Nat. Hazards Earth Syst. Sci. Discuss., doi:10.5194/nhess-2017-12, 2017
Manuscript under review for journal Nat. Hazards Earth Syst. Sci.
Published: 2 February 2017

Natural Hazards and Earth System Sciences
Open Access
Discussions

Using graphs to find economically optimal safety targets for multiple lines of flood defences

Egidius Johanna Cassianus Dupuits[1], Ferdinand Lennaert Machiel Diermanse[2], and Matthijs Kok[1]
[1]Delft University of Technology, Faculty of Civil Engineering and Geosciences, P.O. Box 5048, 2600 GA Delft, Netherlands
[2]Deltares Unit Inland Water Systems, department of Flood Risk Management, P.O. Box 177, 2600 MH Delft, Netherlands
Correspondence to: E.J.C. Dupuits (e.j.c.dupuits@tudelft.nl)

Abstract. Flood defences can be designed as multiple lines of defence. This paper presents an approach for finding an optimal configuration for flood defence systems, based on an economic cost-benefit analysis with an arbitrary number of interdependent lines of defence. The proposed approach is based on a graph algorithm and is, thanks to some beneficial properties of the application, able to traverse large problems. A number of case studies were carried out to compare the optimal paths found by the proposed approach with the results of competing methods, and were found to generate (near) identical results. The work presented here makes cost-benefit analyses of complex flood defence systems with interdependent multiple lines of defence both easier and applicable to a broad range of flood defence systems with multiple lines of defence.

1 Introduction

Concerns regarding the safety of people and assets in flood prone areas has led to the construction of flood defence systems all around the world. Some flood prone areas, for example a large part of the Netherlands, face huge potential loss of life and economic value in case heavy flooding occurs. This has led to extensive research regarding estimating the flood risk of flood prone areas. Coupled to this quantification of the flood risk, is the question of ‘how safe’ a flood prone area should be. An often used approach to help answer this question is a cost-benefit analysis.

Economic optimization of flood defences, as applied in the Netherlands, is a cost-benefit analysis of the flood risk reduction balanced against the investment costs for flood defences. This type of cost-benefit analysis was already developed in the 1950’s by Van Dantzig (1956), and is still used and discussed to this day (Eijgenraam, 2006; Kind, 2014). The basic principle behind the economic optimization of flood defences is finding the minimum of the total costs; the total costs (TC) are the sum of risk (R_{cost}) and investment costs (I_{cost}), as shown in Eq. 1. The risk cost is defined as the probability of flooding times the loss incurred due to flooding. Generally speaking, a larger investment will lead to lower risk and this is where the economic optimization tries to find an optimal (lowest total cost) situation. This optimal situation can be related to an optimal investment scheme over time (e.g. see Eijgenraam (2006); Kind (2014)).

$$TC = R_{cost} + I_{cost} \quad (1)$$

Recent publications regarding economically optimal safety targets, for The Netherlands, can be found in Eijgenraam (2006); Brekelmans et al. (2012); Zwaneveld and Verweij (2014b, a). In Eijgenraam (2006), a set of equations was derived which describe the economically optimal safety target for a single flood defence system (dike ring). These equations describe the quantity of (repeated) investments, as well as the optimal time between these investments. The equations are analytically solvable, and the method resulted in a global minimum of the total costs for relatively simple homogeneous systems. However, because dike rings in the Netherlands often consist of different, non-homogeneous sections, the homogeneous case needed to be extended. In Brekelmans et al. (2012), a possible solution is given by modelling the problem as a mixed-integer nonlinear programming (MINLP) problem. Zwaneveld and Verweij (2014a) improved on this method by rewriting the problem as an integer linear programming (ILP) problem. Zwaneveld and Verweij (2014b) further extended their ILP problem to be applied to a case study with three lines of defence.

The notion of multiple lines of flood defences expresses that failure of one flood defence might alter the flood risk of other components. An example of a flood defence system with multiple lines of defence which can be found in practice is shown in Figure 1. This notion of multiple lines of flood defences has been, from a flood risk perspective, the main topic in a number of recent papers (e.g. Vorogushyn et al. (2010, 2012); Courage et al. (2013); De Bruijn et al. (2014)). All of these papers showed that viewing the flood defence system as a whole, with multiples lines of defence, resulted in different risk estimates than viewing the flood defences as separate, independent elements.

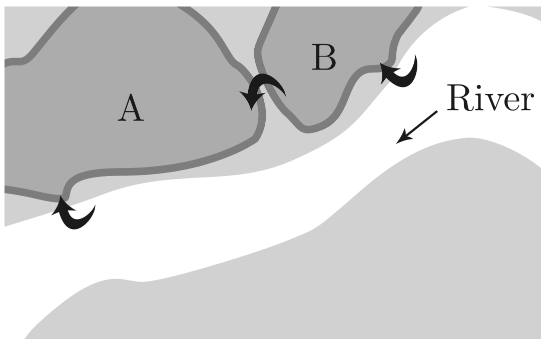

Figure 1. A hypothetical example of a system with multiple lines of defence. Area A has, aside from its own flood defences, an additional flood defence layer in the form of area B and its defences; this is because breaches (indicated by the curved arrows) at area B can impact the flood risk of area A.

As the flood risk changes, so will the associated (flood) risk costs. This in turn will affect the economic optimization. Therefore, it makes sense to explicitly integrate the effect of multiple lines of defence on the flood risk in the economic optimization routines. However, existing cost-benefit analyses tend to focus on flood defence systems with independent lines of defence (Kind, 2014), or are not (readily) generically applicable (e.g. Zwaneveld and Verweij (2014a)).Therefore, the aim of this paper is to find a generic, computationally efficient approach for finding the economically optimal configuration of a flood defence system with an arbitrary number of defence lines. These multiple lines of defence can be dependent on each other (i.e. influence each other's risk). We intend to accomplish this aim with the following approach:

– A generically applicable, flexible representation of the problem space to being able to use an arbitrary number of defences.

– Computational efficiency will be obtained by minimising the number of (time-consuming) risk computations in the algorithm until they are actually required. Risk computations, especially those with multiple lines of defence, can be demanding because of the involved hydrodynamic and flooding simulations (e.g. see Courage et al. (2013) or De Bruijn et al. (2014)).

– The reliability of finding economically optimal targets will be tested by comparing the results of the proposed method with a number of benchmark problems.

