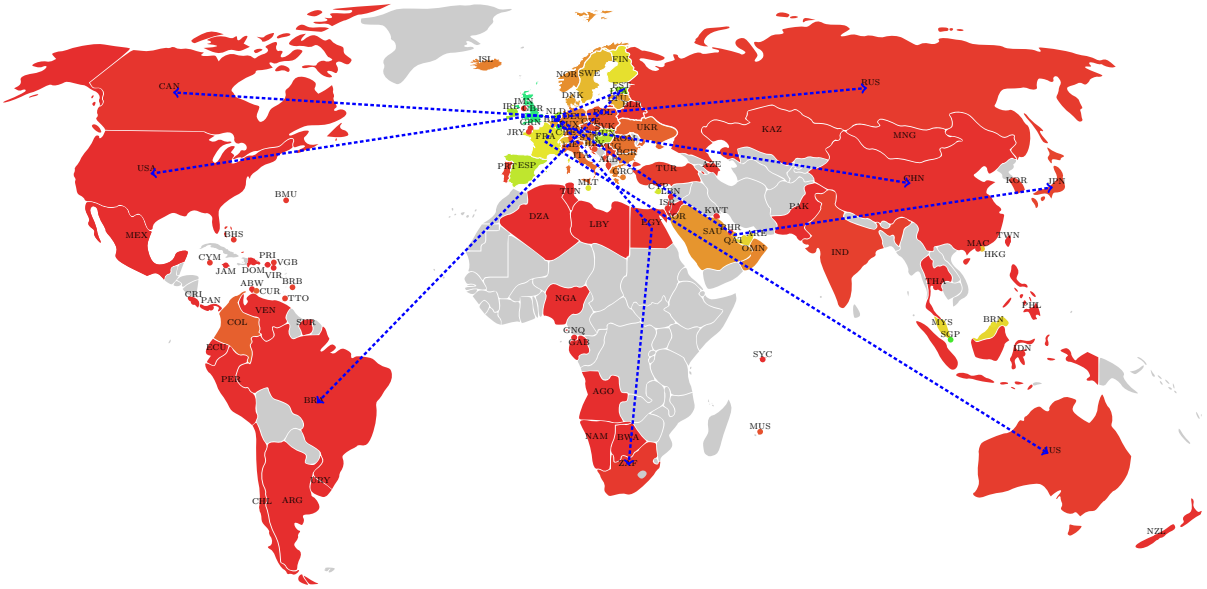
Algorithms for the Network Analysis of Bilateral Tax Treaties

Sven Polak

23 December 2014

Master Thesis

Daily Supervisor (VU / CWI): prof. dr. Guido Schäfer
First Examiner (UvA): prof. dr. Alexander Schrijver
Second Examiner (UvA): dr. Jan Brandts
Internship Supervisor (CPB): drs. Maarten van 't Riet



KdV Instituut voor Wiskunde
Faculteit der Natuurwetenschappen, Wiskunde en Informatica
Universiteit van Amsterdam

Abstract

In this thesis we conduct a network analysis of bilateral tax treaties. We are given tax data of 108 countries. Companies often send money from country to country via indirect routes, because then the tax that must be paid might be lower. In the thesis we will study the most important of these 'tax' routes. Questions that we will answer are, for example:

1. Which countries are the most important 'conduit' countries in the network?
2. How can a country maximize the amount of money that companies send through this country?

The thesis is mainly theoretical: the focus is on the mathematics and the algorithms used for the network analysis. At the end of each chapter we apply the algorithms to the CPB-network of 108 countries. The thesis is a collaboration between the CWI (Centrum voor Wiskunde en Informatica) and the CPB (Netherlands Bureau for Economic Policy Analysis).

Data

Title: Algorithms for the Network Analysis of Bilateral Tax Treaties
Author: Sven Polak, sven.polak@student.uva.nl, 6074251
Daily Supervisor (VU / CWI): prof. dr. Guido Schäfer
First Examiner (UvA): prof. dr. Alexander Schrijver
Second Examiner (UvA): dr. Jan Brandts
Internship Supervisor (CPB): drs. Maarten van 't Riet
Hand-in date: December 23, 2014

Korteweg de Vries Instituut voor Wiskunde
Universiteit van Amsterdam
Science Park 904, 1098 XH Amsterdam
`http://www.science.uva.nl/math`

Acknowledgements

I thank prof. Guido Schäfer for his guidance and his support. Every week we had very constructive meetings, on the theory side of the thesis as well as the practical side. This thesis would not have been possible without his superb supervision. Thank you!

Furthermore, I would like to thank Maarten van 't Riet from CPB for his guidance, all the tax-data, his help with the practical side of this thesis and last but not least for the weekly table tennis sessions we had after lunch. It was fun.

At last, I would like to thank prof. Alexander Schrijver for being the first examiner, and dr. Jan Brandts for being the second examiner for the University of Amsterdam.

Contents

Introduction: A network of countries

Is the Netherlands a tax haven for multinational enterprises? Articles in the press give the impression that this is indeed the case. '*The Netherlands is a tax haven for many multinationals*' [Waa11], '*The Netherlands is an attractive tax country*' [NOS14], '*Dutch masters of tax avoidance*' [GM11], are some examples of headers that may point in this direction.

To investigate whether the Netherlands is indeed a tax haven, the CPB (Netherlands Bureau for Economic Policy Analysis) conducted a research (see [RL14] and in particular [RL13]). Companies mainly use the Netherlands as an intermediary country to send money through on a route from one country to another country. In this sense, the Netherlands is not a *tax haven* (a destination country where the money is stored) like the Bahamas or Bermuda, but a *conduit* country (an intermediary country on a route via which companies send their money).

In this thesis, we will study algorithms for the network analysis of bilateral tax treaties from a mathematical perspective. Furthermore, we will apply the algorithms to investigate the role of the Netherlands and other countries as *conduit* countries, using data provided by the CPB.

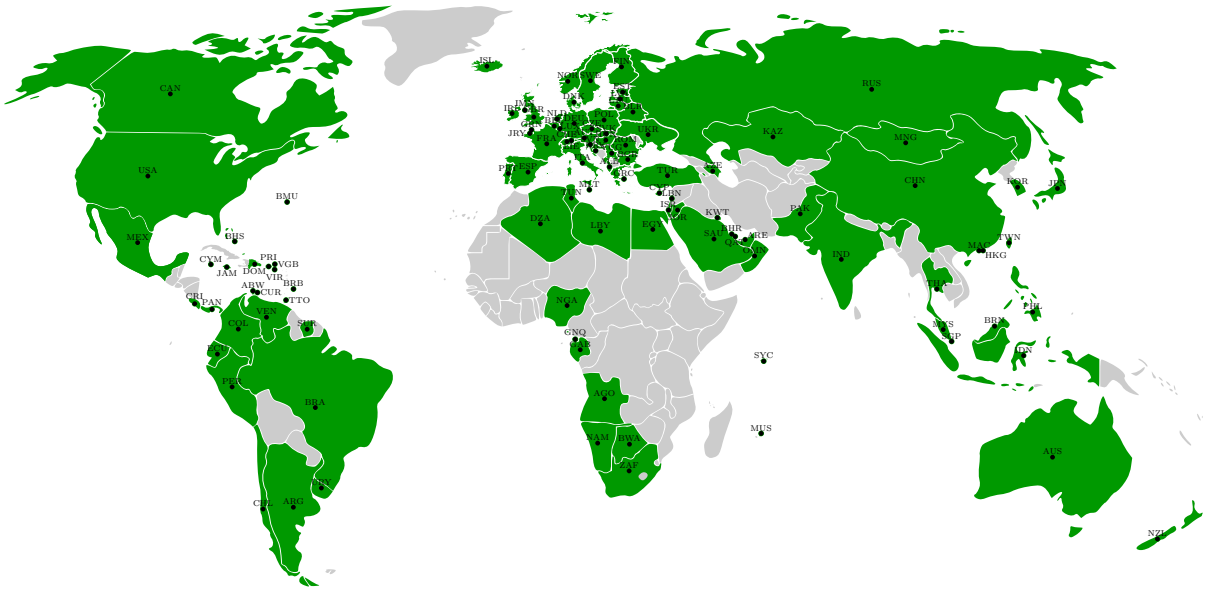


Figure 1: All countries/jurisdictions in the CPB-dataset are colored green.

Model

We are given data of 108 countries/jurisdictions[1], and we are given the tax rates that a company must pay when sending money from one country to another country. The countries in our

[1]Strictly spoken, not all of the given jurisdictions are countries. An example is Hong Kong (HKG). However, in this thesis we will use the term 'country' to refer to one of the 108 jurisdictions in our data, even if this jurisdiction area is in fact not a country.

dataset are shown in Figure 1. Each country is labeled by a code consisting of three letters. For example: the code 'NLD' stands for the country 'the Netherlands'. In the Appendix one can see the country names that correspond to the three-letter codes used throughout the thesis. Let $G = (V, E)$ be a complete directed graph[2], with V consisting of these 108 countries. The graph G is *complete* and *directed*, i.e. for every two countries $u, v \in V$ there are directed edges (u, v) and (v, u) in E.

Suppose that a company from *home* country v has made profits in some other *host* country u, for example because this company started a subsidiary company in country u. If the company wants to return its profits from u to v, it might have to pay some tax (possibly affected by bilateral tax treaties between u and v). Let $t_{u,v}$ be the tax (as a fraction between 0 and 1) that the company must pay if it sends its profits directly from u to v. The CPB provided us with these tax rates. The CPB-data only contains tax rates that companies must pay when sending their *profits* (dividends) from one country to another country. Hence, we do not take other tax-constructions (for example, *royalties*) into account (see [RL14]). Tax rates are usually given as percentages between 0 and 100. We divide this percentage by 100 to obtain a fraction between 0 and 1. Furthermore, we define a function $r : E \to [0, 1]$ by

$$r(u, v) = 1 - t_{u,v} \quad \text{for each } e = (u, v) \in E.$$

That is, $r(u, v)$ is the fraction of money that, when sent directly from u to v, arrives at v. We call the function r a *reliability function*.[3]

Example. A multinational wants to return its profits from country A (the 'host' country) to country B (the 'home' country). Countries A and B have a bilateral tax agreement, in which they have agreed that 20 percent tax must be payed on profits that a company sends from country A to B, i.e. $r(A, B) = 1 - 0.20 = 0.8$. Country C has bilateral tax agreements with country A as well as with country B. On profits that are sent from A to C, 10 percent must be payed. On profits from C to B also 10 percent tax is levied. Now, the company can send the money from A to C and then from C to B. Therefore

$$r(A, C) = r(C, B) = 1 - 0.10 = 0.9.$$

The following network represents the above situation (where we only draw the relevant edges):

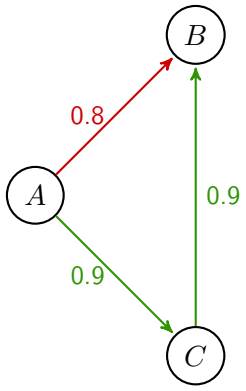


Figure 2: Route $A - C - B$ has reliability $0.9 \cdot 0.9 = 0.81$, while route $A - B$ has reliability 0.8.

If the company sends money directly from A to B, a fraction of 0.8 of this money will arrive at B. But if the company sends money via the route $A - C - B$, a fraction of $0.9 \cdot 0.9 = 0.81$ will arrive

[2]See the preliminaries for a definition of graphs.

[3]Often with a 'reliability function' is meant a function that gives the probability that a given item operates for a certain amount of time without failure. However, we will use the term 'reliability function' throughout the whole thesis in another context: the share of money that, when sent from one country to another country, arrives at the destination country.

at C. Therefore it is more profitable for the company to use the route $A-C-B$ than the direct route $A-B$. In tax percentages: on profits sent over route $A-C-B$ a percentage of $100-81 = 19\%$ tax is withheld. Over route $A-B$ the imposed tax is $100-80 = 20\%$. Therefore it is more profitable for the company to send its money over the indirect route $A-C-B$. The example shows that the most profitable route for a company is not always the direct route. Multinationals can use indirect routes to reduce the tax they must pay. This is called *tax treaty shopping*.

In this thesis we will conduct a network analysis of these 'tax' routes. To investigate which countries are the most important conduit countries, we will study a notion for measuring the centrality of a vertex (country) in the network: *betweenness centrality*. When computing the betweenness centrality we consider all 'most profitable' tax-routes for companies. On what fraction of these 'most profitable' routes does a country appear as a conduit country, i.e. on what fraction of the most profitable routes is a country situated *between* the starting point and the end point of the route? This will give a measure for the 'centrality' of a country in the network. We will in particular consider *weighted betweenness centrality*. In *weighted* betweenness centrality, paths that start and end at important countries (measured according to the size of their economies) get a higher weight and count more for the betweenness centrality than paths between less relevant countries (countries with smaller economies).

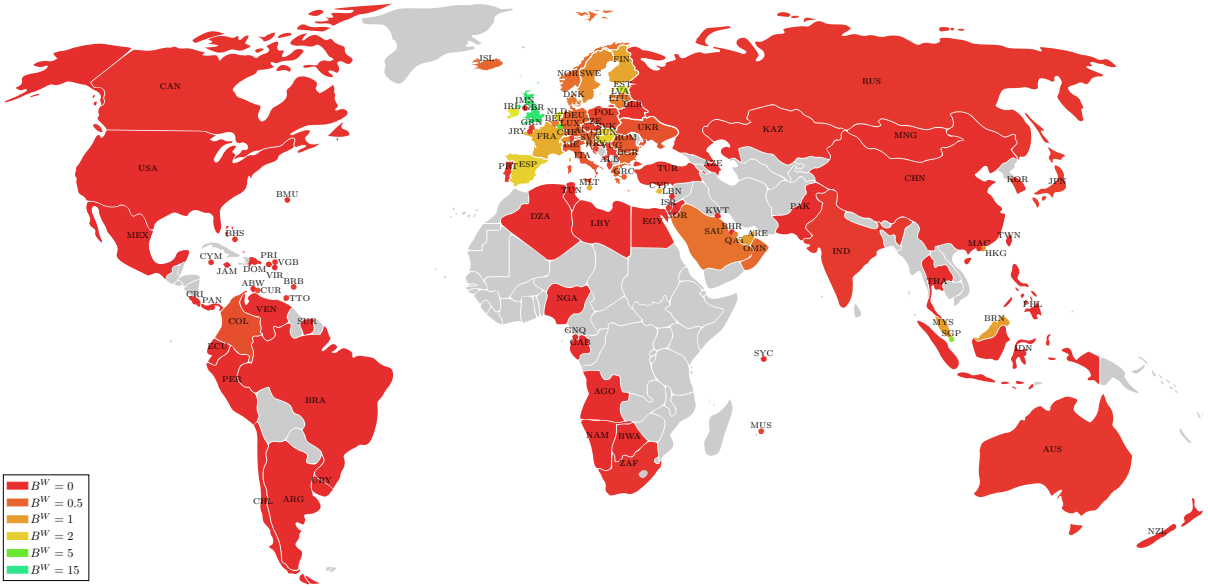


Figure 3: The countries that are more central in the network have a higher weighted betweenness centrality value B^W.

Structure of the thesis

We give an overview of the structure of the thesis.

(C1.) Chapter 1 contains the preliminaries. Readers with some knowledge of graphs, complexity theory and approximation algorithms can skip this chapter.

(C2.) In Chapter 2 adaptations of known shortest path algorithms to the CPB-network will be given and we will study a tool for measuring the centrality of a vertex (country) in the network: *betweenness centrality*. In computing the betweenness centrality we consider all 'most profitable' tax-routes for companies. On what fraction of these 'most profitable' routes does a country appear as a conduit country? This gives a measure for the 'centrality' of a country in the network.

(C3.) Our network contains a lot of edges with reliability 1, i.e. edges along which companies can send money for free. This makes counting maximum reliability paths difficult, as the maximum reliability path graph then can contain cycles. In Chapter 3 we will identify 'clusters' of countries that have many edges with reliability 1 between them and we shortly consider an approach of 'shrinking' these clusters to deal with the difficulty of counting paths.

(C4–5.) In Chapter 4 and 5 we study the following question: what if we are not only interested in the most profitable routes, but also in routes that are almost 'most profitable'? Finding all simple s,t-paths within a certain range is $\#P$-complete. We will find a 'restricted relative range notion' to compute 'relevant paths' in polynomial time.

(C6.) A country may be interested in *maximizing* the amount of money that companies send through this country to other countries. This will attract jobs in the financial sector to this country[4]. In our model the amount of money that companies send through a country will equal the weighted betweenness centrality. We will see that the problem of maximizing the (weighted) betweenness of one node by setting the reliability of at most k outgoing edges to 1 is NP-hard. We will derive a $1 - 1/e$-approximation algorithm based on the approximation algorithm for *Maximum Coverage*. It turns out that the function we try to maximize is a submodular nondecreasing set function. We will finally test this algorithm to maximize the amount of money that companies send through the Netherlands. This all is done in Chapter 6.

(C7.) In Chapter 7 we think about what a country must do in order to maximize the tax it receives on the money that is send through this country. Often countries might be not very interested in maximizing this tax: in order to attract jobs in the financial sector, countries have low taxes for sending money through them. Nevertheless the problem of maximizing tax revenues is interesting from a theoretical point of view.

(C8.) Which countries are the most important in our network? We return to this question in Chapter 8. We will study a new measure from a recent article [SMR12], the *Shapley-value based betweenness centrality*. The original betweenness centrality measures the importance of an individual vertex in the network. How severe are the consequences for the possibility to communicate between vertices in the network if this particular vertex fails? It is argued (see [SMR12]) that the original betweenness centrality is not an adequate measure for many applications, since in practice many nodes can fail simultanuously. The *Shapley-value based betweenness centrality* deals with this limitation: it is a measure that measures the importance of a vertex *as a member of all possible subsets* of vertices in G. We will study this measure and apply it to our network of countries.

The end of each chapter (except for the preliminaries and Chapter 7) will contain a small section 'Results'. There we apply the algorithms to the CPB-network of 108 countries. For example, at the end of Chapter 2 we will consider the most important *conduit* countries, countries that are frequently used to send money through.

Experimental insights

The main conclusions of the experimental part of this thesis can be summarized as follows.

(1.) The Netherlands ranks quite high, they are 5th in the ranking according to weighted betweenness centrality. This seems to give some evidence for the news headers that the Netherlands is an attractive tax country for companies.

[4]For example, this country will attract letterbox companies.

(2.) Great Britain (GBR) is the most central country in the network. Great Britain has a substantially higher (weighted) betweenness centrality, as well as a substantially higher Shapley-value based betweenness centrality, than any other country in the network.

(3.) There exist always a 'most profitable' route from any country to any other country passing through at most 3 conduit countries.

(4.) The results remain quite stable if we also take paths into account that are almost 'most profitable'.

(5.) If the Netherlands wants to improve its role as a conduit country (measured with weighted betweenness centrality), it is a good idea to set the outgoing tax rate to India to zero, and after that to set the the tax rates to China and Brazil to zero. See Chapter 6.

(6.) The Shapley-value based betweenness centrality applied on our network of countries gives similar results as the original betweenness centrality. However, this measure can differentiate more between vertices that have a low betweenness centrality. Interpretation for this fact will be given in Chapter 8.

(7.) The algorithms are implemented in Java and run (considerably) faster than the implementations by the CPB. If the CPB does a follow-up research on [RL14], they will use the algorithms programmed for this thesis.

The results will be illustrated via color-coded maps, similar to the one in Figure 3.

Mathematical contributions

The most important mathematical contributions of this thesis are the following.

(1.) We will see that the problem of maximizing the betweenness of one node by setting the reliability of at most k outgoing edges to 1 is NP-hard. We will derive a $1-1/e$-approximation algorithm based on the approximation algorithm for *Maximum Coverage*. It turns out that the function we try to maximize is a submodular nondecreasing set function. See Chapter 6. This algorithm can be used to investigate how a country can increase its centrality in the network.

(2.) The problem of counting all simple s,t-paths with distance within a certain range of the shortest path is $\#P$-complete. We will find a 'restricted relative range notion' to compute 'relevant paths' in polynomial time.

(3.) We will consider ways of dealing with 'zero'-cost edges in the shortest path graph (edges of reliability 1 in the maximum reliability graph). This is done in Chapter 3.

(4.) Suppose a country controls the tax rate on its outgoing edges. At what reliabilities must a country set its edge reliabilities, so that the tax it receives is maximized? We will formalize this problem and shortly look into it. See Chapter 7.

(5.) As a minor contribution, we discover a small mistake in a recent article (2012) by P. L. Szczepánski, T. Michalak and T. Rahwan (see [SMR12]): the article is about computing the Shapley-value based betweenness centrality in undirected graphs. In the article, an adaptation to directed graphs is given. We will see that this adaptation is not entirely correct. See Chapter 8.

This is only an indicatory summary of the mathematical results found in this thesis. I wish the reader pleasure while reading the whole thesis!

Sven Polak
December 2014

Chapter 1

Preliminaries

This chapter contains an introduction to basic notions in graph theory, graph search algorithms and complexity theory used in this thesis. Readers with some knowledge of the mentioned topics can safely skip this chapter. This chapter freely uses definitions from [Schä13].

1.1 Graphs

A *graph* $G = (V, E)$ consists of a finite set of vertices V together with a finite set E of edges. Each edge $e \in E$ is associated with a pair (u, v) in $V \times V$. The graph is *undirected* if the edges are *unordered* pairs of vertices[1]. For an edge $e = (u, v) \in E$, the vertices u and v are the *endpoints* of e. In this thesis we will often use the notations $n := |V|$ and $m := |E|$.

A graph G is said to be *directed* if the edges $e \in E$ are *ordered* pairs of vertices. In this case the edge $(u, v) \in V \times V$ is different from edge (v, u). For a directed edge $e = (u, v)$, the endpoint u is the *tail* of e and the endpoint v is the *head* of e.

