxRoutes = maxRoutes;
00162 }
```

#### 7.7.3.24. `setMaxTimePerRoute()`

```
void top.TOPTW.setMaxTimePerRoute (
    double maxTimePerRoute) [inline]
```

Establece el tiempo máximo permitido por ruta.

##### Parámetros

|   |                                  |
|---|----------------------------------|
| <input type="checkbox"/> <code>maxTimePerRoute</code> | El nuevo tiempo máximo por ruta. |
|---|----------------------------------|

Definición en la línea 144 del archivo [TOPTW.java](#).

```
00144
00145     this.maxTimePerRoute = maxTimePerRoute;
00146 }
```

Gráfico de llamadas a esta función:



#### 7.7.3.25. `setNodes()`

```
void top.TOPTW.setNodes (
    int nodes) [inline]
```

Establece el número de nodos.

##### Parámetros

|   |                           |
|---|---------------------------|
| <input type="checkbox"/> <code>nodes</code> | El nuevo número de nodos. |
|---|---------------------------|

Definición en la línea 209 del archivo [TOPTW.java](#).

```
00209
00210     this.nodes = nodes;
00211 }
```

#### 7.7.3.26. `setReadyTime()`

```
void top.TOPTW.setReadyTime (
    int index,
    double readyTime) [inline]
```

Establece el tiempo de inicio de la ventana de tiempo (ready time) de un nodo.

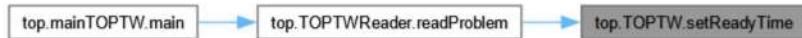
##### Parámetros

|   |                          |
|---|--------------------------|
| <input type="checkbox"/> <code>index</code>     | Índice del nodo.         |
| <input type="checkbox"/> <code>readyTime</code> | El valor del ready time. |

Definición en la línea 293 del archivo [TOPTW.java](#).

```
00293     this.readyTime[index] = readyTime;
00294 }
00295 }
```

Gráfico de llamadas a esta función:



### 7.7.3.27. setScore()

```
void top.TOPTW.setScore (
    int index,
    double score) [inline]
```

Establece la puntuación de un nodo.

Parámetros

|              |                            |
|--------------|----------------------------|
| <i>index</i> | Índice del nodo.           |
| <i>score</i> | El valor de la puntuación. |

Definición en la línea 274 del archivo [TOPTW.java](#).

```
00274     this.score[index] = score;
00275 }
00276 }
```

Gráfico de llamadas a esta función:



### 7.7.3.28. setServiceTime()

```
void top.TOPTW.setServiceTime (
    int index,
    double serviceTime) [inline]
```

Establece el tiempo de servicio de un nodo.

### Parámetros

|                    |                                  |
|--------------------|----------------------------------|
| <i>index</i>       | Índice del nodo.                 |
| <i>serviceTime</i> | El valor del tiempo de servicio. |

Definición en la línea 331 del archivo [TOPTW.java](#).

```
00331
00332     this.serviceTime[index] = serviceTime;
00333 }
```

Gráfico de llamadas a esta función:



### 7.7.3.29. setX()

```
void top.TOPTW.setX (
    int index,
    double x) [inline]
```

Establece la coordenada X de un nodo.

### Parámetros

|              |                              |
|--------------|------------------------------|
| <i>index</i> | Índice del nodo.             |
| <i>x</i>     | El valor de la coordenada X. |

Definición en la línea 228 del archivo [TOPTW.java](#).

```
00228
00229     this.x[index] = x;
00230 }
```

Gráfico de llamadas a esta función:



### 7.7.3.30. setY()

```
void top.TOPTW.setY (
    int index,
    double y) [inline]
```

Establece la coordenada Y de un nodo.

## Parámetros

|                    |                              |
|--------------------|------------------------------|
| <code>index</code> | Índice del nodo.             |
| <code>y</code>     | El valor de la coordenada Y. |

Definición en la línea 247 del archivo [TOPTW.java](#).

```
00247         this.y[index] = y;
00248     }
00249 }
```

Gráfico de llamadas a esta función:



### 7.7.3.31. `toString()`

```
String top.TOPTW.toString () [inline]
```

Genera una representación en formato de cadena de la instancia del problema.

## Devuelve

Una cadena formateada con los detalles de los nodos y vehículos.

Definición en la línea 348 del archivo [TOPTW.java](#).

```
00348     {
00349         final int COLUMN_WIDTH = 15;
00350         String text = "Nodes: " + this.nodes + "\n";
00351         String[] strings = new String[]{"CUST NO.", "XCOORD.", "YCOORD.", "SCORE", "READY TIME", "DUE
DATE", "SERVICE TIME"};
00352         int[] width = new int[strings.length];
00353         Arrays.fill(width, COLUMN_WIDTH);
00354         text += ExpositoUtilities.getFormat(strings, width) + "\n";
00355         for (int i = 0; i < this.nodes; i++) {
00356             strings = new String[strings.length];
00357             int index = 0;
00358             //strings[index++] = Integer.toString("" + i);
00359             strings[index++] = Integer.toString(i);
00360             strings[index++] = "" + this.x[i];
00361             strings[index++] = "" + this.y[i];
00362             strings[index++] = "" + this.score[i];
00363             strings[index++] = "" + this.readyTime[i];
00364             strings[index++] = "" + this.dueTime[i];
00365             strings[index++] = "" + this.serviceTime[i];
00366             text += ExpositoUtilities.getFormat(strings, width);
00367             text += "\n";
00368         }
00369         text += "Vehicles: " + this.vehicles + "\n";
00370         strings = new String[]{"VEHICLE", "CAPACITY"};
00371         width = new int[strings.length];
00372         Arrays.fill(width, COLUMN_WIDTH);
00373         text += ExpositoUtilities.getFormat(strings, width) + "\n";
00374         return text;
00375     }
```

Gráfico de llamadas de esta función:



## 7.7.4. Documentación de datos miembro

### 7.7.4.1. depots

```
int top.TOPTW.depots [private]
```

Definición en la línea 28 del archivo [TOPTW.java](#).

### 7.7.4.2. distanceMatrix

```
double [][] top.TOPTW.distanceMatrix [private]
```

Definición en la línea 31 del archivo [TOPTW.java](#).

### 7.7.4.3. dueTime

```
double [] top.TOPTW.dueTime [private]
```

Definición en la línea 25 del archivo [TOPTW.java](#).

### 7.7.4.4. maxRoutes

```
double top.TOPTW.maxRoutes [private]
```

Definición en la línea 30 del archivo [TOPTW.java](#).

### 7.7.4.5. maxTimePerRoute

```
double top.TOPTW.maxTimePerRoute [private]
```

Definición en la línea 29 del archivo [TOPTW.java](#).

#### 7.7.4.6. nodes

```
int top.TOPTW.nodes [private]
```

Definición en la línea 20 del archivo [TOPTW.java](#).

#### 7.7.4.7. readyTime

```
double [] top.TOPTW.readyTime [private]
```

Definición en la línea 24 del archivo [TOPTW.java](#).

#### 7.7.4.8. score

```
double [] top.TOPTW.score [private]
```

Definición en la línea 23 del archivo [TOPTW.java](#).

#### 7.7.4.9. serviceTime

```
double [] top.TOPTW.serviceTime [private]
```

Definición en la línea 26 del archivo [TOPTW.java](#).

#### 7.7.4.10. vehicles

```
int top.TOPTW.vehicles [private]
```

Definición en la línea 27 del archivo [TOPTW.java](#).

#### 7.7.4.11. x

```
double [] top.TOPTW.x [private]
```

Definición en la línea 21 del archivo [TOPTW.java](#).

#### 7.7.4.12. y

```
double [] top.TOPTW.y [private]
```

Definición en la línea 22 del archivo [TOPTW.java](#).

La documentación de esta clase está generada del siguiente archivo:

- C:/Users/Kefle/Desktop/ExpositoTOP/src/top/[TOPTW.java](#)

## 7.8. Referencia de la clase top.TOPTWEvaluator

Evalúa la calidad de una solución para el problema [TOPTW](#).

### Métodos públicos

- void [evaluate](#) (TOPTWSolution solution)

*Evalúa una solución [TOPTW](#).*

### Atributos públicos estáticos

- static double [NO\\_EVALUATED](#) = -1.0

*Constante para indicar que una solución no ha sido evaluada.*

### 7.8.1. Descripción detallada

Evalúa la calidad de una solución para el problema [TOPTW](#).

Esta clase se encarga de calcular el valor de la función objetivo para una solución dada. Actualmente, el método de evaluación está comentado y no tiene una implementación activa.

Definición en la línea 14 del archivo [TOPTWEvaluator.java](#).

### 7.8.2. Documentación de funciones miembro

#### 7.8.2.1. evaluate()

```
void top.TOPTWEvaluator.evaluate (
    TOPTWSolution solution) [inline]
```

Evalúa una solución [TOPTW](#).

Este método debería calcular la puntuación total de las rutas de la solución, pero su implementación actual está comentada.

#### Parámetros

|   |                        |
|---|------------------------|
| <input type="text"/> <a href="#">solution</a> | La solución a evaluar. |
|---|------------------------|

Definición en la línea 26 del archivo [TOPTWEvaluator.java](#).

```
00026
00027     /*CumulativeCVRP problem = solution.getProblem();
00028     double objectiveFunctionValue = 0.0;
00029     for (int i = 0; i < solution.getIndexDepot().size(); i++) {
00030         double cumulative = 0;
00031         int depot = solution.getAnIndexDepot(i);
00032         int actual = depot;
00033         actual = solution.getSuccessor(actual);
00034         cumulative += problem.getDistanceMatrix(0, actual);
00035         objectiveFunctionValue += problem.getDistanceMatrix(0, actual);
00036         System.out.println("Desde " + 0 + " a " + actual + " = " + cumulative);
00037         while (actual != depot) {
00038             int ant = actual;
00039             actual = solution.getSuccessor(actual);
00040             if (actual != depot) {
00041                 cumulative += problem.getDistanceMatrix(ant, actual);
00042                 objectiveFunctionValue += cumulative;
00043                 System.out.println("Desde " + ant + " a " + actual + " = " + cumulative);
00044             } else {
00045                 cumulative += problem.getDistanceMatrix(ant, 0);
00046                 objectiveFunctionValue += cumulative;
00047                 System.out.println("Desde " + ant + " a " + 0 + " = " + cumulative);
00048             }
00049         }
00050     }
00051     solution.setObjectiveFunctionValue(objectiveFunctionValue);*/
00053 }
```

### 7.8.3. Documentación de datos miembro

#### 7.8.3.1. NO\_EVALUATED

```
double top.TOPTWEvaluator.NO_EVALUATED = -1.0 [static]
```

Constante para indicar que una solución no ha sido evaluada.

Definición en la línea 18 del archivo [TOPTWEvaluator.java](#).

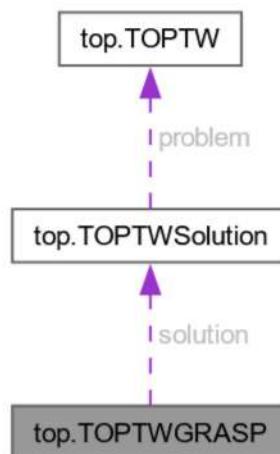
La documentación de esta clase está generada del siguiente archivo:

- [C:/Users/Kefle/Desktop/ExpositoTOP/src/top/TOPTWEvaluator.java](#)

## 7.9. Referencia de la clase top.TOPTWGRASP

Implementa la metaheurística GRASP (Greedy Randomized Adaptive Search Procedure) para resolver el [TOPTW](#).

Diagrama de colaboración de top.TOPTWGRASP:



### Métodos públicos

- [TOPTWGRASP \(TOPTWSolution sol\)](#)  
*Constructor de la clase TOPTWGRASP.*
- [void GRASP \(int maxIterations, int maxSizeRCL\)](#)  
*Ejecuta el algoritmo GRASP durante un número determinado de iteraciones.*
- [int aleatorySelectionRCL \(int maxTRCL\)](#)  
*Selecciona un elemento aleatorio de la Lista Restringida de Candidatos (RCL).*
- [int fuzzySelectionBestFDRCL \(ArrayList< double\[\] > rcl\)](#)

- **Selecciona un candidato de la RCL utilizando una función de pertenencia difusa.**
- int **fuzzySelectionAlphaCutRCL** (ArrayList< double[] > rcl, double alpha)
  - Selecciona un candidato de la RCL utilizando un corte alfa sobre la función de pertenencia difusa.*
- void **computeGreedySolution** (int maxSizeRCL)
  - Construye una solución greedy aleatorizada.*
- void **updateSolution** (double[] candidateSelected, ArrayList< ArrayList< Double > > departureTimes)
  - Actualiza la solución insertando un candidato seleccionado.*
- ArrayList< double[] > **comprehensiveEvaluation** (ArrayList< Integer > customers, ArrayList< ArrayList< Double > > departureTimes)
  - Evaluá de forma exhaustiva todos los posibles movimientos de inserción para los clientes disponibles.*
- **TOPTWSolution getSolution ()**
  - Obtiene la solución actual.*
- void **setSolution (TOPTWSolution solution)**
  - Establece la solución a utilizar.*
- int **getSolutionTime ()**
  - Obtiene el tiempo de ejecución de la solución (actualmente no implementado).*
- void **setSolutionTime (int solutionTime)**
  - Establece el tiempo de ejecución de la solución.*
- double **getMaxScore ()**
  - Obtiene la máxima puntuación posible entre todos los nodos del problema.*

### Atributos públicos estáticos

- static double **NO\_EVALUATED** = -1.0
  - Constante para indicar que una solución no ha sido evaluada.*

### Atributos privados

- **TOPTWSolution solution**
- int **solutionTime**

### Atributos estáticos privados

- static final Random **RANDOM** = new SecureRandom()

## 7.9.1. Descripción detallada

Implementa la metaheurística GRASP (Greedy Randomized Adaptive Search Procedure) para resolver el **TOPTW**.

Esta clase contiene los métodos para ejecutar el algoritmo GRASP, incluyendo la construcción de soluciones aleatorizadas y la gestión de la Lista Restringida de Candidatos (RCL).

Definición en la línea 19 del archivo **TOPTWGRASP.java**.

## 7.9.2. Documentación de constructores y destructores

### 7.9.2.1. TOPTWGRASP()

```
top.TOPTWGRASP.TOPTWGRASP (
    TOPTWSolution sol) [inline]
```

Constructor de la clase **TOPTWGRASP**.

## Parámetros

|                          |            |  |
|--------------------------|------------|--|
| <input type="checkbox"/> | <i>sol</i> | La solución inicial sobre la que operará el algoritmo. |
|--------------------------|------------|--|

Definición en la línea 33 del archivo [TOPTWGRASP.java](#).

```
00033      {
00034          this.solution = sol;
00035          this.solutionTime = 0;
00036      }
```

### 7.9.3. Documentación de funciones miembro

#### 7.9.3.1. aleatorySelectionRCL()

```
int top.TOPTWGRASP.aleatorySelectionRCL (
    int maxTRCL) [inline]
```

Selecciona un elemento aleatorio de la Lista Restringida de Candidatos (RCL).

## Parámetros

|                          |                |  |
|--------------------------|----------------|--|
| <input type="checkbox"/> | <i>maxTRCL</i> | El tamaño de la RCL sobre la que se hará la selección. |
|--------------------------|----------------|--|

## Devuelve

La posición del elemento seleccionado en la RCL.

Definición en la línea 101 del archivo [TOPTWGRASP.java](#).

```
00101      {
00102          return RANDOM.nextInt(maxTRCL);
00103      }
```

Gráfico de llamadas a esta función:



#### 7.9.3.2. comprehensiveEvaluation()

```
ArrayList< double[] > top.TOPTWGRASP.comprehensiveEvaluation (
    ArrayList< Integer > customers,
    ArrayList< ArrayList< Double > > departureTimes) [inline]
```

Evalúa de forma exhaustiva todos los posibles movimientos de inserción para los clientes disponibles.

Para cada cliente no asignado, prueba a insertarlo en cada posición posible de cada ruta existente, verificando la factibilidad de la inserción (ventanas de tiempo).

## Parámetros

|                       |   |
|-----------------------|---|
| <i>customers</i>      | La lista de clientes aún no asignados a ninguna ruta.         |
| <i>departureTimes</i> | Los tiempos de salida actuales para cada nodo en la solución. |

## Devuelve

Una lista de candidatos factibles, donde cada candidato es un array con la información [cliente, ruta, predecesor, coste, score].

Definición en la línea 280 del archivo [TOPTWGRASP.java](#).

```

00280
00281     {
00282         ArrayList< double[] > candidatesList = new ArrayList< double[] >();
00283         double[] infoCandidate = new double[5];
00284         boolean validFinalInsertion = true;
00285         infoCandidate[0] = -1;
00286         infoCandidate[1] = -1;
00287         infoCandidate[2] = -1;
00288         infoCandidate[3] = Double.MAX_VALUE;
00289         infoCandidate[4] = -1;
00290
00291         for(int c = 0; c < customers.size(); c++) { // clientes disponibles
00292             for(int k = 0; k < this.solution.getCreatedRoutes(); k++) { // rutas creadas
00293                 validFinalInsertion = true;
00294                 int depot = this.solution.getIndexRoute(k);
00295                 int pre=-1, suc=-1;
00296                 double costInsertion = 0;
00297                 pre = depot;
00298                 int candidate = customers.get(c);
00299                 do {                                // recorremos la ruta
00300                     validFinalInsertion = true;
00301                     suc = this.solution.getSuccessor(pre);
00302                     double timesUntilPre = departureTimes.get(k).get(pre) +
00303                         this.solution.getDistance(pre, candidate);
00304                     if(timesUntilPre < (this.solution.getProblem().getDueTime(candidate))) {
00305                         double costCand = 0;
00306                         if(timesUntilPre < this.solution.getProblem().getReadyTime(candidate)) {
00307                             costCand = this.solution.getProblem().getReadyTime(candidate);
00308                         } else { costCand = timesUntilPre; }
00309                         costCand += this.solution.getProblem().getServiceTime(candidate);
00310                         if(costCand > this.solution.getProblem().getMaxTimePerRoute()) {
00311                             validFinalInsertion = false; }
00312
00313                         // Comprobar TW desde candidate hasta sucesor
00314                         double timesUntilSuc = costCand + this.solution.getDistance(candidate, suc);
00315                         if(timesUntilSuc < (this.solution.getProblem().getDueTime(suc))) {
00316
00317                             double costSuc = 0;
00318                             if(timesUntilSuc < this.solution.getProblem().getReadyTime(suc)) {
00319                                 costSuc = this.solution.getProblem().getReadyTime(suc);
00320                             } else { costSuc = timesUntilSuc; }
00321                             costSuc += this.solution.getProblem().getServiceTime(suc);
00322                             costInsertion = costSuc;
00323                             if(costSuc > this.solution.getProblem().getMaxTimePerRoute()) {
00324                                 validFinalInsertion = false; }
00325
00326                             int pre2=suc, suc2 = -1;
00327                             if(suc != depot)
00328                                 do {
00329                                     suc2 = this.solution.getSuccessor(pre2);
00330                                     double timesUntilSuc2 = costInsertion +
00331                                         this.solution.getDistance(pre2, suc2);
00332                                     if(timesUntilSuc2 < (this.solution.getProblem().getDueTime(suc2)))
00333                                     {
00334                                         if(this.solution.getProblem().getReadyTime(suc2)) {
00335                                             costInsertion =
00336                                             this.solution.getProblem().getReadyTime(suc2);
00337                                         } else { costInsertion = timesUntilSuc2; }
00338                                         costInsertion +=
00339                                         this.solution.getProblem().getServiceTime(suc2);
00340                                         if(costInsertion >
00341                                             this.solution.getProblem().getMaxTimePerRoute()) { validFinalInsertion = false; }
00342                                         } else { validFinalInsertion = false; }
00343                                         pre2 = suc2;
00344                                     } while((suc2 != depot) && validFinalInsertion);
00345                                     } else { validFinalInsertion = false; }
00346                                 } else { validFinalInsertion = false; }
00347
00348
00349
00350
00351
00352
00353
00354
00355
00356
00357
00358
00359
00360
00361
00362
00363
00364
00365
00366
00367
00368
00369
00370
00371
00372
00373
00374
00375
00376
00377
00378
00379
00380
00381
00382
00383
00384
00385
00386
00387
00388
00389
00390
00391
00392
00393
00394
00395
00396
00397
00398
00399
00400
00401
00402
00403
00404
00405
00406
00407
00408
00409
00410
00411
00412
00413
00414
00415
00416
00417
00418
00419
00420
00421
00422
00423
00424
00425
00426
00427
00428
00429
00430
00431
00432
00433
00434
00435
00436
00437
00438
00439
00440
00441
00442
00443
00444
00445
00446
00447
00448
00449
00450
00451
00452
00453
00454
00455
00456
00457
00458
00459
00460
00461
00462
00463
00464
00465
00466
00467
00468
00469
00470
00471
00472
00473
00474
00475
00476
00477
00478
00479
00480
00481
00482
00483
00484
00485
00486
00487
00488
00489
00490
00491
00492
00493
00494
00495
00496
00497
00498
00499
00500
00501
00502
00503
00504
00505
00506
00507
00508
00509
00510
00511
00512
00513
00514
00515
00516
00517
00518
00519
00520
00521
00522
00523
00524
00525
00526
00527
00528
00529
00530
00531
00532
00533
00534
00535
00536
00537
00538
00539
00540
00541
00542
00543
00544
00545
00546
00547
00548
00549
00550
00551
00552
00553
00554
00555
00556
00557
00558
00559
00560
00561
00562
00563
00564
00565
00566
00567
00568
00569
00570
00571
00572
00573
00574
00575
00576
00577
00578
00579
00580
00581
00582
00583
00584
00585
00586
00587
00588
00589
00590
00591
00592
00593
00594
00595
00596
00597
00598
00599
00600
00601
00602
00603
00604
00605
00606
00607
00608
00609
00610
00611
00612
00613
00614
00615
00616
00617
00618
00619
00620
00621
00622
00623
00624
00625
00626
00627
00628
00629
00630
00631
00632
00633
00634
00635
00636
00637
00638
00639
00640
00641
00642
00643
00644
00645
00646
00647
00648
00649
00650
00651
00652
00653
00654
00655
00656
00657
00658
00659
00660
00661
00662
00663
00664
00665
00666
00667
00668
00669
00670
00671
00672
00673
00674
00675
00676
00677
00678
00679
00680
00681
00682
00683
00684
00685
00686
00687
00688
00689
00690
00691
00692
00693
00694
00695
00696
00697
00698
00699
00700
00701
00702
00703
00704
00705
00706
00707
00708
00709
00710
00711
00712
00713
00714
00715
00716
00717
00718
00719
00720
00721
00722
00723
00724
00725
00726
00727
00728
00729
00730
00731
00732
00733
00734
00735
00736
00737
00738
00739
00740
00741
00742
00743
00744
00745
00746
00747
00748
00749
00750
00751
00752
00753
00754
00755
00756
00757
00758
00759
00760
00761
00762
00763
00764
00765
00766
00767
00768
00769
00770
00771
00772
00773
00774
00775
00776
00777
00778
00779
00780
00781
00782
00783
00784
00785
00786
00787
00788
00789
00790
00791
00792
00793
00794
00795
00796
00797
00798
00799
00800
00801
00802
00803
00804
00805
00806
00807
00808
00809
00810
00811
00812
00813
00814
00815
00816
00817
00818
00819
00820
00821
00822
00823
00824
00825
00826
00827
00828
00829
00830
00831
00832
00833
00834
00835
00836
00837
00838
00839
00840
00841
00842
00843
00844
00845
00846
00847
00848
00849
00850
00851
00852
00853
00854
00855
00856
00857
00858
00859
00860
00861
00862
00863
00864
00865
00866
00867
00868
00869
00870
00871
00872
00873
00874
00875
00876
00877
00878
00879
00880
00881
00882
00883
00884
00885
00886
00887
00888
00889
00890
00891
00892
00893
00894
00895
00896
00897
00898
00899
00900
00901
00902
00903
00904
00905
00906
00907
00908
00909
00910
00911
00912
00913
00914
00915
00916
00917
00918
00919
00920
00921
00922
00923
00924
00925
00926
00927
00928
00929
00930
00931
00932
00933
00934
00935
00936
00937
00938
00939
00940
00941
00942
00943
00944
00945
00946
00947
00948
00949
00950
00951
00952
00953
00954
00955
00956
00957
00958
00959
00960
00961
00962
00963
00964
00965
00966
00967
00968
00969
00970
00971
00972
00973
00974
00975
00976
00977
00978
00979
00980
00981
00982
00983
00984
00985
00986
00987
00988
00989
00990
00991
00992
00993
00994
00995
00996
00997
00998
00999
01000
01001
01002
01003
01004
01005
01006
01007
01008
01009
01010
01011
01012
01013
01014
01015
01016
01017
01018
01019
01020
01021
01022
01023
01024
01025
01026
01027
01028
01029
01030
01031
01032
01033
01034
01035
01036
01037
01038
01039
01040
01041
01042
01043
01044
01045
01046
01047
01048
01049
01050
01051
01052
01053
01054
01055
01056
01057
01058
01059
01060
01061
01062
01063
01064
01065
01066
01067
01068
01069
01070
01071
01072
01073
01074
01075
01076
01077
01078
01079
01080
01081
01082
01083
01084
01085
01086
01087
01088
01089
01090
01091
01092
01093
01094
01095
01096
01097
01098
01099
01100
01101
01102
01103
01104
01105
01106
01107
01108
01109
01110
01111
01112
01113
01114
01115
01116
01117
01118
01119
01120
01121
01122
01123
01124
01125
01126
01127
01128
01129
01130
01131
01132
01133
01134
01135
01136
01137
01138
01139
01140
01141
01142
01143
01144
01145
01146
01147
01148
01149
01150
01151
01152
01153
01154
01155
01156
01157
01158
01159
01160
01161
01162
01163
01164
01165
01166
01167
01168
01169
01170
01171
01172
01173
01174
01175
01176
01177
01178
01179
01180
01181
01182
01183
01184
01185
01186
01187
01188
01189
01190
01191
01192
01193
01194
01195
01196
01197
01198
01199
01200
01201
01202
01203
01204
01205
01206
01207
01208
01209
01210
01211
01212
01213
01214
01215
01216
01217
01218
01219
01220
01221
01222
01223
01224
01225
01226
01227
01228
01229
01230
01231
01232
01233
01234
01235
01236
01237
01238
01239
01240
01241
01242
01243
01244
01245
01246
01247
01248
01249
01250
01251
01252
01253
01254
01255
01256
01257
01258
01259
01260
01261
01262
01263
01264
01265
01266
01267
01268
01269
01270
01271
01272
01273
01274
01275
01276
01277
01278
01279
01280
01281
01282
01283
01284
01285
01286
01287
01288
01289
01290
01291
01292
01293
01294
01295
01296
01297
01298
01299
01300
01301
01302
01303
01304
01305
01306
01307
01308
01309
01310
01311
01312
01313
01314
01315
01316
01317
01318
01319
01320
01321
01322
01323
01324
01325
01326
01327
01328
01329
01330
01331
01332
01333
01334
01335
01336
01337
01338
01339
01340
01341
01342
01343
01344
01345
01346
01347
01348
01349
01350
01351
01352
01353
01354
01355
01356
01357
01358
01359
01360
01361
01362
01363
01364
01365
01366
01367
01368
01369
01370
01371
01372
01373
01374
01375
01376
01377
01378
01379
01380
01381
01382
01383
01384
01385
01386
01387
01388
01389
01390
01391
01392
01393
01394
01395
01396
01397
01398
01399
01400
01401
01402
01403
01404
01405
01406
01407
01408
01409
01410
01411
01412
01413
01414
01415
01416
01417
01418
01419
01420
01421
01422
01423
01424
01425
01426
01427
01428
01429
01430
01431
01432
01433
01434
01435
01436
01437
01438
01439
01440
01441
01442
01443
01444
01445
01446
01447
01448
01449
01450
01451
01452
01453
01454
01455
01456
01457
01458
01459
01460
01461
01462
01463
01464
01465
01466
01467
01468
01469
01470
01471
01472
01473
01474
01475
01476
01477
01478
01479
01480
01481
01482
01483
01484
01485
01486
01487
01488
01489
01490
01491
01492
01493
01494
01495
01496
01497
01498
01499
01500
01501
01502
01503
01504
01505
01506
01507
01508
01509
01510
01511
01512
01513
01514
01515
01516
01517
01518
01519
01520
01521
01522
01523
01524
01525
01526
01527
01528
01529
01530
01531
01532
01533
01534
01535
01536
01537
01538
01539
01540
01541
01542
01543
01544
01545
01546
01547
01548
01549
01550
01551
01552
01553
01554
01555
01556
01557
01558
01559
01560
01561
01562
01563
01564
01565
01566
01567
01568
01569
01570
01571
01572
01573
01574
01575
01576
01577
01578
01579
01580
01581
01582
01583
01584
01585
01586
01587
01588
01589
01590
01591
01592
01593
01594
01595
01596
01597
01598
01599
01600
01601
01602
01603
01604
01605
01606
01607
01608
01609
01610
01611
01612
01613
01614
01615
01616
01617
01618
01619
01620
01621
01622
01623
01624
01625
01626
01627
01628
01629
01630
01631
01632
01633
01634
01635
01636
01637
01638
01639
01640
01641
01642
01643
01644
01645
01646
01647
01648
01649
01650
01651
01652
01653
01654
01655
01656
01657
01658
01659
01660
01661
01662
01663
01664
01665
01666
01667
01668
01669
01670
01671
01672
01673
01674
01675
01676
01677
01678
01679
01680
01681
01682
01683
01684
01685
01686
01687
01688
01689
01690
01691
01692
01693
01694
01695
01696
01697
01698
01699
01700
01701
01702
01703
01704
01705
01706
01707
01708
01709
01710
01711
01712
01713
01714
01715
01716
01717
01718
01719
01720
01721
01722
01723
01724
01725
01726
01727
01728
01729
01729
01730
01731
01732
01733
01734
01735
01736
01737
01738
01739
01740
01741
01742
01743
01744
01745
01746
01747
01748
01749
01750
01751
01752
01753
01754
01755
01756
01757
01758
01759
01759
01760
01761
01762
01763
01764
01765
01766
01767
01768
01769
01770
01771
01772
01773
01774
01775
01776
01777
01778
01779
01779
01780
01781
01782
01783
01784
01785
01786
01787
01788
01789
01790
01791
01792
01793
01794
01795
01796
01797
01798
01799
01800
01801
01802
01803
01804
01805
01806
01807
01808
01809
01809
01810
01811
01812
01813
01814
01815
01816
01817
01818
01819
01820
01821
01822
01823
01824
01825
01826
01827
01828
01829
01829
01830
01831
01832
01833
01834
01835
01836
01837
01838
01839
01839
01840
01841
01842
01843
01844
01845
01846
01847
01848
01849
01849
01850
01851
01852
01853
01854
01855
01856
01857
01858
01859
01859
01860
01861
01862
01863
01864
01865
01866
01867
01868
01869
01869
01870
01871
01872
01873
01874
01875
01876
01877
01878
01879
01879
01880
01881
01882
01883
01884
01885
01886
01887
01888
01889
01889
01890
01891
01892
01893
01894
01895
01896
01897
01898
01899
01899
01900
01901
01902
01903
01904
01905
01906
01907
01908
01909
01909
01910
01911
01912
01913
01914
01915
01916
01917
01918
01919
01919
01920
01921
01922
01923
01924
01925
01926
0
```

