Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für Bau- und Infrastrukturmanagement
Infrastrukturmanagementgruppe
ETH Zürich
Prof. Dr. Bryan T. Adey

Master Project, FS 2014

Optimal grouping of interventions on road networks

Author: Eicher Charel, ETH

Supervisors: Dr. Nam Lethanh, ETH
Prof. Dr. Bryan T. Adey, ETH

Zurich, the 30 of May 2014

ETH Zurich
Institute of Construction and Infrastructure Management
Chair of Infrastructure Management
Prof. Dr. Bryan T. Adey

Sworn Declaration

I hereby declare with this oath, that I have independently completed the part of the presented project listed below, without outside help and only under the application of the stated sources and tools. All sections taken verbatim or nearly verbatim, or correspondingly out of another source, were done so knowingly, the sources of which were appropriately referenced. I further declare that the work has not yet been published, neither as a whole nor in part or extracts, and also that it has not been submitted to any other examining authority.

I have produced the following parts of this work:

- All parts, sections, chapters, pictures and tables if not declared differently.

I hereby agree that the Institute for Construction and Infrastructure Management of the ETH Zurich may use the parts of the presented work completed by me for further use or publication, in whole or part. If I intend to further use or publish the work, or a part thereof, I will communicate these intentions in advance to the Institute of Construction and Infrastructure Management of the ETH Zurich.

Zurich,

Eicher Charel

List of contents

7. Appendix

1. Introduction

1.1 Background and goals

Road networks are comprised of different types of objects, such as road sections, bridges and tunnels. These objects are subject to deterioration and thus they need to be maintained in order to provide an adequate level of service. Maintenance on these roads is done by executing interventions. One of the tasks of road managers is to determine the interventions to be included in work zones and the traffic configurations to be used during the execution of these interventions. This requires taking into account the costs related directly to the objects being intervened on, such as the manual labour required to execute these interventions, and the costs related to the use of the network, e.g. an increase in noise and accidents due to deviated traffic around the work zone. It also requires balancing the increase in costs during and the decrease in costs after the execution of the interventions.
Grouping together multiple interventions into a work zone can often lead to a reduction in costs related to the execution of the interventions. The costs of installing a new traffic configuration for two network objects is not twice as much as for one object. It also, however, requires the explicit consideration of spatial constraints, such as the maximum length of a work zone and the minimum distance between adjacent work zones. In addition, interventions on road networks are usually subjected to cost constraints, such as a limited budget.
Although considerable work has been focused on the determination of the optimal interventions to be executed on road networks over the past years (e.g. the optimal scheduling of resource allocation [1], [2], the optimal scheduling of specific interventions [3], [4], and the optimal scheduling of interventions at the network level [5], [6]), there has been little research focused on the optimal grouping of interventions into work zones. Some of the work, however, include that by [7]–[10].

The optimization model presented by Hajdin, R., & Adey, B. T. (2006) is the starting point for this project. However, the proposed model contains some limitations. The model is limited to only one single work zone in the entire network. The model has to be set up manually, which is very time-consuming and thus only manageable for small networks containing only a few objects. The investigated network is very simple, containing no loops.

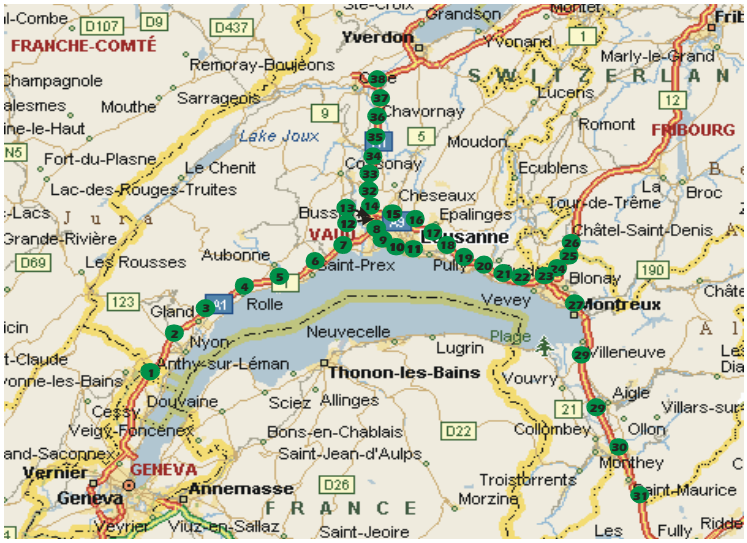


Figure 1: Network investigated by Hajdin, R., & Adey, B. T. (2006)

The goal in this project is to overcome some of the limitations and develop an algorithm that allows the determination of multiple optimal work zones in a road network. Besides the maximum work zone length constraint, the new algorithm needs to include a minimum distance between adjacent work zones constraint. Furthermore, a routing algorithm needs to be developed in order to set up the optimization problem automatically and thus enabling its application to large scale, real world road networks.

1.2 Work sites and work zones

An object in the network subjected to an intervention is considered a work site. For every work site, an intervention type and an intervention specific traffic configuration needs to be assigned. A work zone is comprised of one or several network objects containing at least one work site. If the work zone consists of only one single object, then the work zone is equal to the work site. A work zone can contain network objects being intervened on and objects not being subject to interventions. The objects in a work zone not being intervened on can be assigned a traffic configuration differing from the normal configuration. The first respectively last objects in a work zone need to be subject to an intervention.

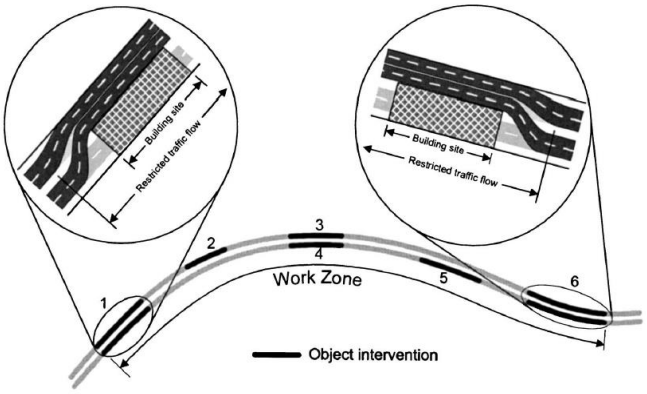


Figure 2: Hajdin, R. & Lindenmann H. P. (2007).

2. Model

2.1 Objective function

The optimal grouping of work sites into work zones is found by maximizing the following objective function:

$$Maximize \;\; Z = \sum_{n=1}^{N}\sum_{k=1}^{K} \delta_{n,k} \cdot (B_{n,k} - C_{n,k})$$

where

$\delta_{n,k}$ is a binary variable, which has a value of 1 if an intervention of type *k* is executed on object *n* and 0 otherwise.

$B_{n,k}$ is the long term benefit of executing an intervention of type *k* on object *n*.

$C_{n,k}$ is the cost of executing an intervention of type *k* on object *n*.

2.2 Continuity constraint

The continuity constraint ensures that for every object *n* in the road network, exactly one intervention type *k* is selected for the given planning period.

$$\sum_{n=1}^{N}\sum_{k=1}^{K} \delta_{n,k} = 1$$

2.3 Budget constraint

The budget for executing interventions on the network is in general limited. The total cost of all interventions on the examined network cannot exceed a certain threshold for the given planning period.

$$\sum_{n=1}^{N}\sum_{k=1}^{K} \delta_{n,k} \cdot C_{n,k} \leq \Omega$$

where

Ω is a the total budget allowance on the investigated period.

2.4 Maximum work zone length constraint

The maximum length of a work zone is restricted and cannot exceed a certain threshold. This means that the shortest path between any pair of nodes in the same work zone must not be larger than the maximum allowed work zone length. The work zone length threshold can depend on the road types subjected to interventions and the intervention types. For instance the maximum allowed work zone length for motorway interventions is larger than the one for cantonal roads.

$$\sum_{l=a_l^w}^{e_l^w} \sum_{n=a_n^w}^{e_n^w} \lambda_{l,n} \leq \Lambda^{MAX} \quad \forall w$$

where

$\lambda_{l,n}$ is the length of the road segment [*l,n*]

$a^w \ \left(l = a_l^w, n = a_n^w\right)$ is the first road segment of the wz $w = (1,...,W)$, and road segment $e^w \ \left(l = e_l^w, n = e_n^w\right)$ is the last road segment in the wz w.

Λ^{MAX} is the maximum work zone length

2.5 Minimum distance between work zones constraint

The minimum distance between two adjacent work zones in the road network cannot be smaller than a specific threshold Λ^{MIN}. This means that the shortest path between any pair of nodes in adjacent work zones must be larger than the minimum distance between work zones.

$$\sum_{l=a_l^d}^{e_l^d} \sum_{n=a_n^d}^{e_n^d} \lambda_{l,n} \geq \Lambda^{MIN} \quad \forall d$$

Where object $a^d \ \left(l = a_l^d, n = a_n^d\right)$ is the first object of the default section d and object $e^d \ \left(l = e_l^d, n = e_n^d\right)$ is the last object in the default section d. Every network has a number of default sections $d = 1,..., D$. A default section is defined as all objects physically connected to each other being in the default intervention type.

2.6 Implementation of the work zone constraints

The maximum work zone length and the minimum distance between work zones constraint is merged into one constraint by defining combinations of objects in the network that cannot be subject to an intervention simultaneously.

$$\sum_{n=1}^{N}\sum_{k=1}^{K}\delta_{n,k}\cdot\gamma_{n,k,i}\leq 1 \quad \forall i$$

$\gamma_{n,k,i}$ is a I-by-J matrix , where I is the total number of rows, with each row containing an object combination that cannot be selected simultaneously.

3. Routing algorithm

3.1 Overview

A routing algorithm has been developed in MATLAB environment that sets up the complete optimization problem. The optimization itself is done in Microsoft Excel using the What's Best! solver. Alternatively, optimization could be executed in MATLAB using the built-in Optimization Toolbox.

3.2 Input

The input file for the routing algorithm is a Microsoft Excel file containing all the necessary information for the setup of the optimization problem. Required as input is the following:

1. Network data:
   - Network objects
   - Lengths of the network objects
   - The nodes of the network objects
2. Number of possible interventions per object (minimum = 1)
3. Maximum work zone length
4. Minimum distance between work zones

Note: The allowed budget for the investigated planning period is not required as input.

Object	Length	Node 1	Node 2
1	1160	1	2
2	2700	2	3
3	830	3	4
4	810	4	5
5	1340	5	6
6	650	6	7
7	480	7	8
8	370	8	9
9	1370	8	10
10	230	10	11

Int. types
3
3
3
3
3
3
3
3
3
3

Maximum work zone length	Minimum distance between work zones
5000	8000
[m]	[m]

Figure 3: Excerpt of the network data, the number of intervention types and both work zone constraints

3.3 The algorithm

List of functions and scripts used by the algorithm:

- objects_selector_1.m
- objects_selector_2.m
- storage_input.m
- storage_output.m
- path_calculator_1.m

- path_calculator_2.m
- all_paths_1.m
- all_paths_2.m
- complete_network_1.m
- complete_network_2.m
- shortening.m
- worksites_transformer.m
- delete_copies_and_subsets_3.m
- find_combinations.m
- establish_adjacency_matrix.m
- establish_continuity_matrix.m
- establish_combinations_matrix.m
- main.m

Note: The code of the functions and additional explanations can be found in the appendix.

