



POLITECNICO DI TORINO

Master's Degree in Ingegneria Matematica

**Helper**

Andrea Baccolo s329570

Accademic Year 2024-2025

# Contenuti

<b>1</b>	<b>Helper</b>	<b>1</b>
1.1	Orbits . . . . .	2
1.1.1	GeosyncCircOrb . . . . .	2
1.2	Satellites . . . . .	2
1.2.1	FuelContainer . . . . .	2
1.2.2	RefillProp . . . . .	3
1.2.3	SpacePosition . . . . .	3
1.2.4	Station . . . . .	3
1.2.5	Target . . . . .	4
1.2.6	SSc . . . . .	4
1.3	Maneuvers . . . . .	5
1.3.1	Maneuver . . . . .	5
1.3.2	Refilling . . . . .	6
1.3.3	OrbitalManeuver . . . . .	6
1.3.4	Phasing . . . . .	6
1.3.5	Planar Change . . . . .	7
1.4	Simulation Folder . . . . .	7
1.4.1	reach . . . . .	8
1.4.2	State . . . . .	8
1.4.3	TourInfo . . . . .	9
1.4.4	Solution . . . . .	11
1.4.5	Simulator . . . . .	12
1.5	Destroy . . . . .	12
1.5.1	DesFirst . . . . .	14
1.5.2	DesLast . . . . .	14
1.5.3	DesRandom . . . . .	14
1.5.4	DesSSc . . . . .	14
1.5.5	DesSScCost . . . . .	15
1.5.6	DesSScRandom . . . . .	15
1.5.7	DesRelated . . . . .	15
1.5.8	DesRelatedGreedy . . . . .	16
1.5.9	DesRelatedRandom . . . . .	16
1.5.10	DesTour . . . . .	17
1.5.11	DesTour . . . . .	17
1.5.12	DesTourCost . . . . .	17

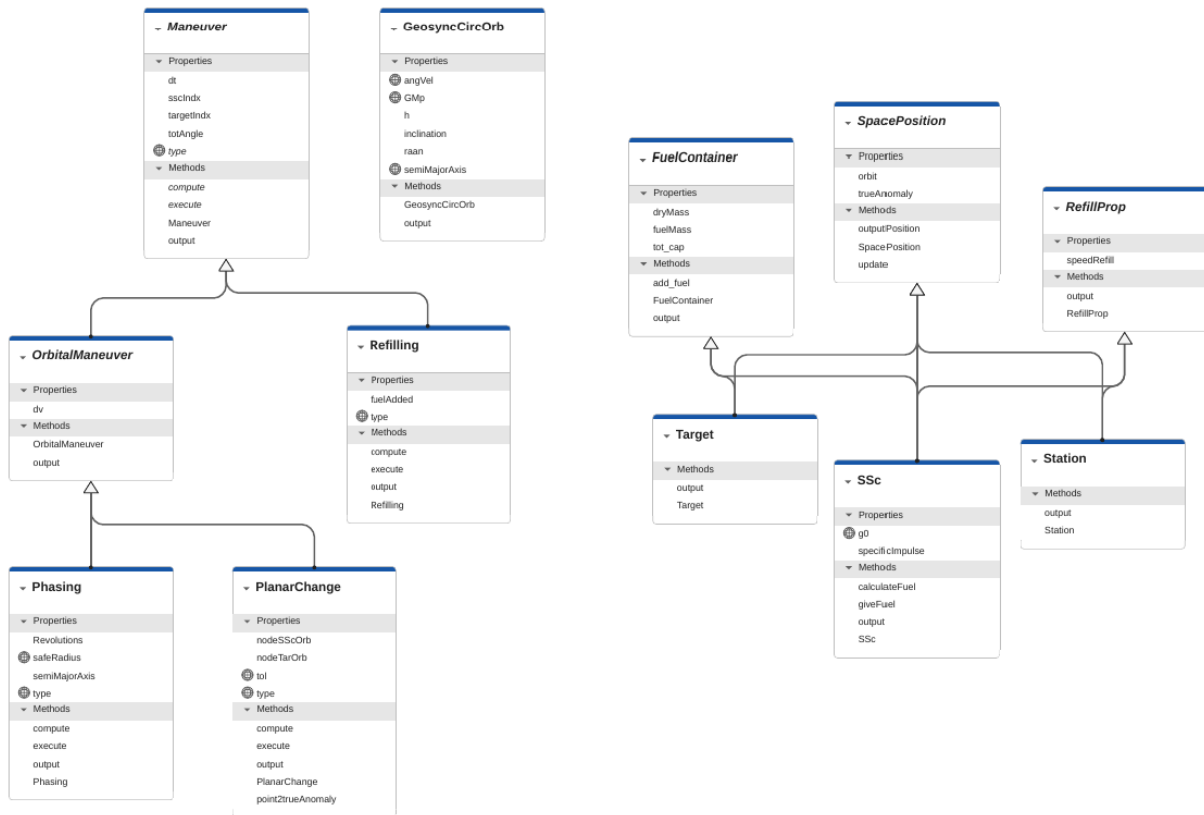
1.5.13	DesTourRandom . . . . .	17
1.5.14	DesTourSmall . . . . .	17
1.5.15	createDesSet . . . . .	18
1.6	Repair . . . . .	18
1.6.1	RepFarIns . . . . .	19
1.6.2	RepFarInsNear . . . . .	20
1.6.3	RepFarInsSim . . . . .	21
1.6.4	RepRandom . . . . .	21
1.6.5	createRepSet . . . . .	22
1.7	Optimizer . . . . .	22
1.7.1	GeneralALNS . . . . .	23
1.7.2	StopNIter . . . . .	26
1.7.3	Acceptance . . . . .	26
1.7.4	AcceptSA . . . . .	26
1.7.5	AcceptGreedy . . . . .	26
1.7.6	Destroy policy . . . . .	26
1.7.7	desFixedPol . . . . .	27
1.7.8	desInceasPol . . . . .	27
1.7.9	desRandomPol . . . . .	27
1.7.10	composed optimizer . . . . .	27
1.8	OtherFiles . . . . .	27
1.8.1	Relatedness . . . . .	27
1.8.2	initialSeq . . . . .	29
1.9	Test . . . . .	29
1.9.1	checkInstance . . . . .	29
1.9.2	createInstanceProblem . . . . .	29
1.9.3	outputTest . . . . .	30

# Chapter 1

## Helper

In this chapter, all classes and functions used in the repository are explained. In the following figure, some information about the maneuvers, orbits and satellites are summarized in a class diagram. Some objects used in simulation has a method called *output* which takes as input a *fid* variable and it writes or display some information about that object. The *fid* variable is obtained when a file is opened, if it is equal to one, then the output is displayed in the command window. The objects that possessed this method are all the object in the *AstroObj* folder, the *Solution*, *TourInfo* and *State* object. The only exemption is the *State* object, which differs slightly from the other methods as explained in it's proper subsection.

Figure 1.1: Class diagram of Maneuvers, geo orbits and Satellites



## 1.1 Orbits

### 1.1.1 GeosyncCircOrb

Class used to make an instance of a geosynchronous circular orbit with respect the equatorial coordinate system.