```

00337             if(validFinalInsertion==true) { // cliente, ruta, predecesor, coste
00338                 if(costInsertion < infoCandidate[3]) {
00339                     infoCandidate[0] = candidate; infoCandidate[1] = k; infoCandidate[2] =
00340                     pre; infoCandidate[3] = costInsertion; infoCandidate[4] =
00341                     this.solution.getProblem().getScore(candidate); // cliente, ruta, predecesor, coste, score
00342                 }
00343             }
00344             pre = suc;
00345         } while(suc != depot);
00346     } //rutas creadas
00347
00348     // almacenamos en la lista de candidatos la mejor posición de inserción para el cliente
00349     if(infoCandidate[0]!=-1 && infoCandidate[1]!=-1 && infoCandidate[2]!=-1 &&
00350         infoCandidate[3] != Double.MAX_VALUE && infoCandidate[4]!=-1) {
00351         double[] infoCandidate2 = new double[5];
00352         infoCandidate2[0] = infoCandidate[0]; infoCandidate2[1] = infoCandidate[1];
00353         infoCandidate2[2] = infoCandidate[2]; infoCandidate2[3] = infoCandidate[3];
00354         infoCandidate2[4] = infoCandidate[4];
00355         candidatesList.add(infoCandidate2);
00356     }
00357     validFinalInsertion = true;
00358     infoCandidate[0] = -1; infoCandidate[1] = -1;
00359     infoCandidate[2] = -1; infoCandidate[3] = Double.MAX_VALUE;
00360     infoCandidate[4] = -1;
00361 } // cliente
00362
00363     return candidatesList;
00363 }
```

Gráfico de llamadas de esta función:

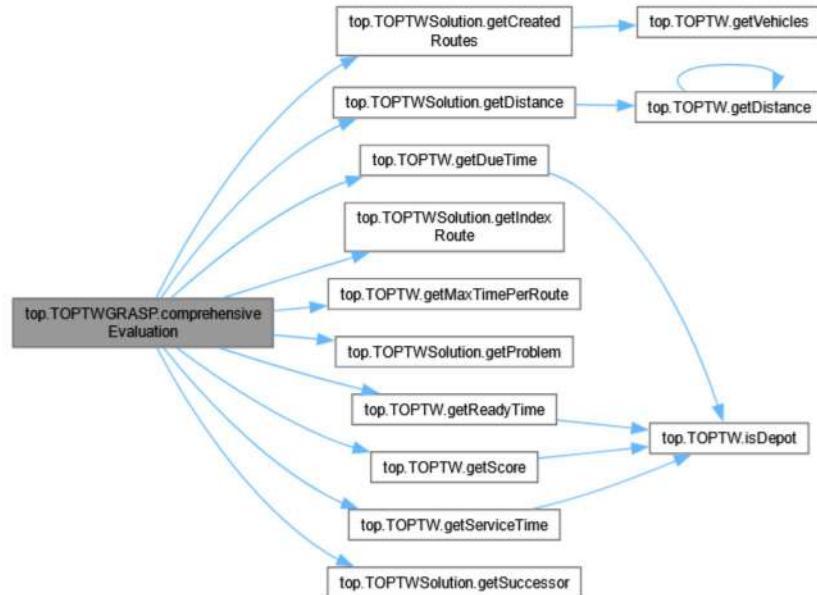


Gráfico de llamadas a esta función:



### 7.9.3.3. computeGreedySolution()

```
void top.TOPTWGRASP.computeGreedySolution (
    int maxSizeRCL) [inline]
```

Construye una solución greedy aleatorizada.

Itera mientras haya clientes sin asignar, evaluando los candidatos, construyendo la RCL, seleccionando un candidato y actualizando la solución.

#### Parámetros

|                          |                   |                             |
|--------------------------|-------------------|-----------------------------|
| <input type="checkbox"/> | <i>maxSizeRCL</i> | El tamaño máximo de la RCL. |
|--------------------------|-------------------|-----------------------------|

Definición en la línea 153 del archivo [TOPTWGRASP.java](#).

```
00153             {
00154         // inicialización
00155         this.solution.initSolution();
00156
00157         // tiempo de salida y score por ruta y cliente
00158         ArrayList<ArrayList<Double>> departureTimesPerClient = new ArrayList<ArrayList<Double>>();
00159         ArrayList<Double> init = new ArrayList<Double>();
00160         for(int z = 0; z <
            this.solution.getProblem().getPOIs() + this.solution.getProblem().getVehicles(); z++) { init.add(0.0);}
00161         departureTimesPerClient.add(0, init);
00162
00163         // clientes
00164         ArrayList<Integer> customers = new ArrayList<Integer>();
00165         for(int j = 1; j <= this.solution.getProblem().getPOIs(); j++) { customers.add(j); }
00166
00167         // Evaluar coste incremental de los elementos candidatos
00168         ArrayList< double[] > candidates = this.comprehensiveEvaluation(customers,
00169             departureTimesPerClient);
00170
00171         Collections.sort(candidates, new Comparator<double[]>() {
00172             public int compare(double[] a, double[] b) {
00173                 return Double.compare(a[a.length-2], b[b.length-2]);
00174             }
00175         });
00176
00177         int maxTRCL = maxSizeRCL;
00178         boolean existCandidates = true;
00179
00180         while(!customers.isEmpty() && existCandidates) {
00181             if(!candidates.isEmpty()) {
00182                 //Construir lista restringida de candidatos
00183                 ArrayList< double[] > rcl = new ArrayList< double[] >();
00184                 maxTRCL = maxSizeRCL;
00185                 if(maxTRCL > candidates.size()) { maxTRCL = candidates.size(); }
00186                 for(int j=0; j < maxTRCL; j++) { rcl.add(candidates.get(j)); }
00187
00188                 //Selección aleatoria o fuzzy de candidato de la lista restringida
00189                 int posSelected = -1;
00190                 int selection = 3;
00191                 double alpha = 0.8;
00192                 switch (selection) {
00193                     case 1: posSelected = this.aleatorySelectionRCL(maxTRCL); // Selección aleatoria
00194                     break;
00195                     case 2: posSelected = this.fuzzySelectionBestFDRCL(rcl); // Selección fuzzy con
00196                         mejor valor de alpha
00197                         break;
00198                     case 3: posSelected = this.fuzzySelectionAlphaCutRCL(rcl, alpha); // Selección
00199                         fuzzy con alpha corte aleatoria
00200                         break;
00201                     default: posSelected = this.aleatorySelectionRCL(maxTRCL); // Selección aleatoria
00202                         por defecto
00203                         break;
00204                 }
00205
00206                 double[] candidateSelected = rcl.get(posSelected);
00207                 for(int j=0; j < customers.size(); j++) {
00208                     if(customers.get(j)==candidateSelected[0]) {
00209                         customers.remove(j);
00210                     }
00211                 }
00212
00213                 updateSolution(candidateSelected, departureTimesPerClient);
00214             }
00215         }
00216     }
```

```

00210
00211     } else { // No hay candidatos a insertar en la solución, crear otra ruta
00212         if(this.solution.getCreatedRoutes() < this.solution.getProblem().getVehicles()) {
00213             int newDepot = this.solution.addRoute();
00214             ArrayList<Double> initNew = new ArrayList<Double>();
00215             for(int z = 0; z <
00216                 this.solution.getProblem().getPOIs() + this.solution.getProblem().getVehicles(); z++)
00217                 initNew.add(0.0);
00218             departureTimesPerClient.add(initNew);
00219         }
00220         existCandidates = false;
00221     }
00222     //Reevaluar coste incremental de los elementos candidatos
00223     candidates.clear();
00224     candidates = this.comprehensiveEvaluation(customers, departureTimesPerClient);
00225     Collections.sort(candidates, new Comparator<double[]>() {
00226         public int compare(double[] a, double[] b) {
00227             return Double.compare(a[a.length-2], b[b.length-2]);
00228         }
00229     });
00230 }
00231
00232 }

```

Gráfico de llamadas de esta función:

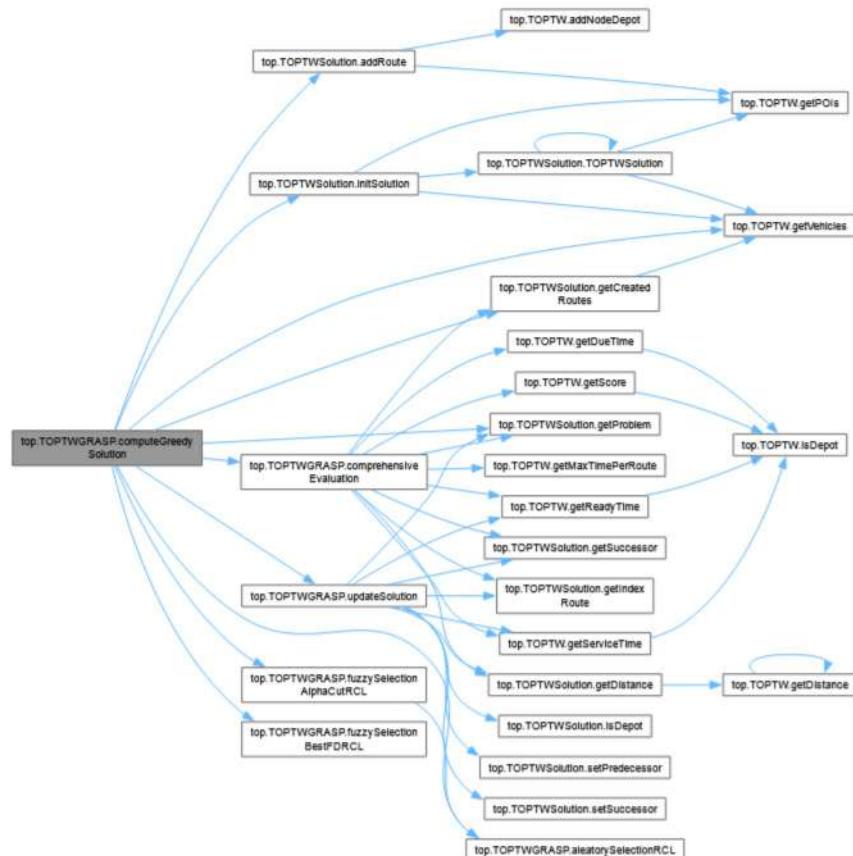


Gráfico de llamadas a esta función:



#### 7.9.3.4. `fuzzySelectionAlphaCutRCL()`

```
int top.TOPTWGRASP.fuzzySelectionAlphaCutRCL (
    ArrayList< double[] > rcl,
    double alpha) [inline]
```

Selecciona un candidato de la RCL utilizando un corte alfa sobre la función de pertenencia difusa.

Filtrá los candidatos cuyo valor de membresía es menor o igual a `alpha` y luego elige uno aleatoriamente. Si ningún candidato cumple la condición, se elige uno aleatoriamente de toda la RCL.

##### Parámetros

|                    |                                     |
|--------------------|-------------------------------------|
| <code>rcl</code>   | La Lista Restringida de Candidatos. |
| <code>alpha</code> | El umbral para el corte alfa.       |

##### Devuelve

La posición del candidato seleccionado en la RCL.

Definición en la línea 131 del archivo [TOPTWGRASP.java](#).

```
00131
00132     ArrayList<Integer> candidates = new ArrayList<>();
00133     for(int i = 0; i < rcl.size(); i++) {
00134         if(rcl.get(i)[2] <= alpha) {
00135             candidates.add(i);
00136         }
00137     }
00138     if(candidates.isEmpty()) {
00139         return aleatorySelectionRCL(rcl.size());
00140     }
00141     else {
00142         int aleatory = RANDOM.nextInt(candidates.size());
00143         return candidates.get(aleatory);
00144     }
00145 }
```

Gráfico de llamadas de esta función:

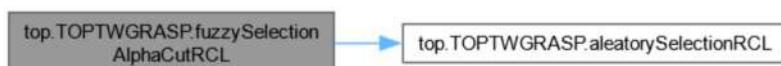


Gráfico de llamadas a esta función:



#### 7.9.3.5. `fuzzySelectionBestFDRCL()`

```
int top.TOPTWGRASP.fuzzySelectionBestFDRCL (
    ArrayList< double[] > rcl) [inline]
```

Selecciona un candidato de la RCL utilizando una función de pertenencia difusa.

Eige el candidato con el mejor valor de membresía (el más "prometedor" según la puntuación).

##### Parámetros

|                  |                                     |
|------------------|-------------------------------------|
| <code>rcl</code> | La Lista Restringida de Candidatos. |
|------------------|-------------------------------------|

##### Devuelve

La posición del candidato seleccionado en la RCL.

Definición en la línea 111 del archivo [TOPTWGRASP.java](#).

```
00111
00112     int bestPosition = 0;
00113     double bestValue = -1.0;
00114     for(int i = 0; i < rcl.size(); i++) {
00115         if(bestValue < rcl.get(i)[2]) {
00116             bestValue = rcl.get(i)[2];
00117             bestPosition = i;
00118         }
00119     }
00120     return bestPosition;
00121 }
```

Gráfico de llamadas a esta función:



#### 7.9.3.6. `getMaxScore()`

```
double top.TOPTWGRASP.getMaxScore () [inline]
```

Obtiene la máxima puntuación posible entre todos los nodos del problema.

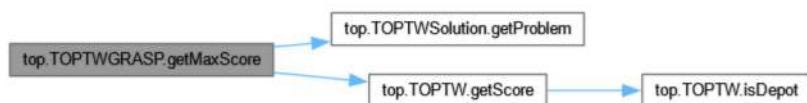
**Devuelve**

El valor de la máxima puntuación.

Definición en la línea 401 del archivo [TOPTWGRASP.java](#).

```
00401     {
00402         double maxSc = -1.0;
00403         for(int i = 0; i < this.solution.getProblem().getScore().length; i++) {
00404             if(this.solution.getProblem().getScore(i) > maxSc)
00405                 maxSc = this.solution.getProblem().getScore(i);
00406         }
00407     return maxSc;
00408 }
```

Gráfico de llamadas de esta función:

**7.9.3.7. getSolution()**

[TOPTWSolution top.TOPTWGRASP.getSolution \(\) \[inline\]](#)

Obtiene la solución actual.

**Devuelve**

La instancia de [TOPTWSolution](#).

Definición en la línea 369 del archivo [TOPTWGRASP.java](#).

```
00369     {
00370         return solution;
00371     }
```

**7.9.3.8. getSolutionTime()**

[int top.TOPTWGRASP.getSolutionTime \(\) \[inline\]](#)

Obtiene el tiempo de ejecución de la solución (actualmente no implementado).

**Devuelve**

El tiempo de solución.

Definición en la línea 385 del archivo [TOPTWGRASP.java](#).

```
00385     {
00386         return solutionTime;
00387     }
```

**7.9.3.9. GRASP()**

```
void top.TOPTWGRASP.GRASP (
    int maxIterations,
    int maxSizeRCL) [inline]
```

Ejecuta el algoritmo GRASP durante un número determinado de iteraciones.

En cada iteración, construye una solución greedy aleatorizada, (opcionalmente aplica una búsqueda local), y actualiza la mejor solución encontrada.

**Parámetros**

|                      |   |
|----------------------|---|
| <i>maxIterations</i> | El número máximo de iteraciones a ejecutar.                   |
| <i>maxSizeRCL</i>    | El tamaño máximo de la Lista Restringida de Candidatos (RCL). |

Definición en la línea 67 del archivo [TOPTWGRASP.java](#).