Section 2 starts with a description of the application and a description of the applied algorithm. Implementation details, focusing on the computational efficiency of the algorithm, are discussed in Section 3, as well as a list of potential future improvements to the algorithm. Next, the proposed approach is applied to some simplified case studies in Section 4, and is followed by a discussion (Section 5) regarding the relevance of the proposed approach. Finally, the results and experiences are concluded in Section 6.

2 An algorithm for flood defence systems with multiple lines of defence

2.1 Programmatic representation of the solution space

A common choice to present optimisation problems is to use graph algorithms (Cormen, 2009). An example of such a graph for a single flood defence is shown in Figure 2. The graph shows the possible investments over time for a single flood defence. In this graph the vertices (dots) are the possible heights the flood defence can have at a certain point in time. In order to go the next point in time, edges are drawn which connect a vertex to all the possible vertices in the next point of time.

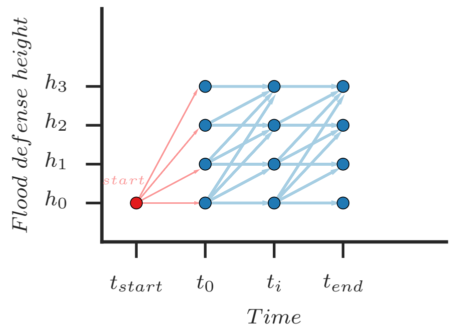

Figure 2. Graph where the vertices (dots) at each time step are connected via edges (arrows) to the next time step.

These points in time are not fixed; the amount and position can be altered to the needs of a particular problem. In practice, these points in time can be related to the (political) decision process of a particular problem: if the relevant flood defences are reviewed and (if necessary) reinforced every five years, it would make sense to have a graph that corresponds to these points in time.

Generally speaking, edges in a graph can be directed or undirected. However, steps backwards in time do not make sense for investment schemes. Therefore, only edges directed forward in time are used. The edge cost (or weight) of an edge is the total cost (risk cost plus investment cost) of moving between the connected vertices. Furthermore, it is assumed that flood defences will not be intentionally decreased to a lower level, which is why, for example, there are no edges running from h_1 to h_0. The starting point of the graph is denoted with start in Figure 2 at time t_{start} at a height equal to the current height (h_0).

In case of multiple lines of defence, our method takes into account that flood defences can be interdependent with regard to the risk of the area protected by the flood defences. This means that each combination of flood defence levels has to be considered relevant. These combinations can be found by computing the Cartesian product of the flood defence levels of all the involved defences. If some defences are independent, this is considered a special case of our approach. In that case, our method is still valid and applicable. However, in case of independence it can be worthwhile to make adaptations to the method for computational efficiency, by making use of the attractive features of independence. Such an approach will be discussed in Section 3.4.

Figure 3 shows an example of the Cartesian product for two flood defences where each flood defence has two possible heights. The graph in Figure 3 resembles the graph in Figure 2 for a single flood defence. Similar to Figure 2, edges in Figure 3 are only drawn to vertices containing sets of heights equal or greater than the set of heights in the vertex at the origin of the edge. However, because Figure 3 has two defences instead of one, the outgoing edges are slightly different when compared to Figure 2. For example, the height combination h_{A0}, h_{B1} is never connected to h_{A1}, h_{B0} (since that would correspond to a reduction in height for defence B).

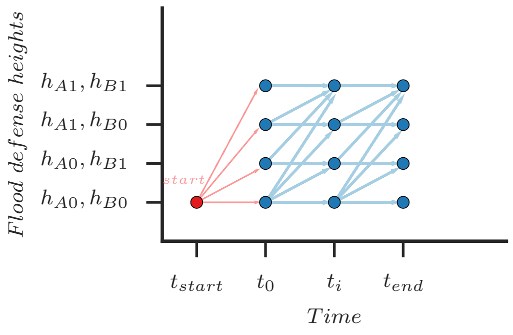

Figure 3. Graph with vertices (dots) and edges (arrows) for two defences (A and B). Each defence has two possible heights.

2.2 Implementation of a graph algorithm

In general terms, a graph algorithm will iterate over vertices in a graph in an effort to find the path with the lowest costs between a given start and end vertex. However, in the graphs of Figure 2 and Figure 3 t_{end} contains a number of possible end points, which means that the algorithm will need to find as many optimal paths as there are end points in the graph. In order to only have to run the algorithm once, a stop vertex is added; the graph of Figure 3 with an additional stop vertex is shown in Figure 4. The edges running towards this stop vertex are all given a weight of zero. Now, the algorithm only has to find a single optimal path between t_{start} and t_{stop}. Why this is an efficient contribution is illustrated in Section 2.4.

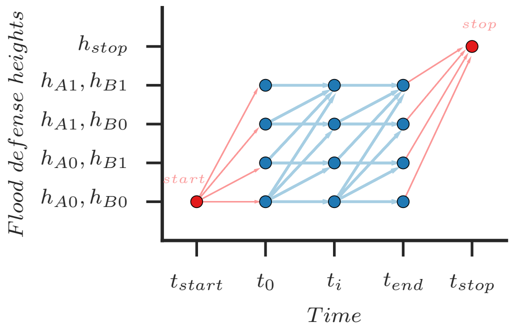

Figure 4. The graph of Figure 3 with an additional stop vertex.

The graph as shown in Figure 4 is a graph with directed non-negative edges. For this kind of graph, a number of algorithms can be used to find the shortest (optimal) path in a graph, for example: the Dijkstra algorithm (Dijkstra, 1959), the A* algorithm (Hart et al., 1968), and the Uniform Cost Search (e.g. (Verwer et al., 1989)). All three can be considered to be part of the family of best-first search algorithms, where both the Dijkstra and the Uniform Cost Search (UCS) algorithms can be seen as a special case of the A* algorithm (i.e. A* without the use of a heuristic graph function). For our application, we did not come up with a heuristic function, which reduces the choice of a graph algorithm to either Dijkstra or UCS.