- Properties:
  - *inclination*: inclination angle in degrees.
  - *raan*: raan angle in degrees.
  - *h*: angular momentum.
  - *semiMajorAxis* = 42165 km.
  - *angVel*: =  $7.2919e - 05$  rad/s.
  - *GMp*: =  $398600 \text{ km}^3/\text{s}^2$ , product between gravitational constant and Earth's mass.
- *GeosyncCircOrb*: Constructor method.
  - *INPUTS*:
    - \* *inclination*: inclination of the orbit.
    - \* *raan*: right ascension of the ascending node of the orbit.
  - *OUTPUTS*:
    - \* *orbit object*: geosynchronous circular orbit instance.

## 1.2 Satellites

### 1.2.1 FuelContainer

Implementing Fuel properties.

- Properties:
  - *dryMass*: mass of the object without the fuel.
  - *fuelMass*: fuel that can be used by the object.
  - *tot\_cap*: total capacity of the satellite's tank.
- *FuelContainer*: Constructor method.
  - *INPUTS*:
    - \* *dryMass*: mass of the empty satellite.
    - \* *fuelMass*: level of fuel in the satellite.
    - \* *capacity*: maximum tank capacity of the satellite.
  - *OUTPUTS*:
    - \* *object*: returns object.
- *add\_fuel*: Refilling completely the tank of the object.
  - *INPUTS*:
    - \* *object*: the object to refill.
  - *OUTPUTS*:
    - \* *object*: the object with full capacity.

### 1.2.2 RefillProp

Implementing Refilling model.

- Properties:
  - *speedRefill*: speed of refueling in L/s.
- RefillProp: Constructor method.
  - *INPUTS*:
    - \* *speedRefill*: refueling speed in L/s.
  - *OUTPUTS*:
    - \* *object*: requested object.

### 1.2.3 SpacePosition

Class used to implement position of the satellite.

- Properties:
  - *orbit*: orbit object containing the orbit information
  - *trueAnomaly*: angle defining the position on the orbit
- *SpacePosition*: Constructor method.
  - *INPUTS*:
    - \* *orbit*: orbit of the satellite.
    - \* *trueAnomaly*: angle that describes the position on the orbit, measured from the ascending node with respect to the direction of rotation.
  - *OUTPUTS*:
    - \* *object*: requested object.
- *update*: Function that updates the position given a certain *dt* of time.
  - *INPUTS*:
    - \* *obj*: object to update.
    - \* *dt*: time used to update.
  - *OUTPUTS*:
    - \* *object*: requested object.

### 1.2.4 Station

Fuel station object.

*Station*: Constructor.

- *INPUTS*:
  - *orbit*: orbit of the satellite.
  - *trueAnomaly*: angle that describes the position on the orbit, measured from the ascending node with respect to the direction of rotation.
  - *speedRefill*: refueling speed in L/s.
- *OUTPUTS*:
  - *station obj*: station object.

### 1.2.5 Target

Target to refill.

*Target*: Constructor.

- *INPUTS*:
  - *orbit*: orbit of the satellite.
  - *trueAnomaly*: angle that describes the position on the orbit.
  - *dryMass*: mass of the empty satellite.
  - *fuelMass*: level of fuel in the satellite.
  - *tot\_cap*: maximum tank capacity.
- *OUTPUTS*:
  - *target obj*: target object.

### 1.2.6 SSc

Service Spacecraft (SSc) class.

- Properties:
  - *specificImpulse*: quantity that measures the efficiency of the spacecraft [ $s^{-1}$ ].
  - $g0 = 9.81m/s^2$
- *SSc*: Constructor method.
  - *INPUTS*:
    - \* *orbit*: orbit of the satellite.
    - \* *trueAnomaly*: angle that describes the position on the orbit.
    - \* *dryMass*: mass of the empty satellite.
    - \* *fuelMass*: level of fuel in the satellite.
    - \* *tot\_cap*: maximum tank capacity.
    - \* *speedRefill*: refueling speed in L/s.
    - \* *specificImpulse*: specific impulse of the ssc.
  - *OUTPUTS*:
    - \* *ssc obj*: service spacecraft object.
- *calculateFuel*: Function to compute the fuel used by the object.
  - *INPUTS*:
    - \* *obj*: the object used.
    - \* *dv*: variation of velocity that has to be applied.
    - \* *fuelSSc*: fuel of the object. This is used to compute the feasibility of reaching before updating the real one.
  - *OUTPUTS*:
    - \* *finalFuelMass*: computed final fuel mass.
    - \* *fuel*: fuel used in the maneuver.
    - \* *infeas*: flag of infeasibility (1 if infeasible, 0 if feasible).
- *giveFuel*: Function that updates the fuel quantity by taking away some fuel.
  - *INPUTS*:
    - \* *obj*: object from which fuel is taken away.
    - \* *quantity*: quantity of fuel to remove.

- *OUTPUTS*:
  - \* *obj*: final object without the removed fuel.

## 1.3 Maneuvers

Here all the considered maneuvers are implemented as classes. The following two methods are implemented through override to all maneuvers:

- *execute*: This method updates only the SSc position, checks infeasibility, and calculates the fuel used during a generic maneuver.
  - *INPUTS*:
    - \* *obj*: maneuver to be executed.
    - \* *simState*: state to update.
  - *OUTPUTS*:
    - \* *simState*: updated state.
    - \* *fuelUsed*: fuel used during execution.
- *compute*: Calculate maneuver.
  - *INPUTS*:
    - \* *obj*: maneuver to be updated.
    - \* *ssc*: ssc object that needs to reach the target.
    - \* *target*: target object to reach.
    - \* *fuelReal*: actual ssc fuel value when maneuver is performed.
  - *OUTPUTS*:
    - \* *obj*: computed maneuver.

### 1.3.1 Maneuver

This general class stores all common information needed to calculate and execute maneuvers for every maneuver,

- Properties:
  - *dt*: time to perform the maneuver.
  - *totAngle*: total angle done in dt of time.
  - *sscIdx*: index that identifies which SSc is involved in the maneuver.
  - *targetIdx*: index that identifies which target is involved in the maneuver.
  - *type*: string that stores the type of maneuver performed.
- *Maneuver*: Constructor method.
  - *METHOD*: Constructor.
  - *INPUTS*:
    - \* *sscIdx*: index of the ssc performing the maneuver.
    - \* *targetIdx*: index of the target to reach.
    - \* *dt*: total duration of the maneuver in seconds.
    - \* *totAngle*: total angle corresponding to time dt.
  - *OUTPUTS*:
    - \* *Maneuver obj*: maneuver object.



### 1.3.2 Refilling

Implementing refueling maneuver.

- Properties:
  - *fuelAdded*: fuel added during the refilling
- *Refilling*: Constructor method.
  - *INPUTS*:
    - \* *sscIdx*: index of the ssc performing the maneuver.
    - \* *targetIdx*: index of the target to reach.
    - \* *dt*: total duration of the maneuver in seconds.
    - \* *totAngle*: total angle corresponding to time dt.
    - \* *fuelAdded*: amount of fuel added to the target/ssc.
  - *OUTPUTS*:
    - \* *Maneuver obj*: refilling maneuver object.

### 1.3.3 OrbitalManeuver

General class that implements common Orbital Maneuvers's properties.

- Properties:
  - *dv*: variations of velocity to apply.
- *OrbitalManeuver*: Constructor method.
  - *METHOD*: Constructor.
  - *INPUTS*:
    - \* *sscIdx*: index of the ssc.
    - \* *targetIdx*: index of the target.
    - \* *dt*: duration in seconds.
    - \* *totAngle*: angle corresponding to time dt.
    - \* *dv*: total velocity increment.
  - *OUTPUTS*:
    - \* *Orbital Maneuver obj*: orbital maneuver object.

