

The  package

LabourMarketAreas

Daniela Ichim

Istat, Department of Production Statistics

[ichim@istat.it](mailto:ichim@istat.it)

The story

Installing  LabourMarketAreas

Content and how to use it  
objects, constraints, possible extensions

Next steps

1. Istat releases LLMA since 1981.
2. 2013 Eurostat TF on Harmonised European Labour Market Areas.
3. April 2014: the CBS script “livework-cluster.r” has been made available to TF members.
4. April 2014: a script in Java developed by Dev-stat implementing the Coombes and Bond (2007) algorithm has also been provided to the TF members.

...

5. June 2014: Istat modified the original code by CBS in order to reproduce the output of the Java code. The script was shared with the TF members.
6. January 2016: Istat implemented a faster version of the R-script. The script was shared with the TF members.
7. June 2016: Istat releases the R-package LabourMarketAreas version 1.0.

Aim

May 2017: LabourMarketAreas - version 2.0

## Script

- a list of commands performing some actions
- functions have to be loaded
- different valid versions may exist even on the same PC
- the documentation is optional

## Package

- a set of functions
- different versions may still exist, but
  - at least a change of the package name/version is required
  - there is an warning about functions overlapping
- **the original version is always available. To everyone.**
- generally, the “official” version is stored on a repository
- the documentation is mandatory

It should be an  $\alpha$  – version.

We are available for any type of discussions/advice/problems etc

We tested it on Italian data, i.e. all we have. Other data, may raise other problems.

Please report (us) errors, failures and ... successes. We'll improve together the European LMAs.

The script will be no more maintained.

LabourMarketAreas is **just another R-package**.

Usual procedures for installation and removing.

Download Install R and Rstudio

<https://www.r-project.org/>

<https://www.rstudio.com/>

(both are free)

Usual procedures for installation and removing.

From the R console:

✓ Install

```
install.packages("YourPath\\LabourMarketAreas.zip")
```

✓ Uninstall

```
remove.packages("LabourMarketAreas",  
lib=~ /R/win-library/3.2")
```



```
> packageDescription("LabourMarketAreas")
Package: LabourMarketAreas
Type: Package
Title: LabourMarketAreas Version: 1.0
Date: 2016-06-07
Author: Daniela Ichim, Luisa Franconi, Michele
D'Alo', Guido van den Heuvel
Maintainer: Luisa Franconi <franconi@istat.it>
Description: Produces Travel-To-Work-Areas from
commuting flows data frame by means of the version of
the TTWA algorithm described in Coombes and Bond
(2008).
Depends: R (>= 3.01), data.table (>= 1.9.6)
License: GPL (>=2)
LazyData: true
Built: R 3.1.3; ; 2016-06-07 14:12:24 UTC; windows
-- File: PATH/LabourMarketAreas/Meta/package.rds
```

The package description may be modified only by the authors.

The package LabourMarketAreas was built and tested **ONLY** on Windows.

The package LabourMarketAreas is free, but there is a licence.

```
> RShowDoc("COPYING")
```

...

**This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.**

.....

This program is distributed in the hope that it will be useful, but **WITHOUT ANY WARRANTY**; without even the implied warranty of **MERCHANTABILITY** or **FITNESS FOR A PARTICULAR PURPOSE**. See the GNU General Public License for more details.

...

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that **any problems introduced by others will not reflect on the original authors' reputations**.



Package data.table – renders faster the implementation  
(IT 3 days to 3 hours; UK: 5 weeks to 1.5 day)

**Speed is important, but not everything.**

Anyway, speed allows us  
to test different parameters set  
early error discovery  
better analyze the input and output data



## LabourMarketAreas installation

- install data.table (from CRAN web-site)
- install LabourMarketAreas
- when it will be stored on CRAN, it will be sufficient to install LabourMarketAreas accepting the installation of dependencies



LabourMarketAreas inherits from data.table:

- syntax
- advantages and drawbacks

Syntax

- **LabourMarketAreas is implemented using the data.table syntax.** To modify any function in LabourMarketAreas, a minimum knowledge of the data.table syntax is required.
- Without data.table, the core functions of the algorithm will not run.

```
> library("LabourMarketAreas")
```

Automatically loads the package data.table

> ?LabourMarketAreas

## Description

Makes Travel-To-Work-Areas from commuting flow data ....

## Details

Package: LabourMarketAreas

Type: Package

Version: 1.0

Labour market areas (LMAs) are sub-regional geographical areas

## Author(s)

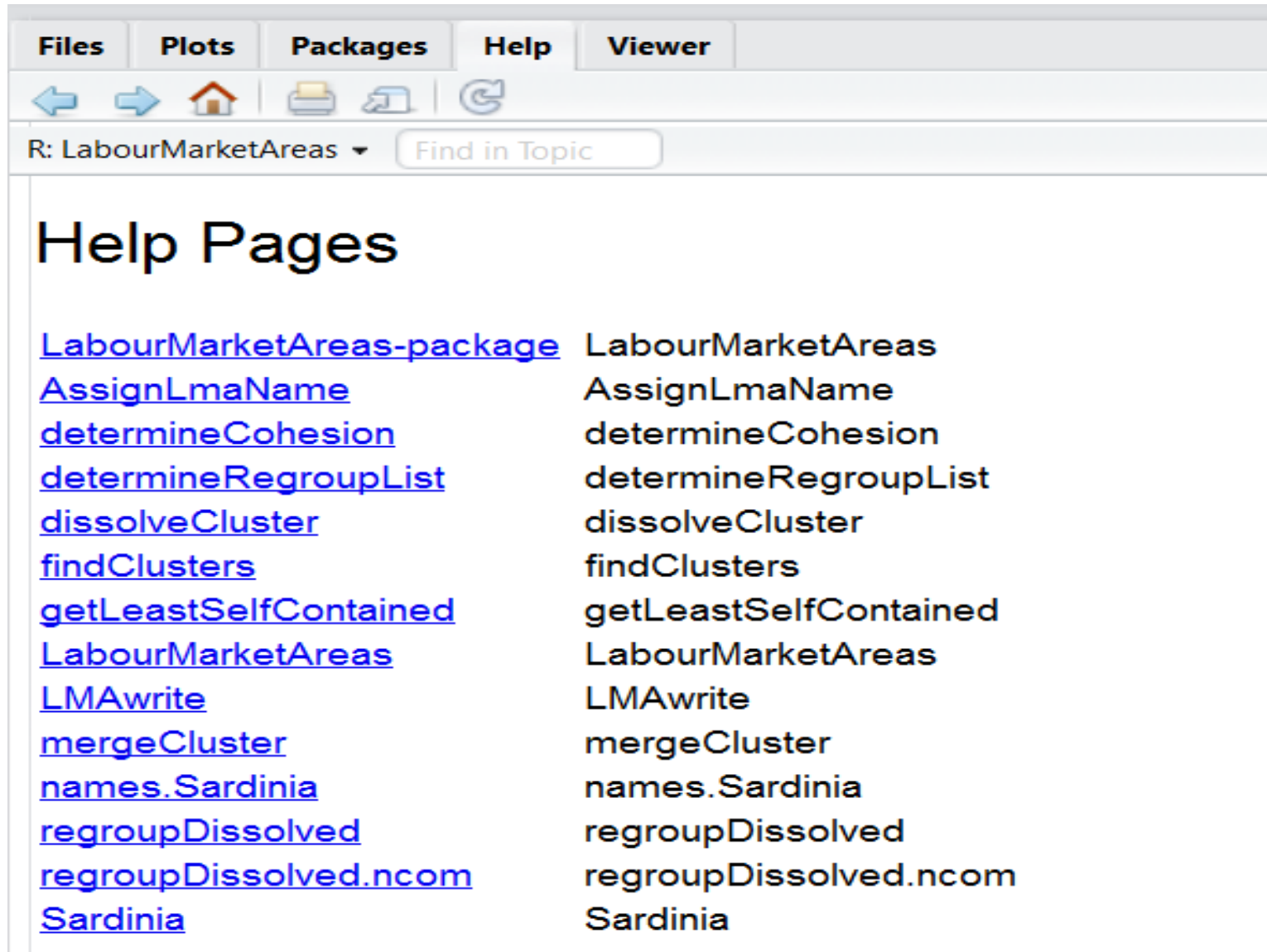
Daniela Ichim, Luisa Franconi, Michele D'Alo' and Guido van den Heuvel

**Maintainer:** Luisa Franconi <franconi at istat.it>

## References

[1] Coombes, M. e Bond, S. (2008).

.....



The screenshot shows a software interface with a menu bar containing 'Files', 'Plots', 'Packages', 'Help', and 'Viewer'. Below the menu bar is a toolbar with icons for back, forward, home, print, copy, and refresh. The address bar shows 'R: LabourMarketAreas' and a search box labeled 'Find in Topic'. The main content area is titled 'Help Pages' and lists various functions and objects from the 'LabourMarketAreas' package in two columns.

