



## Emerging technologies for the Early location of Entrapped victims under Collapsed Structures & Advanced Wearables for risk assessment and First Responders Safety in SAR operations

### D4.5 Development of SOT DSS components

**Workpackage:** WP4 – Data aggregation, Analysis and Decision Support

<b>Authors:</b>	KT
<b>Status:</b>	Final
<b>Due Date:</b>	31/08/2021
<b>Version:</b>	1.00
<b>Submission Date:</b>	27/08/2021
<b>Dissemination Level:</b>	PU

#### Disclaimer:

This document is issued within the frame and for the purpose of the Search and Rescue project. This project has received funding from the European Union's Horizon2020 Framework Programme under Grant Agreement No. 882897. The opinions expressed and arguments employed herein do not necessarily reflect the official views of the European Commission.

This document and its content are the property of the Search and Rescue Consortium. All rights relevant to this document are determined by the applicable laws. Access to this document does not grant any right or license on the document or its contents. This document or its contents are not to be used or treated in any manner inconsistent with the rights or interests of the Search and Rescue Consortium or the Partners detriment and are not to be disclosed externally without prior written consent from the Search and Rescue Partners. Each Search and Rescue Partner may use this document in conformity with the Search and Rescue Consortium Grant Agreement provisions.










(\*) Dissemination level.-PU: Public, fully open, e.g. web; CO: Confidential, restricted under conditions set out in Model Grant Agreement; CI: Classified, Int = Internal Working Document, information as referred to in Commission Decision 2001/844/EC.












## Search and Rescue Project Profile

**Grant Agreement No.:** 882897

<b>Acronym:</b>	Search and Rescue
<b>Title:</b>	Emerging technologies for the Early location of Entrapped victims under Collapsed Structures & Advanced Wearables for risk assessment and First Responders Safety in SAR operations
<b>URL:</b>	<a href="https://search-and-rescue.eu/">https://search-and-rescue.eu/</a>
<b>Start Date:</b>	01/07/2020
<b>Duration:</b>	36 months

### Partners

	NATIONAL TECHNICAL UNIVERSITY OF ATHENS (NTUA) <u>Co-ordinator</u>	Greece
	AIDEAS OÜ (AIDEAS)	Estonia
	SOFTWARE IMAGINATION & VISION S.R.L (SIMAVI)	Romania
	MAGGIOLI SPA (MAG)	Italy
	KONNEKT-ABLE TECHNOLOGIES LIMITED (KT)	Ireland
	THALES ITAIA Italia SPA (THALIT)	Italy
	ATOS IT SOLUTIONS AND SERVICES IBERIA SL (ATOS)	Spain
	ETHNIKO KENTRO EREVNAS KAI TECHNOLOGIKIS ANAPTYXIS (CERTH)	Greece
	UNIVERSITA DEGLI STUDI DI CAGLAIRI (UNICA)	Italy

	UKEMED GLOBAL LTD (UGL)	Cyprus
	PUBLIC SAFETY COMMUNICATION EUROPE FORUM AISBL (PSCE)	Belgium
 UNIVERSITÀ DEGLI STUDI FIRENZE	UNIVERSITA DEGLI STUDI DI FIRENZE (UNIFI)	Italy
	DEUTSCHES FORSCHUNGSZENTRUM FÜR KUNSTLICHE INTELLIGENZ (DFKI)	Germany
	UNIVERSITA CATTOLICA DEL SACRO CUORE (UCSC)	Italy
 VRIJE UNIVERSITEIT BRUSSEL	VRIJE UNIVERSITEIT BRUSSEL	Belgium
	SYNYO GmbH (SYNYO)	Austria
	UNIVERSITEIT HASSELT (UHASSELT)	Belgium
 SPOLECZNA AKADEMIA NAUK UNIVERSITY OF SOCIAL SCIENCES	SPOLECZNA AKADEMIA NAUK (SAN)	Poland
	GIOUMPITEK MELETI SCHEDIASMOS YLOPOIISI KAI POLISI ERGON PLIROFORIKIS ETAIREIA PERIORISMENIS EFTHYNIS (UBITECH)	Greece
<b>Search and Rescue End-Users</b>		
	ELLINIKI OMADA DIASOSIS SOMATEIO (HRT)	Greece

	<p>ENOSI PTYCHIOYCHON AXIOMATIKON YPAXIOOMATIKON PYROSVESTIR OY SOMATEIO (EPAYPS)</p>	<p>Greece</p>
<p><b>DIE JOHANNITER</b> Aus Liebe zum Leben</p> 	<p>JOHANNITER-UNFALL-HILFE EV (JOHAN)</p>	<p>Germany</p>
<p><b>DIE JOHANNITER</b> Aus Liebe zum Leben</p> 	<p>JOHANNITER OSTERREICH AUSBLIDUNG UND FORSCHUNG GEMEINNUTZIGE GMBH (JOAFG)</p>	<p>Austria</p>
 <p>Consiglio Nazionale delle Ricerche</p>	<p>CONSIGLIO NAZIONALE DELLE RICERCHE</p>	<p>Italy</p>
	<p>POMPIERS DE L'URGENCE INTERNATIONALE (PUI)</p>	<p>France</p>
	<p>ASOCIATA CLUSTERUL ROAMN RENTRU PROTECTIE SI ECOLOGIE IN DOMENIUL MATERIALELOR CHIMICE, BIOLOGICE, RADIOLOGICE/NUCLEARE SI EXPLOZIVE (PROECO)</p>	<p>Romania</p>
	<p>SERVICIO MADRILENO DE SALUD (SERMAS)</p>	<p>Spain</p>
 <p>FIIBAP FUNDACIÓN PARA LA INVESTIGACIÓN E INNOVACIÓN BIOSANITARIA DE ATENCIÓN PRIMARIA Servicio Madrileño de Salud</p>	<p>FUNDACIÓN PARA LA INVESTIGACIÓN E INNOVACIÓN BIOSANITARIA DE ATENCIÓN PRIMARIA (FIIBAP)</p>	<p>Spain</p>
	<p>ESCUELA ESPANOLA DE SALVAMENTO Y DETECCION CON PERROS (ESDP)</p>	<p>Spain</p>

## Document History

Version	Date	Author (Partner)	Remarks/Changes
0.1	01/07/2021	Christos Angelidis (KT)	Table of Contents
0.2	15/07/2021	Giorgos Mimoglou (KT)	Initial version
0.3	20/07/2021	Christos Angelidis, Giorgos Mimoglou, Kalliopi Angelaki (KT)	Section Editor
0.4	30/07/2021	George Mimoglou (KT), Christos Angelidis (KT)	Overall Editor
0.5	03/08/2021	Simona Panunzi (CNR)	Review 1 (Technical)
0.6	03/08/2021	Giulia Sedda (UNICA)	Review 2 (Quality)
0.61	25/08/2021	Christodoulos Santorinaios (NTUA)	Quality Control
1.0	27/08/2021	Christos Ntanos (NTUA)	FINAL VERSION TO BE SUBMITTED

## **Executive Summary**

The current document reports the development of the mechanism responsible for assisting the end user in decision support by providing a set of recommendations based on the current status of the system under study and by implementing a SOT DSS engine. The stream, historical and legacy data, originating from sensors, web services and other relational databases, constitute the main source of information and inputs for the Decision Support Systems in S&R. The SOT DSS Engine uses Linear Programming techniques and produces recommendations by combining all the above types of data.

D4.5 "Development of SOT DSS components" follows the design of D4.3 "Design of DSS components" in order to develop an efficient Decision Support System.

In the second version of this deliverable the technical implementation details of the extensions of the SOT DSS will be presented. Service 1 to 4 will be improved and be adapted to the ongoing changes of the needs of the S&R project.

## Table of Contents

<b>List of Abbreviations .....</b>	<b>9</b>
<b>List of Figures .....</b>	<b>10</b>
<b>List of Tables.....</b>	<b>12</b>
<b>1 Introduction.....</b>	<b>13</b>
<b>1.1 Relationship with other Deliverables .....</b>	<b>13</b>
<b>2 Development Analysis of the SOT DSS Components.....</b>	<b>15</b>
<b>2.1 Technical Approach of the SOT DSS Components.....</b>	<b>15</b>
<b>2.2 The architecture analysis of the SOT DSS Components.....</b>	<b>15</b>
2.2.1 Web API.....	15
2.2.2 SOT DSS components .....	18
<b>3 Proposed techniques for the SOT DSS Modules .....</b>	<b>21</b>
<b>3.1 SOT DSS Modules .....</b>	<b>21</b>
3.1.1 Transportation Problem .....	21
3.1.2 UTA package .....	23
3.1.3 EMSAllocation package - Service 1 .....	23
3.1.4 PatientAllocation package - Service 2 .....	24
3.1.5 TaskAllocation package - Service 3.....	26
3.1.6 EarthquakeCasualtyEstimation - Service 4 .....	27
<b>3.2 Demonstrator .....</b>	<b>28</b>
3.2.1 Service 1 .....	29
3.2.1.1 Example 1 - No excess demand.....	29
3.2.1.2 Example 2 - Equal total demand and total supply .....	30
3.2.1.3 Example 3 - Excess demand.....	32
3.2.1.4 Example 4 - Excess demand results to demand less than one .....	34
3.2.2 Service 2 .....	36
3.2.2.1 Example 1 – Skaramagkas + Patient cost formula .....	36
3.2.2.2 Example 2 – Skaramagkas + UTA .....	38
3.2.2.3 Example 3 – Monastiraki + Patient cost formula .....	39
3.2.2.4 Example 4 – Monastiraki + UTA .....	40
3.2.3 Service 3 .....	42
3.2.3.1 Example 1 – Skaramagkas + Task Cost Formula + Excess demand .....	42
3.2.3.2 Example 2 – Skaramagkas + UTA + excess demand .....	44
3.2.3.3 Example 3 – Monastiraki + Task Cost Formula .....	45
3.2.3.4 Example 4 – Monastiraki + UTA .....	47
3.2.4 Service 4 .....	48
3.2.4.1 Example 1 – onePAGER exists.....	48

3.2.4.2 Example 2 – onePAGER exists ..... 49

3.2.4.3 Example 3 – onePAGER does not exist..... 50

**4 Conclusions .....52**

**Annex I: References .....52**



## List of Abbreviations

Abbreviation	Explanation
API	Application Programming Interface
CBC	Coin-or branch and cut
D	Deliverable
DSS	Decision Support System
EMS	Emergency Medical Services
HTTP	Hypertext Transfer Protocol
LP	Linear Programming
ML	Machine Learning
SOT	Strategic Operational and Tactical
S&R	Search & Rescue
UI	User Interface
UUID	Universally Unique Identifier
UTA	UTilités Additives

## List of Figures

---

Figure 2-1: Web API between COncORDE and SOT DSS communication.....	16
Figure 2-2: JSON sample for service 1.....	16
Figure 2-3: Results of API call of service 1.....	17
Figure 2-4: Swagger UI.....	17
Figure 2-5: EMS Allocation endpoint via Swagger UI.....	18
Figure 2-6: SOT DSS & environment abstract architecture.....	19
Figure 2-7: SOT DSS UML package diagram.....	19
Figure 2-8: File structure of the SOT DSS components.....	20
Figure 3-1: Supplier and Demander models in TransportationPorblem package.....	21
Figure 3-2: TransportationProblem constructor.....	22
Figure 3-3: Total demand reduction algorithm.....	22
Figure 3-4: Solution model in UTA package.....	23
Figure 3-5: UTA constructor.....	23
Figure 3-6: EMSStation and Incident models in EMSAllocation package.....	24
Figure 3-7: EMSAllocation constructor.....	24
Figure 3-8: Patient and EMSUnit models in PatientAllocation package.....	25
Figure 3-9: PatientAllocation constructor.....	26
Figure 3-10: Actor and Task models in TaskAllocation package.....	26
Figure 3-11: TaskAllocation constructor.....	27
Figure 3-12: EarthquakeIncident model in EarthquakeCasualtyEstimation package.....	28
Figure 3-13: EarthquakeCasualtyEstimation constructor.....	28
Figure 3-14: EMS Station data.....	29
Figure 3-15: Incident data for example 1.....	29
Figure 3-16: Recommended allocation for example 1.....	30
Figure 3-17: Map Illustration for example 1.....	30
Figure 3-18: Incident data for example 2.....	31
Figure 3-19: Recommended allocation for example 2.....	32
Figure 3-20: Map Illustration for example 2.....	32
Figure 3-21: Incident data for example 3.....	33
Figure 3-22: Recommended allocation for example 3.....	33
Figure 3-23: Map Illustration for example 3.....	34
Figure 3-24: Incident data for example 4.....	35
Figure 3-25: Recommended allocation for example 4.....	36
Figure 3-26: Patient and EMS unit data for example 1 & 2.....	36
Figure 3-27: Recommended allocation for example 1.....	37

Figure 3-28: Map illustration for example 1.....	37
Figure 3-29: Recommended allocation of example 2.....	38
Figure 3-30: Map illustration of example 2.....	38
Figure 3-31: Patient and EMS unit data for examples 3 & 4.....	39
Figure 3-32: Recommended allocation for example 3.....	40
Figure 3-33: Map illustration for example 3.....	40
Figure 3-34: Recommended allocation for example 4.....	41
Figure 3-35: Map illustration for example 4.....	41
Figure 3-36: Role tree for demonstration.....	42
Figure 3-37: Actor and Task data for examples 1 & 2.....	42
Figure 3-38: Recommended allocation for example 1.....	43
Figure 3-39: Map illustration for example 1.....	44
Figure 3-40: Recommended allocation for example 2.....	44
Figure 3-41: Map illustration for example 2.....	45
Figure 3-42: Actor and Task data for example 3 & 4.....	46
Figure 3-43 Recommended allocation for example 3.....	46
Figure 3-44: Map illustration for example 3.....	47
Figure 3-45: Recommended allocation for example 4.....	47
Figure 3-46: Map illustration for example 4.....	48
Figure 3-47: Input and output for example 1.....	48
Figure 3-48: one PAGER for example 1.....	49
Figure 3-49: Input and output for example 2.....	49
Figure 3-50: onePAGER for example 2.....	50
Figure 3-51: Input and output for example 3.....	50

## List of Tables

---

Table 2-1: Service - Package mapping.....	20
Table 3-1: Demand reduction. ....	33
Table 3-2: Detailed demand reduction.....	34
Table 3-3: Demand reduction, first step. ....	35
Table 3-4: Demand reduction, second step.....	35
Table 3-5: Demand reduction, third step. ....	35

# 1 Introduction

---

The deliverable presents the technical approach of the SOT DSS components. More particularly, the document describes the technical implementation details of the four Services of SOT DSS and the tools that have been used. The overall architecture of the four Services is described, as well as an analytical description of the modules is presented. These modules were created for the needs of the implementation of SOT DSS component. SOT DSS modules are the following:

- TransportationProblem
- UTA
- EMSAllocation
- PatientAllocation
- TasksAllocation
- EarthquakeCasualtyEstimation

The technical structure of these modules, and the relations between them will be described in the next chapters. Also, non-ML solutions were used as alternative solutions for Service 4 which are targeted to the needs of the S&R project.