```
00067             {
00068         double averageFitness = 0.0;
00069         double bestSolution = 0.0;
00070         for(int i = 0; i < maxIterations; i++) {
00071             this.computeGreedySolution(maxSizeRCL);
00073
00074             // IMPRIMIR SOLUCION
00075             double fitness = this.solution.evaluateFitness();
00076             System.out.println(this.solution.getInfoSolution());
00077             //System.out.println("Press Any Key To Continue..."); 
00078             //new java.util.Scanner(System.in).nextLine();
00079             averageFitness += fitness;
00080             if(bestSolution < fitness) {
00081                 bestSolution = fitness;
00082             }
00083             //double fitness = this.solution.printSolution();
00084
00085             *****
00086             *
00087             * BÚSQUEDA LOCAL
00088             *
00089             */
00090         }
00091         averageFitness = averageFitness/maxIterations;
00092         System.out.println(" --> MEDIA: "+averageFitness);
00093         System.out.println(" --> MEJOR SOLUCION: "+bestSolution);
00094     }
```

Gráfico de llamadas de esta función:

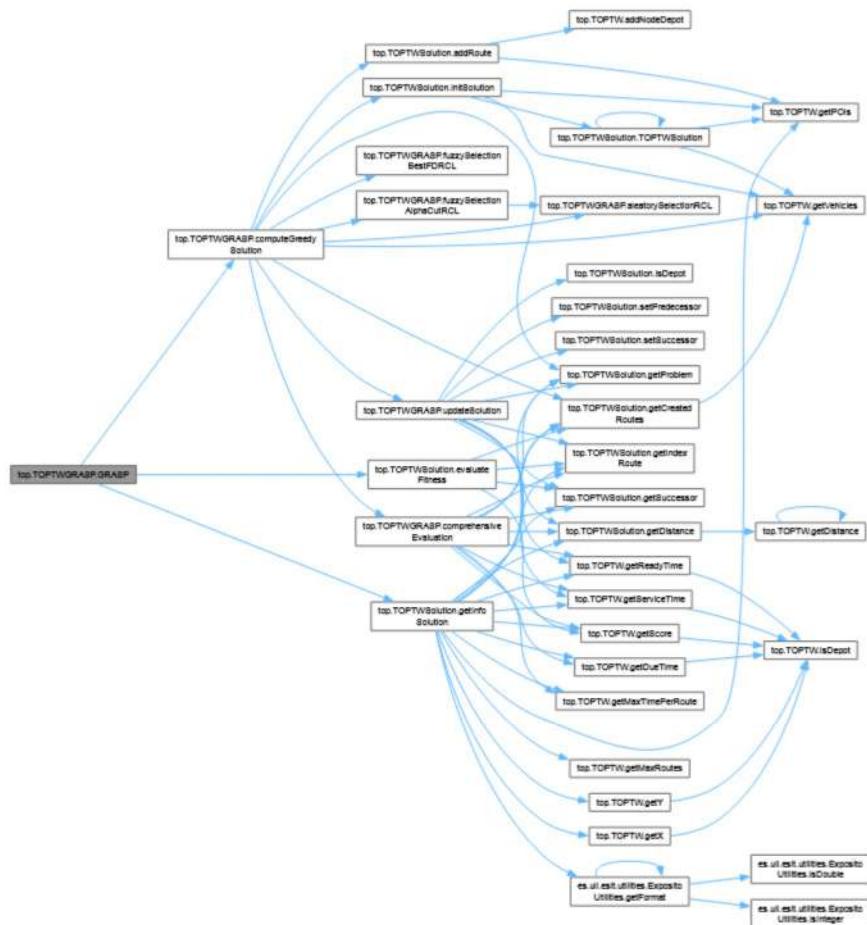


Gráfico de llamadas a esta función:



### 7.9.3.10. setSolution()

```
void top.TOPTWGRASP.setSolution (
    TOPTWSolution solution) [inline]
```

Establece la solución a utilizar.

**Parámetros**

|                          |                 |   |
|--------------------------|-----------------|---|
| <input type="checkbox"/> | <i>solution</i> | La nueva instancia de <a href="#">TOPTWSolution</a> . |
|--------------------------|-----------------|---|

Definición en la línea 377 del archivo [TOPTWGRASP.java](#).

```
00377
00378     this.solution = solution;
00379 }
```

**7.9.3.11. setSolutionTime()**

```
void top.TOPTWGRASP.setSolutionTime (
    int solutionTime) [inline]
```

Establece el tiempo de ejecución de la solución.

**Parámetros**

|                          |                     |                  |
|--------------------------|---------------------|------------------|
| <input type="checkbox"/> | <i>solutionTime</i> | El nuevo tiempo. |
|--------------------------|---------------------|------------------|

Definición en la línea 393 del archivo [TOPTWGRASP.java](#).

```
00393
00394     this.solutionTime = solutionTime;
00395 }
```

**7.9.3.12. updateSolution()**

```
void top.TOPTWGRASP.updateSolution (
    double[] candidateSelected,
    ArrayList< ArrayList< Double > > departureTimes) [inline]
```

Actualiza la solución insertando un candidato seleccionado.

Modifica los punteros de predecesor y sucesor para insertar el nuevo nodo en la ruta y recalcula los tiempos de salida de los nodos afectados en esa ruta.

**Parámetros**

|                          |                          |   |
|--------------------------|--------------------------|---|
| <input type="checkbox"/> | <i>candidateSelected</i> | El candidato seleccionado, un array con la información [cliente, ruta, predecesor, coste, score]. |
|                          | <i>departureTimes</i>    | La estructura de datos que almacena los tiempos de salida de cada cliente.                        |

Definición en la línea 241 del archivo [TOPTWGRASP.java](#).

```
00241
00242     // Inserción del cliente en la ruta return: cliente, ruta, predecesor, coste
00243     this.solution.setPredecessor((int)candidateSelected[0], (int)candidateSelected[2]);
00244     this.solution.setSuccessor((int)candidateSelected[0],
00245         this.solution.getSuccessor((int)candidateSelected[2]));
00246     this.solution.setSuccessor((int)candidateSelected[2], (int)candidateSelected[0]);
00247     this.solution.setPredecessor(this.solution.getSuccessor((int)candidateSelected[0]),
00248         (int)candidateSelected[0]);
00249     // Actualización de las estructuras de datos y conteo a partir de la posición a insertar
00250     double costInsertionPre =
00251         departureTimes.get((int)candidateSelected[1]).get((int)candidateSelected[2]);
00252     ArrayList<Double> route = departureTimes.get((int)candidateSelected[1]);
00253     int pre=(int)candidateSelected[2], suc=-1;
00254     int depot = this.solution.getIndexRoute((int)candidateSelected[1]);
```

```

00253     do {
00254         suc = this.solution.getSuccessor(pre);
00255         costInsertionPre += this.solution.getDistance(pre, suc);
00256
00257         if(costInsertionPre < this.solution.getProblem().getReadyTime(suc)) {
00258             costInsertionPre = this.solution.getProblem().getReadyTime(suc);
00259         }
00260         costInsertionPre += this.solution.getProblem().getServiceTime(suc);
00261
00262         if(!this.solution.isDepot(suc))
00263             route.set(suc, costInsertionPre);
00264         pre = suc;
00265     } while((suc != depot));
00266
00267     // Actualiza tiempos
00268     departureTimes.set((int)candidateSelected[1], route);
00269 }

```

Gráfico de llamadas de esta función:

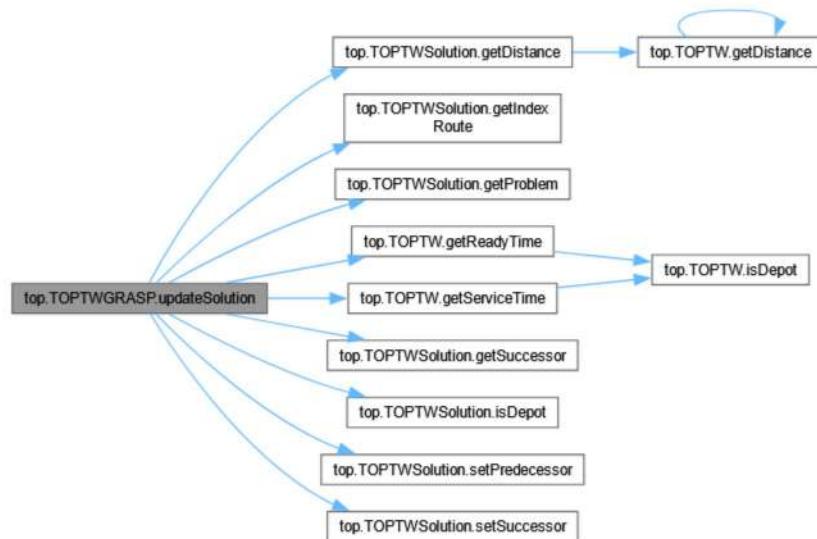


Gráfico de llamadas a esta función:



## 7.9.4. Documentación de datos miembro

### 7.9.4.1. NO\_EVALUATED

```
double top.TOPTWGRASP.NO_EVALUATED = -1.0 [static]
```

Constante para indicar que una solución no ha sido evaluada.

Definición en la línea 23 del archivo [TOPTWGRASP.java](#).

#### 7.9.4.2. RANDOM

```
final Random top.TOPTWGRASP.RANDOM = new SecureRandom() [static], [private]
```

Definición en la línea 24 del archivo [TOPTWGRASP.java](#).

#### 7.9.4.3. solution

```
TOPTWSolution top.TOPTWGRASP.solution [private]
```

Definición en la línea 26 del archivo [TOPTWGRASP.java](#).

#### 7.9.4.4. solutionTime

```
int top.TOPTWGRASP.solutionTime [private]
```

Definición en la línea 27 del archivo [TOPTWGRASP.java](#).

La documentación de esta clase está generada del siguiente archivo:

- C:/Users/Kefle/Desktop/ExpositoTOP/src/top/[TOPTWGRASP.java](#)

## 7.10. Referencia de la clase top.TOPTWReader

Proporciona funcionalidad para leer una instancia del problema [TOPTW](#) desde un archivo de texto.

### Métodos públicos estáticos

- static [TOPTW](#) [readProblem](#) (String filePath)

*Lee una instancia del problema [TOPTW](#) desde un archivo.*

### 7.10.1. Descripción detallada

Proporciona funcionalidad para leer una instancia del problema [TOPTW](#) desde un archivo de texto.

El método estático `readProblem` se encarga de parsear el archivo y construir un objeto [TOPTW](#).

Definición en la línea 19 del archivo [TOPTWReader.java](#).

### 7.10.2. Documentación de funciones miembro

#### 7.10.2.1. [readProblem\(\)](#)

```
TOPTW top.TOPTWReader.readProblem (
    String filePath) [inline], [static]
```

Lee una instancia del problema [TOPTW](#) desde un archivo.

Parsea un archivo de texto con un formato específico para inicializar un objeto [TOPTW](#). El formato esperado es el de las instancias de Solomon, adaptado para [TOPTW](#). La primera línea define el número de vehículos y clientes. Las siguientes líneas definen los datos de cada cliente (coordenadas, tiempo de servicio, puntuación, ventanas de tiempo).

**Parámetros**

|                          |                 |                                     |
|--------------------------|-----------------|-------------------------------------|
| <input type="checkbox"/> | <i>filePath</i> | La ruta al archivo de la instancia. |
|--------------------------|-----------------|-------------------------------------|

**Devuelve**

Un objeto [TOPTW](#) con los datos del problema cargados.

**Excepciones**

|                          |                    |  |
|--------------------------|--------------------|--|
| <input type="checkbox"/> | <i>IOException</i> | Si ocurre un error al leer el archivo. |
|--------------------------|--------------------|--|

Definición en la línea [31](#) del archivo [TOPTWReader.java](#).

```

00031
00032     TOPTW problem = null;
00033     BufferedReader reader = null;
00034     try {
00035         File instaceFile = new File(filePath);
00036         reader = new BufferedReader(new FileReader(instaceFile));
00037         String line = reader.readLine();
00038         line = ExpositoUtilities.simplifyString(line);
00039         String[] parts = line.split(" ");
00040         problem = new TOPTW(Integer.parseInt(parts[2]), Integer.parseInt(parts[1]));
00041         line = reader.readLine();
00042         line = null; parts = null;
00043         for (int i = 0; i < problem.getPOIs() + 1; i++) {
00044             line = reader.readLine();
00045             line = ExpositoUtilities.simplifyString(line);
00046             parts = line.split(" ");
00047             problem.setX(i, Double.parseDouble(parts[1]));
00048             problem.setY(i, Double.parseDouble(parts[2]));
00049             problem.setServiceTime(i, Double.parseDouble(parts[3]));
00050             problem.setScore(i, Double.parseDouble(parts[4]));
00051             if(i==0) {
00052                 problem.setReadyTime(i, Double.parseDouble(parts[7]));
00053                 problem.setDueTime(i, Double.parseDouble(parts[8]));
00054             }
00055             else {
00056                 problem.setReadyTime(i, Double.parseDouble(parts[8]));
00057                 problem.setDueTime(i, Double.parseDouble(parts[9]));
00058             }
00059             line = null; parts = null;
00060         }
00061         problem.calculateDistanceMatrix();
00062     } catch (IOException e) {
00063         System.err.println(e);
00064         System.exit(0);
00065     } finally {
00066         if (reader != null) {
00067             try {
00068                 reader.close();
00069             } catch (IOException ex) {
00070                 System.err.println(ex);
00071                 System.exit(0);
00072             }
00073         }
00074     }
00075     problem.setMaxTimePerRoute(problem.getDueTime(0));
00076     return problem;
00077 }
```

Gráfico de llamadas de esta función:

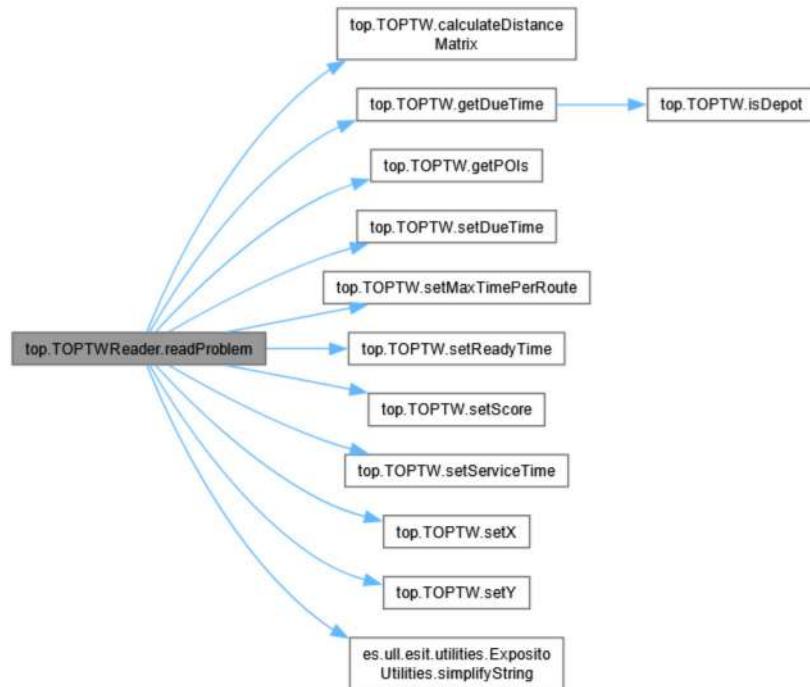


Gráfico de llamadas a esta función:



La documentación de esta clase está generada del siguiente archivo:

- C:/Users/Kefle/Desktop/ExpositoTOP/src/top/[TOPTWReader.java](#)

## 7.11. Referencia de la clase top.TOPTWRoute

Modela una ruta simple mediante la identificación de sus nodos predecesor y sucesor.

## Métodos públicos

- int [getPredecesor \(\)](#)  
*Obtiene el predecesor.*
- int [getSucesor \(\)](#)  
*Obtiene el sucesor.*
- int [getId \(\)](#)  
*Obtiene el ID de la ruta.*
- void [setPredecesor \(int pre\)](#)  
*Establece el predecesor.*
- void [setSucesor \(int suc\)](#)  
*Establece el sucesor.*
- void [setId \(int id\)](#)  
*Establece el ID de la ruta.*

### 7.11.1. Descripción detallada

Modela una ruta simple mediante la identificación de sus nodos predecesor y sucesor.

Esta clase es una estructura de datos básica para almacenar información sobre un segmento de ruta.

Definición en la línea 12 del archivo [TOPTWRoute.java](#).

### 7.11.2. Documentación de funciones miembro

#### 7.11.2.1. getId()

```
int top.TOPTWRoute.getId () [inline]
```

Obtiene el ID de la ruta.

Devuelve

El identificador de la ruta.

Definición en la línea 65 del archivo [TOPTWRoute.java](#).

```
00065      {
00066          return this.id;
00067      }
```

#### 7.11.2.2. getPredecesor()

```
int top.TOPTWRoute.getPredecesor () [inline]
```

Obtiene el predecesor.

Devuelve

El índice del nodo predecesor.

Definición en la línea 49 del archivo [TOPTWRoute.java](#).

```
00049      {
00050          return this.predecessor;
00051      }
```

### 7.11.2.3. getSuccesor()

```
int top.TOPTWRoute.getSuccesor () [inline]
```

Obtiene el sucesor.

Devuelve

El índice del nodo sucesor.

Definición en la línea 57 del archivo [TOPTWRoute.java](#).

```
00057             {  
00058         return this.succesor;  
00059     }
```

### 7.11.2.4. setId()

```
void top.TOPTWRoute.setId (  
    int id) [inline]
```

Establece el ID de la ruta.

Parámetros

|                                |                                    |
|--------------------------------|------------------------------------|
| <input type="text"/> <i>id</i> | El nuevo identificador de la ruta. |
|--------------------------------|------------------------------------|

Definición en la línea 89 del archivo [TOPTWRoute.java](#).

```
00089             {  
00090         this.id = id;  
00091     }
```

### 7.11.2.5. setPredeccesor()

```
void top.TOPTWRoute.setPredeccesor (  
    int pre) [inline]
```

Establece el predecesor.

Parámetros

|                                 |                                      |
|---------------------------------|--------------------------------------|
| <input type="text"/> <i>pre</i> | El nuevo índice del nodo predecesor. |
|---------------------------------|--------------------------------------|

Definición en la línea 73 del archivo [TOPTWRoute.java](#).

```
00073             {  
00074         this.predecessor = pre;  
00075     }
```

### 7.11.2.6. setSuccesor()

```
void top.TOPTWRoute.setSuccesor (  
    int suc) [inline]
```

Establece el sucesor.

### Parámetros

|                          |     |                                   |
|--------------------------|-----|-----------------------------------|
| <input type="checkbox"/> | suc | El nuevo índice del nodo sucesor. |
|--------------------------|-----|-----------------------------------|

Definición en la línea 81 del archivo [TOPTWRoute.java](#).

```
00081
00082     this.sucesor = suc;
00083 }
```

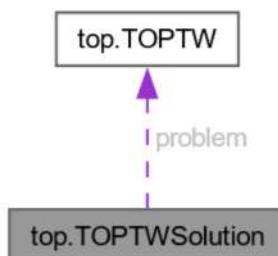
La documentación de esta clase está generada del siguiente archivo:

- C:/Users/Kefle/Desktop/ExpositoTOP/src/top/[TOPTWRoute.java](#)

## 7.12. Referencia de la clase top.TOPTWSolution

Representa una solución a una instancia del problema [TOPTW](#).

Diagrama de colaboración de top.TOPTWSolution:



### Métodos públicos

- [TOPTWSolution \(TOPTW problem\)](#)  
*Constructor de la clase TOPTWSolution.*
- [void initSolution \(\)](#)  
*Inicializa o resetea la solución a un estado básico.*
- [boolean isDepot \(int c\)](#)  
*Comprueba si un nodo es un depósito.*
- [boolean equals \(TOPTWSolution otherSolution\)](#)  
*Compara si dos soluciones son iguales basándose en sus arrays de predecesores.*
- [boolean equals \(Object o\)](#)  
*Compara si dos soluciones son iguales basándose en sus arrays de predecesores.*
- [int hashCode \(\)](#)
- [int getAvailableVehicles \(\)](#)  
*Obtiene el número de vehículos que aún no han sido asignados a una ruta.*
- [int getCreatedRoutes \(\)](#)

- **double getDistance (int x, int y)**  
*Obtiene la distancia entre dos nodos consultando el problema asociado.*
- **void setAvailableVehicles (int availableVehicles)**  
*Establece el número de vehículos disponibles.*
- **int getPredecessor (int customer)**  
*Obtiene el predecesor de un cliente en su ruta.*
- **int[] getPredecessors ()**  
*Obtiene el array completo de predecesores.*
- **TOPTW getProblem ()**  
*Obtiene la instancia del problema asociada a esta solución.*
- **double getObjectiveFunctionValue ()**  
*Obtiene el valor de la función objetivo (puntuación total) de la solución.*
- **int getPositionInRoute (int customer)**  
*Obtiene la posición de un cliente dentro de su ruta.*
- **int getSuccessor (int customer)**  
*Obtiene el sucesor de un cliente en su ruta.*
- **int[] getSuccessors ()**  
*Obtiene el array completo de sucesores.*
- **int getIndexRoute (int index)**  
*Obtiene el índice del nodo que actúa como depósito para una ruta específica.*
- **double getWaitingTime (int customer)**  
*Obtiene el tiempo de espera en un nodo cliente.*
- **void setObjectiveFunctionValue (double objectiveFunctionValue)**  
*Establece el valor de la función objetivo.*
- **void setPositionInRoute (int customer, int position)**  
*Establece la posición de un cliente en su ruta.*
- **void setPredecessor (int customer, int predecessor)**  
*Establece el predecesor de un nodo.*
- **void setSuccessor (int customer, int successor)**  
*Establece el sucesor de un nodo.*
- **void setWaitingTime (int customer, int waitingTime)**  
*Establece el tiempo de espera en un nodo.*
- **String getInfoSolution ()**  
*Genera una cadena con información detallada de la solución.*
- **double evaluateFitness ()**  
*Evalúa la función objetivo de la solución (puntuación total).*
- **int addRoute ()**  
*Añade una nueva ruta a la solución.*
- **double printSolution ()**  
*Imprime la solución en la consola y devuelve la puntuación.*

### Atributos públicos estáticos

- **static final int NO\_INITIALIZED = -1**  
*Constante para indicar que un valor no ha sido inicializado.*

## Atributos privados

- `TOPTW problem`
- `int[] predecessors`
- `int[] successors`
- `double[] waitingTime`
- `int[] positionInRoute`
- `int[] routes`
- `int availableVehicles`
- `double objectiveFunctionValue`

### 7.12.1. Descripción detallada

Representa una solución a una instancia del problema `TOPTW`.

Almacena la estructura de las rutas (mediante predecesores y sucesores), los tiempos de espera, y el valor de la función objetivo. Proporciona métodos para construir, modificar y evaluar la solución.

Definición en la línea 17 del archivo `TOPTWSolution.java`.

### 7.12.2. Documentación de constructores y destructores

#### 7.12.2.1. `TOPTWSolution()`

```
top.TOPTWSolution.TOPTWSolution (
    TOPTW problem) [inline]
```

Constructor de la clase `TOPTWSolution`.

Parámetros

|                      |  |
|----------------------|--|
| <code>problem</code> | La instancia del problema <code>TOPTW</code> para la cual se crea esta solución. |
|----------------------|--|

Definición en la línea 36 del archivo `TOPTWSolution.java`.

```
00036
00037     this.problem = problem;
00038     this.availableVehicles = this.problem.getVehicles();
00039     this.predecessors = new int[this.problem.getPOIs()+this.problem.getVehicles()];
00040     this.successors = new int[this.problem.getPOIs()+this.problem.getVehicles()];
00041     this.waitingTime = new double[this.problem.getPOIs()];
00042     this.positionInRoute = new int[this.problem.getPOIs()];
00043     Arrays.fill(this.predecessors, TOPTWSolution.NO_INITIALIZED);
00044     Arrays.fill(this.successors, TOPTWSolution.NO_INITIALIZED);
00045     Arrays.fill(this.waitingTime, TOPTWSolution.NO_INITIALIZED);
00046     Arrays.fill(this.positionInRoute, TOPTWSolution.NO_INITIALIZED);
00047     this.routes = new int[this.problem.getVehicles()];
00048     this.objectiveFunctionValue = TOPTWEvaluator.NO_EVALUATED;
00049 }
```

Gráfico de llamadas de esta función:

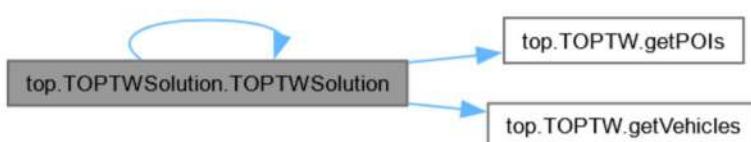


Gráfico de llamadas a esta función:



### 7.12.3. Documentación de funciones miembro

#### 7.12.3.1. addRoute()

```
int top.TOPTWSolution.addRoute () [inline]
```

Añade una nueva ruta a la solución.

Asigna un nuevo depósito a una ruta vacía, decrementa el número de vehículos disponibles y establece la estructura básica de la nueva ruta (depósito apunta a sí mismo).

Devuelve

El índice del nuevo nodo depósito creado.

Definición en la línea 371 del archivo [TOPTWSolution.java](#).

