1. Algorithms’ description

Our code will work based on 3 core algorithms complying to the functional requirements of the specification document.

* 1. FR8 – Determine the quality of air at a given time and location

The goal is to determine the ATMO quality of air at a given time and location, based on the neighbouring’s sensors records. We will proceed by estimating the concentration of the 4 attributes (O3, SO2, NO2 and PM10) at the target location from all measurements made up to 24 hours before of after the target time.

We will then weight these measurements by their relative distance: the closer the sensor is, the more it will count. To proceed, we will take a random sensor and calculate its distance for the target location: it will be our referential and have a weight of 1. The weight of all other sensors will then be inversely proportionate to the distance from the target location: . If the referential is at 100m of the target location, a sensor located at 50m will have a weight of 2, and a sensor at 300m a weight of 1/3.

Finally, we determine the weighted average of measurements for each attribute from our determined weights. We can then apply the ATMO quality of air formula (<https://fr.wikipedia.org/wiki/Indice_de_qualit%C3%A9_de_l%27air>) to return the quality of air of the location at the specified time.

The worst-case complexity of this algorithm is **O(m)** where ‘m’ is the number of measurements.

algorithm FR8\_quality is

    inputs: Period timePeriod, Position askedPosition

    output: String predictedATMOScore

    call: FR8\_quality(Period timePeriod, Position askedPosition)

{

    // returns True if the timestamp is inside the period, False otherwise.

    function isGivenTimeInsideTimePeriod(Pediod period, Time timestamp) -> Boolean isInside

        if period.start <= timestamp and period.end >= timestamp

            return true

        else

            return false

    // returns the distance between 2 positions

    function distanceBetweenPositions(Position a, Position b) -> double distance

    // get a random Sensor, the fastest to get actually

    function getASensor() -> Sensor sensor

    // convert

    function convertValuesAttributesToATMOScore(Map<Attribute, double> values) -> String predictedATMOScore

    // get a list of all measurements

    function getAllMeasurements() -> Measurement[] allMeasurements

    // returns predicted values for ATMO attributes for a given position and a considered period of time for the data

    function FR8\_qualityAttributes(Period timePeriod, Position askedPosition) -> Map<Attribute, double> attributesPredictedValues

        var Attribute[] attributes := getAllAttributes() // an array of all data types (Attibutes)

        var Measurement[] allMeasurements := getAllMeasurements()

        var Map<Attribute, double> numeratorSums := {

            attributes.O3: 0.0,

            attributes.NO3: 0.0,

            attributes.SO2: 0.0,

            attributes.PM10: 0.0

        }

        var Map<Attribute, double> denominatorSums := {

            attributes.O3: 0.0,

            attributes.NO3: 0.0,

            attributes.SO2: 0.0,

            attributes.PM10: 0.0

        }

        var Sensor referentiel := getASensor()

        for each measurement in allMeasurements

            if measurement.getSensor().reliable and isGivenTimeInsideTimePeriod(timePeriod, measurement.timestamp)

                var double coefficient := distanceBetweenPositions(referentiel.position, measurement.getSensor().position)

                numeratorSums[measurement.attribute] := numeratorSums[measurement.attribute] + (coefficient \* measurement.value)

                denominatorSums[measurement.attribute] := denominatorSums[measurement.attribute] + coefficient

        var Map<Attribute, double> attributesPredictedValues := {

            attributes.O3: 0.0,

            attributes.NO3: 0.0,

            attributes.SO2: 0.0,

            attributes.PM10: 0.0

        }

        attributesPredictedValues := numeratorSum/denominatorSums

        return attributesPredictedValues

    // returns a ATMO coefficient for a given position and a considered period of time for the data

    function FR8\_quality(Period timePeriod, Position askedPosition) -> String predictedATMOScore

        var Map<Attribute, double> results := FR8\_qualityAttributes

        return convertValuesAttributesToATMOScore(results)

}

* 1. FR5 – Help determine if a sensor is reliable

The goal is to help determine if a private sensor is reliable or not. This algorithm will use our method designed for FR8 to predict values at a desired location and time based on neighbouring sensors’ data. We will thus create a list of all sensors excepting the one we want to analyse, and for each measurement of the analysed sensor, call the FR8 algorithm for the position of the sensor and the moment of each measurement.

We will then calculate the relative gap between the predicted value and the effective value measured by the sensor, which formula is:

Finally, we will sum these relative gaps for each measurement of the tested sensor and calculate an average relative gap and return it. This average relative gap represents the overall gap of the tested sensor’s measurements compared to the expected value from all other sensors. The government agency can then judge if the sensor will be marked at unreliable or not.