The routing algorithm establishes all the matrices (among them are the continuity and the combinations matrix) needed for the optimization based on the input file. When the calculation is finished, the matrices need to be manually imported into Microsoft Excel.

The functionality of the algorithm is explained using an artificial network.

3.3.1 The artificial network

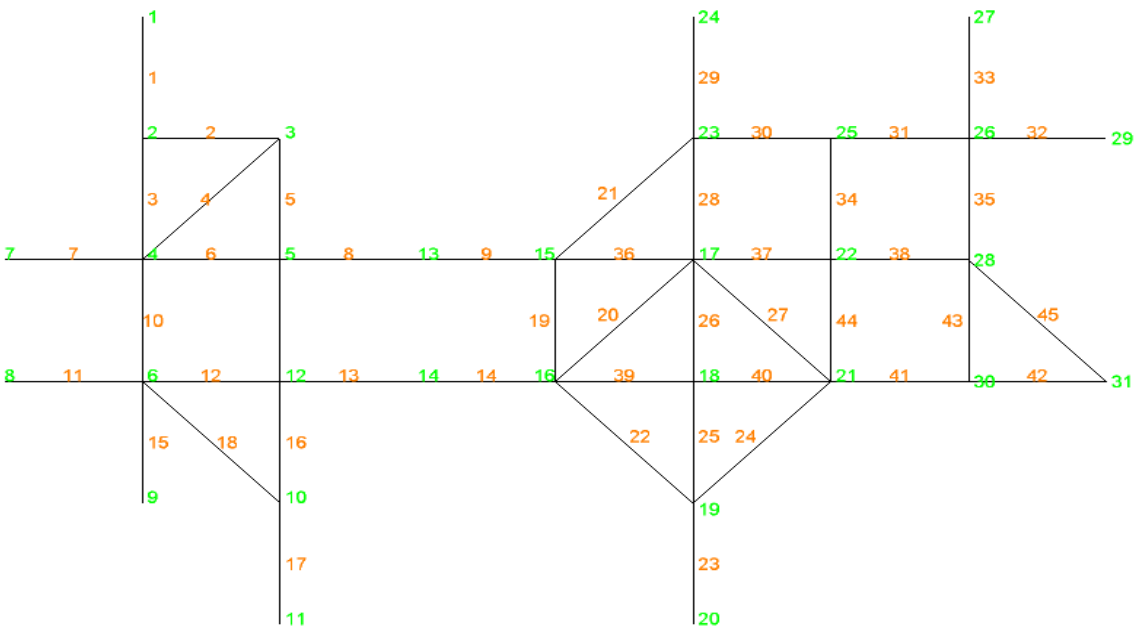


Figure 4: Artificial network

A network consists of nodes and objects. A node number (green) is assigned to every node and an object number (orange) is assigned to every object. The given network consists of 45 objects and 31 nodes. In a network containing intersections, the number of objects always exceeds the number of nodes. For simplicity, every object in the network has the same length of 5 km. The maximum work zone length is limited to 15 km and the minimum distance between work zones needs to be at least 15 km.

The main task of the algorithm is to calculate possible paths in the network, both for the maximum work zone length and the minimum distance between work zones constraint.

For the maximum work zone length constraint, the goal of the path calculation is to find, for a given object *n*, all the possible objects that can be part of the same work zone as object *n*.

For the minimum distance between work zones, the goal of the path calculation is to find, for a given object *n*, all the objects that cannot be part of the same work zone as object *n*. It is assumed that object *n* is the first respectively last object in the work zone.

3.3.2 Maximum work zone length constraint

For a given object *n* in the network, the algorithm calculates all the possible paths starting with this object that do not exceed the maximum work zone length. The length of these paths is defined as the sum of the lengths of all its comprising objects. The calculated paths are stored in a matrix. This procedure is repeated for every object in the network.

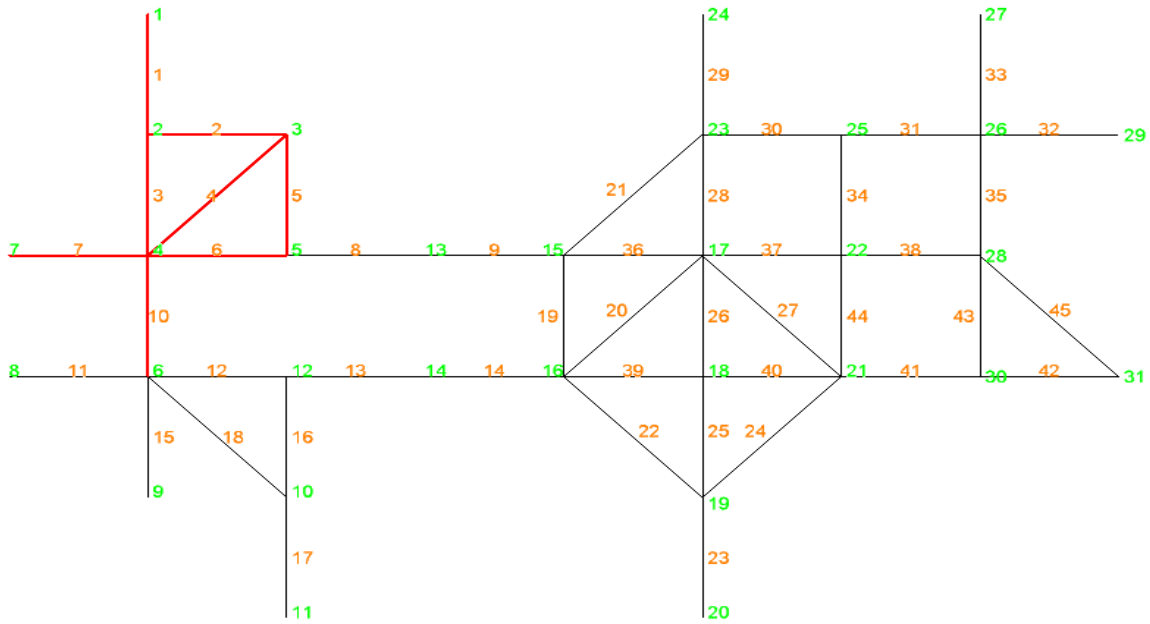


Figure 5: Objects comprising the paths for the maximum work zone length constraint starting with object 1 are highlighted in red

Table 1: Paths for object 1

All possible paths starting with object 1					
1	1	1	1	1	1
2	2	3	3	3	3
4	5	4	6	7	10

All the possible paths starting with object 1 are listed in the above table. The six possible paths are comprised of eight different network objects (1, 2, 3, 4, 5, 6, 7 and 10).

3.3.3 Minimum distance between work zones

Similar to the maximum work zone length constraint, the algorithm calculates, for a given object *n*, all the possible paths starting with this object. Objects are added to a path as long as the path length is smaller than the minimum distance between work zones. This procedure is repeated for every object in the network.

The length of a path is defined as the sum of the lengths of its comprising objects minus the length of the first object in the path. Thus the number of objects comprising the paths for the minimum distance between work zones exceeds the number of objects comprising the paths for the maximum work zone length constraint. Due to the fact that the paths for the minimum work zone length constraint contain more objects, the total number of possible paths starting with a given object is significantly larger than the total number of possible paths starting with a given object for the maximum work zone length constraint.

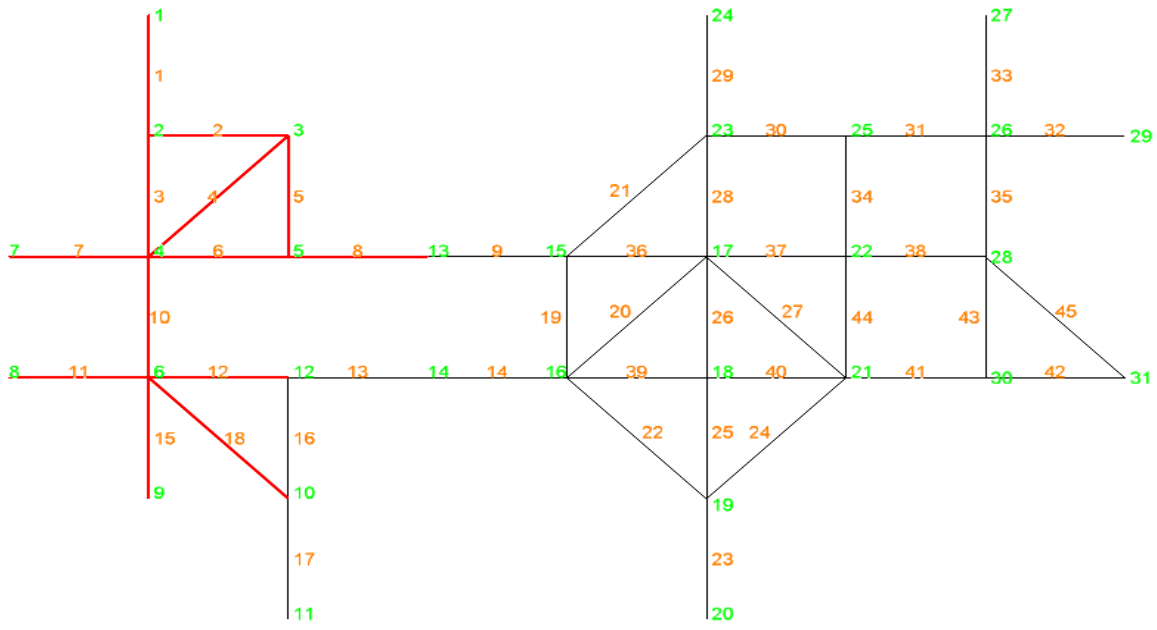


Figure 6: Objects comprising the paths for the minimum distance between work zones starting with object 1 are highlighted in red

Table 2: Paths for object 1

All possible paths starting with object 1														
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	3	3	3	3	3	3	3	3	3
4	4	4	4	5	5	4	4	6	6	7	10	10	10	10
3	6	7	10	6	8	2	5	5	8	0	11	12	15	18

All the possible paths starting with object 1 are listed in the above table. The fifteen possible paths are comprised of thirteen different network objects (1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 15 and 18).

3.3.4 Impossible object combinations

After all the possible paths for all the objects in the network have been calculated for both the maximum work zone length and the minimum distance between work zones constraint, the algorithm identifies impossible object combinations.

Impossible object combinations are pairs of network objects that would violate both work zone length constraints if they were subject to an intervention simultaneously. For a given object in the network, the objects that violate both work zone length constraints are detected. This procedure is repeated for every object in the network. The following example illustrates the principle.

Assuming object 1 is part of a work zone, impossible object combinations (with respect to object 1) are objects that are too far away to be part of the same work zone as object 1, but too close to be part of an adjacent work zone.

Objects comprising the paths for the maximum work zone length constraint							
1	2	3	4	5	6	7	10

Objects comprising the paths for the minimum distance between work zones												
1	2	3	4	5	6	7	8	10	11	12	15	18

Invalid combinations for object 1				
8	11	12	15	18

Figure 7: Objects comprising the paths for both work zone length constraints and the invalid object combinations with respect to object 1

For a given object, the impossible combinations are the objects that are part of a path for the minimum distance between work zones, but not part of a path for the maximum work zone length constraint. If object 1 is part of a work zone, then objects 8, 11, 12, 15 and 18 cannot be part of any work zone in the network because these objects would violate the maximum work zone length and the minimum distance between work zones if intervened on simultaneously.

