DTU Management Engineering
Department of Management Engineering

Data Documentation

This document provides information about the dataset used for maintenance planning of railway system in the biggest region of Denmark, Jutland. The document gives the explanation of the problem followed by data definition. Then it provides how the dataset is created and how the software application generates each data file. Data generation is explained thorough a step by step procedure along with snapshot.

Before starting with the dataset definition, we explain the maintenance planing problem based on European Railway Traffic Management System (ERTMS).

1. Signaling maintenance planning for ERTMS in Jutland

Signaling maintenance in ERTMS includes maintaining of all equipment needed for implementation of ERTMS in any railway network. Generally, we can say that there are three different types of signaling equipment, consequently, three different maintenance tasks, according to the location of the equipment in an ERTMS-based railway system.

The first and the most important key component of ERTMS is an on-board signaling equipment which needs for a complete renewal in the existing signaling system referred to as the European Train Control System (ETCS). Implementing ETCS will naturally ease maintenance of the on-board equipment, coming directly to the workshop, which is easier than with signalling equipment installed along the track, far from any maintenance location. This means that, this type of maintenance tasks are not necessarily maintained or inspected on the railway track position.

The second type of tasks contains track-related equipment like balises and point machine which the crew/engineer is needed to maintain such equipment at the track position.

The third type of the maintenance tasks are those that need to be maintained on their installed position, despite of whether they are track-related or signaling-related. This category includes maintaining of driver screen in the train and also the antenna installed on top of the train. However it is not a track maintenance task but it is located on the track position geographically. Maintaining the equipment installed in the radio block center as another main component in ERTMS could be categorized in this type.

Apart from difference of maintenance tasks based on their geographic locations, they can mainly be divided into two different types according to the expertise they need. In this case, the first type of the maintenance tasks are related to track equipments like point machines, train detection, balises and signalsis can be expanded to include balises. The existing group of maintenance engineers and crew in Denmark are responsible for maintaining of these components in the railway network. The second type of tasks will be related to the electronic interlocking system and on-board equipment. This type of tasks are more complicated tasks which mostly cannot be solved by first set of crew at least on their own. This means that there are some maintenance tasks that cannot be solved by only first set of crew. Similarly for the second type of tasks, although some of the task can be handled by second type of crew individually, there are still some maintenance tasks that require expertise from both of these groups.

The maintenance organisation for ERTMS consists of both a first-line and a second-line maintenance teams. The first team of engineers are responsible for track equipments such as ; point machines, train detection, signals and balises. And the second team, who are highly qualified experts specialised in solving system failures, are responsible for the electronic interlocking system. They are electromechanical engineers and they deal with more complicated problems which the first-line engineers cannot resolve. They also communicate with the different suppliers of the GSM-R, Radio Block Centres (RBC-s), and European Vehicle Computers (EVCs). They generate the failure analysis and are responsible for the software releases of the equipments. Although they handle the task individually, there are some maintenance tasks that require expertise from both of these groups.

Moreover, the extra possibilities include new ways for preventive maintenance and to monitor track-side devices. This is for all kinds of equipment, from train detection and point equipment to Automatic Train Protection(ATP) devices, track-side and on-board. Monitoring task are mostly centralized in the radio block center.

Based on this categorization, we designed three different set of problems. Each of these set is different from the other set according to the location of maintenance tasks. They are geographical data indicating tasks and crew locations, task duration, and positioning of the time windows.

2. Dataset Description

Generally, each dataset consists of a set of geographical points, demand, time window constraint, duration and type as below. The geographical points all are located inside the biggest region of Denmark, Jutland. To standardize our dataset, we follow the file format from classical benchmark testset for Vehicle Routing Problem Time Windows (VRPTW) introduced by Solomon in 1998 (http://w.cba.neu.edu/ msolomon/problems.htm).

Since the maintenance planning in Denmark has a decentralized maintenance structure, the crew are located in different location in the Jutland meaning that they start their daily tasks from their home location rather than a single depot/station. According to this, location of the crew are different from each other in the dataset. We have two set of rows in each dataset indicating number of crew and number of tasks and their specifications, respectively. The first set of rows is related to the crew which are located in different geographical locations over the region and are distinguished by setting demand and duration of the row by zero. In summary, in this set of row we have below information:

- Index
- Crew geographical coordination

- Demand = 0,
- time window $[e_0, l_0]$
- Duration = 0,
- Type = 0,

The time window for each crew is used for working hours of the related crew. In this way, we can differentiate between full time and half time crew.