### 1.3.4 Phasing

Implementing phasing, the maneuver to rendezvous with the target.

- Properties:
  - *semiMajorAxis*: semimajor axis of the resulting orbit.
  - *Revolutions*: number of revolutions used for phasing.
  - *safeRadius* =  $6378 + 2500$ : earth radius + 2500km.
- *Phasing*: Constructor method.
  - *INPUTS*:
    - \* *sscIdx*: index of the ssc.
    - \* *targetIdx*: index of the target.
    - \* *dt*: total duration of the maneuver in seconds.
    - \* *totAngle*: total angle corresponding to time dt.

- \* *dv*: total velocity increment.
- \* *semiMajoAxis*: semimajor axis of the transfer orbit.
- \* *Revolutions*: total number of revolutions to reach the target.
- *OUTPUTS*:
- \* *Phasing obj*: phasing maneuver object.

### 1.3.5 Planar Change

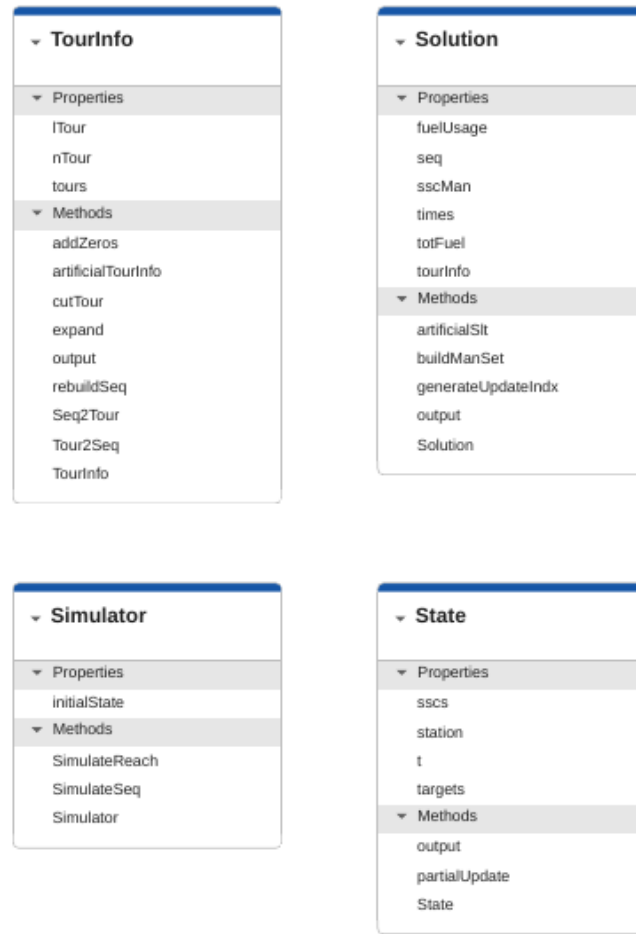
Implementing planar change, a maneuver to change inclination and raan of an orbit.

- Properties:
  - *nodeTarOrb*: node position true anomaly with respect to the target orbit.
  - *nodeSScOrb*: node position true anomaly with respect to the SSc orbit.
  - *tol*: =  $1e - 8$  numerical tolerance.
- *PlanarChange*: Constructor method.
  - *METHOD*: Constructor.
  - *INPUTS*:
    - \* *sscIdx*: index of the ssc.
    - \* *targetIdx*: index of the target.
    - \* *dt*: duration in seconds.
    - \* *totAngle*: angle corresponding to time dt.
    - \* *dv*: total velocity increment.
    - \* *nodeTargetOrb*: true anomaly of the node with respect to the target orbit.
    - \* *nodeSScOrb*: true anomaly of the node with respect to the ssc orbit.
  - *OUTPUTS*:
    - \* *Planar change obj*: planar change maneuver object.
- *point2trueAnomaly* Function used to pass from a point in 3D to the true anomaly vector described with respect to a specific orbit.
- *INPUTS*:
  - *obj*: planar change object.
  - *r*: vector 3x1 containing the coordinates of the point to convert.
  - *omega*: raan of the reference orbit.
  - *i*: inclination of the reference orbit.
  - *a*: semimajor axis of the reference orbit.
- *OUTPUTS*:
  - *phi*: true anomaly with respect to the input orbit data.

## 1.4 Simulation Folder

In this section, some function and classes used to characterize the solution and to simulate are described. The following image shows all the classes used.

Figure 1.2: Class diagram of Simulation Classes



### 1.4.1 reach

Function used to compute the maneuvers to reach a target.

- **INPUTS:**
  - *ssc*: SSc object that has to compute the maneuver.
  - *sscIndx*: index of the ssc that performs the maneuver in the state vector.
  - *target*: target object that needs to be reached by the ssc.
  - *targetIndx*: index of the target in the state vector.
- **OUTPUTS:**
  - *maneuvers*: cell array of maneuvers to compute to reach the target.
  - *infeas*: flag of infeasibility (1 if infeasible, 0 if feasible).

### 1.4.2 State

Class that contains information about the problem in a specific point in time.

- **Properties:**
  - *sscs*: vector of nSSc SSc.
  - *targets*: vector of nTar Targets.
  - *station*: fuel station.

- *t*: starting point of the state.
- *State*: Constructor method.
  - *INPUTS*:
    - \* *sscs*: vector of SSc objects.
    - \* *targets*: vector of Target objects.
    - \* *station*: station object.
    - \* *time*: time instant of the state.
  - *OUTPUTS*:
    - \* *state obj*: state object.
- *partialUpdate*: Function that updates the positions of some targets and the station.
  - *INPUTS*:
    - \* *obj*: state to update.
    - \* *dt*: time interval for updating.
    - \* *updateIndex*: indices of targets to update.
  - *OUTPUTS*:
    - \* *obj*: updated state object.
- *output*: Function that prints or writes the updated position of a given ssc and some targets.
  - *INPUTS*:
    - \* *obj*: state to update.
    - \* *fid*: file identifier or display flag (1 for display, >1 for writing to file).
    - \* *i*: ssc index to print.
    - \* *updateIndex*: set of targets to print.

### 1.4.3 TourInfo

Class of mission information for a specific sequence of targets. Implements information regarding tours and methods to manipulate and update them.

- Properties:
  - *tours*: cell array max(nTour)×nSSc containing targets of tours.
  - *lTour*: matrix max(nTour)×nSSc containing length of the tours.
  - *nTour*: number of tours for every SSc.
- *TourInfo*: Constructor method.
  - *INPUTS*:
    - \* *sequence*: sequence of the solution.
    - \* *nTar*: number of targets.
  - *OUTPUTS*:
    - \* *TourInfo obj*: TourInfo object.
- *artificialTourInfo*: Create an artificial TourInfo object with given data.
  - *INPUTS*:
    - \* *obj*: TourInfo object.
    - \* *tours*: cell arrays of tours.
    - \* *lTour*: length matrix of tours.