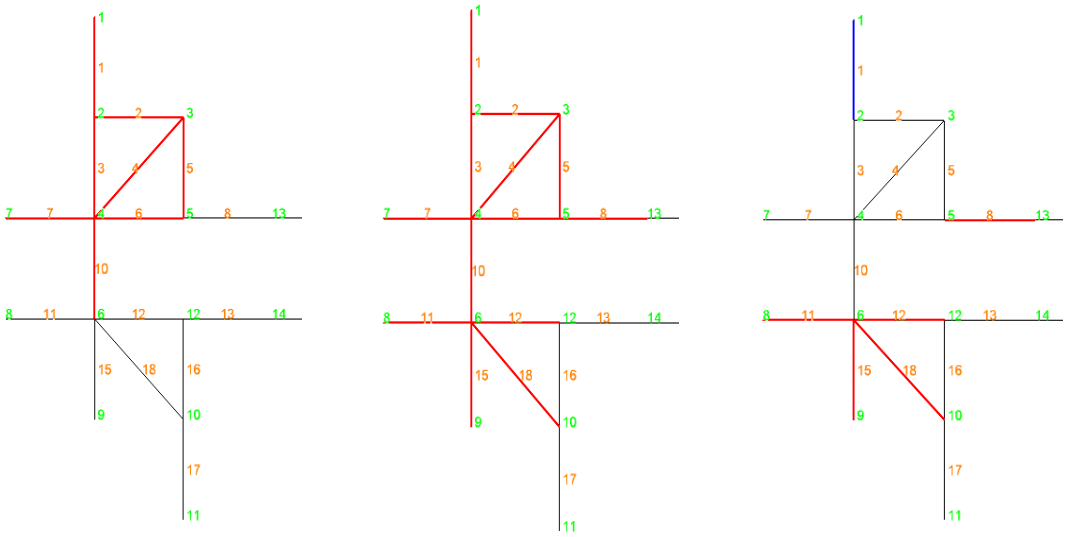


Figure 8: Maximum work zone length Minimum distance between work zones Impossible combinations for object 1

3.3.5 The combinations matrix

A linear constraint is formulated for every impossible object combination. The combinations matrix is a m-by-n matrix where m is the number of constraints and n is the sumproduct of all the objects in the network and the number of intervention types. The number of impossible object combinations and thus the number of constraints depends heavily on the difference between the thresholds for the minimum distance between work zones and the maximum work zone length constraints. With increasing difference between those two thresholds, the number of impossible object combinations grows rapidly as does the number of constraints. Below is an excerpt of the formulation of these linear constraints for the impossible combinations with respect to object 1.

Objects	1	1	2	2	3	3	3	4	4	4	5	5	5	6	6	6	7	7	7	8	8	8	9	9	9	10	10	10	11	11	11	12	12	12
Int. types	0	1	0	1	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2
Binary	0	1																		1	0	0							1	0	0	1	0	0
1 8	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
1 11	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
1 12	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

Figure 9: Excerpt of the combinations matrix: for every impossible combination, a new constraint is formulated

3.3.6 The continuity matrix

The continuity matrix ensures that exactly one intervention type is selected for every object in the network. The continuity matrix is a p-by-n matrix where p is the number of objects in the network and n is the sumproduct of all the objects in the network and the number of intervention types.

Objects	1	1	2	2	3	3	3	4	4	4	5	5	5	6	6	6	7	7	7	8	8	8	9	9	9	10	10	10	11	11	11
Int. types	0	1	0	1	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2
Binary	1	0	1	0	0	0	1	0	0	1	1	0	0	1	0	0	0	0	1	1	0	0	1	0	0	0	0	1	0	0	1

1	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	= 2
2	1	1	0	0	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	= 4
3	1	1	1	1	0	0	0	1	1	1	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	1	1	1	0	0	0	= 6
4	0	0	1	1	1	1	1	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	1	1	1	0	0	0	= 6
5	0	0	1	1	0	0	0	1	1	1	0	0	0	1	1	1	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	= 4
6	0	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1	0	0	0	1	1	1	0	0	0	= 6
7	0	0	0	0	1	1	1	1	1	1	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	= 4

Figure 10: Excerpt of the continuity matrix for objects 1 to 7

The right hand side of the continuity matrix shows the number of connections for every object. For instance, object 1 is connected to two adjacent objects and object 2 is connected to four adjacent objects. The formulation of the continuity matrix does not need any source or sink nodes.

3.3.7 Additional combinations constraint

In a road network, there exist additional reasons to prohibit the simultaneous selection of specific network objects to be part of a work zone besides the maximum work zone length and the minimum distance between work zones constraint. One such reason is avoiding traffic jams due to objects being subject to an intervention and thus reducing the capacity of these road segments by changing the traffic configurations.

Assuming object 8 in the artificial network is being intervened on, you would not want to have a simultaneous intervention on objects 13 and 14 because this could create big traffic disturbances between the left and the right part of the network.

The setup of these undesired object combinations is identical to the one explained for both work zone constraints.

3.4 Optimization

The optimization of the mixed-integer linear problem is done in Microsoft Excel after the matrices from the MATLAB calculation have been imported. The problem is set up as a multiplication of matrices.

Maximize:	(benefit - costs) * binary
Subject to:	continuity_matrix * binary = RHS_continuity
	combinations_matrix * binary ≤ 1
	costs * binary ≤ budget

Figure 11: Setup of the optimization problem in Microsoft Excel

3.5 Results of the optimization

The optimal grouping of interventions into work zones for the artificial example is illustrated in the figure below. Two work zones have been selected and all constraints have been satisfied. The shortest path from any pair of nodes, where both nodes are part of the same work zone, does not exceed the maximum work zone length and the shortest path between any pair of nodes, where the nodes are not part of the same network, does not violate the minimum distance between work zones.

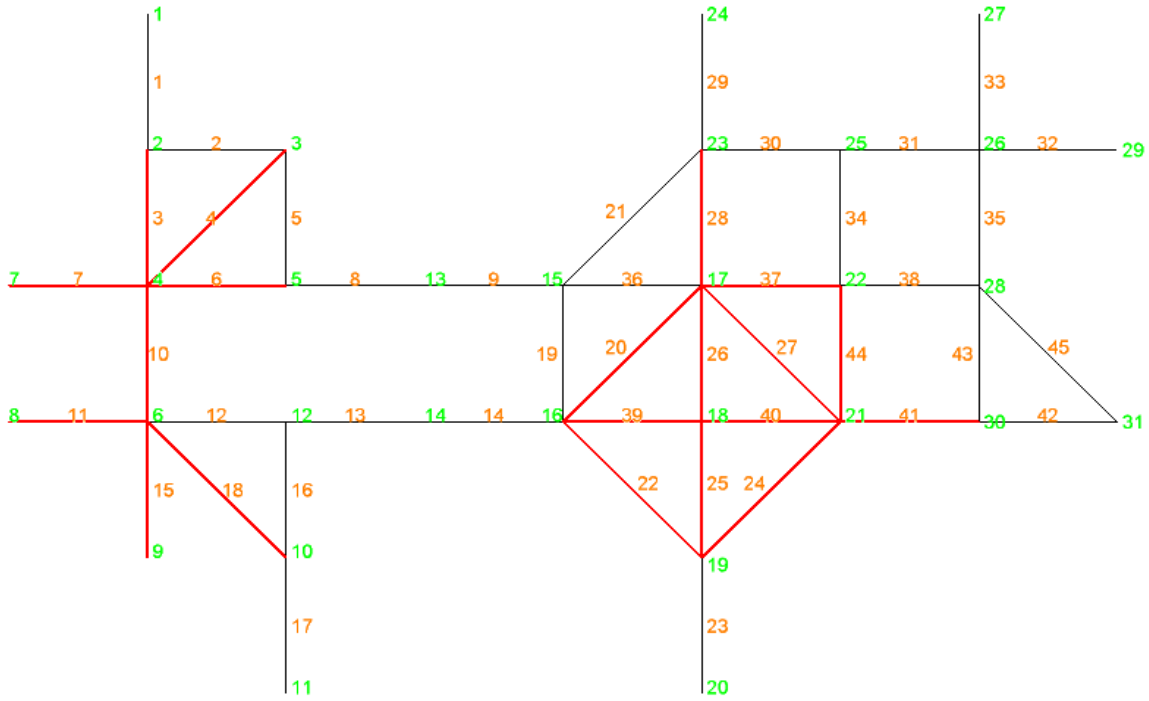


Figure 12: Optimal grouping of work zones in the artificial network

3.6 Calculation times

The computational resources and time needed for the setup of the matrices depend largely on the minimum distance between work zones. The paths that need to be calculated for this constraint are in general significantly larger than the paths calculated for the maximum work zone length constraint. With an increasing threshold for the path length, the number of possible paths for a given object can grow exponentially depending on the network characteristics. The total number of paths calculated depends mainly on the number of intersections in the network and the intersection density.

4. Network in the Canton of Valais

4.1 The network

The algorithm presented in the previous chapter is applied to the road network in the Canton of Valais. The represented network consists of the main roads in the Canton of Valais, containing the motorways, the primary and the secondary roads. The examined network has a total length of 671 km and is divided into 567 objects, with an object being a road segment, a bridge or a tunnel.

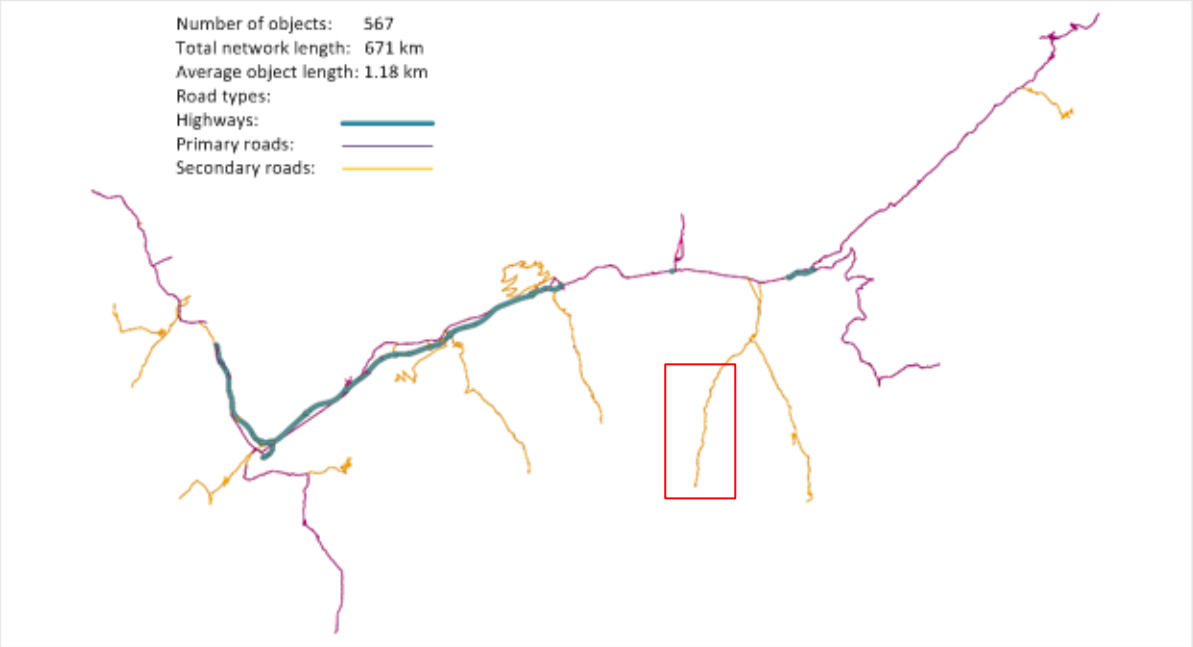


Figure 13: The examined network in the Canton of Valais

4.2 Intervention types and condition states

For every object in the network, three types of interventions can be performed:

- Intervention type 0: Do nothing
- Intervention type 1: Low long term benefit intervention
- Intervention type 2: High long term benefit intervention

All network objects are considered to be in one of five discrete condition states (CS), with decreasing road conditions from CS 1 to CS 5. Performing interventions on objects in CS 1 yield very low benefits whereas intervening on objects in CS 5 results in very high benefits. Having no knowledge on the actual road condition states in the Canton of Valais, the network objects are assigned random condition states following a lognormal distribution with user-defined distribution moments. The investigated scenarios use exactly the same condition states in order to properly assess the influence of changes in the budget as well as in the maximum work zone length and the minimum distance between work zones constraint.

