TransitMapper Documentation
Draft

Patrick Brosi

23rd March 2017

1 Basic Usage

The input file is a graph given as a (simplified) Geo-JSON file, consisting of nodes (represented as "Point"-features) and edges (represented as "LineString"-features). Each edge has a collection of unique "lines" that travel through it. The transitmapper will render these lines in a way that resembles a transit map (Figure 1). The input is read from `stdin`.

```
$ transitmapper -o test.svg < test.json
```

See 4 for an example input.

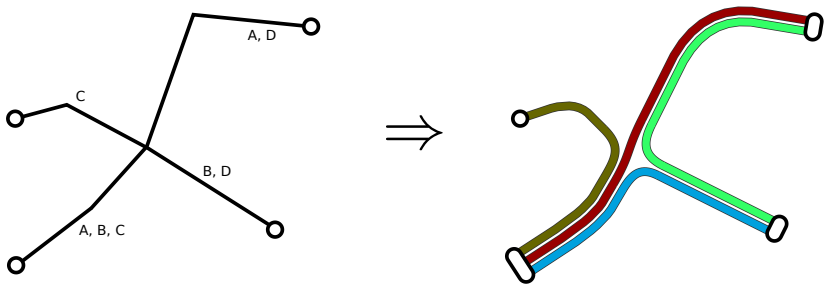

Figure 1: Simple example output

2 Command line parameters

The following command line parameters are accepted by `transitmapper` (see also `--help`).

`--line-width=N` The default width of a line, in output units. 20 by default.

`--line-spacing=N` The default spacing between lines, in output units. 10 by default.

`--render-station-names` Output the station names (experimental).

`--render-node-fronts` Output node fronts, useful for debugging.

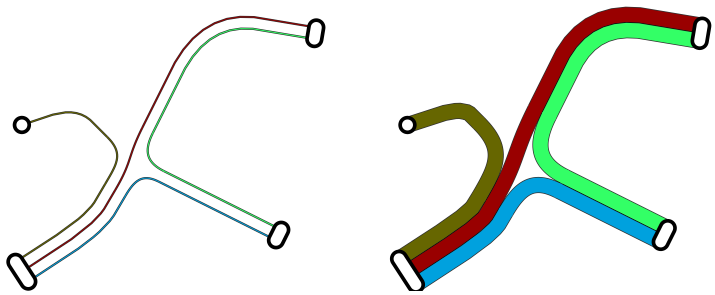

Figure 2: Different settings of `--line-width` and `--line-spacing`

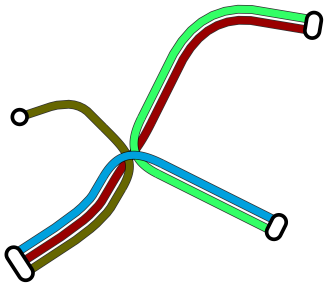

Figure 3: Output without ordering optimization (`-N`)

`--resolution=D` Output resolution. 0.1 by default.

`--no-optim` (`-N`) Disable line-ordering optimization.

`--input-smoothing=D` Level of input-data smoothing. 3 by default.

3 JSON Format

See also ??. The input format is a lightweight subset of GeoJSON. At the top level, the input JSON must contain a `FeatureCollection` object:

```
1 {
2   "type": "FeatureCollection",
```

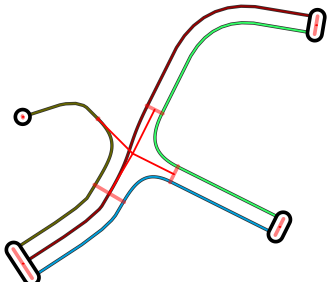

Figure 4: Node-front rendering (`--render-node-fronts`)

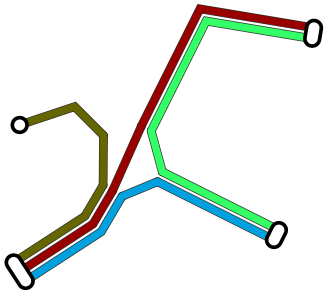

Figure 5: Without any line smoothing (`--input-smoothing=0 --bezier-prec=0`)

```
  "features": [...]
}
```

A `feature` can either be a node (a `point`) or an edge (a `LineString`).

3.1 Nodes

A node consists of a geometry (the node's coordinates) and some properties.

```
{
  "geometry": {
        "coordinates": [0, 0],
        "type": "Point"
  },
  "properties": {
        "id": "1",
        "station_id": "1"
  }
}
```

`Type` is always `"point"`. The `coordinates` are given as an `[x,y]` array.

3.1.1 Node Properties

`id` A dataset-unique string identifier for this node. Will be referenced later in edges.

`station_id` If this node is a station, the station's unique id (optional).

`station_label` If this node is a station, the station's name (optional). Either `station_id` or `station_label` has to be set for a node to become a station.

