Proceedings of the *FOSS/GRASS Users Conference 2004* - Bangkok, Thailand, 12-14 September 2004

Introducing the Linear Reference System in GRASS

Radim Blazek

ITC-irst, Via Sommarive 18, Trento, Italy, tel. +39 0461 314 520, e-mail blazek@itc.it

1 Introduction

The Linear Reference System (LRS) is a system where features (points or segments) are localized by a measure along a linear element.

The LRS is suitable for management of data related to linear features like roads, railways and rivers. It is particularly important for network registers (e.g. roads) and traffic related studies (e.g. traffic accident studies).

The article describes the first implementation of the LRS in GRASS. The data model and the new modules **v.lrs.create, v.lrs.segment, v.lrs.where, v.lrs.stationing** are discussed. These modules can be used to:

1. Generate the LRS from input linear layer and milestones
2. Georeference points and segments from the LRS to 2D/3D space
3. Query LRS measure for points in 2D plane.

An example will be given for MITRIS, a large scale traffic safety project described in section 9, in which thousands of events have been automatically georeferenced via LRS.

2 The LRS

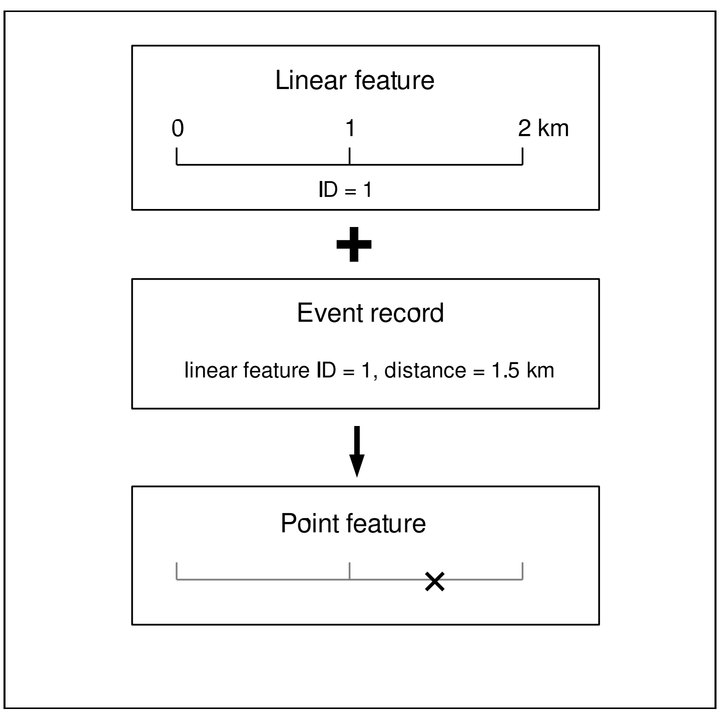

Figure 1: Referencing of point event

The LRS can be used to reference events for any network of linear features, for example roads, railways, rivers, pipelines, electric and telephone lines, water and sewer networks.

An event is defined in LRS by a route ID (LID) and a measure. A route is a path on the network, usually composed from more features in the input map. The LID is stored as an attribute in the table linked to input network map. An LID can be for example a road number. A measure is a distance measured along the linear object.

Events can be either points (Figure 1) or lines (Figure 2). For example, a point event could be for example a traffic accident. A linear event could be the type or quality of road pavement.