In this simplified scenario, the benefits and the costs are defined arbitrarily assuming a non-linear relationship between the condition states and the corresponding intervention benefits. Benefits and costs are expressed in monetary units (MU).

Table 3: Condition states, benefits and costs

CS	Road description	Low Benefit	High Benefit	Costs
1	Like new	0	1	1
2	Good	1	2	1
3	Acceptable	2	4	1
4	Insufficient	4	8	1,5
5	Bad	8	16	1,5

4.3 Scenarios

Four different scenarios were investigated by means of changes in the budget, the maximum work zone length and the minimum distance between work zones. The objective of the optimization remains the same for all scenarios.

Table 4: Investigated scenarios

	Budget (MU)	Maximum work zone length	Minimum distance between work zones
Unit	[MU / 1000 m]	[m]	[m]
Scenario 1	50	5000	5000
Scenario 2	50	5000	8000
Scenario 3	40	6000	8000
Scenario 4	Unlimited	5000	8000

4.4 Optimization results

The optimal grouping of work zones has been obtained by running the optimization model for the different scenarios. The results of the optimization can be seen in the subsequent table.

Table 5: Optimization results

Results	Number of objects selected	Objective function	Total number of constraints
Unit	[-]	[MU]	[-]
Scenario 1	23	483,3	2911
Scenario 2	29	456,2	6109
Scenario 3	17	386,6	5196
Scenario 4	176	872,3	6108

In scenario 1, the budget is restricted to 50 MU, the maximum work zone length and the minimum distance between work zones are set to 5000 m. The objective function has a value of 483.3 MU and the optimization model contains a total of 2911 constraints.
In scenario 2, the budget and the maximum work zone length remain unchanged (with respect to scenario 1), whereas the minimum distance between work zones is increased to 8000 m. Due to this increase, the gap between the minimum distance between work zones and the maximum work zone length increases as well. A larger gap (3000 m) between those two work zone constraints causes the number of combination constraints to rise and the total number of constraints reaches 6109. The number of continuity constraints remains the same for all four scenarios. A slight drop in the value of the objective function is observed.
In scenario 3, the minimum distance between work zones remains constant (with respect to scenario 2). The budget is lowered to 40 MU and the maximum work zone length is increased to 6000 m. This increase reduces the gap between both work zone constraints from 3000 to 2000 m and thus the number of impossible object combinations drops again. Scenario 3 has a total of 5196 constraints. The decrease in the value of the objective function is cause by the reduction in the available budget.
In scenario 4, both work zone length constraints are equal to scenario 2, whereas the budget constraint is lifted. The total number of constraints drops from 6109 to 6108 (no budget constraint).
A strong increase in the total number of objects subject to an intervention from 29 to 176 is observed due to the unlimited budget. Although the number of objects being intervened on rises by a factor of 6, the value of the objective function rises only by a factor of 2. In the first three scenarios, the objects that are subject to an intervention are mainly in condition states 4 and 5. In scenario 4, a lot of objects in good condition states and thus lower benefits are subject to interventions because of the unrestricted budget.

4.5 Graphical representation

The figure below illustrates the graphical representation of the optimization results. The nodes represent the possible interventions for a given object. The nodes are listed in the format of xx.yy with xx being the object number and yy being the intervention type.

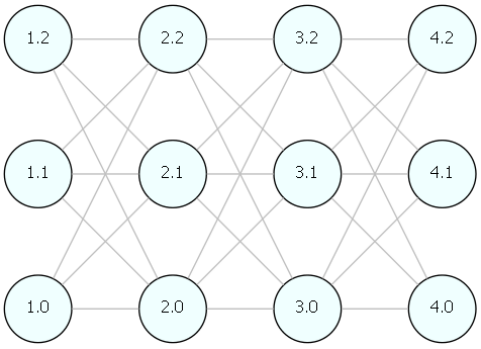
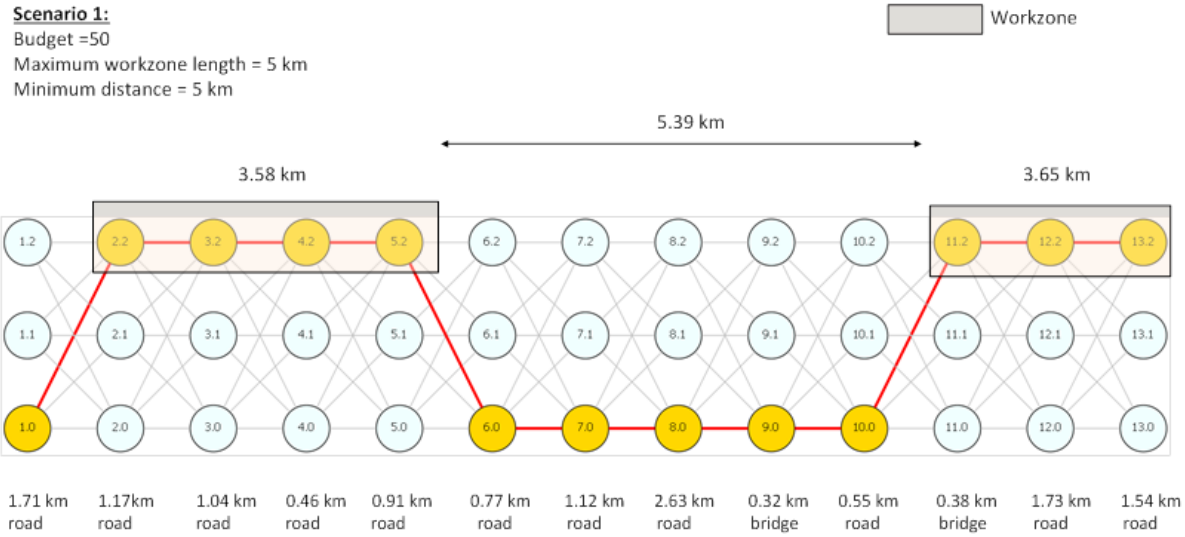


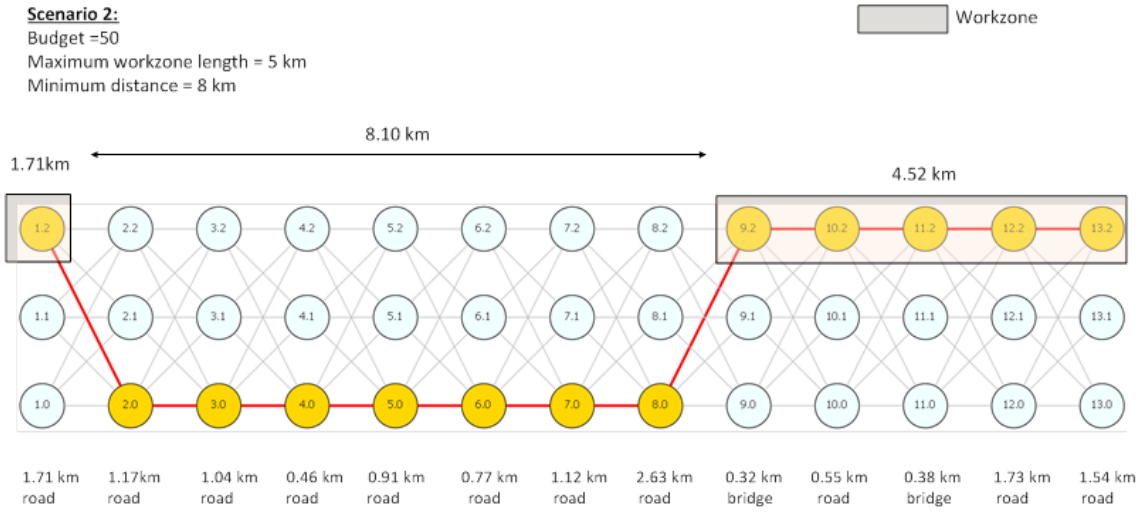
Figure 14: Graphical representation with nodes and links

The optimization results are illustrated for a small portion of the road network in the Canton of Valais (red rectangle in Figure 13).

