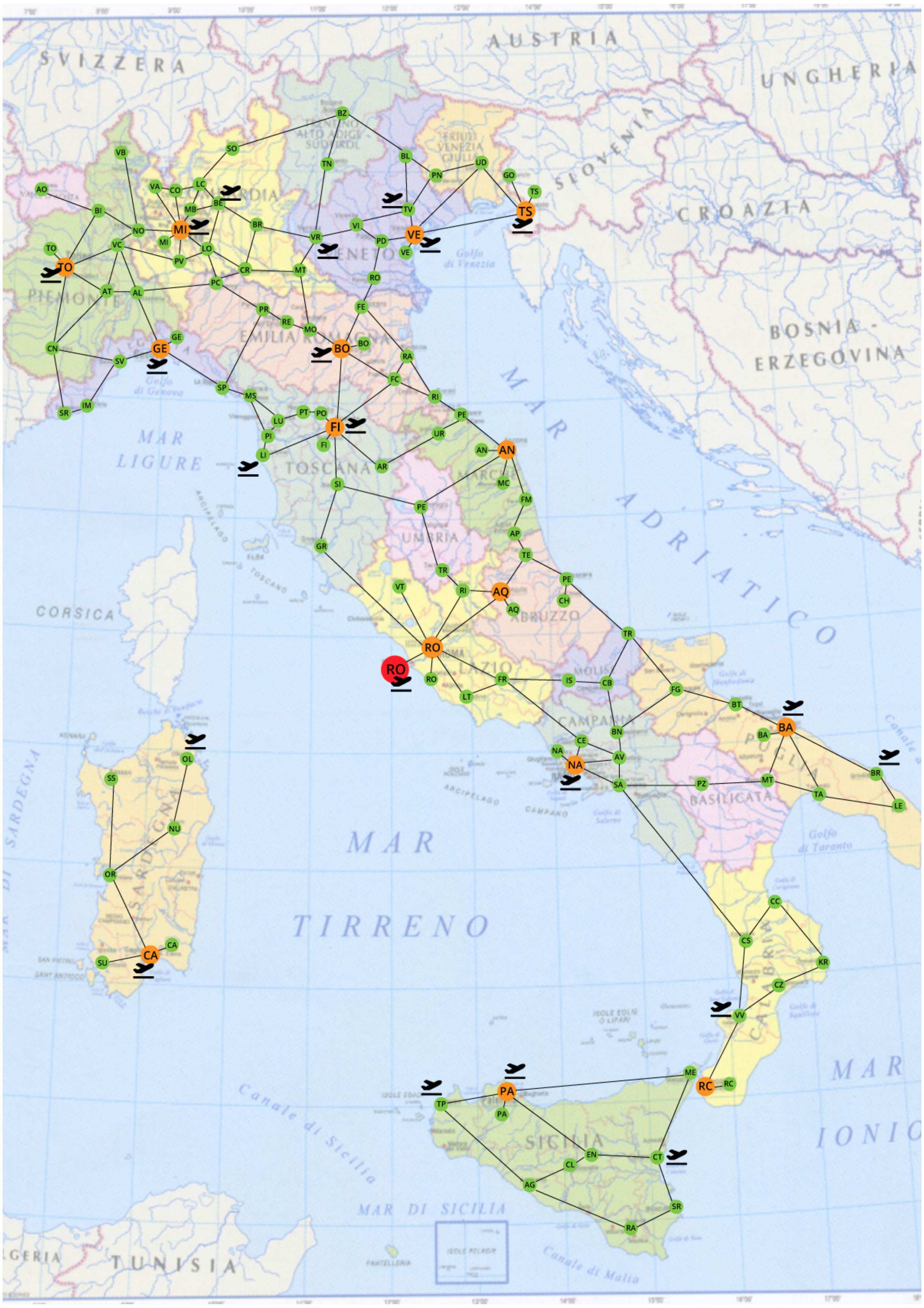

SVIZZERA
AUSTRIA
UNGHERIA
SLOVENIA
CROAZIA
BOSNIA - ERZEGOVINA
MAR ADRIATICO
MAR LIGURE
MAR TIRRENO
MAR IONIO
CORSICA
TUNISIA
MAR DI SICILIA
BZ
SO
TN
BL
UD
PN
GO
TS
VB
AO
VA
CO
LC
BE
MB
MI
BI
NO
TO
VC
PV
LO
BR
VR
VI
TV
VE
PD
CR
MT
PC
AT
AL
CN
GE
SV
IM
SR
PR
RE
MO
BO
FE
RO
RA
FC
RI
SP
MS
LU
PT
PO
FI
PI
LI
AR
PE
UR
AN
MC
FM
AP
TE
SI
GR
VT
TR
RI
AQ
CH
RO
LT
FR
IS
CB
TR
FG
BT
BA
BN
CE
NA
AV
SA
PZ
MT
TA
BR
LE
CC
CS
KR
CZ
VV
RC
ME
PA
TP
AG
CL
EN
CT
SR
RA
SS
OL
NU
OR
CA
SU

National Vaccine Distribution Plan
Autonomous Software Agents - `assignment 1`

Sebastiano Chiari
220527
`sebastiano.chiari@studenti.unitn.it`

1 Introduction

The goal of this project is to build a PDDL/MA-PDDL planner for a national vaccine distribution. It is required to design a PDDL domain and two PDDL problems, one full problem and a smaller version. As an additional bonus, it is asked to implement the same problem in MA-PDDL.

1.1 Delivery

Inside the delivered zip file, along with this report, it can be found:

- `domain.pddl`, the PDDL domain file
- `problem-full.pdll`, the full version of the PDDL problem
- `problem-small.pddl`, the smaller version of the PDDL problem
- `log-FULL.txt` and `log-SMALL.txt`, the two plans generated by respectively the FULL problem and the SMALL problem
- `utils` folder, containing all the scripts and additional files used to generate automatically the `problem-*.pddl` files
- `multi-agent` folder, containing two prototypes for the multi-agent domain and problem (section 4)

2 Domain

In the following section, all the implementation and design choices are reported.