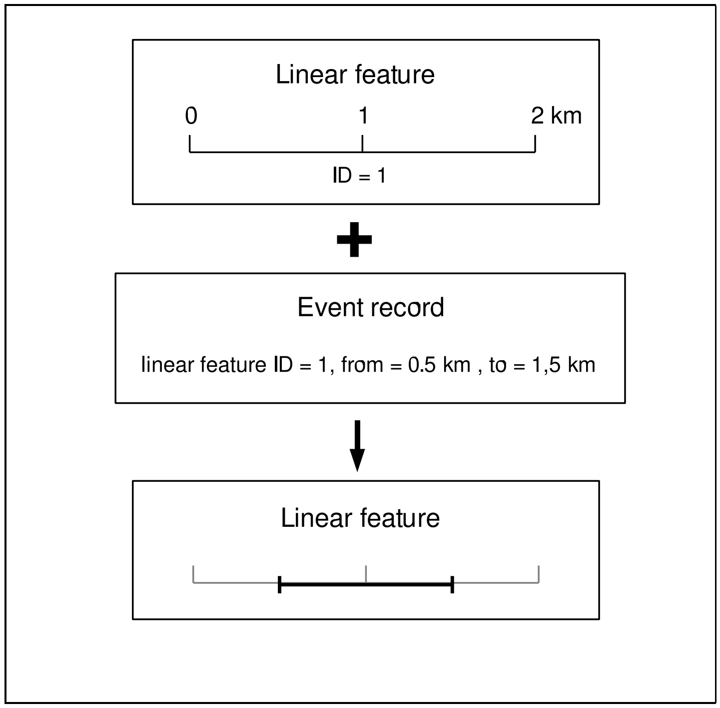

Figure 2: Referencing of linear event

The existing GRASS module **v.segment** provides the functionality similar to LRS. With v.segment it is possible to create points and segments of given measure. This is not sufficient however, for most of real world applications, because:

1. The distances measured in digital map are different from distances measured in real world, due to inaccurately digitized features, projection distortions etc.
2. A physical object (route) could be represented by different features in digital map.
3. The beginning of a physical object often does not correspond to the beginning of the feature in digital map.

Thus a more complete LRS is needed to deal with real data then the basic **v.segment**.

3 Changes of physical objects

Physical objects in real world change frequently. This further complicates the implementation of LRS. Reference systems usually exists also in the form of physical objects (milestones). It would be very expensive and impractical to change the whole reference system every time a small part of one route is changed.

In Figure 3 two types of changes are represented. The new version of the object might be either shorter or longer than the old one.

If the changed part of object is shorter, a step will appear between the 2 adjacent segments. If we take the first example on the figure 3, the original segment km <0,5> must be divided into two segments km <0,2> and km <4,5> at the original site 2. Such step will not cause any problems.

The more difficult case is when the new part of an object is longer then it was before, and a way to describe this new part must be found. One possibility, also used in practice, is to use the last unchanged milestone as a reference point, and measure the distance from this point in sub-units (for

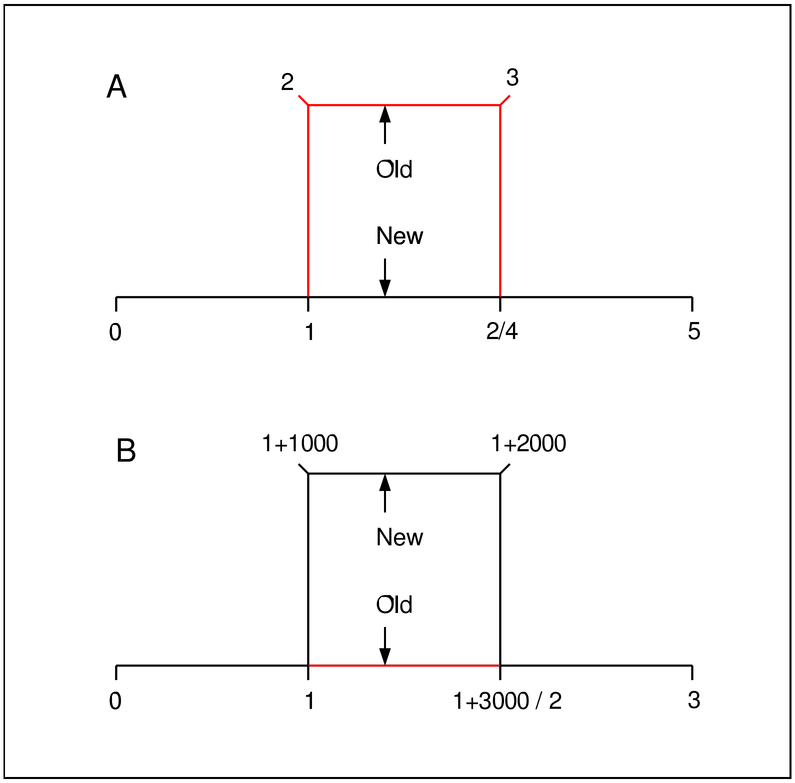

Figure 3: Changes of physical objects

example meters for milestones in kilometers). In example B in the figure the segment km <0,3> must be divided into segments km <0,1+3000> and <2,3>. Notation 1+3000 means the kilometer 1 and 3000 meters, which is the end of the changed part. The milestone between those two segments keeps two measures, the end of previous segment (1+3000) and the beginning of the next one (2+000). For example, using this notation, the events with measure 2+500 and 1+1500 can be distinguished.