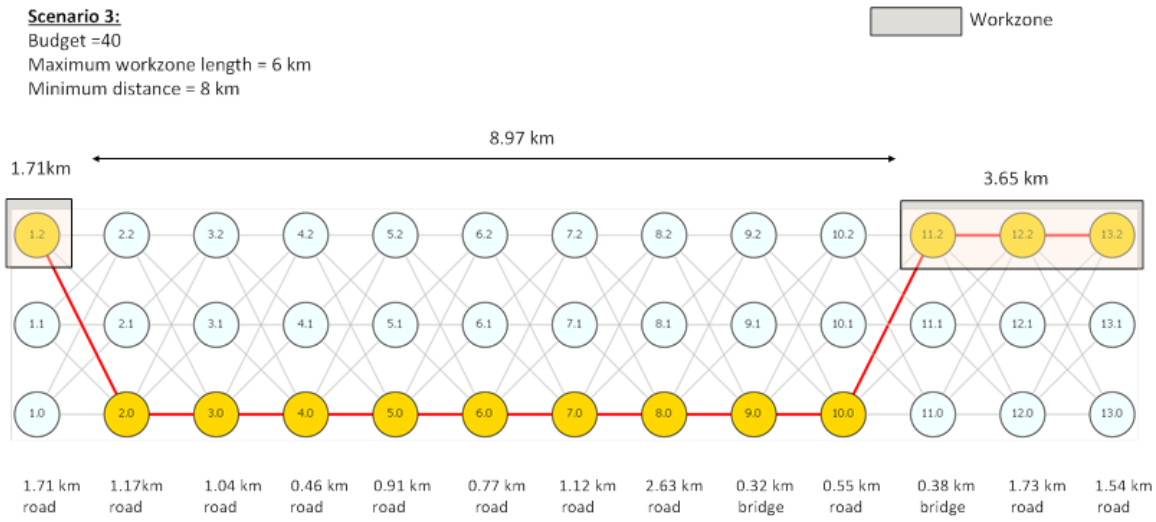
4.5.1 Scenario 1



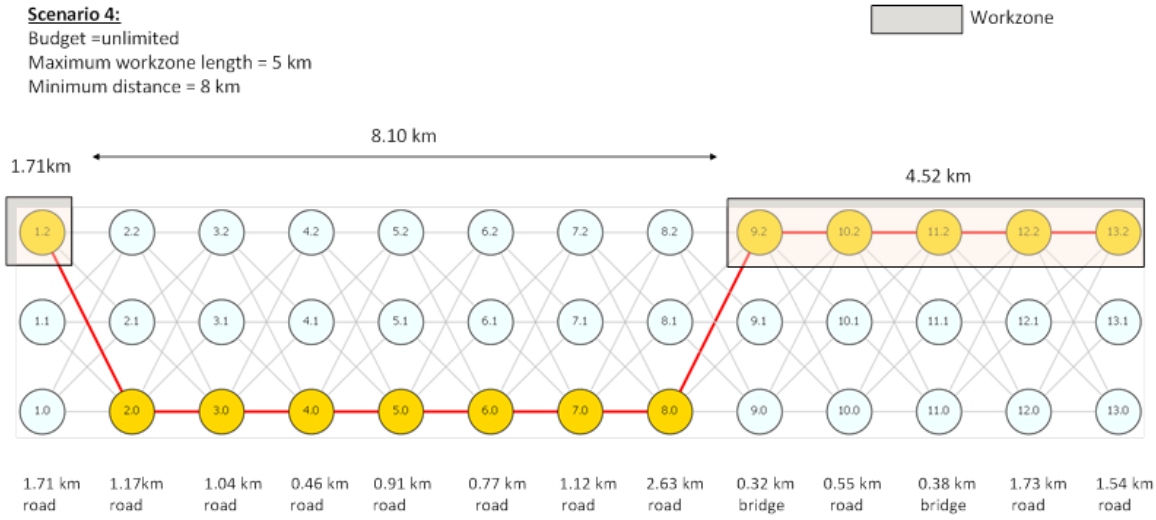
4.5.2 Scenario 2



4.5.3 Scenario 3



4.5.4 Scenario 4



5. Conclusions and outlook

A mixed-integer linear optimization model used for the determination of optimal work zones in a road network has been presented. Its applicability has been demonstrated by optimizing work zones in the road network in the Canton of Valais, Switzerland. An advantage over previous researches is that the model incorporates, in addition to the maximum work zone length, the minimum distance between work zones while still being linear. The computational resources and time required for solving a mixed-integer non-linear optimization problem vastly exceed the effort solving the exact same problem set up in a linear manner. Besides both work zone length constraints, the proposed model includes the budget constraint.
In order to cope with large scale networks when setting up the matrices for the optimization, a routing algorithm allowing a robust and efficient calculation of the continuity and combinations matrices has been developed. Thus the amount of manual work required for setting up the optimization problem has been reduced significantly.
The model has been tested on the road network in the Canton of Valais, the latter being divided into 567 different infrastructure objects. GIS data has been used and three different types of roads were incorporated in the network; motorway roads, as well as primary and secondary roads. Three different types of possible interventions were imposed on each network object. Although the cost and benefit estimation of interventions has been simplified by choosing arbitrary values (MU), the applicability of the presented model has been demonstrated thoroughly.

The performance of the algorithm depends heavily on the characteristics and the scale of the examined network, the maximum work zone length and the minimum distance between work zones constraints. The computational efforts needed for setting up the required matrices for the optimization increase with an increasing number of intersections in the network, the intersection density and the gap between the minimum distance between work zones and the maximum work zone length. Improvements in the algorithm or computational performance can be helpful.
Accurate cost and benefit estimations of interventions are required in order to get meaningful optimization results. The setup of the problem remains unchanged.
In the presented model GIS data has only been used to illustrate the results of the optimization, the input file has been set up by hand. Algorithms setting up the input file by interacting with GIS data could further reduce the amount of manual labour required.
The model can be applied for long term intervention planning taking into account several planning periods. The deterioration of the road network over the investigated time frame can be modelled using Markov models. Monte Carlo simulations allow the quantification of uncertainty in the input parameters (e.g. road condition states, traffic volume, intervention costs) and can be used to make predictions on future costs with given confidence intervals.
Keeping the model linear while incorporating the above mentioned improvements (e.g. integrating traffic configurations) will be a challenging task.

6. References

[1] B. F. Kallas, “Pavement maintenance and rehabilitation,” 1985.

[2] B. Adey, A. A. Khaled, M. M. Maher, P. L. Durango-Cohen, R. Guido, H. Rade, A. Bryan, B. Eugen, C.-Y. Y. Chu, D. M. Frangopol, M. J. Kallen, J. M. V Noortwijk, G. Roelfstra, R. Hajdin, E. Brühwiler, and J. O. Sobanjo, “Probabilistic Models for Life-Cycle Performance of Deterioration Structures: Review and Future Directions,” *ASCE Conf. Infrastruct. Manag. Plan.*, vol. 6, no. 1, pp. 17–32, Feb. 2004.

[3] Y. Wang, R. L. Cheu, and T. F. Fwa, “Highway maintenance scheduling using genetic algorithm with microscopic traffic simulation,” in *Proceeding of TRB 81st Annual Meeting*, 2002.

[4] B. Adey, T. Herrmann, K. Tsafatinos, J. Luking, N. Schindele, and R. Hajdin, “Methodology and base cost models to determine the total benefits of preservation interventions on road sections in Switzerland,” *Struct. Infrastruct. Eng. Maintenance, Manag. Life-Cycle Des. Perform.*, vol. 8, no. 7, pp. 639–654, 2010.

[5] A. Ferreira, L. Picado-Santos, and A. Antunes, “A Segment-linked Optimization Model for Deterministic Pavement Management Systems,” *Int. J. Pavement Eng.*, vol. 3, no. 2, pp. 95–105, 2002.

[6] N. Sathaye and S. Madanat, “A bottom-up solution for the multi-facility optimal pavement resurfacing problem,” *Transp. Res. Part B Methodol.*, vol. 45, no. 7, pp. 1004–1017, Aug. 2011.

[7] R. Hajdin and H. P. Lindenmann, “Algorithm for the Planning of Optimum Highway Work Zones,” *J. Infrastruct. Syst.*, vol. 13, no. 3, pp. 202–214, 2007.

[8] ERA-NET, “Stakeholders Benefits and Road Interventions Strategies,” 2012.

[9] C. H. Chen, P. Schonfeld, and J. Paracha, “Work zone optimization for two-lane highway resurfacing projects with an alternate route,” *Transp. Res. Rec.*, vol. 1911, pp. 51–66, 2005.

[10] Y. Chang, O. B. Sawaya, and A. K. Ziliaskopoulos, “A Tabu Search Based Approach for Work Zone Scheduling,” in *Proceeding of the TRB 80th Annual Meeting*, 2000, no. 01–2950.

7. Appendix

7.1 objects_selector_1.m

```
function [valid_objects] = objects_selector_1...
   (object,position,path,network_data,adjacency_matrix,mwzl)

% paths with maximum workzone length constraint
```

INPUT ARGUMENTS

```
%{
'object': current/last object that has been added to 'path'
'position': number of objects in 'path' / index of the last
          non-zero entry in 'path'
'path': n x 2 matrix
   - n: number of objects in the network
   - column 1: objects that have been selected for the current path
   - column 2: lengths of the selected objects
'network_data': n x 4 matrix
   - column 1: objects of the network
   - column 2: lengths of the objects
   - column 3: node 1 of the object
   - column 4: node 2 of the object
'adjacency_matrix': n x n matrix containing the connections
                 between the objects
'mwzl': maximum workzone length
%}
```

ADJACENT OBJECTS

```
%{
'adjacent_objects' are the objects that are connected (share at least
one common node) to 'object'
%}

% find() returns the indices of a vector
object_connections = find(adjacency_matrix(object,:) == 1);
object_type = length(object_connections);
adjacent_objects = zeros(object_type,4);

%{
'adjacent_objects':
```

```
- column 1: connecting/adjacent objects
- column 2: lenghts of connecting/adjacent objects
- column 3: node 1 of connecting/adjacent objects
- column 4: node 2 of connecting/adjacent objects
%}

for j = 1 : object_type
   adjacent_objects(j,1) = object_connections(j);
   adjacent_objects(j,2:4) = network_data(object_connections(j),2:4);
end
```

POSSIBLE OBJECTS

```
%{
'possible_objects' are 'adjacent_objects' that fulfill the
following two constraints:
- the maximium workzone lenght must not be exceeded when the
  object is added to 'path', and
- object is not yet part of 'path'
%}

possible_objects = zeros(object_type,4);

for j = 1 : object_type
   if and(adjacent_objects(j,1) ~= path(:,1),...
      sum(path(:,2)) + adjacent_objects(j,2) <= mwzl)
   %{
   and()
   and(true,true) = true
   and(true, false) = false
   and(false, true) = false
   and(false, false) = false
   %}
      possible_objects(j,:) = adjacent_objects(j,:);
   end
end

% Delete the preallocated rows that have not been used
possible_objects(possible_objects(:,1) == 0,:) = [];

%{
length() returns the number of elements of the largest dimension of
a matrix. If you want to specify the number of columns or rows:
- for columns: length(1,:);
- for rows: length(:,1);
- or use [rows, columns] = size(matrix);
%}
```

VALID OBJECTS

```
%{
Additional condition for 'valid_objects':
(conditions for 'possible_objects' still apply for 'valid_objects')
- Do not create an intersection when added to 'path':

 A path has exactly one start and one end node.
 A start/end node is a node that is connected to exactly one object
 in the current path; thus a path with intersection(s) is a path
 (= collection of objects) that has 3 or more start/end nodes.
 An object that creates an intersection is still a valid/eligible
 object as long as this object does not create an additional
 start/end node.
%}

valid_objects = zeros(length(possible_objects(:,1)),4);

if position == 1
   % 'path' contains only zero-elements
   valid_objects = possible_objects;
end

if position >= 2

   % find the end node of 'path'
   nodes_1 = network_data(object,3:4);
   nodes_2 = network_data(path((position - 1),1),3:4);
   end_node = nodes_1(nodes_1 ~= intersect(nodes_1,nodes_2));
   %{
   Find commmon elements in two vectors A and B:
   - A(ismember(A,B)) / B(ismember(A,B)), or
   - intersect(A,B)
   %}

   if isempty(end_node)
      % true when previous and current object share the same nodes

      if position == 2
         % Both nodes are end nodes
         valid_objects = possible_objects;
         % return stops any further evaluation of this function
         return
      end

      if position >= 3
         nodes_3 = network_data(path((position - 2),1),3:4);
         end_node = intersect(nodes_1,nodes_3);
         %{
```

```
            Other possible syntax:
            - end_node = intersect(nodes_2,nodes_3);
            - nodes_1 and nodes_2 are equal vectors because
              'object' and the previous object (path((position - 1),1))
              share the same nodes
            %}
        end
    end

    %{
    'possible_objects' that are connected to 'end_node'
    of 'path' are 'valid_objects'
    %}
    for j = 1 : length(possible_objects(:,1))
        if any(possible_objects(j,3:4) == end_node)
            valid_objects(j,:) = possible_objects(j,:);
        end
    end
end

% Delete the preallocated rows that have not been used
valid_objects(valid_objects(:,1) == 0,:) = [];

end
```