A graph is *simple* if it contains no parallel edges[2] and no self-loops[3]. Throughout this thesis we will assume that graphs are *directed* and *simple*, unless stated otherwise. A *subgraph* $H = (V', E')$ of a graph $G = (V, E)$ is a graph with $V' \subseteq V$ and $E' \subseteq E$.

Example 1.1.1. An example of a graph is the 'CPB-network' of 108 countries. The following picture shows this network, along with some edges. The graph $G = (V, E)$ of countries used in this thesis is simple, directed and *complete*. That means that for every $u, v \in V$ with $u \neq v$, there exist edges (u, v) and (v, u) in E.

A *path* $P = \langle v_1, \ldots, v_j \rangle$ in a graph $G = (V, E)$ is a sequence of vertices such that for all $i \in \{1, \ldots, j-1\}$ it holds that (v_i, v_{i+1}) is an edge of G. We also say that P *contains* the edges (v_i, v_{i+1}), $i = 1, \ldots, j-1$. We call P a v_1, v_j-path. If every vertex appears in P at most once, P is called *simple*. If $P_1 = \langle s = v_1, \ldots, u = v_j \rangle$ is an s, u-path and $P_2 = \langle u, w_1, \ldots, t = w_k \rangle$, then we call the s, t-path $\langle s = v_1, \ldots, u = v_j = w_1, \ldots, t = w_k \rangle$ the *concatenation* of P_1 and P_2.

A cycle $C = \langle v_1, \ldots, v_j = v_1 \rangle$ is a path that start and ends at the same vertex. A graph is said to be *acyclic* if it does not contain a cycle. In this thesis we often talk about *directed acyclic graphs* (or shortly *DAG*s). We will see that they have a nice property: it is possible to count simple paths in a directed acyclic graph very efficiently.

A *tree* T is an undirected graph in which any two vertices are connected by *exactly one* path. A *rooted tree* is a tree in which there is a root $s \in V(T)$ (where $V(T)$ stands for the vertices of T) and all edges have an orientation that is either away from or towards the root. Hence, a rooted tree is a directed graph. In this thesis we will assume (unless otherwise mentioned) that

[1] That is, (u, v) is the same edge as (v, u).

[2] Two undirected edges are parallel if they have the same endpoints. Two directed edges are parallel if their tails and their heads are the same.

[3] A self-loop is an edge in which both endpoints are equal.

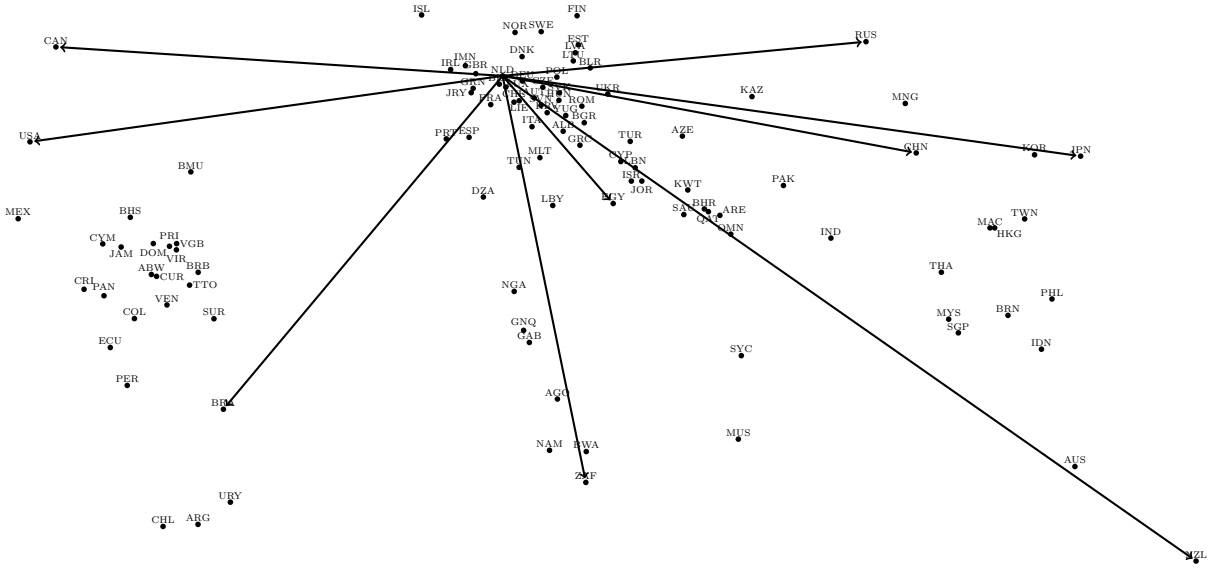


Figure 1.1: The CPB-network we study contains 108 countries (vertices). Every vertex is connected to each other vertex by a directed edge. For simplicity, only some outgoing edges of the Netherlands are depicted.

all trees are rooted trees with orientation *away* from the root. If a vertex u is on the (unique) path from the root s to a vertex v, then u is called an *ancestor* of v and v is called a *descendant* of u. For any node $v \neq s$ in a tree (with s as root), the predecessor u of v on the unique s, v-path is called the *parent* of v, and v is the *child* of u.

1.2 Discrete optimization problems and algorithms

In this section we define discrete optimization problems. After that, we will introduce *algorithms* to solve these problems.

Definition 1.2.1 (Discrete optimization problem)**.** A *discrete (minimization or maximization) optimization problem* A is given by a set of instances $\mathcal{I}$. Every instance $I \in \mathcal{I}$ specifies

(i) a *discrete*[4] set $\mathcal{F}$ of feasible solutions,
(ii) a *cost* function $c : \mathcal{F} \to \mathbb{R}$.

Suppose we are given an instance $\mathcal{I} = (\mathcal{F}, c)$. The goal is to find a feasible solution $F \in \mathcal{F}$ such that $c(F)$ is minimum (in the case of a minimization problem) or maximum (in the case of a maximization problem). Such a solution is called an *optimal* solution of I.

Example 1.2.1. The *Shortest Path Problem* is a minimization problem. An instance is a graph $G = (V, E)$ with edge costs $c : E \to \mathbb{R}$, a source vertex $s \in V$, a sink node $t \in V$, with

$$\mathcal{F} = \{P \subset V : P \text{ is an } s,t\text{-path in } G\} \quad \text{and} \quad c(P) = \sum_{e \in P} c(e).$$

The goal is to find a simple s, t-path in G of minimal cost. The next chapter will be about a multiplicative version of the shortest path problem.

Now we have defined discrete optimization problems, we can talk about *algorithms* to solve them.

Definition 1.2.2 (Algorithm)**.** An algorithm for a discrete optimization problem A is a procedure (a sequence of instructions) that solves every given instance I.[5]

[4] A discrete set is a countably inifinite or finite set.

[5] This definition is somewhat informal. The formal model of 'Turing machines' we will not define here.

We care about the *efficiency* of the algorithm, i.e. about the running time of the algorithm. This time is measured in the number of basic operations. We focus on the *worst case* running time. The size $|L(I)|$ of an instance I we define as the number of bits that are needed to store I on a computer using encoding L. Throughout this thesis we measure the size of an instance not in bits, but in the number of objects (for example vertices and edges).

Remark 1.2.1. The storage space required by a certain instance often relies on the underlying datastructure. For example, a graph $G = (V, E)$ can be stored in different ways. We give two examples. We write $V = \{v_1, \ldots, v_n\}$.

(i) A graph can be stored by an *adjacency matrix* A of size $|V| \times |V|$. The adjacency matrix contains on the i, j-th position a 1 if there exists an edge (v_i, v_j) and a 0 if there does not exists such an edge. This representation of a graph takes $|V|^2$ storage space. An advantage of this representation is that one can see in constant time whether there is an edge (v_i, v_j) in E: one just needs to look at the i, j-th entry of A. To find all neighbours of one vertex v_i, one needs time $|V|$, since the entire i-th row of A must be scanned.

(ii) A graph can be stored by *adjacency lists*. For every vertex $v \in V$, a list of *neighbours* is kept. A vertex w is a neighbour of vertex v if and only if there is an edge (v, w) in E. This representation can be done in $|V| + 2|E|$ storage space for undirected graphs and $|V| + |E|$ storage space for directed graphs. A disadvantage of this representation is that one needs time bounded by $|L_i|$ (where L_i is the adjacency list of vertex i) to check whether there exists an edge (v_i, v_j) in E: the edges in the adjacency list of v_i need to be scanned. An advantage of this representation is that it only takes time $|L_i|$ to find all neighbours of vertex v_i.

As we have seen, both representations have certain advantages and disadvantages. The most suitable representation depends on the application.

Definition 1.2.3 ((Worst case) running time)**.** If A is a discrete optimization problem and L an encoding of the instances, we say that an algorithm ALG solves A in *worst case running time* f if ALG computes for every instance I of size $|L(I)|$ an optimal solution $F \in \mathcal{F}$ using at most $f(|L(I)|)$ operations.

To measure the running time of a function, it is useful to use asymptotic notation.

Definition 1.2.4. Let $g : \mathbb{N} \to \mathbb{R}_{\geq 0}$. We write:

$$\mathcal{O}(g(n)) = \{f : \mathbb{N} \to \mathbb{R}_{\geq 0} \ : \ \exists\, C > 0,\, N \in \mathbb{N} \text{ such that } f(n) \leq C \cdot g(n)\ \forall\, n \geq N\}$$
$$\Theta(g(n)) = \{f : \mathbb{N} \to \mathbb{R}_{\geq 0} \ : \ \exists\, c, C > 0,\, N \in \mathbb{N} \text{ such that } c \cdot g(n) \leq f(n) \leq C \cdot g(n)\ \forall\, n \geq N\}.$$

During this thesis we will consequently use the notation $f = \mathcal{O}(g(n))$ (resp. $f = \Theta(g(n))$) when we formally mean $f \in \mathcal{O}(g(n))$ (resp. $f \in \Theta(g(n))$).

Example 1.2.2. It holds that $80 \cdot 365 \cdot n^{365} = \Theta(n^{365})$, $n \log(n^{7874578934}) = \mathcal{O}(n^2)$, etcetera.

We will often use this notation when talking about the running time of algorithms.

1.3 Complexity theory and approximation algorithms

This section is about complexity theory. We will define the complexity classes P and NP. We also define NP-complete problems. After that we will define the complexity class $\#P$ and specify when a problem is $\#P$-*complete*.

Definition 1.3.1 (Decision problem)**.** A *decision problem* A is defined by a set of instances $\mathcal{I}$, where each instance $I \in \mathcal{I}$ specifies:

(i) a set $\mathcal{F}$ of feasible solutions for I,
(ii) a yes/no-function $c : \mathcal{F} \to \{1, 0\}$.

For an arbitrary instance $I = (\mathcal{F}, c) \in \mathcal{I}$, we would like to decide whether there exists a feasible solution $S \in \mathcal{F}$ with $c(S) = 1$. If there is such a solution, I is a *yes*-instance, otherwise I is a *no*-instance.

Now we define the class *NP* of decision problems which admit a certificate that can be verified in polynomial that.[6]

Definition 1.3.2 (Complexity class *NP*)**.** A *decision problem* A is contained in the 'complexity class' *NP* if every *yes*-instance has a certificate whose validity can be checked in polynomial time, i.e. in time f (cf. definition 1.2.3), where f is a polynomial.[7]

Example 1.3.1. Given a natural number M_1, determine whether there exists a natural number M_2 such that $M_1/M_2 = 2$. This is an example of a (very easy) problem in *NP*. The set $\mathcal{F}$ of all feasible solutions consists of all natural numbers. For $F \in \mathcal{F} = \mathbb{N}$, we have $c(F) = 1$ if and only if $M_1/F = 2$: it can be checked in polynomial time whether F is a certificate for M_1, i.e. whether $F/M_2 = 2$. This can be done in polynomial time, using a long division algorithm.

Next we define the complexity class *P*: the class of complexity problems that can be solved in *polynomial-time*.

Definition 1.3.3 (Complexity class *P*)**.** A decision problem A is contained in the complexity class *P* if there exists an algorithm that determines for every instance $I \in \mathcal{I}$ whether I is a *no*-instance or a *yes*-instance.

Example 1.3.2. Given a natural number M_1, determine whether there exists a natural number M_2 such that $M_1/M_2 = 2$. This problem is in the complexity class *P*. It can be verified whether M_1 is a *yes*-instance by looking at the last digit of M_1. If the last digit is $0, 2, 4, 6$ or 8, then M_1 is a yes-instance. Therefore we can check in constant time whether M_1 is a yes-instance.

It holds that $P \subseteq NP$. This is because a polynomial-time algorithm to solve a problem in P can be seen as a as a polynomial-time algorithm with as input a certificate of zero length. Currently it is not known whether $P = NP$. Most people think that $P \neq NP$. It is one of the open *millennium problems* to solve whether $P = NP$ (see [Coo00]). If you solve it, you can earn *one million dollar*.

We will now define the complexity class of *NP*-complete problems, a subclass of *NP*. The *NP*-complete problems are the 'hardest' problems in *NP*: if one finds a polynomial time algorithm to solve *one* *NP*-complete problem, then there exists a polynomial time algorithm to solve *every* problem in *NP*.

Definition 1.3.4 (*NP*-complete problem)**.** A decision problem A is an *NP*-complete problem if:

(i) A belongs to the complexity class *NP*,
(ii) Every problem in *NP* is polynomial-time reducible to A. By this we mean that for every problem B in *NP* there exists a function $\phi : \mathcal{I}_1 \to \mathcal{I}_2$ that maps every instance $I_1 \in \mathcal{I}_1$ of B to an instance $I_2 \in \mathcal{I}_2$ of A, such that
   - I_1 is a yes-instance of A if and only if I_2 is a yes-instance of B,
   - the mapping can be done in time polynomially bounded in the size of I_1.

[6] *NP* does not stand for "non-polynomial time", but for "non-deterministic polynomial time".
[7] This means that $f = \mathcal{O}(n^M)$ for some $M \in \mathbb{N}$.

This means that if we find a polynomial time algorithm for A, we can solve *every* problem B in NP by the following procedure: map an instance of problem B to an instance of problem A in polynomial time using ϕ and then use the polynomial time algorithm for A to determine whether the instance of A is a *yes*-instance. This is the case if and only if the original instance for B is a *yes*-instance too.

There are many of examples of NP-complete problems (see [GJ79]). One example is the following.

Example 1.3.3 (Exact Cover Problem)**.** Given a universe $U = \{1, \ldots, N\}$ and a collection of subsets of this universe $S = S_1, \ldots, S_t$, find a subcollection of sets $S' \subseteq S$ such that each element $x \in U$ is contained in *exactly one* subset in S'.

There are also problems that are *at least* as hard as any problem in NP, but are themselves not contained in NP. These problems form the complexity class of NP-hard problems.

Definition 1.3.5 (NP-hard problem)**.** A decision problem A is an NP-hard problem if every problem in NP is polynomial-time reducible[8] to A.

The following problem (the *Maximum Coverage Problem*) is known to be NP-hard (see [Fei98]). We will use it in this thesis in some of our reductions.

Example 1.3.4 (Maximum Coverage Problem)**.** Given a universe $U = \{1, \ldots, N\}$, a collection of subsets of this universe $S = S_1, \ldots, S_t$ and a number k, find a subcollection of sets $S' \subseteq S$ such that $|S'| < k$ and the total number of covered elements $|\cup_{S_i \in S'} S_i|$ is maximal.

Sometimes, when A is a decision problem and I is an instance of A, we do not only want to find one certificate showing *whether* an instance is a yes-instance or a no-instance, but we want to know *how many* certificates for the yes-instance there exist. For example, in a graph we might want to know not only *whether* there exists a simple s-t-path, but we might want to know *how many* simple s-t-paths there are. This is a 'counting'-problem. First we properly define a counting problem. Subsequently we define the associated complexity class $\#P$.

Definition 1.3.6 (Counting problem)**.** A *counting problem* A is defined by a set of instances $\mathcal{I}$, where each instance $I \in \mathcal{I}$ specifies:

(i) a set $\mathcal{F}$ of feasible solutions for I,
(ii) a yes/no-function $c : \mathcal{F} \to \{0, 1\}$,

For an arbitrary instance $I = (\mathcal{F}, c) \in \mathcal{I}$, we want the *number of* feasible solutions $S \in \mathcal{F}$ with $c(S) = 1$.

Definition 1.3.7 (Complexity class $\#P$)**.** Let A be a counting problem. We say that A is contained in the 'complexity class' $\#P$ if the decision-version of A is contained in NP.

Next we define $\#P$-completeness.

Definition 1.3.8 ($\#P$-complete problem)**.** A counting problem A is called *$\#P$-complete* (*Sharp P-complete*) if:

(i) $A \in \#P$
(ii) and every problem $B \in \#P$ can be reduced to A by a polynomial time counting reduction. A counting reduction from a problem B to A consists of:
   - A function ϕ that maps each instance $I \in B$ to an instance $\phi(I) \in A$.
   - A function f that retrieves from the count n of $\phi(I)$ in A the count $f(n)$ of I in B.

[8] As defined in the definition of NP-completeness.

Note that the counting version of a problem in *NP* is at least as hard as the decision version: if we can count in polynomial time the number of certificates for a yes-instance, then we also know in polynomial time whether there exists a certificate for a yes-instance.

Example 1.3.5 (Counting simple s,t-paths)**.** Let $G = (V, E)$ be a graph, with s a source and t a sink vertex. The counting version of the simple s,t-path problem is $\#P$-complete. That is, the problem of finding the number of simple s,t-paths in an arbitrary graph is $\#P$-complete (see [Val79]). In Chapter 4 we will use this example to show that counting all paths within a certain range from the shortest path is $\#P$-complete.

Many optimization problems are *NP*-hard and it is unlikely that we find efficient algorithms for these problems. One way to cope with the hardness of a problem is developing an *approximation algorithm*. An approximation algorithm is an efficient algorithm that computes a suboptimal feasible solution with a provable approximation guarantee. We will now give the formal definition.

Definition 1.3.9 (Approximation algorithm)**.** An algorithm ALG for a discrete minimization problem A (resp. for a maximization problem) is an *α-approximation algorithm* with $\alpha \geq 1$ if it computes for every instance $I \in \mathcal{I}$ a feasible solution $F \in \mathcal{F}$ with cost $c(F)$ is at most α times (resp. at least $1/\alpha$ times) the cost $\mathrm{OPT}(I)$ of an optimal solution, i.e.

$$c(F) \leq \alpha \cdot \mathrm{OPT}(I) \qquad \left(\text{resp. } c(F) \geq \frac{1}{\alpha} \cdot \mathrm{OPT}(I)\right).$$

Of course one would like to have α as small as possible.

When analyzing the approximation performance of an approximation algorithm, two questions arise.

(i) Is the approximation ratio α of this particular algorithm *tight*? I.e. does there not exist a better approximation ratio for this particular algorithm? This can be proven by giving an example for which the solution is α (resp. $1/\alpha$) times the optimal one.

(ii) Do there exist no other polynomial time algorithms that give a better approximation bound? I.e. is the found approximation algorithm *the best possible* approximation algorithm for our problem?

In Chapter 6 we will discover an *NP*-hard problem and find an approximation algorithm for it. Furthermore we will see that the bound α for this particular algorithm is tight and that there do not exist polynomial time approximation algorithms with a better approximation ratio, unless $NP \subseteq DTIME(n^{\mathcal{O}(\log\log n)})$. The class $DTIME$ consists of all decision problems that can be solved in a particular time. It is considered very unlikely that $NP \subseteq DTIME(n^{\mathcal{O}(\log\log n)})$.

1.4 Depth-first search and topological sort

In this section we consider a well-known graph search algorithm: *depth-first* search. After that, we will define a *topological sort* of a directed graph and we will give an algorithm based on depth-first search that returns a topological sort of the vertices in a directed acyclic graph. This section is based on the description of both algorithms in [CLR01].

Depth-first search is a graph search algorithm. It *searches* through all the nodes. First, all vertices are colored *white*. The algorithm starts searching at some vertex and discovers all neighbours of this vertex. When a vertex is discovered for the first time, this vertex is colored *green*. The algorithm recursively searches through all the neighbours of this vertex. When the algorithm finishes searching at a vertex (and hence already finished recursively searching the neighbours of this vertex), the vertex is colored *red*.

While executing the algorithm, we keep a time counter $d[v]$ that stores at which step vertex v is colored green (at that moment vertex v is first *discovered*) and a time counter $f[v]$ that stores at which step in the algorithm vertex v is colored red, then the algorithm is *finished* searching at this vertex. Therefore we call $d[v]$ and $f[v]$ *discovery* and *finishing times*, respectively.

When the algorithm considers edge (u, v) and v is colored white, the algorithm discovers v and colours v green. Furthermore, the algorithm sets $\pi[v] := u$, the parent of v is u. The vertex s at which the algorithm started searching has no parent. The algorithm produces a *depth-first* tree $T = (V', E_\pi)$, where V' consists of the vertices reachable from s and

$$E_\pi := \{(\pi[v], v) : v \in V' \text{ and } \pi[v] \neq \text{NIL}\}.$$

Why is T a tree? First we observe that every vertex that is discovered (and gets a parent) is reachable from s by graph edges. Furthermore, a vertex only gets assigned a parent when it is colored white, hence it is clear that there can be no cycles. If there would be a cycle, then last edge on the cycle that the algorithm explores must have a white head, which is not possible since this vertex already is searched (as there is already an edge leaving this vertex in the cycle), so this vertex is already green or red.

Input: Directed graph $G = (V, E)$, source vertex s.
Output: Depth-first search tree T.

Initialize: v is *white* for every $v \in V$, array $\pi[\,]$ of size $|V|$,
arrays $d[\,]$, $f[\,]$ and $\pi[\,]$ of size $|V|$, time counter $t := 0$.

DFS-visit(s)

Procedure DFS-visit(u)
    Color vertex u green
    $t := t + 1$
    $d[u] := t$
    foreach neighbour v of u **do**
        if Color(v) = white **then**
            $\pi[v] := u$
            DFS-visit(v)
        end
    end
    Color vertex u red
    $f[u] := t := t + 1$
end

return π

Algorithm 1: Depth-first search.