This approach was also adopted by the LRS described in this article.

4 Implementation of a LRS for GRASS

4.1 Data model

To store a LRS, the proposed implementation in GRASS is using:

1. Regular vector map (LRS map). The LRS map is a new vector map containing linear features. This map is created from input map. The connected (no gaps) lines of the same route (the same LID) are joined to one continuous line and this line is oriented in the direction of increasing milestone values. Join and orientation of routes makes later use of LRS easier and faster.

2. Database table (LRS table). All additional information about LRS is stored in the database table. Each record in the LRS table represents one reference segment. The structure of the LRS table is shown in table 1.

4.2 New GRASS modules

Four new GRASS modules were written which can be used to create and use LRS:

1. **v.lrs.create** - generates new LRS

2. **v.lrs.segment** - creates events as new features from event records and existing LRS

3. **v.lrs.where** - queries LRS for given points (2D coordinates)

4. **v.lrs.stationing** - generate graphical representation of LRS as linear features and labels

Attribute	Type	Description
rsid	integer	reference segment ID, unique in the table
lcat	integer	category of the line in the LRS map
lid	integer	route ID (LID)
start_map	double precision	distance measured along the line in LRS map from the beginning of the line to the beginning of the segment
end_map	double precision	distance measured along the line in LRS map from the beginning of the line to the end of the segment
start_mp	integer	milepost assigned to the start of the segment
start_off	double precision	distance from start_mp to the start of the segment measured along the physical object
end_mp	integer	milepost assigned to the end of the segment
end_off	double precision	distance from end_mp to end of the segment measured along the physical object