- \* *nTour*: vector giving the number of non-empty tours per ssc.
- *OUTPUTS*:
  - \* *TourInfo obj*: constructed object.
- *rebuildSeq*: From the tour information, create the sequence.
  - *INPUTS*:
    - \* *obj*: TourInfo object from which the sequence will be constructed.
    - \* *nTar*: number of targets to fill the gap and complete the sequence.
  - *OUTPUTS*:
    - \* sequence to construct.
- *addZeros*: Obtain the tour structure with zeros to simulate.
  - *INPUTS*:
    - \* *obj*: TourInfo object.
  - *OUTPUTS*:
    - \* cell array with the same dimensions of the tour attributes, with all initial and final zeros.
- *cutTour*: Function used to cut unnecessary parts of TourInfo.
  - *INPUTS*:
    - \* *obj*: TourInfo object.
  - *OUTPUTS*:
    - \* TourInfo object without empty tour rows in all information.
- *Tour2Seq*: Convert a position in a specific tour to its position on the sequence row of the SSC.
  - *INPUTS*:
    - \* *obj*: TourInfo object.
    - \* *sscIdx*: index of the SSC in the considered tour.
    - \* *tourIdx*: index of the tour.
    - \* *posTour*: index of the position inside the tour.
  - *OUTPUTS*:
    - \* *posSeq*: position on the sequence row corresponding to the SSC.
- *Seq2Tour*: Convert a position on the sequence row of the SSC to a position in a specific tour.
  - *INPUTS*:
    - \* *obj*: TourInfo object.
    - \* *sscIdx*: index of the SSC in the considered tour.
    - \* *posSeq*: position on the sequence row of the SSC.
  - *OUTPUTS*:
    - \* *tourIdx*: index of the tour.
    - \* *posTour*: index of the position inside the tour.
- *expand*: Expand the TourInfo structure by *k* tours.
  - *INPUTS*:
    - \* *obj*: TourInfo object.
    - \* *k*: number of expansions.
  - *OUTPUTS*:
    - \* *obj*: expanded object.

### 1.4.4 Solution

Class used to collect solution information.

- Properties:
  - *seq*: refueling sequence for every ssc ( $nSSc, m$ ).
  - *sscMan*: cell array of maneuvers selected from *seq*.
  - *fuelUsage*: fuel used to reach every target.
  - *times*: time employed for every maneuver.
  - *totFuel*: total fuel used for the solution.
  - *tourInfo*: TourInfo structure.
- *Solution*: Constructor.
  - *INPUTS*:
    - \* *seq*: sequence of the solution.
    - \* *nTar*: total number of targets.
    - \* *initialState*: initial state from which the simulation starts.
  - *OUTPUTS*:
    - \* *obj*: Solution object.
- *artificialSlt*: From existing inputs, create the object.
  - *INPUTS*:
    - \* *obj*: Solution object (MATLAB does not allow multiple constructors).
    - \* *seq*: refueling sequence for every SSC ( $n_{SSC}, m$ ).
    - \* *sscMan*: cell array containing cell arrays of maneuvers selected from *seq*.
    - \* *fuelUsage*: quantity of fuel used to reach every target.
    - \* *times*: time employed for every maneuver.
    - \* *nTar*: total number of targets.
  - *OUTPUTS*:
    - \* *obj*: Solution object with the requested inputs.
- *buildManSet*: Create the set of maneuvers associated with the solution.
  - *INPUTS*:
    - \* *obj*: Solution object.
    - \* *initialState*: state object with the initial targets, SSCs, and station.
    - \* *fid*: optional parameters used to display or write to a file.
  - *OUTPUTS*:
    - \* *obj*: Solution object with SSC maneuvers.
    - \* *state*: final state after executing the sequence.
- *generateUpdateIndx*: Generate the update index of a specific row of a sequence.
  - *INPUTS*:
    - \* *obj*: Solution object (needed as parameter, even if unused).
    - \* *seq*: complete sequence matrix.
    - \* *nTar*: total number of targets.
    - \* *i*: row of the sequence chosen.
  - *OUTPUTS*:
    - \* *updateIndx*: vector of targets that need to be updated during the simulation of SSC *i*.

### 1.4.5 Simulator

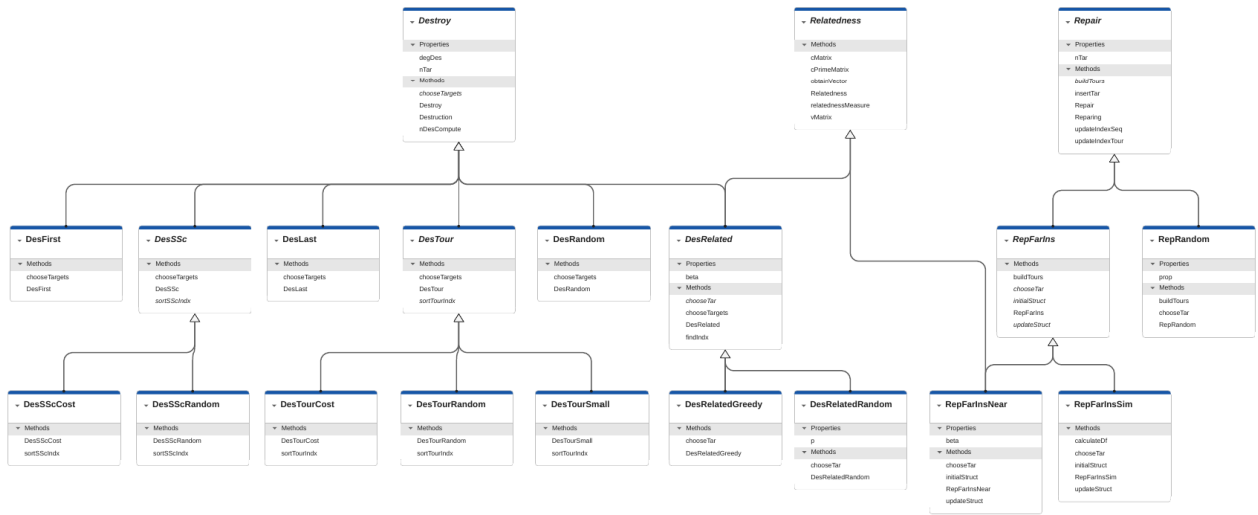
Collection of methods used to update a state through simulation.

- Properties:
  - *initialState*: initial state from which the simulation starts.
- Simulator: Constructor method.
  - *INPUTS*:
    - \* *initialState*: state object with the initial targets and sscs.
  - *OUTPUTS*:
    - \* *Simulator obj*: simulator object.
- *SimulateReach*: Function used to simulate one single SSC reaching one single target.
  - *INPUTS*:
    - \* *obj*: Simulator object (required even if unused).
    - \* *simState*: initial state from which the simulation and maneuvers are computed.
    - \* *sscIdx*: index of the SSC in the initial state vector of SSCs.
    - \* *targetIdx*: index of the target to reach.
    - \* *updateIndex*: indices of targets to update.
    - \* *fid*: optional parameters used to display or write to a file.
  - *OUTPUTS*:
    - \* *simState*: State object containing initial state info.
    - \* *infeas*: infeasibility flag (1 if infeasible, 0 if feasible).
    - \* *totFuel*: total fuel consumption.
    - \* *totTime*: total time required for the reach.
    - \* *maneuvers*: cell array with the maneuvers to be performed.
- *SimulateSeq*: Function used to simulate a sequence given as input (it can represent a full row or a portion of the solution sequence).
  - *INPUTS*:
    - \* *obj*: Simulator object (required even if unused).
    - \* *simState*: initial state from which the simulation and maneuvers are computed.
    - \* *sscIdx*: index of the SSC in the initial state vector of SSCs.
    - \* *seq*: sequence to simulate (can also be a tour or a portion of a sequence row).
    - \* *updateIndex*: indices of targets to update.
    - \* *fid*: optional parameters used to display or write to a file.
  - *OUTPUTS*:
    - \* *simState*: State object containing initial state info.
    - \* *infeas*: infeasibility flag (1 if infeasible, 0 if feasible).
    - \* *totFuel*: total fuel consumption.
    - \* *totTime*: total time required for the simulation.
    - \* *maneuvers*: cell array with the maneuvers to be performed.