Typically, the best-first search algorithms are implemented with a min-priority queue. A min-priority queue holds a sorted list of vertices, where the sorting is based on the cost of reaching that vertex from the start vertex; the vertex with the lowest cost is at the top of the queue. This list of vertices in the priority queue constitutes of, depending on the implementation, either all vertices in the graph (Dijkstra as implemented in Cormen (2009)), or only the vertices already visited by the graph algorithm (UCS). A comparison between the two algorithms can be found in Felner (2011), where the priority queue as implemented by UCS was found to be faster and using less memory. We consider this a relevant advantage, as the number of vertices can be large when using the Cartesian product of flood defence levels (Section 2.1). For this reason, we chose to implement the UCS algorithm.

Applying the UCS algorithm to a graph such as shown in Figure 4 begins with creating a priority queue which only contains the start vertex. After this initialization, the iteration process is started. Each iteration starts with taking out the vertex with the lowest cost known thus far from the priority queue (which is the top entry in the queue). Taking out means the optimal route (lowest cost) from the start vertex to this vertex now known. The vertex that has just been taken out of the priority queue is then queried in the graph to find all the connecting vertices in the next time step. Each connecting vertex is added to the priority queue if the vertex is not already in the queue. If the vertex already exists in the queue, the weight is only updated if the newly proposed cost is lower than the known cost so-far. Iteration continues until at the start of an iteration the stop vertex is the top entry in the priority queue. An actual example of the application of this algorithm will be elaborated in Section 2.3.

2.3 Example application of the algorithm in an economic optimization

This section shows a simple example of an economic optimization for a single flood defence. While this example uses a single flood defence for simplicity, the same principles apply for multiple flood defences. Regarding the investment and risk costs, if a vertex at t_1 is connected to another vertex with a larger height at t_2, it is assumed that the actual heightening occurs at t_1. This leads to a slightly different graph than the conceptual implementation shown in Section 2.1 & 2.2, and is emphasized by drawing the edges of the figures in this example (i.e. Figures 6, 7 & 8) in a way which is visually more consistent with the timing of the investment decision.

The result of the first two iterations is shown in Figure 6, where the start vertex is labelled with the number 1. In this example, the start vertex is associated with a height of 4.25 meter and starts at $t = 0$, identical to vertex 2. Because the path to vertex 2 is the only possible path, vertex 2 is the only addition to the priority queue. In the next iteration, vertex 2 is taken out of the priority queue as it is the vertex with the lowest total cost. The total cost to reach vertex 2 is 0, because there was no heightening (height remains at 4.25 meter) and no time expired ($t_{start} = 0$); the risk cost is zero because time needs to expire for risk to occur. From vertex 2, the number of possible next steps and associated total costs are computed and added to the priority queue, as illustrated in Figure 6. Note that the total costs to reach for example vertex 22 consists of the total cost from t_{start} to $t = 100$, not just the cost from $t = 50$ to $t = 100$.

The algorithm will continue for a while, until the situation of Figure 7 is reached where vertex 24 is taken out of the priority queue. The new found total costs for vertex 29, 30 and 31 are not lower than the total cost for vertex 25, which means the algorithm takes a step back and continues from vertex 25. From vertex 25, vertex 30 and 31 are re-evaluated in Figure 8, where only vertex 30 results in lower costs than the existing options. This means that only vertex 30 is updated with the new, lower, total cost in the priority queue. Additionally, if hypothetically vertex 31 is the vertex with the lowest cost, the optimal path would revert back to using vertex 24 instead of vertex 25 (because the path from vertex 25 to 31 has higher costs than the path from vertex 24 to 31).

2.4 Global optimal solution

The UCS algorithm finds the shortest path in a graph, see for example Felner (2011) for a recent elaboration regarding the ‘correctness’ of the UCS algorithm or Gelperin (1977) for a proof regarding A* (UCS can be considered a special case of A*). What remains is whether the additional stop vertex of Section 2.2 leads to a heuristic solution or still to the optimal path. However, assuming that the optimal path towards the stop vertex is found, whichever vertex at t_{end} is part of that optimal path has to be the optimal choice. Otherwise, the path towards the end vertex is not optimal, which contradicts the earlier mentioned proofs. In order to test the performance of the proposed method, Section 4 will compare numeric results from our proposed method to other approaches. These approaches are known to give global optimal results, or at least very close to the global optimum.

Nat. Hazards Earth Syst. Sci. Discuss., doi:10.5194/nhess-2017-12, 2017
Manuscript under review for journal Nat. Hazards Earth Syst. Sci.
Published: 2 February 2017

2.5 Overview of the approach

A general overview of the approach discussed in the previous sections is shown in Figure 5. The method is composed of four steps: input, pre-processing, processing and post-processing. Of these steps, user interaction is only required at the input step. The rest of the steps run automatically. Specifically, the user needs to supply vectors of flood defence levels per flood defence, a time vector and a function which can calculate the cost of an edge in the graph. In the following steps, the graph is created (pre-process), the optimal path is found (process), and the optimal path is shown (post-process).

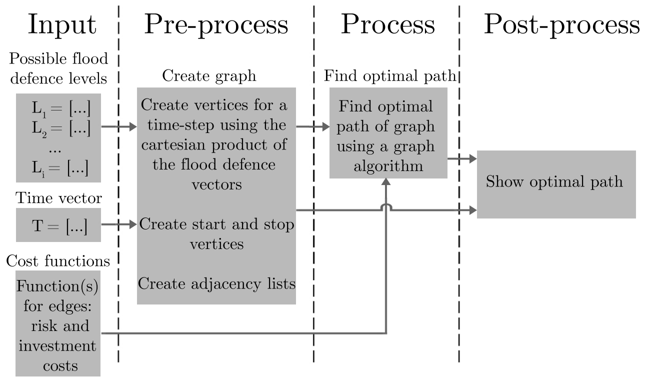
Input
Pre-process
Process
Post-process
Possible flood defence levels
L₁ = [...]
L₂ = [...]
...
Lᵢ = [...]
Time vector
T = [...]
Cost functions
Function(s) for edges: risk and investment costs
Create graph
Create vertices for a time-step using the cartesian product of the flood defence vectors
Create start and stop vertices
Create adjacency lists
Find optimal path
Find optimal path of graph using a graph algorithm
Show optimal path

Figure 5. Overview of the approach using a graph and graph algorithm. In our approach, the graph algorithm is the UCS algorithm. The input column is the only part what the user should provide, the other steps run automatically.