Last but not least, a demonstrator is presented for each of the Services using dummy data in order to have an overview of the possible results of SOT DSS. Finally, an analytical technical overview of the non-ML solutions will be presented.

## 1.1 Relationship with other Deliverables

The current deliverable is addressing the development of the SOT DSS components. The document provides inputs to other S&R components, therefore is linked to the following deliverables:

- D3.1 Requirements to knowledge management and SA Model
- D3.2 Situation Awareness Model-specifications
- D3.6 Multi sensors data fusion and Object detection algorithms for in-disaster scene situation awareness
- D3.7 Requirements to knowledge management and SA Model V2
- D3.8 Situation Awareness Model-specifications V2
- D4.2 Situational Analysis & Impact Assessment
- D4.3 Design of SOT DSS components
- D4.6 Development of PHYSIO DSS component
- D4.7 DSS Validation
- D4.10 Design of PHYSIO DSS component V2
- D4.11 Development of DSS component V2
- D4.12 Development of PHYSIO DSS component V2
- D4.13 DSS Validation V2
- D6.6 Report on legacy systems and their connection to the S&R related technical characteristics
- D6.9 Report on legacy systems and their connection to the S&R related technical characteristics V2
- D7.2 Architecture and Design Specifications of S&R platform

- D7.3 Component interface specifications for interoperability within S&R
- D7.4 Adapted S&R components and services

This document is highly related to D4.3 - 'Design of SOT DSS components', since it contains a detailed description of the implemented services. For this reason, it is strongly recommended to the readers to be familiar with the D4.3.

## 2 Development Analysis of the SOT DSS Components

---

The described SOT DSS components below have been developed to a minimum viable state, to execute their job correctly utilizing the current information about the integration, data format and data type. In the final version, D4.11 – ‘Development of SOT DSS components, V2’, these components will be extended and enriched.

SOT DSS Services are:

- **Service 1:** Recommended (optimal) allocation of available EMS units to incidents, depending on estimated needs
- **Service 2:** Recommended (optimal) allocation of patients to transport vehicles and first receivers (hospitals), based on given order of evacuation and triage results for present injuries
- **Service 3:** Recommended (optimal) allocation of tasks to available actors on the field, given demand pre-defined by the field commander
- **Service 4:** Estimation of expected casualties and demanded resources (EMS units), given historical data on emergency incident recordings

These services will be provided as endpoints of a web service, whereas their functionality is split in packages, described in 2.2.2.

### 2.1 Technical Approach of the SOT DSS Components

The 1st version of the SOT DSS components is developed in **Python v3.8**. Python provides a developer-friendly and analyst-friendly environment in order to implement and test the optimization models. Furthermore, there is a plethora of Python libraries which provide APIs for external services to support the functionality of the services. The main library used for optimization problems is **pyomo v5.7** which provides the right tools for symbolic modelling of linear programming problems.

Furthermore, **Flask v2.0** was used to establish the web service of the SOT DSS and its endpoints. **Jupyter Notebooks** were created for demonstration purposes using dummy data as input for the algorithms. The application has been containerized with **Docker** in order to be deployed independently on the server. The tools that are involved are Docker and docker-compose. Last but not least, the SOT DSS modules can be found at S&R Gitlab group [1].

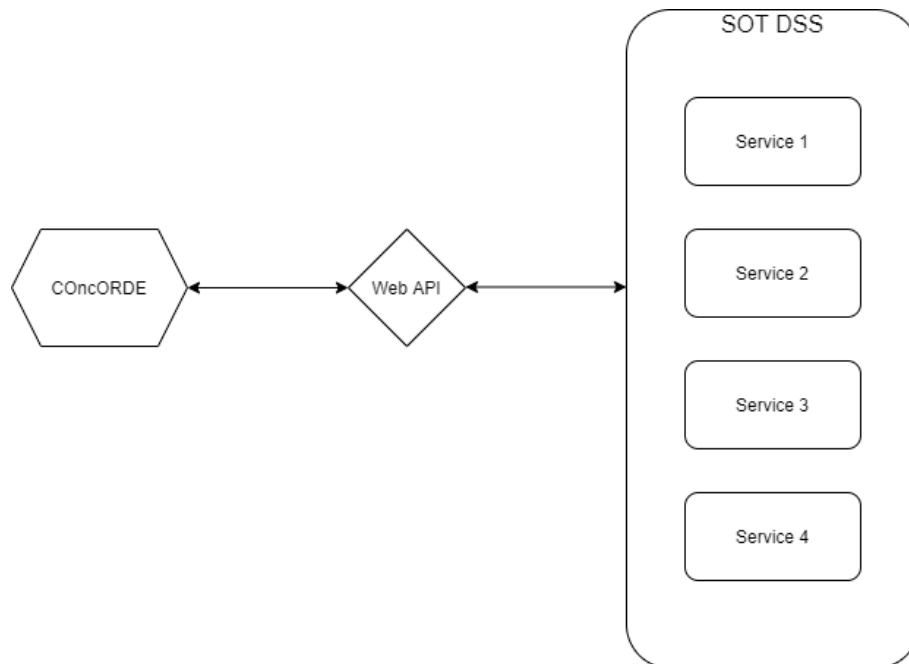
### 2.2 The architecture analysis of the SOT DSS Components

The SOT DSS is considered a web service that exposes its services as endpoints. Thus, a decoupled architecture is achieved, allowing multiple applications to use the SOT DSS when those provide the required input. In the current S&R architecture, the only application that communicates with the SOT DSS is COncORDE, since the latter provides the required data for the services, as described in D4.3- ‘Design of SOT DSS component’. In the future, more S&R components will be integrated. Still this approach creates an extendable environment.

#### 2.2.1 Web API

As a web service, SOT DSS needs a web API to let other applications use its services, so a RESTful API will be developed. The RESTful API services are implemented in Flask, a lightweight and flexible python microframework, in separate endpoints. Before accessing the Services, the user must spin-up the

docker container in order to build the container and run the application. This can be done by typing the command *"docker-compose up -build"*.



**Figure 2-1: Web API between COncORDE and SOT DSS communication**

In order to call and test the functionality of those services the user has to provide a request body with the required data and call the endpoint from a client. For example, the request for Service 1 is POST: **Error! Hyperlink reference not valid.** with body:

```
{
  "EMStationList": [
    {
      "id": 1,
      "location": [37.93341, 23.64579],
      "fleet_size": 6
    },
    {
      "id": 2,
      "location": [37.92944, 23.64379],
      "fleet_size": 3
    },
    {
      "id": 3,
      "location": [37.94267, 23.69757],
      "fleet_size": 4
    }
  ],
  "IncidentList": [
    {
      "id": 1,
      "location": [37.94168, 23.65283],
      "demand": 3
    },
    {
      "id": 2,
      "location": [37.94044, 23.69132],
      "demand": 9
    }
  ]
}
```

**Figure 2-2: JSON sample for service 1.**

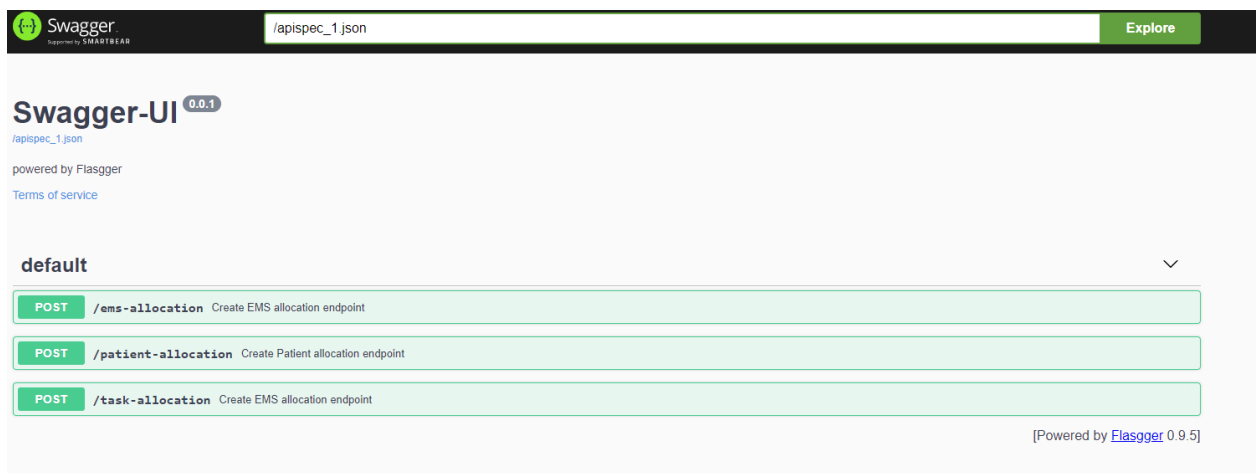


The result of the API call is:

```
{
  "Allocation": {
    "1": {
      "1": 3.0,
      "2": 3.0
    },
    "2": {
      "2": 2.0
    },
    "3": {
      "2": 4.0
    }
  },
  "ReducedDemand": false
}
```

**Figure 2-3: Results of API call of service 1.**

Also, a Swagger UI was created, documenting the endpoints of the implemented services. The Swagger UI is accessed by **Error! Hyperlink reference not valid..** In the second version, swagger will provide complete documentation for all endpoints-services.



**Figure 2-4: Swagger UI**

POST /ems-allocation Create EMS allocation endpoint

EMS Allocation

Parameters Cancel

Name	Description
<b>body</b> * required	Edit Value   Model

object (body)

```
{
  "EMSStationList": [
    {
      "fleet_size": 6,
      "id": 1,
      "location": [
        37.93341,
        23.64579
      ]
    },
    {
      "fleet_size": 3,
      "id": 2,
      "location": [
        37.92944,
        23.64379
      ]
    },
    {
      "fleet_size": 4,
      "id": 3,
      "location": [
        37.94269,
        23.69757
      ]
    }
  ]
}
```

Parameter content type: application/json

Execute Clear

Responses Response content type: application/json

Curl

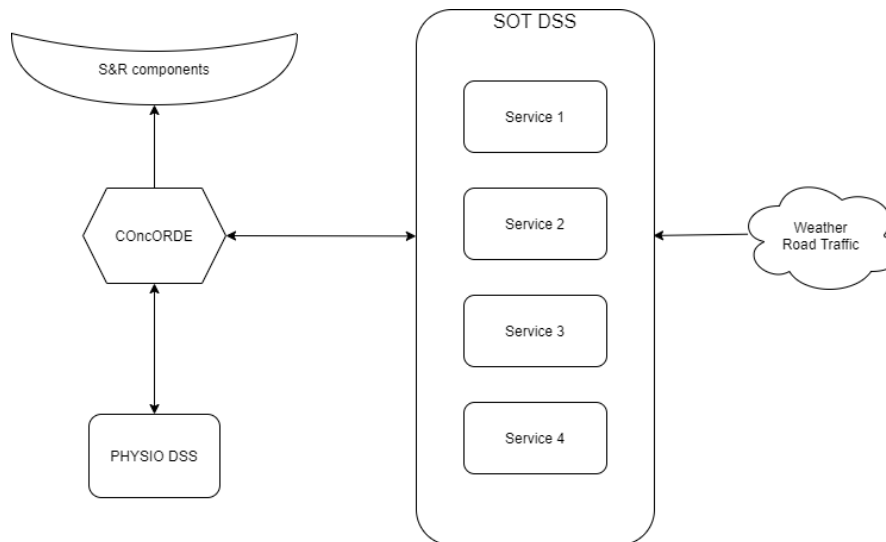
```
curl -X POST "http://192.168.1.139:5000/ems-allocation" -H "accept: application/json" -H "Content-type: application/json" -d '{"EMSStationList": [{"fleet_size": 6, "id": 1, "location": [37.93341, 23.64579 ]}, {"fleet_size": 3, "id": 2, "location": [37.92944, 23.64379 ]}, {"fleet_size": 4, "id": 3, "location": [37.94269, 23.69757 ]}, {"location": [37.97768, 23.74722 ]}, {"fleet_size": 7, "id": 5, "location": [37.98011, 23.79538 ]}, {"fleet_size": 5, "id": 6, "location": [38.0183, 23.66615 ]}], "incidentList": [{"demand": 5, "id": 1, "location": [37.94168, 23.65283 ]}]}'
```

**Figure 2-5: EMS Allocation endpoint via Swagger UI.**

Currently, the SOT DSS services assume that the IDs of the input, e.g., EMS units, Patients, etc., are integers. Many applications use UUIDs that cannot be treated as integers immediately, so a mapping procedure must be developed. It has to be mentioned that the examples in the following chapters do not use any transformation since the IDs are integers. In D4.11 – ‘Development of SOT DSS components V2’, the transformation procedure will be developed.