2.1 Locations

There can be four different types of locations, all defined by the predicate `(location ?l)`:

- `CENTRAL DISTRIBUTION POINT`, the national point that receives all the vaccine boxes and has to start the distribution. There is only one central distribution point, assigned with the predicate `(isCDP ?location)`.
- `REGIONAL DISTRIBUTION CENTRE`, the regional point. There must be at least 15 regions. Each regional point is assigned with the predicate `(isRDC ?location)`.
- `PROVINCIAL DISTRIBUTION CENTRE`, the provincial point. There must be at least 6 provinces per region. Each provincial point is assigned with the predicate `(isPDC ?location)`.
- `HEALTH DISTRICT`, the destination of the vaccine boxes. There must be at least 3 health districts per province. Each health district is assigned with the predicate `(isHDDC ?location)`.

All the locations are interconnected with two types of predicates:

- `(link ?lowrank ?highrank)`, define the tree structure that force the path a vaccine box has to follow. Each path from the central point to the health district represents the steps a single vaccine box has to check in order to be delivered successfully to the final destination.

- `(connected ?l1 ?l2)`, define the physical connection (eg. road) between two locations. The ”connected” graph may or may not coincide with the ”links” graph. A vaccine box can take whatever path wants inside the ”connected” graph but has to stick

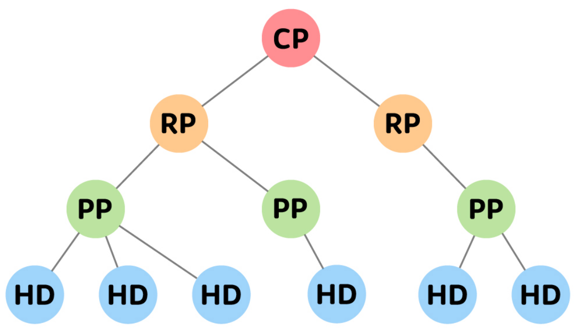


Figure 1: `link` graph

to the path defined in the ”link” graph for the load and unload actions.