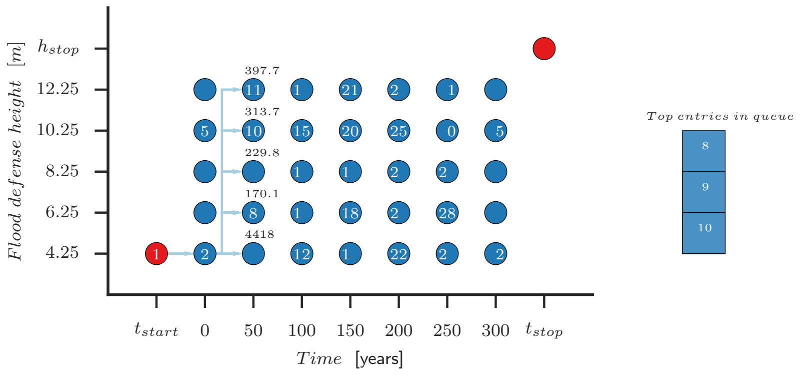

Figure 6. The first two iterations of the graph algorithm with a min-priority queue. The vertices available in the priority queue are those which have total costs above their respective vertices, and the first three entries are shown in the column on the right. Note that because the choice was made to connect the start vertex to vertex 2, vertices 3 - 6 will not be visited.

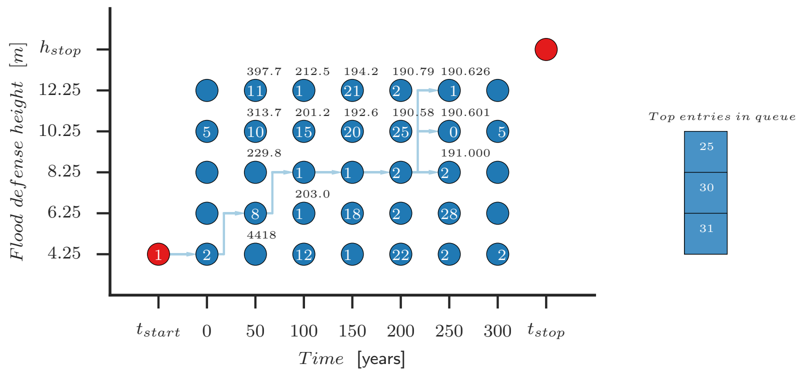

Figure 7. After six iterations with the graph algorithm, vertices 29, 30 and 31 are added to the priority queue. However, in this case the algorithm makes a step back in time, because vertex 25 is the item with the lowest total cost in the priority queue.

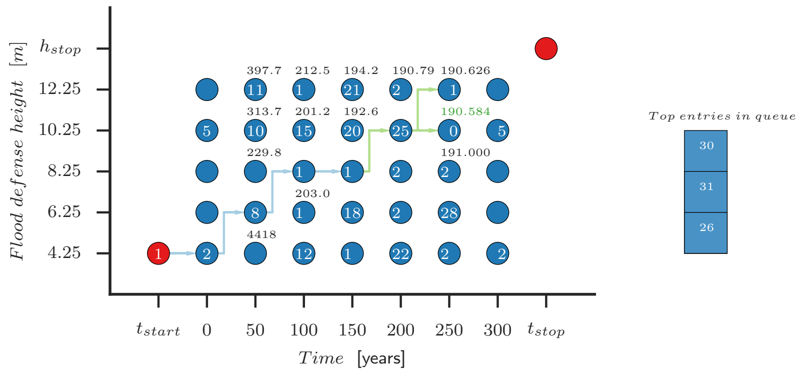

Figure 8. During iteration seven, the old path is abandoned, and an alternative path with vertex 25 instead of vertex 24 is taken. Vertices 30 and 31 are already in the priority queue (calculated from vertex 24), and will only get updated if the total costs from vertex 25 are lower (which is the case for vertex 30).

3 Efficiency improvements

The economic optimization of multiple lines of defence can potentially lead to large numbers of vertices and even large numbers of edges. For example, for eight lines of defences with six possible heights the number of vertices per time step is approximately 1.68 million (6^8), while the number of edges per time step is even larger at approximately 35 billion. For large problems such as these, storing all the possible vertices and edges would lead to huge data structures. Fortunately, the graph representation as presented in Section 2.1 has repetitive properties which can be used reduce the size of the data structures.

3.1 Repetitiveness in lists of vertices

Even though the graphs of Section 2.1 can be classified as sparse graphs (number of edges is much smaller than the number of vertices squared, Cormen (2009)), the number of edges is still much larger than the number of vertices. Therefore, we first focused on data structures related to the edges of a graph. For sparse graphs, these are the adjacency lists: a group of vertices connected via edges stemming from a source vertex in a previous time step. In these adjacency lists, repetitiveness can be found with respect to two aspects.

The first repetitive aspect is the similarity of adjacency lists for the same combination of flood defence levels at different time steps (except for the adjacency lists at t_{end}). In Figure 6, vertices with the same combination of flood defence levels at different time steps are for example vertices 2, 7 and 12. The adjacency lists for these vertices are shown in Figure 8, where it is apparent that the adjacency list for the next time step can be found by adding an offset to the elements of the adjacency list of the current time step. For example, the adjacency list of vertex 2 can be turned into the adjacency list of vertex 7 by adding the total number of combinations in each time step (which is five in Figure 9)

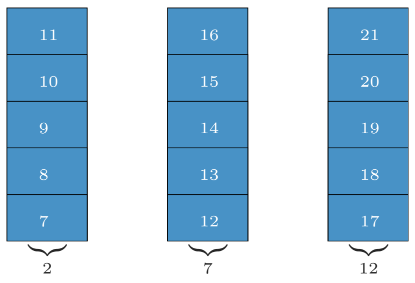

Figure 9. The adjacency lists for vertices 7 and 12 of Figure 6 can be obtained by adding an offset to the adjacency list of vertex 2.

The second repetitive aspect is for adjacency lists between vertices in the same time step. Because the lowest vertex in each time step (e.g. vertices 2, 7, 12, 17, 22 and 27 in Figure 6) has outgoing edges running to each and every vertex in the next time step, higher vertices (e.g. in Figure 6, vertex 8 is ‘higher’ than vertex 7) contain a subset of the adjacency list of the lowest vertex. In other words, outgoing edge lists in a single time step can be generated dynamically by shrinking the adjacency list of the lowest vertex in a time step. This is shown in Figure 10.