<a href="#">LabourMarketAreas-package</a>	LabourMarketAreas
<a href="#">AssignLmaName</a>	AssignLmaName
<a href="#">determineCohesion</a>	determineCohesion
<a href="#">determineRegroupList</a>	determineRegroupList
<a href="#">dissolveCluster</a>	dissolveCluster
<a href="#">findClusters</a>	findClusters
<a href="#">getLeastSelfContained</a>	getLeastSelfContained
<a href="#">LabourMarketAreas</a>	LabourMarketAreas
<a href="#">LMAwrite</a>	LMAwrite
<a href="#">mergeCluster</a>	mergeCluster
<a href="#">names.Sardinia</a>	names.Sardinia
<a href="#">regroupDissolved</a>	regroupDissolved
<a href="#">regroupDissolved.ncom</a>	regroupDissolved.ncom
<a href="#">Sardinia</a>	Sardinia



**Datasets:** [Sardinia](#), [names.Sardinia](#)

**The iterative algorithm:** [findClusters](#)

## Utility functions

algorithm: [determineCohesion](#),  
[determineRegroupList](#), [dissolveCluster](#),  
[getLeastSelfContained](#), [mergeCluster](#),  
[regroupDissolved](#), [regroupDissolved.ncom](#)

input:

output: [AssignLmaName](#), [LMAwrite](#)

Datasets: [Sardinia](#), [names.Sardinia](#)

> data(Sardinia)

> data(names.Sardinia)

Sardinia			
Filter			
	community_live	community_work	amount
1	90001	90001	251
2	90014	90001	1
3	90021	90001	4
4	90023	90001	1
5	90034	90001	2
6	90035	90001	1
7	90036	90001	2
8	90037	90001	2
9	90039	90001	1
10	90041	90001	1
11	90054	90001	1
12	90056	90001	5

names.Sardinia		
Filter		
	Code	com.name
1	90001	Aggius
2	90002	Alà dei Sardi
3	90003	Alghero
4	90004	Anela
5	90005	Ardara
6	90006	Arzachena
7	90007	Banari
8	90008	Benetutti
9	90009	Berchidda
10	90010	Bessude
11	90011	Bonnanaro
12	90012	Bono
13	90013	Bonnanaro

They are the flows occurred in Sardinia according to the Italian 2001 population census.

The entire dataset is available on the Istat web-site, but it is not part of the LabourMarketAreas package → the data should be first loaded in R.

They may be used to test and develop

- package functions
- other functions
- other tools

Fastest way:

In the R console:

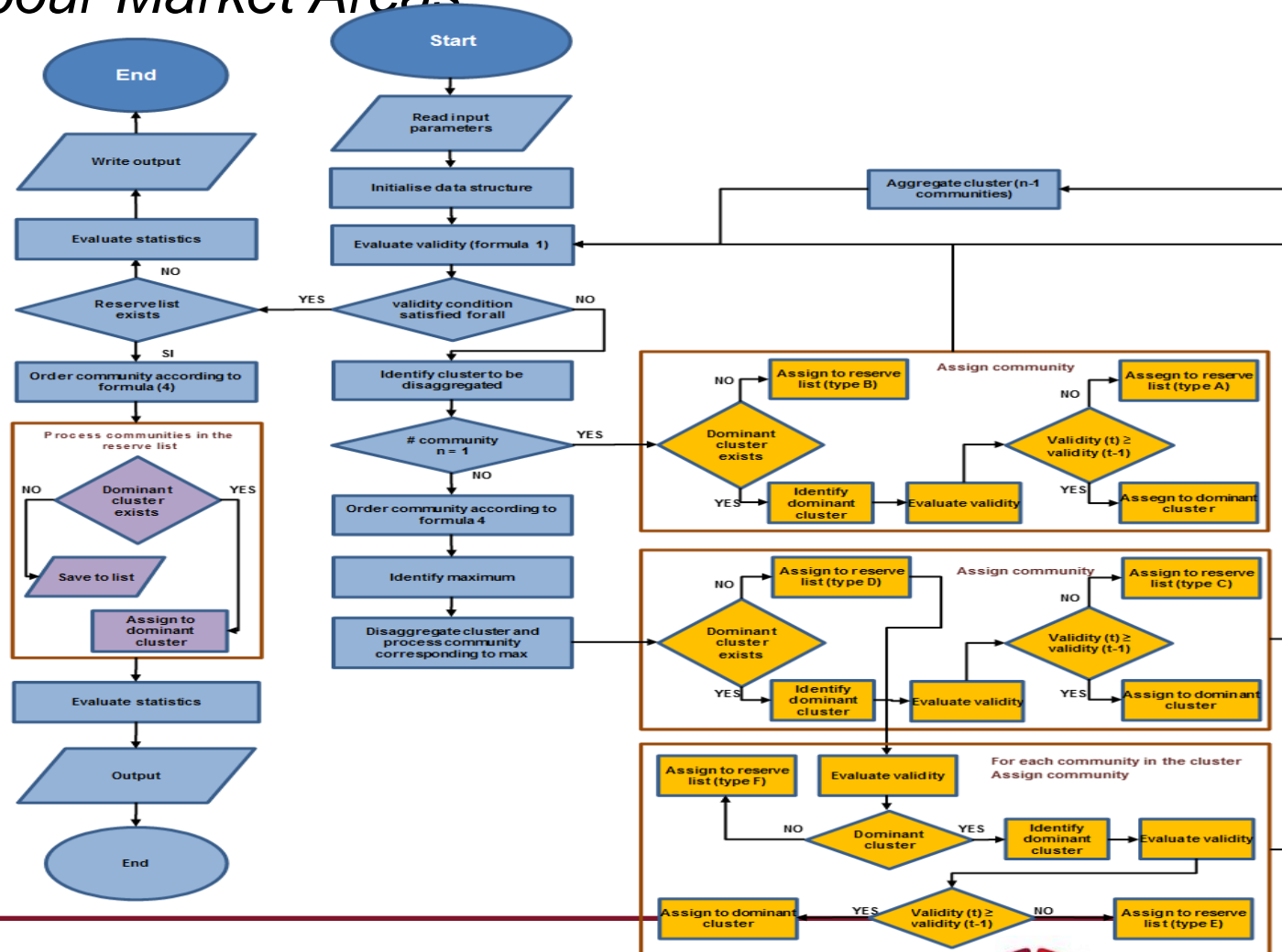
```
>library(data.table) # if not already loaded  
>mydata=fread("path\\filename.csv")
```

?fread

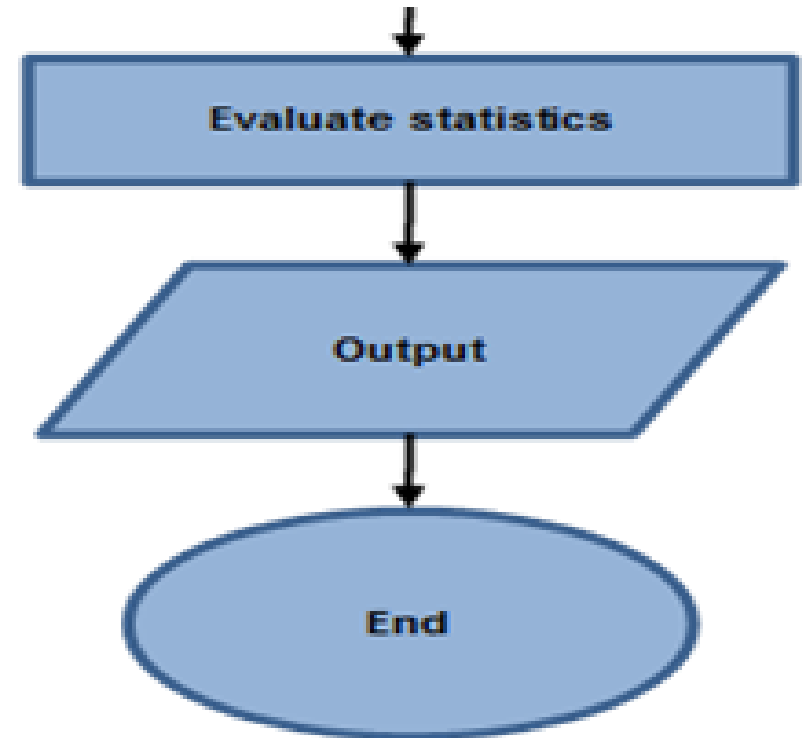
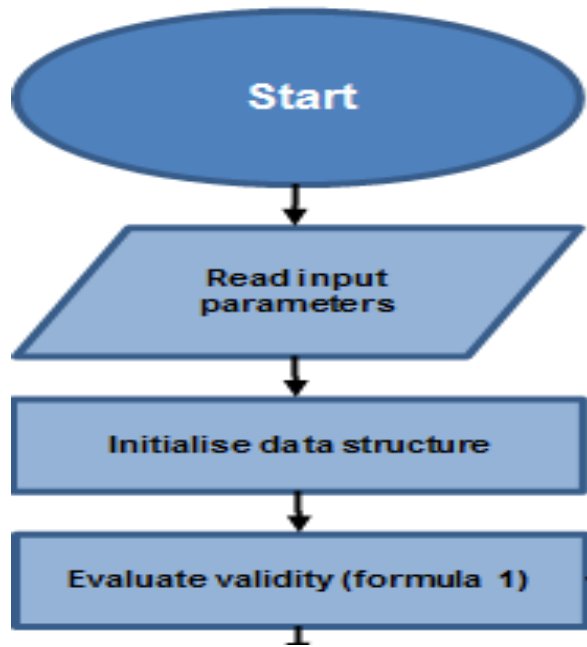
**fread** is a data.table function used for fast reading data from delimited files.