Published with MATLAB® R2013a

7.2 objects_selector_2.m

```
function [valid_objects] = objects_selector_2...
   (object,position,path,network_data,adjacency_matrix,mdbw)
```

```
% paths with minimum distance between workzones constraint
```

INPUT ARGUMENTS

```
%{
'object': current/last object that has been added to 'path'
'position': number of objects in 'path' / index of the last
          non-zero entry in 'path'
'path': n x 2 matrix
   - n: number of objects in the network
   - column 1: objects that have been selected for the current path
   - column 2: lengths of the selected objects
'network_data': n x 4 matrix
   - column 1: objects of the network
   - column 2: lengths of the objects
   - column 3: node 1 of the object
   - column 4: node 2 of the object
'adjacency_matrix': n x n matrix containing the connections
                 between the objects
'mdbw': minimum distance between workzones
%}
```

ADJACENT OBJECTS

```
%{
'adjacent_objects' are the objects that are connected (share at least
one common node) to 'object'
%}

% find() returns the indices of a vector
object_connections = find(adjacency_matrix(object,:) == 1);
object_type = length(object_connections);
adjacent_objects = zeros(object_type,4);

%{
'adjacent_objects':
- column 1: connecting/adjacent objects
- column 2: lenghts of connecting/adjacent objects
- column 3: node 1 of connecting/adjacent objects
```

```
- column 4: node 2 of connecting/adjacent objects
%}

for j = 1 : object_type
   adjacent_objects(j,1) = object_connections(j);
   adjacent_objects(j,2:4) = network_data(object_connections(j),2:4);
end
```

POSSIBLE OBJECTS

```
%{
'possible_objects' are 'adjacent_objects' that are not
yet part of 'path'
%}

possible_objects = zeros(object_type,4);

for j = 1 : object_type
   if adjacent_objects(j,1) ~= path(:,1)
         possible_objects(j,:) = adjacent_objects(j,:);
   end
end

% Delete the preallocated rows that have not been used
possible_objects(possible_objects(:,1) == 0,:) = [];
```

VALID OBJECTS

```
%{
'valid_objects' are 'possible_objects' with the following constraint:
- Do not create an intersection when added to 'path':

 A path has exactly one start and one end node.
 A start/end node is a node that is connected to exactly one object
 in the current path; thus a path with intersection(s) is a path
 (= collection of objects) that has 3 or more start/end nodes.
 An object that creates an intersection is still a valid/eligible
 object as long as this object does not create an additional
 start/end node.
%}

% Initialize 'valid_objects' with zeros
valid_objects = zeros(length(possible_objects(:,1)),4);

if (sum(path(:,2)) - path(1,2)) >= mdbw
   % current path is completed if true and 'valid_objects' is empty
```

```
    %{
    objects_selector_2() is used to calculate all the paths starting
    with a given object (= path(1,1)). Objects are added to 'path' as
    long as the minimum distance between workzones is not yet exceeded.
    The objects in the calculated paths are the objects that can not be
    selected for another workzone with respect to path(1:1).
    For this reason, path(1,2) is substracted from sum(path(:,2)).

    The condition in the above if-statement could theoretically be checked
    at the beginning of this function, but it is done here because the
    size of 'possible_objects' is not known at the beginning of the
    function.
    %}

    % 'valid_objects' has to be initalized first before its zero-columns
    % are deleted again, otherwise an error-message will occur
    valid_objects(valid_objects(:,1) == 0,:) = [];
    % return stops any further evaluation of this function
    return
end

if position == 1
    % 'path' contains only zero-elements
    valid_objects = possible_objects;
end

if position >= 2

    % Find the end node of 'path'
    nodes_1 = network_data(object,3:4);
    nodes_2 = network_data(path((position - 1),1),3:4);
    end_node = nodes_1(nodes_1 ~= intersect(nodes_1,nodes_2));
     %{
    Find commmon elements in two vectors A and B:
    - A(ismember(A,B)) / B(ismember(A,B)), or
    - intersect(A,B)
    %}

    if isempty(end_node)
        % true when previous and current object share the same nodes

        if position == 2
            % Both nodes are end nodes
            valid_objects = possible_objects;
            return
        end
        if position >= 3
            nodes_3 = network_data(path((position - 2),1),3:4);
            end_node = intersect(nodes_1,nodes_3);
```

```
        %{
        Other possible syntax:
        - end_node = intersect(nodes_2,nodes_3);
        - nodes_1 and nodes_2 are equal vectors because
          'object' and the previous object (path((position - 1),1))
          share the same nodes
        %}
      end
    end

    %{
    'possible_objects' that are connected to 'end_node'
    of 'path' are 'valid_objects'
    %}
    for j = 1 : length(possible_objects(:,1))
      if any(possible_objects(j,3:4) == end_node)
        valid_objects(j,:) = possible_objects(j,:);
      end
    end
  end

  % Delete the preallocated rows that have not been used
  valid_objects(valid_objects(:,1) == 0,:) = [];

end
```

Published with MATLAB® R2013a

7.3 storage_input.m

```
function [storage] = storage_input(storage,position,valid_objects)
```

INPUT ARGUMENTS

```
%{
'storage': matrix with n rows and variable number of columns
        n: number of objects in the network
'position': number of objects in 'path' / index of the last
         non-zero entry in 'path'
'valid_objects': objects that are eligible to be added to a path
%}
```

ASSIGNING OBJECTS TO STORAGE

```
%{
storage_input() is called from path_calculator_1() / path_calculator_2()
if 'valid_objects' contains two or more objects.
%}

l = 1;

for j = 1 : (length(valid_objects(:,1)) - 1)
   %{
   (length(valid_objects(:,1)) - 1):
   - first object from 'valid_objects' is used in the current
     path calculation
   - remaining objects are assigned to storage...
     (position,1:(length(valid_objects(:,1)) - 1))
   %}
   storage(position,l) = valid_objects((l + 1),1);
   l = l + 1;
end

end
```

Published with MATLAB® R2013a

7.4 storage_output.m

```
function [object,position,storage] = storage_output(storage)
```

```
% This function returns an error message when 'storage' is a zero-matrix
```

INPUT ARGUMENTS

```
%{
'storage': matrix with n rows and variable number of columns
        n: number of objects in the network
%}
```

OUTPUT REQUEST

```
%{
storage_output() is called from all_paths_1() / all_paths_2() only if
'storage' contains at least one non-zero element
%}

j = 0;
len = length(storage(:,1));

while true
   position = len - j;
   row = storage(position,:);
   if any(row)
      non_zeros = find(row);
      object = row(non_zeros(1));
      storage(position,non_zeros(1)) = 0;
     break;
     % break-statment will terminate the loop
   end
      j = j + 1;
end
```

```
end
```

Published with MATLAB® R2013a

7.5 path_calculator_1.m

```
function [path,storage] = path_calculator_1...
   (object,path,storage,position,network_data,adjacency_matrix,mwzl)

% paths with maximum workzone length constraint
```

INPUT ARGUMENTS

```
%{
'object': current/last object that has been added to 'path'
'path': n x 2 matrix
   - n: number of objects in the network
   - column 1: objects that have been selected for the current path
   - column 2: lengths of the selected objects
'storage': matrix with n rows and variable number of columns
'position': number of objects in 'path' / index of the last
         non-zero entry in 'path'
'network_data': n x 4 matrix
   - column 1: objects of the network
   - column 2: lengths of the objects
   - column 3: node 1 of the object
   - column 4: node 2 of the object
'adjacency_matrix': n x n matrix containing the connections
               between the objects
'mwzl': maximum workzone length
%}
```

CHECK OBJECT LENGTH

```
if network_data(object,2) > mwzl
   %{
   Check if object is longer than the maximium workzone length.
   This if-statement should usually never be entered because no
   object in the road network should be longer than the maximum
   workzone length.
   %}
   return
end
```

CALCULATE A SINGLE PATH

```
%{
path_calculator_1() calculates a single path where 'object' is the last
object in the current path. If several objects (valid_objects) are
eligible to be added to the path, then the object with the lowest
number (= name/reference) is added to the current path an the other
object(s) are assigned to 'storage'.
%}

path(position,:) = network_data(object,1:2);
[valid_objects] = objects_selector_1...
   (object,position,path,network_data,adjacency_matrix,mwzl);

while true
   position = position + 1;

   if length(valid_objects(:,1)) >= 2
      storage = storage_input(storage,position,valid_objects);
   end

   %{
   isempty() returns logical 1 (true) if 'valid_objects' is
   an empty matrix (= contains no elements).
   An empty matrix is not a zero-matrix!
   %}
   empty = isempty(valid_objects);
   if not(empty)
      %{
      not(true) = false
      not(false) = true
      %}
      path(position,:) = valid_objects(1,1:2);
   else
      break
      % break-statment will terminate the loop
   end

   [valid_objects] = objects_selector_1(valid_objects(1,1),...
      position,path,network_data,adjacency_matrix,mwzl);
   %{
   Other possible syntax:
   path(position,1) instead of valid_objects(1,1) because
   valid_objects(1,1) is added to path(position,1)
   %}
end
```

End

7.6 path_calculator_2.m

```
function [path,storage] = path_calculator_2...
   (object,path,storage,position,network_data,adjacency_matrix,mdbw)

% paths with minimum distance between workzones constraint
```

INPUT ARGUMENTS

```
%{
'object': current/last object that has been added to 'path'
'path': n x 2 matrix
   - n: number of objects in the network
   - column 1: objects that have been selected for the current path
   - column 2: lengths of the selected objects
'storage': matrix with n rows and variable number of columns
'position': number of objects in 'path' / index of the last
         non-zero entry in 'path'
'network_data': n x 4 matrix
   - column 1: objects of the network
   - column 2: lengths of the objects
   - column 3: node 1 of the object
   - column 4: node 2 of the object
'adjacency_matrix': n x n matrix containing the connections
               between the objects
'mdbw': minimum distance between workzones
%}
```

CALCULATE A SINGLE PATH

```
%{
path_calculator_2() calculates a single path where 'object' is the last
object in the current path. If several objects (valid_objects) are
eligible to be added to the path, then the object with the lowest
number (= name/reference) is added to the current path an the other
object(s) are assigned to 'storage'.
%}

path(position,:) = network_data(object,1:2);
[valid_objects] = objects_selector_2...
   (object,position,path,network_data,adjacency_matrix,mdbw);

while true
   position = position + 1;
```

```
    if length(valid_objects(:,1)) >= 2
        storage = storage_input(storage,position,valid_objects);
    end

    %{
    isempty() returns logical 1 (true) if 'valid_objects' is
    an empty matrix (= contains no elements).
    An empty matrix is not a zero-matrix!
    %}
    empty = isempty(valid_objects);
    if not(empty)
        %{
        not(true) = false
        not(false) = true
        %}
        path(position,:) = valid_objects(1,1:2);
    else
        break
        % break-statment will terminate the loop
    end

    [valid_objects] = objects_selector_2(valid_objects(1,1),...
        position,path,network_data,adjacency_matrix,mdbw);
     %{
    Other possible syntax:
    path(position,1) instead of valid_objects(1,1) because
    valid_objects(1,1) is added to path(position,1)
    %}
end

end
```

Published with MATLAB® R2013a

7.7 all_paths_1.m

```
function [worksites, number] =...
   all_paths_1(object,network_data,adjacency_matrix,mwzl)
```

```
% paths with maximum workzone length constraint
```