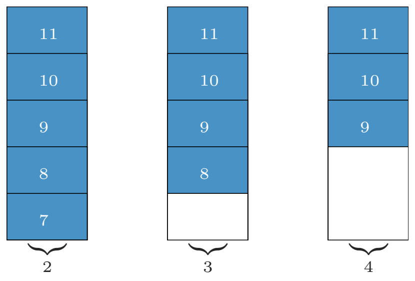

Figure 10. The adjacency lists for vertices 3 and 4 of Figure 6 are reduced sets of the adjacency list for vertex 2.

The combination of these two repetitive characteristics results in that only a single adjacency list needs to be stored in memory (i.e. the adjacency list of the lowest vertex in the first time step). This single adjacency list can be adapted to most vertices in the graph by means of offsetting and shrinking the stored adjacency list. Notable exceptions are the adjacency lists for the vertices at t_{end}, but the adjacency lists for these vertices are already known and only contain the stop vertex.

3.2 Conditionally removing edge connections

Besides reducing the size of the data structures associated with a graph, the adjacency list associated with a vertex can also be reduced under certain conditions. Typically, the time between improvements in flood defences is large (in the order of 50 years), due to either high (fixed and variable) costs associated with investments in flood defences, or long planning periods (Zwaneveld and Verweij, 2014b). Therefore, if one or multiple flood defences have been strengthened recently, the adjacency list can be reduced to only contain vertices that keep the recently strengthened flood defence(s) at the current level(s). However, this so called ‘waiting time’ before new investments are considered has to be chosen with care, because the waiting time should not influence the optimal time between investments. Nevertheless, a correctly chosen waiting time can greatly improve the run time of the algorithm, because of the significant reduction in number of edges that need to be evaluated. This reduction is shown in Figure 11, where the total number of visited vertices is plotted as a function of the ‘waiting time’; the underlying problem that is solved by the algorithm is the same problem as shown in Section 4.3.

3.3 Reducing the number of risk calculations

In the overview of Figure 5 it is implied that the risk calculations belonging to an edge are only carried out when that edge is visited by the graph algorithm. Provided that a graph algorithm does not visit all vertices, delaying (risk) calculations belonging to an edge until that edge is visited leads to less risk calculations than the total number of edges. In contrast, if (risk) calculations are done before a graph algorithm is initialized, all edges need to be calculated.

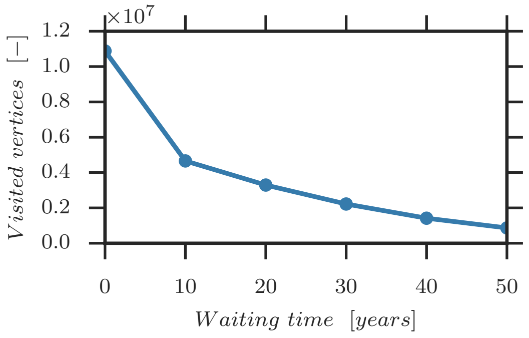

Figure 11. Reduction in the total number of visited vertices as a function of the waiting time for the example of Section 4.3.

As an example, Figure 12 shows the number of times each vertex is visited is in the example of Section 2.2. The majority of the vertices in Figure 12 get visited once, but a significant proportion is never visited by the graph algorithm; these vertices have a zero above their indices. A small proportion of the vertices, specifically vertices 30 and 31, are visited twice; the reason for this re-visiting can be seen in Figure 8. To avoid completely re-doing risk calculations upon a revisit, parts of a calculation can be cached in order to reduce the computational penalty incurred by revisiting vertices.

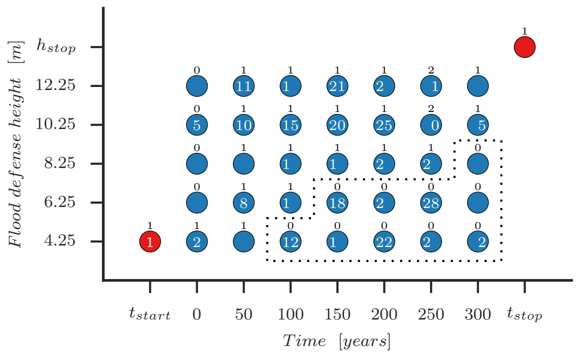

Figure 12. Number of times each vertex is visited by the algorithm for the example in Section 2.2. The dotted area emphasizes that a part of the graph is never visited, while vertex 30 and 31 get visited twice.

5

3.4 Potential improvements and special cases

Further improvements can be made both to the graph implementation and to the implementation of the algorithm. The algorithm was implemented as a single process; a performance improvement might be found by utilizing parallel programming. The first place where parallel programming could be beneficial is the loop over an adjacency list. This is because the risk cost calculations (i.e. edge weights) are potentially expensive to compute, therefore parallelizing the loop over an adjacency list can lead to significant performance improvements.

Furthermore, regarding the graph implementation, a special case is a flood defence system which has independent flood defences. Section 2.1 uses the Cartesian product of flood defence options, which has the underlying notion that all flood defence lines are interdependent. If some flood defences are independent (i.e. the defences protect different, independent areas), this leads to an inefficient graph. The independency of flood defences can be used in an adapted graph representation in order to get an efficient graph. While we did not implement this, a way to solve this for the system in Figure 13 is shown in Figure 14, which uses 'subgraphs' to reduce the number of combinations.

These subgraphs are small graphs which only contain the number of strengthening options for a single defence for a single time period (e.g. in Figure 14, from t_{i-1} to t_i). Additionally, the subgraphs take into account what the level is of the influential defences (e.g. in Figure 14, the front defence B is the only influential defence for the rear defences). The use of subgraphs leads to a smaller number of combinations, as the Cartesian product would have resulted in a total number of combinations (or vertices per time step) of 5120 ($5*4^5$). With subgraphs, the number of combinations is reduced to 100 ($5*5*4$).

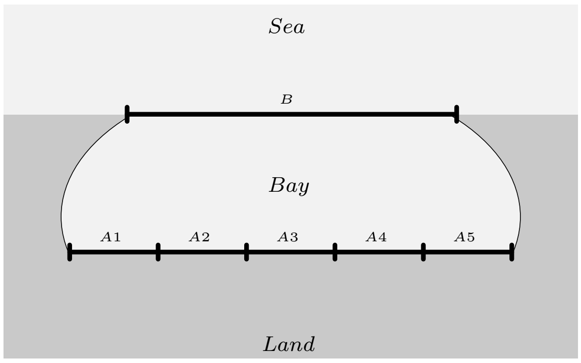

Figure 13. A top view of a system with a front line defence (B, five possible safety levels) and five rear defences ($A1 - A5$, each has four possible safety levels). The front defence influences the rear defences, but the rear defences do not influence each other.