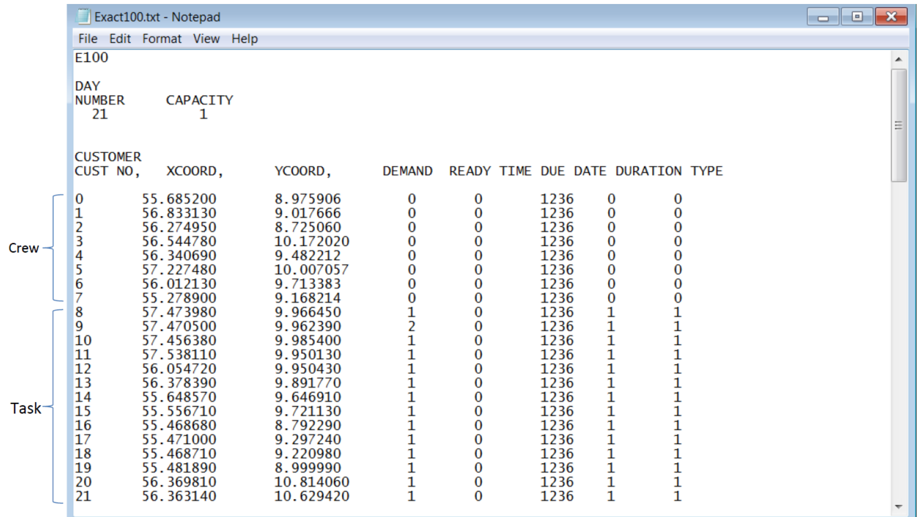
The second set of rows belongs to the maintenance tasks consist of the geographical coordination, demand of the tasks which is used for synchronization tasks. If a demand of task is 1 it, means that the task should be done by one crew, if it is two means that two crew should jointly do the task and so on.

- Index
- Maintenance task geographical coordination
- Demand $q_i > 0$,
- Time window $[e_i, l_i]$,
- Duration = 0,
- Type = 0,

The locations of crew are identical for all problems, while the set of the maintenance tasks has been randomly generated by utilizing Google Map API in the following categories:

- Random points on whole Jutland area
- points on the railways lines in Jutland
- Mixed of random points and the exact points on the railways

To test the scheduler on different time-horizon, each set of problems has four different numbers of tasks which should be serviced by a number of crew which are: 100, 500, 1000, and 5000. These numbers are chosen respectively for the number of maintenance tasks needed to be done on daily, weekly, monthly, and half yearly bases according to the current scale of maintenance planning in Denmark. In addition to different maintenance task locations, this help us to evaluate our approach on clustering the maintenance tasks in different situations when the coordination of the tasks are scattered through the area randomly, are located densely in the railway lines and are a mixture of scattered and on track points. Figure 1 is a snapshot of text file of one of data instances.



Exact100.txt - Notepad

File Edit Format View Help

E100

DAY
NUMBER CAPACITY
21 1

CUSTOMER

CUST NO,	XCOORD,	YCOORD,	DEMAND	READY TIME	DUE DATE	DURATION	TYPE
0	55.685200	8.975906	0	0	1236	0	0
1	56.833130	9.017666	0	0	1236	0	0
2	56.274950	8.725060	0	0	1236	0	0
3	56.544780	10.172020	0	0	1236	0	0
4	56.340690	9.482212	0	0	1236	0	0
5	57.227480	10.007057	0	0	1236	0	0
6	56.012130	9.713383	0	0	1236	0	0
7	55.278900	9.168214	0	0	1236	0	0
8	57.473980	9.966450	1	0	1236	1	1
9	57.470500	9.962390	2	0	1236	1	1
10	57.456380	9.985400	1	0	1236	1	1
11	57.538110	9.950130	1	0	1236	1	1
12	56.054720	9.950430	1	0	1236	1	1
13	56.378390	9.891770	1	0	1236	1	1
14	55.648570	9.646910	1	0	1236	1	1
15	55.556710	9.721130	1	0	1236	1	1
16	55.468680	8.792290	1	0	1236	1	1
17	55.471000	9.297240	1	0	1236	1	1
18	55.468710	9.220980	1	0	1236	1	1
19	55.481890	8.999990	1	0	1236	1	1
20	56.369810	10.814060	1	0	1236	1	1
21	56.363140	10.629420	1	0	1236	1	1