Every vertex is searched once, and when a vertex is searched, all neighbours of this vertex are searched. Therefore a depth-first search can be performed in time $\mathcal{O}(|V| + |E|)$. Note that the depth-first search T contains a path from s to every vertex $v \in V$ that is reachable from s, and hence by depth-first search we obtain a path from $s \in V$ to every $v \in V$ in time $\mathcal{O}(|V| + |E|)$.

The algorithm can be adapted to produce a *forest* (a collection of trees): each time that there are only red and white vertices, the algorithm can continue searching at an arbitrary white vertex. By doing this, depth-first search produces a collection of trees: a *depth-first* forest F.

We now will prove some important properties of depth-first search. These properties will also help us in finding a *topological sort*.

Lemma 1.4.1 (Parenthesis lemma)**.** *Suppose we perform a depth-first search on a graph $G = (V, E)$. After the depth-first search is finished, it holds for any two vertices u, v in V that*

(i) Either the intervals $[d[u], f[u]]$ and $[d[v], f[v]]$ are entirely disjoint,
(ii) or one of these two two intervals is fully contained in the other interval, i.e. it holds that

$$[d[u], f[u]] \subseteq [d[v], f[v]] \quad or \quad [d[v], f[v]] \subseteq [d[u], f[u]].$$

Proof. If $u = v$ then the lemma is obviously true since then both intervals are the same. Therefore we assume that $u \neq v$. Without loss of generality we assume that $d[u] < d[v]$, otherwise we interchange u and v. Suppose that (ii) does not hold: it does not hold that $[d[u], f[u]] \subseteq [d[v], f[v]]$. We will prove that $f[u] < d[v]$, implying that condition (i) holds and thereby proving the theorem (where we note that (i) and (ii) cannot occur at the same time).

Suppose to the contrary that $d[v] < f[u]$.[9] At the time that vertex v is discovered, vertex u was still green (because $d[u] < d[v] < f[u]$). Therefore v is a descendant of u. Since descendant v is discovered later than u, all neighbours of v are searched and finished before the search returns to and finishes u. Therefore it holds that $f[v] < f[u]$, i.e. condition (ii) holds, $[d[v], f[v]] \subseteq [d[u], f[u]]$, in contradiction with our assumption. □

Corollary 1.4.2 (Descendant corollary)**.** *Vertex v is a proper descendant*[10] *of vertex u in a depth-first forest for a graph $G = (V, E)$ if and only if $[d[v], f[v]] \subsetneq [d[u], f[u]]$.*

Proof. Vertx v is a proper proper descendant of u if and only if $d[u] < d[v] < f[u]$. By Lemma 1.4.1 this holds if and only if $[d[v], f[v]] \subsetneq [d[u], f[u]]$. □

Theorem 1.4.3 (White path theorem)**.** *Suppose we perform a depth-first search on a graph $G = (V, E)$. It holds that vertex $v \in V$ is a descendant of vertex $u \in V$ if and only if at the time $d[u]$ that u is discovered, vertex v can be reached from u along a path consisting only of white vertices.*

Proof. "$\Longrightarrow$". Suppose that v is a descendant of u. Let w be an arbitrary vertex on the u, v-path in the depth-first tree. Then w is a descendant of u. Hence $d[u] < d[w]$ so w is white at the time that vertex u is discovered.

"$\Longleftarrow$". Suppose that there is a vertex that reachable from u along a path of white vertices at time $d[u]$, but that does not become a descendant of u in the depth-first tree. Let v be the closest vertex to u along the path with this property. If w is the predecessor of v in the path, then is w a descendant of u (by the choice of v). Hence it holds that $f[w] \leq f[u]$. Vertex v is discovered after vertex u (since v was still white at the time that u was discovered) but before vertex w is discovered (otherwise v would be a descendant of w, and hence of u). Therefore it holds that

$$d[u] < d[v] < f[w] \leq f[u].$$

By Lemma 1.4.1 we conclude that $[d[v], f[v]] \subseteq [d[u], f[u]]$. Hence, vertex v must be a descendant of u. □

As an application of *depth-first* search, we will consider *topological sort*. Topological sorting of graphs will be often used throughout this thesis.

Definition 1.4.1 (Topological sort)**.** Let $G = (V, E)$ be a directed graph. A *topological sort* is an ordering of the vertices V such that for each edge (u, v), vertex u appears before vertex v in the ordering.

[9] Note that it cannot hold that $d[v] = f[u]$. This is only possible if $u = v$.
[10] A proper descendant of u is a descendant of u different from u.

If a graph contains a cycle, then clearly a topological sort is not possible. Conversely, we will see that *any* directed acyclic graph can be topologically sorted. A topological sort can be found in any directed acyclic graph (as we will see) with the following procedure.

Algorithm 1.4.1 (Algorithm: Topological sort)**.** Let $G = (V, E)$ be a directed acyclic graph. The following procedure yields a topological ordering of the vertices.

(i) Perform a *depth-first search* on the graph G, starting at an arbitrary source vertex s.
(ii) Each time a vertex is colored *red*, we put it at the end of a linked list[11].
(iii) The resulting list is a topological sort of V.

It is clear that the running time of this procedure is bounded by $\mathcal{O}(|V| + |E|)$, the running time of a depth-first search of graph G. We will discuss the correctness of the algorithm. First we state the definition of a *frond* edge. The concept of frond edges will be useful when proving the correctness of the Algorithm 1.4.1.

Definition 1.4.2 (Frond edge)**.** Let (u, v) be an arbitrary edge in $G = (V, E)$. Suppose we have performed a depth-first search on G. If (u, v) connects a vertex u to an ancestor v in the depth-first tree, then (u, v) is caled a *frond*.[12]

Remark 1.4.1. Note that a frond can be discovered during the depth-first search: if vertex v is green when edge (u, v) is considered by the algorithm for the first time, then edge (u, v) is a frond. To see this, note that the set of green vertices always forms a chain of descendants, and that is searched from the last green vertex in this chain. So if v is already green when edge (u, v) is first considered, then (u, v) is a frond.

We prove an auxiliary lemma. This lemma will help us to prove the correctness of Algorithm 1.4.1.

Lemma 1.4.4. *A directed graph $G = (V, E)$ is acyclic if and only if in a depth-first search of G, no fronds are discovered.*

Proof. "$\Longrightarrow$". If there is a frond (u, v) in G then u is a descendant of v and hence there is a path from v to u in G. Concatenating this path with edge (u, v) yields a cycle.

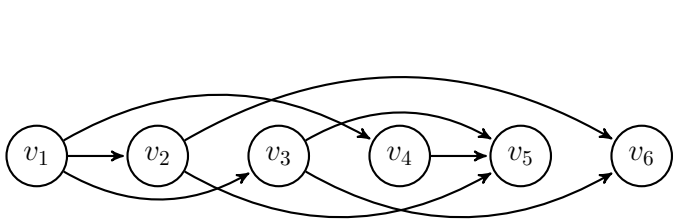
"$\Longleftarrow$". Suppose there is a cycle C in G. Let v be the first vertex of this cycle that is discovered by the depth-first search. Let (u, v) be the edge in C with head v. Since u can be reached along a path of white vertices at the time that v is discovered (by the choice of v), it holds that u is a descendant of v, by the white-path theorem (Theorem 1.4.3). Hence, (u, v) is a frond. □

Theorem 1.4.5. *Algorithm* 1.4.1 *finds a correct topological sorting in any directed acyclic graph $G = (V, E)$.*

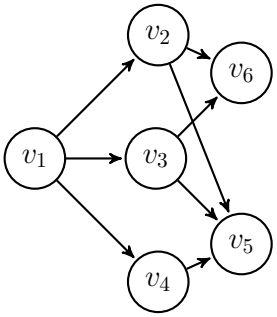
Proof. Suppose there is an edge (u, v) in a directed graph $G = (V, E)$. We will show that a depth-first search then gives $f[v] < f[u]$. When edge (u, v) is first explored by the depth first search, vertex v cannot be green. Then (u, v) would be a frond and hence, by Lemma 1.4.4, G would be not acyclic. If v is white, then v becomes a descendant of u and hence (remember Corollary 1.4.2) $f[v] < f[u]$. If v is red, $f[v]$ already has been determined, while $f[u]$ still needs to be determined (as the algorithm is searching at u). Therefore $f[v] < f[u]$. We conclude that in all cases it holds that $f[v] < f[u]$. Hence Algorithm 1.4.1 produces a correct topological sort of any directed graph $G = (V, E)$. □

With the proof of correctness of topological sort, we end the preliminary chapter of this thesis. If the reader would like to know more about elementary graph algorithms, the book of Cormen (see [CLR01]) can be recommended.

[11] A *linked list* is a list in which each element has a link to a certain successor in the list. The last element is linked to a terminator (which signifies the end of the list).
[12] In the literature (see [CLR01]) a frond is also called a *back(ward) edge*.



(a) A topologically sorted directed acyclic graph.



(b) The same topologically sorted graph, drawn in a different way.

Figure 1.2: A topologically sorted directed acyclic graph. Note that all edges are oriented from left to right.

Chapter 2

Maximum reliability paths

In this chapter we will see that 'maximum reliability paths' can be computed by using well-known shortest path algorithms from the literature (see for example the book of [CLR01], or [Schä13] or [Schr13]). We will see that the additive shortest path problem (see Example 1.2.1) and the multiplicative maximum reliability path problem (which we will define) are essentialy the same problems. Furthermore, we will see how to efficiently compute the (weighted) *betweenness centrality*: which countries are used as conduit countries the most? We conclude the chapter with results, where we rank all countries in our dataset according to their (weighted) betweenness centrality.

Let $G = (V, E)$ be a directed graph with a reliability function $r : E \to (0, 1]$ (if we have a graph with edge reliabilities of 0, then we simply remove these edges). For a path $P = \langle u_1, \ldots, u_k \rangle$ we define the *reliability* $r(P)$ of the path as follows:

$$r(P) := \prod_{i=1}^{k-1} r(u_i, u_{i+1}).$$

If $u, v \in V$ such that there exists a path from u to v, then there always exists a *simple* u, v-path of *maximum reliability*, as the next lemma shows.

Lemma 2.0.6. *Let $u, v \in V$ be two vertices of $G = (V, E)$, such that v is reachable from u. Then there exists a path of maximum reliability from u to v. Moreover, we can assume without loss of generality that this path is a simple path, i.e. it does not contain cycles.*

Proof. Suppose P is a path from u to v (by assumption such a path exists). If we remove all cycles from P, we obtain a simple path P'. The reliability of any cycle is in the interval $(0, 1]$. Therefore we see that $r(P')$ is obtained from $r(P)$ by repeatedly dividing by numbers in $(0, 1]$. Hence, it holds that $r(P') \geq r(P)$. It now remains to show that there exists a u, v-path of maximum reliability among all simple u, v-paths. This is trivial, since the set of all simple u, v-paths is finite. □

A u, v-path of maximum reliability is a most profitable path for companies to send their profits over. We now formally define the *Maximum Reliability Path Problem*.

Problem 1. The *Maximum Reliability Path Problem* is a maximization problem. An instance is a graph $G = (V, E)$ with edge reliabilities $r : E \to (0, 1]$, a source vertex $s \in V$, a sink node $t \in V$, with

$$\mathcal{F} = \{P \subset V : P \text{ is an } s, t\text{-path in } G\} \quad \text{and} \quad r(P) = \prod_{e \in P} r(e).$$

The goal is to find a simple s, t-path in G of maximum reliability. There are two important more general versions of the maximum reliability path problem:

(i) *The Single Source Maximum Reliability Path Problem.* Let $s \in V$ be a source vertex. For every $v \in V$, compute a path (or if possible: all paths) of maximum reliability from s to v, and count how many maximum reliability s, v-paths there are.

(ii) *The All Pairs Maximum Reliability Path Problem.* For every pair $(u, v) \in V \times V$ of vertices, compute a path (or if possible: all paths) of maximum reliability from u to v, and count how many maximum reliability u, v-paths there are.

Those problems the same as their respective shortest path problems (see [Schä13]), except that 'minumum cost' is replaced by 'maximum reliability'.

2.1 From additive edge costs to multiplicative edge reliabilities and vice versa

In this section we will see that the additive shortest path problem (see Example 1.2.1) and the multiplicative maximum reliability path problem are essentially the same problems. Therefore, by using the well-known algorithms to solve the shortest path problem, that can be found in [CLR01], [Schä13] and [Schr13], one can solve the maximum reliability path problem.

Let $G = (V, E)$ be a directed graph. Suppose $c : E \to \mathbb{R}_{\geq 0}$ is a cost function and edge costs are additive. We want to solve the additive shortest path problem. Suppose that there exists a function

$$\phi : \mathbb{R}_{\geq 0} \to (0, 1],$$

with the following properties:

(i) $\phi(x + y) = \phi(x) \cdot \phi(y)$ for all $x, y \in \mathbb{R}_{\geq 0}$,
(ii) ϕ is monotonely decreasing,
(iii) ϕ is bijective, and therefore $\phi^{-1} : (0, 1] \to \mathbb{R}_{\geq 0}$ is well-defined.

Note that property (iii) in combination with property (ii) implies that $\phi(0) = 1$ and that $\lim_{x \to \infty} \phi(x) = 0$. Define a reliability function

$$\begin{aligned} r : E &\to (0, 1] \\ r &= \phi \circ c. \end{aligned}$$

Lemma 2.1.1. *If a u, v-path $P = \langle u = u_1, \ldots, u_k = v \rangle$ is a maximum reliability path with respect to the reliability function r then it is a shortest path with respect to the cost function c.*

Proof. We have, by definition of the reliability function r,

$$r(P) = \prod_{e \in P} r(e) = \prod_{e \in P} \phi \circ c(e) = \phi \left(\sum_{e \in P} c(e) \right) = \phi(c(P)),$$

where the next-to last equality follows from property (i) of ϕ. Since P is a u, v-path of maximum-reliability, it holds that $r(P) \geq r(P')$ for every u, v-path P'. This means that $\phi(c(P)) \geq \phi(c(P'))$ for every path P' and hence by property (ii) of ϕ it holds that $c(P) \leq c(P')$ for every path P'. We conclude that P is a shortest path with respect to the cost function c. □

Now, suppose that $G = (V, E)$ is a directed graph and that multiplicative edge reliabilities $r : E \to (0, 1]$ are given.

Lemma 2.1.2. *If a u, v-path $P = \langle u = u_1, \ldots, u_k = v \rangle$ is a shortest path with respect to the cost function $c = \phi^{-1} \circ r$ then it is a maximum reliability path with respect to reliability function r.*

Proof. We have, by definition of the cost function c,

$$c(P) = \sum_{e \in P} c(e) = \sum_{e \in P} \phi^{-1} \circ r(e) = \phi^{-1} \left(\prod_{e \in P} r(e) \right) = \phi^{-1}(r(P)),$$

where the next-to last equality follows from property (i) of ϕ (this property implies that for ϕ^{-1} it holds that for all $x, y \in (0, 1]$ we have $\phi^{-1}(x \cdot y) = \phi^{-1}(x) + \phi^{-1}(y)$). Since P is a shortest u, v-path, we have $c(P) \leq c(P')$ for every u, v-path P'. This means that $\phi^{-1}(r(P)) \leq \phi^{-1}(r(P'))$ for every path P' and hence by property (ii) of ϕ (which implies that ϕ^{-1} is monotonely decreasing as well) it holds that $r(P) \geq r(P')$ for every path P'. We conclude that P is a maximum reliability path with respect to the cost function r. □

Hence, if we find a suitable function ϕ, we see that the additive shortest path problem and the multiplicative maximum reliability path problem are in fact the same problems[1]: if we can solve one of the two problems we have a solution for the other problem.

Remark 2.1.1. It is easily observed that $\phi : \mathbb{R}_{\geq 0} \to (0, 1] : x \mapsto e^{-x}$, with inverse $\phi^{-1} : (0, 1] \to \mathbb{R}_{\geq 0} : y \mapsto -\log(y)$, has the desired properties. We leave this (simple) check to the reader.

We conclude that the maximum reliability problem can be solved by solving the additive shortest path problem. However, in the next sections we will adapt known algorithms for the shortest path problem to solve the maximum reliability path problem *directly*. There are two main reasons for doing this.

(i) Computing the logarithm of numbers in $(0, 1]$ might involve very large numbers. Therefore it is more efficient to adapt shortest path algorithms *directly* to the maximum reliability path problem, without using the logarithm.
(ii) Readers without knowledge of shortest-path algorihtms can read about the adaptations, without the necessity of reading the literature first. Therefore the thesis will be self-contained.

Readers with knowledge of shortest-path algorithms, as *Dijkstra's Algorithm* or the *Algorithm of Floyd-Warshall*, can quickly scan through the next sections. The adaptations made to solve the maximum reliability path problem, are straightforward.

2.2 Algorithms for computing maximum reliability paths directly

Before adapting shortest path algorithms to compute maximum reliability paths, we introduce some notation and prove some auxiliary lemmas. The next two sections will be devoted to computing the 'distances'. Thereafter, we will see how knowing the distances enables us to find the maximum reliability paths. If $G = (V, E)$ is a graph with multiplicative edge reliabilities $r : E \to (0, 1]$, we define a distance function $\delta : V \times V \to [0, 1]$ as

$$\delta(u, v) = \begin{cases} \sup\{r(P) \,:\, P \text{ is a path from } u \text{ to } v\} & \text{if } v \text{ is reachable from } u \\ 1 & \text{if } u = v \\ 0 & \text{if } v \text{ is not reachable from } u. \end{cases}$$

[1] We could ask for a function $\phi : \mathbb{R} \to (0, \infty)$ as well, and the ϕ that we will give has all necessary properties with this domain and range. Then we would have identified the additive shortest path problem with arbitrary edge costs with the maximum reliability path problem with positive edge costs. Then a negative cycle in the additive case (which we then not allow) corresponds to a cycle of reliability bigger than 1 in the multiplicative case. However, we do not need this identification, as we are in this thesis only interested in reliabilities in the interval $(0, 1]$.

We see that

$$\begin{aligned} &\delta(u,v) = 0 && \text{if } v \text{ is not reachable from } u,\\ &\delta(u,v) \in (0,1], && \text{else.} \end{aligned}$$

The following lemma is an 'analogon' of the triangle inequality for reliability-paths.

Lemma 2.2.1. *Let $u, v \in V$ be vertices. For every edge $e = (v,w) \in E$, it holds that $\delta(u,w) \geq \delta(u,v) \cdot r(v,w)$.*

Proof. If $\delta(u,v) = 0$ then the relation holds trivially. If $\delta(u,v) > 0$ then there is a path from u to v of reliability $\delta(u,v)$. By appending the edge (v,w) to this path, we obtain a path of reliability $\delta(u,v) \cdot r(v,w)$. A maximum reliability path can only have bigger or equal reliability so therefore it holds that $\delta(u,w) \geq \delta(u,v) \cdot r(v,w)$. □

We now show that subpaths of maximum reliability paths are again maximum reliability paths.

Lemma 2.2.2. *Let $P = \langle u_1, \ldots, u_k \rangle$ be a maximum reliability path from u_1 to u_k. Then every subpath $P' = \langle u_i, \ldots, u_j \rangle$ of P with $1 \leq i < j \leq k$ is again a maximum reliability path from u_i to u_j.*

Proof. Suppose that there exists a path $P'' = \langle u_i, v_1, \ldots, v_l, u_j \rangle$ that has larger reliability than P'. Then the path $\langle u_1, \ldots, u_i, v_1, \ldots, v_l, u_j, \ldots, u_k \rangle$ is a u_1, u_k-path that has larger reliability than P. In fact the reliability of this new path is $r(P) \cdot r(P'')/r(P') > r(P)$. (Here we note that the reliabilities of the paths are nonzero positive numbers). This gives a contradiction with the assumption that P is a maximum reliability path. □

Definition 2.2.1 (Tight edge)**.** We call an edge $e = (v,w)$ *tight* with respect to the distance function $\delta(u,\cdot)$ if $\delta(u,w) = \delta(u,v) \cdot r(v,w)$.

We will prove that every edge in a maximum reliability path that starts at u is tight with respect to $\delta(u,\cdot)$.

Lemma 2.2.3. *Let $P = \langle u, \ldots, v, w \rangle$ be a u,w-path of maximum reliability. Then $\delta(u,w) = \delta(u,v) \cdot r(v,w)$.*

Proof. By Lemma 2.2.2 the subpath $P' = \langle u, \ldots, v \rangle$ of P is a u,v-path of maximum reliability, so $\delta(u,v) = r(P')$. Because P is a u,w-path of maximum reliability, it holds that $\delta(u,w) = r(P) = r(P') \cdot r(v,w) = \delta(u,v) \cdot r(v,w)$, as desired. □

In the next two sections we concentrate on calculating the function δ. First we *Dijkstra's algorithm* to compute $\delta(s,\cdot)$ in case one source node $s \in V$ is given. After that, we study the *Floyd-Warshall*-algorithm to compute $\delta(u,v)$ for all pairs $(u,v) \in V \times V$. Finally we will see how knowing the δ-values helps with solving Problem 1 (i) and (ii).

2.2.1 Dijkstra's algorithm for maximum reliabilities

We give the multiplicative version of the Dijkstra algorithm, an algorithm for computing $\delta(s,\cdot)$ in case one source node $s \in V$ is given. This will help solving Problem 1 (i). To compute the distances $\delta(s,v)$, for all $v \in V$, we keep track of *tentative* distances d. We begin with $d(s) = 1$ and $d(v) = 0$ for $v \in V \setminus \{s\}$. The function d will be our approximation of $\delta(s,\cdot)$. We will refine the value of d, until eventually $\delta(s,v) = d(v)$ for every $v \in V$. To do this, we will *relax* edges $(u,v) \in E$:

$$\text{Relax}(u,v): \ \textbf{if } d(v) < d(u) \cdot r(u,v) \ \textbf{then } d(v) = d(u) \cdot r(u,v).$$

Edge relaxations (by definition) can only increase (or keep constant) the d-values. Furthermore, if we relax edges then we always have $d(v) \leq \delta(s,v)$ for all $v \in V$. This we will prove in a lemma.