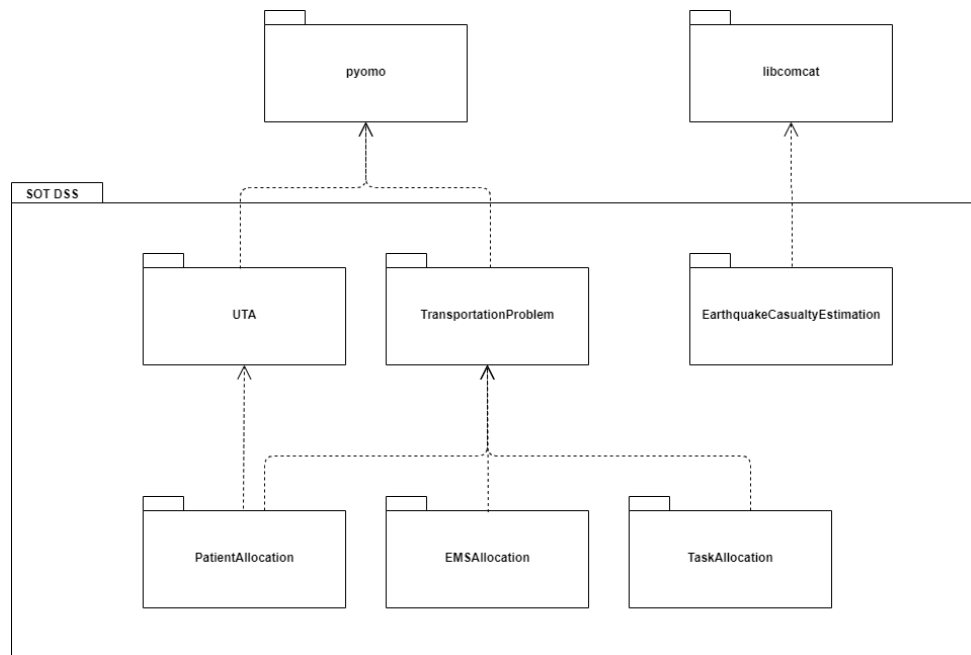
### 2.2.2 SOT DSS components

An abstract architecture of the SOT DSS and its environment has been presented in D4.3, but for the sake of completeness, the image is added in this deliverable too. With few words, CONCORDE sends the input data to the SOT DSS, the latter executes its algorithms and returns the results. Then, CONCORDE displays the results, or it distributes them to other S&R components. PHYSIO DSS and other web services support the SOT DSS, providing essential data. This illustration does not display the inner architecture and modules of SOT DSS.



**Figure 2-6: SOT DSS & environment abstract architecture.**

The following figure is a Unified Modelling Language (UML) package diagram, illustrating the current structure and the dependencies in the SOT DSS.



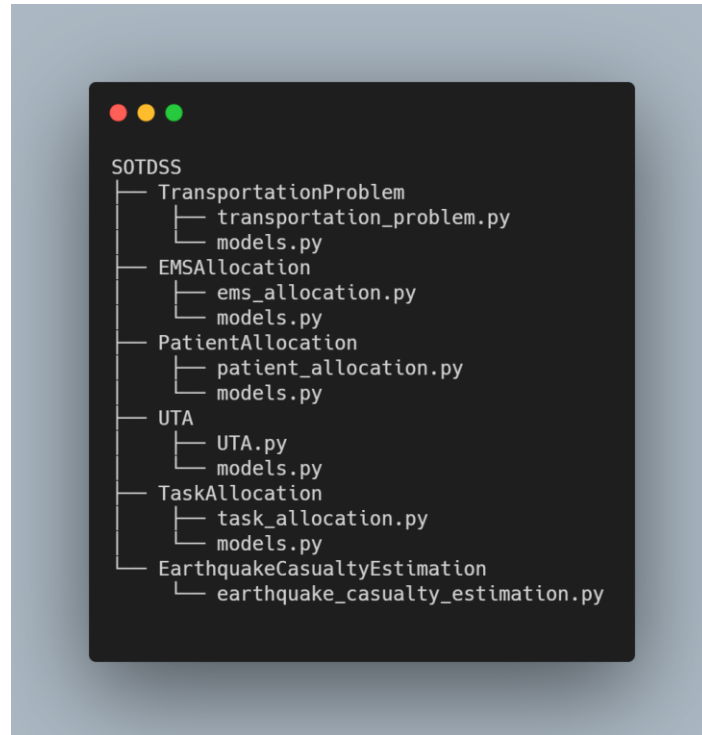
**Figure 2-7: SOT DSS UML package diagram.**

In the following bullets a short description of each package is presented, whereas chapter 3 contains the description of their functionality.

- **Pyomo**, a Python-based open-source software package that supports a diverse set of optimization capabilities for formulating, solving, and analyzing optimization models. [2]
- **TransportationProblem**, a python package which contains modules required by the implementation of the transportation problem of linear programming using pyomo.
- **UTA**, a python package which contains modules required by the implementation of the UTA method using pyomo.
- **EMSAllocation**, a python package which contains modules required by the implementation of the EMS allocation to incidents service.

- **PatientAllocation**, a python package which contains modules required by the implementation of the patient allocation to transport vehicles and first receivers' service.
- **TasksAllocation**, a python package which contains modules required by the implementation of the task allocation to actors' service.
- **Libcomcat**, a project designed to provide a Python equivalent to the ANSS ComCat search API. This includes a Python library that provides various classes and functions wrapping around the ComCat API [3].
- **EarthquakeCasualtyEstimation**, a python package which contains modules required by the implementation of casualty estimation in case of earthquake services.

Likewise, the structure of the packages for this version is:



**Figure 2-8: File structure of the SOT DSS components**

Each package provides a main or utility functionality for the Services of the SOT DSS. Consider the main packages as the packages that are used directly by the endpoints to provide results, while the utility packages as the packages that are used by the main packages to achieve that latter's goal. It must be mentioned that for service 4 only the earthquake event case has been developed in this version. The mapping of services to main packages is:

Service	Main package
<b>Service 1</b>	EMSAllocation
<b>Service 2</b>	PatientAllocation
<b>Service 3</b>	TaskAllocation
<b>Service 4</b>	EarthquakeCasualtyEstimation

**Table 2-1: Service - Package mapping**

## 3 Proposed techniques for the SOT DSS Modules

In D4.3 optimization techniques were proposed for the first three services implementation, whereas a web services utilization approach will solve the last service's complexity problems. In this chapter, the implementation of these services is presented using python code, as well as a short demonstration will highlight their common usage.

Generally, the code for the SOT DSS modules is written in such a way that any extension or modification can be achieved easily. For example, the TransportationProblem package can be used for allocating drones in high severity areas, providing the data required accordingly. Also, the packages follow a consistent architecture, creating a developer-friendly environment.

It is also critical to mention that from this point on, D4.5 has included screenshots with high-level code, in order to describe and demonstrate the work in a more understandable way.

### 3.1 SOT DSS Modules

The usage of each service differs, but they follow similar approaches and techniques. More specifically, the first three Services are modelled by using the transportation problem of linear programming. Many problems can be modelled this way when one defines the proper cost function. For this reason, a transportation problem module is included providing that generic functionality. Finally, the linear model is built using the pyomo library and solved by the CBC solver [4].

#### 3.1.1 Transportation Problem

This package is used by three services with a potential of more services; thus, the generality concept has to be integrated into its architecture and implementation. The package contains two modules, one for the transportation problem functionality and one for the models, i.e., the data classes that the transportation problem class uses.

The basic components of the transportation problem are the demand and supply nodes, and can be defined as:

A screenshot of a code editor window showing Python code for two classes: Supplier and Demander. The code is as follows:

```
class Supplier:
    def __init__(self, id: int, supply: int):
        self.id = id
        self.supply = supply

class Demander:
    def __init__(self, id: int, demand: int):
        self.id = id
        self.demand = demand
```

**Figure 3-1: Supplier and Demander models in TransportationPorblem package.**

The transportation problem consists of a list of Suppliers, a list of Demanders and the cost matrix between them. The cost matrix has to be defined by the user (developer) since the

TransportationProblem class can be used in different cases with different cost functions, for example, services 1 & 2.

Furthermore, the optimization can be set to 'min' for minimization if *costs* is a cost matrix, or to 'max' if *costs* is a profit matrix.

```
class TransportationProblem:
    def __init__(self, suppliers: List[Supplier],
                 demanders: List[Demandeur],
                 costs: np.ndarray,
                 optimization: Literal['min', 'max']='min'):
        ...
        ...
        ...
```

**Figure 3-2: TransportationProblem constructor.**

Also, the TransportationProblem class has private methods required for the linear programming problem declaration using pyomo, such as constraint and objective function declaration. A major function that must be mentioned is the demand reduction on excess demand cases. The reduction described in D4.3, is implemented in python as:

```
percent = self.supply / self.demand
for d in self.demanders:
    new_demand = int(d.demand*percent)
    d.demand = new_demand if new_demand != 0 else new_demand + 1
self.demand = sum(d.demand for d in self.demanders)

# Reduce max demand by one, until there is no excess demand.
while self.demand > self.supply:
    max_demander = max(self.demanders, key=attrgetter('demand'))
    max_demander.demand -= 1
    self.demand -= 1

# Increase min demand by one, until there is no excess supply.
while self.supply > self.demand:
    min_demander = min(self.demanders, key=attrgetter('demand'))
    min_demander.demand += 1
    self.demand += 1
```

**Figure 3-3: Total demand reduction algorithm.**

There are cases that the excess demand cannot be treated by the above algorithm. For example, in Service 2, each demand node (patient) has a unit of demand, so the excess demand has to be treated with alternative methods. Finally, there is the method *solve()* which solves the linear model using the CBC solver. The returned result is a tuple of a dictionary of dictionaries, containing the IDs of the suppliers, the IDs of the demanders and the allocated values (EMS units), as well as a Boolean variable indicating if the demand has been fairly reduced or not, as described in D4.3.

### 3.1.2 UTA package

UTA package is another supportive package for the SOT DSS main services, but it can be used for other multicriteria analysis problems too. Currently, UTA is used for the calculation of the cost matrix in *PatientAllocation – Service 2*, more information will be discussed in the corresponding chapter.

UTA is a multi-criteria decision analysis method which aims at inferring one or more additive value functions for the alternative solutions, from their given ranking. That is, the data model for this package is the *Solution* class:

```
class Solution:
    def __init__(self, id: int, criteria: List[float], rank: int):
        self.id = id
        self.criteria = criteria
        self.num_criteria = len(self.criteria)
        self.rank = rank
        self.utility_function = []
        self.utility_value = 0
```

**Figure 3-4: Solution model in UTA package.**

The UTA class requires a list of *Solution* objects that have the same sequence of criteria. Their ranks must be values of a non-decreasing function, that is  $r: A \rightarrow N_0$  for all  $a_1, a_2 \in A$  such that  $a_1 \leq a_2$  one has  $r(a_1) \leq r(a_2)$ . During object instantiation, the criteria are normalized and the linear model is set, which means that, the sets, parameters, variables, constraints and objective function are declared, using the pyomo library.

```
class UTA:
    def __init__(self, alt_solutions: List[Solution], num_intervals: List[int], e: float = 0.01):
        ...
        ...
        ...
```

**Figure 3-5: UTA constructor.**

Once an UTA object has been instantiated, the *solve* method is available, which solves the linear problem using the CBC solver, and returns the inferred ranking.

### 3.1.3 EMSAllocation package - Service 1

The objective of this package is to provide the functionality to Service 1, i.e., recommendation of optimal allocation of available EMS units to Incidents, depending on estimated needs. The model of this problem is a transportation problem model, so this package utilizes the TransportationProblem package.

First, the data must be in the correct form. For this version, besides the required demand and supply (fleet size) data, the allocation is based on the locations of the nodes. Thus, the models for this problem can be defined as:

```
class EMSStation:
    def __init__(self, id: int, fleet_size: int, location: Tuple[float, float]):
        self.id = id
        self.location = location # (lat, lon)
        self.fleet_size = fleet_size # This is the supply

class Incident:
    def __init__(self, id: int, demand: int, location: Tuple[float, float]):
        self.id = id
        self.location = location # (lat, lon)
        self.demand = demand
```

**Figure 3-6: EMSStation and Incident models in EMSAllocation package.**

The concept of the *EMSAllocation* class is in line with the concept of *TransportationProblem*. So, the *EMSAllocation* requires a list of *EMSStation* objects and a list of *Incident* objects.

```
class EMSAllocation:
    def __init__(self, ems_stations: List[EMSStation], incidents: List[Incident]):
        ...
        ...
        ...
```

**Figure 3-7: EMSAllocation constructor.**

Moreover, *EMSAllocation* class provides the *allocation()* method, which first calculates the cost matrix, then solves the corresponding transportation problem, utilizing the *TransportationProblem* class, and finally returns the resulted allocation-solution, as well as a Boolean variable indicating if the demand has been reduced or not. In this Service, despite the fact that the cases of excess demand are encountered by the *TransportationProblem*, there are extreme cases where the number of incidents is more than the total supply and their demand is 1. The way that the fair demand reduction works does not allow reductions below 1, so random incidents are not considered for the allocation. The concept is made clear in 3.2.1.4 chapter, using an example.

### 3.1.4 PatientAllocation package - Service 2

The consistency of the packages makes the *PatientAllocation* package description similar to the previous package. This package contains two models, *EMSUnit* and *Patient*.



```
class Patient:

    def __init__(self, id: int, physiological_score: int,
                 evacuation_order: int,
                 is_child: Literal[0, 1],
                 location: Tuple[float, float]):

        self.id = id
        self.physiological_score = physiological_score
        self.evacuation_order = evacuation_order
        self.is_child = is_child
        self.location = location

class EMSUnit:

    def __init__(self, id: int, location: Tuple[float, float]):

        self.id = id
        self.location = location
```

**Figure 3-8: Patient and EMSUnit models in PatientAllocation package.**

In this version, *Patient* attributes are:

- ID: The unique identifier
- Physiological Score: A score indicating the physiological state of the patient
- Evacuation Order: The order of evacuation among other patients
- Child: Indicates if the patient is a child
- Location: The location of the patient in the field

Also, an *EMSUnit* consists of:

- ID: The unique identifier
- Location: The location of the EMS unit.

It must be mentioned that the physiological state does not refer to any output of the PHYSIO DSS. The PHYSIO DSS will be utilized by this Service in the second version. Furthermore, in the second version more attributes will be added, based on the provided data, as well as the domain knowledge by the end users. That is, possible extensions are the type of vehicle and type of traumas.