Table 1: LRS table structure

5 Creating the LRS

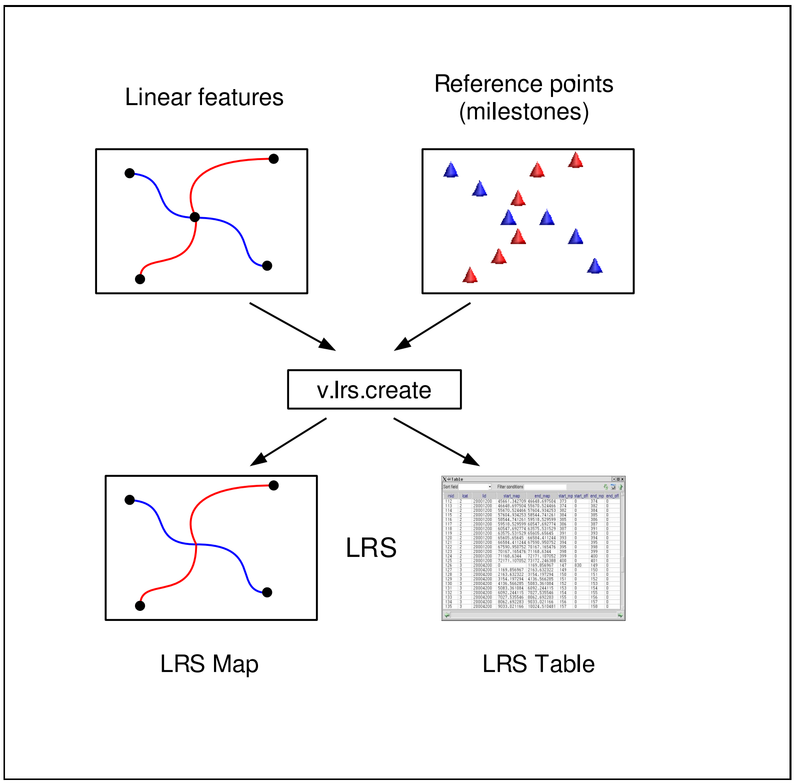

Figure 4: Creating LRS

The LRS is generated from input vector maps of linear features and reference points. The schema of the process is in Figure 4. This can be done by new GRASS module **v.lrs.create**:

```
Description:
 Create Linear reference system

Usage:
 v.lrs.create in_lines=name out_lines=name [err=name]
   points=name [lfield=value] [pfield=value] lidcol=name
   pidcol=name [start_mp=name] [start_off=name] [end_mp=name]
```

```
   [end_off=name] rstable=name [thresh=value]

Parameters:
   in_lines   Input map containing lines
  out_lines   Output map where oriented lines are written
        err   Output map of errors
     points   Input map containing reference points
     lfield   Line field
              default: 1
     pfield   Point field
              default: 1
     lidcol   Column containing line identifiers for lines
     pidcol   Column containing line identifiers for points
   start_mp   Column containing milepost position for
              the beginning of next segment.
              default: start_mp
  start_off   Column containing offset from milepost for
              the beginning of next segment.
              default: start_off
     end_mp   Column containing milepost position for the end of
              previous segment.
              default: end_mp
    end_off   Column containing offset from milepost for the end of
              previous segment.
              default: end_off
    rstable   Name of table where the reference system will
              be written. (New table is created by this module)
     thresh   Maximum distance of point to line allowed
              default: 1
```

rsid	lcat	lid	start_map	end_map	start_mp	start_off	end_mp	end_off
112	2	20001200	45661.342709	46648.697504	373	0	374	0
113	2	20001200	46648.697504	55670.524466	374	0	382	0
114	2	20001200	55670.524466	57604.934253	382	0	384	0
115	2	20001200	57604.934253	58544.741261	384	0	385	0
116	2	20001200	58544.741261	59518.529599	385	0	386	0
117	2	20001200	59518.529599	60547.692774	386	0	387	0
118	2	20001200	60547.692774	63575.531529	387	0	391	0
119	2	20001200	63575.531529	65605.65645	391	0	393	0
120	2	20001200	65605.65645	66584.411244	393	0	394	0
121	2	20001200	66584.411244	67590.950752	394	0	395	0
122	2	20001200	67590.950752	70167.165476	395	0	398	0
123	2	20001200	70167.165476	71168.6344	398	0	399	0
124	2	20001200	71168.6344	72171.107052	399	0	400	0
125	2	20001200	72171.107052	73172.246388	400	0	401	0
126	3	20004200	0	1169.856967	147	830	149	0
127	3	20004200	1169.856967	2163.632322	149	0	150	0
128	3	20004200	2163.632322	3154.197294	150	0	151	0
129	3	20004200	3154.197294	4136.566285	151	0	152	0
130	3	20004200	4136.566285	5083.361084	152	0	153	0
131	3	20004200	5083.361084	6092.244115	153	0	154	0
132	3	20004200	6092.244115	7027.535546	154	0	155	0
133	3	20004200	7027.535546	8062.692283	155	0	156	0
134	3	20004200	8062.692283	9033.021166	156	0	157	0
135	3	20004200	9033.021166	10024.510481	157	0	158	0

Figure 5: LRS table

An example of resulting LRS table is on Figure 5.

6 Creating events

It is possible to create point or linear events using the new module **v.lrs.segment**:

```
Description:
 Create points/segments from input lines, linear reference system
 and positions read from stdin in format:
P <pid> <lid> <milepost>+<offset> [<side offset>]
L <sid> <lid> <milepost>+<offset> <milepost>+<offset> [<side off>]

Usage:
 v.lrs.segment input=name output=name [lfield=value] rsdriver=name
   rsdatabase=name rstable=name

Parameters:
       input   Input map containing lines
      output   Output map where segments will be written
      lfield   Line field
               default: 1
    rsdriver   Driver name for reference system table
  rsdatabase   Database name for reference system table
     rstable   Name of the reference system table
```

’pid’/’sid’ is category assigned to the output feature. ’lid’ is route ID (LID).