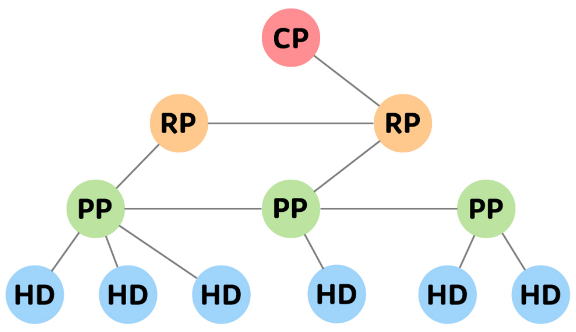


Figure 2: `connected` graph

Each regional, provincial or health district point can have an airport, defined by the predicate `(hasAirport ?location)`. This allows plane deliveries also from non directly connected locations.

2.2 Means of Transport

According to the project specifications, there are three types of means of transport:

- `planes`, maximum capacity equals to 20
  Planes are allowed to travel between cities that have an airport (predicate `hasAirport ?location`). Departure and arrival cities do not need to be directly connected with the predicate `(connected ?l1 ?l2)`.

- `trucks`, maximum capacity equals to 10
  Trucks can travel only between cities directly connected (predicate `(connected ?l1 ?l2)`). More, trucks are not allowed to reach health districts (this constraint has been added because planners tended not to use drones for the ”last mile” delivery without it).

- `drones`, maximum capacity equals to 1
  Drones can only move between a provincial distribution center and one of its health districts or viceversa. Drones can be relocated between provinces inside the same region with the `:action relocateDrone`.
  Given the choice to use a capacity equal to 1, instead of fluents (computationally more heavy) the `(hasLoaded? drone)` predicate is in charge of managing the load and unload operations of a vaccine box.

Given the choice of carrying out general and non-specific functions for each transport type, each plane, truck or drone has two predicates: `(transport ?t)` which defines the object `?t` as a means of transport (unlocking the move, load and unload actions) and `(plane ?transport)` or `(truck ?transport)` or `(drone ?transport)`, to define whether the means of transport is a plane, truck or a drone.

Inside the actions `load`, `unload` and `move`, different preconditions and effects are taken into consideration based on the type of the means of transport. This generalization was possible thanks to the use of additional requirements (on top of `:strips`, `:equality` and `:negative-preconditions`), like:

- `:disjunctive-preconditions`, allows the usage of `or` in goals and preconditions

- `:conditional-effects`, allows for the usage of `when` in expressing action effects

All means of transport have by default at spawn time capacity equals to 0, defined by the function `(capacity ?transport ?amount)` (except drones). This is modeled thanks to `:fluents`. Each vaccine box loaded inside a means of transport increases its capacity by 1: once the vaccine box is unloaded at the chosen location, the capacity is decreased by 1.

No restrictions have been added regarding the use of fuel (even if forcing the drone to move only from province to health district or vice versa simulates the need to recharge it after a delivery).

Predicate `(at ?what ?where)` is used to determine where a certain means of transport `?t` is located.