Figure 1

3. Data Generation

We have generated our dataset through three following steps:

1. Finding the Jutland boundary
2. Finding the geographical points on the rail track
3. Generating random points for each dataset

3.1. Finding the boundary of Jutland

we have used a drawing application for polyline, polygon, polygon with holes, rectangle, circle, marker(icon), direction(route, path). This application uses the Google Maps API Version 3 (V3). It has all the features of Google Maps MyMaps and has direct access to the code for the shapes (overlays) we create. While we drew and created a map on the region (Jutland), KML or Javascript code was presented in the textarea. We copied KML code and pasted it into a text editor. Then we had a KML file including all of geogeraphical points of the boundary.

Figure 2 shows the interface of the Google Maps API v3 Tool and the created boundary of Jutland through this application. For more information, the reader is referred to http://www.birdtheme.org/useful/v3tool.html

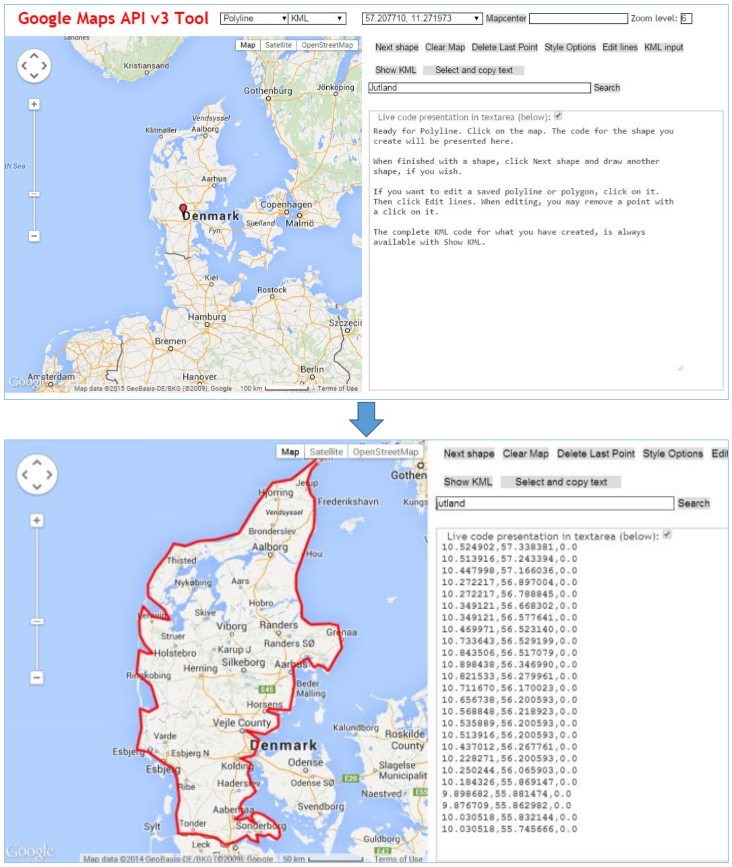


Figure 2

3.2. Finding the geographical points on the rail track

To generate random point particularly on the rail track, we prepared a list of routes from different origin and Destination. The list covers the whole track routes on the Jutland region. Figure 3 shows some of routes included in whole set of routes.

	Orgin	Destination
▸	Skagen	Padborg
	Hirtshals	Hjorring
	Grenaa	Aarhus
	Thisted	Tonder
	Struer Train Station, 7600 ...	Langaa
	Holstebro	Vejle County
	Herning	Bredgade 55A, 6900 Skjern
	Bramming	Lunderskov Municipality
	Tinglev	Sonderborg
	Vemb	Thyboron
	Varde	Nymindegab

Figure 3

3.3. Generating random points for each dataset

To generate the random point inside the boundary, we have used a java script. The boundary of the region is given as input to the script and it generates a number of desired points inside the boundary.