The worst-case complexity of this algorithm is **O(m²)**.

Algorithm FR5\_malfunctioningAnalysis is

    input: Sensor sensorToCheck

    output: Double averageRelativeGap

    call: FR5\_malfunctioningAnalysis(Sensor sensorToCheck)

{

    // get a list of all measurements

    function getAllMeasurements() -> Measurement[] allMeasurements

    // removes all the measurements from a measurement list of a given sensor

    function removeAllMeasurementsFromSensor(Measurement[] measurements, Sensor sensor) -> Measurement[] remainingMeasurements

    function FR5\_malfunctioningAnalysis(Sensor sensorToCheck) -> Boolean isReliable

        var Measurement[] measurements := getAllMeasurements()

        measurements := removeAllMeasurementsFromSensor(measurements, sensorToCheck)

        var Double relativeSum := 0.0

        var Integer nbOfMeasurementsForSensorToCheck := 0

        // for every measurement of the sensor, check if it is close to the expected one or not by adding to relative sum

        for each measurement in sensorToCheck.getMeasurements()

            var Map<Attribute, Double> expectedValues := FR8\_qualityAttributes(ALWAYS, measurement.getSensor().position)

            var Double expectedValue := expectedValues[measurement.attribute]

            var Double relativeDiff := abs(expectedValue - measurement.value) / expectedValue

            relativeSum := relativeSum + relativeDiff

            nbOfMeasurementsForSensorToCheck := nbOfMeasurementsForSensorToCheck + 1

        var Double averageRelativeGap := relativeSum / nbOfMeasurementsForSensorToCheck

        return averageRelativeGap

}

* 1. FR7 – Find the level of similarity between one specified sensor and all the others for a specific period

The goal is to estimate a level of similarity between one specified sensor and all the others for a specific period. A level of similarity of 1 means the two sensors found the same values for the period, while a similarity tending towards 0 means the two sensors had their measurements with a gap close to infinity.

To proceed, we will reuse our principle of relative gap used for the FR5 algorithm: for each Attribute, we will calculate their average value for all sensors and find the relative gap between the target sensor’s average and all other sensors using the previous formula. We will then finally average the relative gaps of all attributes to get a global relative gap between the specified sensor and the others.

Finally, we will have to determine a level of similarity, not a relative gap. It means finding a function for which (a relative gap of zero means the sensors are 100% identical) and . A fitting candidate would be an inverse function matching , thus:

We then store the level of similarity in a map along with its linked sensor and return it to the user.

The worst-case complexity of this algorithm is **O(max(s,m))** where ‘s’ is the number of sensors and ‘m’ the number of measurements.

Algorithm FR7\_sensorComparison is

    inputs: Sensor sensorToCompare, timestamp t1, timestamp t2

    output: Map<Sensor,double> proximity

    Pre-condition: sensorToCompare has measurements during the specified period

    call: FR7\_sensorComparison(Sensor sensorToCompare, timestamp t1, timestamp t2)

{

    function getAllSensors() -> Sensor[] allSensors

    // Calculates the average value of a targetted attribute of a sensor between t1 and t2

    function FR7\_averageValue(Sensor sensor, Attibute targetAttribute, timestamp t1, timestamp t2)

        var Double sum := 0

        var Integer checkedMeasurement :=0

        for each measurement in sensor.measurements

            if measurement.attribute.identifier = targetAttribute.identifier and measurement.timestamp > t1 and measurement.timestamp < t2

                sum += measurement.value

                checkedMeasurement+=1

        return sum/checedMeasurement

    //Calculates the proximity of all sensors compared to the targetted sensor: Calculates the relative gap between the average target value and the other sensors. Returns a map of all sensors with their target's proximity

    function FR7\_sensorComparison(Sensor sensorToCompare, timestamp t1, timestamp t2) {

        var Map<Sensor, double> proximity

        var Double refValues[4]; //Stores the average value of the target sensor

        var Integer i:=0

        for each attribute //NO2, O3, PM10...

            refValues[i] := FR7\_averageValue(sensorToCompare,attribute,t1,t2)

            i+=1

        for each sensor in getAllSensors()

            if sensor!=sensorToCompare

                var Double relative\_gap := 0

                var Integer i := 0

                for each attribute

                    var Double average := FR7\_averageValue(sensor,attribute,t1,t2)

                    relative\_gap += absolute(average-refValues[i])/refValues[i]

                    i+=1

                relative\_gap := relative\_gap/i

                proximity[sensor] = 1/(relative\_gap+1)

        return proximity

    }

}