INPUT ARGUMENTS

```
%{
'object': current/last object that has been added to 'path'
'network_data': n x 4 matrix
   - n: number of objects in the network
   - column 1: objects of the network
   - column 2: lengths of the objects
   - column 3: node 1 of the object
   - column 4: node 2 of the object
'adjacency_matrix': n x n matrix containing the connections
                    between the objects
'mwzl': maximum workzone length
%}
```

INITIALIZING

Initialize path, storage, worksites, position and j

```
path = zeros(length(adjacency_matrix),2);
storage = zeros(length(adjacency_matrix),20);
%{
Make sure that storage has enough columns in order to be able to store
all the eligible objects. The number of columns does not matter as
long as it exceeds the largest intersection.
%}
worksites = zeros(length(adjacency_matrix),1000);
%{
'worksites' contains all the possible paths starting with 'object'
Make sure that enough columns are preallocated in order to store
every calculated path
%}
position = 1;
j = 1;
```

CALCULATE ALL PATHS

```
%{
all_paths_1() calculates all possible paths that start with 'object'
and stores the calculated paths in 'worksites'
%}

while true

   % Calculate a single path
   [path,storage] = path_calculator_1(object,path,storage,...
      position,network_data,adjacency_matrix,mwzl);
   % Store calculated path in 'worksites'
   worksites(:,j) = path(:,1);

   if all(all(storage == 0))
      % true if storage contains only zero-elements
      break;
      % break-statement will terminate the loop
   else
      % Request output for next path calculation
      [object,position,storage] = storage_output(storage);
   end

   path(position:end,:) = 0;
   j = j + 1;
end

% Delete the preallocated worksites columns that have not been used
worksites = worksites(:,any(worksites));
number = length(worksites(1,:));

end
```

Published with MATLAB® R2013a

7.8 all_paths_2.m

```
function [worksites, number] =...
   all_paths_2(object,network_data,adjacency_matrix,mdbw)
```

```
% paths with minimum distance between workzones constraint
```

INPUT ARGUMENTS

```
%{
'object': current/last object that has been added to 'path'
'network_data': n x 4 matrix
   - n: number of objects in the network
   - column 1: objects of the network
   - column 2: lengths of the objects
   - column 3: node 1 of the object
   - column 4: node 2 of the object
'adjacency_matrix': n x n matrix containing the connections
                    between the objects
'mdbw': minimum distance between workzones
%}
```

INITIALIZING

Initialize path, storage, worksites, position and j

```
path = zeros(length(adjacency_matrix),2);
storage = zeros(length(adjacency_matrix),20);
%{
Make sure that storage has enough columns in order to be able to store
all the eligible objects. The number of columns does not matter as
long as it exceeds the largest intersection.
%}
worksites = zeros(length(adjacency_matrix),1000);
%{
'worksites' contains all the possible paths starting with 'object'
Make sure that enough columns are preallocated in order to store
every calculated path
%}
position = 1;
j = 1;
```

CALCULATE ALL PATHS

```
%{
all_paths_2() calculates all possible paths that start with 'object'
and stores the calculated paths in 'worksites'
%}

while true

  % Calculate a single path
  [path,storage] = path_calculator_2(object,path,...
     storage,position,network_data,adjacency_matrix,mdbw);
  % Store calculated path in 'worksites'
  worksites(:,j) = path(:,1);

  if all(all(storage == 0))
     % true if storage contains only zero-elements
     break;
      % break-statement will terminate the loop
  else
     % Request output for next path calculation
     [object,position,storage] = storage_output(storage);
  end

  path(position:end,:) = 0;
  j = j + 1;
end

% Delete the preallocated worksites columns that have not been used
worksites = worksites(:,any(worksites));
number = length(worksites(1,:));

end
```

Published with MATLAB® R2013a

7.9 complete_network_1.m

```
function [all_worksites_1,unique_worksites_1] =...
   complete_network_1(network_data,adjacency_matrix,mwzl)
```

```
% paths with maximum workzone length constraint
```

INPUT ARGUMENTS

```
%{
'network_data': n x 4 matrix
   - n: number of objects in the network
   - column 1: objects of the network
   - column 2: lengths of the objects
   - column 3: node 1 of the object
   - column 4: node 2 of the object
'adjacency_matrix': n x n matrix containing the connections
                    between the objects
'mwzl': maximum workzone length
%}
```

PREALLOCATION

```
%{
Important remark: make sure that 'all_worksites_1' has enough space;
if 'all_worksites_1' does not have enough preallocated columns to store
all the calculated paths, then this will result in an error message
or some paths will be missing.
%}

all_worksites_1 = zeros(length(adjacency_matrix),1000000);
column = 1;
```

CALCULATE ENTIRE NETWORK

```
%{
Calculate all possible paths for every object and assign them
to 'all_worksites_1'. 'all_worksites_1' will contain a lot of double
entries and subsets of other paths. These surplus paths can be
removed (if desired) with the delete_copies_and_subsets_3()-function.
%}
```

```
for j = 1 : length(adjacency_matrix)

  [worksites, number] = all_paths_1(network_data(j,1),...
     network_data,adjacency_matrix,mwzl);
   all_worksites_1(:,column:(column + number - 1)) = worksites;
   column = column + number;
end

% Delete preallocated columns that have not been used
all_worksites_1 = all_worksites_1(:,any(all_worksites_1));
```

REMOVE SURPLUS PATHS

```
[unique_worksites_1] = ...
   delete_copies_and_subsets_3(all_worksites_1,network_data);
```

DELETE ROWS THAT CONTAIN ONLY ZEROS

```
[all_worksites_1] = shortening(all_worksites_1);
%{
'unique_worksites_1' is already shorted/stripped in
delete_copies_and_subsets_3()
%}

end
```

Published with MATLAB® R2013a

7.10 complete_network_2.m

```
function [all_worksites_2] =...
   complete_network_2(network_data,adjacency_matrix,mdbw)

% paths with minimum distance between workzones constraint
```

INPUT ARGUMENTS

```
%{
'network_data': n x 4 matrix
   - n: number of objects in the network
   - column 1: objects of the network
   - column 2: lengths of the objects
   - column 3: node 1 of the object
   - column 4: node 2 of the object
'adjacency_matrix': n x n matrix containing the connections
                    between the objects
'mdbw': minimum distance between workzones
%}
```

UNIQUE WORKSITES

```
%{
complete_network_2() does not calculate the unique paths (see output
arguments of complete_network_1()) because the computational time of
delete_copies_and_subsets_3() grows exponentially with increasing
size of 'all_worksites_2'. The size (number of paths) in 'all_worksites_2'
is usually significantly larger than the size of 'all_worksites_1'
because the minimum distance between workzones is larger than the
maximum workzone length and thus the number of possible paths.

Even in complete_network_1(), the calculation of 'unique_worksites_1' is
not needed for the setup of the minimum distance between workzones /
maximum workzone length constarint, but it is only calculated to show
all the possible worksites.
%}
```

PREALLOCATION

```
%{
Important remark: make sure that 'all_worksites_2' has enough space;
```

```
if 'all_worksites_2' does not have enough preallocated columns to store
all the calculated paths, then this will result in an error message
or some paths will be missing.
%}

all_worksites_2 = zeros(length(adjacency_matrix),1000000);
column = 1;
```

CALCULATE ENTIRE NETWORK

```
%{
Calculate all possible paths for every object and assign them
to 'all_worksites_2'. The double entries and subsets won't be
removed as explained above.
%}

for j = 1 : length(adjacency_matrix)

  [worksites, number] = all_paths_2(network_data(j,1),...
     network_data,adjacency_matrix,mdbw);
   all_worksites_2(:,column:(column + number - 1)) = worksites;
   column = column + number;
end

% Delete preallocated columns that have not been used
all_worksites_2 = all_worksites_2(:,any(all_worksites_2));
```

REMOVE ROWS THAT CONTAIN ONLY ZEROS

```
[all_worksites_2] = shortening(all_worksites_2);

end
```

Published with MATLAB® R2013a

7.11 delete_copies_and_subsets_3.m

```
function [unique_worksites_1] =...
   delete_copies_and_subsets_3(all_worksites_1,network_data)
```

INPUT ARGUMENTS

```
%{
'all_worksites_1':
   - matrix with n rows and varying number of columns
   - n: number of objects in the network
   - contains all the paths calculated in complete_network_1()
'network_data': n x 4 matrix
   - column 1: objects of the network
   - column 2: lengths of the objects
   - column 3: node 1 of the object
   - column 4: node 2 of the object
%}
```

EXPLANATIONS

```
%{
'all_worksites_1' contains a lot of double entries (same paths) and
subsets of other paths. complete_network_1() calculates all possible
paths for all the objects in the network where the object is the
starting point of the calculation; this leads to 'all_worksites_1'
containing the same path (calculated in opposite directions) twice.
For every start/end point several subsets of existing paths are
created.

'unique_worksites_1' contains all possible paths only once and has no
subsets. Thus the size (number of columns) in 'unique_worksites_1' is
less than half the size of 'all_worksites_1'.

A myriad of comparisons need to be done and this can take several
minutes (or even more) to calculate. The required time to execute all
comparisons depends on the number of paths in 'all_worksites_1'.

The number of paths calculated in complete_network_1() depends on
the following variables:
- maximum workzone length
- number of objects in the network
- number of intersections and the proximity of these intersections
  in the network; with an increasing amount of intersections and
```

```
 an increasing intersection density, the number of possible paths
 grows exponentially.

The most time for the network calculation is spent in this function
and it's child functions.
%}
```

STEP 1: REMOVE THE DOUBLE ENTRIES

```
%{
Every path, which is constrainted by the maximum workzone length
is stored exactly twice (calculated from both start/end objects)
in 'all_worksites_1'. In step 1, these double entries are removed.
%}

[all_worksites_1] = shortening(all_worksites_1);
sorted_worksites_1 = sort(all_worksites_1);
length_worksites_1 = length(sorted_worksites_1(1,:));

for j = 1 : (length_worksites_1 - 1)

  for l = (j + 1) : length_worksites_1

    if isequal(sorted_worksites_1(:,j),sorted_worksites_1(:,l))
      all_worksites_1(:,l) = 0;
      break
      %{
      As soon as the double entry is found, the inner for-loop
      can be exited because there is no other copy of
      sorted_worksites_1(:,j) in the remaining network. This
      reduces the amount of comparisons significantly.
      %}
    end
  end
end

% Delete the columns of the double entries (zero-columns)
reduced_worksites_1 = all_worksites_1(:,any(all_worksites_1));
```

STEP 2: TRANSFORM THE PATHS INTO ANOTHER FORMAT

```
%{
This transformation is done in order to speed the program up.
This transformation allows to get rid of the intersect()-function.
The execution of intersect() takes a lot of time and increases the
calculation time significantly.
```

```
%}

[reduced_worksites_2] =...
   worksites_transformer(reduced_worksites_1,network_data);
```

STEP 3: REMOVE THE SUBSETS

```
length_worksites_2 = length(reduced_worksites_2(1,:));

for j = 1 : (length_worksites_2 - 1)
   path_1 = reduced_worksites_2(:,j);

   for l = (j + 1) : length_worksites_2
      path_2 = reduced_worksites_2(:,l);

      if all((path_1 - path_2) <= 0)
         % true when path_1 is a subset of path_2
         reduced_worksites_1(:,j) = 0;
         break
      end
      if all((path_1 - path_2) >= 0)
         % true when path_2 is a subset of path_1
         reduced_worksites_1(:,l) = 0;
      end
   end
end

% Delete the columns of the subsets (zero-columns)
unique_worksites_1 = reduced_worksites_1(:,any(reduced_worksites_1));

end
```