4 Results for simplified flood defence systems

In order to test the performance of the proposed algorithm versus some existing approaches, three cases are investigated. For simplicity, these three cases will be based upon a common set of investment and risk relations, as well as a common set of input values. The values and symbols used in this section are largely copied from (Eijgenraam, 2006, page 34) and reproduced in Table 1, with only minimal changes.

The common set of investment (I) and risk (R) relations are similar to the relations used with the data of Table 1 in Eijgenraam (2006). The sum of the investment and risk is the total cost, which needs to be minimized in order to get economically

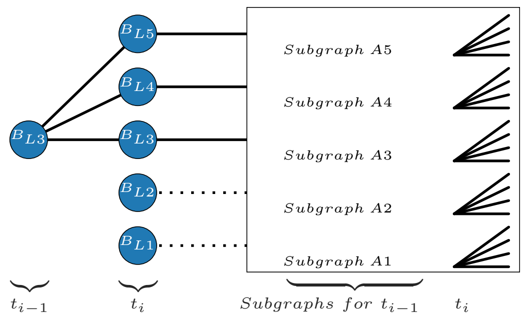

Figure 14. Part of the graph belonging to the system of Figure 13 for the period t_{i-1} to t_i. Because the rear defences do not influence each other, subgraphs are used for the rear defences.

Table 1. Variables and values taken from Eijgenraam (2006) for the risk and investment equations in this section. NLG refers to the currency used in the Netherlands prior to the euro.

Name	Unit	Symbol	Value
Height above mean sea level, base	cm	H_0	425
Annual exceedance probability belonging to H_0	-	P_0	0.0038
Parameter exponential distribution water level	1/cm	α	0.026
Increase water level	cm/year	η	1
Damage by flooding in 1953	10^6 NLG	V_0	20000
Economic growth	1/year	γ	0.02
Rate of interest (real)	1/year	δ	0.04
Variable costs of investment	10^6 NLG/cm	C_v	0.42
Fixed costs of investment	10^6 NLG	C_f	61.7
Heightening of the flood defence at time t	cm	u_t	-
Height of the flood defence at time t	cm	H_t	-

optimal safety targets:

$$\text{Total Cost} = \int_0^\infty R(t)dt + \sum_{t=0}^{\infty} I(t) \tag{2}$$

$$R(t) = P_0 e^{-\alpha(H_t - H_0 - \eta t)} V_0 e^{\gamma t} e^{-\delta t} \tag{3}$$

$$I(t) = (C_v u_t + C_f \text{sign}(u_t)) e^{-\delta t} \tag{4}$$

where sign(u_t) is used to prevent fixed costs in case there is no heightening u_t. This sign(u_t) function returns zero if the heightening u_t is equal to zero, and returns one when the heightening u_t is larger than zero.

4.1 Single flood defence

For a single flood defence, with the values of Table 1, an analytical solution can be found in (Eijgenraam, 2006, page 35). This solution consists out of an initial dike height increase coupled with a periodical, constant dike increase over an infinite time horizon. The initial increase was found to be 236 centimetres, with a periodical increase of 129 centimetres every 73 years.

Because the approach introduced in this paper is a numerical approach, a finite time period has to be used instead of an infinite time horizon. Similar to Zwaneveld and Verweij (2014b), we choose to use a time period of 300 years with, for this application, steps of one year. The possible heights were discretized using a range starting from 425 to 1225 cm, with steps of one centimetre. Note that these step sizes (and dimensions) were deliberately chosen to be on par with the accuracy level of the analytical solution. In practice, these step sizes would probably be too detailed for the practical attainable accuracy in flood defence construction (see also Zwaneveld and Verweij (2014b)).

A comparison of the results found using the algorithm and the analytical solution is shown in Figure 15. The algorithm found an initial increase of 235 cm, with three additional increases in height at 73 years apart. These three were found to be 129 cm, 130 cm, and 132 cm. The last increase is different from the analytical solution, and can be attributed to being close to the end of the time horizon. A finite time horizon implies that there is no risk beyond the time horizon, which explains why there is no investment found by the algorithm in year 292. To compensate for the lack of an investment in year 292, the investment in year 219 is slightly larger. This explanation is supported by results with a time horizon of 400 years, where the heightening in year 219 changes to an expected 129 centimetres. These deviations near to the time horizon underline that if a certain point in time is considered relevant, the used time horizon should stretch significantly beyond that point in time. However, this is a general problem with all numerical methods, because of the required finite time horizon, and not a specific issue related to the approach proposed in this paper. Furthermore, in practice this problem can be circumvented by setting the time horizon used in the algorithm to sufficiently exceed the practically required time horizon.

4.2 Two independent lines of defence

In the next example two defences are investigated using the graph algorithm, both with the same characteristics as the single flood defence in the previous section. However, the step size for the heights is increased to 20 cm in order to test the response of

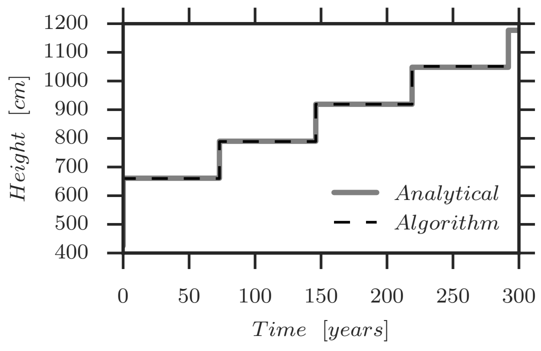

Figure 15. The investment scheme found using the algorithm is almost identical to the analytical solution.

the algorithm to larger step sizes. Expected is that, despite the less detailed step size, the investment scheme for both defences should be identical to each other and close to the analytical solution provided in the previous section.