It automatically detects the separator and the header lines.

This function implements the iterative algorithm, as described in Franconi, D'Alo, Ichim, *Istat implementation of the algorithm to develop Labour Market Areas*



Includes some input and output utility functions.



## &gt; ?findClusters

R: findClusters

findClusters {LabourMarketAreas} R Documentation

## findClusters

### Description

This function builds labour market areas (LMAs) starting from commuting data between communities i.e. elementary zones (municipalities, census output areas, provinces, etc.). The function implements the algorithm described in Coombes and Bond (2008) and further detailed in Franconi and D'Alo' (2016).

### Usage

```
findClusters(LWCom, minSZ, minSC, tarSZ, tarSC, verbose = F, sink.output = NULL)
```

### Arguments

LWCom	data frame containing the commuting data. Each row corresponds to an observation i.e. a flow and each column corresponds to a variable. The variables are named: <code>community_live</code> , integer contains the id number of the elementary zone of residence, <code>community_work</code> , integer, contains the id number of the working community (elementary zone of work), <code>amount</code> , integer contains the number of employees commuting between the "community_live" and "community_work" (direction is important). Missing values (NAs) are not allowed. The community id must be positive. Only positive flows are present in the data frame.
minSZ	integer specifying the parameter containing the acceptable minimum size of an area. Must be positive.
minSC	numeric in the interval (0,1) specifying the parameter representing the acceptable minimum self containment of an area. Usually values range from 0.6 to 0.7.
tarSZ	integer specifying the parameter containing the target size of an area. It must be greater than minSZ.
tarSC	numeric in the interval (0,1) specifying the parameter representing the target self containment of an area. It must be greater than minSZ. Usually

*LWCom* = the flows data.frame/data.table

*minSZ* = a numeric value

*tarSZ* = a numeric value

**NO DEFAULT VALUE!**

*minSC* = a numeric value

*tarSC* = a numeric value

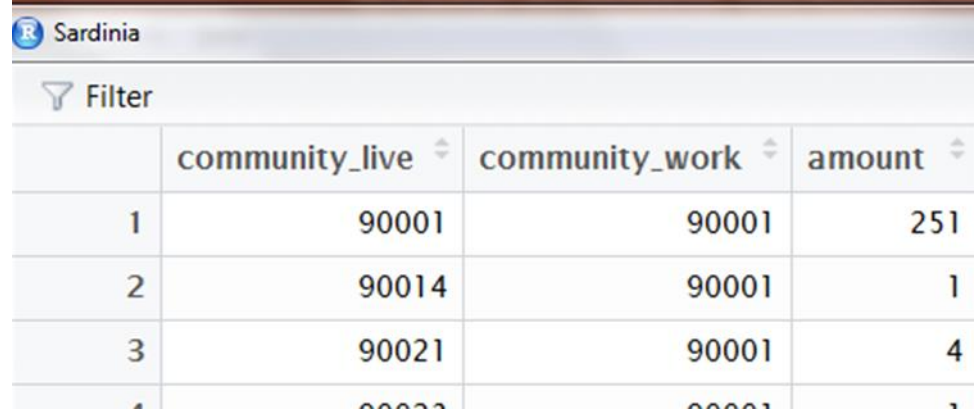
verbose = a logical value

sink.output = a character value

**An error is produced when the above rules are not satisfied.**



LWCom



The screenshot shows a data table with the following structure:

	community_live	community_work	amount
1	90001	90001	251
2	90014	90001	1
3	90021	90001	4
4	90000	90001	1

The column names are **ESSENTIAL**:  
*community\_live*, *community\_work*, *amount*.

The order is not important.

**NO MISSING VALUES.**

*community\_live* and *community\_work* **MUST** be positive integer vectors.

*amount* may be whatever number.

In future versions, some controls on the previous rules will be added:

- input variables type
- names of the input variables
- missing values
- integer values of *community\_live*, *community\_work*

In future versions, some statistics on the input flows may be added (even outside the function findCluster):

- number of communities
- number of traits
- mean number of input/output traits/flows per community

```
out=findClusters(Sardinia,1000,0.6667,10000,0.75)
```

```
LWCom <- data.table(LWCom)    ##LWCom is sardinia in this example
```

Contrary to the script, the input flows are NOT read inside the findClusters function.

The input flows object must exist before the function findClusters is used.

There is no utility for this action (in this version).

It doesn't matter how you provide LWCom, it will be transformed in a data.table object.

```
...  
LWCom = LWCom[order(community_live)]  
  
LIST.COM = data.table(c(LWCom[,community_live],  
LWCom[,community_work]))  
LIST.COM = unique(LIST.COM)[order(v1)]
```

The list/set of communities is derived from the input flows object. It is not possible to read the list of communities from an external source.

Hence it is assumed that when a community is not registered in the input flows object (either as *community\_live* or *community\_work*), it should not be considered by the algorithm. (the algorithm is self-contained)

...

```
residents = setcolorder(LWCom[, .(residents =  
sum(amount)), by = .(Code = community_live)],  
c("residents", "Code"))  
residents = merge(residents, LIST.COM, by.x = "Code",  
by.y = "v1", all = T)  
residents[is.na(residents), `:=`(residents, 0)]
```

... and a similar block for “workers”.

For each community, the residents and the workers are computed from the input flows object. Contrary to the script, there is no way to load this information from an external source. (the algorithm is self-contained).

...

Create the first piece of output

`zero.list`

?findClusters #/ section value

`zero.list`

**List of four objects:** they contain information on communities (elementary areas or municipalities) that could not be processed by the algorithm for various reasons; either the number of residents is 0 or the number of workers/jobs is 0 or the community has no interaction with any other community. In such cases the algorithm eliminates the communities from the initial list and let the user the choice to allocate them at a later stage.

**Communities:** integer containing the ids of the communities that could not be processed by the algorithm.

**LWCom:** data frame containing the flows involving the above communities. The data frame is in a sense a subset of the initial data frame containing the commuting data. Its variables are `community_live`, integer containing the id of the community where the employees live, `community_work`, integer containing the id of the community where the employees work, `amount`, numeric, containing the number of employees commuting from `community_live` to `community_work`.

**Residents:** data frame containing the variables `Code`, integer representing the id of the community and `residents`, integer representing the number of employees who are resident in the community.

**Workers:** data frame containing the variables `Code`, integer representing the id of the community and `workers`, integer representing the number of employees who work in the community.

... in practice, IDs, flows, residents and workers of whatever community having no residents and/or workers.

A warning is printed on the console. `zero.list` object is included in the output.

...

**Fictitious community and its fictitious flows.**

```
fict.community = LIST.COM[, max(V1) * 10]
```

```
LWCom= rbind(LWCom,data.table(matrix(rep(fict.community,  
ncol(LWCom)), 1)), use.names = F)
```

...

The fictitious community should not exist in the LIST.COM.  
We chose to multiply the maximum community id by 10.

The community\_live and community\_work codes (ids) **MUST be numerical values**. As all joins are made by these ids, they should be integer values.

There is no check on these rules in version 1.0. If needed, **in future versions it will be added**.

## Fictitious community and its fictitious flows.

### Not evaluated/implemented:

The fictitious cluster is a cluster. It is a particular cluster. In the algorithm, the fictitious community is never “dissolved”. It might only “receive” some communities.

We called it “fictitious” because it was created inside the findClusters function, but it may be loaded from an external source. In such situation, it would be **a cluster that is never dissolved**. This option might allow you to find partitions containing a pre-defined area. (see also later)

In version 1.0, a single fictitious community is considered. If such an extension is valuable, different “do-not-dissolve” starting communities might be dealt with (version XXX.0).

---



We must know what we want.

The core output is a list of several `data.frames/data.tables` grouped in lists.

The first list called *clusterData*:

- clusterList
- LWClus
- marginals

The remaining pieces of the output contain information about the application of the algorithm (`zero.list`, `reserve.list`, etc).