## 1.5 Destroy

Class that implement the general destroy method. Its purpose is to delete part of the solutions to later rebuild in a hopefully better way. In the following figure is presented the inheritance of the destroy methods

Figure 1.3: Class diagram of Destroy methods



- Properties
  - $nTar$ : total number of target.
  - $degDes$ : degree of destruction of the destroyer.
- Destroy: Constructor.
  - INPUTS:
    - \*  $nTar$ : number of targets.
    - \*  $degDes$ : degree of destruction, a number between 0 and 100.
  - OUTPUTS:
    - \*  $obj$ : destroy object.
- Destruction: Application of the destruction of the specific destroyer.
  - INPUTS:
    - \*  $obj$ : destroy object.
    - \*  $slt$ : solution to destroy.
    - \*  $initialState$ : state object that contains the initial info.
  - OUTPUTS:
    - \*  $destroyedSet$ : row vector of destroyed set index.
    - \*  $tourInfos$ : tourInfo object with info of tours after destruction.
- $nDesCompute$ : Computes the total number of destroyed targets approximated using ceiling function.
  - INPUTS:
    - \*  $obj$ : destroy object.
  - OUTPUTS:
    - \*  $nDestroy$ : total number of destroyers.
- chooseTargets: Function that gives the total number of destroyed targets and their indexes.
  - INPUTS:
    - \*  $obj$ : destroy object.



- \* *slt*: solution to destroy.
- \* *initialState*: state object that contains the initial info.
- *OUTPUTS*:
  - \* *nDestroy*: total number of destroyers.
  - \* *destroyIndx*: matrix nDestroy x 3 where the first column is the sscIndx, the second the tourIndx, and the third the posTour.

### 1.5.1 DesFirst

Destroyer that deletes every first target from a tour.

*DesFirst*: Constructor.

- *INPUTS*:
  - *nTar*: number of targets.
  - *degDes*: degree of destruction, a number between 0 and 100.
- *OUTPUTS*:
  - *obj*: destroyFirst object.

### 1.5.2 DesLast

Destroyer that deletes every last target from a tour. *DesLast*: Constructor.

- *INPUTS*:
  - *nTar*: number of targets.
  - *degDes*: degree of destruction, a number between 0 and 100.
- *OUTPUTS*:
  - *obj*: destroyLast object.

### 1.5.3 DesRandom

Destroy method that removes random targets from the solution.

*DesRandom*: Constructor.

- *INPUTS*:
  - *nTar*: number of targets.
  - *degDes*: degree of destruction, a number between 0 and 100.
- *OUTPUTS*:
  - *obj*: destroyRandom object.

### 1.5.4 DesSSc

Destroy method that chooses all the targets on SSCs.

- *DesSSc*: Constructor.
  - *INPUTS*:
    - \* *nTar*: number of targets.
    - \* *degDes*: degree of destruction, a number between 0 and 100.
  - *OUTPUTS*:

- \* *obj*: destroySSc object.
- *sortSScIndx*: Sorting the SSC with respect to the cost.
  - *INPUTS*:
    - \* *obj*: destroy object.
    - \* *slt*: solution to destroy.
  - *OUTPUTS*:
    - \* *SSCs*: sorted SSC index.

### 1.5.5 DesSScCost

Destroy method that deletes targets of one or more SSCs from the solution by taking the most costly SSC tour.

*DesSScCost*: Constructor.

- *INPUTS*:
  - *nTar*: number of targets.
  - *degDes*: degree of destruction, a number between 0 and 100.
- *OUTPUTS*:
  - *obj*: destroySScCost object.

### 1.5.6 DesSScRandom

Destroy method that deletes targets of random SSCs.

*DesSScRandom*: Constructor.

- *INPUTS*:
  - *nTar*: number of targets.
  - *degDes*: degree of destruction, a number between 0 and 100.
- *OUTPUTS*:
  - *obj*: destroySScRandom object.

### 1.5.7 DesRelated

Abstract class that implements a general choice of targets to remove using a relatedness measure.

- properties:
  - *beta*: parameter used to give weights to the planar change and the phasing.
- *DesRelated* Constructor.
  - *INPUTS*:
    - \* *nTar*: number of targets.
    - \* *degDes*: degree of destruction, a number between 0 and 100.
    - \* *beta*: parameter used in the relatedness measure, between 0 and 1.
  - *OUTPUTS*:
    - \* *obj*: DestroyRelated object.
- *findIndx*: Function used to find the position of a target in a sequence.

- *INPUTS*:
  - \* *obj*: destroyFirst object.
  - \* *tarChosen*: chosen target to find.
  - \* *slt*: solution to destroy.
- *OUTPUTS*:
  - \* *indx*: position of the chosen target.
- *chooseTar*: Function that gives the total number of destroyed targets and their indexes.
- *INPUTS*:
  - \* *obj*: destroy object.
  - \* *slt*: solution to destroy.
  - \* *initialState*: state object that contains the initial info.
- *OUTPUTS*:
  - \* *nDestroy*: total number of destroyers.
  - \* *destroyIndx*: matrix nDestroy x 3 where the first column is the sscIndx, the second the tourIndx, and the third the posTour.

### 1.5.8 DesRelatedGreedy

Destroy method that applies a greedy policy to choose the best related target.

*DesRelatedGreedy*: Constructor.

- *INPUTS*:
  - *nTar*: number of targets.
  - *degDes*: degree of destruction, a number between 0 and 100.
  - *beta*: parameter used in the relatedness measure, between 0 and 1.
- *OUTPUTS*:
  - *obj*: DestroyRelatedGreedy object.

### 1.5.9 DesRelatedRandom

Destroy method that extracts the best related target using a probability from Han et al.

- properties:
  - *p*: parameter used to add randomness and not always take the best; lower values of *p* correspond to more randomness.
- *DesRelatedRandom*: Constructor.
- *INPUTS*:
  - \* *nTar*: number of targets.
  - \* *degDes*: degree of destruction, a number between 0 and 100.
  - \* *beta*: parameter used in the relatedness measure, between 0 and 1.
  - \* *p*: parameter to control the randomness of choosing the best targets,  $p \geq 1$ , 1 for complete randomness.
- *OUTPUTS*:
  - \* *obj*: DestroyRelatedRandom object.

### 1.5.10 DesTour

Destroy methods that delete targets from some tours of the solutions.

### 1.5.11 DesTour

- *DesTour*: Constructor.
  - *INPUTS*:
    - \* *nTar*: number of targets.
    - \* *degDes*: degree of destruction, a number between 0 and 100.
  - *OUTPUTS*:
    - \* *obj*: destroyTour object.
- *sortTourIndx*: Sorting the SSC with respect to the cost.
  - *INPUTS*:
    - \* *obj*: destroy object.
    - \* *slt*: solution to destroy.
  - *OUTPUTS*:
    - \* *SSCs*: sorted SSC index.

### 1.5.12 DesTourCost

Destroy methods that delete targets from the most expensive tours.

*DesTourCost*: Constructor.