Indeed, the results of the algorithm, illustrated in Figure 16, show that both defences are initially increased with 240 cm, while in both year 75 and 143 the defences are increased with 120 cm, and finally in year 212 with 140 cm. Clearly, the larger step size in height leads to larger differences when compared to the analytical solution. Nevertheless, any overshoot/undershoot of the height is 'repaired' in the duration between investments, keeping the solution of the optimal path stable and close to the analytical solution.

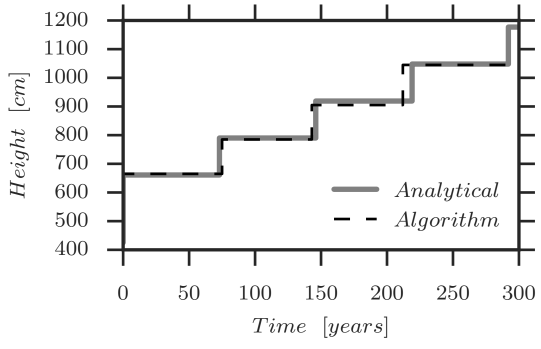

Figure 16. The two (independent) lines of defence have an identical solution with the approach proposed in this paper, and are (even with the usage of larger step sizes) good approximations of the known analytical solution.

4.3 Two dependent lines of defence

The final case is similar to the case with two independent lines of defence, however the second defence is now dependent on the performance of the first defence. This dependency is illustrated in Figure 16, and is a simplified version of the case discussed in Dupuits et al. (2016).

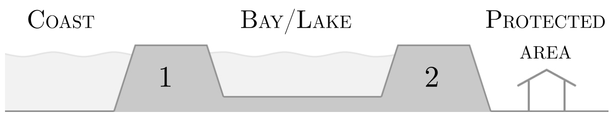

Figure 17. A coastal system with two lines of defence. This figure is an adaption from an illustration found in Dupuits et al. (2016).

The dependency between the defences in Figure 17 is implemented by adapting the risk equation of Eq. 3 as follows:

$$R(t) = \left(P_1 P_{2|1} + (1 - P_1) P_{2|\bar{1}}\right) V_0 e^{\gamma t} e^{-\delta t} \tag{5}$$

$$P_i = P_0 e^{-\alpha_i (H_{i,t} - H_0 - \eta t)} \tag{6}$$

where P_i is a generic formulation used for the failure probabilities P_1, $P_{2|1}$ and $P_{2|\bar{1}}$. The probabilities and are the failure probabilities of the second defence, dependent on the failure ($P_{2|1}$) or non-failure ($P_{2|\bar{1}}$) of the first defence, where the failure probability of the first defence is denoted by P_1. Similarly, the investment equation in Eq. 4 is expanded to include different costs for the two lines of defence:

$$I(t) = (C_{v1} u_1 + C_f \text{sign}(u_1) C_{v2} u_2 + C_f \text{sign}(u_2)) e^{-\delta t} \tag{7}$$

The new variables used in Eqs. 5, 6 & 7 are listed in Table 2. The solution found with the approach proposed in this paper was checked with the method proposed in Zwaneveld and Verweij (2014a); the outcomes of both methods were found to be identical and are shown in Figure 18.

Table 2. Additional variables used in Eqs. 5, 6 & 7, complementary to Table 1.

Name	Unit	Symbol	Value
Annual exceedance probability belonging to H_o	-	P_0	0.01
Exponential parameter for defence 1	1/cm	α_1	0.026
Exponential parameter for defence 2 for $P_{2\|\bar{1}}$	1/cm	$\alpha_{1\|\bar{2}}$	0.052
Exponential parameter for defence 2 for $P_{2\|1}$	1/cm	$\alpha_{1\|2}$	0.026
Variable costs of investment for defence 1	10^6 NLG/cm	C_{v1}	0.21
Variable costs of investment for defence 2	10^6 NLG/cm	C_{v2}	0.42

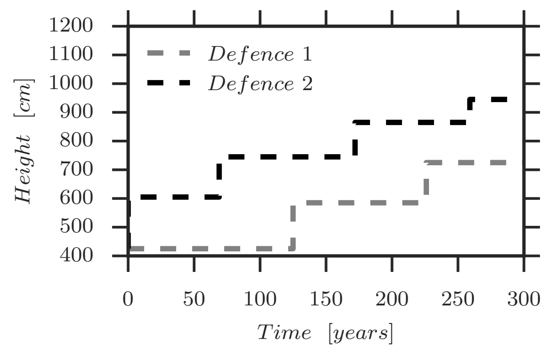

Figure 18. Optimal investment schemes for the case with two dependent lines of defence.

5 Discussion

The proposed approach in this paper is based on a best-first graph algorithm, which is relatively easy to implement in most general or scientific programming languages. In our opinion, this is a significant advantage over linear programming type algorithms, especially for those who are not familiar with the implementations of linear programming as proposed by Kind (2014); Zwaneveld and Verweij (2014a). The application area is roughly similar as for Kind (2014); Zwaneveld and Verweij (2014a), with the notable difference that our approach focuses on flood defence systems with multiple (interdependent) lines of defence, while their applications seem to be more focused towards systems with solely (Kind, 2014) or a majority (Zwaneveld and Verweij, 2014a) of independent elements. In Section 3.4, a possible improvement is mentioned which can make our approach more applicable towards systems with independent elements.

An inherent problem of working with flood defence systems where most, if not all, elements are dependent on each other, is that the number of system combinations grows exponentially with the number of lines of defence. The sheer number of combinations means that the total number of lines of defence should probably be kept below ten. This is nothing more than a rule of thumb based on our experience running the best-first graph algorithm on a consumer laptop. The true maximum depends on a number of factors: the number of height options per defence, the performance of the particular implementation of the proposed approach, the computational cost of the associated risk functions and the computational power of the used computer.

Even though all examples in this research make use of flood defence heights, this was only done to illustrate the approach. Other measures besides flood defences can be incorporated as well. For example, if a retention area is considered (as illustrated in Figure 19), a list with possible sizes of the retention area could also be used in the approach of Figure 5; as long as a measure has a number of options or levels in increasing order that can be quantified and monetized, it can be included in the approach. This makes the actual application range much wider than flood defence systems with only height-dependent flood defences such as levees or (storm surge) barriers.