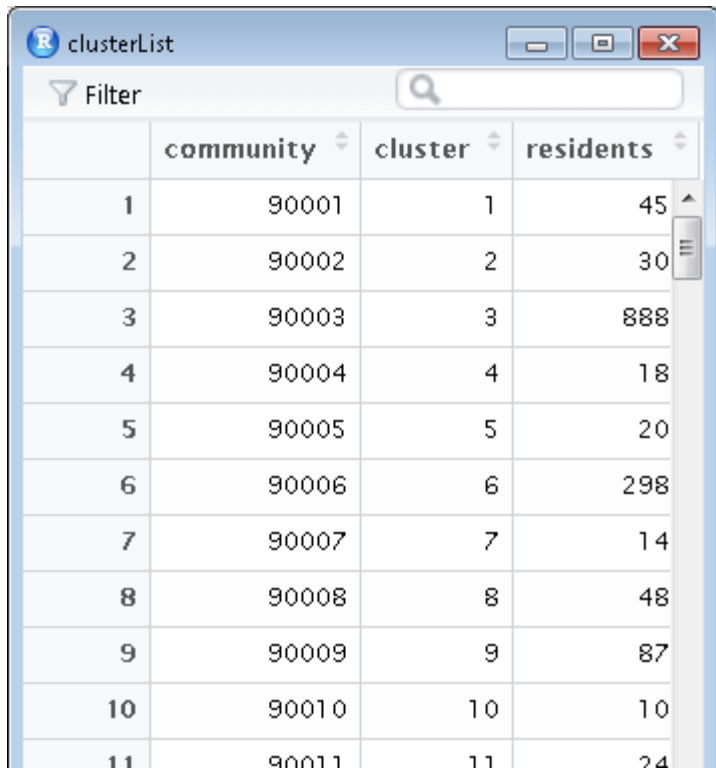
The output is a list because in R, functions may have an unique output.

# findClusters – initialize output

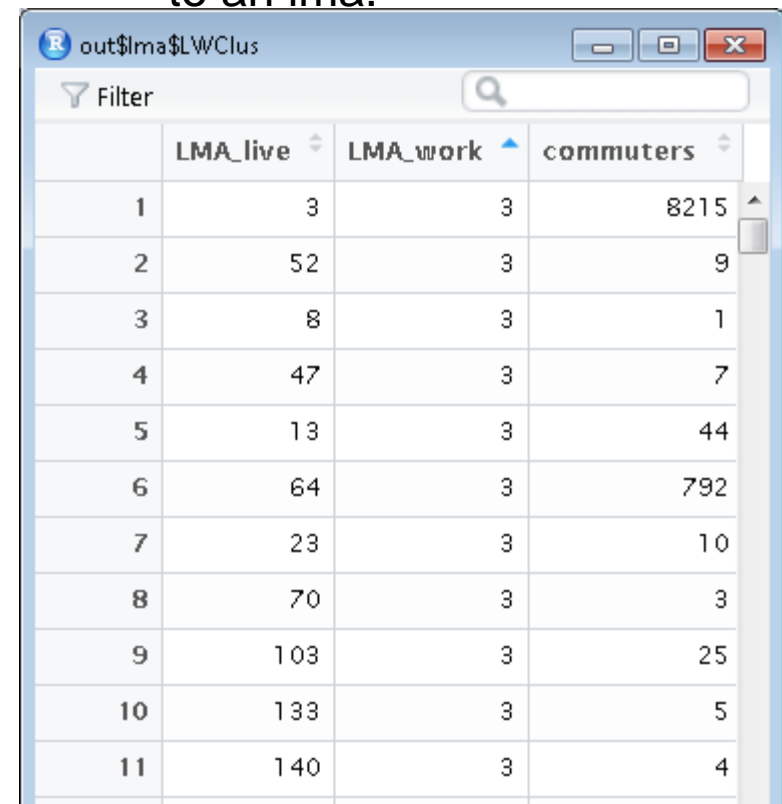
**clusterList** - allocation of each community to the corresponding lma.

Initially, each community is a distinct lma.

At the end, it contains the allocation of each community to an lma.



	community	cluster	residents
1	90001	1	45
2	90002	2	30
3	90003	3	888
4	90004	4	18
5	90005	5	20
6	90006	6	298
7	90007	7	14
8	90008	8	48
9	90009	9	87
10	90010	10	10
11	90011	11	24

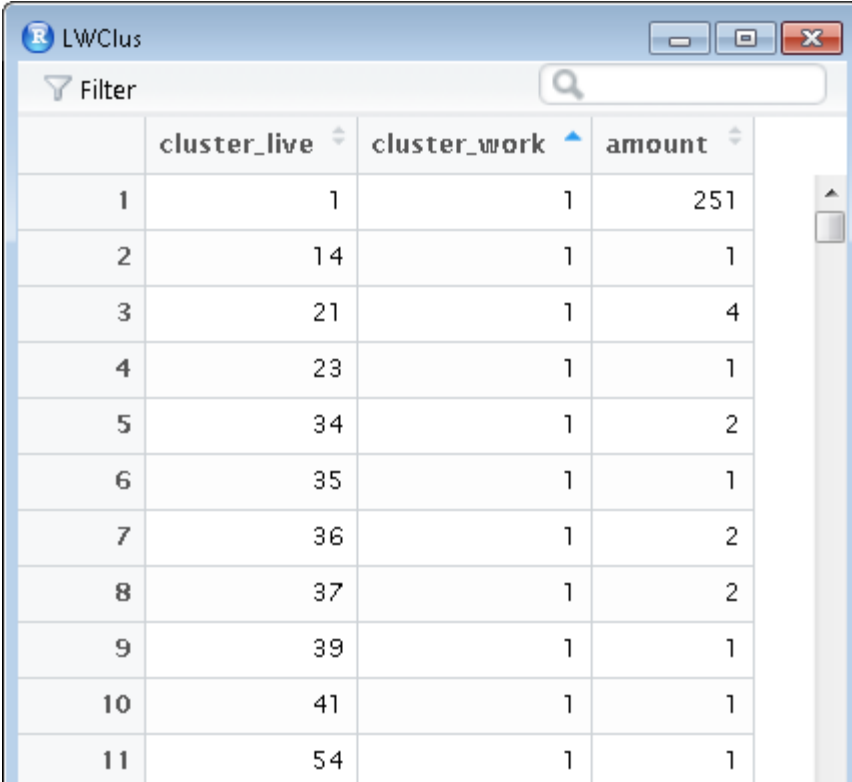


	LMA_live	LMA_work	commuters
1	3	3	8215
2	52	3	9
3	8	3	1
4	47	3	7
5	13	3	44
6	64	3	792
7	23	3	10
8	70	3	3
9	103	3	25
10	133	3	5
11	140	3	4

# findClusters – initialize output

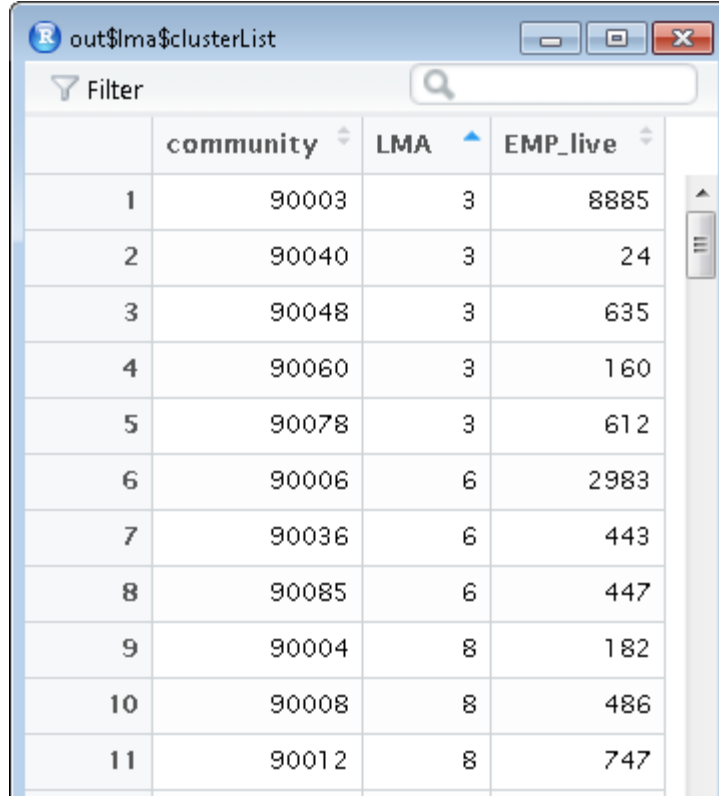
**LWClus**- flows between LMAs.