For the random set of the problems, we generated a maximum set of random points through the mentioned JavaScript and then we have chosen number of requested task for each problem randomly through a random generator function in C#. Accordingly, for the problems with tasks on the track, we have chosen required number of task from collected points on different track routes by the same random function in C#. Figure 4 represents the schematic picture of chosen random tasks after the mentioned procedure.

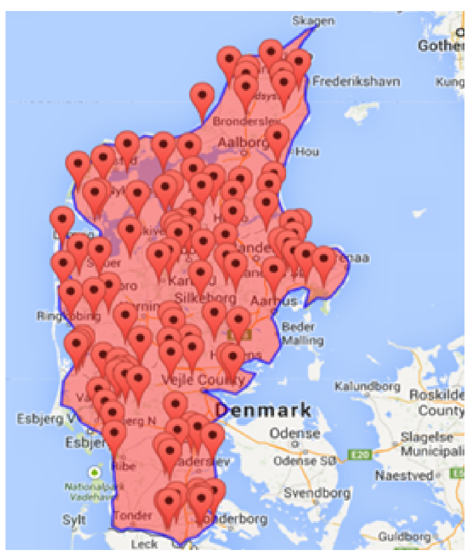


Figure 4

3.4. Software Application

To generate each data file, we developed a software in Microsoft Visual Studio C.Net. Figure 5 represents the user interface of the application. The inputs are number of the tasks and mode of geographical location and the output is a text file according to the format of Solomon dataset. The other parameters in each data has considered as constant value in the code. For example, the number of crew has constant value of 8 in all of data instances. However the software can be updated to get every parameter as input, later on.

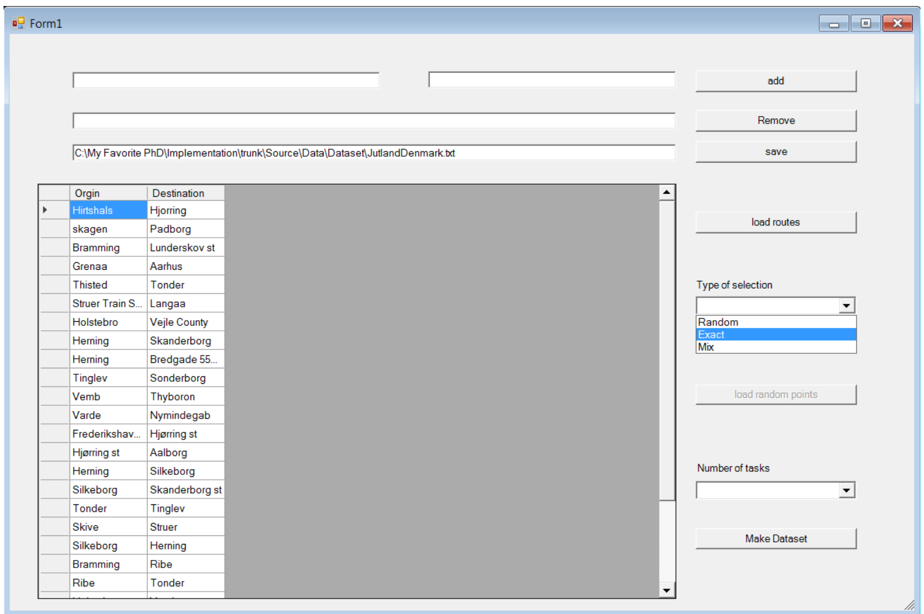


Figure 5

To generate the geographical coordination exactly on railway tracks, the software makes use of previously added routes on the rail track of Jutland. However the software gives possibilities to add extra routes to include more regions into the dataset. Accordingly, to generate the geographical coordination randomly scattered all over the network, the software loads a pool of generated random points in Jutland by the JavaScript and randomly chooses the number of needed point according to the size of the dataset. Finally, to generate a mix of points, the software generate points randomly(By Random.Next() function in *C*) our of two sets of mentioned geographical sources.