Lemma 2.2.4. *For every $v \in V$, we always have $d(v) \leq \delta(s, v)$, if only edge relaxations are applied.*

Proof. We use induction on the number of edge relaxations. If no edge relaxations are applied, the claim holds since $d(s) = 1 = \delta(s, s)$ and $d(v) = 0 \leq \delta(s, v)$ for $v \in V \setminus \{s\}$. Now, assume that the claim holds before an edge $e = (u, v)$ is relaxed. Relaxing the edge (u, v) only possibly affects $d(v)$. If $d(v)$ is modified, then we have after the relaxation

$$d(v) = d(u) \cdot r(u, v) \leq \delta(s, u) \cdot r(u, v) \leq \delta(s, v),$$

where the second equality follows from the triangle inequality (Lemma 2.2.1). □

Therefore $d(v)$ can increase while relaxing edges but it will never be bigger than the distance $\delta(s, v)$. Also $d(v) = \delta(s, v) = 0$ for all nodes $v \in V$ that are not reachable from s.

Lemma 2.2.5. *Let $P = \langle s, \ldots, u, v \rangle$ be a s, v-path of maximum reliability. Suppose $d(u) = \delta(s, u)$ before the relaxation of edge (u, v). Then $d(v) = \delta(s, v)$ after the relaxation of the edge (u, v).*

Proof. After the relaxation we have $d(v) = d(u) \cdot r(u, v) = \delta(s, u) \cdot r(u, v) = \delta(s, v)$, where the last equality holds because of Lemma 2.2.3. □

Now we are ready to give Dijkstra's algorithm and prove correctness of the algorithm.

Input: Directed graph $G = (V, E)$, reliability function $r : e \to (0, 1]$,
source vertex s.
Output: For each $v \in V$, the value $\delta(s, v)$.

Initialize: $d(s) = 1$ and $d(v) = 0$ for every $v \in V \setminus \{s\}$
$W = V$
while $W \neq \emptyset$ **do**
  Choose a vertex $u \in W$ with $d(u)$ maximum.
  foreach $(u, v) \in E$ **do** *Relax*(u, v).
  Remove u from W.
end
return d

Algorithm 2: Adaptation of Dijkstra's algorithm to help solving problem i.

We will prove that this algorithm correctly computes the maximum reliabilities.

Theorem 2.2.6 (Adapted Dijkstra)**.** *Algorithm 2 correctly computes the maximum reliabilities in time $\mathcal{O}(n^2)$ or, when Fibonacci heaps are used in time $\mathcal{O}(m + n \log n)$.*

Proof. First we prove that when a vertex u is removed from W, it holds that $d(u) = \delta(s, u)$. Suppose this claim does not hold. Consider the first iteration in which a vertex u is removed from W, but $d(u) < \delta(s, u)$. Then u must be reachable from s, since $\delta(s, u) > d(u) \geq 0$. Let P be a maximum-reliability s, u-path. Define

$$N := \{v \in V \ : \ d(v) = \delta(s, v)\}.$$

If we traverse P from s to u, there must be an edge (x, y) on P with $x \in N$ and $y \notin N$ because $s \in N$ and $u \notin N$. Let (x, y) be the first such edge on P. Then it holds that

$$d(x) = \delta(s, x) \geq \delta(s, u) > d(u),$$

where the second relation holds because edge reliabilities are in $[0, 1]$. Hence, vertex x was removed before u from W. By our choice of u, it holds that $d(x) = \delta(s, x)$ at the moment that x is removed from W. But then it holds (by Lemma 2.2.5) that $d(y) = \delta(s, y)$ after the relaxation of edge (x, y), in contradiction with the assumption that $y \notin N$. Therefore the claim holds.

It follows that the algorithm computes the correct distances. The algorithm also clearly terminates (it relaxes each edge exactly once and removes all nodes from W), therefore the algorithm is correct. The algorithm takes time $\mathcal{O}(n^2)$, since it consists of n iterations that each take $\mathcal{O}(n)$ time. However, the running time of this algorithm can be improved by using *Fibonacci heaps*. The interested reader can read more about Fibonacci heaps in [Schr13] or [CLR01]. With Fibonacci heaps we can do n insert operations (initialization), n delete-min operations (remove vertices with minimum d-values) and m decrease priority operations (relaxing edges) in time $\mathcal{O}(m + n \log n)$. Therefore by using Fibonacci heaps the algorithm runs in time $\mathcal{O}(m + n \log n)$. □

The CPB-graph is a complete graph and therefore $\mathcal{O}(m + n \log n) = \mathcal{O}(n^2)$. Implementing Fibonacci heaps takes time, and for this thesis the choice was made not to implement them.

2.2.2 Floyd-Warshall algorithm for maximum reliabilities

In this section we give a version of the *Floyd-Warshall*-algorithm to compute $\delta(u, v)$ for all pairs $(u, v) \in V \times V$. That will help solving Problem 1 (*ii*). We identify the vertices in V with the set $\{1, \ldots, n\}$. Consider a simple u, v-path $P = \langle u = u_1, \ldots, u_l = v \rangle$. We call the vertices $u_2, \ldots, u_{l-1}$ the *interior vertices* of P. If $l \leq 2$, then P does not have interior vertices. A u, v-path P with interior vertices contained in the set $\{1, \ldots, k\}$ is called a (u, v, k)-path. We define:

$$\delta_k(u, v) := \begin{cases} \sup\{r(P) \,:\, P \text{ is an } (u, v, k)\text{-path})\} & \text{if there exists at least one } (u, v, k)\text{-path} \\ 1 & \text{if } u = v \\ 0 & \text{otherwise.} \end{cases}$$

This is the maximum reliability of a (u, v, k)-path. With this definition we have $\delta(u, v) = \delta_n(u, v)$. Therefore we need to compute $\delta_n(u, v)$ for every $u, v \in V$. Consider the following algorithm.

Input: directed graph $G = (V, E)$, reliability function $r : e \to (0, 1]$.
Output: For each pair $(u, v) \in V \times V$, the value $\delta(u, v)$.

Initialize: **foreach** $(u, v) \in V \times V$ **do** $d(u, v) := \begin{cases} 1 & \text{if } u = v \\ r(u, v) & \text{if } (u, v) \in E \\ 0 & \text{otherwise.} \end{cases}$

for $k = 1 \ldots n$ **do**
  foreach $(u, v) \in V \times V$ **do**
    if $d(u, v) < d(u, k) \cdot d(k, v)$ **then** $d(u, v) = d(u, k) \cdot d(k, v)$
  end
end
return d

Algorithm 3: Adaptation of the Floyd-Warshall algorithm to help solving problem ii.

We will prove that this algorithm correctly computes the maximum reliabilities.

Theorem 2.2.7 (Adapted Floyd-Warshall)**.** *Algorithm* 3 *correctly computes the maximum reliabilities in time* $\Theta(n^3)$.

Proof. The running time follows directly from the steps in the algorithm; the algorithm consists of a for-loop of size n^2 within a for-loop of size n. It therefore suffices to prove correctness. Suppose we are able to compute $\delta_{k-1}(u,v)$ for all $u,v \in V$. Consider a maximum reliability (u,v,k)-path $P = \langle u = u_1, \ldots, u_l = v \rangle$. Note that we can assume without loss of generality that P is simple, as observed in Lemma 2.0.6. All interior vertices of P belong to the set $\{1,\ldots,k\}$ by definition. Now there are two possible cases. Either k is not an interior vertex of P, or k is an interior vertex of P.

(i) If k is not an interior vertex of P then all interior vertices of P are in the set $(1,\ldots,k-1)$, i.e. P is a maximum reliability $(u,v,k-1)$-path and therefore $\delta_k(u,v) = \delta_{k-1}(u,v)$.

(ii) If k is an interior vertex of P, then we can write $P = \langle u,\ldots,k,\ldots,v\rangle$. We now decompose P into two paths $P_1 = \langle u,\ldots,k\rangle$ and $P_2 = \langle k,\ldots,v\rangle$. We observe that P_1 and P_2 are $(u,v,k-1)$ paths because P is simple. Furthermore, P_1 and P_2 are *maximum reliability* $(u,v,k-1)$-paths, because subpaths of maximum reliability paths are maximum reliability paths by Lemma 2.2.2. Therefore $\delta_k(u,v) = \delta_{k-1}(u,k)\cdot\delta_{k-1}(k,v)$.

Now, if we set:

$$\delta_0(u,v) := \begin{cases} 1 & \text{if } u = v \\ r(u,v) & \text{if } (u,v) \in E \\ 0 & \text{otherwise.} \end{cases}$$

and

$$\delta_k(u,v) = \max\{\delta_{k-1}(u,v), \delta_{k-1}(u,k)\cdot\delta_{k-1}(k,v)\} \quad \text{if } k \geq 1,$$

we simply compute the $\delta_k(u,v)$ in a bottum-up manner. Algorithm 3 exactly does this, with as final output function $d = \delta_n = \delta$. □

2.2.3 Computing and counting the maximum reliability paths

If we want to calculate the maximum reliability distances from a single fixed source $s \in V$, we can compute with Dijkstra's algorithm the values $\delta(s,v)$ for all $v \in V$ in time $\mathcal{O}(m + n\log n)$. If we are interested in all maximum reliability distances, and do not want to fix one source-node, we can compute with Floyd-Warshall's algorithm the values $\delta(u,v)$ for every $u,v \in V \times V$ in time $\Theta(n^3)$.

Now, fix a vertex $s \in V$. We will see that we can efficiently obtain a maximum reliability path from s to every other vertex $v \in V$ with $\delta(s,v) \in (0,1]$. The following definition will be useful.

Definition 2.2.2 (Maximum reliability path graph)**.** Let $G = (V,E)$ be a graph with edge reliabilities $r := E \to (0,1]$, and let $s \in V$ be a source node. Let

$$V' := \{v \in V \mid \delta(s,v) \in (0,1]\} \subseteq V,$$

be the set of vertices reachable from s. Let $E' \subseteq E$ be the set of edges that are *tight* (cf. Definition 2.2.1) with respect to $\delta(s,\cdot)$, i.e.

$$\begin{aligned} E' &:= \{(v,w) \in E \ : \ (v,w) \text{ tight with respect to } \delta(s,\cdot)\} \\ &= \{(v,w) \in E \ : \ \delta(s,w) = \delta(s,v)\cdot r(v,w)\} \subseteq E. \end{aligned}$$

We define $G' := (V',E')$ to be the *maximum reliability path graph* of G with respect to the source node s.

Note that G', *given* the distances $\delta(s,\cdot)$ in G, can be constructed in time $\mathcal{O}(|V| + |E|)$. The following lemma explains the name *maximum reliability path graph*: the graph G' consists *exactly* of all simple maximum reliability paths starting at some source node s in G.

Lemma 2.2.8. *Let $G = (V, E)$ be a graph with edge reliabilities $E \to (0, 1]$, $s \in V$ a source node, and $G' = (V', E')$ be the maximum reliability path graph of G with respect to the source node s. It holds that* every *simple path in G' from s to any vertex $v \in V'$ is a maximum reliability s, v-path in G and vice versa: every simple maximum reliability s, v-path in G is a simple s, v-path in G'.*

Proof. "$\Longrightarrow$". We use Lemma 2.2.3: every edge of a maximum reliability path from s to an arbitrary other vertex v is tight with respect to $\delta(s, \cdot)$. Therefore every vertex $v \in V'$ is reachable from $u \in G'$. Consider a s, v-path $P = \langle s = u_1, \ldots, u_k = v \rangle$ in G'. Then

$$r(P) = \prod_{i=1}^{k-1} r(u_i, u_{i+1}) = \prod_{i=1}^{k-1} \left(\frac{\delta(s, u_{i+1})}{\delta(s, u_i)} \right) = \frac{\delta(s, v)}{\delta(s, s)} = \delta(s, v).$$

Therefore, P is a maximum reliability path from s to v in G.

"$\Longleftarrow$". Conversely, let P be a simple maximum reliability path from s to v in G. This means that v is reachable from s, so $v \in V'$. Furthermore, by Lemma 2.2.3, all edges of P are tight. Hence, P is a path in G', as desired. □

Now we can find *one* simple s, v-path of maximum reliability in G by finding an arbitrary simple s, v-path in G' using (for example) depth-first search, see the Preliminaries (Chapter 1).

To find all (potentially exponentially many) simple maximum reliability s, v-paths we must find all simple paths from s to v in G'. This can not be done efficiently: it can take exponential time, as there may be exponenially many paths. By adapting depth-first search (see Chapter 5) one could obtain all s, v-paths, but not in polynomial time.

Example 2.2.1. Note that, if G is a complete graph with all edge reliabilities equal to 1, all edges are tight (with respect to $\delta(s, \cdot)$ for *any* $s \in V$) and hence G' is also a complete graph. Now we want to list all simple s, v-paths in G. We cannot expect a polynomial time algorithm. To see this, we count the number of s, v-paths in G'. Label the vertices $(1), \ldots, (n)$ such that $(1) = s$ and $(n) = v$. A simple $(1), (n)$-path visits some subset of the other $(n-2)$ nodes. Suppose the intermediary path visiting this subset of the other $(n-2)$ nodes is of length i. There are $\binom{n-2}{i}$ possible subsets of size i, and $i!$ possible orders for each of these. Therefore there are

$$\frac{(n-2)!}{(n-2-i)!}$$

possible subpaths of length i. It follows that there are

$$\sum_{i=0}^{n-2} \frac{(n-2)!}{(n-2-i)!} \geq (n-2)!$$

simple $(1), (n)$-paths in G', which grows (faster than) exponentially. We conclude that in this case we cannot list all simple paths in polynomial time.

Suppose we do not want to *list* the paths, but that we just want count the *number of* simple s, v-paths in G'. Note that G' may contain cycles. For example, consider a graph G in which all edge reliabilities are 1 and which contains a cycle that is reachable from s. Then all edges on this cycle are tight with respect to $\delta(s, \cdot)$, i.e. G' contains a cycle. However, *if* G' contains a cycle, this cycle has reliability 1.

Lemma 2.2.9. *Suppose G' contains a cycle, then all edges on this cycle will have reliability* 1*.*

Proof. Let $C = \langle v = v_1, \ldots, v_j = v \rangle$ be a cycle in G'. Let P be a s, v-path in G. Then P has maximum reliability and P concatenated with C is also a maximum reliability s, v-path (since it is a s, v-path contained in G'). But the reliability of this new path is $r(P) \cdot r(C)$. It follows that $r(P) = r(P) \cdot r(C)$ and hence $r(C) = 1$. We conclude that all edges on the cycle must have edge reliability 1. □

To count all simple maximum reliability paths in an efficient way, we assume that our graph G' does not contain cycles of edge reliability 1. This is, for example, the case if all edge reliabilities in G are in $(0, 1)$.

Suppose now that we want to compute the *number* of maximum reliability simple paths from s to v, without generating these paths explicitly. It is possible to calculate the number of simple paths from a fixed $s \in V'$ to each $v \in V'$ in linear time. To this end, we topologically sort $G' = (V', E')$, starting at s, which is possible since G' is a DAG. This means that in the topological sorted G', for each directed edge (x, y), it holds that x is before y in the ordering. To calculate the number of simple maximum reliability paths from s to v, we start at s in the topologically sorted G'. Then we scan vertices v in topological order (where the topological ordering starts at s), and keep track of how many paths in G' there are from s to this vertex v. In this way, we get our desired answer. This is summarized in the following algorithm:

Input: Topologically sorted *DAG* $G' = (V', E')$ w.r.t. source vertex s.
Output: The number of simple s, v-paths $N(v)$, for every $v \in V$.

Initialize: $N(s) = 1$, $N(x) = 0$ for all $x \in V' \setminus \{s\}$.
foreach $x \in V'$ in topological order **do**
    foreach child $y \in V'$ of x **do**
        $N(y) := N(y) + N(x)$
    end
end
return N

Algorithm 4: Algorithm to count the number of simple s, v-paths, for all $v \in V$ in a topologically sorted directed acyclic graph (*DAG*).

Note that we only have to pass each edge once, giving us an $\mathcal{O}(|V| + |E|)$ algorithm. It is also possible to calculate the number of u,v-paths, for *any* $u \in V$ and for *fixed* $v \in V$, in the maximum reliability graph rooted at s. To this end, we topologically sort $G' = (V', E')$, starting at s. To calculate the number of simple maximum reliability paths from u to v, we start at v in the topologically sorted G'. Then we scan vertices u backwards from v, and keep track of how many paths in G' there are from each vertex u to v. If we arrive at s, we get the number of paths from s to v.

Input: Topologically sorted *DAG* $G' = (V', E')$ w.r.t source vertex s, sink v.
Output: The number of simple u, v-paths $N(u)$, for every $u \in V$.

Initialize: $N(v) = 1$, $N(x) = 0$ for all $x \in V' \setminus \{x\}$.
foreach $x \in V'$ in *reverse* topological order **do**
    foreach child $y \in V'$ of x **do**
        $N(x) := N(y) + N(x)$
    end
end
return N

Algorithm 5: Algorithm to count the number of simple u, v-paths, for all $u \in V$ and a fixed $v \in V$, in a topologically sorted directed acyclic graph (*DAG*).

Sometimes we also want the number of simple maximum reliability paths from s to v passing *through* a given vertex w. We can do this by using Algorithm 5. Then, backtracking from v to w

in the topologically sorted DAG G' gives the number of simple paths from w to v. Backtracking from w to s gives the number of simple paths in G' from s to w. Multiplying those two numbers gives the number of simple maximum reliability paths from s to v passing through w (note that this is correct since G' is a directed *acyclic* graph). Hence, we solved Problem 1 (i) and (ii) in this section.

2.2.4 Definitions used for the additive shortest path problem

In this (short) section we will define the notions for maximum reliability paths from last sections also for shortest paths and we will prove similar results, to keep the thesis self-contained. Readers with knowledge of the shortest path problem can skip this section.

Let $G = (V, E)$ with an additive cost function $c : E \to \mathbb{R}_{\geq 0}$. As in the case of maximum reliability paths, we define a distance function $\delta_c : V \times V \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ as

$$\delta_c(u, v) = \begin{cases} \inf\{c(P) \,:\, P \text{ is a path from } u \text{ to } v\} & \text{if } v \text{ is reachable from } u \\ 0 & \text{if } u = v \\ \infty & \text{if } v \text{ is not reachable from } u. \end{cases}$$

If we write δ_c for the distance function with respect to an additive edge cost function c and δ_r for the distance function with respect to a multiplicative edge reliability function r, it follows from Lemmas 2.1.1 and 2.1.2 that

$$\delta_{\phi \circ c} = \phi \circ \delta_c \quad \text{and that} \quad \delta_{\phi^{-1} \circ r} = \phi^{-1} \circ \delta_r.$$

We define tight edges (similar to Definition 2.2.1) in the case of (additive) shortest paths.

Definition 2.2.3 (Tight edge (additive edge costs))**.** We call an edge $e = (v, w)$ *tight* with respect to the distance function $\delta(s, \cdot)$ if $\delta(s, w) = \delta(s, v) + c(v, w)$.

Definition 2.2.4 (Shortest path graph)**.** Let $G = (V, E)$ be a graph with edge costs $c := E \to \mathbb{R}_{\geq 0}$, and let $s \in V$ be a source node. Let

$$V' := \{v \in V \mid \delta(s, v) \in \mathbb{R}_{\geq 0}\} \subseteq V,$$

be the set of vertices reachable from s, and

$$\begin{aligned} E' &:= \{(v, w) \in E \;:\; (v, w) \text{ tight with respect to } \delta(s, \cdot)\} \\ &= \{(v, w) \in E \;:\; \delta(s, w) = \delta(s, v) + c(v, w)\} \subseteq E. \end{aligned}$$

We define $G' := (V', E')$ to be the *shortest path graph* of G with respect to the source node s.

The shortest path graph G' consists *exactly* of all simple shortest paths starting at some source node s in G.

Lemma 2.2.10. *Let $G = (V, E)$ be a graph with edge costs $E \to \mathbb{R}_{\geq 0}$, $s \in V$ a source node, and $G' = (V', E')$ be the shortest path graph of G with respect to the source node s. It holds that*

(i) every *simple path in G' from s to any vertex $v \in V'$ is a shortest s, v-path in G and vice versa: every simple shortest s, v-path in G is a simple s, v-path in G'.*

(ii) if G' contains a cycle, then all edges on this cycle will have cost 0.

Proof. Use the reliability function $r := \phi \circ c$. Tight edges with respect to r are exactly tight edges with respect to c and vice versa. Furthermore, shortest paths with respect to c are maximum reliability paths with respect to r and vice versa. For G with the reliability-function r, Lemma 2.2.8 (for proving i) and Lemma 2.2.9 (for proving ii, where we note that $\phi(0) = 1$)) hold. Now apply ϕ^{-1} again to get the desired result. □

We have seen that the shortest path problem and the maximum reliability path problem are essentially the same problems: an instance of the shortest path problem gives us immediately an instance of the maximum reliability path problem and vice versa, with corresponding solutions via the function ϕ of Section 2.1. In both cases we can define a graph of tight edges, which consists exactly of all the solution paths starting at some source vertex s. We can also *directly* adapt shortest path algorithms to compute maximum reliability paths, and vice versa.

Hereby we end the discussion of basic shortest path (resp. maximum reliability path) algorithms. In the next section we will use these shortest path algorithms to compute a 'centrality measure' for vertices in the network: *betweenness centrality*.

2.3 Betweenness centrality

This section is about *betweenness centrality*. The betweenness centrality measures 'centrality' of a node in a network. The betweenness centrality will be higher if a vertex lies *between* the endpoints (as an intermediary vertex) on a lot of maximum reliability paths (in the case of reliability paths) or shortest paths (in the case of paths with additive edge costs). We will use the betweenness centrality to identify the most important conduit countries in the CPB-network.