Initial



	cluster_live	cluster_work	amount
1	1	1	251
2	14	1	1
3	21	1	4
4	23	1	1
5	34	1	2
6	35	1	1
7	36	1	2
8	37	1	2
9	39	1	1
10	41	1	1
11	54	1	1

Final

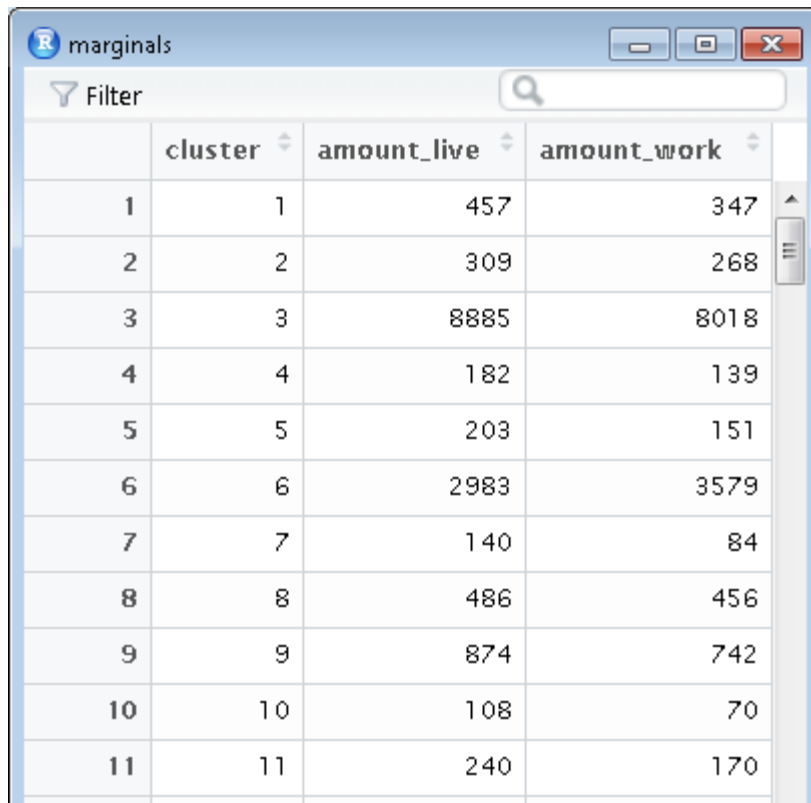


	community	LMA	EMP_live
1	90003	3	8885
2	90040	3	24
3	90048	3	635
4	90060	3	160
5	90078	3	612
6	90006	6	2983
7	90036	6	443
8	90085	6	447
9	90004	8	182
10	90008	8	486
11	90012	8	747

# findClusters – initialize output

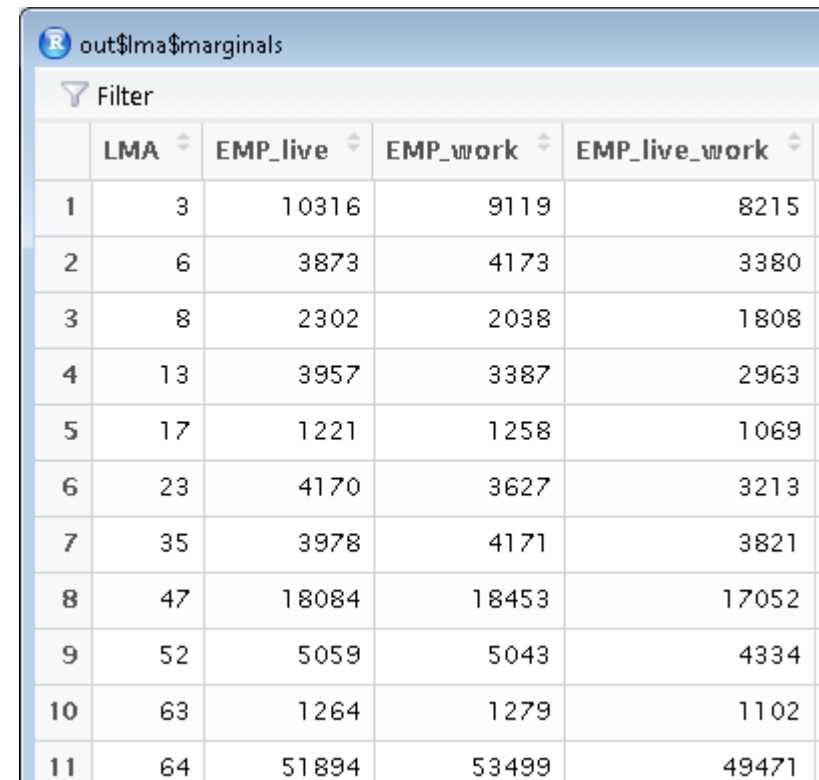
**marginals** - main characteristics of the LMAs.

Initial



	cluster	amount_live	amount_work
1	1	457	347
2	2	309	268
3	3	8885	8018
4	4	182	139
5	5	203	151
6	6	2983	3579
7	7	140	84
8	8	486	456
9	9	874	742
10	10	108	70
11	11	240	170

Final



	LMA	EMP_live	EMP_work	EMP_live_work
1	3	10316	9119	8215
2	6	3873	4173	3380
3	8	2302	2038	1808
4	13	3957	3387	2963
5	17	1221	1258	1069
6	23	4170	3627	3213
7	35	3978	4171	3821
8	47	18084	18453	17052
9	52	5059	5043	4334
10	63	1264	1279	1102
11	64	51894	53499	49471

# findClusters – the iterative process

The elements of the first list are iteratively modified **simultaneously**.

	community	cluster	residents
1	90001	1	45
2	90002	2	30
3	90003	3	888

	cluster_live	cluster_work	amount
1	1	1	25
2	14	1	
3	21	1	

	cluster	amount_live	amount_work
1	1	457	347
2	2	309	268

	LMA_live	LMA_work	commuters
1	3	3	8215
2	52	3	9
3	8	3	1
4	47	3	7

	community	LMA	EMP_live
1	90003	3	8885
2	90040	3	24
3	90048	3	635

	LMA	EMP_live	EMP_work	EMP_live_work
1	3	10316	9119	8215
2	6	3873	4173	3380
3	8	2302	2038	1808
4	13	3957	3387	2963
5	17	1221	1258	1065
6	23	4170	3627	3213

The first list called *clusterData*:

- clusterList
- LWClus
- marginals

At this stage, no missing values are allowed: rows containing missing values in either column *community\_live*, *community\_work*, *amount* are cancelled.

The entire input data is used: in this version, there is NO way to subset it.

Possible ways of subsetting:

- *amount* (flow) above/below a given threshold
- selected communities
- self-flows ( $community\_live=community\_work$ )
- commuters to/from abroad

...

```
counter <- 1  
reserve.list = list()  
counter.list = 1  
ComNotAssigned.list = list()
```

### reserve.list

Communities that do not improve the value of the validity when assigned to the dominating cluster or that do not have a dominating cluster are put into the reserve list.