7 Querying the LRS

Querying LRS is the inverse task of creating an event. When we query a LRS, we want to know the route ID and the LRS measure of a point, given by its coordinates in 2D plane. The new module **v.lrs.where** can be used for this purpose:

```
Description:
 Find line id and real km+offset for given points in vector map
 using linear reference system

Usage:
 v.lrs.where lines=name points=name [lfield=value] [pfield=value]
   rstable=name [thresh=value]

Parameters:
    lines   Input map containing lines
   points   Input map containing points
   lfield   Line field
            default: 1
   pfield   Point field
            default: 1
  rstable   Name of the reference system table
   thresh   Maximum distance to nearest line
            default: 1000
```

8 Visualization of LRS

It is often practical to display LRS on screen or print it on a paper map. For this purpose the new module **v.lrs.stationing** was written. It generates labels and linear features which can graphically represent the LRS.

```
Description:
 Create stationing from input lines, and linear reference system

Usage:
 v.lrs.stationing input=name output=name [lfield=value]
   rstable=name [labels=name] [offset=name[,name,...]]
   [xoffset=value] [yoffset=value] [reference=name] [font=name]
   [size=value] [color=name] [width=value] [hcolor=name]
   [hwidth=value] [background=name] [border=name] [opaque=name]

Parameters:
       input   Input map containing lines
      output   Output map where stationing will be written
      lfield   Line field
               default: 1
     rstable   Name of the reference system table
      labels   Label file
      offset   PM left, MP right, stationing left, stationing
               right offset
               default: 50,100,25,25
     xoffset   Offset label in label x-direction in map units
               default: 25
     yoffset   Offset label in label y-direction in map units
               default: 5
   reference   Reference position
               options: center,left,right,upper,lower
               default: center
        font   Font
               default: standard
        size   Label size (in map-units)
               options: 1-1000
               default: 100
       color   Text color
               options: aqua,black,blue,brown,cyan,gray,green,
                        grey,indigo,magenta, orange,purple,red,
                        violet,white,yellow
               default: black
       width   Line width of text (only for p.map output)
               options: 1-100
               default: 1
      hcolor   Highlight color for text (only for p.map output)
               options: aqua,black,blue,brown,cyan,gray,green,
                        grey,indigo,magenta, orange,purple,red,
                        violet,white,yellow
               default: none
      hwidth   Line width of highlight color (only for p.map)
               options: 0-100
               default: 0
  background   Background color
               options: aqua,black,blue,brown,cyan,gray,green,
                        grey,indigo,magenta, orange,purple,red,
                        violet,white,yellow
               default: none
      border   Border color
               options: aqua,black,blue,brown,cyan,gray,green,
                        grey,indigo,magenta, orange,purple,red,
                        violet,white,yellow
               default: none
```

```
      opaque    Opaque to vector (only relevant if background
                color is selected)
                options: yes,no
                default: yes
```

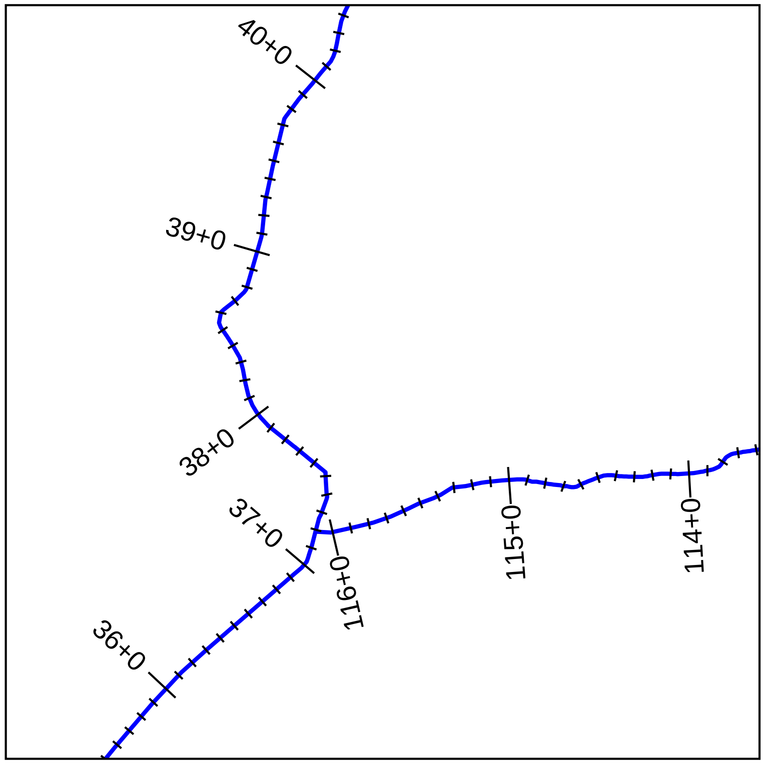

Figure 6: Stationing

The generated lines and labels can be used to produce Postscript maps with ps.map module. An example of such output is displayed in Figure 6.