Remark 2.3.1 (Motivation and comparison to other centrality measures)**.** *Betweenness centrality* is not the only available tool to measure 'centrality' of a vertex in a network. There are other centrality measures too. See for example [Fre78] and [New01]. We will *informally* discuss other two centrality measures: *closeness centrality* and *degree centrality*. Then we will argue why in this thesis, *betweenness centrality* is chosen as centrality measure.

1. The *closeness centrality* (see [New01]) of a vertex is higher if the vertex is situated *close* to other vertices, i.e. if the average distance to other vertices is smaller (in the case of additive shortest paths). Applied to our network: if the reliability of maximum reliability paths from one country to other countries is higher, then this country has a higher closeness centrality. Closeness centrality does thus not give information about the role of a country as a 'conduit' country, but more as an 'endpoint' of a path. Therefore we will not use this measure.

2. The (weighted) *degree centrality* is higher if a node has many edges of low cost to other nodes. Hence, it only measures the 'local structure' directly around the node and it does not take the whole network into account.

Both closeness and degree centrality are not aimed at identifying conduit (intermediary) countries in the network. However, *betweenness centrality* is very useful for this purpose, since it measures how often, on average, a country appears as an intermediary country (*between* the endpoints) on a 'most profitable' tax route (a maximum reliability path).

Remark 2.3.2. Throughout the whole section, let $G = (V, E)$ be a weighted directed graph, with multiplicative edge reliabilities $r : E \to (0, 1]$ or additive edge costs $c : E \to \mathbb{R}_{\geq 0}$, so that there are no cycles of reliability 1 (resp. cost 0) in G (then we can count maximum reliability paths (resp. shortest paths) efficiently using the methods from Algorithms 4 and 5). We will use the term 'shortest paths' throughtout the whole section, but one may replace it by 'maximum reliability paths'.

Definition 2.3.1 (Betweenness centrality, unweighted and weighted version)**.** We write σ_{st} for the number of shortest s, t-paths, for $s, t \in V$. Furthermore, we write $\sigma_{st}(u)$ for the number of shortest s, t-paths, for $s, t \in V$, that pass *through* vertex $u \in V$, where $u \neq s \neq t$. The

(unweighted) *betweenness centrality* of $u \in V$ in G is defined as

$$B(u) = \sum_{\substack{s,t,u \in V \\ s \neq u \neq t}} \frac{\sigma_{st}(u)}{\sigma_{st}}.$$

If we write $B_{st}(u) := \sigma_{st}(u)/\sigma_{st}$, then it holds that

$$B(u) = \sum_{\substack{s,t,u \in V \\ s \neq u \neq t}} B_{s,t}(u).$$

Next we define *weighted betweenness centrality*. Suppose that we are given weights $w_{s,t} > 0$, for all $s, t \in V$, $s \neq t$. We define the *weighted betweenness centrality* as:

$$B^W(u) = \sum_{\substack{s,t,u \in V \\ s \neq u \neq t}} w_{s,t} B_{s,t}(u).$$

In the next section we will see how to compute the (weighted) betweenness centrality efficiently, using a recursive formula found by Brandes (see [Bra01]).

2.3.1 Computing the betweenness centrality efficiently

This section deals with the computation of the betweenness values described in the previous section. We first provide a naive, intuitive way of computing the betweenness and analyze its running time. Subsequently we prove an auxiliary lemma to reduce the running time and the storage space needed for computing the (unweighted) betweenness centrality, due to Brandes [Bra01]. Finally we show that this method can also be used to compute the weighted betweenness centrality.

Remark 2.3.3 (Naive approach to compute the betweenness)**.** It is possible to compute the betweenness for all nodes in a graph G in $\mathcal{O}(n^3)$, using $\mathcal{O}(n^2)$ storage space.

Proof. By using Algorithm 4 in directed acyclic graphs, we are able to compute the number of maximum reliability paths from one source to all $v \in V$ in a graph in $\mathcal{O}(m + n \log n)$, the running time of Dijkstra's algorithm. Note that, if $\delta(s,t) = \delta(s,v) + \delta(v,t)$,[2] then every simple shortest path from s to v can be extended to a simple shortest path from s to t through v, and $\sigma_{st}(v) = \sigma_{sv} \cdot \sigma_{vt}$ (this holds since G contains no cycles of cost 0). If we do this for all $n = |V|$ different source nodes, we achieve a running time of $\mathcal{O}(nm + n^2 \log n)$. After that, we need to compute the sum $\sum_{s \neq v \neq t} \sigma_{st}(v)/\sigma_{st}$. This can be done in $\mathcal{O}(n^2)$ running time for *one* vertex v. Therefore, the running time of the naive betweenness computation algorithm is $\mathcal{O}(n^3)$, for computing the betweenness centrality of *all* vertices $v \in V$.

For each source node $s \in V$, we store the number σ_{sv} of shortest path from s to v (for all $v \in V$). We conclude that the naive algorithm uses $\mathcal{O}(n^2)$ storage space. □

We will improve the algorithm for computing the betweenness. First we define, for $s, t \in V$,

$$B_{s,\bullet}(u) = \sum_{t \in V} B_{s,t}(u) \quad \text{and} \quad B^W_{s,\bullet}(u) = \sum_{t \in V} w_{s,t} B_{s,t}(u).$$

Furthermore, we define the *predecessors* along shortest paths of a vertex $v \in V$.

Definition 2.3.2 (Predecessors along shortest paths)**.** For a vertex $v \in V$ we define

$$P_s(v) := \{u \in V \ : \ (u,v) \in E \text{ is a tight edge with respect to } \delta(s,\cdot)\}, \tag{2.1}$$

as the set of *predecessors* of v along shortest paths from $s \in V$.

[2] $\delta(s,t) = \delta(s,v) \cdot \delta(v,t)$ in the case of maximum reliability paths.

Using a recursive relation (due to U. Brandes in [Bra01]) we will be able to speed up the calculation of the betweenness.

Theorem 2.3.1 (Brandes' recursive relation)**.** *It holds, for $s \in V$ and $u \in V$, that*

$$B_{s,\bullet}(u) = \sum_{v\,:\,u \in P_s(v)} \frac{\sigma_{su}}{\sigma_{sv}} \cdot (1 + B_{s,\bullet}(v)). \tag{2.2}$$

Proof. We observe that $B_{s,t}(u) > 0$ only if u lies (as an intermediary vertex) on at least one shortest path from s to t, and on any such path there is exactly *one* edge (u, v) with $u \in P_s(v)$. We extend the definition of betweenness to an edge, by denoting, for $e = (u, v) \in E$:

$$B_{s,t}(e) = B_{s,t}[(u, v)] = \frac{\sigma_{st}[(u, v)]}{\sigma_{st}},$$

where $\sigma_{st}[(u, v)]$ denotes the number of shortest s, t-paths passing through edge $e = (u, v)$. With the previous observations it holds that

$$B_{s,\bullet}(u) = \sum_{t \in V} \sum_{v\,:\,u \in P_s(v)} B_{s,t}[(u, v)] = \sum_{v\,:\,u \in P_s(v)} \sum_{t \in V} B_{s,t}[(u, v)]. \tag{2.3}$$

Let $v \neq t$ be a vertex with $u \in P_s(v)$. Note that $\sigma_{st}[(u, v)] = \sigma_{su} \cdot \sigma_{vt}$. Furthermore, it holds that $\sigma_{st}(v) = \sigma_{sv} \cdot \sigma_{vt}$. Combining both equalities gives that

$$\sigma_{st}[(u, v)] = \frac{\sigma_{su}}{\sigma_{sv}} \cdot \sigma_{st}(v),$$

if $v \neq t$ is a vertex with $u \in P_s(v)$. It follows that

$$B_{s,t}[(u, v)] = \begin{cases} \frac{\sigma_{su}}{\sigma_{sv}} \cdot B_{s,t}(v) & \text{if } t \neq v \\ \frac{\sigma_{su}}{\sigma_{sv}} & \text{if } t = v. \end{cases} \tag{2.4}$$

We insert this equation in (2.3) to get

$$\begin{aligned} B_{s,\bullet}(u) &= \sum_{v\,:\,u \in P_s(v)} \sum_{t \in V} B_{s,t}[(u, v)] = \sum_{v\,:\,u \in P_s(v)} \left(\frac{\sigma_{su}}{\sigma_{sv}} + \sum_{t \in V \setminus \{v\}} \frac{\sigma_{su}}{\sigma_{sv}} \cdot B_{s,t}(v) \right) \\ &= \sum_{v\,:\,u \in P_s(v)} \frac{\sigma_{su}}{\sigma_{sv}} \cdot (1 + B_{s,\bullet}(v)), \end{aligned} \tag{2.5}$$

which is the desired equality. □

It is not hard to prove that we can prove a similar recursive relation to compute the weighted betweenness.

Lemma 2.3.2 (Recursive relation to compute the weighted betweenness centrality)**.** *It holds, for $s \in V$ and $v \in V$, that*

$$B^W_{s,\bullet}(u) = \sum_{v\,:\,u \in P_s(v)} \frac{w_{s,v} \cdot \sigma_{su}}{\sigma_{sv}} \cdot \left(1 + \frac{B_{s,\bullet}(v)}{w_{s,v}} \right). \tag{2.6}$$

Proof. Analogous to the proof of theorem 2.3.1, but we replace equation (2.3) with

$$B^W_{s,\bullet}(u) = \sum_{t \in V} \sum_{v\,:\,u \in P_s(v)} w_{s,t} B_{s,t}[(u, v)] = \sum_{v\,:\,u \in P_s(v)} \sum_{t \in V} w_{s,t} B_{s,t}[(u, v)],$$

equation (2.4) with

$$w_{s,t}B_{s,t}[(u,v)] = \begin{cases} w_{s,t} \cdot \frac{\sigma_{su}}{\sigma_{sv}} \cdot B_{s,t}(v) & \text{if } t \neq v \\ w_{s,v} \cdot \frac{\sigma_{su}}{\sigma_{sv}} & \text{if } t = v, \end{cases} \tag{2.7}$$

and finally we replace equation (2.5) with

$$\begin{aligned} B^W_{s,\bullet}(u) &= \sum_{v\,:\,u\in P_s(v)} \sum_{t\in V} w_{s,t}B_{s,t}[(u,v)] = \sum_{v\,:\,u\in P_s(v)} \left(\frac{w_{s,v}\cdot\sigma_{su}}{\sigma_{sv}} + \sum_{t\in V\setminus\{v\}} \frac{w_{s,t}\cdot\sigma_{su}}{\sigma_{sv}} \cdot B_{s,t}(v) \right) \\ &= \sum_{v\,:\,u\in P_s(v)} \frac{w_{s,v}\cdot\sigma_{su}}{\sigma_{sv}} \cdot \left(1 + \frac{B^W_{s,\bullet}(v)}{w_{s,v}}\right), \end{aligned} \tag{2.8}$$

to complete the proof. □

The following algorithm now computes the (weighted) betweenness sums $B^W_{s,\bullet}(u)$ (where $s \neq u$), using the above recursive relation.

Input: Top. sorted shortest path *DAG* $G = (V, E)$ w.r.t source vertex s.
Output: For each $u \in V$, the value $B^W_{s,\bullet}(u)$.

Initialize: $\sigma_s[s] = 1$, $\sigma_s[v] = 0$ for all $v \in V \setminus \{s\}$, $B^W_{s,\bullet}[v] = 0$ for all $v \in V$,
empty list $P[v]$ for all $v \in V$.
foreach $x \in V$ in topological order **do**
  foreach child $y \in V$ of x **do**
    $\sigma_s[y] := \sigma_s[y] + \sigma_s[x]$
    append x to $P[y]$
  end
end
foreach $v \in V$ in reverse topological order **do**
  foreach $u \in V$ in $P[v]$ **do if** $s \neq u$:
    $B^W_{s,\bullet}[u] := B^W_{s,\bullet}[u] + \frac{w_{s,v}\cdot\sigma_s[u]}{\sigma_s[v]} \cdot \left(1 + \frac{B^W_{s,\bullet}[v]}{w_{s,v}}\right)$
  end
end
return $B^W_{s,\bullet}$

Algorithm 6: Algorithm to efficiently compute $B^W_{s,\bullet}(u)$ for each $u \in V$ in a topologically sorted directed acyclic maximum reliability path graph (*DAG*). By setting all weights $w_{s,v}$ to 1, we efficiently compute $B_{s,\bullet}(u)$ for computing the *unweighted* betweenness centrality.

By applying the algorithm to all $s \in V$ and summing the outcomes, we easily compute $B^W(u)$, or $B(u)$ (by setting all weights to 1) for all $u \in V$. This is summarized in the following algorithm.

If $A[\,]$ and $B[\,]$ are two same-sized arrays (say, of size n), we write $A \oplus B$ for the array that has as i-th entry the sum of the i-th entries of A and B, i.e. $(A \oplus B)[i] = A[i] + B[i]$, $i = 1, \ldots, n$. Computing $A \oplus B$ can be done in time $\mathcal{O}(n)$.

Input: Graph $G = (V, E)$ with edge reliabilities $r : E \to (0, 1)$ or costs $c : E \to \mathbb{R}_{>0}$.
Output: For each $v \in V$, the value $B^W(v)$.

Initialize: array $B^W[\,]$ consisting of $|V|$ zeroes,
foreach $s \in V$ **do**
    Compute topologically sorted shortest path *DAG* G' rooted at s.
    Apply Algorithm 6 to compute $B^W_{s,\bullet}[\,]$.
    $B^W := B^W \oplus B^W_{s,\bullet}$.
end
return B^W.

Algorithm 7: Algorithm to efficiently compute $B^W(u)$ for all $u \in V$.

Theorem 2.3.3 (Computing the betweenness efficiently)**.** *It is possible to compute the betweenness for all nodes in a directed weigthed graph $G = (V, E)$ in $\mathcal{O}(nm + n^2 \log n)$, using $\mathcal{O}(n + m)$ storage space.*

Proof. By using the path-count algorithm in directed acyclic graphs, we are able to compute the number of shortest paths from one source in a graph in $\mathcal{O}(m + n \log n)$, the running time of Dijkstra's algorithm. If we do this for all $n = |V|$ different source nodes, we achieve a running time of $\mathcal{O}(nm + n^2 \log n)$. After that, we need to compute the sum $\sum_{s \neq v \neq t} \sigma_{st}(v)/\sigma_{st}$. This can be done in $\mathcal{O}(n(n + m))$ running time, by applying the above algorithm for each source node s. Therefore, the total running time of the betweenness computation algorithm is bounded by $\mathcal{O}(nm + n^2 \log n)$.

We store, for a node s and for all nodes $u \in V$ the number of shortest paths σ_{su} between them, therefore we now only need $\mathcal{O}(n)$ storage space to store the numbers of the paths. The algorithm to compute $B^W_{s,\bullet}(u)$ therefore requires at most $\mathcal{O}(n + m)$ storage space. Running the algorithm $n - 1$ times (for each $s \in V \setminus \{u\}$) does not increase the storage space, as we can delete all information after one run, only storing and adding the $B^W_{s,\bullet}(u)$-values. We conclude that the algorithm uses $\mathcal{O}(n + m)$ storage space. □

2.3.2 Edge flows and edge betweenness centrality

Suppose an amount of flow is shipped from s to t, which we denote by $w_{s,t}$. We assume that the flow is equally distributed along all shortest paths, so the flow along each shortest path P from s to t is

$$f^P_{st} = \frac{w_{s,t}}{\sigma_{st}}.$$

Now, let $\mathcal{P}_{st}$ be the set of all shortest simple paths from s to t. Then we define the s, t-flow of an edge $e = (u, v)$ as the total flow shipped from s to t that passes through this edge:

$$f_{st}(e) = \sum_{\substack{P \in \mathcal{P}_{st}: \\ e \in P}} f_P = \sum_{\substack{P \in \mathcal{P}_{st}: \\ e \in P}} \frac{w_{s,t}}{\sigma_{st}} = \frac{w_{s,t} \cdot \sigma_{st}[(u, v)]}{\sigma_{st}}.$$

where $\sigma_{st}[(u, v)]$ denotes the number of shortest s, t-paths passing through edge $e = (u, v)$, just as in the proof of Theorem 2.3.1.

Definition 2.3.3 (Total edge flow)**.** We define the total flow $f(e)$ passing through an edge $e = (u, v)$ as the sum of all the s, t-flows of e, over all vertices $s, t \in V$ with $s \neq t$. Formally,

$$f(e) = f[(u, v)] = \sum_{\substack{s,t \in V \\ s \neq t}} f_{st}[(u, v)] = \sum_{\substack{s,t \in V \\ s \neq t}} \frac{w_{st} \cdot \sigma_{st}[(u, v)]}{\sigma_{st}}.$$

Note that it is possible to compute the total edge flows efficiently. If we define, for an edge $e = (u, v)$,

$$B_{s,t}[(u,v)] := \frac{\sigma_{st}[(u,v)]}{\sigma_{st}} \quad \text{and} \quad B^W_{s,\bullet}[(u,v)] := \sum_{t \in V} w_{s,t} B_{s,t}((u,v)),$$

then we have, using equations (2.7) and (2.8), that

$$B^W_{s,\bullet}[(u,v)] = \frac{w_{s,v} \cdot \sigma_{su}}{\sigma_{sv}} \cdot \left(1 + \frac{B^W_{s,\bullet}(v)}{w_{s,v}}\right).$$

This equality allows us to calculate the edge flows very fast, while calculating the betweenness of the vertices.

Definition 2.3.4 ((Weighted) edge betweenness centrality)**.** Sometimes the total flow through an edge $e = (u, v)$ is also called the *(weighted) edge betweenness centrality*. If all s, t-flow weights are equal to 1, we call the total edge flow through edge (u, v) the *unweighted edge betweenness centrality* of edge (u, v).

It is possible to extend the definition of edge flows to vertices.

Definition 2.3.5 (Vertex flow)**.** The s, t-flow of a *vertex* $u \in V \setminus \{s, t\}$ is

$$f_{st}(u) = \sum_{\substack{e=(u,v) \in E: \\ v \in V}} f_{st}(e) = \sum_{\substack{e=(w,u) \in E: \\ w \in V}} f_{st}(e),$$

where the last equality follows since all shortest paths that pass through u have exactly one edge with head u and one edge with tail u, or more explicitely:

$$\sum_{e=(u,v) \in E} \sum_{\substack{P \in \mathcal{P}_{st}: \\ e \in P}} f_P = \sum_{\substack{P \in \mathcal{P}_{st}: \\ u \in P}} f_P = \sum_{e=(w,u) \in E} \sum_{\substack{P \in \mathcal{P}_{st}: \\ e \in P}} f_P.$$

Therefore, the flow conservation law (see [Schä13]) is observed. Furthermore, we observe the equality

$$f_{st}(u) = \sum_{e=(u,v) \in E} \sum_{\substack{P \in \mathcal{P}_{st}: \\ e \in P}} f^P_{st} = \sigma_{st}(u) \cdot f^P_{st} = w_{st} \frac{\sigma_{st}(u)}{\sigma_{st}} = w_{s,t} B_{s,t}(u),$$

Therefore the total edge flow that passes *through* a vertex u (as an inner vertex[3] on shortest paths) equals the *weighted betweenness* centrality of u.

In this section we defined betweenness centrality and edge flows, and we have seen how to calculate them efficiently. We will apply these concepts on our CPB-network.

2.4 Maximum reliability paths in the network of countries

In this section we adapt the developed theory to our CPB-network and give some more details about the network. Let $G = (V, E)$ be the complete directed graph where V consists of the given 108 countries. We want to compute the minimum tax (as a fraction between 0 and 1) that a company is required to pay when sending money from a country $u \in V$ to country $v \in V$, and we want to do this for all pairs $u, v \in V \times V$. Therefore we will need to compute maximum-reliability paths.

It seems to be a good idea to use Floyd-Warshall to compute the distances for all pairs at once. In our case we can *not* do this, and now we will explain why. Let $s \in V$ be a

[3] An inner vertex of a path is a vertex lying on the path that is not an endpoint of this path.

source node. If $e = (v_1, v_2)$ is an edge, the tax distance t_{v_1,v_2} depends on whether v_1, the tail of e, is the first vertex of our path.[4] Therefore each edge $e = (v_1, v_2)$ has two possible reliabilities. We write $r(v_1, v_2)$ as the reliability of an edge $e = (v_1, v_2)$ when v_1 is *not* the starting point of our desired path, i.e. $v_1 \neq s$. Furthermore, we denote with $r'(v_1, v_2)$ the reliability of edge $e = (v_1, v_2)$ when v_1 is the starting point (source node) of our desired path, i.e. $v_1 = s$. Therefore the reliability function $r_s : E \to (0, 1]$ depends on the source node. Given a source node $s \in V$, it holds that, for $e = (v_1, v_2) \in E$,

$$r_s(v_1, v_2) = \begin{cases} r(v_1, v_2) & \text{if } v_1 \neq s \\ r'(v_1, v_2) & \text{if } v_1 = s. \end{cases}$$

Hence, we have for each source node $s \in V$ a seperate graph G_s, which is a complete directed graph on all 108 countries with edge reliabilities $r_s : E \to (0, 1]$. On each graph G_s we will use Dijkstra's algorithm to compute the maximum reliability paths from s.

2.4.1 Justification for introducing a small penalty

When we compute strictly maximum reliability paths, we assume that for every step in the network after the first step, a small extra penalty of ε% tax is levied (i.e. all edge reliabilities are multiplied by $(1 - \varepsilon)$ except for the outgoing edges of s). We denote G_{s_ε} for the graph G with the new edge reliabilities. Then the graph of tight edges rooted at a source node $s \in V$ does not contain reliability 1 edges (since at least some ε% tax is payed), except for possibly the outgoing edges of the source node s. Hence, the graph of tight edges rooted at s does not contain cycles, and we can efficiently count all maximum reliability paths using Algorithm 4 or 5. In the next lemma, we write G for G_s and G_ε for G_{s_ε}, to simplify notation, since we assume that a source node $s \in V$ is fixed.