2.3 Vaccine Boxes

In order to simulate a real scenario, all vaccine boxes have to follow a mandatory path: from the central distribution point they have to reach a regional distribution centre. There, each vaccine box is unloaded and marked with the predicates `(getToRDC ?vaccineBox)` and `(reachRDC ?vaccineBox ?location)`. These predicates are essential in order to track the path each vaccine box takes and prevent it from being delivered to the wrong place. Then, the vaccine box is free to be loaded and can be delivered to a provincial distribution point inside the region (this is checked by the predicate `(link ?province ?region)`): once the vaccine box reaches its provincial destination, it is marked with the predicates `(getToPDC ?vaccineBox)` and `(reachPDC ?vaccineBox ?location)`. The same process goes from the provincial distribution center to the health district center, until the vaccine box reaches a fresh health district, which triggers the predicates `reachHDDC ?vaccineBox` and `(hasVaccineBox ?location)`. Now, the vaccine box has reached its final destination and cannot be moved anywhere else, nor the health district can receive other vaccine boxes.

As mentioned in section 2.1, all these forced links are modeled with the predicates `(link ?lowrank ?highrank)`, which define a tree structure (central point $\rightarrow$ regional point $\rightarrow$ provincial point $\rightarrow$ health district): this forces the logic path a vaccine box has to follow.

As mentioned in section 2.1, the physical path a vaccine box follows and the logical path may not coincide due to the different levels of logistic controls given by the predicates `(connected ?l1 ?l2)` and `(link ?lowrank ?highrank)`.

Predicate `(at ?what ?where)` is used to determine where a vaccine box `?t` is located, while predicate `(in ?what ?where)` is used to simulate the vaccine box being inside a means of transport.

3 Problem

3.1 Full problem definition

The idea is to model the Italian scenario as faithfully as possible. On the last page you will find a representation of the scenario up to the province level and the full representation of all the locations, airports and means of transport position can be found inside the delivered zip file.

In the full problem, we can find the following locations:

- `1 central point`
- `15 regions`
- `113 provinces`
- `412 health districts`
- `21 airports`
- `10 planes`
- `22 trucks`
- `78 drones`
- `412 vaccine boxes`

Due to the large amount of objects and predicates, the `vaccine-problem-FULL.pddl` file is generated via a python script `generator.py`, which can be found in the `utils` folder.

The script uses some `.txt` files in order to create the full problem in `.pddl` format:

- `regions.txt`, contains the names of all the region distribution centers
- `provinces.txt`, contains (in row format, separated by a space) all the province distribution centers belonging to the corresponding region
- `healthdistricts.txt`, contains (in row format, separated by a space) all the health district centers belonging to the corresponding province
- `connections.txt`, contains all the connections (already formatted in `.pddl` style) between regions, provinces and the central distribution center
- `airports.txt`, contains the names of all the cities with an airport

Inside the script, there are some constants to configure according to the size of the problem:

- `CENTRALPOINT`, the name of the central distribution point
- `NPLANES`, number of generated planes

- `NTRUCKS`, number of generated trucks
- `NDRONES`, number of generated drones
- `NVACCINEBOXES`, number of vaccine boxes. Due to specification constraints, if the chosen number of vaccine boxes is lower than the number of health districts, `NVACCINEBOXES` is set equal to `#HD`

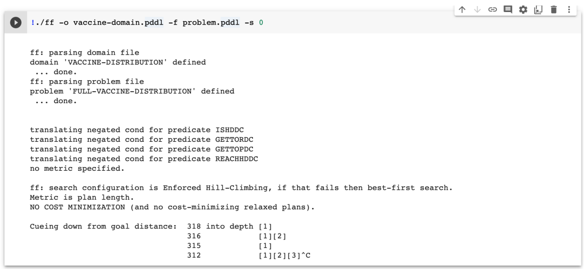
The connections between provinces and regions have been made with reference to the Italian road map. Regions can be found not directly connected to the central point but through other regional or provincial distribution centers. Some provinces may not be directly connected with the regional distribution point but through other provinces within the same region. To reduce complexity, the health districts are independent and connected only to the province of reference.

Airports' locations have been assigned taking into consideration the most important Italian airports and can be found in both region distribution centers and provincial distribution centers.