The *PatientAllocation* class requires a list of *EMSUnit* objects and a list of *Patient* objects, as the *uta* and *intervals* arguments are optional and for testing purposes. More specifically, if *uta* is *True*, the cost matrix will be created based on the UTA method, using the *intervals*. Specifically, the patients are modelled as alternative solutions, having a set of criteria (Physiological score, distance from vehicle, child/adult) and a subjective rank (order of evacuation). Then, for each vehicle, the LP is solved to obtain a new ranking of the alternative solutions, as consistent as possible with the subjective one. The intuition behind the values of intervals is that UTA tries to construct an approximation of the utility function for each criterion based on its values. Adding more intervals for a criterion, the approximation is more accurate, but increases the number of variables of the linear program. The default number of intervals came out of experimentation. Otherwise, if *uta* is *False*, the cost matrix is calculated using the *Patient Cost Formula*, proposed in D4.3.

```
class PatientAllocation:

    def __init__(self, ems_units: List[EMSUnit],
                 patients: List[Patient],
                 uta: bool = True,
                 intervals: Tuple[int, int, int] = (5,3,1)):
        ...
        ...
        ...
```

**Figure 3-9: PatientAllocation constructor.**

Finally, the *PatientAllocation* class provides the *allocate* method, which calculates the cost matrix using the selected method, then instantiates the required objects for the *TransportationProblem* and solves the problem. The returned result is a dictionary which has the EMS unit IDs as key and the patients' IDs as values. In case of excess demand, i.e., more patients to transfer than EMS units provided, an item with key -1 and value a list of IDs of the remaining patients is returned too.

### 3.1.5 TaskAllocation package - Service 3

The *TaskAllocation* package constitutes the realization of Service 3 requirements by using the *TransportationProblem* package. The data models for this package are *Actor* and *Task*.

```
class Actor:

    def __init__(self, id: int, role: str, physiological_score: int, location: Tuple[float, float]):
        self.id = id
        self.role = role
        self.physiological_score = physiological_score
        self.location = location

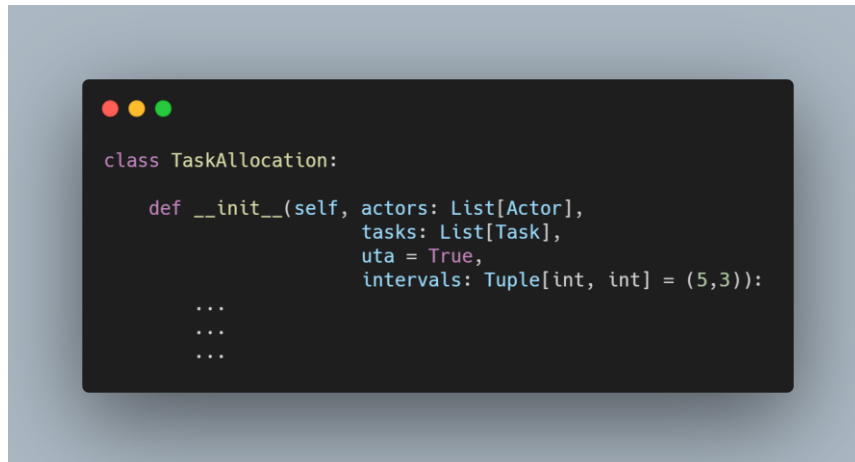
class Task:

    def __init__(self, id: int, role: str, demand: int, location: Tuple[float, float]):
        self.id = id
        self.role = role
        self.demand = demand
        self.location = location
```

**Figure 3-10: Actor and Task models in TaskAllocation package.**

In this version, the role has a string value from a predefined set of roles. The main issue is that the SOT DSS, as a stand-alone web service, must be aware of the role tree of the requested problem. Specifically, if multiple applications with different role trees use the SOT DSS, the role tree must be either pre-defined for that specific application or the application sends it through HTTP with every request. This will be clarified in the second version.

Also, the *TaskAllocation* package contains the *TaskAllocation* class that provides the required functionality. Again, arguments *uta* and *intervals* are used for the UTA method, so one can test both methods of cost matrix calculation.

A screenshot of a code editor window with a dark background and light-colored text. The code defines a Python class named 'TaskAllocation'. The class has an '\_\_init\_\_' method with four parameters: 'self', 'actors: List[Actor]', 'tasks: List[Task]', and 'intervals: Tuple[int, int] = (5,3)'. Inside the method, there is a line 'uta = True,' followed by three lines of ellipses '...' indicating that there are more parameters or attributes in the class definition.

```
class TaskAllocation:
    def __init__(self, actors: List[Actor],
                 tasks: List[Task],
                 uta = True,
                 intervals: Tuple[int, int] = (5,3)):
        ...
        ...
        ...
```

**Figure 3-11: TaskAllocation constructor.**

Ultimately, the *allocate* method solves the transportation problem using the selected cost matrix calculation method and returns the recommended allocation, as well as a Boolean variable indicating if the demand had been reduced or not. The cases of excess demand are treated like Service 2, i.e., a dummy supply node with zero cost is added.

### 3.1.6 EarthquakeCasualtyEstimation - Service 4

The *EarthCasualtyEstimation* package provides functionality to Service 4 in case of an earthquake event type. Due to the nature of such a problem ML technique could not be used. An approach proposed in D4.3 exploiting the PAGER [5] system designed, developed and maintained by the United States Geological Survey (USGS) [6].

PAGER is an automated system that collects and produces information about the impact of earthquakes around the world. The algorithms of the system are capable of assessing the critical impact and provide shaking intensity maps, economic and fatality losses and other useful information. This information is collected into a document called *onePAGER*. This report contains the most critical information that helps the decision makers in disaster response operations. The document is published approximately 30 minutes after the incident if the earthquake magnitude was greater than 5.5 Richter.

The goal of this package is to obtain the onePAGER of the registered incident, as well as to keep it up to date until the end of the operations. For this purpose, the libcomcat [3] python library is used which provides wrapper scripts for accessing Advanced National Seismic System (ANSS) [7] Comprehensive Earthquake Catalog (ComCat) data.

Following the concept of the previous packages, the *EarthquakeCasualtyEstimation* package consists of a *models* module, containing the *EarthquakeIncident* class model.

```
class EarthquakeIncident:
    def __init__(self, location: Tuple[float, float], stime: datetime, etime: datetime):
        self.location = location # (lat, lon)
        self.stime = stime
        self.etime = etime
```

**Figure 3-12: EarthquakeIncident model in EarthquakeCasualtyEstimation package.**

It has to be mentioned that the start and end time in the production environment will be automatically set to real time.

Moreover, there is the *EarthquakeCasualtyEstimation* class which gets an *EarthquakeIncident* as an argument and provides an asynchronous method *estimate*. This method searches ComCat, by using the libcomcat library, for earthquake events around the location in a specified radius (for example 50km) that have the loss PAGER product. Thus, in an earthquake event it may have been registered to the catalog but the product has not been produced, yet. So, the method iteratively searches until the product is ready. If there is no event, the above process sleeps asynchronously for some time and then searches again. Once an event has been found, the related onePAGER URL is obtained and returned to the user.

```
class EarthquakeCasualtyEstimation:
    def __init__(self, incident: EarthquakeIncident):
        self.incident = incident
        self.url = None
```

**Figure 3-13: EarthquakeCasualtyEstimation constructor.**

To sum up, all four Services use different techniques and methods. The transportation problem of Linear Programming is common for the first three Services and the *EarthCasualtyEstimation* package provides functionality to Service 4.

## 3.2 Demonstrator

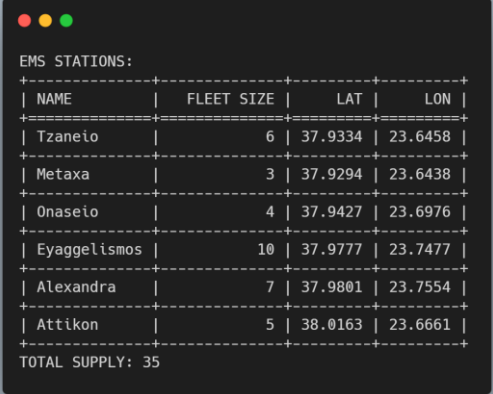
The integration at this time of writing does not allow the SOT DSS to be tested. Also, as the project is in an early stage, the lack of field data has led the SOT DSS to create dummy data in order to demonstrate the usage of the services. It has to be mentioned that the validation of the results will be analyzed in D4.7.

For the sake of this demonstration, Jupyter notebooks have been used, while the data and results are displayed in tabular format and maps created using the QGIS application [8].

### 3.2.1 Service 1

It is important to note that any name and location of EMS stations and incidents are used for reference and intuitive displaying on maps. Fleet size and demand values are dummy and do not correspond to the reality.

Service 1 demonstration is split into three examples, varying on the incidents' demand. Assume the following EMS stations Tzaneio, Metaxa, Onaseio, Eyaggelismos, Alexandra, Attikon, which are hospitals located in Athens, Greece, are the same for every example. Regarding the incidents' areas, PAPEI is University of Piraeus, Omonoia and Skaramagkas are areas in Athens, Greece.



```

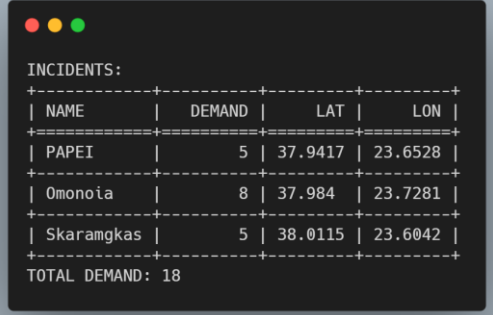
EMS STATIONS:
+-----+-----+-----+-----+
| NAME | FLEET SIZE | LAT | LON |
+-----+-----+-----+-----+
| Tzaneio | 6 | 37.9334 | 23.6458 |
+-----+-----+-----+-----+
| Metaxa | 3 | 37.9294 | 23.6438 |
+-----+-----+-----+-----+
| Onaseio | 4 | 37.9427 | 23.6976 |
+-----+-----+-----+-----+
| Eyaggelismos | 10 | 37.9777 | 23.7477 |
+-----+-----+-----+-----+
| Alexandra | 7 | 37.9801 | 23.7554 |
+-----+-----+-----+-----+
| Attikon | 5 | 38.0163 | 23.6661 |
+-----+-----+-----+-----+
TOTAL SUPPLY: 35

```

**Figure 3-14: EMS Station data.**

#### 3.2.1.1 Example 1 - No excess demand

For the first example, the total demand is less than the total supply, that is there is no excess demand. The incident data is the following:



```

INCIDENTS:
+-----+-----+-----+-----+
| NAME | DEMAND | LAT | LON |
+-----+-----+-----+-----+
| PAPEI | 5 | 37.9417 | 23.6528 |
+-----+-----+-----+-----+
| Omonoia | 8 | 37.984 | 23.7281 |
+-----+-----+-----+-----+
| Skaramgkas | 5 | 38.0115 | 23.6042 |
+-----+-----+-----+-----+
TOTAL DEMAND: 18

```

**Figure 3-15: Incident data for example 1.**

Using the *EMSAAllocation* class and its method *allocate*, the obtained results are:



```

ALLOCATION:
+-----+-----+-----+
| FROM   | TO     | # EMS UNITS |
+-----+-----+-----+
| Tzaneio | PAPEI  | 5           |
+-----+-----+-----+
| Eyaggelismos | Omonoia | 8           |
+-----+-----+-----+
| Attikon | Skaramgkas | 5           |
+-----+-----+-----+
REDUCED DEMAND: False

```

**Figure 3-16: Recommended allocation for example 1.**

The above results mean that *Tzaneio* should dispatch 5 EMS units to *PAPEI*, and so on. The *REDUCED DEMAND* variable is *False*, since there is no excess demand, i.e., the total supply fulfils the total demand. The example can be well understood using a map that illustrates the nodes and the links between them.

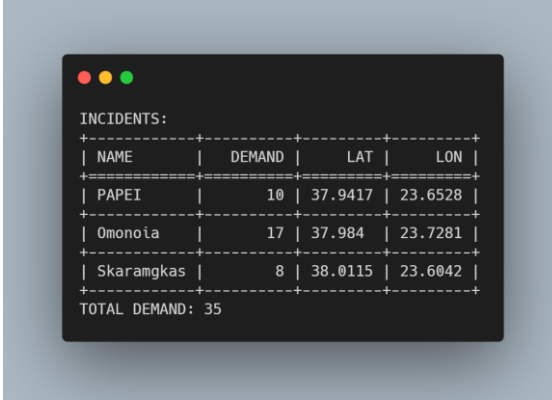


**Figure 3-17: Map Illustration for example 1.**

It is remarkable that the workload is not distributed. As mentioned in D4.3, the allocation is based on the total minimum distance, and does not consider other factors. For example, *Tzaneio* provides 6 EMS units and 5 of them are allocated to *PAPEI*. Since *Tzaneio* and *Metaxa* are near each other, the workload could be split into 3 dispatched EMS units from *Tzaneio* and 2 dispatched EMS units from *Metaxa*. In the second version the output of the Service will be analyzed and validated by the end users.