4. Appendix (Java Script code)

```
        var map;
        var boundaryPolygon;
```

```
        function initialize() {

            var mapProp = {
                center: new google.maps.LatLng(26.038586842564317,
                    75.06787185438634),
                zoom: 6,
                mapTypeId: google.maps.MapTypeId.ROADMAP
            };

            map = new google.maps.Map(document.getElementById("map-
                canvas"), mapProp);

            google.maps.Polygon.prototype.Contains = function (
                point) {
                // ray casting alogrithm http://rosettacode.org/
                    wiki/Ray-casting_algorithm
                var crossings = 0,
            path = this.getPath();

                // for each edge
                for (var i = 0; i < path.getLength() ; i++) {
                    var a = path.getAt(i),
                j = i + 1;
                    if (j >= path.getLength()) {
                        j = 0;
                    }
                    var b = path.getAt(j);
                    if (rayCrossesSegment(point, a, b)) {
                        crossings++;
                    }
                }

                // odd number of crossings?
                return (crossings % 2 == 1);

                function rayCrossesSegment(point, a, b) {
                    var px = point.lng(),
                py = point.lat(),
                ax = a.lng(),
                ay = a.lat(),
                bx = b.lng(),
                by = b.lat();
                    if (ay > by) {
                        ax = b.lng();
                        ay = b.lat();
                        bx = a.lng();
                        by = a.lat();
                    }
                    if (py == ay || py == by) py += 0.00000001;
                    if ((py > by || py < ay) || (px > Math.max(ax,
                        bx))) return false;
                    if (px < Math.min(ax, bx)) return true;

                    var red = (ax != bx) ? ((by - ay) / (bx - ax))
                        : Infinity;
                    var blue = (ax != px) ? ((py - ay) / (px - ax))
                         : Infinity;
```

```
                    return (blue >= red);
                }
            };


            google.maps.event.addListener(map, 'click', function (event) {
                if (boundaryPolygon != null && boundaryPolygon.Contains(event.latLng)) {

                    document.getElementById("spnMsg").innerText = " This location is " + event.latLng + " inside the polygon.";
                } else {
                    document.getElementById("spnMsg").innerText = " This location is " + event.latLng + " outside the polygon.";
                }

            });
        }
        function randomLeftSidepoint(min,max) {
            return (Math.random() * (max - min + 1 ) + min);
        }

        function randomRightSidepoint(min, max) {
            //var xx = [];
            //xx = random.uniform(min, max).split(".");
            //return  xx[1];
            return Math.random() * (max - min + 0.000001) + min;
        }

        function test() {
            var mingx = 8;
            var mingy = 54;
            var maxgx = 13;
            var maxgy = 57;

            var minlx = 0.033350;
            var minly = 0.010940;
            var maxlx = 0.948807;
            var maxly = 0.983637;

            var points = "";

            var x = [];
            // 1000 is the number of tasks.

            for (var i = 0; i < 1000; i++) {

                var lat = randomLeftSidepoint(mingx, maxgx) + randomRightSidepoint(minlx, maxlx);
                var longa = randomLeftSidepoint(mingy, maxgy) + randomRightSidepoint(minly, maxly);
                var myLatlng = new google.maps.LatLng(longa, lat);
                while (!boundaryPolygon.Contains(myLatlng)) {
                    lat = randomLeftSidepoint(mingx, maxgx) +
```

```
                        randomRightSidepoint(minlx, maxlx);
                    longa = randomLeftSidepoint(mingy, maxgy) + randomRightSidepoint(minly, maxly);
                     myLatlng = new google.maps.LatLng(longa, lat);
                }
                 addpoint(lat, longa);
                 points =points+ longa + ' ' + lat + '\r\n';


            }
            var blob = new Blob([points ], { type: "text/plain;charset=utf-8" });
            saveAs(blob, "generatepoints.txt");

            document.getElementById("spnMsg").innerText = lat+"  " +longa;

        }
        function addpoint(lat, longa) {
            var myLatlng = new google.maps.LatLng(longa,lat);
            var marker = new google.maps.Marker({
                position: myLatlng,
                map: map,
                title: 'Hello World!'
            });

         }