As indicated in the assignment guidelines, initially all vaccine boxes are placed at the central distribution point. Instead, planes, trucks and drones are positioned within the locations, taking into account some parameters:

- `planes` are randomly spawned according to where airports are located
- `trucks` can be found in the central or in regional points and they are fully randomly distributed
- `drones` can be found in provincial points or in health districts. Within the same region, drones are distributed randomly but on a national scale the number of drones for one region is determined by the GDP of the single region (normalized over the total number of provinces and health districts). The number of drones per region is contained inside the `drones.txt` file.

The full problem has been tested both with the provided virtual machine and with a Colab notebook (MetricFF was used as planner of choice because it was able to handle the added requirements). With both options, even with long computation times (beyond 12h on the virtual machine), the process (due to the large amount of objects and states that saturates the RAM) gets killed and the planner is not able to find a solution for the problem.



```
!./ff -o vaccine-domain.pddl -f problem.pddl -s 0

ff: parsing domain file
domain 'VACCINE-DISTRIBUTION' defined
 ... done.
ff: parsing problem file
problem 'FULL-VACCINE-DISTRIBUTION' defined
 ... done.


translating negated cond for predicate ISHDDC
translating negated cond for predicate GETTORDC
translating negated cond for predicate GETTOPDC
translating negated cond for predicate REACHHDDC
no metric specified.

ff: search configuration is Enforced Hill-Climbing, if that fails then best-first search.
Metric is plan length.
NO COST MINIMIZATION (and no cost-minimizing relaxed plans).

Cueing down from goal distance:   318 into depth [1]
                                  316            [1][2]
                                  315            [1]
                                  312            [1][2][3]^C
```

Figure 3: Output from the Colab Notebook after the killed execution of the planner

For this reason, it has been modeled also a small version of the problem, discussed in the following section.

3.2 Small problem definition

The goal was to build a problem that respected as far as possible the constraints given (e.g., at least one region without airport). More, the small problem should have all the conditions that were reported in the full problem (ex. one island region, connected to the central point only via an airport).

The small problem was built by trial and error starting from a basic problem and by adding various pieces a little at a time in order to make it resembles the full problem: the parameter used to validate a possible solution was to submit the problem to the planner and see if it was able to produce a solution under half an hour without getting stuck.

Due to the large amount of predicates used to control the vaccine distribution pipeline that quickly saturates the available RAM, the small problem results in a very little (but still meaningful) representation of the full problem (MetricFF launch on the Virtual Machine with 14GB of RAM is able to solve this problem in under 2 hours).

In the small problem, we can find the following configuration (the full representation of the scenario can be found later in this report):

- `1 central point`
- `3 regions`
- `6 provinces`
- `15 health districts`
- `2 airports`

- `1 plane`
- `3 trucks`
- `3 drones`
- `15 vaccine boxes`

As requested, along with this report it can be found the output of the planner with the complete solution of the problem (`small-LOG.txt` file).

4 Multi-agent implementation

Inside the `multi-agent` folder, two attemped solutions can be found using MA-PDDL. In order to test the implementation, the planner of choice is `maplan`, which has some important limitations (as mentioned in the following paragraphs).

Most of the functions and predicates are the same as in the single agent implementation. Due to the planner's limitations, the structure is slightly different: ex. the absence of a generalized version for the move, with specific functions for each means of transport.

The folder `working` contains the biggest version of the implementation that the planner is able to solve. In this version, the basic skeleton is implemented, with all means of transport correctly functioning and a simple and minimal problem to be solved. The domain and the problem have been modeled starting from the solution proposed during the lab sessions for the taxi exercise, using the means of transport (equivalent of taxi) and the vaccine boxes (equivalent of passengers) as agents.