### 3.2.1.2 Example 2 - Equal total demand and total supply

The second examples test the case of equal total demand and total supply. The demand of the incidents for this example is:

A terminal window with a dark background and light text. It displays a table of incident data. The table has four columns: NAME, DEMAND, LAT, and LON. There are three rows of data. Below the table, it shows 'TOTAL DEMAND: 35'. The terminal window has three colored dots (red, yellow, green) in the top left corner.

```
INCIDENTS:
+-----+-----+-----+-----+
| NAME   | DEMAND | LAT   | LON   |
+-----+-----+-----+-----+
| PAPEI  | 10     | 37.9417 | 23.6528 |
+-----+-----+-----+-----+
| Omonia | 17     | 37.984  | 23.7281 |
+-----+-----+-----+-----+
| Skaramgkas | 8     | 38.0115 | 23.6042 |
+-----+-----+-----+-----+
TOTAL DEMAND: 35
```

**Figure 3-18: Incident data for example 2.**

Solving the transportation problem, the results are:



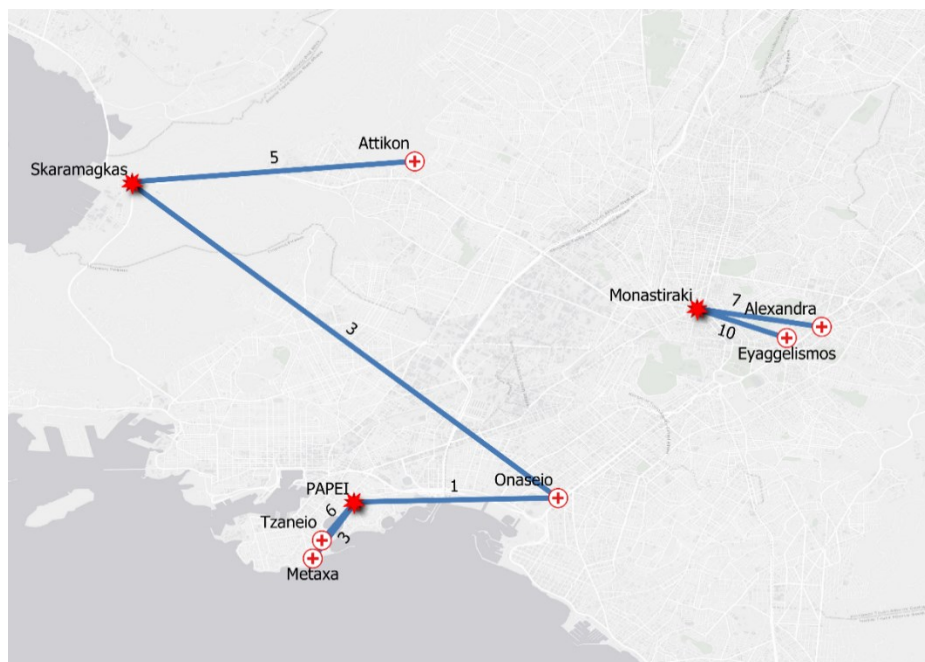
```

ALLOCATION:
+-----+-----+-----+
| FROM   | TO     | # EMS UNITS |
+-----+-----+-----+
| Tzaneio | PAPEI  | 6           |
+-----+-----+-----+
| Metaxa  | PAPEI  | 3           |
+-----+-----+-----+
| Onaseio  | PAPEI  | 1           |
+-----+-----+-----+
| Onaseio  | Skaramgkas | 3           |
+-----+-----+-----+
| Eyaggelismos | Omonoia | 10          |
+-----+-----+-----+
| Alexandra | Omonoia | 7           |
+-----+-----+-----+
| Attikon  | Skaramgkas | 5           |
+-----+-----+-----+

REDUCED DEMAND: False

```

**Figure 3-19: Recommended allocation for example 2.**



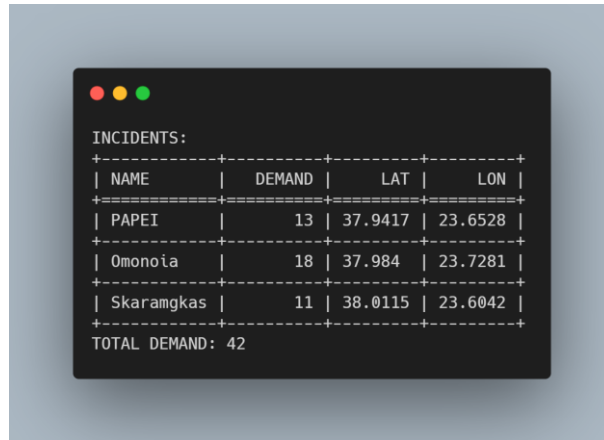
**Figure 3-20: Map Illustration for example 2.**

The comment for this map is that all EMS stations dispatch their fleets to fulfil the demand. Furthermore, there is no reduction of the demand, since the total demand and the total supply are equal.

### 3.2.1.3 Example 3 - Excess demand

There are cases that the total demand is greater than the total supply. The *TransportationProblem*, used by the *EMSAallocation*, will deal with this problem by reducing the demand of each node fairly. Assume the following incidents:





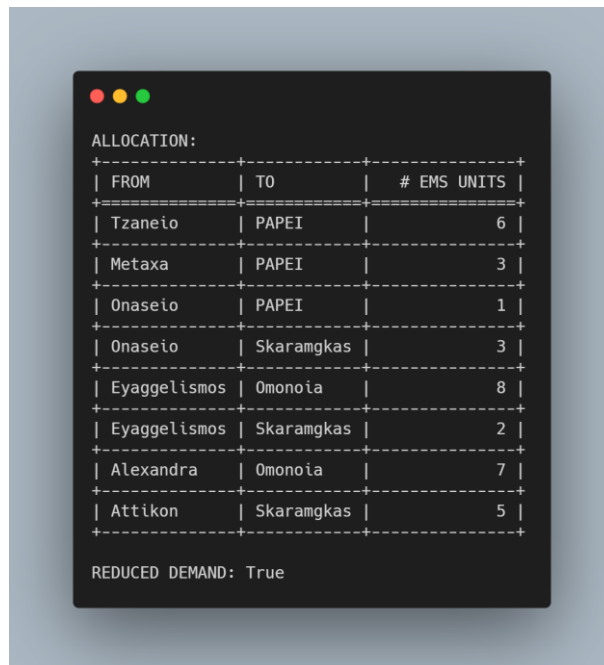
```

INCIDENTS:
+-----+-----+-----+-----+
| NAME      | DEMAND | LAT   | LON   |
+-----+-----+-----+-----+
| PAPEI     | 13     | 37.9417 | 23.6528 |
+-----+-----+-----+-----+
| Omonoia   | 18     | 37.984  | 23.7281 |
+-----+-----+-----+-----+
| Skaramgkas | 11     | 38.0115 | 23.6042 |
+-----+-----+-----+-----+
TOTAL DEMAND: 42

```

**Figure 3-21: Incident data for example 3.**

The results of *EMSAAllocation*'s *allocate* method is:



```

ALLOCATION:
+-----+-----+-----+
| FROM      | TO      | # EMS UNITS |
+-----+-----+-----+
| Tzaneio   | PAPEI   | 6           |
+-----+-----+-----+
| Metaxa    | PAPEI   | 3           |
+-----+-----+-----+
| Onaseio   | PAPEI   | 1           |
+-----+-----+-----+
| Onaseio   | Skaramgkas | 3           |
+-----+-----+-----+
| Eyaggelismos | Omonoia | 8           |
+-----+-----+-----+
| Eyaggelismos | Skaramgkas | 2           |
+-----+-----+-----+
| Alexandra | Omonoia | 7           |
+-----+-----+-----+
| Attikon   | Skaramgkas | 5           |
+-----+-----+-----+
REDUCED DEMAND: True

```

**Figure 3-22: Recommended allocation for example 3.**

Here, the *REDUCED DEMAND* is *True* meaning that the demand of each incident has been reduced. More specifically, each demand reduced approximately by  $\left(1 - \frac{\text{Total supply}}{\text{Total demand}}\right) \cdot 100\% = \left(1 - \frac{35}{42}\right) \cdot 100\% = 16.66\%$ .

Indeed, adding the recommended number of dispatched EMS units per Incident, the result is:

Incident	Initial demand	Reduction (%)	Final demand
<b>PAPEI</b>	13	23.07	10
<b>Omonoia</b>	18	16.66	15
<b>Skaramagkas</b>	11	9.09	10

**Table 3-1: Demand reduction.**

One can see that the reduction is not the same for all incidents, and some differ quite a lot from 16.66%. This happens due to the floor function that the reduced demand is applied to, and the recursive addition of 1 to the minimum demand, in order to balance the total demand and total supply. In details, the steps are:

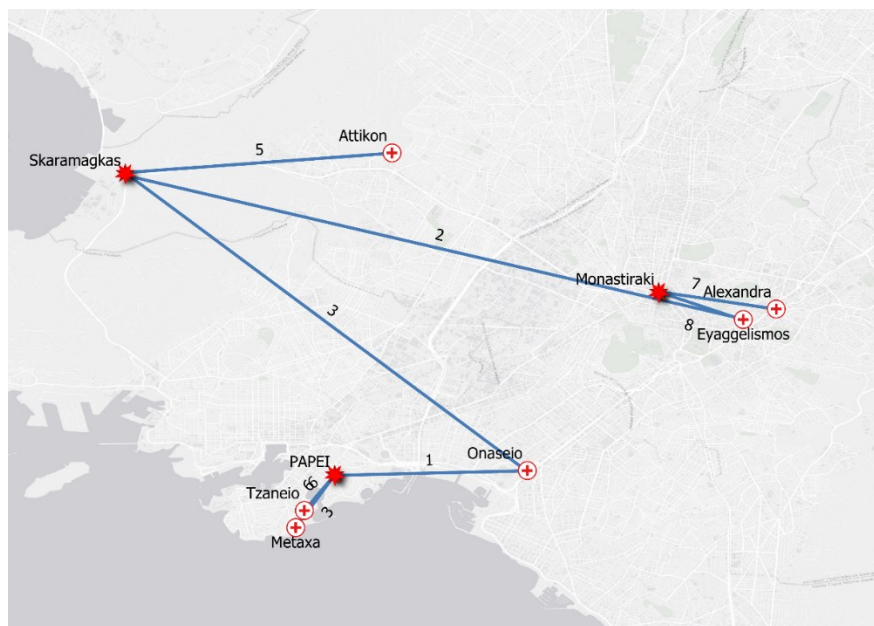
1. Calculate the reduction percentage, 16.66%
2. Calculate the new demand and take the integer part

Incident	Old demand	New demand	Integer part
<b>PAPEI</b>	13	10.82	10
<b>Omonoia</b>	18	14.99*	15*
<b>Skaramagkas</b>	11	9.13	9

**Table 3-2: Detailed demand reduction.**

\* The result of Omonoia's new demand is 14.9999... but Python's floating-point calculations will return 15 as result.

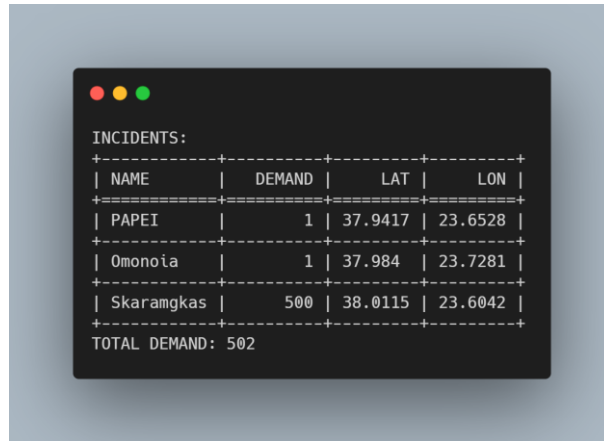
3. Check if total demand is less than total supply, Total demand = 34 < 35 = Total supply
4. If yes, find the minimum demand and add 1 and go to 3. Skaramagkas' demand: 9 -> 10
5. Else, return result.



**Figure 3-23: Map Illustration for example 3.**

#### 3.2.1.4 Example 4 - Excess demand results to demand less than one

There are extreme cases in which the demand reduction results in new demand less than one, for example an incident with demand = 2, after a reduction of 80%, will have a demand of  $[0.4] = 0$ . In order to avoid situations with zero demand, a unit of demand is added. Suppose a case with the following demand:



```

INCIDENTS:
+-----+-----+-----+-----+
| NAME   | DEMAND | LAT   | LON   |
+-----+-----+-----+-----+
| PAPEI  | 1      | 37.9417 | 23.6528 |
+-----+-----+-----+-----+
| Omonoia | 1      | 37.984  | 23.7281 |
+-----+-----+-----+-----+
| Skaramgkas | 500   | 38.0115 | 23.6042 |
+-----+-----+-----+-----+
TOTAL DEMAND: 502

```

**Figure 3-24: Incident data for example 4.**

Calculate the reduction percentage as  $\left(1 - \frac{35}{502}\right) \cdot 100\% = 93.02\%$ . Thus, the new demand is:

Incident	Initial demand	New demand	Integer part
PAPEI	1	0.06	0
Omonoia	1	0.06	0
Skaramagkas	500	34.86	34

**Table 3-3: Demand reduction, first step.**

Set 1 the zero demand and calculate the new total demand.

Incident	Integer part	New demand	Total demand
PAPEI	0	1	36
Omonoia	0	1	
Skaramagkas	34	34	

**Table 3-4: Demand reduction, second step.**

Still, there is excess demand, so reduce the maximum demand by 1 until there is balance between total demand and total supply. The final demand is:

Incident	Initial demand	Final demand	Total demand
PAPEI	1	1	35
Omonoia	1	1	
Skaramagkas	500	33	

**Table 3-5: Demand reduction, third step.**

The recommended allocation for this example is:

```

ALLOCATION:
+-----+-----+-----+
| FROM   | TO     | # EMS UNITS |
+-----+-----+-----+
| Tzaneio | PAPEI  | 1           |
+-----+-----+-----+
| Tzaneio | Skaramgkas | 5           |
+-----+-----+-----+
| Metaxa  | Skaramgkas | 3           |
+-----+-----+-----+
| Onaseio | Skaramgkas | 4           |
+-----+-----+-----+
| Eyaggelismos | Skaramgkas | 10          |
+-----+-----+-----+
| Alexandra | Omonoia | 1           |
+-----+-----+-----+
| Alexandra | Skaramgkas | 6           |
+-----+-----+-----+
| Attikon  | Skaramgkas | 5           |
+-----+-----+-----+
REDUCED DEMAND: True

```

**Figure 3-25: Recommended allocation for example 4.**

### 3.2.2 Service 2

It is important to note that any location of patients and EMS units are used for reference and intuitive displaying on maps. Patients' attributes are purely random.