- *INPUTS*:
  - *nTar*: number of targets.
  - *degDes*: degree of destruction, a number between 0 and 100.
- *OUTPUTS*:
  - *obj*: destroyTourCost object.

### 1.5.13 DesTourRandom

Destroy methods that delete targets from random tours.

*DesTourRandom*: Constructor.

- *INPUTS*:
  - *nTar*: number of targets.
  - *degDes*: degree of destruction, a number between 0 and 100.
- *OUTPUTS*:
  - *obj*: destroyTourRandom object.

### 1.5.14 DesTourSmall

Destroy methods that delete targets from the smallest tours.

*DesTourSmall*: Constructor.

- *INPUTS*:
  - *nTar*: number of targets.

- *degDes*: degree of destruction, a number between 0 and 100.
- *OUTPUTS*:
  - *obj*: destroyTourSmall object.

### 1.5.15 createDesSet

- *METHOD*: General function to create a destroy set.
- *INPUTS*:
  - *nTar*: number of targets.
  - *degDes*: destruction degree, a number between 0 and 100.
  - *nDestroy*: total number of destroyers.
  - *beta*: a number between 0 and 1 used in the relatedness measure.
  - *p*: related random parameter, a number greater than 1.
- *OUTPUTS*:
  - *desSet*: cell array with nDestroy destroyers.

## 1.6 Repair

General Repair method used to rebuild a new solution from the destroyed solution.

- Properties:
  - *nTar*: number of targets
- *Repair*: Constructor
  - *INPUTS*:
    - \* *nTar*: number of targets
  - *OUTPUTS*:
    - \* *obj*: Repair object
- *Repairing*: general repair function used to repair the solution
  - *INPUTS*:
    - \* *obj*: repair object
    - \* *initialState*: state object that contains the initial info
    - \* *destroyedSet*: row vector of destroyed set index
    - \* *tourInfo*: tourInfo object with info of tours after the destruction
  - *OUTPUTS*:
    - \* *slt*: final feasible solution object obtained by the repair
- *updateIndexTour*: function used to obtain the update index from a specific tour of a specific ssc united with the destroyedSet
  - *INPUTS*:
    - \* *obj*: repair object
    - \* *tourInfo*: tourInfo object with info of tours after the destruction
    - \* *destroyedSet*: row vector of destroyed set index
    - \* *currTour*: current tour from which new targets will be added
    - \* *currSSc*: considered ssc

- *OUTPUTS*:
  - \* *updateIndex*: row vector of update index to use in the simulation
- *updateIndexSeq*: function used to obtain the update index from the sequence united with the destroyedSet
- *INPUTS*:
  - \* *obj*: repair object
  - \* *seq*: row vector of the specific ssc considered
  - \* *destroyedSet*: row vector of destroyed set index
- *OUTPUTS*:
  - \* *updateIndex*: row vector of update index to use in the simulation
- *insertTar*: this function adds the new target in the position posSelect, shifting the vector tour by one position to the right
- *INPUTS*:
  - \* *obj*: repair object
  - \* *tour*: tour from which the target will be inserted, empty if new tour needs to be made
  - \* *tarSelect*: target index to add to the tour
  - \* *posSelect*: position in the tour where the target needs to be added
- *OUTPUTS*:
  - \* *newTour*: new requested tour, if the tour is empty, it returns just the targets without zeros; if the tour is not empty, it returns the new tour ready to simulate
- *buildTours*: general function that adds to the TourInfo the targets in a feasible way
- *INPUTS*:
  - \* *obj*: repair object
  - \* *destroyedSet*: row vector of destroyed set index
  - \* *tourInfo*: tourInfo object with info of tours after the destruction
  - \* *stateSsc*: state object that contains the initial info
- *OUTPUTS*:
  - \* *tourInfo*: the updated tourInfo information ready to be transformed into a sequence

### 1.6.1 RepFarIns

Class used to implement the farthest insertion repairs.

- *RepFarIns*: Constructor
- *INPUTS*:
  - \* *nTar*: number of targets
- *OUTPUTS*:
  - \* *obj*: Repair object
- *initialStruct*: function that creates the structure used to decide the target to insert. The initial structure will be the distance between the destroyed targets and all the other targets
- *INPUTS*:
  - \* *obj*: repair object
  - \* *state*: state object that contains the initial info
  - \* *destroyedSet*: row vector of destroyed set index

- \* *currSeq*: matrix of the current sequence
- *OUTPUTS*:
  - \* *struct*: initial structure used to decide the target to insert in the sequence
- chooseTar: function that chooses the target to insert using the structure
- *INPUTS*:
  - \* *obj*: repair object
  - \* *struct*: structure used
  - \* *initialState*: state object that contains the initial info
  - \* *currSeq*: matrix of the current sequence
  - \* *destroyedSet*: row vector of destroyed set index
- *OUTPUTS*:
  - \* *tarIndx*: index of the chosen target
  - \* *sscIndx*: index of the ssc where the target needs to be inserted
  - \* *posSeq*: index of the position of the sequence where the target needs to be inserted
- updateStruct: function used to update the structure after inserting the new target
- *INPUTS*:
  - \* *obj*: repair object
  - \* *struct*: initial structure used to decide the target to insert
  - \* *state*: state object that contains the initial info
  - \* *currSeq*: matrix of the current sequence
  - \* *sscIndx*: index of the ssc to consider
  - \* *tarIndx*: target index that has been inserted
  - \* *destroyedSet*: row vector of destroyed set index
  - \* *currDestroyed*: index of the target that has been inserted
- *OUTPUTS*:
  - \* *struct*: the updated structure

### 1.6.2 RepFarInsNear

Repair method that repairs using the farthest insertion heuristic based on the relatedness measure. The struct used by this method is a cell array of dimension 2. In the first cell there is a vector containing all the remaining destroyed target, in the second cell there is the cost matrix of the relatedness measure between the destroyed target and all the fixed target.

- Properties:
  - *beta*: parameters used to give weights to the planar change and the phasing
- RepFarInsNear: Constructor
- *INPUTS*:
  - \* *nTar*: number of targets
  - \* *beta*: parameters used in the relatedness measure between 0 and 1
- *OUTPUTS*:
  - \* *obj*: Repair object

### 1.6.3 RepFarInsSim

Repair method that repairs using the farthest insertion heuristic based on computing the variation of the objective function. The struct used by this method is a cell array long nSSc full of matrices nDestroyTar x number of position of the specific SSc considered.

- RepFarInsSim: Constructor
  - *INPUTS*:
    - \* *nTar*: number of targets
  - *OUTPUTS*:
    - \* *obj*: Repair object
- *calculateDf*: Constructor
  - *INPUTS*:
    - \* *obj*: repair object
    - \* *initialState*: state object that contains the initial info
    - \* *sscIndx*: index of the ssc where the target needs to be inserted
    - \* *destroyedSet*: row vector of destroyed set index
    - \* *currSeq*: row vector of the sscIndx row of the original sequence
  - *OUTPUTS*:
    - \* *df\_i*: single matrix of insertion cost regarding ssc sscIndx
- *chooseTar*: function that chooses the target to insert using the structure
  - *INPUTS*:
    - \* *obj*: repair object
    - \* *struct*: structure used
    - \* *initialState*: state object that contains the initial info
    - \* *currSeq*: matrix of the current sequence
    - \* *destroyedSet*: row vector of destroyed set index
  - *OUTPUTS*:
    - \* *tarIndx*: index of the chosen target
    - \* *sscIndx*: index of the ssc where the target needs to be inserted
    - \* *posSeq*: index of the position of the sequence where the target needs to be inserted

### 1.6.4 RepRandom

Class that implement the repair random (pseudo code on the thesis).