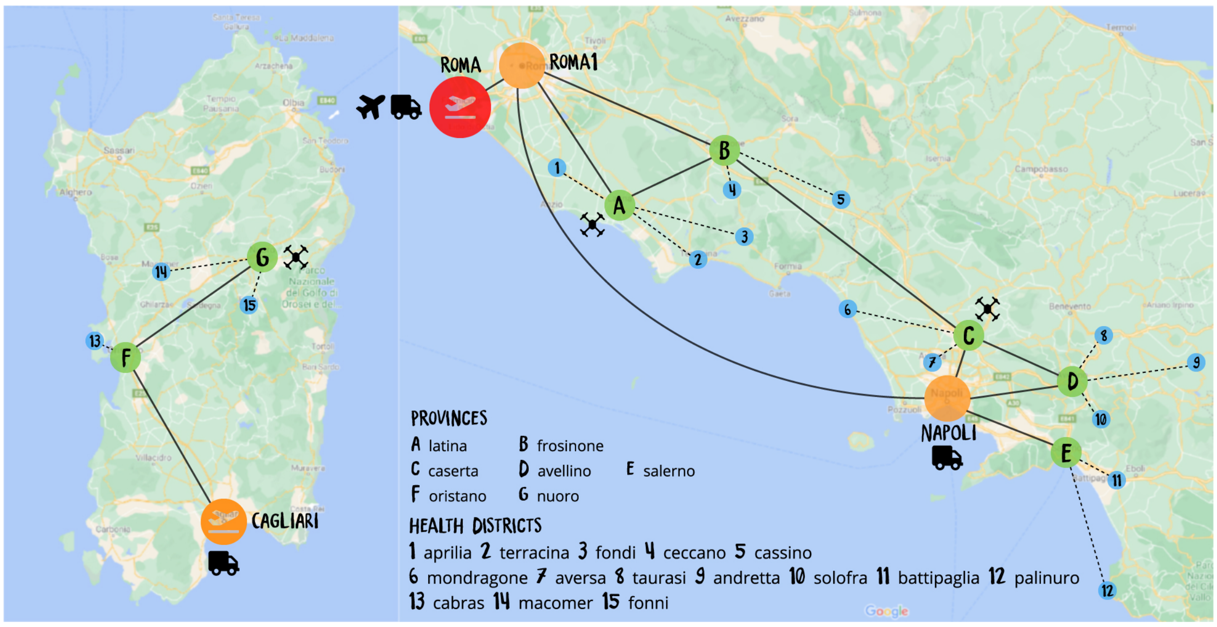
The folder `attemp` contains the full theoretical MA-PDDL version. Maplan doesn't support plenty of libraries useful to solve the problem (such as `:conditional-effects` or `:fluents`), so the attempt-version is not solvable by the planner. In this case, on the other hand, only the means of transport (drone, plane, truck) are considered as agents, while the vaccine box is defined as object. This solution was discarded with the working multi-agent implementation as it is not working but it would have been the final implementation choice.

5 Results

The original goal to develop an PDDL for a national vaccine distribution problem has been achieved. The planner is able to find a solution according to all the guidelines and constraints added in the domain file and specified in the assignment guidelines (although the problem dimension is not as close as the one requested due to computational limitations).

More, it is important to underline that the planner is not trying to find the optimal solution: this is why most of the time the capacity of a vehicle is not fully used (the planner tend to load only one vaccine box at the time), producing a ping-pong behaviour between locations. For example:

```
0:  LOAD VB15 PLANE1 ROMA
1:  MOVE PLANE1 ROMA CAGLIARI
...
42:  MOVE PLANE1 CAGLIARI ROMA
43:  LOAD VB14 PLANE1 ROMA
44:  MOVE PLANE1 ROMA CAGLIARI
```


ROMA
ROMA1
CAGLIARI
NAPOLI
A
B
C
D
E
F
G
PROVINCES
A latina
B frosinone
C caserta
D avellino
E salerno
F oristano
G nuoro
HEALTH DISTRICTS
1 aprilia 2 terracina 3 fondi 4 ceccano 5 cassino
6 mondragone 7 aversa 8 taurasi 9 andretta 10 solofra 11 battipaglia 12 palinuro
13 cabras 14 macomer 15 fonni