The Service 2 demonstration is split into four examples, varying on the patient and EMS unit locations and the cost function. The results must be evaluated by a domain expert

#### 3.2.2.1 Example 1 – Skaramagkas + Patient cost formula

For the 1<sup>st</sup> example, assume an incident in Skaramagkas area in Athens, Greece, with the following patients and EMS units:

```

PATIENTS: 6
+-----+-----+-----+-----+-----+-----+
| ID | SCORE | ORDER | IS CHILD | LAT | LON |
+-----+-----+-----+-----+-----+-----+
| 1 | 3 | 5 | 0 | 38.0107 | 23.6022 |
+-----+-----+-----+-----+-----+-----+
| 2 | 3 | 4 | 1 | 38.0154 | 23.598 |
+-----+-----+-----+-----+-----+-----+
| 3 | 5 | 6 | 0 | 38.0059 | 23.5957 |
+-----+-----+-----+-----+-----+-----+
| 4 | 2 | 3 | 0 | 38.0115 | 23.6012 |
+-----+-----+-----+-----+-----+-----+
| 5 | 1 | 1 | 0 | 38.0136 | 23.5942 |
+-----+-----+-----+-----+-----+-----+
| 6 | 5 | 2 | 1 | 38.003 | 23.6028 |
+-----+-----+-----+-----+-----+-----+

EMS UNITS: 3
+-----+-----+-----+
| ID | LAT | LON |
+-----+-----+-----+
| 1 | 38.0005 | 23.5941 |
+-----+-----+-----+
| 2 | 38.0107 | 23.6042 |
+-----+-----+-----+
| 3 | 38.0123 | 23.6022 |
+-----+-----+-----+

```

**Figure 3-26: Patient and EMS unit data for example 1 & 2.**

Using the *allocate* method of *PatientAllocation*, having set *uta=False*, the following results are obtained:



```

ALLOCATION:
+-----+-----+
| EMS UNIT | PATIENT |
+-----+-----+
|         1 |      3  |
+-----+-----+
|         2 |      6  |
+-----+-----+
|         3 |      2  |
+-----+-----+
|        -1 | [1, 4, 5] |
+-----+-----+

```

**Figure 3-27: Recommended allocation for example 1.**

These results may not be meaningful, so a map illustration of the allocation has been created.



**Figure 3-28: Map illustration for example 1.**

In Figure 3-28, the number of EMS units is less than the number of patients, so some patients have to wait for the next 'wave' of transportation. Patients 1, 4 and 5 have not been allocated to any EMS unit. Looking at the data table, patient 1 has a medium physiological score and is 5<sup>th</sup> in order out of 6. Also, patients 4 and 5 have low physiological scores and are high in order, but they did not be allocated. On the contrary, patients 2, 3 and 6 have high physiological scores, generally mixed order and two of them are children.

### 3.2.2.2 Example 2 – Skaramagkas + UTA

This example uses the same data as the previous example but tests the utility of the UTA method for the cost matrix calculation. Following the same steps, the results are:

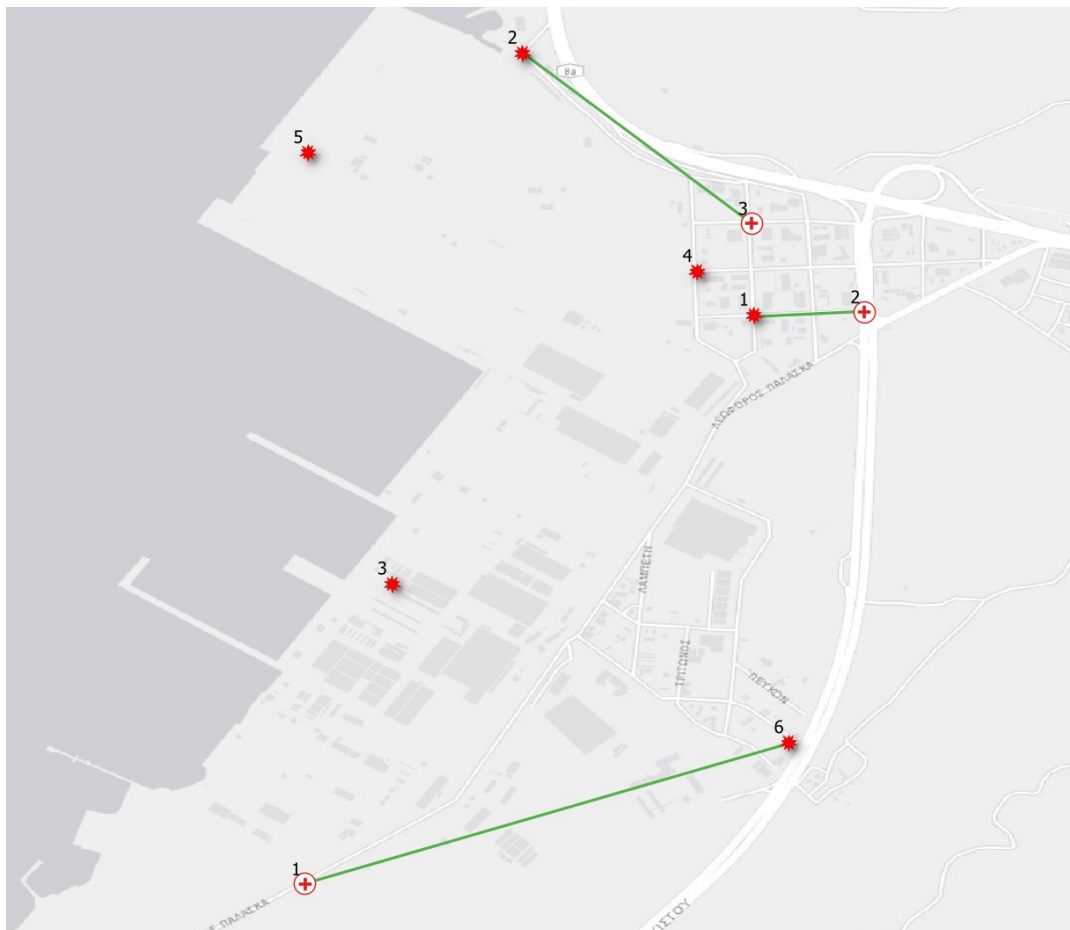


```

ALLOCATION:
+-----+-----+
| EMS UNIT | PATIENT |
+-----+-----+
|         1 |      6  |
+-----+-----+
|         2 |      1  |
+-----+-----+
|         3 |      2  |
+-----+-----+
|        -1 | [3, 4, 5] |
+-----+-----+

```

**Figure 3-29: Recommended allocation of example 2.**



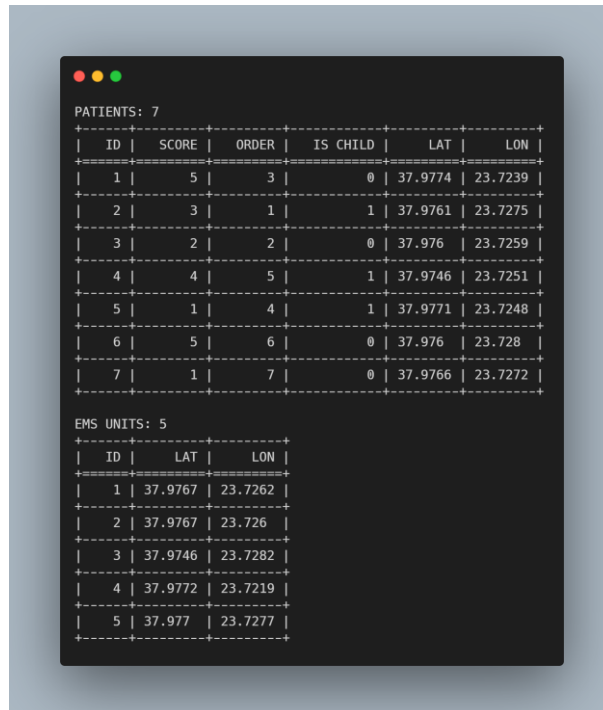
**Figure 3-30: Map illustration of example 2.**

It is noteworthy that using the UTA method, the allocation is similar to the previous example but this time, patient 1 is allocated instead of patient 3. Patient 3 is in a more critical situation than patient 1

but patient 1 has a higher order of evacuation. One can see that the UTA emphasizes on the subjective ranking, i.e., order of evacuation, as described in D4.3.

### 3.2.2.3 Example 3 – Monastiraki + Patient cost formula

This example takes place in Monastiraki, Athens, Greece with the following patients and EMS units:



```

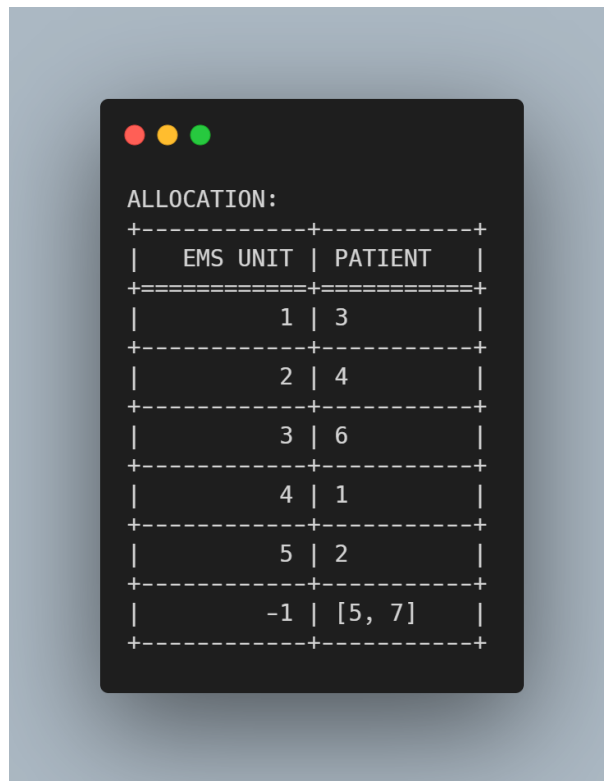
PATIENTS: 7
+-----+-----+-----+-----+-----+-----+
| ID | SCORE | ORDER | IS CHILD | LAT | LON |
+-----+-----+-----+-----+-----+-----+
| 1 | 5 | 3 | 0 | 37.9774 | 23.7239 |
+-----+-----+-----+-----+-----+-----+
| 2 | 3 | 1 | 1 | 37.9761 | 23.7275 |
+-----+-----+-----+-----+-----+-----+
| 3 | 2 | 2 | 0 | 37.976 | 23.7259 |
+-----+-----+-----+-----+-----+-----+
| 4 | 4 | 5 | 1 | 37.9746 | 23.7251 |
+-----+-----+-----+-----+-----+-----+
| 5 | 1 | 4 | 1 | 37.9771 | 23.7248 |
+-----+-----+-----+-----+-----+-----+
| 6 | 5 | 6 | 0 | 37.976 | 23.728 |
+-----+-----+-----+-----+-----+-----+
| 7 | 1 | 7 | 0 | 37.9766 | 23.7272 |
+-----+-----+-----+-----+-----+-----+

EMS UNITS: 5
+-----+-----+-----+
| ID | LAT | LON |
+-----+-----+-----+
| 1 | 37.9767 | 23.7262 |
+-----+-----+-----+
| 2 | 37.9767 | 23.726 |
+-----+-----+-----+
| 3 | 37.9746 | 23.7282 |
+-----+-----+-----+
| 4 | 37.9772 | 23.7219 |
+-----+-----+-----+
| 5 | 37.977 | 23.7277 |
+-----+-----+-----+

```

**Figure 3-31: Patient and EMS unit data for examples 3 & 4.**

The results of *allocate* method of *PatientAllocation*, with *uta=False* are:



```

ALLOCATION:
+-----+-----+
| EMS UNIT | PATIENT |
+-----+-----+
|         1 |      3  |
+-----+-----+
|         2 |      4  |
+-----+-----+
|         3 |      6  |
+-----+-----+
|         4 |      1  |
+-----+-----+
|         5 |      2  |
+-----+-----+
|        -1 | [5, 7] |
+-----+-----+

```

**Figure 3-32: Recommended allocation for example 3.**



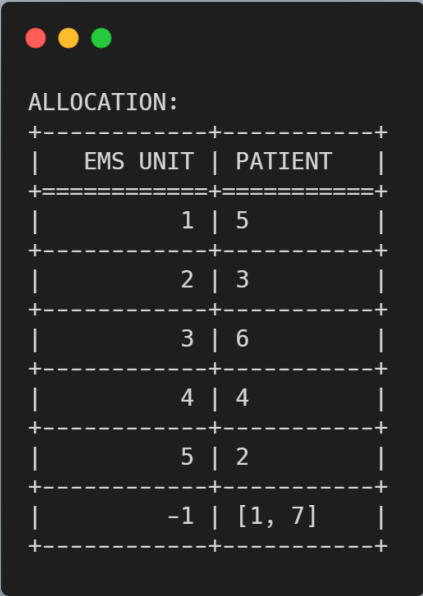
**Figure 3-33: Map illustration for example 3.**

Same as before, the excess demand results in unallocated patients. Patients 5 and 7 have low physiological scores and low order of evacuation. Also, patient 5 is a child but still not allocated due to the other factors.

#### 3.2.2.4 Example 4 – Monastiraki + UTA

The last example has the same data as example 3 but uses the UTA method for the calculation of cost matrix. So, the results are:



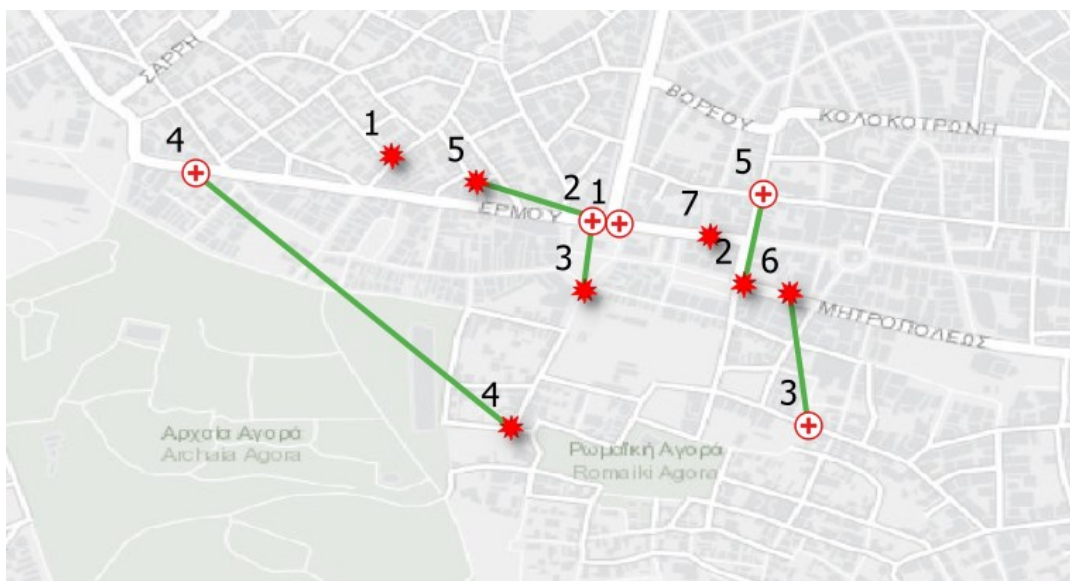


```

ALLOCATION:
+-----+-----+
| EMS UNIT | PATIENT |
+-----+-----+
|         1 |        5 |
+-----+-----+
|         2 |        3 |
+-----+-----+
|         3 |        6 |
+-----+-----+
|         4 |        4 |
+-----+-----+
|         5 |        2 |
+-----+-----+
|        -1 | [1, 7] |
+-----+-----+

```

**Figure 3-34: Recommended allocation for example 4.**



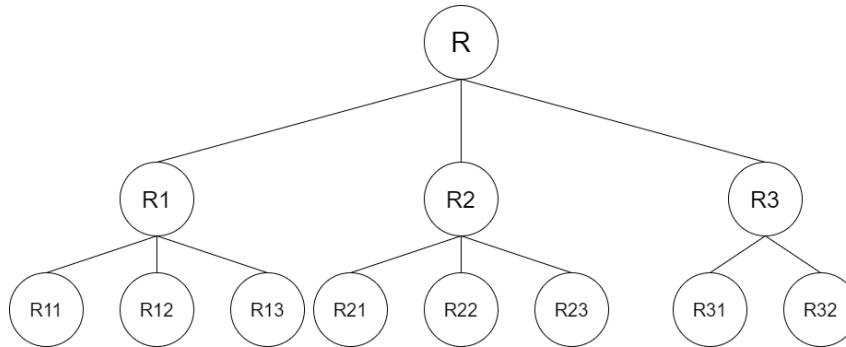
**Figure 3-35: Map illustration for example 4.**

In figure 3-35, the unallocated patients are 1 and 7, that is patient 1 instead of patient 5 of the previous example. UTA prioritized a child with a low physiological score (patient 5) over an adult with a high physiological score (patient 5). Still, these results have to be evaluated by a domain expert to choose the most suitable cost function.