Lemma 2.4.1. *If $\varepsilon > 0$ is small enough, the maximum reliability s, t-paths in G_ε are exactly the maximum reliability s, t-paths in G that contain the smallest number of edges.*

Proof. Let P' be simple a path in G that is either not of maximum reliability, or it contains not a minimal number of edges. Let P be a simple maximum reliability path in G that contains a minimal number of edges.

(i) If P' is not a maximum reliability path in G, then it holds that[5]

$$r_\varepsilon(P) \geq (1 - \varepsilon)^{n-1} r(P) > r(P') \geq r_\varepsilon(P'), \tag{2.9}$$

for sufficiently small $\varepsilon > 0$, namely iff

$$(1 - \varepsilon)^{n-1} > \frac{r(P')}{r(P)}.$$

Note that we only have a finite number of simple paths in a graph G, so one can estimate the fraction $\frac{r(P')}{r(P)} \leq \alpha < 1$ for path P' in G that is not a strictly maximum reliability path.[6] So then P' is also not of maximum reliability in G_ε.

[4] This has to do with the so-called *credit* method for tax-relief that some countries apply. For information about 'tax-relief' methods and about how the tax-distances are constructed, see the CPB-report [RL14]. In this thesis the tax-rates provided by the CPB are simply used as given.

[5] We write r_ε for the reliability of a path in G_ε.

[6] Note that finding a particular second-shortest (in cost) simple path in an arbitrary graph is *NP*-hard, if we allow edges of cost 0. (See [LP97])

(ii) If P' is a maximum reliability path in G, but does not contain a minimal number of edges, then

$$r_\varepsilon(P) = (1-\varepsilon)^i r(P) > (1-\varepsilon)^j r(P') = r_\varepsilon(P'),$$

where i is the number of edges of P, j is the number of edges in P' and $i < j$ by assumption.

We see that only maximum reliability paths of minimal length[7] in G can possibly be maximum reliability paths in G_ε. Since all those paths (maximum reliability paths of minimal length in G) have the same reliability and G_ε contains at least one maximum reliability path, it follows that the maximum reliability paths in G_ε are exactly the maximum reliability paths in G that contain the smallest number of edges. □

As a consequence of this lemma it seems sensible to add a small penalty, and then compute shortest paths. Companies are not going to send money along 0-tax edges through conduit countries if it is possible to send the money through a fewer number of countries to their destination and pay the same tax.

Remark 2.4.1. The reliabilities that are provided by the CPB are rounded at 8 decimals. Therefore the smallest difference that can appear between two reliabilities is 10^{-8}. Therefore, in situation (i) of Lemma 2.4.1, it holds that

$$\frac{r(P')}{r(P)} < \frac{1-10^{-8}}{1} = 1-10^{-8},$$

so if $1-\varepsilon \geq 1-10^{-11} > \sqrt[107]{1-10^{-8}}$, i.e. $\varepsilon \leq 10^{-11}$, then ε is 'small enough' (see (2.9)) for our application, in the sense of Lemma 2.4.1.

Throughout this thesis, *unless otherwise mentioned*, we will assume that always a penalty of $\varepsilon < 10^{-11}$ tax is added to each step after the first step in the network, as described in this section. Then the graphs G_s (for each source node s) will not contain cycles of reliability 1, and we will be able to count maximum reliability paths efficiently.

2.4.2 Weights in the network of countries

Recall that we defined $w_{s,t}$ as the flow shipped from s to t. The CPB does not have empirical data about the flow between two countries, so we must think of sensible hypothetical weights to use. We assume that an amount proportional to the size of the economy of t (measured according to GDP, *gross domestic product*) leaves country t and is invested in countries s (where $s \neq t$) in proportion to the sizes of their economies. The total investments from country t to country s equal

$$\mathrm{GDP}_t \cdot \frac{\mathrm{GDP}_s}{\sum_{v \in V \setminus \{t\}} \mathrm{GDP}_v}.$$

We assume that all investments made in country s (the host country) are finally repatriated to country t. Therefore we get

$$w'_{s,t} := \mathrm{GDP}_t \cdot \frac{\mathrm{GDP}_s}{\sum_{v \in V \setminus \{t\}} \mathrm{GDP}_v}.$$

We divide all these weights by some constant, such that $\sum_{s,t \in V : s \neq t} w_{s,t} = 1$. This normalization is not necessary, but we do this normalization so that all results will finally be numbers between 0

[7] Here *length* means: the number of edges in a path P.

and 100 (to get nicer looking numbers for comparison). Let $\mathrm{GDP}_{\mathrm{Total}} := \sum_{u \in V} \mathrm{GDP}_u$. Then $w_{s,t} := w'_{s,t}/\mathrm{GDP}_{\mathrm{Total}}$ gives the desired weights, since

$$\sum_{\substack{s,t \in V \\ s \neq t}} w_{s,t} = \sum_{\substack{s,t \in V \\ s \neq t}} \frac{\mathrm{GDP}_t}{\mathrm{GDP}_{\mathrm{Total}}} \cdot \frac{\mathrm{GDP}_s}{\sum_{v \in V \setminus \{t\}} \mathrm{GDP}_v} = \sum_{t \in V} \sum_{\substack{s \in V \\ s \neq t}} \frac{\mathrm{GDP}_t}{\mathrm{GDP}_{\mathrm{Total}}} \cdot \frac{\mathrm{GDP}_s}{\mathrm{GDP}_{\mathrm{Total}} - \mathrm{GDP}_t}$$
$$= \sum_{t \in V} \frac{\mathrm{GDP}_t}{\mathrm{GDP}_{\mathrm{Total}}} \cdot \frac{\mathrm{GDP}_{\mathrm{Total}} - \mathrm{GDP}_t}{\mathrm{GDP}_{\mathrm{Total}} - \mathrm{GDP}_t} = \frac{\mathrm{GDP}_{\mathrm{Total}}}{\mathrm{GDP}_{\mathrm{Total}}} = 1.$$

When we talk about 'weighted' betweenness centrality or edge flows, we will use these weights, but *multiplied* by 100, i.e. $\sum_{s \neq t} w_{s,t} = 100$, to get nicer numbers, so that all numbers are between 0 and 100.[8]

2.4.3 Betweenness in the network of countries

At the beginning of Section 2.4 we observed that the network of countries consists of $|V| = 108$ graphs G_s (where each graph contains the edge reliabilities starting from one particular country $s \in V$). Therefore, to compute betweenness centrality in the CPB-network, we cannot simply compute the usual betweenness centrality in one graph. However, the (weighted) betweenness centrality is defined as

$$B^W(u) = \sum_{\substack{s,t \in V, \\ s \neq u \neq t}} w_{s,t} \frac{\sigma_{st}(u)}{\sigma_{st}},$$

where σ_{st} denotes the number of maximum reliability paths from s to t and $\sigma_{st}(u)$ denotes the number of maximum reliability s,t-paths that pass through vertex u. This definition still makes sense in our CPB-network: we just count paths from every source in a different graph G_s (that depends on the source $s \in V$). Note that Brandes' Algorithm can still be used in the CPB-network, since it computes the sums $B^W_{s,\bullet}$ starting from each source node $s \in V$, i.e. it computes each sum $B^W_{s,\bullet}(u) = w_{s,t}\sigma_{s,t}(u)/\sigma_{st}$ in G_s, the CPB-graph with source node s. The following remark summarizes the running time of computing betweenness centrality in the CPB-network.

Remark 2.4.2. Our CPB-network consists of $n = 108$ graphs, one graph for each source s, on each of which we use a single source shortest path algorithm. This gives a total running time of $\mathcal{O}(n \cdot (m + n \log n)) = \mathcal{O}(n^3)$, since the n graphs are complete.

By the same arguments as in the proof of Theorem 2.3.3, we now only need $\mathcal{O}(n + m)$ storage space.

When, in this thesis, we compute the *unweighted* betweenness centrality, we multiply it by a constant to be able to compare it with the *weighted* betweenness centrality. We now choose this constant. Note that for the weights $w_{s,t}$ it holds that $\sum_{s \neq t} w_{s,t} = 100$. If all weights are equal to some constant c, then it holds that

$$\sum_{\substack{s,t \in V, \\ s \neq t}} c = 108 \cdot 107 \cdot c = 100,$$

so $c = 1/(108 \cdot 1.07)$. This is the constant by which we multiply the *unweighted* betweenness centrality, to be able to compare the numbers with the *weighted* betweenness centrality.

[8] Note that a (weighted) betweenness centrality of 100 can never be achieved. In the computation of the (weighted) betweenness of v, only weights $w_{s,t}$ with $s \neq v \neq t$ are used for computing the weighted betweenness centrality, and $100 = \sum_{s \neq t} w_{s,t} > \sum_{s \neq t \neq v} w_{s,t}$, since all weights $w_{s,t}$ with $s \neq t$ are strictly positive.

2.5 Results

The results for the *weighted betweenness* centrality can be seen in the following table. Great Britain (GBR) has the highest weighted betweenness centrality of all countries in the network, and other countries have a significantly lower weighted betweenness centrality. The number 2, Luxembourg (LUX), has a betweenness centrality that is only (a bit more than) half as big as the weighted betweenness centrality of Great Britain. The Netherlands (NLD) ranks 5th in the weighted betweenness centrality measure: this seems to give evidence for the statement that the Netherlands is an attractive conduit country for multinationals. Some explanations for the fact that Great Britain is ranking very high:

(i) Great Britain has a *standard* dividend-tax of 0%. This means that entirely no tax needs to be payed in Britain on dividends *leaving* the country.
(ii) Great Britain is an EU-country, and companies can send dividends between EU-countries for *free*, *without* paying tax.
(iii) Great Britain has a high number of bilateral tax treaties with other countries: 51.

The Netherlands has an even *higher* number of bilateral tax treaties (74), and is also an EU-country, but has a standard dividend-tax of 15% (although with specific countries a lower percentage is often agreed in one of the bilateral tax treaties), which is higher than the 0%-percent dividend tax in Great Britain. For more (economical) explanations, see [RL14].

Position	Country u	$B^W(u)$
1	GBR	12.80779
2	LUX	6.96023
3	SGP	4.23889
4	EST	2.92012
5	NLD	2.63225
6	IRL	2.57268
7	HUN	2.11862
8	ESP	2.03112
9	SVK	2.00355
10	CYP	1.67547
11	MLT	1.48476
12	FRA	1.31831
13	FIN	1.22550
14	BRN	1.18916
15	MYS	1.04301

Position	Country u	$B(u)$
1	GBR	10.61932
2	NLD	6.27001
3	SGP	4.80919
4	CYP	4.60625
5	HUN	3.78398
6	ESP	3.51100
7	EST	3.38580
8	MLT	2.97467
9	LUX	2.70852
10	MYS	2.52015
11	SVK	2.50322
12	QAT	2.29019
13	ARE	2.11230
14	BRB	1.79871
15	HKG	1.71729

Table 2.1: The 15 countries with the highest *weighted*, respectively *unweighted* betweenness centrality values.

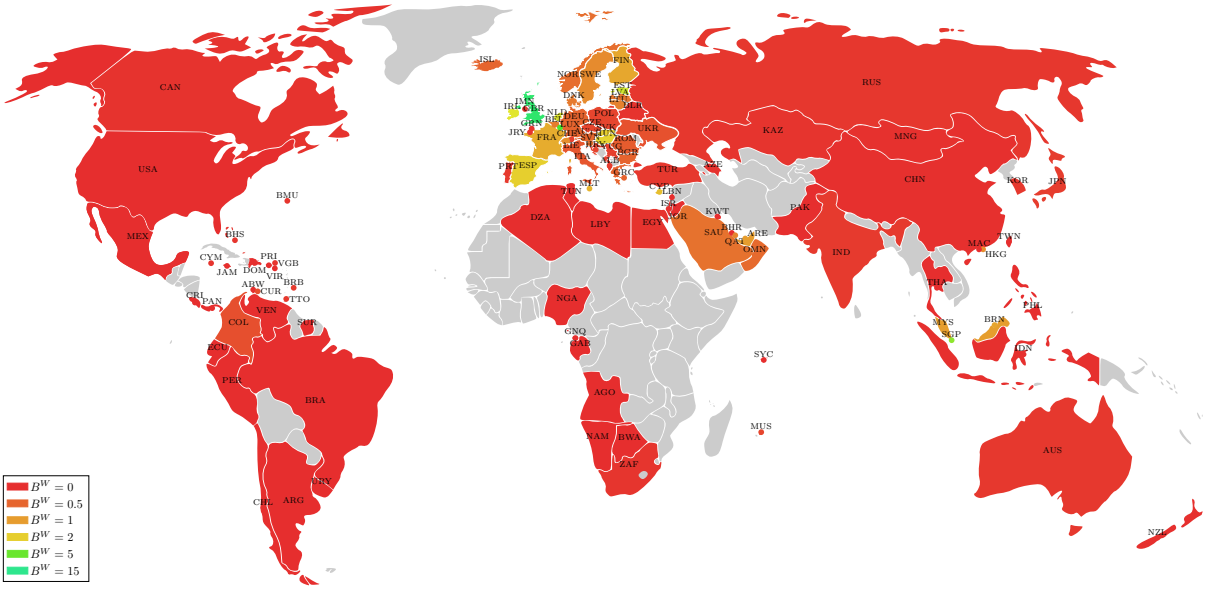


Figure 2.1: The countries that are more central in the network have a higher weighted betweenness centrality value B^W.

Now we examine the *unweighted* betweenness centrality. Here Great Britain also ranks first. The Netherlands (NLD) is second in this measure.

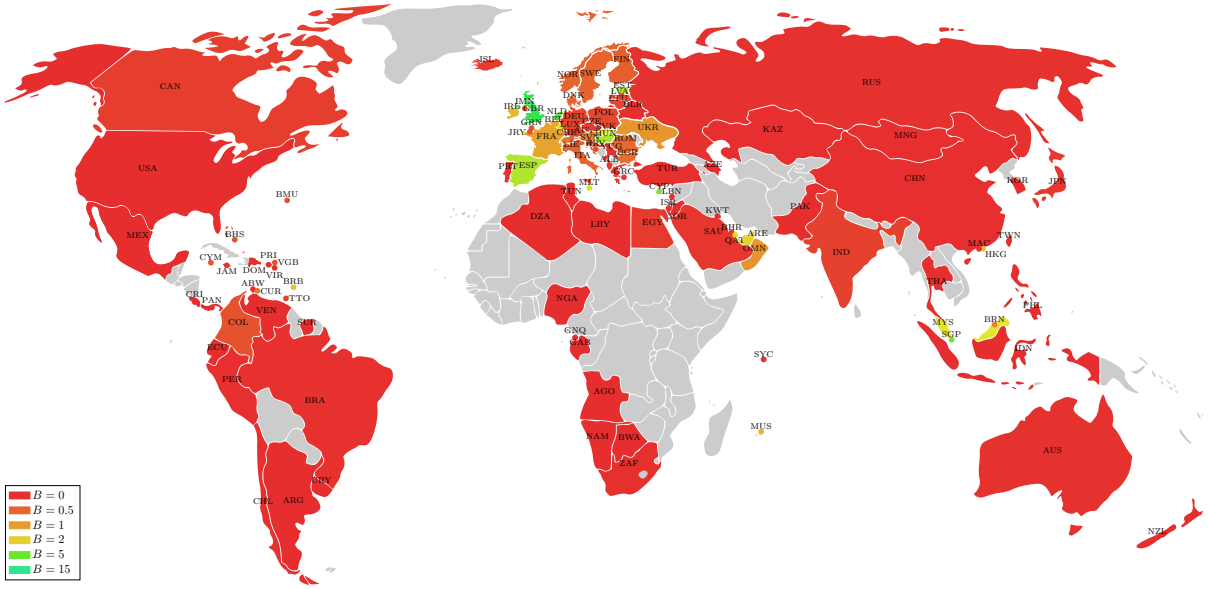


Figure 2.2: The countries that are more central in the *unweighted* network have a higher *unweighted* betweenness centrality value B.

We conclude this results-section with a short examination of the edge betweenness centralities (edge flows). The following table gives the first 10 edges, sorted according to their edge betweenness centrality, respectively edge *weighted* betweenness centrality (edge flow). One notable difference between the unweighted and the weighted edge flows is that the flows (of the top-10 edges) are higher than in the unweighted case. If an edge is often used on a path ending or starting at a big country (measured according to gdp), it gets a high weighted edge flow. Note also that edges with big economies as endpoints often occur in the top-10 of edges (ranked according to weighted edge flow). Also Great Britain (which is a quite large economy) very often occurs as one of the edge-endpoints, since it is the most important conduit country.

Pos. (weight)	Edge (u, v)	$B^W[(u, v)]$
1	USA – GBR	5.90293
2	GBR – CHN	4.34116
3	USA – LUX	4.22889
4	CHN – USA	3.87535
5	JPN – GBR	2.91545
6	LUX – CHN	2.57707
7	IDN – GBR	1.65730
8	IND – USA	1.45947
9	JPN – USA	1.44194
10	GBR – IND	1.31551
..		
23	USA – NLD	0.92941

Pos. (unweight.)	Edge (u, v)	$B[(u, v)]$
1	BRB – GBR	1.58869
2	NLD – EGY	1.03431
3	SAU – ESP	0.93851
4	TWN – SGP	0.92638
5	IDN – GBR	0.92593
6	ESP – CRI	0.92593
7	UKR – LBY	0.92593
8	URY – ESP	0.92593
9	DZA – QAT	0.91727
10	LBN – QAT	0.91727
2	NLD – EGY	1.03431

Table 2.2: The 10 edges with the highest (weighted/unweighted) edge betweenness (flow). The edge with the highest (weighted/unweighted) edge betweenness that has NLD as one of its two endpoints, is also contained in the table.

The following two figures show the first 20 edges, sorted according to their edge *weighted* betweenness centrality (edge flow), respectively edge unweighted betweenness centrality. Thicker edges have a higher value.

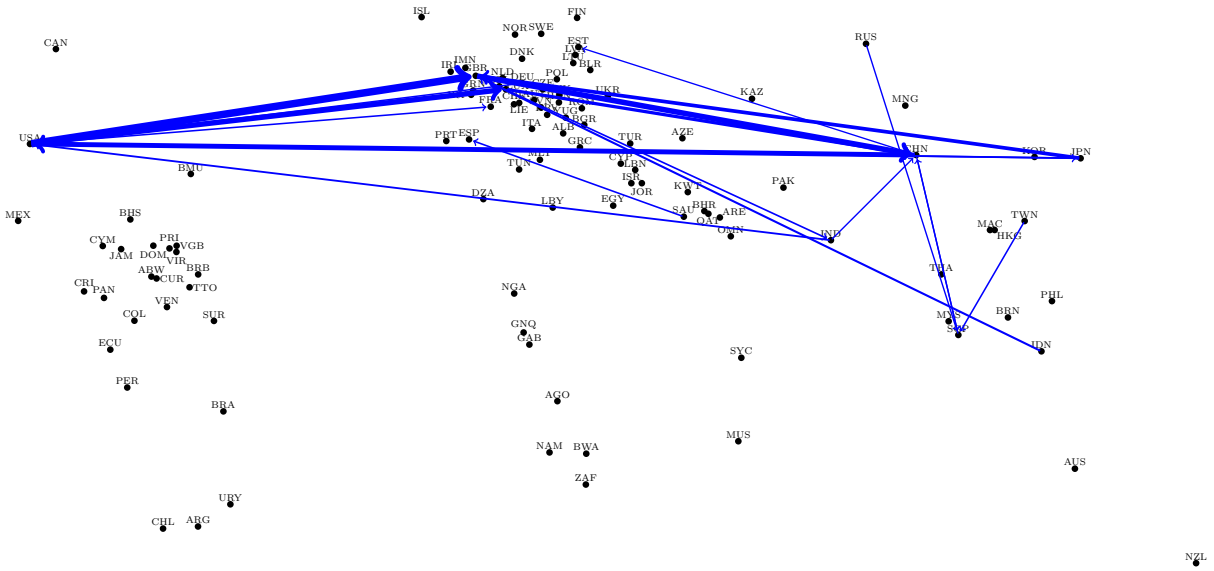


Figure 2.3: The 20 edges with the highest *weighted edge* betweenness centrality (edge flow).

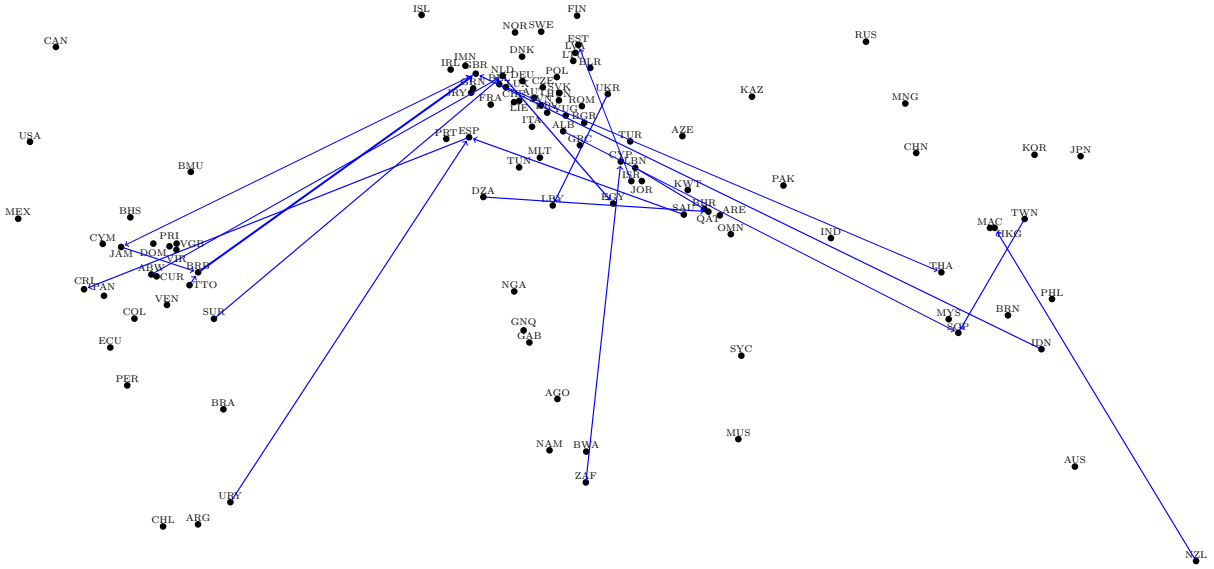


Figure 2.4: The 20 edges with the highest *unweighted edge* betweenness centrality (unweighted edge flow).