- properties:
  - *prop*: percentage of targets to check when using the random repair, between 1 and 100 , 100 is default.
  - *RepRandom*: Constructor
  - *INPUTS*:
    - \* *nTar*: number of targets
    - \* *prop*: percentage of targets to check when using the random repair, between 1 and 100 , 100 is default.
  - *OUTPUTS*:
    - \* *obj*: Repair object



- *chooseTar*: function used to choose the target
  - *INPUTS*:
    - \* *obj*: Repair object
    - \* *destroyedSet*: vector from which the random target is extracted
  - *OUTPUTS*:
    - \* *tarIndx*: the target extracted

### 1.6.5 createRepSet

- *METHOD*: General function to create a repair set
- *INPUTS*:
  - *nTar*: number of targets
  - *nRepair*: total number of destroyers, 4 is default.
  - *prop*: percentage of targets to check when using the random repair, between 1 and 100 , 100 is default.
  - *beta*: a number between 0 and 1 used in the relatedness measure (default 0.5)
- *OUTPUTS*:
  - *repSet*: cell array with nRepair repairs

## 1.7 Optimizer

Figure 1.4: Class diagram of Maneuvers



### 1.7.1 GeneralALNS

general ALNS algorithms with general destroy policy, accept criteria and stopping criteria. In the following image are shown all the inheritance done for those classes.

- properties:
  - *currSlt*: solution object that has the current solutions
  - *bestSlt*: solution object that has the best solutions
  - *state*: state object that contains the state info
  - *outCurrFuelSim*: cell array with length nRep with all the total value of the current solution for every replica
  - *outCurrIndxSim*: cell array with length nRep with all the index where the current solution value changes for every replica
  - *outBestFuelSim*: cell array with length nRep with all the total value of the best solution for every replica
  - *outBestIndxSim*: cell array with length nRep with all the index where the best solution value changes for every replica
  - *outBestSlt*: cell array with length nRep containing the best solutions for every replica
  - *outWeightsDes*: cell array with length nRep with the last destroy weights for every replica
  - *outWeightsRep*: cell array with length nRep with the last repair weights for every replica
  - *outNSelDes*: cell array with length nRep containing how many times a destroy has been called for every replica
  - *outNSelRep*: cell array with length nRep containing how many times a repair has been called for every replica
  - *nDestroy*: total number of destroyers
  - *nRepair*: total number of repairs
  - *nIter*: total number of iterations
  - *nRep*: total number of replicas
  - *desSet*: cell array containing all destroyers
  - *repSet*: cell array containing all repairs
  - *desWeights*: column vector of destroyers' weights
  - *repWeights*: column vector of repairs' weights
  - *sumRep*: sum of the current repairs weights
  - *sumDes*: sum of the current destroyers weights
  - *deltas*: column vector set in this way
    - \* delta1: the new solution is the best one so far
    - \* delta2: the new solution is better than the current one
    - \* delta3: the new solution is accepted
    - \* delta4: the new solution is rejected
  - *decay*: decay parameters used to update the weights
- *GeneralALNS*: Constructor
  - *INPUTS*:
    - \* *destroySet*: cell array of destroy methods
    - \* *repairSet*: cell array of repair methods
    - \* *deltas*: column vector set in this way

- *delta1*: the new solution is the best one so far
- *delta2*: the new solution is better than the current one
- *delta3*: the new solution is accepted
- *delta4*: the new solution is rejected
- \* *decay*: parameter that considers the previous weights, between 0 and 1
- \* *nIter*: maximum number of iterations
- \* *initialSlr*: Solution object with the starting solution
- \* *initialState*: state object with the info about the initial state
- \* *nRep*: total number of replicas
- *OUTPUTS*:
  - \* *obj*: object with updated weights and sum of weights
- *updateWeights*: function that updates the weights and the sum of the weights. In order to avoid computing every time the sum, the sum of the weights is continuously updated to reduce the number of computations
- *INPUTS*:
  - \* *obj*: GeneralALNS object
  - \* *boolAccept*: boolean value that expresses if temporary solution has been accepted
  - \* *boolCurr*: boolean value that expresses if temporary solution is better than the current solution
  - \* *desIndx*: index of the destroy method used
  - \* *repIndx*: index of the repair method used
- *OUTPUTS*:
  - \* *obj*: object with updated weights and sum of weights
- *getProbability*: function used to compute the probability associated to every operator
- *INPUTS*:
  - \* *obj*: GeneralALNS object
- *OUTPUTS*:
  - \* *probD*: probability column vector of destroyers
  - \* *probR*: probability column vector of repairs
- *extract*: function used to extract the destroy and repair methods
- *INPUTS*:
  - \* *obj*: GeneralALNS object
- *OUTPUTS*:
  - \* *destroyIndx*: index of the extracted destroy
  - \* *repairIndx*: index of the extracted repair
- *Schedule*: Scheduling function ALNS algorithm
- *INPUTS*:
  - \* *obj*: GeneralALNS object
  - \* *seed*: optional parameter random seed for replicability
- *OUTPUTS*:
  - \* *obj*: object with the scheduling done
- *restore*: function used to restore the ALNS object for a new replica
- *INPUTS*:

- \* *initialSlt*: solution object that represents the initial solution
- *OUTPUTS*:
  - \* *obj*: the restored object
- *createPlot*: function used to create the plots
  - *INPUTS*:
    - \* *obj*: GeneralALNS object
    - \* *cellX*: cell array to print in the x axis plot
    - \* *cellY*: cell array to print in the x axis plot
    - \* *plotTitle*: string with the plot title
    - \* *savePath*: path to the folder without the name of the file, the name will be the plot title
- *tableConstruction*: function that builds and returns the result table
  - *INPUTS*:
    - \* *obj*: GeneralALNS object
  - *OUTPUTS*:
    - \* *outputCell*: cell array with the tables
- *writeFile*: function used to write the table results on file
  - *INPUTS*:
    - \* *obj*: GeneralALNS object
    - \* *outputCell*: cell array with the tables
    - \* *nameTxt*: name with the path of the txt file where to save
- *accept*: accept method that decides if the current solution needs to be updated.
  - *INPUTS*:
    - \* *obj*: initial object
    - \* *newSlt*: new solution to accept
  - *OUTPUTS*:
    - \* *acceptBool*: boolean that expresses if the solution is accepted
    - \* *obj*: object modification of the temperature
- *stoppingCriteria*: stopping criteria computation
  - *INPUTS*:
    - \* *obj*: initial object
    - \* *currIter*: current iteration
  - *OUTPUTS*:
    - \* *stop*: 1 if it needs to stop, 0 otherwise
- *destroyPolicy*: function used to apply the destroy policy
  - *INPUTS*:
    - \* *obj*: initial object
  - *OUTPUTS*:
    - \* *obj*: updated object with new destroy update

### 1.7.2 StopNIter

Class that must be used as a mix-in with the GeneralALNS object. It gives the stopping criteria and the fraction computing for the destroy policy. *fractionComputing*: function used to compute the fraction of increasing for the increasing policy method

- *INPUTS*:
  - *obj*: initial object
  - *countIter*: number of the current iteration
- *OUTPUTS*:
  - *fraction*: fraction of increasing for the increasing policy method

### 1.7.3 Acceptance

In this chapter are presented some acceptance criteria classes that have to be combined with a stopping criteria and the destroy policy. Every acceptance criteria has the accept method cited before.