### 3.2.3 Service 3

It is important to note that any location of actors and tasks are used for reference and intuitive displaying on maps.

The service 3 demonstration is split into four examples, using the same geographic data as the service 2 demonstrator. The results must be evaluated by a domain expert. For this and only demonstration a dummy role tree was created and used for the role distance calculations.



**Figure 3-36: Role tree for demonstration.**

#### 3.2.3.1 Example 1 – Skaramagkas + Task Cost Formula + Excess demand

The example takes place in Skaramagkas, Athens, Greece with six actors and three tasks. The same data will be used for example 2.

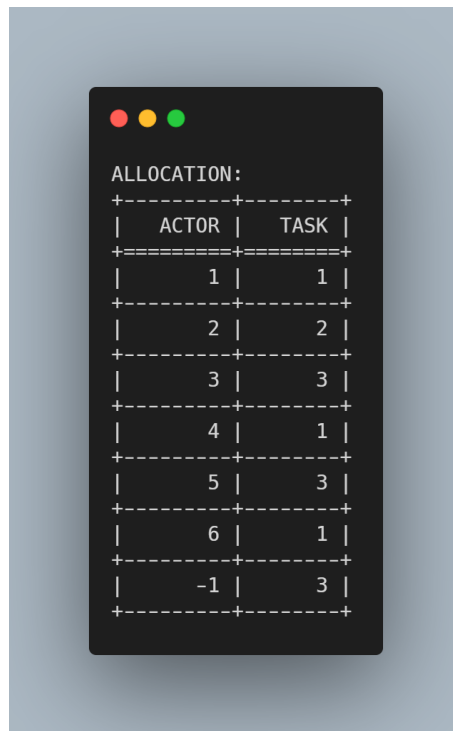
```

ACTORS: 6
+-----+-----+-----+-----+-----+
| ID | ROLE | SCORE | LAT | LON |
+-----+-----+-----+-----+-----+
| 1 | R11 | 5 | 38.0107 | 23.6022 |
+-----+-----+-----+-----+-----+
| 2 | R23 | 4 | 38.0154 | 23.598 |
+-----+-----+-----+-----+-----+
| 3 | R31 | 6 | 38.0059 | 23.5957 |
+-----+-----+-----+-----+-----+
| 4 | R12 | 3 | 38.0115 | 23.6012 |
+-----+-----+-----+-----+-----+
| 5 | R22 | 1 | 38.0136 | 23.5942 |
+-----+-----+-----+-----+-----+
| 6 | R11 | 2 | 38.003 | 23.6028 |
+-----+-----+-----+-----+-----+

TASKS: 3
+-----+-----+-----+-----+-----+
| ID | ROLE | DEMAND | LAT | LON |
+-----+-----+-----+-----+-----+
| 1 | R11 | 3 | 38.0005 | 23.5941 |
+-----+-----+-----+-----+-----+
| 2 | R23 | 1 | 38.0107 | 23.6042 |
+-----+-----+-----+-----+-----+
| 3 | R22 | 3 | 38.0123 | 23.6022 |
+-----+-----+-----+-----+-----+
  
```

**Figure 3-37: Actor and Task data for examples 1 & 2.**

From the Figure 3-37, the total supply as 6 (number of actors) and the total demand as 7 (sum of demand) can be calculated, so there is excess demand, and a dummy supply node will be added. Solving the problem with the *allocate* method of *TaskAllocation* class and Task Cost Formula method, the recommended allocation is:



```
ALLOCATION:
+-----+-----+
|  ACTOR |  TASK |
+-----+-----+
|     1  |     1 |
+-----+-----+
|     2  |     2 |
+-----+-----+
|     3  |     3 |
+-----+-----+
|     4  |     1 |
+-----+-----+
|     5  |     3 |
+-----+-----+
|     6  |     1 |
+-----+-----+
|    -1  |     3 |
+-----+-----+
```

**Figure 3-38: Recommended allocation for example 1.**

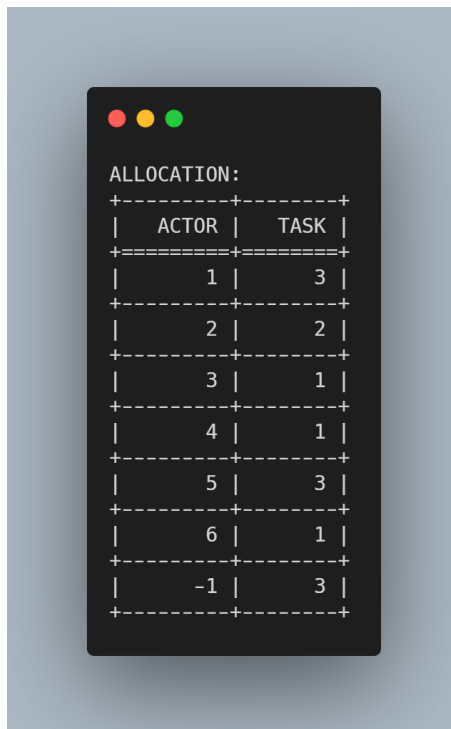
Actors 1 and 6, who are in role 'R11' are assigned to task 1, which requires 3 actors of role 'R11'. To fulfil the demand, actor 4 who is in role 'R12' has also been assigned to that task. It has to be mentioned that another actor with greater role distance, for example 'R32', could be assigned to that task if his/her location was closer to the task and his/her physiological score was lower, i.e., better performance. The allocation depends on these attributes too, as well as on the relative costs of the other actors. Last but not least, one can see that one demand of task 3 is not fulfilled since the dummy node has been assigned to deal with the excess demand.



**Figure 3-39: Map illustration for example 1.**

3.2.3.2 Example 2 – Skaramagkas + UTA + excess demand

In this example the same data is used, as the example 1. The main and only difference is the cost matrix calculation method. The UTA method is used and the recommended allocation is:



**Figure 3-40: Recommended allocation for example 2.**

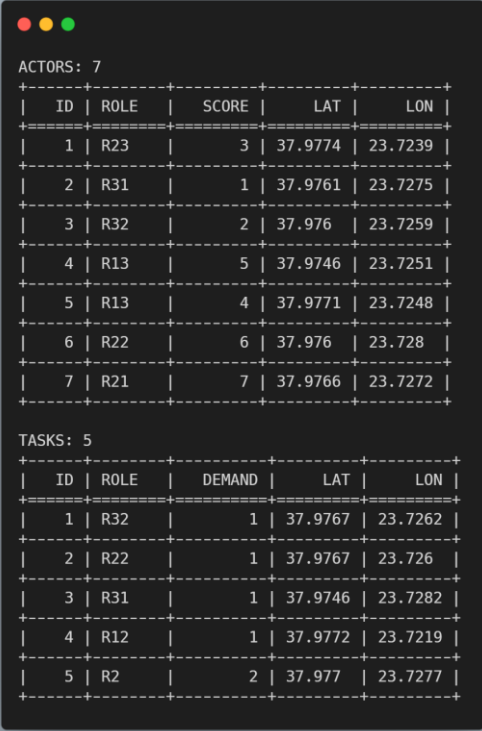
Actor 6, who is in the role 'R11' is assigned to task 1, which requires 3 actors of the role 'R11'. To fulfil the demand, actors 3 and 4 who are in the roles 'R31' and 'R12', respectively, have also been assigned to that task. On the contrary, actor 1 who is in role the 'R11', is not assigned to task 1 since that task is far and the actor has a high physiological score, i.e., he/she is tired.



**Figure 3-41: Map illustration for example 2.**

### 3.2.3.3 Example 3 – Monastiraki + Task Cost Formula

For the third example, the incident takes place in Monastiraki, Athens, Greece with seven actors and five tasks. The Task Cost Formula will be used for the cost matrix calculation. The following data is the same for examples 3 and 4.



```

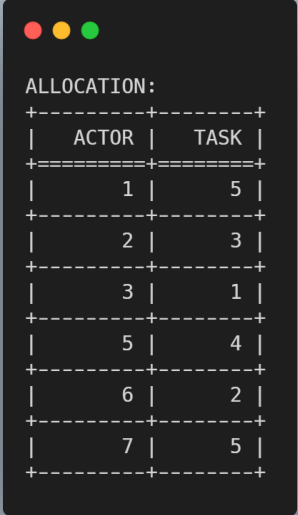
ACTORS: 7
+-----+-----+-----+-----+-----+
| ID | ROLE | SCORE | LAT | LON |
+-----+-----+-----+-----+-----+
| 1 | R23 | 3 | 37.9774 | 23.7239 |
+-----+-----+-----+-----+-----+
| 2 | R31 | 1 | 37.9761 | 23.7275 |
+-----+-----+-----+-----+-----+
| 3 | R32 | 2 | 37.976 | 23.7259 |
+-----+-----+-----+-----+-----+
| 4 | R13 | 5 | 37.9746 | 23.7251 |
+-----+-----+-----+-----+-----+
| 5 | R13 | 4 | 37.9771 | 23.7248 |
+-----+-----+-----+-----+-----+
| 6 | R22 | 6 | 37.976 | 23.728 |
+-----+-----+-----+-----+-----+
| 7 | R21 | 7 | 37.9766 | 23.7272 |
+-----+-----+-----+-----+-----+

TASKS: 5
+-----+-----+-----+-----+-----+
| ID | ROLE | DEMAND | LAT | LON |
+-----+-----+-----+-----+-----+
| 1 | R32 | 1 | 37.9767 | 23.7262 |
+-----+-----+-----+-----+-----+
| 2 | R22 | 1 | 37.9767 | 23.726 |
+-----+-----+-----+-----+-----+
| 3 | R31 | 1 | 37.9746 | 23.7282 |
+-----+-----+-----+-----+-----+
| 4 | R12 | 1 | 37.9772 | 23.7219 |
+-----+-----+-----+-----+-----+
| 5 | R2 | 2 | 37.977 | 23.7277 |
+-----+-----+-----+-----+-----+

```

**Figure 3-42: Actor and Task data for example 3 & 4.**

The output of the *allocate* method of *TaskAllocation* class is:



```

ALLOCATION:
+-----+-----+
| ACTOR | TASK |
+-----+-----+
| 1 | 5 |
+-----+-----+
| 2 | 3 |
+-----+-----+
| 3 | 1 |
+-----+-----+
| 5 | 4 |
+-----+-----+
| 6 | 2 |
+-----+-----+
| 7 | 5 |
+-----+-----+

```

**Figure 3-43 Recommended allocation for example 3.**

In this example, the total demand is less than the total supply. So, actor 4 is not allocated to any task. Also, the demanded role of task 5 is 'R2'. This means that any sub-role is equally accepted, that is 'R21', 'R22' and 'R23'.



**Figure 3-44: Map illustration for example 3.**

#### 3.2.3.4 Example 4 – Monastiraki + UTA

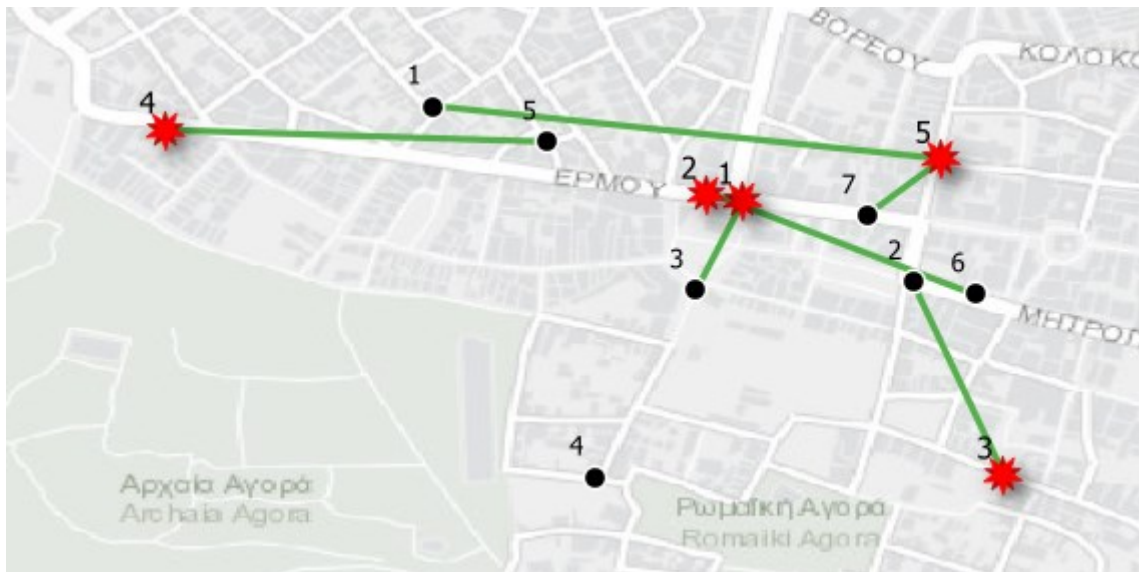
The data is the same as before, but the cost matrix is calculated using the UTA method. The recommended allocation is:

```

ALLOCATION:
+-----+
| ACTOR | TASK |
+-----+
| 1 | 5 |
+-----+
| 2 | 3 |
+-----+
| 3 | 1 |
+-----+
| 5 | 4 |
+-----+
| 6 | 2 |
+-----+
| 7 | 5 |
+-----+

```

**Figure 3-45: Recommended allocation for example 4.**



**Figure 3-46: Map illustration for example 4.**

Example 4 has the same results as example 3. This might be because the role and demand requirements are fulfilled perfectly, for example for a task with the role demand 'R32' there is at least one actor in the role 'R32'.