The (weighted) betweenness centrality values of all 108 countries, and larger prints of most figures, can be found in the Appendix (Chapter *A*).

Chapter 3

Shrinking strongly connected components

Let $G = (V, E)$ be a directed graph, with (multiplicative) edge reliabilities $r : E \to (0, 1]$. (If a graph had edge reliabilities of 0, we delete those edges). The reliability of the path is the product of the reliabilities of the edges in the path. As we have seen in Lemma 2.2.9, the graph G' of tight edges with respect to $\delta(s, \cdot)$ (for some $s \in V$) of E can contain cycles, but only cycles of reliability 1 (i.e. all edges on the cycle must have reliability 1). Therefore the graph G' is not necessarily acyclic. That means that we might not be able to count the paths in G' in polynomial time, i.e. we might not be able to count maximum reliability paths in G in polynomial time. Returning to our network of 108 countries, we can deal with this problem in two ways:

(i) Add a small penalty of ε% tax to each edge (except possibly the first edge along a path) in the network, as is done in Section 2.4.1. With this approach, it is possible to count exactly all maximum reliability paths *of shortest length*. This solution is used throughout the thesis.

(ii) Try to identify 'areas' that consist of cycles of reliability 1 and contract them to supernodes. In each area, it is possible to go from any country to another (within the same area) along reliability 1 edges (which means that zero tax is paid along those edges). We will define an 'area' to be a *strongly connected component*, which will be defined in the next section. We will contract the areas to supernodes. In the resulting graph we will be able to count paths efficiently. That enables us to compute centrality measures for the areas.

This chapter will deal with the approach proposed under point (ii). In this chapter we will therefore assume that there is *no* penalty of ε% tax (as under point (i)) added to the network.

3.1 Strongly connected components

In this section we introduce the notion of a *strongly connected component*. Later we discuss algorithms to identify strongly connected components in a directed graph $G = (V, E)$. This section is mainly based on [Tar72], although also [MS07] is used.

Definition 3.1.1 (Strongly connected graph)**.** A directed graph $G = (V, E)$ is *strongly connected* if for each pair u, v of vertices in G there exist paths P_1 from u to v and P_2 from v to u in G.

Let $G = (V, E)$ be a directed graph. Suppose we define a relation $\sim$ on V by: $u \sim v$ if and only if there exists paths P_1 from u to v and P_2 from v to u in G. It is not hard to see that $\sim$ defines an equivalence relation on V. (reflexivity: immediate, symmetry: swap P_1 and P_2, transitivity: concatenate paths). Let the different equivalence classes under this relation be

called V_i, for $i = 1, \ldots, N$. Let G_i be the subgraph of G *induced* by V_i, i.e. $E_i = \{(v, w) \in E \mid v, w \in V_i\}$ and $G_i = (V_i, E_i)$. By construction, each G_i is strongly connected. Moreover, each G_i cannot be a proper subgraph of a strongly connected subgraph of G, because otherwise the vertices in V_i would be $\sim$-equivalent to vertices not in V_i.

Definition 3.1.2 (Strongly connected component)**.** The subgraphs G_i, $i = 1, \ldots, N$ are the *strongly connected components* of G. Often we refer to a strongly connected component G_i by its corresponding set of vertices V_i.

We want to identify the strongly connected components in a directed graph $G = (V, E)$. The algoritm we will give was found by Tarjan [Tar72]. Suppose we perform a depth-first search on a directed graph G starting at some vertex $u \in V$. The set of edges which lead to a new vertex (a white vertex in the description in Algorithm 1) when traversed are *tree* edges, and they form a depth-first tree T of G rooted at u. We can divide the remaining explored edges (u, v) of G (i.e. the edges explored in the depth-first search of G that are no tree edges) into three sets:

(i) The set of edges from descendants to ancestors in T. These edges we call *fronds*, cf. Definition 1.4.2. In Remark 1.4.1 we have seen how to identify a frond edge: edge $e = (v, w)$ is a frond if vertex w is *green* when edge (v, w) is considered by the algorithm for the first time.

(ii) The set of edges from ancestors to descendants in T. These edges are called *forward edges*. We ignore these edges, since given T, these edges have no effect on the strongly connected components of G. Note that, when (v, w) is a forward edge, it holds that v is discovered before w by the depth-first search (as there is a tree path from v to w), but w is already colored red when edge (v, w) is considered by the depth-first search (if w would be white, edge (v, w) would be a tree edge. If w would be green, edge (v, w) would be a frond).

(iii) The set of edges from one subtree to another subtree in T. These edges we call *cross-links*. An edge (v, w) can only be a cross link if w is red when edge (v, w) is explored for the first time[1]. Note that if (v, w) is a cross-link, then v is discovered later than w during the depth-first search (otherwise the edge (v, w) would be either part of our tree T or a forward edge).

A tree edge we will denote by $v \to w$. A frond or a cross-link we denote by $v \rightsquigarrow w$. We can apply depth-first search on G repeatedly until all the edges are explored. Then we will obtain a set of trees (that contain all the vertices of G), the *spanning forest* F of G and sets of fronds and cross-links. The other edges we throw away. We call the resulting directed graph consisting of a spanning forest and sets of fronds and cross-links a *jungle*. Suppose we number the vertices according to the order in which they are *discovered* (and colored green) during the search. We refer to the vertices by their number.[2]

Lemma 3.1.1. *Suppose v and w are in the same strongly connected component of a graph G. Let F be a spanning forest of G obtained by repeated depth-first search. Then v and w have a common ancestor in F. Moreover, if we call the highest numbered common ancestor u, it holds that u lies in the same strongly connected component as v and w.*

Proof. We assume $v \leq w$ (otherwise we interchange v and w). Let P be a path from v to w in G. Let T_u with root u be the smallest subtree of a tree in F containing all the vertices in P. Note that such a tree exists, since every edge of a path cannot lead to another tree in F with

[1] This holds since if w is white edge (v, w) is first examined by the depth-first search, (v, w) would be a tree edge, and if w is green when edge (v, w) is first examined, edge (v, w) would be a frond.

[2] Note that these numbers correspond to the *discovery times* $d[\,]$ from Algorithm 1, the original description of *depth-first* search.

larger numbered vertices (otherwise the edge would be a forward edge (which is not possible) or a tree edge, joining the two trees).

An edge of a path could lead to another tree with lower numbered vertices, but then the path cannot end at w since $v \leq w$. Therefore the path P must be contained in *one* tree in F and hence we can construct T_u, which means that v and w have a common ancestor in F.

Moreover, the path P must pass *through* vertex u. If $v = u$ or $w = u$ this is clear. Otherwise, let T_{u_1} and T_{u_2} be two distinct subtrees of T_u containing vertices on P such that $u \to u_1$ and $u \to u_2$. If only one such tree exist, then u lies on P (since T_u is a smallest subtree of a tree in F containing all vertices in P). If two such subtrees exist, path P can only get from T_{u_1} to T_{u_2} by passing through u since no point in T_{u_1} is an ancestor of a point in T_{u_2} and vice versa, which means that the path can only cross from T_{u_1} to T_{u_2} if the latter has smaller numbered vertices, and this is not possible since the path leads from v to w (a higher numbered vertex than v). Therefore the path needs to pass through u. □

Corollary 3.1.2. *Let C be a strongly connected component of G. Then the vertices of C define a subtree of a tree in F, the spanning forest of G. We call the root of this subtree the* root *of the strongly connected component C.*

We would like to find the roots of the strongly connected components.

Definition 3.1.3 (Lowlink)**.** For $v \in V$, we denote the smallest vertex which is in the same component as v and is reachable from v by traversing (possibly zero) tree edges followed by at most one frond or cross-link, by Lowlink(v). We call Lowlink(v) the *lowlink* of v.

Lemma 3.1.3. *A vertex v is the root of some strongly connected component if and only if* Lowlink(v) $= v$.

Proof. "$\Longrightarrow$" Suppose v is the root of a strongly connected component of G. This implies that Lowlink(v) $= v$, since otherwise Lowlink(v) $< v$ which means that a proper ancestor of v would be in the same connected component as v (and then v cannot be the root of a strongly connected component).

"$\Longleftarrow$" Suppose Lowlink(v) $= v$, but that v is *not* the root of a strongly connected component. Let u be the root of the strongly connected component that contains v. Note that v is a vertex (in this component) different from u. There must be a path P from v to u in G. Let e be the first edge on this path with head w not in de subtree T_v with root v of the tree containing v. Then e is a frond or a cross-link and by definition (of the Lowlink) we have Lowlink(v) $\leq w$ (since v and w are in the same strongly connected component). But note that $w < v$ since the path can only lead to subtrees with smaller numbered vertices (otherwise w would be contained in the subtree T_v). Therefore Lowlink(v) $< v$, in contradiction with our assumption. □

It follows that, to compute the strongly connected components of G, we need to find the Lowlink-values of the vertices. The vertices with Lowlink(v) $= v$ (where the vertices are numbered according to the order in which they are reached during the search) are exactly the roots of subtrees that form strongly connected components of G. We claim that Algorithm 8 does the trick.

Theorem 3.1.4. *Algorithm 8 finds the strongly connected components of a directed graph $G = (V, E)$ correctly in time $\mathcal{O}(|V| + |E|)$.*

Proof. First we prove that the algorithm terminates in time $\mathcal{O}(|V| + |E|)$. Note that the algorithm is a modified depth-first search algorithm, with the addition that we calculate lowlink-values during the search and that each vertex is placed on a stack S once and is removed from this stack once. During the search, every vertex and every edge is visited once. Also, testing to see if a vertex is on the stack S can be done in a fixed time if we keep a boolean array of size $|V|$ which answers this question for each vertex. The amount of time added by the

Input: Directed graph $G = (V, E)$.
Output: The strongly connected components of G (as sets of vertices).

Initialize: $i := 0$, $S = \emptyset$, (all vertices $v \in V$ are colored *white*).
foreach $v \in V$ **do**
  if v is not yet numbered **then**
    Strongconnect(v)
  end
end
return strongly connected components of G (as sets of vertices).

Procedure *Strongconnect*(v)
Number$(v) = i$, Lowlink$(v) = i$, $i = i + 1$
put v on the stack of vertices S, (color v *green*).
foreach edge (v, w) in E **do**
  if w is not yet numbered **then** // w is white and (v, w) is a *tree edge*.
    Strongconnect(w)
    Lowlink$(v) = \min($Lowlink$(v),$ Lowlink$(w))$
  else if Number$(w) <$ Number(v) **then** // (v, w) is a *cross edge* or a *frond*.
    if w is in S **then** // w, v contained in the same SCC (proof of Theorem 3.1.4).
      Lowlink$(v) = \min($Lowlink$(v),$ Number$(w))$
    end
  end
end
if Lowlink$(v) =$ Number(v) **then**
  Start a new strongly connected component
  while w on top of S satisfies Number$(w) \geq$ Number(v) **do**
    Delete w from S and put w in the current strongly connected component.
  end
end
(color v *red*)

Algorithm 8: Tarjan's algorithm to find the strongly connected components of G. The coloring steps (between brackets) are meant for understanding of the depth-first search, and are not necessary for the functioning of the algorithm.

operations with the stack and the lowlinks is therefore linear in $|V|$ and $|E|$. We conclude that the algorithm terminates in time $\mathcal{O}(|V|+|E|)$ (where we observe that the storage space required by the algorithm is bounded in $\mathcal{O}(|V| + |E|)$).

It remains to prove that the algorithm is correct. We prove that each time *Strongconnect*(u) is finished for a vertex v (i.e. when u is colored red), it holds that

(i) Lowlink(u) is correctly computed,
(ii) and all the strongly connected components in T_u, the subtree rooted at u in the DFS-forest, are already returned by the algorithm.

To do this, we assume (inductively) that during the execution of the algorithm for all vertices that are already finished (and colored red) just before some vertex v is finished, both (ii) and (i) hold. We prove that both conditions also hold for vertex v.

Suppose an edge (v, w) with $w < v$ is explored by the algorithm. It holds that $w \in S$ if and only if w and v are in the same strongly connected component.

"$\Longrightarrow$" Suppose first that v and w are *not* in the same strongly connected component. If v and w have no common ancestor, then w was finished before v by the algorithm (note that $w < v$), removed from S and placed in a component (there must be an ancestor of w that is a component root), so $w \notin S$.

If v and w have a common ancestor u but are not in the same strongly connected component, there must be a strongly connected component root r on the tree path $u \to w$. Since $w < v$, this root r was discovered and finished by the algorithm before the algorithm considers (v, w), and hence $w \notin S$ at the moment that the algorithm considers (v, w).

"$\Longleftarrow$" Conversely, if $w \notin S$ then, since $w < v$, the recursive call to w already must have been completed and w is already colored red. Hence, using the induction hypothesis (property (ii) holds for w), the strongly connected component containing w has already been returned by the algorithm, so v and w are not in the same strongly connected component.

We conclude that it holds indeed that $w \in S$ if and only if w and v are in the same strongly connected component, and hence Lowlink(v) is correctly computed, as follows from the definition of the *lowlink*.

We now prove property (ii) for v, *using* that property (i) holds for v. We only need to prove (ii) for a vertex v for which the algorithm reaches the output-phase, i.e. Lowlink$(v) = v$. If a strongly connected component is returned by the algorithm, by property (i), Lowlink(v) is computed correctly. All vertices at the top of the stack S (including $v =$ Lowlink(v)) are placed into a strongly connected component. These vertices are exactly the vertices that are descendants of v but that are not contained in another strongly connected component (using property (ii) for the vertices finished before v). By Corollary 3.1.2 in combination with Lemma 3.1.3, these vertices form exactly the strongly connected component with v as root. □

Suppose we have identified the strongly connected components (which will be the 'areas' as described at the beginning of this chapter) in a directed graph $G = (V, E)$. Then we can 'contract' them to supervertices .

Definition 3.1.4 (Condensation graph)**.** Let $G = (V, E)$ be a directed graph with $G_i = (V_i, E_i)$, $i = 1, \ldots, N$ as strongly connected components. The *condensation graph* $G_c = (V_c, E_c)$ of G has as vertex set

$$V_c := \{V_i \,:\, i = 1, \ldots, N\},$$

i.e. G_c contains exactly one vertex for each strongly connected component of G. The edge set of G_c we define as follows:

$$E_c := \{(V_i, V_j) \,:\, i \neq j \text{ and } \exists\, e \in E \text{ from some vertex in } V_i \text{ to some vertex in } V_j\}.$$

If edges in G are assigned additive edge costs $c : E \to R_{\geq 0}$, then we let the cost of edge (V_i, V_j) be the smallest edge cost of the edges in G that go from some vertex in V_i to some vertex in V_j in G. Similarly, if the edges in G are assigned multiplicative edge reliabilities $r : E \to (0, 1]$, then we let the reliability of edge (V_i, V_j) be the biggest edge reliability of the edges in G that go from some vertex in V_i to some vertex in V_j in G.

Proposition 3.1.5. *The condensation graph G_c of G is an acyclic graph.*

Proof. Suppose the condensation graph is not acyclic. Let C be a cycle rooted at some vertex V_i. Note that the edge (V_i, V_i) is by construction not contained in G_c. Therefore there should be a V_j in C unequal to V_i and we can split the cycle into two paths, one from V_i to V_j and one from V_j to V_i. But then in our original graph G there is a path from some, and therefore from *each* vertex in V_i to some, and therefore to *each* vertex in V_j and conversely there is a path from each vertex in V_j to each vertex in V_i. That is, the vertices from V_i and V_j belong to the same strongly connected component of G. This is in contradiction with the fact that V_i and V_j are different strongly connected components of G. □

Note that we can count simple paths between two vertices (if necessary passing through some given third vertex) in linear time in acyclic directed graphs, with Algorithm 5. We will use this property in the next section. We conclude this section with a useful property of Tarjan's algortithm: Tarjan's algorithm outputs the strongly connected components in *reverse topological order*. The proof of this statement is not in the paper [Tar72], but we provide it here.

Lemma 3.1.6 (Reverse topological order as a consequence of Tarjan's algorithm)**.** *Algorithm* 8 *outputs the strongly connected components of* G *in* reverse topological order*. This means, that if* V_i *and* V_j *are two strongly connected components such that there exists an edge* (v, w) *from a vertex* $v \in V_i$ *to a vertex* $w \in V_j$*, then* V_j *is returned* before V_i *by the algorithm.*

Proof. Suppose for the sake of contradiction that there exists an edge (v, w) from a vertex $v \in V_i$ to a vertex $w \in V_j$ but that V_i is returned *before* V_j by the algorithm. Let r_j (resp. r_i) be the root of the strongly connected component V_j (resp. V_i). Then, r_i is finished *before* r_j is finished by the algorithm. Note that v is a descendant of r_i so it is finished *before* r_i by the algorithm (as the structure of the algorithm is an ordinary depth-first search). Hence, w would also be a descendant of r_i (which is not possible since then w would be returned in the same strongly connected component as v) *unless* w is already discovered before v by the algorithm. But then w would be contained in S when (v, w) is explored (as every vertex is placed on S and removed from S once, and w will only be removed from S when V_j is returned) and $w < v$. This would mean that w and v are in the same strongly connected component (by the same argument as in the proof of Theorem 3.1.4), which is not the case. □

3.2 Counting super-vertices

Let $G = (V, E)$ be a directed graph with (additive) edge costs $c := E \to \mathbb{R}_{\geq 0}$. We want to shrink strongly connected components with edges of cost 0 into super-vertices. All results in this section also hold for *multiplicative* edge reliabilities, by replacing 'edges of cost 0' by 'edges of reliability 1' and 'shortest path' by 'maximum reliability path'. We will compare two procedures.

Algorithm 3.2.1 (Counting procedure – I)**.** The first procedure is as follows.

(i) Let $H = (V, E_0)$, where
$$E_0 = \{e \in E : c(e) = 0\}.$$
The graph H is a graph consisting of the vertices of G and all edges of G that have cost 0.

(ii) Use Tarjan's algorithm (Algorithm 8) to identify all strongly connected components of H as sets of vertices V_i. Note that the V_i are subsets of V, the vertex set of G.

(iii) Now, construct the condensation graph of G (as in Definition 3.1.4), but using the strongly connected components V_i of H instead of the strongly connected components of G itself. The resulting graph we call G_c.

(iv) Construct the graph of tight edges (with respect to some source node $V_s \in G_c$ such that $s \in V_s$) $(G_c)'$ out of G_c.

(v) We will be able to count paths in $(G_c)'$ since $(G_c)'$ will not contain cycles. This is because $(G_c)'$ could only contain cycles of cost zero, but these are deleted by the previous procedure.

Algorithm 3.2.2 (Counting procedure – II)**.** The second procedure is as follows.

(i) Let G' be the graph of tight edges with respect to $\delta(s, \cdot)$ for some node $s \in V$.

(ii) Use Tarjan's algorithm (Algorithm 8) to identify all strongly connected components of G' as sets of vertices V_i.

(iii) Construct the condensation graph $(G')_c$ of G' (as in Definition 3.1.4).

(iv) Count the paths in the resulting graph. Here we can use the (reverse) topological order of the strongly connected components given by Tarjan's algorithm (see Lemma 3.1.6).

Both procedures can give different results.

Lemma 3.2.1. *We compare Algorithm 3.2.1 with Algorithm 3.2.2. The graphs* $(G_c)'$ *and* $(G')_c$*, do both* not *contain cycles, but it does* not *necessarily hold that* $(G_c)' = (G')_c$*.*

Proof. Note that $(G_c)'$ does not contain cycles, since G_c does not contain cycles of cost 0 and the shortest path graph can only possibly contain cycles if they have cost 0. Also $(G')_c$ does not contain cycles, since it does not contain strongly connected components that consist of more than one vertex.

Let $G = (V, E)$ be a graph on the vertices s, u, v, w with edges and edge costs as in Figure 3.1.

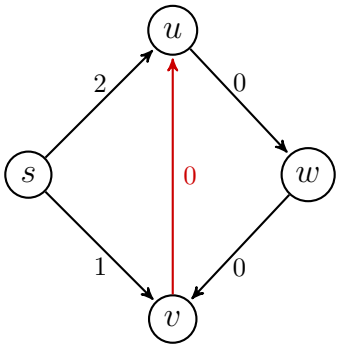


Figure 3.1: Example graph. The edge (v, u) is *not* tight with respect to $\delta(s, \cdot)$. All other edges are tight with respect to $\delta(s, \cdot)$.

Note that edge (v, u) is *not* tight with respect to $\delta(s, \cdot)$, while it has cost 0. Furthermore, $(G_c)'$ will consist of two vertices, given by $\{s\}$ and $\{u, v, w\}$. On the other hand, $(G')_c$ will consist of four vertices, $\{s\}$, $\{u\}$, $\{v\}$ and $\{w\}$. Hence, in this example we have $(G_c)' \neq (G')_c$. □

Both the graphs $(G_c)'$ and $(G')_c$ will be acyclic, and hence we can count simple paths in those graphs efficiently. Which of the two procedures is the most reasonable to use? If we use Algorithm 3.2.2, for each $s \in V$, the obtained strongly connected components can differ and we have to contract the vertices depending on the source vertex $s \in V$. This will not allow us to compute betweenness for the strongly connected components: they can differ depending on the source node.