        function drawPolygon() {

            initialize();
            // Jutland Boundary
            var boundary = '10.600433 57.742281,10.517178
                57.720314,10.431175 57.679276,10.261917
                57.610396,10.171795 57.590683,10.076180
                57.581274,9.953785 57.581471,9.897308
                57.524221,9.826241 57.483308,9.766159
                57.436489,9.678955 57.322135,9.516907
                57.198608,9.394684 57.151597,9.242935
                57.130338,9.085693 57.129947,8.979950
                57.144266,8.794556 57.088532,8.682632
                57.095344,8.589935 57.096187,8.442993
                56.973898,8.342743 56.900827,8.289185
                56.826265,8.331070 56.735041,8.427887
                56.676953,8.555603 56.612296,8.560753
                56.553361,8.510971 56.524696,8.378448
                56.561212,8.182068 56.591816,8.175201
                56.444204,8.342056 56.294402,8.161812
                56.304913,8.179321 56.165969,8.323517
                56.051575,8.423767 55.924909,8.296051
                55.843596,8.206787 55.762283,8.189621
                55.629687,8.279572 55.608629,8.329353
                55.577584,8.368149 55.527906,8.536377
                55.470536,8.608303 55.444748,8.674736
                55.387797,8.676624 55.297220,8.689499
                55.205069,8.709755 55.115360,8.667698
                55.072085,8.680573 55.013083,8.667870
```

```
                54.923993,8.757648 54.903097,8.819962
                54.916936,8.996773 54.898843,9.157104
                54.879169,9.237270 54.858106,9.317436
                54.814907,9.384384 54.848705,9.475536
                54.861229,9.539223 54.899042,9.632864
                54.945540,9.660587 54.982573,9.570208
                55.024316,9.463348 55.010940,9.513302
                55.084769,9.469872 55.153863,9.657669
                55.210399,9.632263 55.309465,9.605999
                55.374576,9.585228 55.438124,9.635997
                55.475603,9.546025 55.480333,9.467039
                55.497514,9.623508 55.528894,9.733200
                55.579236,9.785299 55.604378,9.727535
                55.629520,9.675179 55.665850,9.532013
                55.707529,9.688396 55.716817,9.844780
                55.688967,9.948807 55.738502,10.030861
                55.806534,9.940052 55.822031,9.846497
                55.852958,10.033350 55.892624,10.157032
                55.862967,10.272560 55.965395,10.230160
                56.072406,10.182266 56.127278,10.226383
                56.178310,10.286980 56.223227,10.351696
                56.274229,10.427399 56.288624,10.482416
                56.297731,10.531940 56.279400,10.488167
                56.181695,10.616055 56.238167,10.696478
                56.229759,10.743942 56.242724,10.854149
                56.321904,10.902386 56.372136,10.917664
                56.439075,10.872688 56.475219,10.802994
                56.515889,10.685749 56.513132,10.579491
                56.492187,10.400448 56.514446,10.276337
                56.606379,10.326118 56.662977,10.244064
                56.795008,10.236168 56.893823,10.299683
                56.983637,10.377960 57.119815,10.511169
                57.244063,10.531082 57.266087,10.524559
                57.309762,10.501556 57.341558,10.503616
                57.388821,10.527649 57.458987,10.475464
                57.505496,10.437012 57.534239,10.409546
                57.576252,10.434952 57.618493,10.476837
                57.659263,10.601807 57.743747';


            var boundarydata = new Array();

            var latlongs = boundary.split(",");

            for (var i = 0; i < latlongs.length; i++) {
                latlong = latlongs[i].trim().split(" ");
                boundarydata[i] = new google.maps.LatLng(latlong
                    [1], latlong[0]);
            }

            boundaryPolygon = new google.maps.Polygon({
                path: boundarydata,
                strokeColor: "#0000FF",
                strokeOpacity: 0.8,
                strokeWeight: 2,
                fillColor: 'Red',
                fillOpacity: 0.4
```

```

            });

            google.maps.event.addListener(boundaryPolygon, 'click', function (event) {
                document.getElementById("spnMsg").innerText = '';
                if (boundaryPolygon.Contains(event.latLng)) {
                    document.getElementById("spnMsg").innerText = "This location is " + event.latLng + " inside the polygon.";
                } else {
                    document.getElementById("spnMsg").innerText = "This location is " + event.latLng + " outside the polygon.";
                }

            });
            map.setZoom(5);
            map.setCenter(boundarydata[0]);
            boundaryPolygon.setMap(map);

        }
```