### 3.2.4 Service 4

This service requires real time data to work properly. For the sake of examples, the start and end time of earthquakes have been specified.

#### 3.2.4.1 Example 1 – onePAGER exists

For this example, the earthquake incident took place south of Punta de Burica, Panama on 2021-07-21. Using the EarthquakeCasualtyEstimation class and the result is:

```

EARTHQUAKE INCIDENT (INPUT):
+-----+-----+-----+-----+
|  LAT  |  LON  | START TIME | END TIME |
+-----+-----+-----+-----+
| 8.209 | -81.502 | 2021/07/20 | 2021/07/23 |
+-----+-----+-----+-----+

onePAGER URL:
https://earthquake.usgs.gov/product/losspager/us6000exs5/us/1626988879142/onepager.pdf

```

**Figure 3-47: Input and output for example 1**

The onePAGER document is:



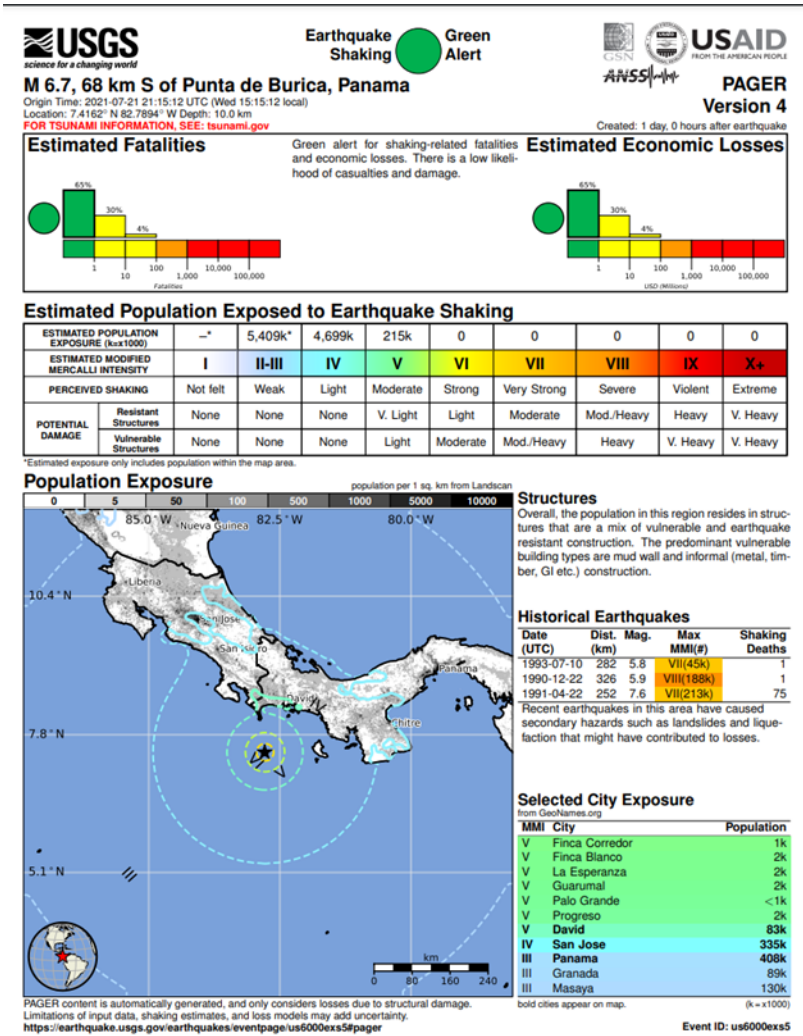


Figure 3-48: one PAGER for example 1

So, the estimated casualties are 0-1 with 65% probability and 1-10 with 30% probability.

3.2.4.2 Example 2 – onePAGER exists

Similarly for this example, an earthquake with bigger impact in search.

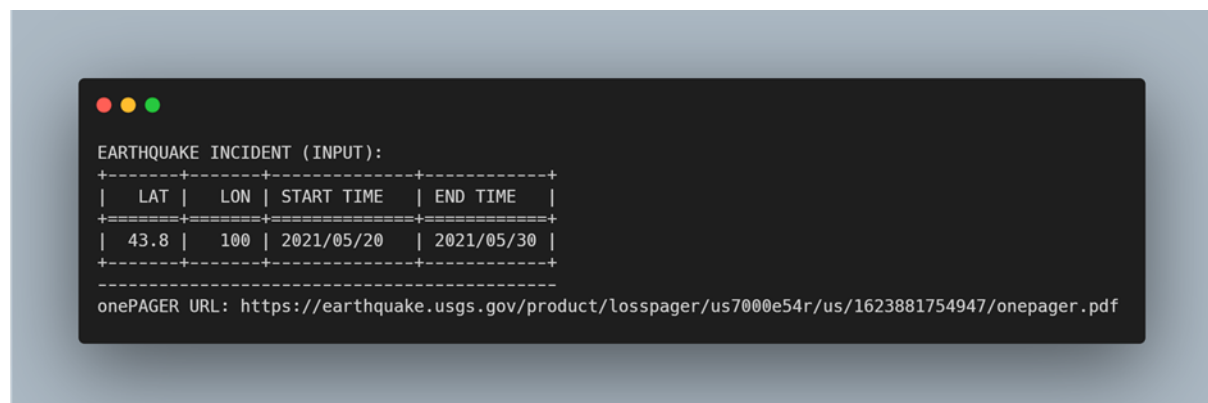


Figure 3-49: Input and output for example 2

The onePAGER document for this example:

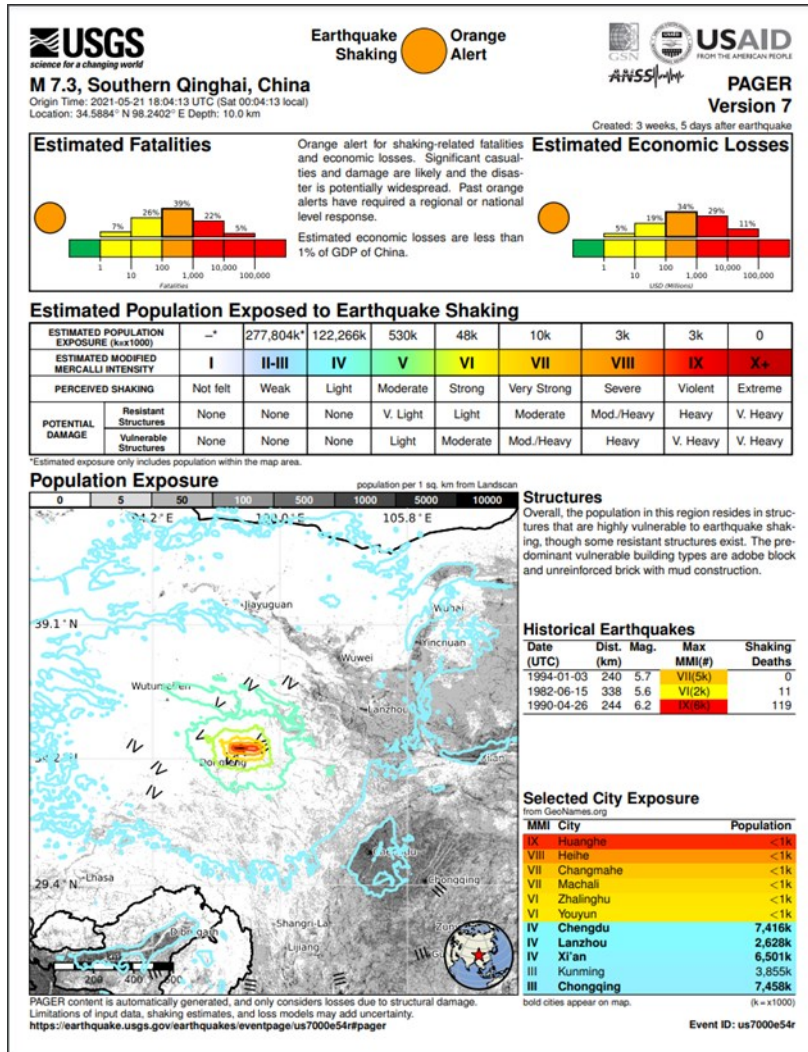


Figure 3-50: onePAGER for example 2.

Despite the fact that the earthquake was intense, there is a low population around the area. For that reason, the casualty estimation probabilities are spread among bins.

3.2.4.3 Example 3 – onePAGER does not exist

For the last example, the search area is at Monasthaki, Athens but there was not any earthquake that happened that time. So, the searching process is continuing until the earthquake report is found.

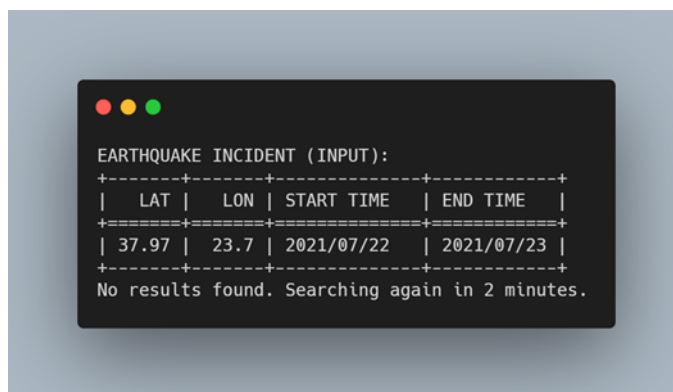


Figure 3-51: Input and output for example 3

To conclude, the above examples provide a detailed view of the outcomes of all the Services in different scenarios.

For Service 1 the examples have taken into consideration three scenarios. The first one is that the total demand for EMS Units is less than the supply of EMS Units. The second scenario is that the total demand is equal to the total supply and the third scenario is that the total demand of EMS Units is greater than the total supply of EMS Units.

The examples of the Service 2 are demonstrated by using two different methods. The first one is the Patient cost formula and the second is the UTA method.

For Service 3, the examples have taken into consideration two scenarios. The first one is that the total supply of the actors is greater than the total demand of the actors and the second is that the total demand of actors is less than the total supply of actors. In both scenarios two different methods, the Task Cost Formula and the UTA are implemented.

For Service 4 the use case related to an earthquake has been implemented by using the PAGER system designed, developed and maintained by the United States Geological Survey (USGS). Three scenarios are implemented in order to present a detailed view of the PAGER's outcome.

## 4 Conclusions

---

The technical details of the SOT DSS were described in the document. Six modules were created in order to provide the functionality to each of the four Services. Moreover, the technical details for the non-ML solutions were presented.

Furthermore, demonstrators for each of the four services were presented using dummy data in order to see the full functionality of them. Different examples were depicted for each Service by taking into consideration more than one scenario for a more detailed view of outcomes of the Services.

The second version of the deliverable, the D4.11 "Development of SOT DSS components v2" will provide enhanced technical implementation details about the four Services. More specifically, the main scope will be the improvement of Services 1 to 3 by taking into consideration more parameters for the allocation of resources to incidents, allocation of patients to hospitals and allocation of tasks to available actors on the field. Moreover, a further analysis will take place for Service 4 which will expand the functionalities of the Service to cover more emergency situations.

### Annex I: References

- [1] «SnR Gitlab,» [Ηλεκτρονικό]. Available: [https://gitlab.com/konnektable-devops/horizon-2020/s-r-central-repo-group/sot\\_dss](https://gitlab.com/konnektable-devops/horizon-2020/s-r-central-repo-group/sot_dss).
- [2] «Pyomo,» [Ηλεκτρονικό]. Available: <http://www.pyomo.org/>. [Πρόσβαση 25 08 2021].
- [3] «GitHub - usgs/libcomcat: Library of functions and wrapper scripts for accessing ANSS ComCat server data,» [Ηλεκτρονικό]. Available: <https://github.com/usgs/libcomcat>. [Πρόσβαση 25 08 2021].
- [4] «GitHub - coin-or/Cbc: COIN-OR Branch-and-Cut solver,» [Ηλεκτρονικό]. Available: <https://github.com/coin-or/Cbc>. [Πρόσβαση 25 08 2021].
- [5] «PAGER,» [Ηλεκτρονικό]. Available: <https://earthquake.usgs.gov/data/pager/>. [Πρόσβαση 25 08 2021].
- [6] «USGS.gov | Science for a changing world,» [Ηλεκτρονικό]. Available: <https://www.usgs.gov/>. [Πρόσβαση 25 08 2021].
- [7] «ANSS - Advanced National Seismic System,» [Ηλεκτρονικό]. Available: [https://www.usgs.gov/natural-hazards/earthquake-hazards/anss-advanced-national-seismic-system?qt-science\\_support\\_page\\_related\\_con=4#qt-science\\_support\\_page\\_related\\_con](https://www.usgs.gov/natural-hazards/earthquake-hazards/anss-advanced-national-seismic-system?qt-science_support_page_related_con=4#qt-science_support_page_related_con). [Πρόσβαση 25 08 2021].

[8] «Welcome to the QGIS project!»,» [Ηλεκτρονικό]. Available: <https://www.qgis.org/en/site/>.  
[Πρόσβαση 25 08 2021].