```

00371         {
00372             int depot = this.problem.getPOIs();
00373             depot++;
00374             int routePos = 1;
00375             for(int i = 0; i < this.routes.length; i++) {
00376                 if(this.routes[i] != -1 && this.routes[i] != 0) {
00377                     depot = this.routes[i];
00378                     depot++;
00379                     routePos = i+1;
00380                 }
00381             }
00382             this.routes[routePos] = depot;
00383             this.availableVehicles--;
00384             this.predecessors[depot] = depot;
00385             this.successors[depot] = depot;
00386             this.problem.addNodeDepot();
00387             return depot;
00388         }
  
```

Gráfico de llamadas de esta función:

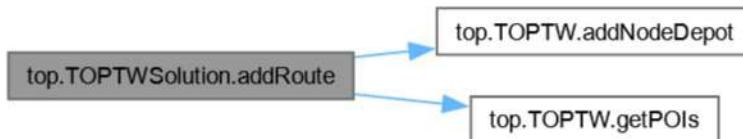


Gráfico de llamadas a esta función:



### 7.12.3.2. equals() [1/2]

```
boolean top.TOPTWSolution.equals (
    Object o) [inline]
```

Compara si dos soluciones son iguales basándose en sus arrays de predecesores.

#### Parámetros

|                                   |                                     |
|-----------------------------------|-------------------------------------|
| <input type="checkbox"/> <i>o</i> | El otro objeto con el que comparar. |
|-----------------------------------|-------------------------------------|

#### Devuelve

`true` si las soluciones son idénticas, `false` en caso contrario.

Definición en la línea 102 del archivo [TOPTWSolution.java](#).

```
00102         {
00103             if (this == o) {
00104                 return true;
00105             }
00106             if (o == null || getClass() != o.getClass()) {
00107                 return false;
00108             }
00109             TOPTWSolution that = (TOPTWSolution) o;
00110             return Arrays.equals(predecessors, that.predecessors);
00111         }
```

Gráfico de llamadas de esta función:



### 7.12.3.3. equals() [2/2]

```
boolean top.TOPTWSolution.equals (
    TOPTWSolution otherSolution) [inline]
```

Compara si dos soluciones son iguales basándose en sus arrays de predecesores.

#### Parámetros

|   |                                       |
|---|---------------------------------------|
| <input type="checkbox"/> <i>otherSolution</i> | La otra solución con la que comparar. |
|---|---------------------------------------|

Devuelve

true si las soluciones son idénticas, false en caso contrario.

Definición en la línea 87 del archivo [TOPTWSolution.java](#).

```
00087     {
00088         for (int i = 0; i < this.predecessors.length; i++) {
00089             if (this.predecessors[i] != otherSolution.predecessors[i]) {
00090                 return false;
00091             }
00092         }
00093     }
00094 }
```

Gráfico de llamadas de esta función:



#### 7.12.3.4. evaluateFitness()

`double top.TOPTWSolution.evaluateFitness () [inline]`

Evaluá la función objetivo de la solución (puntuación total).

Suma las puntuaciones de todos los nodos visitados en todas las rutas.

Devuelve

La puntuación total de la solución.

Definición en la línea 348 del archivo [TOPTWSolution.java](#).

```
00348     {
00349         double objectiveFunction = 0.0;
00350         double objectiveFunctionPerRoute = 0.0;
00351         for(int k = 0; k < this.getCreatedRoutes(); k++) {
00352             int depot = this.getIndexRoute(k);
00353             int pre=depot, suc = -1;
00354             do {
00355                 suc = this.getSuccessor(pre);
00356                 objectiveFunctionPerRoute = objectiveFunctionPerRoute + this.problem.getScore(suc);
00357                 pre = suc;
00358             } while((suc != depot));
00359             objectiveFunction = objectiveFunction + objectiveFunctionPerRoute;
00360             objectiveFunctionPerRoute = 0.0;
00361         }
00362     }
00363 }
```

Gráfico de llamadas de esta función:

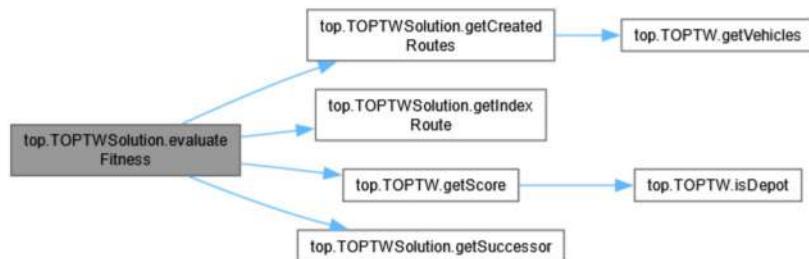
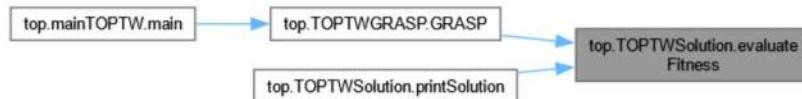


Gráfico de llamadas a esta función:



#### 7.12.3.5. `getAvailableVehicles()`

```
int top.TOPTWSolution.getAvailableVehicles () [inline]
```

Obtiene el número de vehículos que aún no han sido asignados a una ruta.

**Devuelve**

El número de vehículos disponibles.

Definición en la línea 122 del archivo [TOPTWSolution.java](#).

```
00122      {
00123          return this.availableVehicles;
00124      }
```

#### 7.12.3.6. `getCreatedRoutes()`

```
int top.TOPTWSolution.getCreatedRoutes () [inline]
```

Obtiene el número de rutas que han sido creadas en la solución.

**Devuelve**

El número de rutas activas.

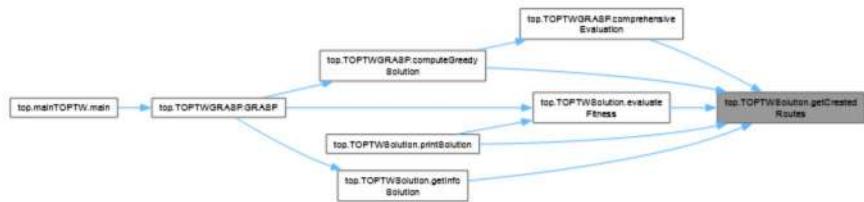
Definición en la línea 130 del archivo [TOPTWSolution.java](#).

```
00130      {
00131          return this.problem.getVehicles() - this.availableVehicles;
00132      }
```

Gráfico de llamadas de esta función:



Gráfico de llamadas a esta función:



### 7.12.3.7. `getDistance()`

```
double top.TOPTWSolution.getDistance (
    int x,
    int y) [inline]
```

Obtiene la distancia entre dos nodos consultando el problema asociado.

Parámetros

|                |                          |
|----------------|--------------------------|
| <code>x</code> | Índice del primer nodo.  |
| <code>y</code> | Índice del segundo nodo. |

Devuelve

La distancia entre `x` e `y`.

Definición en la línea 140 del archivo `TOPTWSolution.java`.

```
00140
00141     return this.problem.getDistance(x, y);
00142 }
```

Gráfico de llamadas de esta función:



Gráfico de llamadas a esta función:



### 7.12.3.8. getIndexRoute()

```
int top.TOPTWSolution.getIndexRoute (
    int index) [inline]
```

Obtiene el índice del nodo que actúa como depósito para una ruta específica.

#### Parámetros

|                          |              |  |
|--------------------------|--------------|--|
| <input type="checkbox"/> | <i>index</i> | El índice de la ruta (de 0 a <a href="#">getCreatedRoutes () -1</a> ). |
|--------------------------|--------------|--|

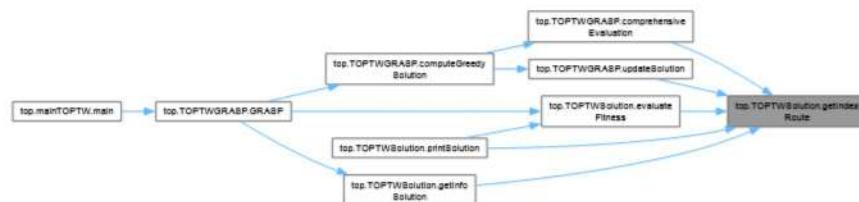
#### Devuelve

El índice del nodo depósito.

Definición en la línea 216 del archivo [TOPTWSolution.java](#).

```
00216
00217     return this.routes[index];
00218 }
```

Gráfico de llamadas a esta función:



### 7.12.3.9. getInfoSolution()

```
String top.TOPTWSolution.getInfoSolution () [inline]
```

Genera una cadena con información detallada de la solución.

Recorre cada ruta, calcula los tiempos de llegada y salida, y verifica la factibilidad de la solución con respecto a las ventanas de tiempo y el tiempo máximo por ruta.

#### Devuelve

Una cadena formateada con el resumen de la solución y los detalles de cada ruta.

Definición en la línea 279 del archivo [TOPTWSolution.java](#).

```
00279
00280     final int COLUMN_WIDTH = 15;
00281     String text = "\n" + "NODES: " + this.problem.getPOIs() + "\n" + "MAX TIME PER ROUTE: " +
    this.problem.getMaxTimePerRoute() + "\n" + "MAX NUMBER OF ROUTES: " + this.problem.getMaxRoutes() +
    "\n";
00282     String textSolution = "\n" + "SOLUTION: " + "\n";
00283     double costTimeSolution = 0.0, fitnessScore = 0.0;
00284     boolean validSolution = true;
00285     for(int k = 0; k < this.getCreatedRoutes(); k++) { // rutas creadas
        String[] strings = new String[]{"\n" + "ROUTE " + k };
```

```

00287         int[] width = new int[strings.length];
00288         Arrays.fill(width, COLUMN_WIDTH);
00289         text += ExpositoUtilities.getFormat(strings, width) + "\n";
00290         strings = new String[1+{"CUST NO.", "X COORD.", "Y. COORD.", "READY TIME", "DUE DATE",
00291             "ARRIVE TIME", "LEAVE TIME", "SERVICE TIME"}];
00292         width = new int[strings.length];
00293         Arrays.fill(width, COLUMN_WIDTH);
00294         text += ExpositoUtilities.getFormat(strings, width) + "\n";
00295         strings = new String[strings.length];
00296         int depot = this.getIndexRoute(k);
00297         int pre=-1, suc=-1;
00298         double costTimeRoute = 0.0, fitnessScoreRoute = 0.0;
00299         pre = depot;
00300         int index = 0;
00301         strings[index++] = "" + pre;
00302         strings[index++] = "" + this.getProblem().getX(pre);
00303         strings[index++] = "" + this.getProblem().getY(pre);
00304         strings[index++] = "" + this.getProblem().getReadyTime(pre);
00305         strings[index++] = "" + 0;
00306         strings[index++] = "" + 0;
00307         strings[index++] = "" + this.getProblem().getServiceTime(pre);
00308         text += ExpositoUtilities.getFormat(strings, width);
00309         text += "\n";
00310     do {           // recorremos la ruta
00311         index = 0;
00312         suc = this.getSuccessor(pre);
00313         textSolution += pre+" - ";
00314         strings[index++] = "" + suc;
00315         strings[index++] = "" + this.getProblem().getX(suc);
00316         strings[index++] = "" + this.getProblem().getY(suc);
00317         strings[index++] = "" + this.getProblem().getReadyTime(suc);
00318         strings[index++] = "" + this.getProblem().getDueTime(suc);
00319         costTimeRoute += this.getDistance(pre, suc);
00320         if(costTimeRoute < (this.getProblem().getDueTime(suc))) {
00321             if(costTimeRoute < this.getProblem().getReadyTime(suc)) {
00322                 costTimeRoute = this.getProblem().getReadyTime(suc);
00323             }
00324             strings[index++] = "" + costTimeRoute;
00325             costTimeRoute += this.getProblem().getServiceTime(suc);
00326             strings[index++] = "" + costTimeRoute;
00327             strings[index++] = "" + this.getProblem().getServiceTime(pre);
00328             if(costTimeRoute > this.getProblem().getMaxTimePerRoute()) { validSolution =
00329                 false; }
00330             fitnessScoreRoute += this.problem.getScore(suc);
00331         } else { validSolution = true; }
00332         pre = suc;
00333         text += ExpositoUtilities.getFormat(strings, width);
00334         text += "\n";
00335     } while(suc != depot);
00336     textSolution += suc+"\n";
00337     costTimeSolution += costTimeRoute;
00338     fitnessScore += fitnessScoreRoute;
00339     textSolution += "FEASIBLE SOLUTION: "+validSolution+"\n"+SCORE: "+fitnessScore+"\n"+TIME
00340     COST: "+costTimeSolution+"\n";
00341     return textSolution+text;
00342 }
```

Gráfico de llamadas de esta función:

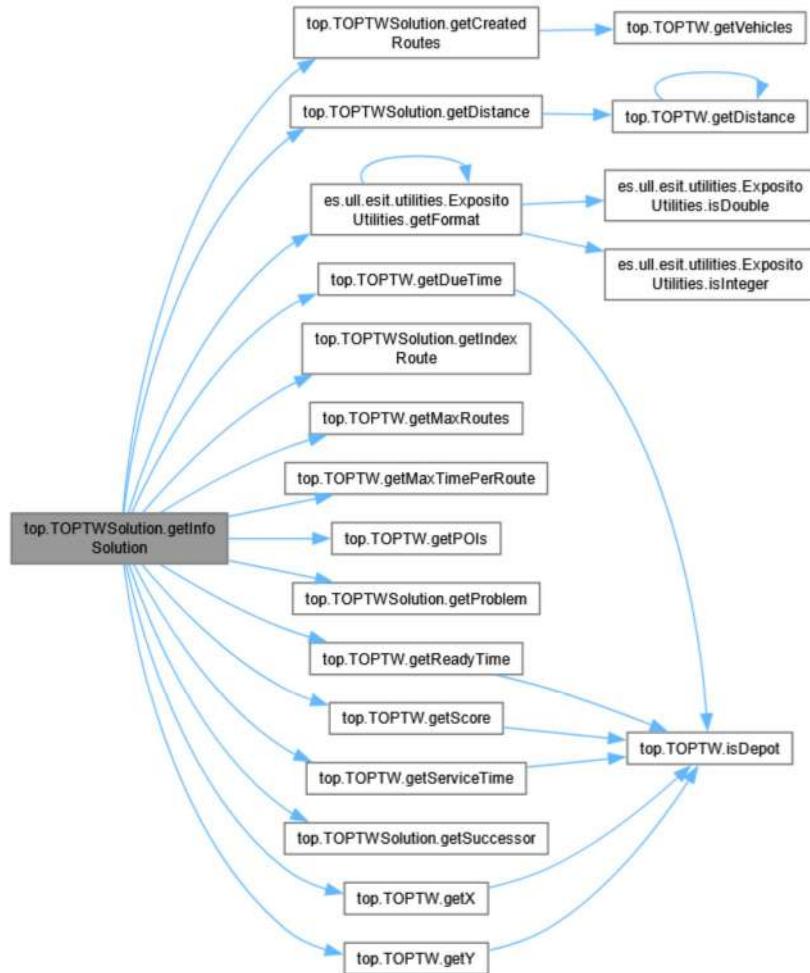


Gráfico de llamadas a esta función:



#### 7.12.3.10. `getObjectiveFunctionValue()`

```
double top.TOPTWSolution.getObjectiveFunctionValue () [inline]
```

Obtiene el valor de la función objetivo (puntuación total) de la solución.

**Devuelve**

El valor de la función objetivo.

Definición en la línea 181 del archivo [TOPTWSolution.java](#).

```
00181         {
00182             return this.objectiveFunctionValue;
00183 }
```

**7.12.3.11. getPositionInRoute()**

```
int top.TOPTWSolution.getPositionInRoute (
    int customer) [inline]
```

Obtiene la posición de un cliente dentro de su ruta.

**Parámetros**

|                 |                        |
|-----------------|------------------------|
| <i>customer</i> | El índice del cliente. |
|-----------------|------------------------|

**Devuelve**

La posición en la ruta.

Definición en la línea 190 del archivo [TOPTWSolution.java](#).

```
00190         {
00191             return this.positionInRoute[customer];
00192 }
```

**7.12.3.12. getPredecessor()**

```
int top.TOPTWSolution.getPredecessor (
    int customer) [inline]
```

Obtiene el predecesor de un cliente en su ruta.

**Parámetros**

|                 |                        |
|-----------------|------------------------|
| <i>customer</i> | El índice del cliente. |
|-----------------|------------------------|

**Devuelve**

El índice del nodo predecesor.

Definición en la línea 157 del archivo [TOPTWSolution.java](#).

```
00157         {
00158             return this.predecessors[customer];
00159 }
```

### 7.12.3.13. `getPredecessors()`

```
int[] top.TOPTWSolution.getPredecessors () [inline]
```

Obtiene el array completo de predecesores.

Devuelve

El array de predecesores.

Definición en la línea 165 del archivo [TOPTWSolution.java](#).

```
00165
00166     return this.predecessors;
00167 }
```

### 7.12.3.14. `getProblem()`

```
TOPTW top.TOPTWSolution.getProblem () [inline]
```

Obtiene la instancia del problema asociada a esta solución.

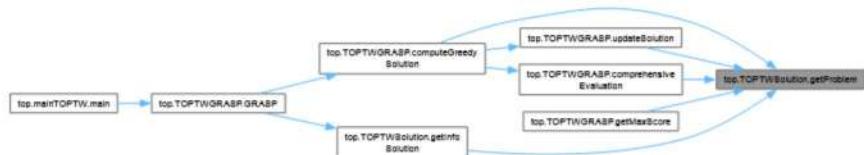
Devuelve

La instancia de [TOPTW](#).

Definición en la línea 173 del archivo [TOPTWSolution.java](#).

```
00173
00174     return this.problem;
00175 }
```

Gráfico de llamadas a esta función:



### 7.12.3.15. `getSuccessor()`

```
int top.TOPTWSolution.getSuccessor (
    int customer) [inline]
```

Obtiene el sucesor de un cliente en su ruta.

Parámetros

|                          |                 |                        |
|--------------------------|-----------------|------------------------|
| <input type="checkbox"/> | <i>customer</i> | El índice del cliente. |
|--------------------------|-----------------|------------------------|

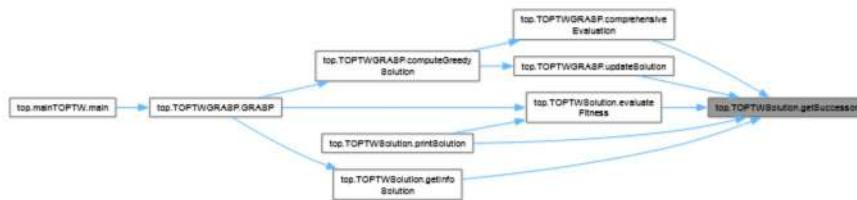
Devuelve

El índice del nodo sucesor.

Definición en la línea 199 del archivo [TOPTWSolution.java](#).

```
00199         return this.successors[customer];
00200     }
00201 }
```

Gráfico de llamadas a esta función:



#### 7.12.3.16. `getSuccessors()`

```
int[] top.TOPTWSolution.getSuccessors () [inline]
```

Obtiene el array completo de sucesores.

Devuelve

El array de sucesores.

Definición en la línea 207 del archivo [TOPTWSolution.java](#).

```
00207
00208     return this.successors;
00209 }
```

#### 7.12.3.17. `getWaitingTime()`

```
double top.TOPTWSolution.getWaitingTime (
    int customer) [inline]
```

Obtiene el tiempo de espera en un nodo cliente.

Parámetros

|                          |                 |                        |
|--------------------------|-----------------|------------------------|
| <input type="checkbox"/> | <i>customer</i> | El índice del cliente. |
|--------------------------|-----------------|------------------------|

Devuelve

El tiempo de espera.

Definición en la línea 225 del archivo [TOPTWSolution.java](#).

```
00225
00226     return this.waitingTime[customer];
00227 }
```

### 7.12.3.18. hashCode()

```
int top.TOPTWSolution.hashCode () [inline]
```

Definición en la línea 114 del archivo [TOPTWSolution.java](#).

```
0014         {
0015             return Arrays.hashCode(predecessors);
0016     }
```

### 7.12.3.19. initSolution()

```
void top.TOPTWSolution.initSolution () [inline]
```

Inicializa o resetea la solución a un estado básico.

Crea una única ruta con el depósito principal (índice 0) y establece los vehículos disponibles.

Definición en la línea 55 del archivo [TOPTWSolution.java](#).

```
00055     {
00056         this.predecessors = new int[this.problem.getPOIs() + this.problem.getVehicles()];
00057         this.successors = new int[this.problem.getPOIs() + this.problem.getVehicles()];
00058         Arrays.fill(this.predecessors, TOPTWSolution.NO_INITIALIZED);
00059         Arrays.fill(this.successors, TOPTWSolution.NO_INITIALIZED);
00060         this.routes = new int[this.problem.getVehicles()];
00061         Arrays.fill(this.routes, TOPTWSolution.NO_INITIALIZED);
00062         this.routes[0] = 0;
00063         this.predecessors[0] = 0;
00064         this.successors[0] = 0;
00065         this.availableVehicles = this.problem.getVehicles() - 1;
00066     }
```

Gráfico de llamadas de esta función:



Gráfico de llamadas a esta función:



### 7.12.3.20. isDepot()

```
boolean top.TOPTWSolution.isDepot (
    int c) [inline]
```

Comprueba si un nodo es un depósito.

**Parámetros**

|                          |   |                                 |
|--------------------------|---|---------------------------------|
| <input type="checkbox"/> | c | El índice del nodo a comprobar. |
|--------------------------|---|---------------------------------|

**Devuelve**

true si el nodo es un depósito de alguna de las rutas, false en caso contrario.

Definición en la línea 73 del archivo [TOPTWSolution.java](#).

```

00073      {
00074          for(int i = 0; i < this.routes.length; i++) {
00075              if(c==this.routes[i]) {
00076                  return true;
00077              }
00078          }
00079      return false;
00080  }
```

Gráfico de llamadas a esta función:

**7.12.3.21. printSolution()**

```
double top.TOPTWSolution.printSolution () [inline]
```

Imprime la solución en la consola y devuelve la puntuación.

Muestra la secuencia de nodos para cada ruta y la puntuación total.

**Devuelve**

La puntuación (fitness) de la solución.

Definición en la línea 395 del archivo [TOPTWSolution.java](#).

```

00395      {
00396          for(int k = 0; k < this.getCreatedRoutes(); k++) {
00397              int depot = this.getIndexRoute(k);
00398              int pre=depot, suc = -1;
00399              do {
00400                  suc = this.getSuccessor(pre);
00401                  System.out.print(pre+" - ");
00402                  pre = suc;
00403              } while((suc != depot));
00404              System.out.println(suc+" ");
00405          }
00406          double fitness = this.evaluateFitness();
00407          System.out.println("SC="+fitness);
00408          return fitness;
00409      }
```

Gráfico de llamadas de esta función:



### 7.12.3.22. **setAvailableVehicles()**

```
void top.TOPTWSolution.setAvailableVehicles (
    int availableVehicles) [inline]
```

Establece el número de vehículos disponibles.

#### Parámetros

|   |   |
|---|---|
| <input type="checkbox"/> <i>availableVehicles</i> | El nuevo número de vehículos disponibles. |
|---|---|

Definición en la línea 148 del archivo [TOPTWSolution.java](#).

```
00148
00149     this.availableVehicles = availableVehicles;
00150 }
```

### 7.12.3.23. **setObjectiveFunctionValue()**

```
void top.TOPTWSolution.setObjectiveFunctionValue (
    double objectiveFunctionValue) [inline]
```

Establece el valor de la función objetivo.

#### Parámetros

|  |                 |
|--|-----------------|
| <input type="checkbox"/> <i>objectiveFunctionValue</i> | El nuevo valor. |
|--|-----------------|

Definición en la línea 233 del archivo [TOPTWSolution.java](#).

```
00233
00234     this.objectiveFunctionValue = objectiveFunctionValue;
00235 }
```

### 7.12.3.24. **setPositionInRoute()**

```
void top.TOPTWSolution.setPositionInRoute (
    int customer,
    int position) [inline]
```

Establece la posición de un cliente en su ruta.

#### Parámetros

|  |                        |
|--|------------------------|
| <input type="checkbox"/> <i>customer</i> | El índice del cliente. |
| <input type="checkbox"/> <i>position</i> | La nueva posición.     |

Definición en la línea 242 del archivo [TOPTWSolution.java](#).