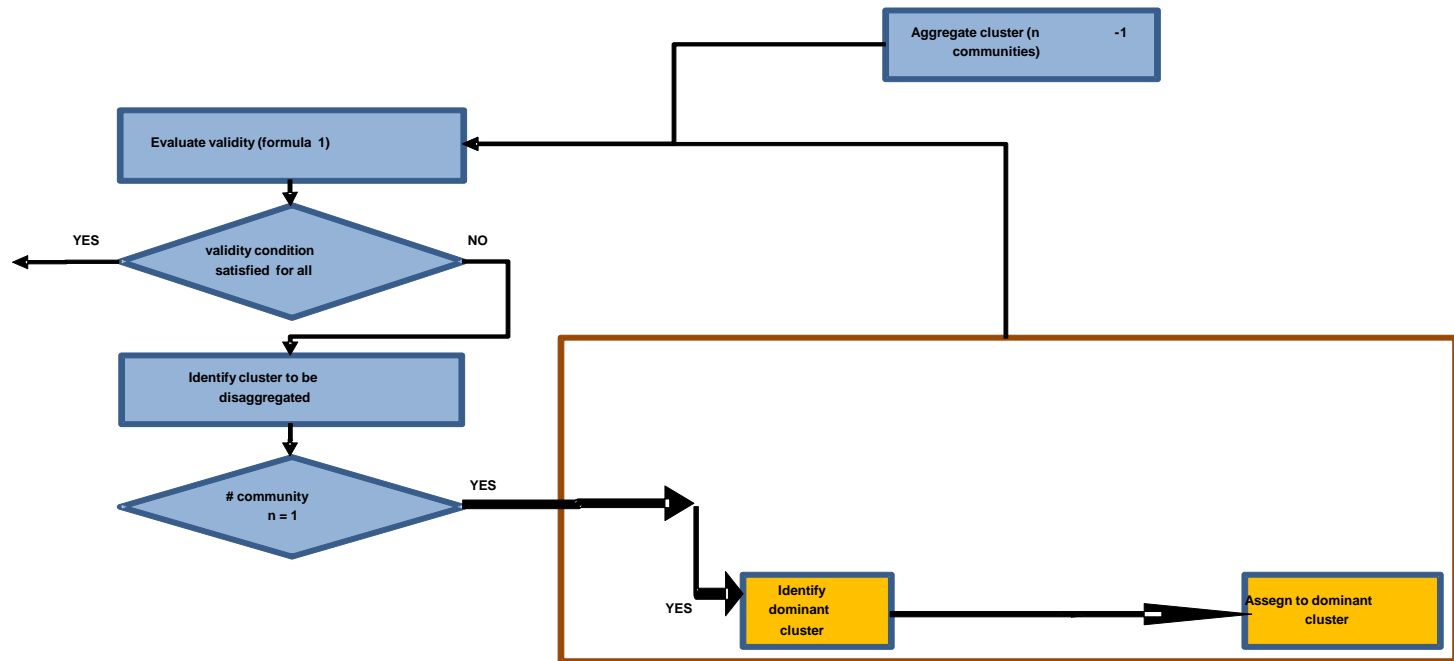
### ComNotAssigned.list

Components: integer containing the id of the community in the reserve list that the algorithm was not able to assign to any existing cluster. One list for each of such community. NULL otherwise.

```
repeat{
```

```
  ...
}
```

STOP

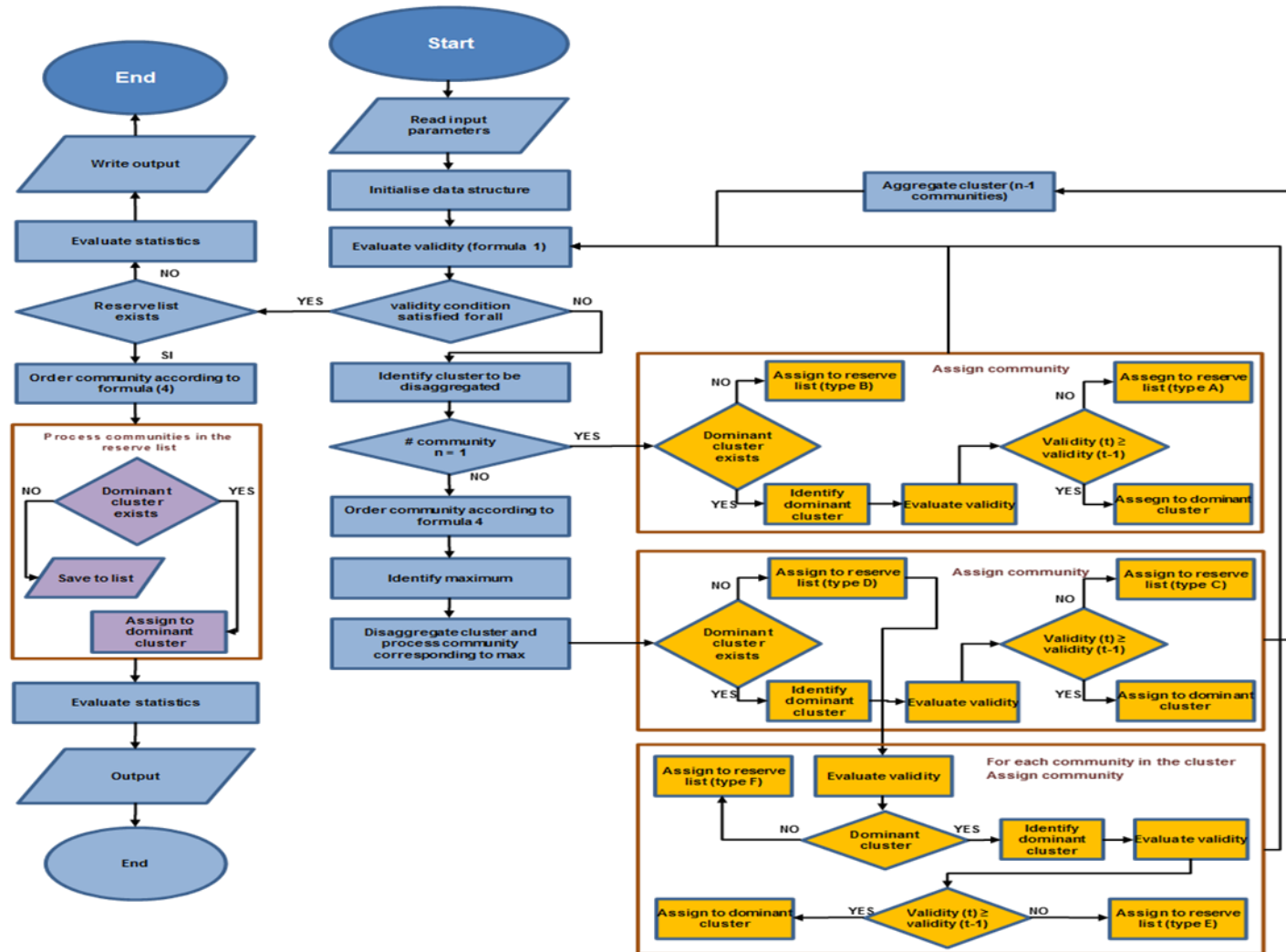


Repeat

1. Find the least validity cluster.
  2. Dissolve it
  3. Assign each community to its dominant cluster.
- Until the validity condition is satisfied.



# findClusters – the iterations



# findClusters – the iterations

```
repeat{  
  ...  
}
```

At each iteration, the input is a *clusterData* structure.  
At each iteration, the output is a *clusterData* structure

The image shows three R Studio window screenshots displaying data tables. Each window has a 'Filter' search bar at the top.

	community	LMA	EMP_live
1	90003	3	8885
2	90040	3	24
3	90048	3	635
4	90060	3	160
-	-----	-	---

	LMA_live	LMA_work	commuters
1	3	3	8215
2	52	3	9
3	8	3	1
4	47	3	7
-	--	~	..

	LMA	EMP_live	EMP_work	EMP_live_wor
1	3	10316	9119	
2	6	3873	4173	
3	8	2302	2038	
4	13	3957	3387	
5	17	1221	1258	
6	23	4170	3627	

Repeat

1. Find the least validity cluster.
2. Dissolve it
3. Assign each community to its dominant cluster.

Until the validity condition is satisfied.

We need:

- a validity function
- a strategy to dissolve clusters
- a way to identify a dominant cluster

$$\left[ 1 - \left( 1 - \frac{\min SC}{\text{tar} SC} \right) \cdot \max \left( \frac{\text{tar} SZ - SZ}{\text{tar} SZ - \min SZ}, 0 \right) \right] \cdot \left[ \frac{\min(SC, \text{tar} SC)}{\text{tar} SC} \right]$$

getLeastSelfContained(LWClus, marginals, minSZ, minSC, tarSZ, tarSC)

- excludes flows to/from unknown clusters (missing)
- excludes flows to/from cluster zero
- the validity of cluster zero is always equal to 1 (that's why it is never dissolved)
- computes the validity of each cluster
- returns the cluster corresponding to the minimum validity value and its validity
- returns the computations performed for each cluster

# findClusters – validity function

```
obj= getLeastSelfContained(LWClus, marginals, minsZ,  
minsC, tarsZ, tarSC)
```

Console ~/ ↻

```
> obj=getLeastSelfContained(LWClus, marginals, minsZ, minsC, tarsZ, tarSC)
```

```
> str(obj)
```

```
List of 2
```

```
$ :Classes 'data.table' and 'data.frame':      1 obs. of  2 variables
```

```
..$ cluster : num 179
```

```
..$ validity: num 0.201
```

```
..- attr(*, ".internal.selfref")=<externalptr>
```

```
$ :Classes 'data.table' and 'data.frame':     379 obs. of  9 variables
```

```
..$ cluster      : num [1:379] 1 2 3 4 5 6 7 8 9 10 ...
```

```
..$ amount_live: num [1:379] 457 309 8885 182 203 ...
```

```
..$ amount_work: num [1:379] 347 268 8018 139 151 ...
```

```
..$ amount      : num [1:379] 251 240 7082 111 115 ...
```

```
..$ msc         : num [1:379] 0.549 0.777 0.797 0.61 0.567 ...
```

```
..$ Y          : num [1:379] 0.732 1 1 0.813 0.755 ...
```

```
..$ sizeMeasure: num [1:379] 1.06 1.077 0.124 1.091 1.089 ...
```

```
..$ Z          : num [1:379] 0.882 0.88 0.986 0.879 0.879 ...
```

```
..$ validity   : num [1:379] 0.727 0.99 1.109 0.804 0.747 ...
```

```
..- attr(*, ".internal.selfref")=<externalptr>
```

```
..- attr(*, "index")= atomic (0)
```

```
..- attr(*, "cluster")= int [1:379] 379 1 2 3 4 5 6 7 8 9
```