Published with MATLAB® R2013a

7.12 find_combinations.m

```
function [combinations] = find_combinations...
   (all_worksites_1,all_worksites_2,network_data)
```

INPUT ARGUMENTS

```
%{
'all_worksites_1': all possible paths with maximum workzone
                   length constraint
'all_worksites_2': all possible paths with minimum distance
                   between workzones constraint
'network_data': n x 4 matrix
   - n: number of objects in the network
   - column 1: objects of the network
   - column 2: lengths of the objects
   - column 3: node 1 of the object
   - column 4: node 2 of the object
%}
```

FIND COMBINATIONS

```
%{
find_combinations() finds all the possible combinations of objects
in the network that violate both the maximum workzone length constraint
and the minimum distance between workzones.

Fictive example:
If object 1 is part of a workzone, then objects 4 and 5 cannot be part
of any workzone in the network:
- Objects 4 and 5 are too far away to be part of the same workzone as
  object 1 (they violate the maximum workzone length constraint)
- Objects 4 and 5 are too close to object 1 to be part of another
  workzone (they violate the minimum distance between workzones constraint)

find_combinations() calculates for every object in the network these
combinations of objects that cannot be chosen simultaneously.
%}

rows = length(network_data(:,1));

combinations = zeros(rows);

for j = 1 : rows
```

```
    % Find all paths that start with j
    indices_1 = all_worksites_1(1,:) == j;
    indices_2 = all_worksites_2(1,:) == j;

    % Find all the objects that are part of paths starting with j
    elements_1 = unique(all_worksites_1(:,indices_1));
    elements_2 = unique(all_worksites_2(:,indices_2));

    % Delete zeros because zero is not an object
    elements_1 = elements_1(elements_1 ~= 0);
    elements_2 = elements_2(elements_2 ~= 0);

    % Remove j
    elements_1 = elements_1(elements_1 ~= j);
    elements_2 = elements_2(elements_2 ~= j);

    for l = 1 : length(elements_1)
        % Find common objects
        index = (elements_2 == elements_1(l));
        % Remove the common object from 'elements_2'
        elements_2(index) = [];
    end

    combinations(j,1:length(elements_2)) = elements_2;
end

% Delete preallocated columns that have not been used
combinations = combinations(:,any(combinations));

end
```

Published with MATLAB® R2013a

7.13 establish_adajacency_matrix.m

```
function [adjacency_matrix,network_nodes] =...
   establish_adjacency_matrix(network_data)
```

INPUT ARGUMENTS

```
%{
'network_data': n x 4 matrix
   - n: number of objects in the network
   - column 1: objects of the network
   - column 2: lengths of the objects
   - column 3: node 1 of the object
   - column 4: node 2 of the object
%}
```

NETWORK NODES

```
%{
'network_nodes' is a matrix with  a varying number of rows and
two columns. The number of rows is equal to the number of
nodes in the network.
- network_nodes(:,1): nodes of the network
- network_nodes(:,2): number of objects attached to each node
%}

% Find number of nodes in the network
nodes = unique(cat(1,network_data(:,3),network_data(:,4)));
%{
- cat(dim,A,B): concatenates arrays A and B along dimension dim
  dim = 1: concatenate along rows
- unique(A): returns all the different elements contained in A
%}

network_nodes = zeros(length(nodes),2);

network_nodes(:,1) = nodes;

% Find the number of objects connected to each node
for j = 1 : length(nodes)
   network_nodes(j,2) =...
      length(find(network_data(:,3:4) == network_nodes(j,1)));
end
```

ADJACENCY MATRIX

```
%{
'adjacency_matrix' is a square matrix that contains the object
connections. 1 stands for connection and zero stands for no
connection.
It is important that the objects are listed in ascending
order starting at 1 and that no numbers are missing.
This is due to the fact that this program uses find()
in several functions; find() returns the index of an array and
the indices must be equal to the object numbers. If the objects
do not match the indices, then references to the wrong objects
will be made and wrong/impossible paths will be calculated.
%}

adjacency_matrix = zeros(length(network_data(:,1)));    % Square matrix

for j = 1 : length(network_data(:,1))

   node_1 = network_data(j,3);
   node_2 = network_data(j,4);
   [r_1,~] = find(network_data(:,3:4) == node_1);
   [r_2,~] = find(network_data(:,3:4) == node_2);

   for l = 1 : length(r_1)
      if r_1(l) == j
         % Do nothing if true (= remains 0)
      else
      adjacency_matrix(j,r_1(l)) = 1;
      end
   end

   for s = 1 : length(r_2)
      if r_2(s) == j
         % Do nothing if true (= remains 0)
      else
      adjacency_matrix(j,r_2(s)) = 1;
      end
   end

end

end
```

Published with MATLAB® R2013a

7.14 establish_continuity_matrix.m

```
function [continuity_matrix] =...
   establish_continuity_matrix(adjacency_matrix,int_types)
```

INPUT ARGUMENTS

```
%{
'adjacency_matrix': n x n matrix containing the connections
                between the objects
                n : number of objects in the network
'int_types': n x 1 matrix
          contains the number of possible intervention types for every
          object
%}
```

ESTABLISH THE CONTINUITY MATRIX

```
%{
'continuity_matrix' is a matrix with n rows and a varying number of
columns. The number of columns depends on the number of possible
intervention types for every object.

Fictive example:
Assume that every object in the network can have 3 different intervention
types; the size of 'continuity_matrix' will be n x (3 * n).
%}

continuity_matrix = zeros(length(adjacency_matrix),...
   (max(int_types) * length(adjacency_matrix)));

l = 1;

for j = 1 : length(adjacency_matrix)
   continuity_matrix(:,l:(l + int_types(j) - 1)) =...
      repmat(adjacency_matrix(:,j),1,int_types(j));
      %{
      B = repmat(A,m,n) consists of copies of A
      Size of B: [length(A(:,1))*m, length(A(1,:))*n]
      %}

   l = l + int_types(j);

end
```

```
% Delete the columns that conatin only zeros
continuity_matrix = continuity_matrix(:,any(continuity_matrix));

end
```

Published with MATLAB® R2013a

7.15 establish_combinations_matrix.m

```
function [pairs,combinations_matrix] =...
   establish_combinations_matrix(combinations,int_types)
```

INPUT ARGUMENTS

```
%{
'combinations':
matrix with n rows and varying number of columns containing the
combinations of objects that cannot be selected simultaneously
'int_types': n x 1 matrix
contains the number of possible interventions for every
object
%}
```

CALCULATE OBJECT PAIRS

```
%{
For every entry in 'combinations', an object pair is created and stored
in 'pairs'.
Fictive example:
The first row of 'combinations' contains the objects that cannot be
selcted if object 1 is selected; in this case 4 and 5 are the non-zero
entries.
- pairs(:,1) = [1 ; 4]
- pairs(:,2) = [1 ; 5]
....
%}

[rows,~] = size(combinations);
n = 1;

% Append additional column to avoid out-of-bounds error message
combinations = [combinations, zeros(rows,1)];

% Make sure that enough space is preallocated
pairs = zeros(2,1000000);

for j = 1 : rows

   l = 1;
   while combinations(j,l) ~= 0
      pairs(1,n) = j;
```

```
        pairs(2,n) = combinations(j,l);
        l = l + 1;
        n = n + 1;
    end

end

% Delete preallocated columns that have not been used
pairs = pairs(:,any(pairs));
```

DETECT AND DELETE DOUBLE ENTRIES

```
[~,n] = size(pairs);

% Sort the pairs to make them comparable
sorted_pairs = sort(pairs);

for j = 1 : (n - 1)
    for l = (j + 1) : n
        if isequal(sorted_pairs(:,j),sorted_pairs(:,l))
            pairs(:,l) = 0;
            break
        end
    end
end

% Delete the columns that contain only zeros
pairs = pairs(:,any(pairs));
```

ESTABLISH THE COMBINATIONS MATRIX

```
%{
The combinations matrix contains zeros and ones and is used to formulate
the maximum workzone length and the minimum distance between workzones
constraint. The number of rows is equal to the number of object pairs
and the number of columns depends on the number of objects in the
network and the number of possible interventions for every
object.
%}

pairs = pairs';
[n,~] = size(pairs);

pairs_matrix = zeros(n,length(int_types));

for j = 1 : n
```

```
    pairs_matrix(j,pairs(j,1)) = 1;
    pairs_matrix(j,pairs(j,2)) = 1;

end

combinations_matrix = zeros(n,sum(int_types));
l = 1;

for j = 1 : length(int_types)

    n = int_types(j);
    combinations_matrix(:,l:(l + n - 1)) = repmat(pairs_matrix(:,j),1,n);
    combinations_matrix(:,l) = 0;

    l = l + n;

end
```

```
end
```

Published with MATLAB® R2013a

7.16 main.m

LOAD THE DATA

```
% Turn the profiler on
profile on

% Define the Excel file
excel_file = 'Input_Network.xlsx';

% Define the Excel sheet
sheet = 3;

% Load the network from the input excel file
network_data = xlsread(excel_file,sheet,'A:D');

% Load the intervention types from the input excel file
int_types = xlsread(excel_file,sheet,'F:F');

% Load the maximum workzone length from the input excel file
mwzl = xlsread(excel_file,sheet,'H1');

% Load the minimum distance between workzones from the input file
% IMPORTANT REMARK: mdbw >= mwzl
mdbw = xlsread(excel_file,sheet,'J1');
```

ESTABLISH THE MATRICES NEEDED FOR THE OPTIMIZATION

```
% Establish the adjacency matrix
[adjacency_matrix,network_nodes] =...
   establish_adjacency_matrix(network_data);

% Establish the continuity matrix
[continuity_matrix] =...
   establish_continuity_matrix(adjacency_matrix,int_types);

% Calculate the paths for the maximum workzone length constraint
[all_worksites_1] =...
   complete_network_1(network_data,adjacency_matrix,mwzl);

% Calculate the paths for the minimum distance between workzones
[all_worksites_2] =...
   complete_network_2(network_data,adjacency_matrix,mdbw);

% Calculate the object combinations that cannot be chosen simultaneously
[combinations] = find_combinations...
```

```
    (all_worksites_1,all_worksites_2,network_data);

% Establish the combinations matrix
[pairs,combinations_matrix] =...
    establish_combinations_matrix(combinations,int_types);

% Create additional vectors needed for the optimazation
[RHS_continuity,object_matrix] = create_vectors...
    (adjacency_matrix,int_types);

% View the profiler
profile viewer
```

WRITE TO EXCEL FILE

This just shows how to import the matrices into Excel

```
%{
xlswrite('Output_Network.xlsx',continuity_matrix,2,'J14')

xlswrite('Output_Network.xlsx',combinations_matrix,2,'J61')

xlswrite('Output_Network.xlsx',object_matrix,2,'J10')

xlswrite('Output_Network.xlsx',pairs,2,'G61')

xlswrite('Output_Network.xlsx',RHS_continuity,2,'EI14')
%}
```

Published with MATLAB® R2013a

Remark: Only the important MATLAB functions and scripts have been published. The routing algorithms uses some small functions that are not listed above.