9 Use of LRS in project MITRIS

This implementation of LRS we successfully used in the project MITRIS [2]. The objective of the project MITRIS is the development of complete service for monitoring of road accidents risk. Its first prototype has been developed on the 3200 km network of Trento Province (Italy).

The database of accidents is updated in two modes. New data are inserted into the database in the WebGIS interface, which permits the user to georeference the accident interactively on the map or specify coordinates directly (if measured by GPS). Another possibility is bulk update of the database in batch mode; the option is used for old data, recorded before the Web interface was available. The LRS was used to georeference older data sources, where geographic coordinates are missing, but the road number and measure in the system used by province or state authorities are available. Such records were automatically georeferenced in batches.

The LRS for MITRIS was generated from the existing layer of state and province roads and the layer of milestones. The layers are synthesis of the maps digitized on scanned paper map 1:10000 and mapping done with GPS. The distance between digitized milestones is 1 kilometer.

The quality of the data in the LRS can be measured by the difference between the length of the segment measured in the GIS and the length calculated from two milestones in real world. After some corrections of roads and reference points we reached the distribution of errors on Figure 7. In the histogram of errors we can see more segments with error < 0 (i.e. if the length measured in GIS is shorter than that in real world). This is an expected result, because linear objects are substituted by polylines in GIS.

A thematic map can be produced for error size attribute and used to increase efficiency of data corrections. An example of error map is on Figure 8. Only the segments with the errors lower than

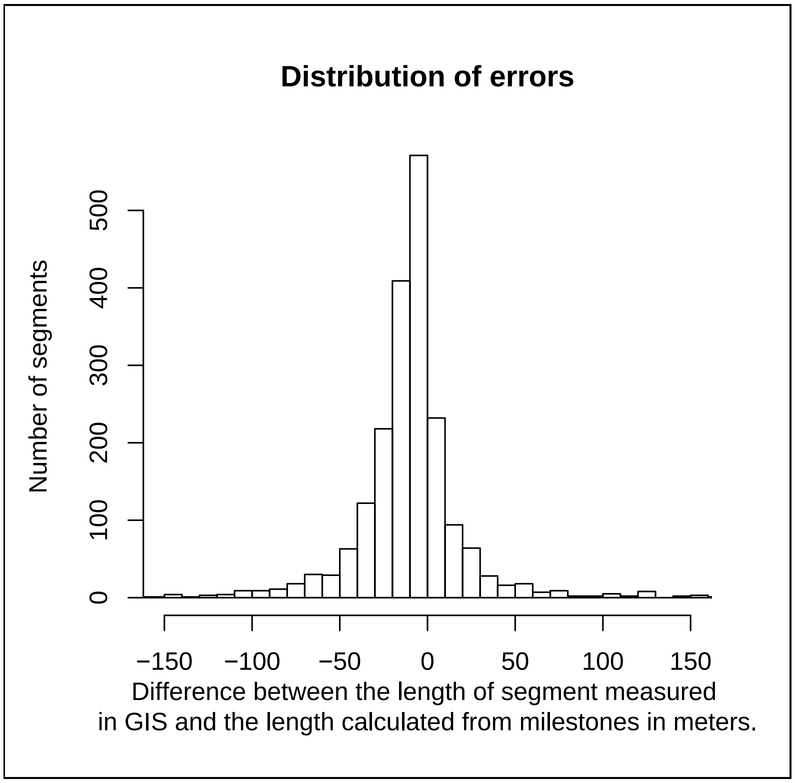

Figure 7: Distribution of errors

certain limit were used to georeference events. Segments with errors greater than the limit were excluded from LRS and events falling to those segments were later georeferenced manually.