# findClusters – validity function

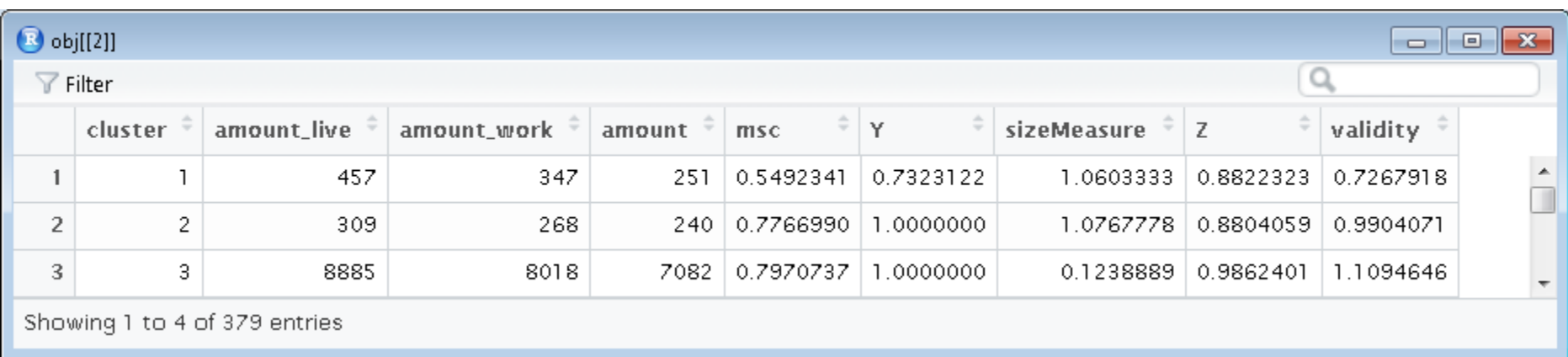
```
obj= getLeastSelfContained(LwClus, marginals, minSZ,  
minSC, tarSZ, tarSC)
```

```
> obj[[1]]
```

```
cluster validity
```

```
1: 179 0.2007328
```

```
> View(obj[[2]])
```



obj[[2]]

Filter

	cluster	amount_live	amount_work	amount	msc	Y	sizeMeasure	Z	validity
1	1	457	347	251	0.5492341	0.7323122	1.0603333	0.8822323	0.7267918
2	2	309	268	240	0.7766990	1.0000000	1.0767778	0.8804059	0.9904071
3	3	8885	8018	7082	0.7970737	1.0000000	0.1238889	0.9862401	1.1094646

Showing 1 to 4 of 379 entries

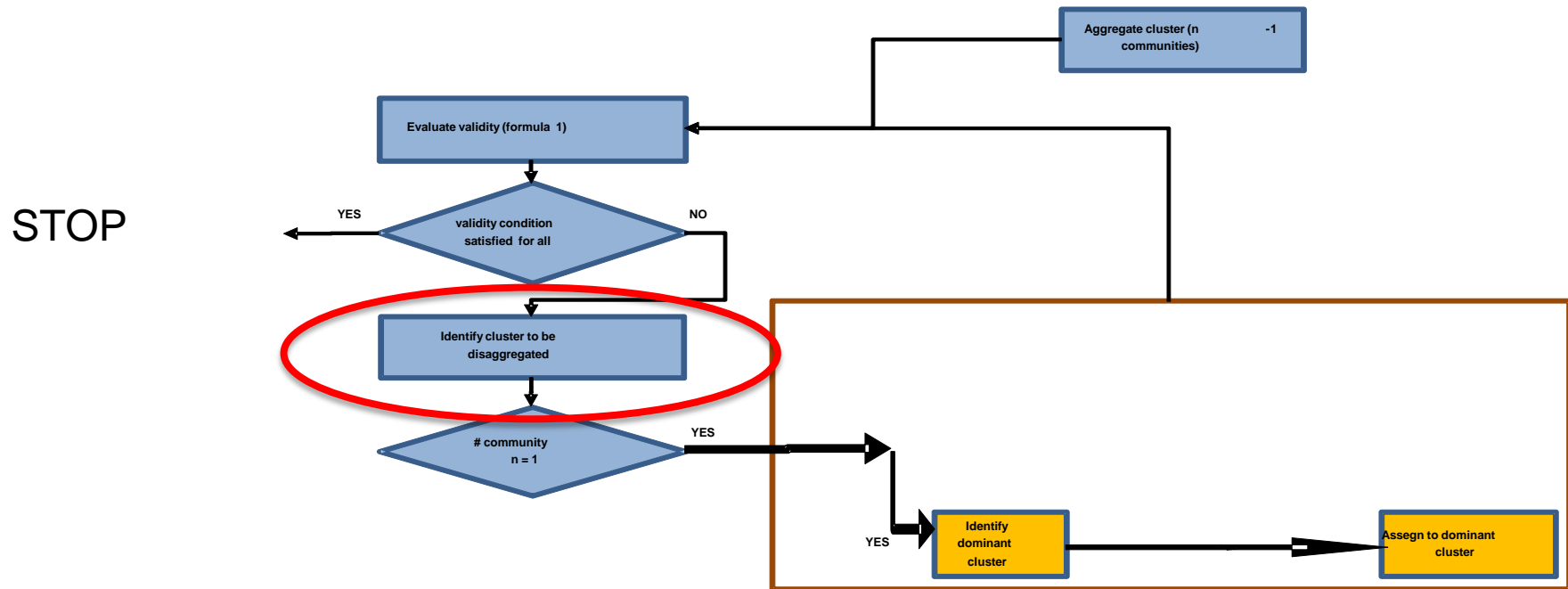
```
obj= getLeastSelfContained(LWClus, marginals, minSZ,  
minSC, tarSZ, tarSC)
```

A **unique** minimum is returned.

Multiple minimum values may be dealt with, but different distinct clusters should be simultaneously dissolved (how to identify the dominant clusters?). Is this a common or rare situation?

At each iteration, the validity function is computed for each cluster.

An improvement (smaller datasets) could be achieved by evaluating the validity **ONLY** for the clusters involved in the latest dissolving/assignment operations.



If there are multiple communities in the minimum validity cluster, they are ordered in decreasing order by  $NoToFrom + residents - live\_work$  (incoming workers but not from the same cluster)



Repeat

1. Find the least validity cluster.
  2. **Dissolve it**
  3. Assign each community to its dominant cluster.
- Until the validity condition is satisfied.

```
dissolveCluster(clusterData, cluster, LWCom)
```

This function dissolves a selected cluster into its constituent communities. Such communities are given temporary cluster IDs: **negative integers** (that's why the original IDs must be positive integer numbers).

Input: a *clusterData* structure

Output: a *clusterData* structure (some clusters have negative IDs)

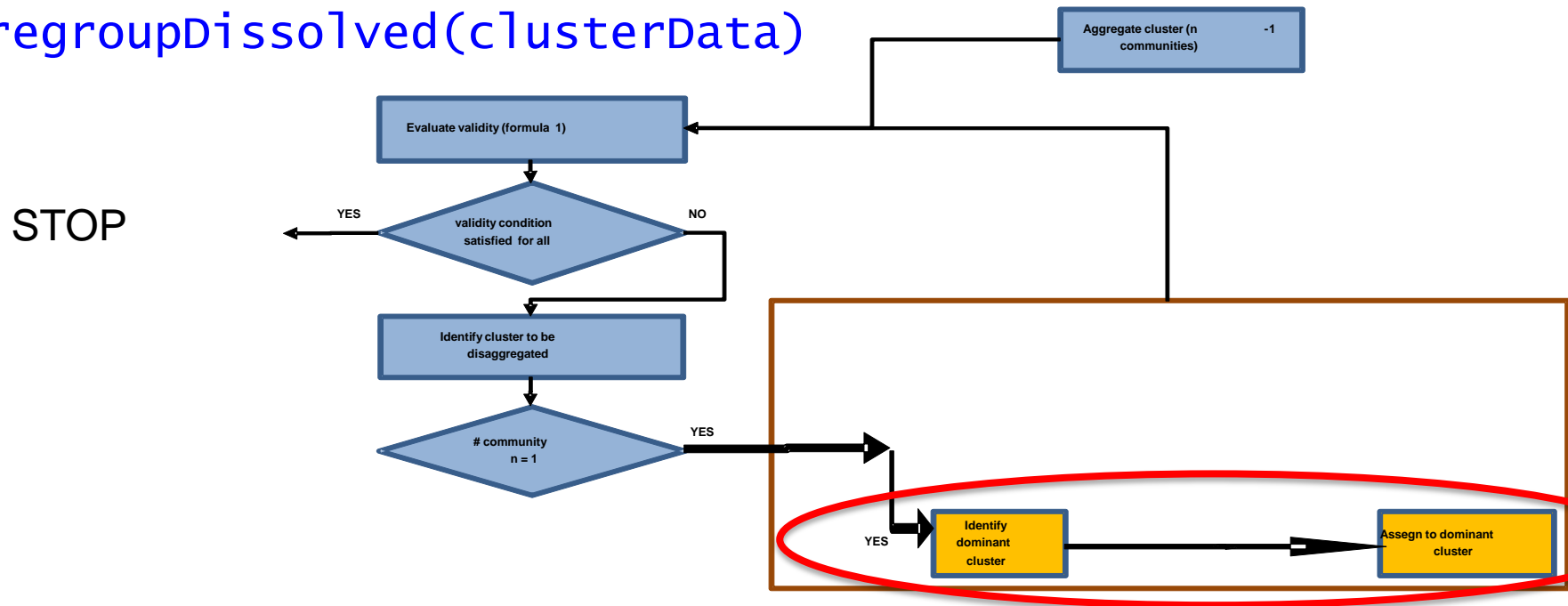
# findClusters – regroupDissolve

Repeat

1. Find the least validity cluster.
2. Dissolve it
3. Assign each community to its dominant cluster.

Until the validity condition is satisfied.