```
00242
00243     this.positionInRoute[customer] = position;
00244 }
```

### 7.12.3.25. **setPredecessor()**

```
void top.TOPTWSolution.setPredecessor (
    int customer,
    int predecessor) [inline]
```

Establece el predecesor de un nodo.

## Parámetros

|                   |                                 |
|-------------------|---------------------------------|
| <i>customer</i>   | El índice del nodo.             |
| <i>predecesor</i> | El índice del nuevo predecesor. |

Definición en la línea 251 del archivo [TOPTWSolution.java](#).

```
00251
00252     this.predecessors[customer] = predecessor;
00253 }
```

Gráfico de llamadas a esta función:



### 7.12.3.26. setSuccessor()

```
void top.TOPTWSolution.setSuccessor (
    int customer,
    int successor) [inline]
```

Establece el sucesor de un nodo.

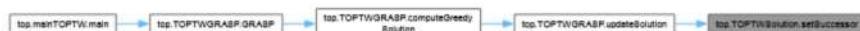
## Parámetros

|                 |                              |
|-----------------|------------------------------|
| <i>customer</i> | El índice del nodo.          |
| <i>sucesor</i>  | El índice del nuevo sucesor. |

Definición en la línea 260 del archivo [TOPTWSolution.java](#).

```
00260
00261     this.successors[customer] = sucessor;
00262 }
```

Gráfico de llamadas a esta función:



### 7.12.3.27. setWaitingTime()

```
void top.TOPTWSolution.setWaitingTime (
    int customer,
    int waitingTime) [inline]
```

Establece el tiempo de espera en un nodo.

### Parámetros

|  |                    |                            |
|--|--------------------|----------------------------|
|  | <i>customer</i>    | El índice del nodo.        |
|  | <i>waitingTime</i> | El nuevo tiempo de espera. |

Definición en la línea 269 del archivo [TOPTWSolution.java](#).

```
00269
00270     this.waitingTime[customer] = waitingTime;
00271 }
```

## 7.12.4. Documentación de datos miembro

### 7.12.4.1. availableVehicles

```
int top.TOPTWSolution.availableVehicles [private]
```

Definición en la línea 29 del archivo [TOPTWSolution.java](#).

### 7.12.4.2. NO\_INITIALIZED

```
final int top.TOPTWSolution.NO_INITIALIZED = -1 [static]
```

Constante para indicar que un valor no ha sido inicializado.

Definición en la línea 21 del archivo [TOPTWSolution.java](#).

### 7.12.4.3. objectiveFunctionValue

```
double top.TOPTWSolution.objectiveFunctionValue [private]
```

Definición en la línea 30 del archivo [TOPTWSolution.java](#).

### 7.12.4.4. positionInRoute

```
int [] top.TOPTWSolution.positionInRoute [private]
```

Definición en la línea 26 del archivo [TOPTWSolution.java](#).

### 7.12.4.5. predecessors

```
int [] top.TOPTWSolution.predecessors [private]
```

Definición en la línea 23 del archivo [TOPTWSolution.java](#).

#### 7.12.4.6. problem

```
TOPTW top.TOPTWSolution.problem [private]
```

Definición en la línea 22 del archivo [TOPTWSolution.java](#).

#### 7.12.4.7. routes

```
int [] top.TOPTWSolution.routes [private]
```

Definición en la línea 28 del archivo [TOPTWSolution.java](#).

#### 7.12.4.8. successors

```
int [] top.TOPTWSolution.successors [private]
```

Definición en la línea 24 del archivo [TOPTWSolution.java](#).

#### 7.12.4.9. waitingTime

```
double [] top.TOPTWSolution.waitingTime [private]
```

Definición en la línea 25 del archivo [TOPTWSolution.java](#).

La documentación de esta clase está generada del siguiente archivo:

- C:/Users/Kefle/Desktop/ExpositoTOP/src/top/[TOPTWSolution.java](#)



## Capítulo 8

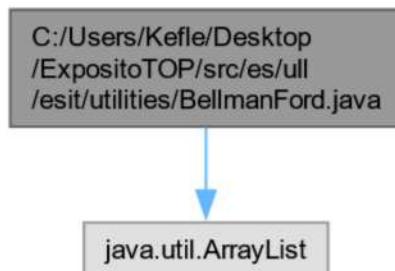
# Documentación de archivos

### 8.1. Referencia del archivo C:/Users/Kefle/Desktop/ExpositoTOP/src/es/ull/esit/utilities/BellmanFord.java

Contiene la implementación del algoritmo de Bellman-Ford para encontrar el camino más corto en un grafo con pesos.

```
import java.util.ArrayList;
```

Gráfico de dependencias incluidas en BellmanFord.java:



#### Clases

- class [es.ull.esit.utilities.BellmanFord](#)  
*Implementa el algoritmo de Bellman-Ford.*

#### Paquetes

- package [es.ull.esit.utilities](#)

### 8.1.1. Descripción detallada

Contiene la implementación del algoritmo de Bellman-Ford para encontrar el camino más corto en un grafo con pesos.

Definición en el archivo [BellmanFord.java](#).

## 8.2. BellmanFord.java

[Ir a la documentación de este archivo.](#)

```

00001
00005 package es.ull.esit.utilities;
00006
00007 import java.util.ArrayList;
00008
00015 public class BellmanFord {
00016
00020     private static final int INFINITY = 999999;
00024     private final int[][] distanceMatrix;
00028     private ArrayList<Integer> edges1 = null;
00032     private ArrayList<Integer> edges2 = null;
00036     private final int nodes;
00040     private final ArrayList<Integer> path;
00044     private int[] distances = null;
00048     private int value;
00049
00056     public BellmanFord(int[][] distanceMatrix, int nodes, ArrayList<Integer> path) {
00057         this.distanceMatrix = distanceMatrix;
00058         this.nodes = nodes;
00059         this.path = path;
00060         this.calculateEdges();
00061         this.value = BellmanFord.INFINITY;
00062     }
00063
00069     private void calculateEdges() {
00070         this.edges1 = new ArrayList<>();
00071         this.edges2 = new ArrayList<>();
00072         for (int i = 0; i < this.nodes; i++) {
00073             for (int j = 0; j < this.nodes; j++) {
00074                 if (this.distanceMatrix[i][j] != Integer.MAX_VALUE) {
00075                     this.edges1.add(i);
00076                     this.edges2.add(j);
00077                 }
00078             }
00079         }
00080     }
00081
00086     public int[] getDistances() {
00087         return this.distances;
00088     }
00089
00094     public int getValue() {
00095         return this.value;
00096     }
00097
00104     public void solve() {
00105         int numEdges = this.edges1.size();
00106         int[] predecessor = new int[this.nodes];
00107         this.distances = new int[this.nodes];
00108         for (int i = 0; i < this.nodes; i++) {
00109             this.distances[i] = BellmanFord.INFINITY;
00110             predecessor[i] = -1;
00111         }
00112         this.distances[0] = 0;
00113         for (int i = 0; i < (this.nodes - 1); i++) {
00114             for (int j = 0; j < numEdges; j++) {
00115                 int u = this.edges1.get(j);
00116                 int v = this.edges2.get(j);
00117                 if (this.distances[v] > this.distances[u] + this.distanceMatrix[u][v]) {
00118                     this.distances[v] = this.distances[u] + this.distanceMatrix[u][v];
00119                     predecessor[v] = u;
00120                 }
00121             }
00122         }
00123         this.path.add(this.nodes - 1);
00124         int pred = predecessor[this.nodes - 1];
00125         while (pred != -1) {

```

```

00126         this.path.add(pred);
00127         pred = predecessor[pred];
00128     }
00129     this.value = -this.distances[this.nodes - 1];
00130 }
00131 }
```

### 8.3. Referencia del archivo C:/Users/Kefle/Desktop/ExpositoTOP/src/es/ull/esit/utilities/ExpositoUtilities.java

Contiene una colección de métodos de utilidad estáticos para diversas tareas.

```

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.text.DecimalFormat;
import java.text.DecimalFormatSymbols;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashSet;
import java.util.Iterator;
import java.util.logging.Level;
import java.util.logging.Logger;
```

Gráfico de dependencias incluidas en ExpositoUtilities.java:



#### Clases

- class [es.ull.esit.utilities.ExpositoUtilities](#)

*Proporciona un conjunto de herramientas estáticas de propósito general.*

#### Paquetes

- package [es.ull.esit.utilities](#)

#### 8.3.1. Descripción detallada

Contiene una colección de métodos de utilidad estáticos para diversas tareas.

Definición en el archivo [ExpositoUtilities.java](#).

## 8.4. ExpositoUtilities.java

[Ir a la documentación de este archivo.](#)

```

00001
00005 package es.ull.esit.utilities;
00006
00007 import java.io.BufferedReader;
00008 import java.io.BufferedWriter;
00009 import java.io.FileReader;
00010 import java.io.FileWriter;
00011 import java.io.IOException;
00012 import java.text.DecimalFormat;
00013 import java.text.DecimalFormatSymbols;
00014 import java.util.ArrayList;
00015 import java.util.Arrays;
00016 import java.util.HashSet;
00017 import java.util.Iterator;
00018 import java.util.logging.Level;
00019 import java.util.logging.Logger;
00020
00027 public class ExpositoUtilities {
00028
00032     public static final int DEFAULT_COLUMN_WIDTH = 10;
00036     public static final int ALIGNMENT_LEFT = 1;
00040     public static final int ALIGNMENT_RIGHT = 2;
00041
00048     private static int getFirstAppearance(int[] vector, int element) {
00049         for (int i = 0; i < vector.length; i++) {
00050             if (vector[i] == element) {
00051                 return i;
00052             }
00053         }
00054         return -1;
00055     }
00056
00061     public static void printFile(String file) {
00062         try (BufferedReader reader = new BufferedReader(new FileReader(file))) {
00063             String line = reader.readLine();
00064             while (line != null) {
00065                 System.out.println(line);
00066                 line = reader.readLine();
00067             }
00068         } catch (IOException ex) {
00069             Logger.getLogger(ExpositoUtilities.class.getName()).log(Level.SEVERE, null, ex);
00070         }
00071     }
00072
00079     public static String simplifyString(String string) {
00080         string = string.replaceAll("\t", " ");
00081         for (int i = 0; i < 50; i++) {
00082             string = string.replaceAll("  ", " ");
00083         }
00084         string = string.trim();
00085         return string;
00086     }
00087
00094     public static double[][] multiplyMatrices(double a[][][], double b[][][]) {
00095         if (a.length == 0) {
00096             return new double[0][0];
00097         }
00098         if (a[0].length != b.length) {
00099             return null;
00100         }
00101         int n = a[0].length;
00102         int m = a.length;
00103         int p = b[0].length;
00104         double ans[][] = new double[m][p];
00105         for (int i = 0; i < m; i++) {
00106             for (int j = 0; j < p; j++) {
00107                 for (int k = 0; k < n; k++) {
00108                     ans[i][j] += a[i][k] * b[k][j];
00109                 }
00110             }
00111         }
00112         return ans;
00113     }
00114
00121     public static void writeTextToFile(String file, String text) throws IOException {
00122         try (BufferedWriter writer = new BufferedWriter(new FileWriter(file))) {
00123             writer.write(text);
00124         }
00125     }
00126
00133     public static String getFormat(String string) {
00134         if (!ExpositoUtilities.isInteger(string)) {

```

```
00135         if (ExpositoUtilities.isDouble(string)) {
00136             double value = Double.parseDouble(string);
00137             string = ExpositoUtilities.getFormat(value);
00138         }
00139     }
00140     return string;
00141 }
00142
00143 public static String getFormat(double value) {
00144     DecimalFormat decimalFormatter = new DecimalFormat("0.000");
00145     DecimalFormatSymbols symbols = new DecimalFormatSymbols();
00146     symbols.setDecimalSeparator('.');
00147     decimalFormatter.setDecimalFormatSymbols(symbols);
00148     return decimalFormatter.format(value);
00149 }
00150
00151 public static String getFormat(double value, int zeros) {
00152     String format = "0.";
00153     for (int i = 0; i < zeros; i++) {
00154         format += "0";
00155     }
00156     DecimalFormat decimalFormatter = new DecimalFormat(format);
00157     DecimalFormatSymbols symbols = new DecimalFormatSymbols();
00158     symbols.setDecimalSeparator('.');
00159     decimalFormatter.setDecimalFormatSymbols(symbols);
00160     return decimalFormatter.format(value);
00161 }
00162
00163 public static String getFormat(String string, int width) {
00164     return ExpositoUtilities.getFormat(string, width, ExpositoUtilities.ALIGNMENT_RIGHT);
00165 }
00166
00167 public static String getFormat(String string, int width, int alignment) {
00168     String format = "";
00169     if (alignment == ExpositoUtilities.ALIGNMENT_LEFT) {
00170         format = "%" + width + "s";
00171     } else {
00172         format = "%" + 1 + "$" + width + "s";
00173     }
00174     DecimalFormatSymbols symbols = new DecimalFormatSymbols();
00175     symbols.setDecimalSeparator('.');
00176     String[] data = new String[]{string};
00177     return String.format(format, (Object[]) data);
00178 }
00179
00180 public static String getFormat(ArrayList<String> strings, int width) {
00181     String format = "";
00182     for (int i = 0; i < strings.size(); i++) {
00183         format += "%" + (i + 1) + "$" + width + "s";
00184     }
00185     String[] data = new String[strings.size()];
00186     for (int t = 0; t < strings.size(); t++) {
00187         data[t] = "" + ExpositoUtilities.getFormat(strings.get(t));
00188     }
00189     return String.format(format, (Object[]) data);
00190 }
00191
00192 public static String getFormat(ArrayList<Integer> strings) {
00193     String format = "";
00194     for (int i = 0; i < strings.size(); i++) {
00195         format += "%" + (i + 1) + "$" + DEFAULT_COLUMN_WIDTH + "s";
00196     }
00197     Integer[] data = new Integer[strings.size()];
00198     for (int t = 0; t < strings.size(); t++) {
00199         data[t] = strings.get(t);
00200     }
00201     return String.format(format, (Object[]) data);
00202 }
00203
00204 public static String getFormat(String[] strings, int width) {
00205     int[] alignment = new int[strings.length];
00206     Arrays.fill(alignment, ExpositoUtilities.ALIGNMENT_RIGHT);
00207     int[] widths = new int[strings.length];
00208     Arrays.fill(widths, width);
00209     return ExpositoUtilities.getFormat(strings, widths, alignment);
00210 }
00211
00212 public static String getFormat(String[][] matrixStrings, int width) {
00213     String result = "";
00214     for (int i = 0; i < matrixStrings.length; i++) {
00215         String[] strings = matrixStrings[i];
00216         int[] alignment = new int[strings.length];
00217         Arrays.fill(alignment, ExpositoUtilities.ALIGNMENT_RIGHT);
00218         int[] widths = new int[strings.length];
00219         Arrays.fill(widths, width);
00220         result += ExpositoUtilities.getFormat(strings, widths, alignment);
00221         if (i < (matrixStrings.length - 1)) {
```

```

00269             result += "\n";
00270         }
00271     }
00272     return result;
00273 }
00274
00280     public static String getFormat(String[] strings) {
00281         int[] alignment = new int[strings.length];
00282         Arrays.fill(alignment, ExpositoUtilities.ALIGNMENT_RIGHT);
00283         int[] widths = new int[strings.length];
00284         Arrays.fill(widths, ExpositoUtilities.DEFAULT_COLUMN_WIDTH);
00285         return ExpositoUtilities.getFormat(strings, widths, alignment);
00286     }
00287
00294     public static String getFormat(String[] strings, int[] width) {
00295         int[] alignment = new int[strings.length];
00296         Arrays.fill(alignment, ExpositoUtilities.ALIGNMENT_RIGHT);
00297         return ExpositoUtilities.getFormat(strings, width, alignment);
00298     }
00299
00307     public static String getFormat(String[] strings, int[] width, int[] alignment) {
00308         String format = "";
00309         for (int i = 0; i < strings.length; i++) {
00310             if (alignment[i] == ExpositoUtilities.ALIGNMENT_LEFT) {
00311                 format += "%" + (i + 1) + "$-" + width[i] + "s";
00312             } else {
00313                 format += "%" + (i + 1) + "$" + width[i] + "s";
00314             }
00315         }
00316         String[] data = new String[strings.length];
00317         for (int t = 0; t < strings.length; t++) {
00318             data[t] = "" + ExpositoUtilities.getFormat(strings[t]);
00319         }
00320         return String.format(format, (Object[]) data);
00321     }
00322
00328     public static boolean isInteger(String str) {
00329         try {
00330             Integer.parseInt(str);
00331             return true;
00332         } catch (Exception e) {
00333         }
00334         return false;
00335     }
00336
00342     public static boolean isDouble(String str) {
00343         try {
00344             Double.parseDouble(str);
00345             return true;
00346         } catch (Exception e) {
00347         }
00348         return false;
00349     }
00350
00356     public static boolean isAcyclic(int[][] distanceMatrix) {
00357         int numRealTasks = distanceMatrix.length - 2;
00358         int node = 1;
00359         boolean acyclic = true;
00360         while (acyclic && node <= numRealTasks) {
00361             if (ExpositoUtilities.thereIsPath(distanceMatrix, node)) {
00362                 return false;
00363             }
00364             node++;
00365         }
00366         return true;
00367     }
00368
00375     public static boolean thereIsPath(int[][] distanceMatrix, int node) {
00376         HashSet<Integer> visits = new HashSet<>();
00377         HashSet<Integer> noVisits = new HashSet<>();
00378         for (int i = 0; i < distanceMatrix.length; i++) {
00379             if (i != node) {
00380                 noVisits.add(i);
00381             }
00382         }
00383         visits.add(node);
00384         while (!visits.isEmpty()) {
00385             Iterator<Integer> it = visits.iterator();
00386             int toCheck = it.next();
00387             visits.remove(toCheck);
00388             for (int i = 0; i < distanceMatrix.length; i++) {
00389                 if (toCheck != i && distanceMatrix[toCheck][i] != Integer.MAX_VALUE) {
00390                     if (i == node) {
00391                         return true;
00392                     }
00393                     if (noVisits.contains(i)) {
00394                         noVisits.remove(i);
00395                     }
00396                 }
00397             }
00398         }
00399     }

```

```

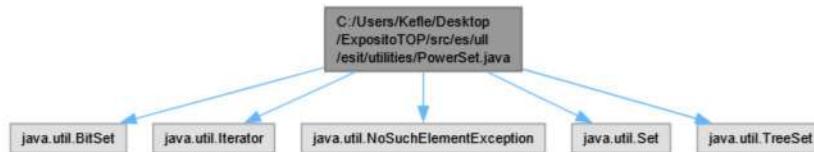
00395             visits.add(i);
00396         }
00397     }
00398   }
00399   return false;
00400 }
00401 }
00402 }
```

## 8.5. Referencia del archivo C:/Users/Kefle/Desktop/ExpositoTOP/src/es/ull/esit/utilities/PowerSet.java

Contiene la implementación de un iterador para generar el conjunto potencia de un conjunto dado.

```

import java.util.BitSet;
import java.util.Iterator;
import java.util.NoSuchElementException;
import java.util.Set;
import java.util.TreeSet;
Gráfico de dependencias incluidas en PowerSet.java:
```



### Clases

- class [es.ull.esit.utilities.PowerSet< E >](#)  
*Implementa un iterador para generar todos los subconjuntos (conjunto potencia) de un conjunto dado.*
- class [es.ull.esit.utilities.PowerSet< E >.PowerSetIterator](#)

### Paquetes

- package [es.ull.esit.utilities](#)

#### 8.5.1. Descripción detallada

Contiene la implementación de un iterador para generar el conjunto potencia de un conjunto dado.

Definición en el archivo [PowerSet.java](#).

## 8.6. PowerSet.java

[Ir a la documentación de este archivo.](#)

```

00001
00005 package es.ull.esit.utilities;
00006
00007 import java.util.BitSet;
00008 import java.util.Iterator;
00009 import java.util.NoSuchElementException;
00010 import java.util.Set;
00011 import java.util.TreeSet;
00012
00019 // Sirve para calcular todos los subconjuntos de un conjunto dado
00020 public class PowerSet<E> implements Iterable<Set<E>> {
00021
00022     private E[] arr = null;
00023
00028     @SuppressWarnings("unchecked")
00029     public PowerSet(Set<E> set) {
00030         this.arr = (E[]) set.toArray();
00031     }
00032
00037     @Override
00038     public Iterator<Set<E>> iterator() {
00039         return new PowerSetIterator();
00040     }
00041
00042     private class PowerSetIterator implements Iterator<Set<E>> {
00043         private BitSet bset;
00044
00045         PowerSetIterator() {
00046             this.bset = new BitSet(PowerSet.this.arr.length + 1);
00047         }
00048
00053         @Override
00054         public boolean hasNext() {
00055             return !this.bset.get(PowerSet.this.arr.length);
00056         }
00057
00063         @Override
00064         public Set<E> next() {
00065             if (!hasNext()) {
00066                 throw new NoSuchElementException();
00067             }
00068             Set<E> returnSet = new TreeSet<E>();
00069             for (int i = 0; i < PowerSet.this.arr.length; i++) {
00070                 if (this.bset.get(i)) {
00071                     returnSet.add(PowerSet.this.arr[i]);
00072                 }
00073             }
00074             // Incrementa el bitset para la siguiente combinación
00075             for (int i = 0; i < this.bset.size(); i++) {
00076                 if (!this.bset.get(i)) {
00077                     this.bset.set(i);
00078                     break;
00079                 } else {
00080                     this.bset.clear(i);
00081                 }
00082             }
00083             return returnSet;
00084         }
00085
00090         @Override
00091         public void remove() {
00092             throw new UnsupportedOperationException("Not Supported!");
00093         }
00094     }
00095 }
```

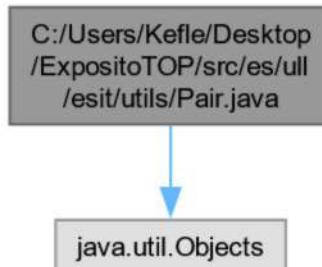
## 8.7. Referencia del archivo

**C:/Users/Kefle/Desktop/ExpositoTOP/src/es/ull/esit/utils/Pair.java**

Contiene la definición de una clase genérica para almacenar un par de objetos.