- *restoreAccept*: function that restores the acceptance parameters
  - *OUTPUTS*:
    - \* *obj*: the updated object

### 1.7.4 AcceptSA

Class that implements the Simulation Annealing acceptance criteria.

- Properties:
  - *T0*: initial Temperature
  - *alpha*: decay parameter of the temperature
  - *T*: current value of the temperature
- *AcceptSA*: Constructor.
  - *INPUTS*:
    - \* *T0*: initial temperature
    - \* *alpha*: decay temperature parameter
  - *OUTPUTS*:
    - \* *obj*: initialized object

### 1.7.5 AcceptGreedy

Class that implements the greedy acceptance criteria.

### 1.7.6 Destroy policy

In this chapter the classes used to address the destroy policy are presented. Every destroy policy class has the destroyPolicy method cited before and the following method. *restoreDestroy*: function used to restore the destroy degree.

- *OUTPUTS*:
  - *obj*: updated object

### 1.7.7 desFixedPol

Class that must be used as a mix-in with the GeneralALNS object. Implementing destroy fixed policy.

### 1.7.8 desIncreasesPol

Class that must be used as a mix-in with the GeneralALNS object. Implementing destroy increasing policy. Properties:

- Properties:
  - *degDes0*: vector of initial destroy degrees
- *desIncreasesPol*: constructor
  - *OUTPUTS*:
    - \* *obj*: constructed object

### 1.7.9 desRandomPol

Class that must be used as a mix-in with the GeneralALNS object. Implementing destroy random policy.

- Properties:
  - *degDes0*: vector of initial destroy degrees
- *desRandomPol*: constructor
  - *OUTPUTS*:
    - \* *obj*: constructed object

### 1.7.10 composed optimizer

Every optimizer that can be seen in the figure above is a mix-in of the previously discussed classes. The name suggests the mix-in:

- *ALNS*: mix-in with GeneralALNS
- *Gr*: mix-in with AcceptGreedy
- *SA*: mix-in with AcceptSA
- *I*: mix-in with StopNIter
- *dF*: mix-in with desFixedPol
- *dR*: mix-in with desRandomPol
- *dI*: mix-in with desIncreasesPol

Every constructor calls first of all the same parameters of the GeneralALNS constructor, then it calls the AcceptSA parameters if there is SA in the name, nothing else otherwise.

## 1.8 OtherFiles

### 1.8.1 Relatedness

Class used to implement the relatedness measure between satellites.

- *relatednessMeasure*: Creation of relatedness matrix between set K and set J. consider that the order is important, since there is not simmetry in the relatedness measure.
- *INPUTS*:
  - \* *obj*: the specific object used
  - \* *kIndx*: vectors of target index k in K to calculate  $R(k,j)$
  - \* *jIndx*: vectors of target index j in J to calculate  $R(k,j)$
  - \* *targets*: vectors of targets to get all the information needed
  - \* *seq*: total sequence solution
  - \* *beta*: parameters used to weight phasing and planar change
- *OUTPUTS*:
  - \* *R*: related matrix  $R(k,j)$ , for every k in kIndx and j in jIndx
- *obtainVector*: vector used to obtain vector of specific dimension to compute the relatedness matrix
- *INPUTS*:
  - \* *obj*: the considered object
  - \* *kIndx*: vectors of target index k in K to calculate  $R(k,j)$
  - \* *jIndx*: vectors of target index j in J to calculate  $R(k,j)$
  - \* *targets*: vectors of targets to get all the information needed
- *OUTPUTS*:
  - \* *ik*: column vector of inclination of targets from kIndx
  - \* *ok*: column vector of raan of targets from kIndx
  - \* *nuk*: column vector of true anomaly of targets from kIndx
  - \* *ij*: row vector of inclination of targets from jIndx
  - \* *oj*: row vector of raan of targets from jIndx
  - \* *nuj*: row vector of true anomaly of targets from jIndx
- *cMatrix*: compute the not normalized cost matrix  $l_k \times l_j$
- *INPUTS*:
  - \* *ik*: column vector of inclination of targets from kIndx
  - \* *ok*: column vector of raan of targets from kIndx
  - \* *nuk*: column vector of true anomaly of targets from kIndx
  - \* *ij*: row vector of inclination of targets from jIndx
  - \* *oj*: row vector of raan of targets from jIndx
  - \* *nuj*: row vector of true anomaly of targets from jIndx
  - \* *beta*: parameters used to weight phasing and planar change
- *OUTPUTS*:
  - \* *c*: not normalized cost matrix  $l_k \times l_j$
- *cPrimeMatrix*: method used to normalize the cost matrix
- *INPUTS*:
  - \* *obj*: object used
  - \* *c*: not normalized cost matrix
- *OUTPUTS*:
  - \* *cPrime*: normalized cost matrix
- *vMatrix*: Construction of similarity V matrix

- *INPUTS*:
  - \* *obj*: the considered object
  - \* *kIdx*: vectors of target index k in K to calculate R(k,j)
  - \* *jIdx*: vectors of target index j in J to calculate R(k,j)
  - \* *seq*: total sequence
- *OUTPUTS*:
  - \* *V*: similarity V matrix

### 1.8.2 initialSeq

- *FUNCTION*: Function that creates a feasible solution for the assumptions made in the thesis. Creates missions composed by reaching one single target for all targets
- *INPUTS*:
  - *nTar*: number of targets
  - *nSSc*: number of sscs
- *OUTPUTS*:
  - *seq*: initial sequence

## 1.9 Test

### 1.9.1 checkInstance

- *FUNCTION*: Checks whether an instance is acceptable. It performs the following checks:
  - $0 \leq i < 180$
  - $0 \leq o < 360$
  - all couples (i,o) are unique
  - for every possible couple, the following condition is **not** satisfied:  $(i_1 + i_2) = 180$  and  $|o_1 - o_2| = 180$
- *INPUTS*:
  - *i*: vector of inclinations
  - *o*: vector of RAANs
- *OUTPUTS*:
  - *infeas*: 1 if infeasible, 0 if feasible

### 1.9.2 createInstanceProblem

Function that helps to create an instance of the problem by building the initial state and initial solution information based on the provided orbital and spacecraft/target data.

- *INPUTS*:
  - *nSSc*, *nTar*: number of SScs and Targets
  - *i\_S*, *i\_T*: inclinations of SScs and Targets
  - *ω\_S*, *ω\_T*: RAANs of SScs and Targets
  - *ν\_S*, *ν\_T*: true anomalies of SScs and Targets



- *dryMass\_S*, *dryMass\_T*: dry masses
- *fuelMass\_S*, *fuelMass\_T*: fuel masses
- *totCap\_S*, *totCap\_T*: total tank capacities
- *specificImpulse\_S*: specific impulse of SScs
- *refillSpeedSSc*, *refillSpeed*: refilling speeds of SScs and Station
- *seq*: initial sequence
- **OUTPUTS:**
  - *initialState*: object containing initial state information
  - *initialSlit*: object containing the initial solution

### 1.9.3 outputTest

function used to test the output functionalities of the object by writing specific information to a file.

- **INPUTS:**
  - *obj*: object whose output method is tested
  - *filename*: name of the file where the output is written
- **OUTPUTS:**
  - None