10 Conclusions

The suggested implementation of LRS in GRASS was proven to be working well with real data. The system currently supports only static data. The possibility to extend the system to support also dynamic data must be examined in future. There are at least three possible ways to incorporate dynamic data:

1. Define a new GRASS format, equivalent of 'native' and 'ogr'. The format file ('frmt') would define LRS map and LRS table, features would be generated dynamically when the map is opened and the topology and other support data would be always generated dynamical.

2. Add new functions to GRASS Server [3] and use the LRS in PHP Web applications.

3. Store the LRS table in Postgres database, upload LRS map to Postgres database in PostGIS format and extend Postgres to support LRS in this format. This approach was tested in an experimental application.

One disadvantage of this implementation of LRS is that the LRS map generated by v.lrs.create duplicates data. Even if this does not appear to be a problem, it should be considered also a possibility to use directly the original map, without necessity to generate a new one.

Missing are analytical tools for operations with events, for example overlap of point and linear events etc. It is however possible to generate new features for events and analyse them by standard GRASS modules (v.distance, v.overlay).

11 Acknowledgement

The work was partly supported by the WILMA project [1].

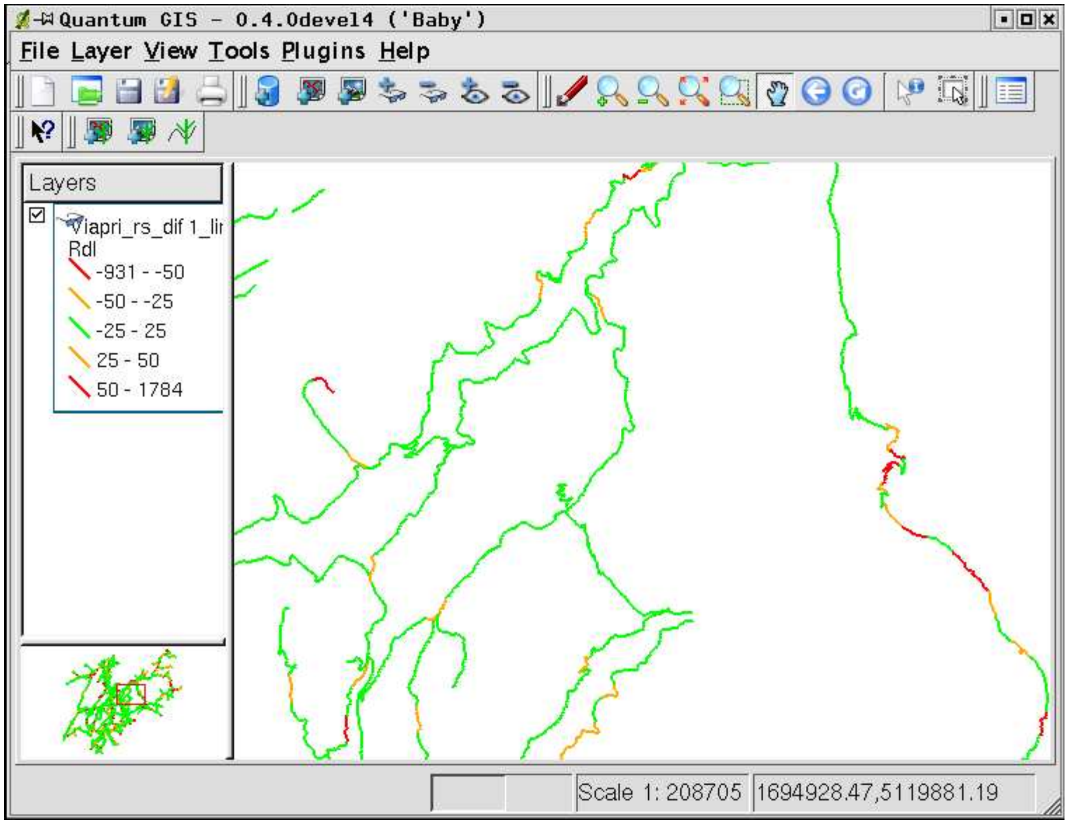

Figure 8: Thematic map of errors

References

[1] WILMA, URL: http://www.wilmaproject.org/

[2] MITRIS, URL: http://mitris.itc.it/

[3] Radim Blazek, Luca Nardelli, 2004, The GRASS Server, Proceedings of the *Free/Libre and Open Source Software for Geoinformatics: GIS-GRASS Users Conference*