```
import java.util.Objects;
```

Gráfico de dependencias incluidas en Pair.java:



## Clases

- class [es.ull.esit.utils.Pair< F, S >](#)  
*Una clase genérica que almacena un par de objetos inmutables.*

## Paquetes

- package [es.ull.esit.utils](#)

### 8.7.1. Descripción detallada

Contiene la definición de una clase genérica para almacenar un par de objetos.

Definición en el archivo [Pair.java](#).

## 8.8. Pair.java

[Ir a la documentación de este archivo.](#)

```
00001
00005 package es.ull.esit.utils;
00006 import java.util.Objects;
00007
00016 public class Pair<F, S> {
00020     public final F first;
00024     public final S second;
00025
00031     public Pair(F first, S second) {
00032         this.first = first;
00033         this.second = second;
00034     }
00035
00041     @Override
00042     public boolean equals(Object o) {
00043         if (!(o instanceof Pair)) {
00044             return false;
00045         }
00046         Pair<?, ?> p = (Pair<?, ?>) o;
```

```

00047     return Objects.equals(p.first, first) && Objects.equals(p.second, second);
00048 }
00049
00054     @Override
00055     public int hashCode() {
00056         return (first == null ? 0 : first.hashCode()) ^ (second == null ? 0 : second.hashCode());
00057     }
00058
00068     public static <A, B> Pair <A, B> create(A a, B b) {
00069         return new Pair<A, B>(a, b);
00070     }
00071 }
```

## 8.9. Referencia del archivo

**C:/Users/Kefle/Desktop/ExpositoTOP/src/top/mainTOPTW.java**

Punto de entrada principal para ejecutar el solver GRASP en un conjunto de instancias TOPTW.

### Clases

- class [top.mainTOPTW](#)  
*Clase principal que contiene el método main para ejecutar el experimento.*

### Paquetes

- package [top](#)

#### 8.9.1. Descripción detallada

Punto de entrada principal para ejecutar el solver GRASP en un conjunto de instancias TOPTW.

Definición en el archivo [mainTOPTW.java](#).

## 8.10. mainTOPTW.java

[Ir a la documentación de este archivo.](#)

```

00001
00005 package top;
00006
00013 public class mainTOPTW {
00014
00022     public static void main(String[] args) {
00023
00024         String[] instances = new String[29];
00025
00026         instances[0] = "c101.txt"; instances[3] = "c104.txt"; instances[6] = "c107.txt";
00027         instances[1] = "c102.txt"; instances[4] = "c105.txt"; instances[7] = "c108.txt";
00028         instances[2] = "c103.txt"; instances[5] = "c106.txt"; instances[8] = "c109.txt";
00029
00030         instances[9] = "r101.txt"; instances[12] = "r104.txt"; instances[15] = "r107.txt";
00031         instances[10] = "r102.txt"; instances[13] = "r105.txt"; instances[16] = "r108.txt";
00032         instances[11] = "r103.txt"; instances[14] = "r106.txt"; instances[17] = "r109.txt";
00033         instances[18] = "r110.txt"; instances[19] = "r111.txt"; instances[20] = "r112.txt";
00034
00035         instances[21] = "rc101.txt"; instances[24] = "rc104.txt"; instances[27] = "rc107.txt";
00036         instances[22] = "rc102.txt"; instances[25] = "rc105.txt"; instances[28] = "rc108.txt";
00037         instances[23] = "rc103.txt"; instances[26] = "rc106.txt";
00038
00039         for(int i = 0; i < instances.length; i++) {
```

```

00040     String INSTANCE = "Instances/TOPTW/" + instances[i];
00041     TOPTW problem = TOPTWReader.readProblem(INSTANCE);
00042     TOPTWSolution solution = new TOPTWSolution(problem);
00043     TOPTWGRASP grasp = new TOPTWGRASP(solution);
00044
00045     System.out.println(" --> Instance: " + instances[i]);
00046     grasp.GRASP(10000, 3);
00047     grasp.GRASP(10000, 5);
00048     grasp.GRASP(10000, 7);
00049     System.out.println("");
00050 }
00051 }
00052 }
00053 }
```

## 8.11. Referencia del archivo

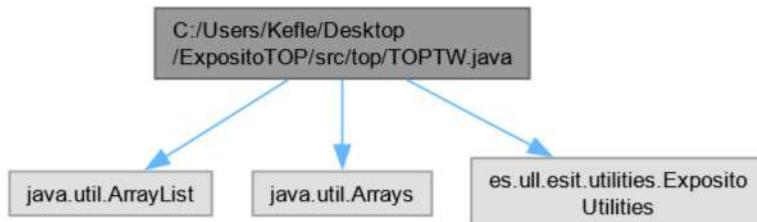
### C:/Users/Kefle/Desktop/ExpositoTOP/src/top/TOPTW.java

Contiene la definición de la clase TOPTW, que modela una instancia del Team Orienteering Problem with Time Windows.

```

import java.util.ArrayList;
import java.util.Arrays;
import es.ull.esit.utilities.ExpositoUtilities;
```

Gráfico de dependencias incluidas en TOPTW.java:



#### Clases

- class [top.TOPTW](#)

*Modela una instancia del problema [TOPTW](#) (Team Orienteering Problem with Time Windows).*

#### Paquetes

- package [top](#)

### 8.11.1. Descripción detallada

Contiene la definición de la clase TOPTW, que modela una instancia del Team Orienteering Problem with Time Windows.

Definición en el archivo [TOPTW.java](#).

## 8.12. TOPTW.java

[Ir a la documentación de este archivo.](#)

```

00001
00005 package top;
00006
00007 import java.util.ArrayList;
00008 import java.util.Arrays;
00009
00010 import es.uLL.esit.utilities.ExpositoUtilities;
00011
00019 public class TOPTW {
00020     private int nodes;
00021     private double[] x;
00022     private double[] y;
00023     private double[] score;
00024     private double[] readyTime;
00025     private double[] dueTime;
00026     private double[] serviceTime;
00027     private int vehicles;
00028     private int depots;
00029     private double maxTimePerRoute;
00030     private double maxRoutes;
00031     private double[][] distanceMatrix;
00032
00038     public TOPTW(int nodes, int routes) {
00039         this.nodes = nodes;
00040         this.depots = 0;
00041         this.x = new double[this.nodes + 1];
00042         this.y = new double[this.nodes + 1];
00043         this.score = new double[this.nodes + 1];
00044         this.readyTime = new double[this.nodes + 1];
00045         this.dueTime = new double[this.nodes + 1];
00046         this.serviceTime = new double[this.nodes + 1];
00047         this.distanceMatrix = new double[this.nodes + 1][this.nodes + 1];
00048         for (int i = 0; i < this.nodes + 1; i++) {
00049             for (int j = 0; j < this.nodes + 1; j++) {
00050                 this.distanceMatrix[i][j] = 0.0;
00051             }
00052         }
00053         this.maxRoutes = routes;
00054         this.vehicles = routes;
00055     }
00056
00062     public boolean isDepot(int a) {
00063         if(a > this.nodes) {
00064             return true;
00065         }
00066         return false;
00067     }
00068
00074     public double getDistance(int[] route) {
00075         double distance = 0.0;
00076         for (int i = 0; i < route.length - 1; i++) {
00077             int node1 = route[i];
00078             int node2 = route[i + 1];
00079             distance += this.getDistance(node1, node2);
00080         }
00081         return distance;
00082     }
00083
00089     public double getDistance(ArrayList<Integer> route) {
00090         double distance = 0.0;
00091         for (int i = 0; i < route.size() - 1; i++) {
00092             int node1 = route.get(i);
00093             int node2 = route.get(i + 1);
00094             distance += this.getDistance(node1, node2);
00095         }
00096         return distance;
00097     }
00098
00104     public double getDistance(ArrayList<Integer>[] routes) {
00105         double distance = 0.0;
00106         for (ArrayList<Integer> route : routes) {
00107             distance += this.getDistance(route);
00108         }
00109         return distance;
00110     }
00111
00117     public void calculateDistanceMatrix() {
00118         for (int i = 0; i < this.nodes + 1; i++) {
00119             for (int j = 0; j < this.nodes + 1; j++) {
00120                 if (i != j) {
00121                     double diffXs = this.x[i] - this.x[j];
00122                     double diffYs = this.y[i] - this.y[j];

```

```
00123         this.distanceMatrix[i][j] = Math.sqrt(diffXs * diffXs + diffYs * diffYs);
00124         this.distanceMatrix[j][i] = this.distanceMatrix[i][j];
00125     } else {
00126         this.distanceMatrix[i][j] = 0.0;
00127     }
00128 }
00129 }
00130 }
00131
00132 public double getMaxTimePerRoute() {
00133     return maxTimePerRoute;
00134 }
00135
00136 public void setMaxTimePerRoute(double maxTimePerRoute) {
00137     this.maxTimePerRoute = maxTimePerRoute;
00138 }
00139
00140 public double getMaxRoutes() {
00141     return maxRoutes;
00142 }
00143
00144 public void setMaxRoutes(double maxRoutes) {
00145     this.maxRoutes = maxRoutes;
00146 }
00147
00148 public int getPOIs() {
00149     return this.nodes;
00150 }
00151
00152 public double getDistance(int i, int j) {
00153     if(this.isDepot(i)) { i=0; }
00154     if(this.isDepot(j)) { j=0; }
00155     return this.distanceMatrix[i][j];
00156 }
00157
00158 public double getTime(int i, int j) {
00159     if(this.isDepot(i)) { i=0; }
00160     if(this.isDepot(j)) { j=0; }
00161     return this.distanceMatrix[i][j];
00162 }
00163
00164 public int getNodes() {
00165     return this.nodes;
00166 }
00167
00168 public void setNodes(int nodes) {
00169     this.nodes = nodes;
00170 }
00171
00172 public double getX(int index) {
00173     if(this.isDepot(index)) { index=0; }
00174     return this.x[index];
00175 }
00176
00177 public void setX(int index, double x) {
00178     this.x[index] = x;
00179 }
00180
00181 public double getY(int index) {
00182     if(this.isDepot(index)) { index=0; }
00183     return this.y[index];
00184 }
00185
00186 public void setY(int index, double y) {
00187     this.y[index] = y;
00188 }
00189
00190 public double getScore(int index) {
00191     if(this.isDepot(index)) { index=0; }
00192     return this.score[index];
00193 }
00194
00195 public double[] getScore() {
00196     return this.score;
00197 }
00198
00199 public void setScore(int index, double score) {
00200     this.score[index] = score;
00201 }
00202
00203 public double getReadyTime(int index) {
00204     if(this.isDepot(index)) { index=0; }
00205     return this.readyTime[index];
00206 }
00207
00208 public void setReadyTime(int index, double readyTime) {
00209     this.readyTime[index] = readyTime;
```

```

00295     }
00296
00302     public double getDueTime(int index) {
00303         if(this.isDepot(index)) { index=0; }
00304         return this.dueTime[index];
00305     }
00306
00312     public void setDueTime(int index, double dueTime) {
00313         this.dueTime[index] = dueTime;
00314     }
00315
00321     public double getServiceTime(int index) {
00322         if(this.isDepot(index)) { index=0; }
00323         return this.serviceTime[index];
00324     }
00325
00331     public void setServiceTime(int index, double serviceTime) {
00332         this.serviceTime[index] = serviceTime;
00333     }
00334
00339     public int getVehicles() {
00340         return this.vehicles;
00341     }
00342
00347     @Override
00348     public String toString() {
00349         final int COLUMN_WIDTH = 15;
00350         String text = "Nodes: " + this.nodes + "\n";
00351         String[] strings = new String[]{"CUST NO.", "XCOORD.", "YCOORD.", "SCORE", "READY TIME", "DUE
DATE", "SERVICE TIME"};
00352         int[] width = new int[strings.length];
00353         Arrays.fill(width, COLUMN_WIDTH);
00354         text += ExpositoUtilities.getFormat(strings, width) + "\n";
00355         for (int i = 0; i < this.nodes; i++) {
00356             strings = new String[strings.length];
00357             int index = 0;
00358             //strings[index++] = Integer.toString("" + i);
00359             strings[index++] = Integer.toString(i);
00360             strings[index++] = "" + this.x[i];
00361             strings[index++] = "" + this.y[i];
00362             strings[index++] = "" + this.score[i];
00363             strings[index++] = "" + this.readyTime[i];
00364             strings[index++] = "" + this.dueTime[i];
00365             strings[index++] = "" + this.serviceTime[i];
00366             text += ExpositoUtilities.getFormat(strings, width);
00367             text += "\n";
00368         }
00369         text += "Vehicles: " + this.vehicles + "\n";
00370         strings = new String[]{"VEHICLE", "CAPACITY"};
00371         width = new int[strings.length];
00372         Arrays.fill(width, COLUMN_WIDTH);
00373         text += ExpositoUtilities.getFormat(strings, width) + "\n";
00374         return text;
00375     }
00376
00382     public int addNode() {
00383         this.nodes++;
00384         return this.nodes;
00385     }
00386
00392     public int addNodeDepot() {
00393         this.depots++;
00394         return this.depots;
00395     }
00396 }

```

## 8.13. Referencia del archivo

**C:/Users/Kefle/Desktop/ExpositoTOP/src/top/TOPTWEvaluator.java**

Contiene la clase para evaluar la función objetivo de una solución TOPTW.

### Clases

- class **top.TOPTWEvaluator**

*Evalúa la calidad de una solución para el problema TOPTW.*

## Paquetes

- package top

### 8.13.1. Descripción detallada

Contiene la clase para evaluar la función objetivo de una solución TOPTW.

Definición en el archivo [TOPTWEvaluator.java](#).

## 8.14. TOPTWEvaluator.java

[Ir a la documentación de este archivo.](#)

```

00001
00005 package top;
00006
00014 public class TOPTWEvaluator {
00018     public static double NO_EVALUATED = -1.0;
00019
00026     public void evaluate(TOPTWSolution solution) {
00027         /*CumulativeCVRP problem = solution.getProblem();
00028         double objectiveFunctionValue = 0.0;
00029         for (int i = 0; i < solution.getIndexDepot().size(); i++) {
00030             double cumulative = 0;
00031             int depot = solution.getAnIndexDepot(i);
00032             int actual = depot;
00033             actual = solution.getSuccessor(actual);
00034             cumulative += problem.getDistanceMatrix(0, actual);
00035             objectiveFunctionValue += problem.getDistanceMatrix(0, actual);
00036             System.out.println("Desde " + 0 + " a " + actual + " = " + cumulative);
00037             while (actual != depot) {
00038                 int ant = actual;
00039                 actual = solution.getSuccessor(actual);
00040                 if (actual != depot) {
00041                     cumulative += problem.getDistanceMatrix(ant, actual);
00042                     objectiveFunctionValue += cumulative;
00043                     System.out.println("Desde " + ant + " a " + actual + " = " + cumulative);
00044                 } else {
00045                     cumulative += problem.getDistanceMatrix(ant, 0);
00046                     objectiveFunctionValue += cumulative;
00047                     System.out.println("Desde " + ant + " a " + 0 + " = " + cumulative);
00048                 }
00049             }
00050             System.out.println("");
00051         }
00052         solution.setObjectiveFunctionValue(objectiveFunctionValue);/*
00053     }
00054 }
```

## 8.15. Referencia del archivo

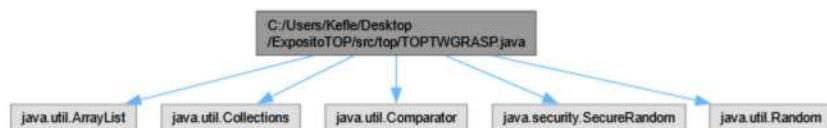
[C:/Users/Kefle/Desktop/ExpositoTOP/src/top/TOPTWGRASP.java](#)

Contiene la implementación de la metaheurística GRASP para el problema TOPTW.

```

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.security.SecureRandom;
```

Gráfico de dependencias incluidas en TOPTWGRASP.java:



## Clases

- class `top.TOPTWGRASP`

Implementa la metaheurística GRASP (Greedy Randomized Adaptive Search Procedure) para resolver el TOPTW.

## **Paquetes**

- package top

#### **8.15.1. Descripción detallada**

Contiene la implementación de la metaheurística GRASP para el problema TOPTW.

Definición en el archivo **TOPTWGRASP.java**.

## 8.16. TOPTWGRASP.java

[Ir a la documentación de este archivo](#)

```
00001
00005 package top;
00006
00007 import java.util.ArrayList;
00008 import java.util.Collections;
00009 import java.util.Comparator;
00010 import java.security.SecureRandom;
00011 import java.util.Random;
00012
00019 public class TOPTWGRASP {
00023     public static double NO_EVALUATED = -1.0;
00024     private static final Random RANDOM = new SecureRandom();
00025
00026     private TOPTWSolution solution;
00027     private int solutionTime;
00028
00033     public TOPTWGRASP(TOPTWSolution sol) {
00034         this.solution = sol;
00035         this.solutionTime = 0;
00036     }
00037
00038     /*procedure GRASP(Max Iterations,Seed)
00039         1 Read Input();
00040         2 for k = 1, . . . , Max Iterations do
00041             3 Solution  $\leftarrow$  Greedy Randomized Construction(Seed);
00042             4 Solution  $\leftarrow$  Local Search(Solution);
00043             5 Update Solution(Solution,Best Solution);
00044         6 end;
00045         7 return Best Solution;
00046     end GRASP*/
```

```

00048     /*procedure Greedy Randomized Construction(Seed)
00049         Solution ← ;
00050         Evaluate the incremental costs of the candidate elements;
00051         while Solution is not a complete solution do
00052             Build the restricted candidate list (RCL);
00053             Select an element s from the RCL at random;
00054             Solution ← Solution ∪ {s};
00055             Reevaluate the incremental costs;
00056         end;
00057         return Solution;
00058     end Greedy Randomized Construction.*/
00059
00067     public void GRASP(int maxIterations, int maxSizeRCL) {
00068         double averageFitness = 0.0;
00069         double bestSolution = 0.0;
00070         for(int i = 0; i < maxIterations; i++) {
00071             this.computeGreedySolution(maxSizeRCL);
00073
00074             // IMPRIMIR SOLUCION
00075             double fitness = this.solution.evaluateFitness();
00076             System.out.println(this.solution.getInfoSolution());
00077             //System.out.println("Press Any Key To Continue..."); 
00078             //new java.util.Scanner(System.in).nextLine();
00079             averageFitness += fitness;
00080             if(bestSolution < fitness) {
00081                 bestSolution = fitness;
00082             }
00083             //double fitness = this.solution.printSolution();
00084
00085             *****
00086             *
00087             * BÚSQUEDA LOCAL
00088             *
00089             */
00090         }
00091         averageFitness = averageFitness/maxIterations;
00092         System.out.println(" --> MEDIA: "+averageFitness);
00093         System.out.println(" --> MEJOR SOLUCION: "+bestSolution);
00094     }
00095
00101     public int aleatorySelectionRCL(int maxTRCL) {
00102         return RANDOM.nextInt(maxTRCL);
00103     }
00104
00111     public int fuzzySelectionBestFDRCL(ArrayList< double[] > rcl) {
00112         int bestPosition = 0;
00113         double bestValue = -1.0;
00114         for(int i = 0; i < rcl.size(); i++) {
00115             if(bestValue < rcl.get(i)[2]) {
00116                 bestValue = rcl.get(i)[2];
00117                 bestPosition = i;
00118             }
00119         }
00120         return bestPosition;
00121     }
00122
00131     public int fuzzySelectionAlphaCutRCL(ArrayList< double[] > rcl, double alpha) {
00132         ArrayList<Integer> candidates = new ArrayList<>();
00133         for(int i = 0; i < rcl.size(); i++) {
00134             if(rcl.get(i)[2] <= alpha) {
00135                 candidates.add(i);
00136             }
00137         }
00138         if(candidates.isEmpty()) {
00139             return aleatorySelectionRCL(rcl.size());
00140         }
00141         else {
00142             int aleatory = RANDOM.nextInt(candidates.size());
00143             return candidates.get(aleatory);
00144         }
00145     }
00146
00153     public void computeGreedySolution(int maxSizeRCL) {
00154         // inicialización
00155         this.solution.initSolution();
00156
00157         // tiempo de salida y score por ruta y cliente
00158         ArrayList<ArrayList<Double>> departureTimesPerClient = new ArrayList<ArrayList<Double>>();
00159         ArrayList<Double> init = new ArrayList<Double>();
00160         for(int z = 0; z <
00161             this.solution.getProblem().getPOIs() + this.solution.getProblem().getVehicles(); z++) { init.add(0.0); }
00162             departureTimesPerClient.add(0, init);
00163
00164             // clientes
00165             ArrayList<Integer> customers = new ArrayList<Integer>();
00166             for(int j = 1; j <= this.solution.getProblem().getPOIs(); j++) { customers.add(j); }

```

```

00166     // Evaluar coste incremental de los elementos candidatos
00167     ArrayList< double[] > candidates = this.comprehensiveEvaluation(customers,
00168         departureTimesPerClient);
00169
00170     Collections.sort(candidates, new Comparator<double[]>() {
00171         public int compare(double[] a, double[] b) {
00172             return Double.compare(a[a.length-2], b[b.length-2]);
00173         }
00174     });
00175
00176     int maxTRCL = maxSizeRCL;
00177     boolean existCandidates = true;
00178
00179     while(!customers.isEmpty() && existCandidates) {
00180         if(!candidates.isEmpty()) {
00181             //Construir lista restringida de candidatos
00182             ArrayList< double[] > rcl = new ArrayList< double[] >();
00183             maxTRCL = maxSizeRCL;
00184             if(maxTRCL > candidates.size()) { maxTRCL = candidates.size(); }
00185             for(int j=0; j < maxTRCL; j++) { rcl.add(candidates.get(j)); }
00186
00187             //Selección aleatoria o fuzzy de candidato de la lista restringida
00188             int posSelected = -1;
00189             int selection = 3;
00190             double alpha = 0.8;
00191             switch (selection) {
00192                 case 1: posSelected = this.aleatorySelectionRCL(maxTRCL); // Selección aleatoria
00193                 break;
00194                 case 2: posSelected = this.fuzzySelectionBestFDRCL(rcl); // Selección fuzzy con
00195                     mejor valor de alpha
00196                     break;
00197                     case 3: posSelected = this.fuzzySelectionAlphaCutRCL(rcl, alpha); // Selección
00198                         fuzzy con alpha corte aleatoria
00199                         break;
00200                     default: posSelected = this.aleatorySelectionRCL(maxTRCL); // Selección aleatoria
00201                         por defecto
00202                         break;
00203             }
00204
00205             double[] candidateSelected = rcl.get(posSelected);
00206             for(int j=0; j < customers.size(); j++) {
00207                 if(customers.get(j)==candidateSelected[0]) {
00208                     customers.remove(j);
00209                 }
00210             }
00211             updateSolution(candidateSelected, departureTimesPerClient);
00212         } else { // No hay candidatos a insertar en la solución, crear otra ruta
00213             if(this.solution.getCreatedRoutes() < this.solution.getProblem().getVehicles()) {
00214                 int newDepot = this.solution.addRoute();
00215                 ArrayList<Double> initNew = new ArrayList<Double>();
00216                 for(int z = 0; z <
this.solution.getProblem().getPOIs()+this.solution.getProblem().getVehicles(); z++)
00217                     departureTimesPerClient.add(initNew);
00218                 }
00219                 else {
00220                     existCandidates = false;
00221                 }
00222             //Reevaluar coste incremental de los elementos candidatos
00223             candidates.clear();
00224             candidates = this.comprehensiveEvaluation(customers, departureTimesPerClient);
00225             Collections.sort(candidates, new Comparator<double[]>() {
00226                 public int compare(double[] a, double[] b) {
00227                     return Double.compare(a[a.length-2], b[b.length-2]);
00228                 }
00229             });
00230         }
00231     }
00232 }
00233
00241     public void updateSolution(double[] candidateSelected, ArrayList< ArrayList< Double > >
00242         departuretimes) {
00243         // Inserción del cliente en la ruta return: cliente, ruta, predecesor, coste
00244         this.solution.setPredecessor((int)candidateSelected[0], (int)candidateSelected[2]);
00245         this.solution.setSuccessor((int)candidateSelected[0],
00246             this.solution.getSuccessor((int)candidateSelected[2]));
00247         this.solution.setSuccessor((int)candidateSelected[2], (int)candidateSelected[0]);
00248         this.solution.setPredecessor(this.solution.getSuccessor((int)candidateSelected[0]),
00249             (int)candidateSelected[0]);
00250
00251         // Actualización de las estructuras de datos y conteo a partir de la posición a insertar
00252         double costInsertionPre =
00253             departureTimes.get((int)candidateSelected[1]).get((int)candidateSelected[2]);

```