The proposed approach works best if the type of flood defence for each line is known and singular. In the case that a number of different flood defence types are considered at the same line of defence, it would be better to do an optimization run per system type configuration. An example of this would be the choice between a closure dam or a storm surge barrier at the same location. In this case, the algorithm should be run twice, first with a closure dam and then with a storm surge barrier. This should result in two optimal configurations (one with a closure dam, the other with a storm surge barrier), which can then be compared using some metric, for example their benefit-cost ratios.

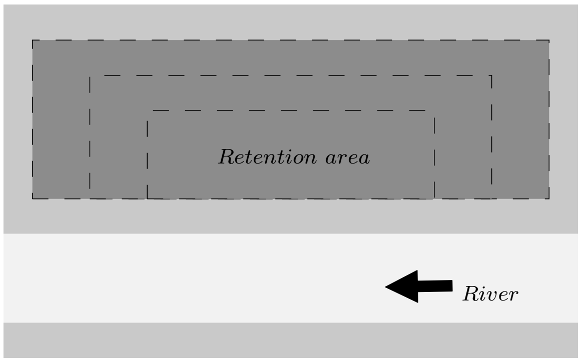

Figure 19. A retention area can also be optimized using the approach proposed in this paper. In this example, the surface area of the retention basin is used instead of the height of a flood defence.

6 Conclusions

This paper presented an easy to use, flexible approach for finding the economically optimal investment scheme for flood defence systems with multiple lines of defence. We consider this approach to be easy to implement because it uses a best-first graph algorithm, which itself is a simple algorithm. It is flexible because the graph representation shown in this paper can trivially accommodate an arbitrary number of lines of defence. Furthermore, the approach proposed does not need to pre-calculate risk estimates prior to running the best-first graph algorithm (which linear programming solutions do). Especially in cases where the risk estimates impose a significant computational cost, this can be an advantageous property.

The proposed approach utilizes the repetitive properties of the graphs in order to efficiently compute problems containing roughly up to ten interdependent lines of defence on a consumer laptop. In case independent flood defences are present in a system, the proposed approach can easily be adapted to a more efficient method which makes use of the attractive properties of independence. To that end, an improvement in the generation of the graphs has been proposed.

Assuming that the graph and combinations of flood defences are portrayed correctly, the best-first graph algorithm has been proven in literature to return the shortest (or optimal) path in a graph. To address this, the method was tested on a number of benchmark problems with known solutions. The tests show that indeed the optimal path is found with the approach proposed in this paper, which indicates that both the generated graphs and the algorithm work as expected.

Author contributions.

Competing interests.

Disclaimer.

Acknowledgements. We are grateful for the financial support of the Dutch Technology Foundation STW, which is part of the Netherlands Organization for Scientific Research, and which is partly funded by the Ministry of Economic Affairs.

References

Brekelmans, R., Den Hertog, D., Roos, K., and Eijgenraam, C.: Safe Dike Heights at Minimal Costs: The Nonhomogeneous Case, Operations Research, 60, 1342–1355, doi:10.1287/opre.1110.1028, 2012.

Cormen, T. H.: Introduction to Algorithms, 3rd Edition:, MIT Press, 2009.

Courage, W., Vrouwenvelder, T., van Mierlo, T., and Schweckendiek, T.: System behaviour in flood risk calculations, Georisk: Assessment and Management of Risk for Engineered Systems and Geohazards, 7, 62–76, doi:10.1080/17499518.2013.790732, http://dx.doi.org/10.1080/17499518.2013.790732, 2013.

De Bruijn, K. M., Diermanse, F. L. M., and Beckers, J. V. L.: An advanced method for flood risk analysis in river deltas, applied to societal flood fatality risks in the Netherlands, Natural Hazards and Earth System Sciences Discussions, 2, 1637–1670, doi:10.5194/nhessd-2-1637-2014, 2014.

Dijkstra, E. W.: A Note on Two Probles in Connexion with Graphs, Numerische Mathematik, 1, 269–271, doi:10.1007/BF01386390, 1959.

Dupuits, E., Schweckendiek, T., and Kok, M.: [In preparation] Economic Optimization of Coastal Flood Defence Systems, Reliability Engineering & System Safety, 2016.

Eijgenraam, C.: Optimal safety standards for dike-ring areas, Tech. Rep. 62, CPB, The Hague, 2006.

Felner, A.: Position paper: Dijkstra's algorithm versus uniform cost search or a case against dijkstra's algorithm, in: Fourth Annual Symposium on Combinatorial Search, 2011.

Gelperin, D.: On the optimality of A*, Artificial Intelligence, 8, 69–76, 1977.

Hart, P., Nilsson, N., and Raphael, B.: A Formal Basis for the Heuristic Determination of Minimum Cost Paths, IEEE Transactions on Systems Science and Cybernetics, 4, 100–107, doi:10.1109/TSSC.1968.300136, 1968.

Kind, J.: Economically efficient flood protection standards for the Netherlands, Journal of Flood Risk Management, 7, 103–117, doi:10.1111/jfr3.12026, http://doi.wiley.com/10.1111/jfr3.12026, 2014.

Van Dantzig, D.: Economic Decision Problems for Flood Prevention, Econometrica, 24, 276–287, 1956.

Verwer, B. J. H., Verbeek, P. W., and Dekker, S. T.: An Efficient Uniform Cost Algorithm Applied to Distance Transforms, IEEE Transactions on Pattern Analysis and Machine Intelligence, 11, 425–429, 1989.

Vorogushyn, S., Merz, B., Lindenschmidt, K.-E., and Apel, H.: A new methodology for flood hazard assessment considering dike breaches, Water Resources Research, 46, doi:10.1029/2009WR008475, http://dx.doi.org/10.1029/2009WR008475, 2010.

Vorogushyn, S., Lindenschmidt, K.-E., Kreibich, H., Apel, H., and Merz, B.: Analysis of a detention basin impact on dike failure probabilities and flood risk for a channel-dike-floodplain system along the river Elbe, Germany, Journal of Hydrology, 436-437, 120–131, doi:10.1016/j.jhydrol.2012.03.006, http://www.sciencedirect.com/science/article/pii/S0022169412001928, 2012.

Zwaneveld, P. and Verweij, G.: Economisch optimale waterveiligheid in het IJsselmeergebied, Tech. Rep. 10, CPB, The Hague, 2014a.

Zwaneveld, P. J. and Verweij, G.: Safe Dike Heights at Minimal Costs, Tech. rep., CPB, The Hague, 2014b.