`excluded_line_conns` A list of lines that are not connected in this node, even though two or more edges containing the line start/end in the node (Figure 6). The `line_id` as well as the two edges this line is not connected in has to be given. Both edges are identified by the adjacent node. Example:

```
"excluded_line_conns": [
  {
        "edge1_node": "2",
        "edge2_node": "3",
        "route": "A"
```

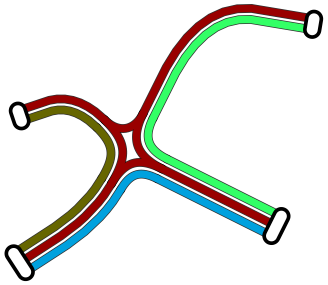

Figure 6: The red line is not connected between the two main axes (via `exluded_line_connections`)

```
  },
  {
        "edge1_node": "4",
        "edge2_node": "6",
        "route": "B"
  }
]
```

3.2 Edges

An edge consists also of a geometry (a linestring) and some properties.

```
{
  "geometry": {
        "coordinates": [
          [0, 0], [500, 900], [1000, 950]
        ],
        "type": "LineString"
  },
  "properties": {
        "from": "1",
        "to": "2",
        "lines": [
          {"color": "00a1de"},
          {"color": "990000"}
        ]
  }
}
```

3.2.1 Edge Properties

Fields `from` and `to` hold the IDs of the nodes this edges connects. Property `lines` holds the lines that occur on that edge.

`from` Holds the ID of the node this edge originates from.

`to` Holds the ID of the node this edge ends in.

`lines` An array of line objects, see below.

3.2.2 Line object

A line object is a single line occurance on an edge. Lines can either be identified accross edges by a unique ID or by the lines color as a shortcut. If you give a line an ID, you don't have to re-define its attributes again for an occurance in another edge. A single definition is enough.

The following properties are available for line occurances at the moment:

`id` (optional) A unique string-identifier of this line. Will not be visible in the rendered output.

`label` (optional) A string label that will be visible in the output as a line marker, if configured. Defaults to `""`.

`color` (optional) The color this line will be rendered in, as a hexadecimal string (for example, `"660000"`. **Important:** if the line has no id, the color will be used for matching lines in adjacent edges.

`direction` (optional) You may want to define a line that only occurs in one direction on the edge. This field allows you to specifiy the ID of the node this line is headed to. Node can either be `from` or `to`. **Note:** may be changed to a binary parameter. Defaults to a line that occurs in both directions. A single-direction line will be rendered with a little arrow indicating the direction. (Figure 7.)

`style` (optional) A style attribute (still experimental). See below.

3.2.3 Style object

The style object can be used to define additional styles for lines. A style definition only changes the look of a line in the edge it is listed in. It is not global.

`dash-array` (optional) A comma- or space-separated list of doubles. Starting at 1, the uneven positions are the lengths of filled line segments, the even positions are the lengths of the gaps. For example, `5, 10` produced a dashed line with segments of length 5, spaced by 10 units. (Figure 8.)

`line-width` (optional, **not yet implemented**) Overrides the width for this line, on this edge.

`line-spacing` (optional, **not yet implemented**) Overrides the spacing for this line, on this edge.

`css` (optional, **experimental!, only works with SVG output**) A free-hand css style for this line. Will be handed directly to the SVG renderer. The CSS defined here is not by the transitmapper to calculate spacing or optimize placements. For example, if you increase the width here, the line will overlap other lines.

4 Example Input

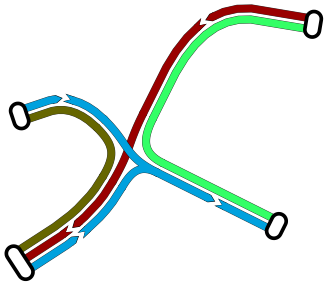

Figure 7: Directed line (via `direction`)

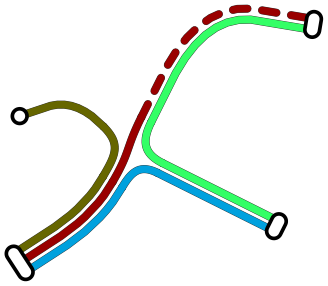

Figure 8: Line with `dash-array = "10 5"`

```geojson
{
  "type": "FeatureCollection",
  "features": [
    {
      "geometry": {
        "coordinates": [0, 0],
        "type": "Point"
      },
      "properties": {
        "id": "1",
        "station_id": "1"
      }
    },
    {
      "geometry": {
        "coordinates": [1000, 1000],
        "type": "Point"
      },
      "properties": {
        "id": "2",
        "station_id": "2"
      },
      "type": "Feature"
    },
    {
      "geometry": {
        "coordinates": [
          [0, 0], [500, 900], [1000, 950]
        ],
        "type": "LineString"
      },
      "properties": {
        "from": "1",
        "to": "2",
        "lines": [
          {"color": "00a1de"},
          {"color": "990000"}
```

```
        ]
      }
    }
  ]
}
```