```

00250     ArrayList<Double> route = departureTimes.get((int)candidateSelected[1]);
00251     int pre=(int)candidateSelected[2], suc=-1;
00252     int depot = this.solution.getIndexRoute((int)candidateSelected[1]);
00253     do {
00254         suc = this.solution.getSuccessor(pre);
00255         costInsertionPre += this.solution.getDistance(pre, suc);
00256
00257         if(costInsertionPre < this.solution.getProblem().getReadyTime(suc)) {
00258             costInsertionPre = this.solution.getProblem().getReadyTime(suc);
00259         }
00260         costInsertionPre += this.solution.getServiceTime(suc);
00261
00262         if(!this.solution.isDepot(suc))
00263             route.set(suc, costInsertionPre);
00264         pre = suc;
00265     } while((suc != depot));
00266
00267     // Actualiza tiempos
00268     departureTimes.set((int)candidateSelected[1], route);
00269 }
00270
00271 //return: cliente, ruta, predecesor, coste tiempo, score
00272 public ArrayList<double[]> comprehensiveEvaluation(ArrayList<Integer> customers, ArrayList<
ArrayList< Double >> departureTimes) {
00273     ArrayList< double[] > candidatesList = new ArrayList< double[] >();
00274     double[] infoCandidate = new double[5];
00275     boolean validFinalInsertion = true;
00276     infoCandidate[0] = -1;
00277     infoCandidate[1] = -1;
00278     infoCandidate[2] = -1;
00279     infoCandidate[3] = Double.MAX_VALUE;
00280     infoCandidate[4] = -1;
00281
00282     for(int c = 0; c < customers.size(); c++) { // clientes disponibles
00283         for(int k = 0; k < this.solution.getCreatedRoutes(); k++) { // rutas creadas
00284             validFinalInsertion = true;
00285             int depot = this.solution.getIndexRoute(k);
00286             int pre=-1, suc=-1;
00287             double costInsertion = 0;
00288             pre = depot;
00289             int candidate = customers.get(c);
00290             do { // recorremos la ruta
00291                 validFinalInsertion = true;
00292                 suc = this.solution.getSuccessor(pre);
00293                 double timesUntilPre = departureTimes.get(k).get(pre) +
00294                     this.solution.getDistance(pre, candidate);
00295                 if(timesUntilPre < (this.solution.getProblem().getDueTime(candidate))) {
00296                     double costCand = 0;
00297                     if(timesUntilPre < this.solution.getProblem().getReadyTime(candidate)) {
00298                         costCand = this.solution.getProblem().getReadyTime(candidate);
00299                     } else { costCand = timesUntilPre; }
00300                     costCand += this.solution.getServiceTime(candidate);
00301                     if(costCand > this.solution.getProblem().getMaxTimePerRoute()) {
00302                         validFinalInsertion = false; }
00303
00304                     // Comprobar TW desde candidate hasta sucesor
00305                     double timesUntilSuc = costCand + this.solution.getDistance(candidate, suc);
00306                     if(timesUntilSuc < (this.solution.getProblem().getDueTime(suc))) {
00307
00308                         double costSuc = 0;
00309                         if(timesUntilSuc < this.solution.getProblem().getReadyTime(suc)) {
00310                             costSuc = this.solution.getProblem().getReadyTime(suc);
00311                         } else { costSuc = timesUntilSuc; }
00312                         costSuc += this.solution.getServiceTime(suc);
00313                         costInsertion = costSuc;
00314                         if(costSuc > this.solution.getProblem().getMaxTimePerRoute()) {
00315                             validFinalInsertion = false; }
00316
00317                     int pre2=suc, suc2 = -1;
00318                     if(suc != depot)
00319                         do {
00320                             suc2 = this.solution.getSuccessor(pre2);
00321                             double timesUntilSuc2 = costInsertion +
00322                             this.solution.getDistance(pre2, suc2);
00323                             if(timesUntilSuc2 < (this.solution.getProblem().getDueTime(suc2)))
00324                             {
00325                                 if(timesUntilSuc2 <
00326                                     this.solution.getProblem().getReadyTime(suc2)) {
00327                                     costInsertion =
00328                                         this.solution.getProblem().getReadyTime(suc2);
00329                                     } else { costInsertion = timesUntilSuc2; }
00330                                     costInsertion +=
00331                                         this.solution.getServiceTime(suc2);
00332                                     if(costInsertion >
00333                                         this.solution.getProblem().getMaxTimePerRoute()) { validFinalInsertion = false; }
00334                                     } else { validFinalInsertion = false; }
00335                             pre2 = suc2;
00336
00337
00338
00339
00340
00341
00342
00343
00344
00345
00346
00347
00348
00349
00350
00351
00352
00353
00354
00355
00356
00357
00358
00359
00360
00361
00362
00363
00364
00365
00366
00367
00368
00369
00370
00371
00372
00373
00374
00375
00376
00377
00378
00379
00380
00381
00382
00383
00384
00385
00386
00387
00388
00389
00390
00391
00392
00393
00394
00395
00396
00397
00398
00399
00400
00401
00402
00403
00404
00405
00406
00407
00408
00409
00410
00411
00412
00413
00414
00415
00416
00417
00418
00419
00420
00421
00422
00423
00424
00425
00426
00427
00428
00429
00430
00431
00432
00433
00434
00435
00436
00437
00438
00439
00440
00441
00442
00443
00444
00445
00446
00447
00448
00449
00450
00451
00452
00453
00454
00455
00456
00457
00458
00459
00460
00461
00462
00463
00464
00465
00466
00467
00468
00469
00470
00471
00472
00473
00474
00475
00476
00477
00478
00479
00480
00481
00482
00483
00484
00485
00486
00487
00488
00489
00490
00491
00492
00493
00494
00495
00496
00497
00498
00499
00500
00501
00502
00503
00504
00505
00506
00507
00508
00509
00510
00511
00512
00513
00514
00515
00516
00517
00518
00519
00520
00521
00522
00523
00524
00525
00526
00527
00528
00529
00530
00531
00532
00533
00534
00535
00536
00537
00538
00539
00540
00541
00542
00543
00544
00545
00546
00547
00548
00549
00550
00551
00552
00553
00554
00555
00556
00557
00558
00559
00560
00561
00562
00563
00564
00565
00566
00567
00568
00569
00570
00571
00572
00573
00574
00575
00576
00577
00578
00579
00580
00581
00582
00583
00584
00585
00586
00587
00588
00589
00590
00591
00592
00593
00594
00595
00596
00597
00598
00599
00600
00601
00602
00603
00604
00605
00606
00607
00608
00609
00610
00611
00612
00613
00614
00615
00616
00617
00618
00619
00620
00621
00622
00623
00624
00625
00626
00627
00628
00629
00630
00631
00632
00633
00634
00635
00636
00637
00638
00639
00640
00641
00642
00643
00644
00645
00646
00647
00648
00649
00650
00651
00652
00653
00654
00655
00656
00657
00658
00659
00660
00661
00662
00663
00664
00665
00666
00667
00668
00669
00670
00671
00672
00673
00674
00675
00676
00677
00678
00679
00680
00681
00682
00683
00684
00685
00686
00687
00688
00689
00690
00691
00692
00693
00694
00695
00696
00697
00698
00699
00700
00701
00702
00703
00704
00705
00706
00707
00708
00709
00710
00711
00712
00713
00714
00715
00716
00717
00718
00719
00720
00721
00722
00723
00724
00725
00726
00727
00728
00729
00730
00731
00732
00733
00734
00735
00736
00737
00738
00739
00740
00741
00742
00743
00744
00745
00746
00747
00748
00749
00750
00751
00752
00753
00754
00755
00756
00757
00758
00759
00760
00761
00762
00763
00764
00765
00766
00767
00768
00769
00770
00771
00772
00773
00774
00775
00776
00777
00778
00779
00780
00781
00782
00783
00784
00785
00786
00787
00788
00789
00790
00791
00792
00793
00794
00795
00796
00797
00798
00799
00800
00801
00802
00803
00804
00805
00806
00807
00808
00809
00810
00811
00812
00813
00814
00815
00816
00817
00818
00819
00820
00821
00822
00823
00824
00825
00826
00827
00828
00829
00830
00831
00832
00833
00834
00835
00836
00837
00838
00839
00840
00841
00842
00843
00844
00845
00846
00847
00848
00849
00850
00851
00852
00853
00854
00855
00856
00857
00858
00859
00860
00861
00862
00863
00864
00865
00866
00867
00868
00869
00870
00871
00872
00873
00874
00875
00876
00877
00878
00879
00880
00881
00882
00883
00884
00885
00886
00887
00888
00889
00890
00891
00892
00893
00894
00895
00896
00897
00898
00899
00900
00901
00902
00903
00904
00905
00906
00907
00908
00909
00910
00911
00912
00913
00914
00915
00916
00917
00918
00919
00920
00921
00922
00923
00924
00925
00926
00927
00928
00929
00930
00931
00932
00933
00934
00935
00936
00937
00938
00939
00940
00941
00942
00943
00944
00945
00946
00947
00948
00949
00950
00951
00952
00953
00954
00955
00956
00957
00958
00959
00960
00961
00962
00963
00964
00965
00966
00967
00968
00969
00970
00971
00972
00973
00974
00975
00976
00977
00978
00979
00980
00981
00982
00983
00984
00985
00986
00987
00988
00989
00990
00991
00992
00993
00994
00995
00996
00997
00998
00999
01000
01001
01002
01003
01004
01005
01006
01007
01008
01009
01010
01011
01012
01013
01014
01015
01016
01017
01018
01019
01020
01021
01022
01023
01024
01025
01026
01027
01028
01029
01030
01031
01032
01033
01034
01035
01036
01037
01038
01039
01040
01041
01042
01043
01044
01045
01046
01047
01048
01049
01050
01051
01052
01053
01054
01055
01056
01057
01058
01059
01060
01061
01062
01063
01064
01065
01066
01067
01068
01069
01070
01071
01072
01073
01074
01075
01076
01077
01078
01079
01080
01081
01082
01083
01084
01085
01086
01087
01088
01089
01090
01091
01092
01093
01094
01095
01096
01097
01098
01099
01100
01101
01102
01103
01104
01105
01106
01107
01108
01109
01110
01111
01112
01113
01114
01115
01116
01117
01118
01119
01120
01121
01122
01123
01124
01125
01126
01127
01128
01129
01130
01131
01132
01133
01134
01135
01136
01137
01138
01139
01140
01141
01142
01143
01144
01145
01146
01147
01148
01149
01150
01151
01152
01153
01154
01155
01156
01157
01158
01159
01160
01161
01162
01163
01164
01165
01166
01167
01168
01169
01170
01171
01172
01173
01174
01175
01176
01177
01178
01179
01180
01181
01182
01183
01184
01185
01186
01187
01188
01189
01190
01191
01192
01193
01194
01195
01196
01197
01198
01199
01200
01201
01202
01203
01204
01205
01206
01207
01208
01209
01210
01211
01212
01213
01214
01215
01216
01217
01218
01219
01220
01221
01222
01223
01224
01225
01226
01227
01228
01229
01230
01231
01232
01233
01234
01235
01236
01237
01238
01239
01240
01241
01242
01243
01244
01245
01246
01247
01248
01249
01250
01251
01252
01253
01254
01255
01256
01257
01258
01259
01260
01261
01262
01263
01264
01265
01266
01267
01268
01269
01270
01271
01272
01273
01274
01275
01276
01277
01278
01279
01280
01281
01282
01283
01284
01285
01286
01287
01288
01289
01290
01291
01292
01293
01294
01295
01296
01297
01298
01299
01300
01301
01302
01303
01304
01305
01306
01307
01308
01309
01310
01311
01312
01313
01314
01315
01316
01317
01318
01319
01320
01321
01322
01323
01324
01325
01326
01327
01328
01329
01330
01331
01332
01333
01334
01335
01336
01337
01338
01339
01340
01341
01342
01343
01344
01345
01346
01347
01348
01349
01350
01351
01352
01353
01354
01355
01356
01357
01358
01359
01360
01361
01362
01363
01364
01365
01366
01367
01368
01369
01370
01371
01372
01373
01374
01375
01376
01377
01378
01379
01380
01381
01382
01383
01384
01385
01386
01387
01388
01389
01390
01391
01392
01393
01394
01395
01396
01397
01398
01399
01400
01401
01402
01403
01404
01405
01406
01407
01408
01409
01410
01411
01412
01413
01414
01415
01416
01417
01418
01419
01420
01421
01422
01423
01424
01425
01426
01427
01428
01429
01430
01431
01432
01433
01434
01435
01436
01437
01438
01439
01440
01441
01442
01443
01444
01445
01446
01447
01448
01449
01450
01451
01452
01453
01454
01455
01456
01457
01458
01459
01460
01461
01462
01463
01464
01465
01466
01467
01468
01469
01470
01471
01472
01473
01474
01475
01476
01477
01478
01479
01480
01481
01482
01483
01484
01485
01486
01487
01488
01489
01490
01491
01492
01493
01494
01495
01496
01497
01498
01499
01500
01501
01502
01503
01504
01505
01506
01507
01508
01509
01510
01511
01512
01513
01514
01515
01516
01517
01518
01519
01520
01521
01522
01523
01524
01525
01526
01527
01528
01529
01530
01531
01532
01533
01534
01535
01536
01537
01538
01539
01540
01541
01542
01543
01544
01545
01546
01547
01548
01549
01550
01551
01552
01553
01554
01555
01556
01557
01558
01559
01560
01561
01562
01563
01564
01565
01566
01567
01568
01569
01570
01571
01572
01573
01574
01575
01576
01577
01578
01579
01580
01581
01582
01583
01584
01585
01586
01587
01588
01589
01590
01591
01592
01593
01594
01595
01596
01597
01598
01599
01600
01601
01602
01603
01604
01605
01606
01607
01608
01609
01610
01611
01612
01613
01614
01615
01616
01617
01618
01619
01620
01621
01622
01623
01624
01625
01626
01627
01628
01629
01630
01631
01632
01633
01634
01635
01636
01637
01638
01639
01640
01641
01642
01643
01644
01645
01646
01647
01648
01649
01650
01651
01652
01653
01654
01655
01656
01657
01658
01659
01660
01661
01662
01663
01664
01665
01666
01667
01668
01669
01670
01671
01672
01673
01674
01675
01676
01677
01678
01679
01680
01681
01682
01683
01684
01685
01686
01687
01688
01689
01690
01691
01692
01693
01694
01695
01696
01697
01698
01699
01700
01701
01702
01703
01704
01705
01706
01707
01708
01709
01710
01711
01712
01713
01714
01715
01716
01717
01718
01719
01720
01721
01722
01723
01724
01725
01726
01727
01728
01729
01730
01731
01732
01733
01734
01735
01736
01737
01738
01739
01740
01741
01742
01743
01744
01745
01746
01747
01748
01749
01750
01751
01752
01753
01754
01755
01756
01757
01758
01759
01760
01761
01762
01763
01764
01765
01766
01767
01768
01769
01770
01771
01772
01773
01774
01775
01776
01777
01778
01779
01780
01781
01782
01783
01784
01785
01786
01787
01788
01789
01790
01791
01792
01793
01794
01795
01796
01797
01798
01799
01800
01801
01802
01803
01804
01805
01806
01807
01808
01809
01810
01811
01812
01813
01814
01815
01816
01817
01818
01819
01820
01821
01822
01823
01824
01825
01826
01827
01828
01829
01830
01831
01832
01833
01834
01835
01836
01837
01838
01839
01840
01841
01842
01843
01844
01845
01846
01847
01848
01849
01850
01851
01852
01853
01854
01855
01856
01857
01858
01859
01860
01861
01862
01863
01864
01865
01866
01867
01868
01869
01870
01871
01872
01873
01874
01875
01876
01877
01878
01879
01880
01881
01882
01883
01884
01885
01886
01887
01888
01889
01890
01891
01892
01893
01894
01895
01896
01897
01898
01899
01900
01901
01902
01903
01904
01905
01906
01907
01908
01909
01910
01911
01912
01913
01914
01915
01916
01917
01918
01919
01920
01921
01922
01923
01924
01925
01926
01927
01928
01929
01930
01931
01932
01933
01934
01935
01936
01937
01938
01939
01940
01941
01942
01943
01944
01945
01946
01947
01948
01949
01950

```

```

00334                     } while((suc2 != depot) && validFinalInsertion);
00335                 } else { validFinalInsertion = false; }
00336             } else { validFinalInsertion = false; }
00337
00338             if(validFinalInsertion==true) { // cliente, ruta, predecesor, coste
00339                 if(costInsertion < infoCandidate[3]) {
00340                     infoCandidate[0] = candidate; infoCandidate[1] = k; infoCandidate[2] =
00340 pre; infoCandidate[3] = costInsertion; infoCandidate[4] =
00340 this.solution.getProblem().getScore(candidate); // cliente, ruta, predecesor, coste, score
00341                 }
00342             }
00343
00344             pre = suc;
00345             } while(suc != depot);
00346         } //rutas creadas
00347
00348         // almacenamos en la lista de candidatos la mejor posición de inserción para el cliente
00349         if(infoCandidate[0]!=-1 && infoCandidate[1]!=-1 && infoCandidate[2]!=-1 &&
00349 infoCandidate[3] != Double.MAX_VALUE && infoCandidate[4]!=-1) {
00350             double[] infoCandidate2 = new double[5];
00351             infoCandidate2[0] = infoCandidate[0]; infoCandidate2[1] = infoCandidate[1];
00352             infoCandidate2[2] = infoCandidate[2]; infoCandidate2[3] = infoCandidate[3];
00353             infoCandidate2[4] = infoCandidate[4];
00354             candidatesList.add(infoCandidate2);
00355         }
00356         validFinalInsertion = true;
00357         infoCandidate[0] = -1; infoCandidate[1] = -1;
00358         infoCandidate[2] = -1; infoCandidate[3] = Double.MAX_VALUE;
00359         infoCandidate[4] = -1;
00360     } // cliente
00361
00362     return candidatesList;
00363 }
00364
00365 public TOPTWSolution getSolution() {
00366     return solution;
00367 }
00368
00369 public void setSolution(TOPTWSolution solution) {
00370     this.solution = solution;
00371 }
00372
00373 public int getSolutionTime() {
00374     return solutionTime;
00375 }
00376
00377 public void setSolutionTime(int solutionTime) {
00378     this.solutionTime = solutionTime;
00379 }
00380
00381 public double getMaxScore() {
00382     double maxSc = -1.0;
00383     for(int i = 0; i < this.solution.getProblem().getScore().length; i++) {
00384         if(this.solution.getProblem().getScore(i) > maxSc)
00385             maxSc = this.solution.getProblem().getScore(i);
00386     }
00387     return maxSc;
00388 }
00389
00390 }
00391 }
```

## 8.17. Referencia del archivo

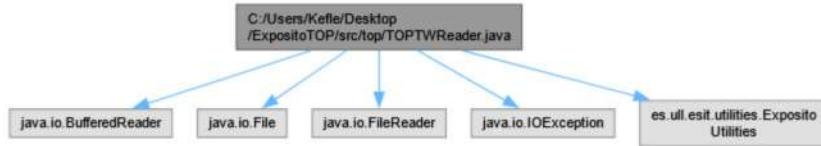
**C:/Users/Kefle/Desktop/ExpositoTOP/src/top/TOPTWReader.java**

Contiene la clase TOPTWReader para leer instancias del problema TOPTW desde archivos.

```

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import es.ull.esit.utilities.ExpositoUtilities;
```

Gráfico de dependencias incluidas en TOPTWReader.java:



## Clases

- class [top.TOPTWReader](#)

*Proporciona funcionalidad para leer una instancia del problema [TOPTW](#) desde un archivo de texto.*

## Paquetes

- package [top](#)

### 8.17.1. Descripción detallada

Contiene la clase TOPTWReader para leer instancias del problema TOPTW desde archivos.

Definición en el archivo [TOPTWReader.java](#).

## 8.18. TOPTWReader.java

[Ir a la documentación de este archivo.](#)

```

00001
00005 package top;
00006
00007 import java.io.BufferedReader;
00008 import java.io.File;
00009 import java.io.FileReader;
00010 import java.io.IOException;
00011
00012 import es.ull.esit.utilities.ExpositoUtilities;
00013
00019 public class TOPTWReader {
00020
00031     public static TOPTW readProblem(String filePath) {
00032         TOPTW problem = null;
00033         BufferedReader reader = null;
00034         try {
00035             File instaceFile = new File(filePath);
00036             reader = new BufferedReader(new FileReader(instaceFile));
00037             String line = reader.readLine();
00038             line = ExpositoUtilities.simplifyString(line);
00039             String[] parts = line.split(" ");
00040             problem = new TOPTW(Integer.parseInt(parts[2]), Integer.parseInt(parts[1]));
00041             line = reader.readLine();
00042             line = null; parts = null;
00043             for (int i = 0; i < problem.getPOIs()+1; i++) {
00044                 line = reader.readLine();
00045                 line = ExpositoUtilities.simplifyString(line);
00046                 parts = line.split(" ");
00047                 problem.setX(i, Double.parseDouble(parts[1]));
00048                 problem.setY(i, Double.parseDouble(parts[2]));
00049                 problem.setServiceTime(i, Double.parseDouble(parts[3]));
  
```

```

00050     problem.setScore(i, Double.parseDouble(parts[4]));
00051     if(i==0) {
00052         problem.setReadyTime(i, Double.parseDouble(parts[7]));
00053         problem.setDueTime(i, Double.parseDouble(parts[8]));
00054     }
00055     else {
00056         problem.setReadyTime(i, Double.parseDouble(parts[8]));
00057         problem.setDueTime(i, Double.parseDouble(parts[9]));
00058     }
00059     line = null; parts = null;
00060 }
00061 problem.calculateDistanceMatrix();
00062 } catch (IOException e) {
00063     System.err.println(e);
00064     System.exit(0);
00065 } finally {
00066     if (reader != null) {
00067         try {
00068             reader.close();
00069         } catch (IOException ex) {
00070             System.err.println(ex);
00071             System.exit(0);
00072         }
00073     }
00074 }
00075 problem.setMaxTimePerRoute(problem.getDueTime(0));
00076 return problem;
00077 }
00078 }
00079 }
```

## 8.19. Referencia del archivo

**C:/Users/Kefle/Desktop/ExpositoTOP/src/top/TOPTWRoute.java**

Contiene la definición de la clase TOPTWRoute, que representa una ruta simple.

### Clases

- class [top.TOPTWRoute](#)

*Modela una ruta simple mediante la identificación de sus nodos predecessor y sucesor.*

### Paquetes

- package [top](#)

#### 8.19.1. Descripción detallada

Contiene la definición de la clase TOPTWRoute, que representa una ruta simple.

Definición en el archivo [TOPTWRoute.java](#).

## 8.20. TOPTWRoute.java

[Ir a la documentación de este archivo.](#)

```
00001
00005 package top;
00006
00012 public class TOPTWRoute {
00016     int predecessor;
00020     int successor;
00024     int id;
00025
00029     TOPTWRoute() {
00030
00031 }
00032
00039     TOPTWRoute(int pre, int succ, int id) {
00040         this.predecessor = pre;
00041         this.successor = succ;
00042         this.id = id;
00043     }
00044
00049     public int getPredecessor() {
00050         return this.predecessor;
00051     }
00052
00057     public int getSuccessor() {
00058         return this.successor;
00059     }
00060
00065     public int getId() {
00066         return this.id;
00067     }
00068
00073     public void setPredecessor(int pre) {
00074         this.predecessor = pre;
00075     }
00076
00081     public void setSuccessor(int suc) {
00082         this.successor = suc;
00083     }
00084
00089     public void setId(int id) {
00090         this.id = id;
00091     }
00092 }
```

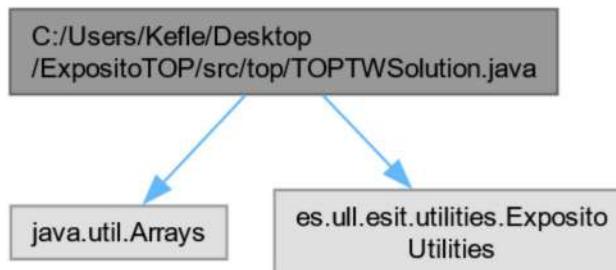
## 8.21. Referencia del archivo

**C:/Users/Kefle/Desktop/ExpositoTOP/src/top/TOPTWSolution.java**

Contiene la definición de la clase TOPTWSolution, que representa una solución para el problema TOPTW.