regroupDissolved(clusterData)



## regroupDissolved

Input: a *clusterData* structure

Output: a *clusterData* structure (no clusters with negative IDs)  
or 1 (the reserve list is the only candidate cluster)

Two internal functions are called:

determineRegroupList

determineCohesion  $L_{hk} = \left[ \frac{(f_{hk})^2}{(R_h W_k)} \right] + \left[ \frac{(f_{kh})^2}{(R_k W_h)} \right]$

As for the validity function, an **unique** dominant cluster is identified.

Multiple maximum values may be dealt with,  
but a further criteria is strictly required

Is this a common or rare situation?

## regroupDissolved

Come back to the fictitious community:

If there is no dominant cluster, the community is assigned to the reserve.list. It means that the reserve.list is the last option.

Not implemented:

If the fictitious community (reserve.list) is a pre-defined cluster which is never dissolved and it is always the last option, this might be an option to deal with very large communities (Paris, London ... ):

a new community is assigned to these large communities (pre-defined areas) ONLY if there is no other option (link to other communities/clusters).

In two words:

Step 0. Define a validity function. Define a cohesion function. Each community is a cluster.

Step 1. Identify the minimum validity cluster  $C$  not satisfying the validity condition.

Step 2 For each community  $c_i$  in  $C$ ,

Step2.1 Find the dominant cluster  $D$ .

Step 2.2 Temporarily assign the community to  $D$ .

Step 2.3 If the validity of  $D$  is improved w.r.t. the previous situation, the assignment is definitive. Otherwise,  $c_i$  is saved into the reserve list

In case of multiple communities in the minimum validity cluster C

- consider the communities in the dissolved cluster C, use the order defined above (NoToFrom + residents - live\_work ),
- the first community is assigned to a cluster D.
- If  $D=0$  (reserve list), then all the other communities in the dissolved cluster are assigned to their corresponding clusters (they may be different clusters --- there is no constraint).
- If  $D \neq 0$ , the dissolved cluster C is regrouped. The regrouped cluster will contain all the communities, except the first one in C (the one which was already assigned).

Inside `findClusters`

`AssignLmaName`

`writeLma`

Inside `findClusters`

At the end of the algorithm:

1. - eliminate the fictitious community (not the `reserve.list`)
2. - save the partition into `clusterDataBeforeZeroCluster`
3. - try to assign the communities in the `reserve.list` to “standard” clusters  $\rightarrow$  the validity of such clusters is reduced
4. - save the partition `clusterData`
5. - compute some statistics for both `clusterData`
  - - the validities of the clusters
  - - self-containment (supply and demand)
1. - assign final names to variables
2. - computes the number of communities in each cluster



## AssignLmaName

This function assigns names to the labour market areas given their codes.

The lma name corresponds to the community name (elementary area or municipality) having the highest number of jobs among all the communities in the lma.

The community names are in lowercase, except the the first letter.

The lma names are in uppercase.

See `names.Sardinia` for example.

## writeLma

This function saves the lists composing the output of the Ima package into separate data frames as `.RData`.

The files are saved in the `path_wd` directory. (your working directory)

The main output, the characteristics of the created labour market areas and the characteristics of the areas before the final assignment of the reserve list - are also saved in a `.csv` file.

In the next future:

- parameters should be included in the output
- more statistics on the output partition
  - numbers of workers (min,max,mean)
  - number of residents (min,max,mean)
  - number of traits (min,max,mean)
  - modularity
- statistics on the reserve.list

Find.cluster argument  
`sink.file`

character string containing the name of the .txt file that will contain optional information for each iteration of the algorithm.

We mainly used this option in the development phase.

If you think it could be useful, we might enrich it. Anyway, the information about each iteration is saved.

Finally, NO intermediate output (*clusterData* structures) is saved.

There is no possibility to start from a pre-defined structure.  
(it might be useful in case of error)

Console ~/

```
Error in vecseq(f__, len__, if (allow.cartesian || notjoin ||  
!anyDuplicated(f__, :
```

Join results in 260 rows; more than 218 = nrow(x)+nrow(i).

check for duplicate key values in i each of which join to the same group in x over and over again. If that's ok, try `by=.EACHI` to run `j` for each group to avoid the large allocation. If you are sure you wish to proceed, rerun with `allow.cartesian=TRUE`. Otherwise, please search for this error message in the FAQ, wiki, stack overflow and datatable-help for advice.

It seems an error, but it is an warning.

This is a `data.table` feature.

If you look into the `data.table` help package, we'll discover that:

The package `data.table` especially introduces this kind of error instead of an warning:

`in order to make efficient joins, particular situations are warned in this way.`

In version 1.0 ,we avoided this situation by using the option «allow.cartesian=T» where necessary.

Anyway, it is not even a real cartesian product: it just allows us to perform the join we need.

?data.table

Argument `allow.cartesian`:

«The word 'cartesian' is used loosely in this context»

There might be other ways to deal with this warning: we'll and eventually implement in future versions.

Invalid `.internal.selfref` detected and fixed by taking a (shallow) copy of the `data.table` so that `:=` can add this new column by reference. At an earlier point, this `data.table` has been copied by R (or been created manually using `structure()` or similar). Avoid `key<-`, `names<-` and `attr<-` which in R currently (and oddly) may copy the whole `data.table`. Use `set*` syntax instead to avoid copying: `?set`, `?setnames` and `?setattr`. Also, in R<=v3.0.2, `list(DT1,DT2)` copied the entire DT1 and DT2 (R's `list()` used to copy named objects); please upgrade to R>v3.0.2 if that is biting. If this message doesn't help, please report to `datatable-help` so the root cause can be fixed.



Don't worry!

In R, the command

```
names(MyData.frame)
```

makes a copy of the object MyData.frame

In data.table, this may be avoided by using other tools.

It means: the implementation could be even faster than it is in version 1.0.

In rm(clusterList) : object 'clusterList' not found

In rm(LWclus) : object 'LWclus' not found

In rm(marginals) : object 'marginals' not found

In rm(index) : object 'index' not found

Don't worry!

In order to be sure that the objects are created correctly at each iteration, sometimes we removed them too many times.

It's boring, but we'd prefer them in this way. No improvement could be achieved (removing nothing costs nothing!)

LabourMarketAreas version 1.0 implements

- an unique validity function
- an unique cohesion function

In future versions, more validity and cohesion functions may be added.

```
out1=findClusters(LWCom=Sardinia,minsZ=1000,minSC=0.6667,tarsZ=10000,tarsC=0.75,verbose=TRUE)  
LMAwrite(out1,suff="whateveryouwant")
```

```
out2=findClusters(LWCom=Sardinia,minsZ=1000,minSC=0.6667,tarsZ=10000,tarsC=0.9,verbose=TRUE)  
LMAwrite(out2,suff="done")
```

## Testing

Italian cases might not be sufficient

Report the errors/bugs and results

Add some other utilities

usefull statistics

visualisation tools

automatic fine tuning

Add some other functions – validity, cohesion

....

Add other features(+testing, reporting, etc)

- strategies to deal with the fictitious communities
- constraints
  - on the dimension of LMAs **yesterday**  
number of communities, inhabitants, workers
  - on overlapping geografies **yesterday**
  - other constraints
- distance functions **yesterday**
- sensitivity to small changes
- comparison between partitions
- enclaves – version 1.0 does **NOT** deal with the enclaves

...

Enjoy the LabourMarketAreas  package.

For further information:

Daniela Ichim, Luisa Franconi, Michele d'Alò

[ichim@istat.it](mailto:ichim@istat.it)

[franconi@istat.it](mailto:franconi@istat.it)

[dalo@istat.it](mailto:dalo@istat.it)