```
import java.util.Arrays;
import es.ull.esit.utilities.ExpositoUtilities;
```

Gráfico de dependencias incluidas en TOPTWSolution.java:



## Clases

- class [top.TOPTWSolution](#)  
*Representa una solución a una instancia del problema TOPTW.*

## Paquetes

- package [top](#)

### 8.21.1. Descripción detallada

Contiene la definición de la clase TOPTWSolution, que representa una solución para el problema TOPTW.

Definición en el archivo [TOPTWSolution.java](#).

## 8.22. TOPTWSolution.java

[Ir a la documentación de este archivo.](#)

```

00001
00005 package top;
00006
00007 import java.util.Arrays;
00008
00009 import es.ull.esit.utilities.ExpositoUtilities;
00010
00017 public class TOPTWSolution {
00021     public static final int NO_INITIALIZED = -1;
00022     private TOPTW problem;
00023     private int[] predecessors;
00024     private int[] successors;
00025     private double[] waitingTime;
00026     private int[] positionInRoute;
00027
00028     private int[] routes;
00029     private int availableVehicles;
00030     private double objectiveFunctionValue;
00031
00036     public TOPTWSolution(TOPTW problem) {
00037         this.problem = problem;
  
```

```

00038     this.availableVehicles = this.problem.getVehicles();
00039     this.predecessors = new int[this.problem.getPOIs() + this.problem.getVehicles()];
00040     this.successors = new int[this.problem.getPOIs() + this.problem.getVehicles()];
00041     this.waitingTime = new double[this.problem.getPOIs()];
00042     this.positionInRoute = new int[this.problem.getPOIs()];
00043     Arrays.fill(this.predecessors, TOPTWSolution.NO_INITIALIZED);
00044     Arrays.fill(this.successors, TOPTWSolution.NO_INITIALIZED);
00045     Arrays.fill(this.waitingTime, TOPTWSolution.NO_INITIALIZED);
00046     Arrays.fill(this.positionInRoute, TOPTWSolution.NO_INITIALIZED);
00047     this.routes = new int[this.problem.getVehicles()];
00048     this.objectiveFunctionValue = TOPTWEvaluator.NO_EVALUATED;
00049 }
00050
00055 public void initSolution() {
00056     this.predecessors = new int[this.problem.getPOIs() + this.problem.getVehicles()];
00057     this.successors = new int[this.problem.getPOIs() + this.problem.getVehicles()];
00058     Arrays.fill(this.predecessors, TOPTWSolution.NO_INITIALIZED);
00059     Arrays.fill(this.successors, TOPTWSolution.NO_INITIALIZED);
00060     this.routes = new int[this.problem.getVehicles()];
00061     Arrays.fill(this.routes, TOPTWSolution.NO_INITIALIZED);
00062     this.routes[0] = 0;
00063     this.predecessors[0] = 0;
00064     this.successors[0] = 0;
00065     this.availableVehicles = this.problem.getVehicles() - 1;
00066 }
00067
00073 public boolean isDepot(int c) {
00074     for(int i = 0; i < this.routes.length; i++) {
00075         if(c==this.routes[i]) {
00076             return true;
00077         }
00078     }
00079     return false;
00080 }
00081
00087 public boolean equals(TOPTWSolution otherSolution) {
00088     for (int i = 0; i < this.predecessors.length; i++) {
00089         if (this.predecessors[i] != otherSolution.predecessors[i]) {
00090             return false;
00091         }
00092     }
00093     return true;
00094 }
00095
00101 @Override
00102 public boolean equals(Object o) {
00103     if (this == o) {
00104         return true;
00105     }
00106     if (o == null || getClass() != o.getClass()) {
00107         return false;
00108     }
00109     TOPTWSolution that = (TOPTWSolution) o;
00110     return Arrays.equals(predecessors, that.predecessors);
00111 }
00112
00113 @Override
00114 public int hashCode() {
00115     return Arrays.hashCode(predecessors);
00116 }
00117
00122 public int getAvailableVehicles() {
00123     return this.availableVehicles;
00124 }
00125
00130 public int getCreatedRoutes() {
00131     return this.problem.getVehicles() - this.availableVehicles;
00132 }
00133
00140 public double getDistance(int x, int y) {
00141     return this.problem.getDistance(x, y);
00142 }
00143
00148 public void setAvailableVehicles(int availableVehicles) {
00149     this.availableVehicles = availableVehicles;
00150 }
00151
00157 public int getPredecessor(int customer) {
00158     return this.predecessors[customer];
00159 }
00160
00165 public int[] getPredecessors() {
00166     return this.predecessors;
00167 }
00168
00173 public TOPTW getProblem() {
00174     return this.problem;
}

```

```

00175     }
00176
00181     public double getObjectiveFunctionValue() {
00182         return this.objectiveFunctionValue;
00183     }
00184
00190     public int getPositionInRoute(int customer) {
00191         return this.positionInRoute[customer];
00192     }
00193
00199     public int getSuccessor(int customer) {
00200         return this.successors[customer];
00201     }
00202
00207     public int[] getSuccessors() {
00208         return this.successors;
00209     }
00210
00216     public int getIndexRoute(int index) {
00217         return this.routes[index];
00218     }
00219
00225     public double getWaitingTime(int customer) {
00226         return this.waitingTime[customer];
00227     }
00228
00233     public void setObjectiveFunctionValue(double objectiveFunctionValue) {
00234         this.objectiveFunctionValue = objectiveFunctionValue;
00235     }
00236
00242     public void setPositionInRoute(int customer, int position) {
00243         this.positionInRoute[customer] = position;
00244     }
00245
00251     public void setPredecessor(int customer, int predecessor) {
00252         this.predecessors[customer] = predecessor;
00253     }
00254
00260     public void setSuccessor(int customer, int successor) {
00261         this.successors[customer] = successor;
00262     }
00263
00269     public void setWaitingTime(int customer, int waitingTime) {
00270         this.waitingTime[customer] = waitingTime;
00271     }
00272
00279     public String getInfoSolution() {
00280         final int COLUMN_WIDTH = 15;
00281         String text = "\n" + "NODES: " + this.problem.getPOIs() + "\n" + "MAX TIME PER ROUTE: " +
00282             this.problem.getMaxTimePerRoute() + "\n" + "MAX NUMBER OF ROUTES: " + this.problem.getMaxRoutes() +
00283             "\n";
00284         String textSolution = "\n" + "SOLUTION: " + "\n";
00285         double costTimeSolution = 0.0, fitnessScore = 0.0;
00286         boolean validSolution = true;
00287         for(int k = 0; k < this.getCreatedRoutes(); k++) { // rutas creadas
00288             String[] strings = new String[]{"\n" + "ROUTE " + k };
00289             int[] width = new int[strings.length];
00290             Arrays.fill(width, COLUMN_WIDTH);
00291             text += ExpositoUtilities.getFormat(strings, width) + "\n";
00292             strings = new String[]{"CUST NO.", "X COORD.", "Y. COORD.", "READY TIME", "DUE DATE",
00293             "ARRIVE TIME", "LEAVE TIME", "SERVICE TIME"};
00294             width = new int[strings.length];
00295             Arrays.fill(width, COLUMN_WIDTH);
00296             text += ExpositoUtilities.getFormat(strings, width) + "\n";
00297             strings = new String[strings.length];
00298             int depot = this.getIndexRoute(k);
00299             int pre=-1, succ=-1;
00300             double costTimeRoute = 0.0, fitnessScoreRoute = 0.0;
00301             pre = depot;
00302             int index = 0;
00303             strings[index++] = "" + pre;
00304             strings[index++] = "" + this.getProblem().getX(pre);
00305             strings[index++] = "" + this.getProblem().getY(pre);
00306             strings[index++] = "" + this.getProblem().getReadyTime(pre);
00307             strings[index++] = "" + this.getProblem().getDueTime(pre);
00308             strings[index++] = "" + 0;
00309             strings[index++] = "" + 0;
00310             strings[index++] = "" + this.getProblem().getServiceTime(pre);
00311             text += ExpositoUtilities.getFormat(strings, width);
00312             text += "\n";
00313             do { // recorremos la ruta
00314                 index = 0;
00315                 suc = this.getSuccessor(pre);
00316                 textSolution += pre + " - ";
00317                 strings[index++] = "" + suc;
00318                 strings[index++] = "" + this.getProblem().getX(suc);
00319                 strings[index++] = "" + this.getProblem().getY(suc);

```

```

00317             strings[index++] = "" + this.getProblem().getReadyTime(suc);
00318             strings[index++] = "" + this.getProblem().getDueTime(suc);
00319             costTimeRoute += this.getDistance(pre, suc);
00320             if(costTimeRoute < (this.getProblem().getDueTime(suc))) {
00321                 if(costTimeRoute < this.getProblem().getReadyTime(suc)) {
00322                     costTimeRoute = this.getProblem().getReadyTime(suc);
00323                 }
00324                 strings[index++] = "" + costTimeRoute;
00325                 costTimeRoute += this.getProblem().getServiceTime(suc);
00326                 strings[index++] = "" + costTimeRoute;
00327                 strings[index++] = "" + this.getProblem().getServiceTime(pre);
00328                 if(costTimeRoute > this.getProblem().getMaxTimePerRoute()) { validSolution =
00329                     false; }
00330                     fitnessScoreRoute += this.problem.getScore(suc);
00331                 } else { validSolution = true; }
00332                 pre = suc;
00333                 text += ExpositoUtilities.getFormat(strings, width);
00334                 text += "\n";
00335             } while(suc != depot);
00336             textSolution += suc+"\n";
00337             costTimeSolution += costTimeRoute;
00338             fitnessScore += fitnessScoreRoute;
00339         }
00340         textSolution += "FEASIBLE SOLUTION: "+validSolution+"\n"+"SCORE: "+fitnessScore+"\n"+TIME
00341         COST: "+costTimeSolution+"\n";
00342     }
00343
00344     public double evaluateFitness() {
00345         double objectiveFunction = 0.0;
00346         double objectiveFunctionPerRoute = 0.0;
00347         for(int k = 0; k < this.getCreatedRoutes(); k++) {
00348             int depot = this.getIndexRoute(k);
00349             int pre=depot, suc = -1;
00350             do {
00351                 suc = this.getSuccessor(pre);
00352                 objectiveFunctionPerRoute = objectiveFunctionPerRoute + this.problem.getScore(suc);
00353                 pre = suc;
00354             } while((suc != depot));
00355             objectiveFunction = objectiveFunction + objectiveFunctionPerRoute;
00356             objectiveFunctionPerRoute = 0.0;
00357         }
00358         return objectiveFunction;
00359     }
00360
00361     public int addRoute() {
00362         int depot = this.problem.getPOIs();
00363         depot++;
00364         int routePos = 1;
00365         for(int i = 0; i < this.routes.length; i++) {
00366             if(this.routes[i] != -1 && this.routes[i] != 0) {
00367                 depot = this.routes[i];
00368                 depot++;
00369                 routePos = i+1;
00370             }
00371         }
00372         this.routes[routePos] = depot;
00373         this.availableVehicles--;
00374         this.predecessors[depot] = depot;
00375         this.successors[depot] = depot;
00376         this.problem.addNodeDepot();
00377         return depot;
00378     }
00379
00380     public double printSolution() {
00381         for(int k = 0; k < this.getCreatedRoutes(); k++) {
00382             int depot = this.getIndexRoute(k);
00383             int pre=depot, suc = -1;
00384             do {
00385                 suc = this.getSuccessor(pre);
00386                 System.out.print(pre+" - ");
00387                 pre = suc;
00388             } while((suc != depot));
00389             System.out.println(suc+" ");
00390         }
00391         double fitness = this.evaluateFitness();
00392         System.out.println("SC="+fitness);
00393         return fitness;
00394     }
00395 }
00396

```



# Índice alfabético

addNode  
    top.TOPTW, 48  
addNodeDepot  
    top.TOPTW, 48  
addRoute  
    top.TOPTWSolution, 91  
aleatorySelectionRCL  
    top.TOPTWGRASP, 71  
ALIGNMENT\_LEFT  
    es.ull.esit.utilities.ExpositoUtilities, 34  
ALIGNMENT\_RIGHT  
    es.ull.esit.utilities.ExpositoUtilities, 34  
arr  
    es.ull.esit.utilities.PowerSet< E >, 43  
availableVehicles  
    top.TOPTWSolution, 106  
  
BellmanFord  
    es.ull.esit.utilities.BellmanFord, 14  
bset  
    es.ull.esit.utilities.PowerSet< E >.PowerSetIterator,  
        46  
  
C:/Users/Kefle/Desktop/ExpositoTOP/src/es/ull/esit/utilities/BellmanFord.java  
    Pair.java, 38  
    top.TOPTWSolution, 91, 92  
C:/Users/Kefle/Desktop/ExpositoTOP/src/es/ull/esit/utilities/ExpositoUtilities.java  
    Pair.java, 13  
C:/Users/Kefle/Desktop/ExpositoTOP/src/es/ull/esit/utilities/PowerSet.java  
    BellmanFord, 14  
    calculateEdges, 15  
C:/Users/Kefle/Desktop/ExpositoTOP/src/es/ull/esit/utils/Pair.java  
    distanceMatrix, 17  
    distances, 17  
C:/Users/Kefle/Desktop/ExpositoTOP/src/top/mainTOPTW.java  
    edges1, 17  
    edges2, 17  
C:/Users/Kefle/Desktop/ExpositoTOP/src/top/TOPTW.java  
    getDistances, 15  
    getValue, 15  
C:/Users/Kefle/Desktop/ExpositoTOP/src/top/TOPTWEvaluator.java  
    INFINITY, 17  
    nodes, 17  
C:/Users/Kefle/Desktop/ExpositoTOP/src/top/TOPTWGRASP.java  
    path, 17  
    solve, 16  
C:/Users/Kefle/Desktop/ExpositoTOP/src/top/TOPTWReader.java  
    value, 18  
    es.ull.esit.utilities.ExpositoUtilities, 18  
C:/Users/Kefle/Desktop/ExpositoTOP/src/top/TOPTWRoute.java  
    ALIGNMENT\_LEFT, 34  
    ALIGNMENT\_RIGHT, 34  
C:/Users/Kefle/Desktop/ExpositoTOP/src/top/TOPTWSolution.java  
    DEFAULT\_COLUMN\_WIDTH, 34  
    getFirstAppearance, 19  
    getFormat, 20, 21, 23–28  
    isAcyclic, 29  
    isDouble, 29  
    isInteger, 30  
    multiplyMatrices, 31  
calculateDistanceMatrix  
    top.TOPTW, 49  
calculateEdges  
    es.ull.esit.utilities.BellmanFord, 15  
comprehensiveEvaluation

printFile, 31  
 simplifyString, 32  
 thereIsPath, 32  
 writeTextToFile, 33  
**es.ull.esit.utilities.PowerSet< E >**, 40  
 arr, 43  
 iterator, 42  
 PowerSet, 41  
**es.ull.esit.utilities.PowerSet< E >.PowerSetIterator**, 43  
 bset, 46  
 hasNext, 44  
 next, 44  
 remove, 45  
**es.ull.esit.utils**, 11  
**es.ull.esit.utils.Pair< F, S >**, 36  
 create, 38  
 equals, 38  
 first, 40  
 hashCode, 39  
 Pair, 37  
 second, 40  
**evaluate**  
 top.TOPTWEvaluator, 68  
**evaluateFitness**  
 top.TOPTWSolution, 93  
**first**  
 es.ull.esit.utils.Pair< F, S >, 40  
**fuzzySelectionAlphaCutRCL**  
 top.TOPTWGRASP, 76  
**fuzzySelectionBestFDRCL**  
 top.TOPTWGRASP, 77  
**getAvailableVehicles**  
 top.TOPTWSolution, 94  
**getCreatedRoutes**  
 top.TOPTWSolution, 94  
**getDistance**  
 top.TOPTW, 49–51  
 top.TOPTWSolution, 95  
**getDistances**  
 es.ull.esit.utilities.BellmanFord, 15  
**getDueTime**  
 top.TOPTW, 52  
**getFirstAppearance**  
 es.ull.esit.utilities.ExpositoUtilities, 19  
**getFormat**  
 es.ull.esit.utilities.ExpositoUtilities, 20, 21, 23–28  
**getId**  
 top.TOPTWRoute, 86  
**getIndexRoute**  
 top.TOPTWSolution, 95  
**getInfoSolution**  
 top.TOPTWSolution, 96  
**getMaxRoutes**  
 top.TOPTW, 53  
**getMaxScore**  
 top.TOPTWGRASP, 77  
**getMaxTimePerRoute**  
 top.TOPTW, 53  
**getNodes**  
 top.TOPTW, 54  
**getObjectiveFunctionValue**  
 top.TOPTWSolution, 98  
**getPOIs**  
 top.TOPTW, 54  
**getPositionInRoute**  
 top.TOPTWSolution, 99  
**getPredeccesor**  
 top.TOPTWRoute, 86  
**getPredecessor**  
 top.TOPTWSolution, 99  
**getPredecessors**  
 top.TOPTWSolution, 99  
**getProblem**  
 top.TOPTWSolution, 100  
**getReadyTime**  
 top.TOPTW, 54  
**getScore**  
 top.TOPTW, 55  
**getServiceTime**  
 top.TOPTW, 56  
**getSolution**  
 top.TOPTWGRASP, 78  
**getSolutionTime**  
 top.TOPTWGRASP, 78  
**getSuccesor**  
 top.TOPTWRoute, 86  
**getSuccessor**  
 top.TOPTWSolution, 100  
**getSuccessors**  
 top.TOPTWSolution, 101  
**getTime**  
 top.TOPTW, 57  
**getValue**  
 es.ull.esit.utilities.BellmanFord, 15  
**getVehicles**  
 top.TOPTW, 58  
**getWaitingTime**  
 top.TOPTWSolution, 101  
**getX**  
 top.TOPTW, 58  
**getY**  
 top.TOPTW, 59  
**GRASP**  
 top.TOPTWGRASP, 78  
**hashCode**  
 es.ull.esit.utils.Pair< F, S >, 39  
 top.TOPTWSolution, 101  
**hasNext**  
 es.ull.esit.utilities.PowerSet< E >.PowerSetIterator,  
 44  
**INFINITY**  
 es.ull.esit.utilities.BellmanFord, 17  
**initSolution**  
 top.TOPTWSolution, 102

isAcyclic  
    es.ull.esit.utilities.ExpositoUtilities, 29

isDepot  
    top.TOPTW, 60  
    top.TOPTWSolution, 102

isDouble  
    es.ull.esit.utilities.ExpositoUtilities, 29

isInteger  
    es.ull.esit.utilities.ExpositoUtilities, 30

iterator  
    es.ull.esit.utilities.PowerSet< E >, 42

Lista de obsoletos, 1

main  
    top.mainTOPTW, 35

maxRoutes  
    top.TOPTW, 66

maxTimePerRoute  
    top.TOPTW, 66

multiplyMatrices  
    es.ull.esit.utilities.ExpositoUtilities, 31

next  
    es.ull.esit.utilities.PowerSet< E >.PowerSetIterator,  
        44

NO\_EVALUATED  
    top.TOPTWEvaluator, 69  
    top.TOPTWGRASP, 82

NO\_INITIALIZED  
    top.TOPTWSolution, 106

nodes  
    es.ull.esit.utilities.BellmanFord, 17  
    top.TOPTW, 66

objectiveFunctionValue  
    top.TOPTWSolution, 106

Pair  
    es.ull.esit.utils.Pair< F, S >, 37

path  
    es.ull.esit.utilities.BellmanFord, 17

positionInRoute  
    top.TOPTWSolution, 106

PowerSet  
    es.ull.esit.utilities.PowerSet< E >, 41

predecessors  
    top.TOPTWSolution, 106

printFile  
    es.ull.esit.utilities.ExpositoUtilities, 31

printSolution  
    top.TOPTWSolution, 103

problem  
    top.TOPTWSolution, 106

RANDOM  
    top.TOPTWGRASP, 82

readProblem  
    top.TOPTWReader, 83

readyTime  
    top.TOPTWGRASP, 83

remove  
    es.ull.esit.utilities.PowerSet< E >.PowerSetIterator,  
        45

routes  
    top.TOPTWSolution, 107

score  
    top.TOPTW, 67

second  
    es.ull.esit.utils.Pair< F, S >, 40

serviceTime  
    top.TOPTW, 67

setAvailableVehicles  
    top.TOPTWSolution, 103

setDueTime  
    top.TOPTW, 61

setId  
    top.TOPTWRoute, 87

setMaxRoutes  
    top.TOPTW, 61

setMaxTimePerRoute  
    top.TOPTW, 61

setNodes  
    top.TOPTW, 62

setObjectiveFunctionValue  
    top.TOPTWSolution, 104

setPositionInRoute  
    top.TOPTWSolution, 104

setPredeccesor  
    top.TOPTWRoute, 87

setPredecessor  
    top.TOPTWSolution, 104

setReadyTime  
    top.TOPTW, 62

setScore  
    top.TOPTW, 63

setServiceTime  
    top.TOPTW, 63

setSolution  
    top.TOPTWGRASP, 80

setSolutionTime  
    top.TOPTWGRASP, 81

setSuccesor  
    top.TOPTWRoute, 87

setSuccessor  
    top.TOPTWSolution, 105

setWaitingTime  
    top.TOPTWSolution, 105

setX  
    top.TOPTW, 64

setY  
    top.TOPTW, 64

simplifyString  
    es.ull.esit.utilities.ExpositoUtilities, 32

solution  
    top.TOPTWGRASP, 83

solutionTime  
    top.TOPTWGRASP, 83

solve  
     es.ull.esit.utilities.BellmanFord, 16  
 successors  
     top.TOPTWSolution, 107  
  
 thereIsPath  
     es.ull.esit.utilities.ExpositoUtilities, 32  
  
 top, 11  
 top.mainTOPTW, 34  
     main, 35  
  
 top.TOPTW, 46  
     addNode, 48  
     addNodeDepot, 48  
     calculateDistanceMatrix, 49  
     depots, 66  
     distanceMatrix, 66  
     dueTime, 66  
     getDistance, 49–51  
     getDueTime, 52  
     getMaxRoutes, 53  
     getMaxTimePerRoute, 53  
     getNodes, 54  
     getPOIs, 54  
     getReadyTime, 54  
     getScore, 55  
     getServiceTime, 56  
     getTime, 57  
     getVehicles, 58  
     getX, 58  
     getY, 59  
     isDepot, 60  
     maxRoutes, 66  
     maxTimePerRoute, 66  
     nodes, 66  
     readyTime, 67  
     score, 67  
     serviceTime, 67  
     setDueTime, 61  
     setMaxRoutes, 61  
     setMaxTimePerRoute, 61  
     setNodes, 62  
     setReadyTime, 62  
     setScore, 63  
     setServiceTime, 63  
     setX, 64  
     setY, 64  
     TOPTW, 48  
     toString, 65  
     vehicles, 67  
     x, 67  
     y, 67  
  
 top.TOPTWEvaluator, 68  
     evaluate, 68  
     NO\_EVALUATED, 69  
  
 top.TOPTWGRASP, 69  
     aleatorySelectionRCL, 71  
     comprehensiveEvaluation, 71  
     computeGreedySolution, 73  
     fuzzySelectionAlphaCutRCL, 76  
  
     fuzzySelectionBestFDRCL, 77  
     getMaxScore, 77  
     getSolution, 78  
     getSolutionTime, 78  
     GRASP, 78  
     NO\_EVALUATED, 82  
     RANDOM, 82  
     setSolution, 80  
     setSolutionTime, 81  
     solution, 83  
     solutionTime, 83  
     TOPTWGRASP, 70  
     updateSolution, 81  
  
 top.TOPTWReader, 83  
     readProblem, 83  
  
 top.TOPTWRoute, 85  
     getId, 86  
     getPredeccesor, 86  
     getSuccesor, 86  
     setId, 87  
     setPredeccesor, 87  
     setSuccesor, 87  
  
 top.TOPTWSolution, 88  
     addRoute, 91  
     availableVehicles, 106  
     equals, 91, 92  
     evaluateFitness, 93  
     getAvailableVehicles, 94  
     getCreatedRoutes, 94  
     getDistance, 95  
     getIndexRoute, 95  
     getInfoSolution, 96  
     getObjectiveFunctionValue, 98  
     getPositionInRoute, 99  
     getPredecessor, 99  
     getPredecessors, 99  
     getProblem, 100  
     getSuccessor, 100  
     getSuccessors, 101  
     getWaitingTime, 101  
     hashCode, 101  
     initSolution, 102  
     isDepot, 102  
     NO\_INITIALIZED, 106  
     objectiveFunctionValue, 106  
     positionInRoute, 106  
     predecessors, 106  
     printSolution, 103  
     problem, 106  
     routes, 107  
     setAvailableVehicles, 103  
     setObjectiveFunctionValue, 104  
     setPositionInRoute, 104  
     setPredecessor, 104  
     setSuccessor, 105  
     setWaitingTime, 105  
     successors, 107  
     TOPTWSolution, 90

waitingTime, 107  
TOPTW  
    top.TOPTW, 48  
TOPTWGRASP  
    top.TOPTWGRASP, 70  
TOPTWSolution  
    top.TOPTWSolution, 90  
toString  
    top.TOPTW, 65  
  
updateSolution  
    top.TOPTWGRASP, 81  
  
value  
    es.ull.esit.utilities.BellmanFord, 18  
vehicles  
    top.TOPTW, 67  
  
waitingTime  
    top.TOPTWSolution, 107  
writeTextToFile  
    es.ull.esit.utilities.ExpositoUtilities, 33  
  
x  
    top.TOPTW, 67  
  
y  
    top.TOPTW, 67