If we use Algorithm 3.2.1, we only have to contract the vertices into supernodes *once*, after which we can count shortest paths between any pair of nodes V_i, V_j efficiently in the resulting graph G_c (where G_c is as in Algorithm 3.2.1). This will enable us to compare the strongly connected components in H (the vertices of G') and calculate betweenness centrality measures for these strongly connected components. Therefore we will use this approach.

Note that we can not use the trick of shrinking supernodes to count paths between two vertices s, t efficiently in the *original* graph $G = (V, E)$. This already follows from the fact that counting simple paths is $\#P$-hard. But if we (magically) could count paths between pairs of vertices efficiently *inside* strongly connected components, we would be able to count shortest paths from s to t efficiently.

Theorem 3.2.2. *Suppose that* $G' = (V', E')$ *is the shortest path graph of* $G = (V, E)$*, with edge costs* $c : E \to \mathbb{R}_{\geq 0}$*, source node* s *and sink node* t*. Let* $(G')_c$ *be the topologically sorted condensation graph of* G'*, the graph in which all strongly connected components* $V'_1, \ldots, V'_N$ *of* G' *are contracted into supernodes. Suppose we can count in polynomial time inside any strongly*

connected component V_i the number of simple paths from any $u \in V_i$ to any $v \in V_i$. Then we can count the number of simple paths from s to t in G' in polynomial time.

Proof. We claim that Algorithm 9 (which runs in linear time) counts the paths correctly. By initializing, the number of simple paths from each $v \in V_k$ to t is set correctly. We proceed by induction. Suppose that for each $v \in V_j$ with $1 < j \leq k$ the number of simple paths from v to t is set correctly. We will prove that the number of simple paths for each $v \in V_{j-1}$ to t is set correctly by the algorithm.

An edge out of V_{j-1} can only have a vertex in V_l, with $l > j-1$, as an endpoint. Therefore the first *foreach*–loop within the outer *foreach*–loop sets correctly the number $N'(u)$ of simple paths from each $u \in V_{j-1}$ to t *without* taking the paths inside V_{j-1} into account. In the second *foreach*–loop (within the outer *foreach*–loop) the paths inside V_{j-1} come into the counting as well. Then the number $N(u)$ of simple paths from each $u \in V_{j-1}$ to t is set correctly. □

Input: Shortest path graph $G' = (V', E')$ of G. Source $s \in V'$, sink $t \in V'$.
Topologically sorted DAG $(G')_c = ((V')_c, (E')_c)$ with vertex $s \in V'_1$,
sink $t \in V'_k$. Topological order $V'_1, \ldots, V'_k, \ldots$
σ_{uv} is the number of simple paths from any vertex $u \in V'_i$
to any vertex $v \in V'_i$ *inside* SCC V'_i. If $u = v$ then $\sigma_{uv} = 1$.
Output: The number of simple s,t-paths $N(s)$.

Initialize: $N(v_k)$ is the number of simple paths from any vertex $v_k \in V'_k$ to t inside V'_k.
If $v_k = t$ then $N(v_k) := 1$. For all $v \in V' \setminus V'_k$ we set $N(v) := N'(v) := 0$.
foreach $Y \in (V')_c$ in reverse topological order (i.e. in order $\ldots V'_k \ldots V'_1$) **do**
  foreach child $C \in V'$ of Y **do**
    List $L = \emptyset$
    foreach edge (v, w) such that $v \in Y$ and $w \in C$ **do**
      $N'(v) = N'(v) + N(w)$
      add v to L
    end
    foreach pair of vertices $u \in Y$, $v \in L$ **do**
      $N(u) = N(u) + N'(v) \cdot \sigma_{uv}$
    end
  end
end
return $N(s)$

Algorithm 9: Algorithm to count the number of simple paths between s and t in the shortest path graph of G with respect to $\delta(s, \cdot)$, *given* the number of simple paths between any two nodes in any SCC of G' (the shortest path graph of G).

Theorem 3.2.3 (Counting simple s,t-paths in a strongly connected graph is $\#P$-complete)**.** *Let H be a strongly connected graph, with a source node s and a sink node t. The problem of counting all simple s,t-paths is $\#P$-hard.*

Proof. First, checking whether a given path is a simple s,t-path in a strongly connected graph can be done in polynomial time: $\mathcal{O}(|V| + |E|)$.

Suppose it would be possible to count all simple u,v-paths for each pair of vertices u, v in a *strongly connected graph* in polynomial time. With the help of Algorithm 9 one could then count exactly all simple paths from some source s to some sink t in the shortest path graph G' of an arbitrary directed graph G in polynomial time. However, the shortest path

graph can have an arbitrary form (in particular, if a graph consists only of edges cost 0, then the shortest path graph with respect to $\delta(s,\cdot)$ is the graph itself[3]). Therefore we would be able to count simple s,t-paths in an arbitrary graph in polynomial time, which is $\#P$-hard (see Example 1.3.5). □

3.3 Results

First, we will examine the strongly connected components in our network of countries. As we have seen in Section 2.4, our network of countries consists of $|V| = 108$ graphs, one for each source country $s \in V$. We compute the strongly connected components over the zero-tax edges (reliability 1 edges) in each graph G_s. It turns out that there is just *one* strongly connected component that consists of more than one country in *each* graph G_s. We denote this strongly connected component by $\mathrm{SCC}_0(G_s)$. We find that the intersection (over all $s \in V$) of all these components consists of 64 countries.

$$\begin{aligned}\cap_{s\in V}\mathrm{SCC}_0(G_s) = \{&\mathrm{ALB},\ \mathrm{AUT},\ \mathrm{BHS},\ \mathrm{BRB},\ \mathrm{BLR},\ \mathrm{BEL},\ \mathrm{BMU},\ \mathrm{BRN},\ \mathrm{BGR},\ \mathrm{CYM},\\ &\mathrm{COL},\ \mathrm{HRV},\ \mathrm{CUR},\ \mathrm{CYP},\ \mathrm{CZE},\ \mathrm{DNK},\ \mathrm{EGY},\ \mathrm{EST},\ \mathrm{FIN},\ \mathrm{FRA},\\ &\mathrm{DEU},\ \mathrm{GRC},\ \mathrm{GRN},\ \mathrm{HKG},\ \mathrm{HUN},\ \mathrm{ISL},\ \mathrm{IDN},\ \mathrm{IRL},\ \mathrm{IMN},\ \mathrm{ISR},\\ &\mathrm{ITA},\ \mathrm{JAM},\ \mathrm{JRY},\ \mathrm{LVA},\ \mathrm{LBN},\ \mathrm{LIE},\ \mathrm{LTU},\ \mathrm{LUX},\ \mathrm{MYS},\ \mathrm{MLT},\\ &\mathrm{MUS},\ \mathrm{MNG},\ \mathrm{NLD},\ \mathrm{NZL},\ \mathrm{NOR},\ \mathrm{OMN},\ \mathrm{POL},\ \mathrm{PRT},\ \mathrm{QAT},\ \mathrm{ROM},\\ &\mathrm{SAU},\ \mathrm{SGP},\ \mathrm{SVK},\ \mathrm{SVN},\ \mathrm{ZAF},\ \mathrm{ESP},\ \mathrm{SWE},\ \mathrm{CHE},\ \mathrm{TWN},\ \mathrm{TTO},\\ &\mathrm{UKR},\ \mathrm{ARE},\ \mathrm{GBR},\ \mathrm{VGB}\}.\end{aligned}$$

Note that for every pair of countries in $\cap_{s\in V}\mathrm{SCC}_0(G_s)$, companies can send money from one country to the other country without paying taxes. Because we take the intersection of the sets $\mathrm{SCC}_0(G_s)$, there is a reliability 1 path from every country in $\cap_{s\in V}\mathrm{SCC}_0(G_s)$ to every other country in $\cap_{s\in V}\mathrm{SCC}_0(G_s)$.[4] Furthermore, we find that

$$\bigcup_{s\in V}\mathrm{SCC}_0(G_s) = \left(\bigcap_{s\in V}\mathrm{SCC}_0(G_s)\right) \bigcup \mathrm{CHL} \bigcup \mathrm{IND} \bigcup \mathrm{AUS} \bigcup \mathrm{VEN},$$

i.e. the union of all components $\mathrm{SCC}_0(G_s)$ consists of 68 countries. If we define

$$S := \{s \in V \,:\, \delta_s(s,\mathrm{NLD}) = 1 \text{ and } \delta_{\mathrm{NLD}}(\mathrm{NLD},s) = 1\},$$

where δ_s denotes the distance function in G_s for any $s \in V$, then we find with our data that

$$S = \bigcap_{s\in V}\mathrm{SCC}_0(G_s),$$

i.e. the countries to which companies can send money for free from the Netherlands and back (possibly via a tax-route), form exactly the set $\mathrm{SCC}_0(G_s)$. We would like to *shrink* strongly connected components of reliability 1 edges (zero-tax edges), as proposed in the previous section. However, the set of strongly connected components in G_s depends on $s \in V$, and hence we have not *one* set of strongly connected components over reliability 1 edges that we can shrink and for which we can compute centrality measures in the network. But since, for each $s \in V$, there is

[3]Here we are assuming that all vertices in G are reachable from s.

[4]In fact, for any two countries in $\cap_{s\in V}\mathrm{SCC}_0(G_s)$ there is a reliability 1 path from one country to the other country such that the edge reliability on the first edge of the path is the edge reliability for a first edge from the source on a path (where the edge reliabilities are as in Section 2.4), and there *also* is a reliability 1 path such that the first edge on the path has the edge reliability for an edge with tail not equal to the source.

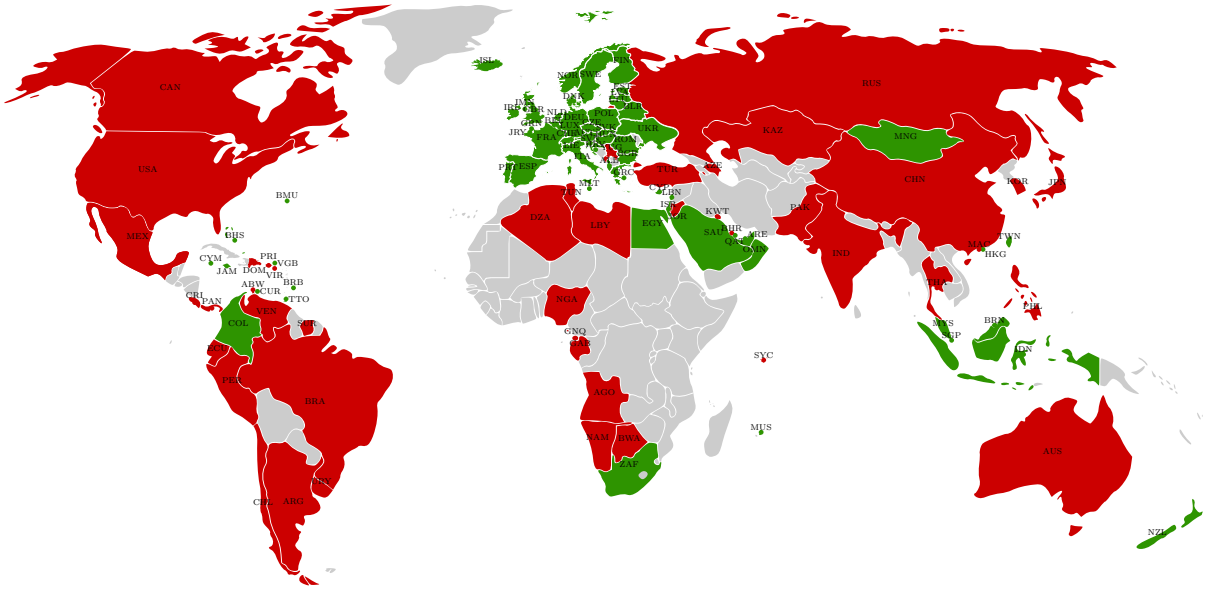


Figure 3.2: All countries in $\cap_{s \in V} \mathrm{SCC}_0(G_s)$ are colored green. Companies can send dividends without paying tax from *any* green country to *any* other green country, possibly via a route passing through conduit countries.

only *one* strongly connected component $\mathrm{SCC}_0(G_s)$ over the zero-tax edges in G_s that contains more than one country, we decide to shrink $\cup_{s \in V} \mathrm{SCC}_0(G_s)$ into one big supernode[5], which we denote by U. If a path starts at U, it is not clear how to define the edge reliabilities in the condensation graph in the CPB-problem. Let $s \in U$ be a country in the supernode if the path starts at U. Should we choose the reliability for the *first* edge on a path (cf. Section 2.4) or the reliability for *another* edge on the path, when the path leaves U? This depends on the starting vertex $s \in U$, and hence there is no canonical choice for paths starting from U.

Nevertheless, we shortly did an experiment with paths *not* starting (or ending) at U. If $v \notin U$ is a country we write $\{v\}$ for the vertex in the component graph (depending on the starting point $s \in V \setminus U$) in which the strongly connected component containing only v is shrunk. We write V_c for all the vertices in the resulting network: $V_c = U \cup \big(\cup_{v \in V \setminus U} \{v\}\big)$.

In the resulting network, with U is shrunk into a supernode, there are in total 3741 shortest paths not starting or ending at U. The supernode U is a *conduit* country on 2860 of those paths. Furthermore, we can compute the (unweighted) betweenness of U, and we find

$$B(U) = \sum_{\substack{\{s\},\{t\} \in V_c \\ \{s\} \neq \{t\} \neq U}} \frac{\sigma_{\{s\}\{t\}}(U)}{\sigma_{\{s\}\{t\}}} = 1207.245,$$

Normalizing by the number of pairs of vertices in the condensation graph $(41 \cdot 40)$ and[6] multiplying by 100 gives 73.6. This is a *very* high betweenness. We conclude that countries not in the supernode *very* often use the supernode as a conduit country for sending money to another country not in the supernode. Since the CPB preferred to have betweenness measures for the *individual* countries and not for the supernodes, we end the discussion of the supernodes here.

In the next section we move from *strictly* shortest (or maximum reliability) paths to *almost* shortest paths: we will try to count paths that are *almost* as short as the shortest path, but not necessarily the strictly shortest.

[5] If we shrink the intersection the result might contain reliability 1 cycles.

[6] A better normalization for unweighted betweenness is dividing by the number of $\{s\}, \{t\}$ pairs with $\{s\} \neq \{t\} \neq U$ (here $40 \cdot 39$). However, throughout this thesis we normalize by the number of pairs $s, t \in V$ with $s \neq t$ to be better able to compare weighted and unweighted betweenness. To be consequent, we do that here also.

Chapter 4

Finding paths of relevant reliability

In this chapter we try to find paths that are *almost* of maximum reliability, instead of only paths of strict maximum reliability. We begin by proving that counting all paths within a certain range from the shortest path is $\#P$-hard. After that, we try to find a 'restricted' within range notion to be able to count 'restricted within range' paths in polynomial time.

4.1 APSP, nonrestricted relative range version

Let $G = (V, E)$ be a directed graph, and $c : E \to \mathbb{R}_{\geq 0}$ be a cost function, such that G does not contain zero-cost cycles. In this section we want to find the complexity of determining all relevant paths, using one of the following two notions of a 'relevant path'.

Definition 4.1.1 (*Relevant path*)**.** Given $\alpha \in \mathbb{R}_{\geq 0}$, We call a s, v-path P *relevant*

(i) within *additive* range if $c(P) \leq \delta(s, v) + \alpha$,

(ii) within *multiplicative* range if $c(P) \leq (1 + \alpha) \cdot \delta(s, v)$.

We would like to find all relevant s, v-paths within additive (or multiplicative) range.

Problem 2 (Counting relevant paths)**.** For any pair $(s, t) \in V \times V$ of vertices and for one of the two notions of relevance as in Definition 4.1.1, find the number of (or: all) relevant paths (with either of the two notions of relevance) from s to t. What is the complexity of solving this problem?

We bring Example 1.3.5 into remembrance: the counting version of the simple s, t-path problem is $\#P$-complete. That is, the problem of finding the number of simple s, t-paths in an arbitrary graph is $\#P$-complete. From this it follows that Problem 2 (with either of the two notions of relevance) is $\#P$-complete.

Theorem 4.1.1 (Counting relevant paths is $\#P$-complete)**.** *Given a graph $G = (V, E)$, a source node s, a sink node t (with $t \neq s$), an additive cost function $c : E \to R_{\geq 0}$, such that G does not contain zero-cost cycles, and an $\alpha \in \mathbb{R}_{\geq 0}$, we want to count the number of simple paths with cost smaller than $\delta(s, t) + \alpha$ (respectively smaller than $\delta(s, t) \cdot (1 + \alpha)$), where $\delta(s, t)$ is the shortest path distance from s to t. We claim that this problem, if $\alpha > 0$, is $\#P$-complete.*

Proof. We need to show that the decision version of this problem is in NP and that every problem in $\#P$ can be reduced to this problem.

(i) Given a set of edges $P \subset E$, we can determine in polynomial time, $\mathcal{O}(|V| + |E|)$, whether this is a simple s, t-path of cost smaller than $\delta(s, t) + \alpha$, (respectively smaller than $\delta(s, t) \cdot (1 + \alpha)$). Therefore the decision version of this problem is in NP.

(ii) Suppose that we have an instance of the s,t-simple path counting problem (a $\#P$-complete problem). We transform this instance into an instance of Problem 2 (with relevance within additive resp. multiplicative range) as follows.

(+) For within additive range paths, let all edge costs be ε, with $\varepsilon > 0$ some small number (to be determined). We want to find all s,t-paths with cost within the interval $[\delta(s,t)\,,\ \delta(s,t)+\alpha]$, for $\alpha > 0$ a given small number. If $n \cdot \varepsilon \leq \alpha$ then all simple s,t-paths in G are within additive range paths and vice versa.

($\cdot$) For within multiplicative range paths, let all outgoing edge costs of s be 1 and all other edge costs be ε, with $\varepsilon > 0$ some small number (to be determined). We want to find all s,t-paths with cost within range $[\delta(s,t)\,,\ \delta(s,t)\cdot(1+\alpha)]$, for $\alpha > 0$ a given small number. If we choose $\varepsilon \leq \alpha/n$ then it holds for each s,t-path P that

$$c(P) \leq 1 + n \cdot \varepsilon \leq 1 + \alpha \leq \delta(s,t) \cdot (1+\alpha),$$

since $\delta(s,t) \geq 1$ by construction. Therefore, if $\varepsilon \leq \alpha/n$ then all simple s,t-paths in G are within multiplicative range paths and vice versa.

Hence we reduced the count simple s,t-paths-problem to the count relevant paths problem (Problem 2). Note that this reduction can be done in polynomial time.

We conclude that the problem of counting relevant paths within additive (resp. multiplicative) range in G is $\#P$-complete. □

The above complexity result may be not surprising: it even holds that the problem of computing *one* strictly-second shortest path is *NP*-hard, as is proven by Lalgudi and Papaefthymiou in [LP97].

Theorem 4.1.2. *The problem of finding one simple strictly-second shortest s,t-path (i.e. finding a shortest path among all paths with cost strictly larger than $\delta(s,t)$) in a graph is NP-hard, provided that 0 cost edges are allowed.*

4.2 Restricted relative range notion for additive edge costs

To construct a 'relevant' path graph (containing all 'relevant' paths starting at some source vertex $s \in V$) we would like a *subpath optimality condition* to hold. For example, for shortest paths it holds that concatenating paths in the shortest path graph G_s (containing all shortst paths from s) gives a new path in the shortest path graph and therefore a new shortest path starting at s. Moreover, subpaths of shortest paths are again shortest paths. Implicitely, we make use of this condition in using the concept of a 'shortest path graph'. We will formally state the *subpath optimality condition*. In this section we deal with additive edge costs, in the next section we make the translation to multiplicative edge reliabilities.

Definition 4.2.1 (Subpath optimality condition)**.** Let $G = (V,E)$ be a directed graph, and $c : E \to \mathbb{R}_{\geq 0}$ be a cost function. Suppose we are given a notion of 'relevant' paths in G starting at any vertex $s \in V$. The *subpath optimality condition* holds if for all vertices $w \in V$, and for any *relevant* s,w-path P and *any* vertex v on P it holds that

(i) the s,v-subpath of P is relevant,
(ii) and for *any* relevant s,v-path P_1 it holds that P_1 concatenated with the v,w-subpath of P is again relevant.

Note that both conditions hold for shortest paths, since it holds that $\delta(s,v) + \delta(v,w) = \delta(s,w)$ if v lies on a shortest s,w-path.

With conditions (i) and (ii), a graph G_s constructed from relevant paths[1] (a 'relevant path graph') starting at s has the property that *all* paths from s are relevant paths.

Remark 4.2.1. Note that we can *not* replace conditions (i) and (ii) of Definition 4.2.1 by the condition that every x, y-subpath P' of an s, w-path P must be relevant. Suppose we call an s, w-path P 'relevant' if for every x, y-subpath P' of P it holds that $c(P) \leq \delta(x, y) \cdot (1 + \alpha)$. Then subpaths of relevant paths are again relevant. But condition (ii) of Definition 4.2.1 is not necessarily satisfied.

Consider the following example. Let $G_s = (V, E)$ be a graph with vertices s, x, v and w. Let there be a directed edges with costs as in Figure 4.1. Let $\alpha = 0.1$.

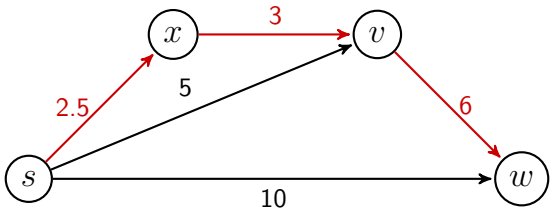


Figure 4.1: Example graph, with $\alpha = 0.1$. All edges are contained in a 'relevant path', but the path $\langle s, x, v, w \rangle$ is not relevant.

Note that the paths $P_1 := \langle s, x, v \rangle$, $P_2 := \langle s, v, w \rangle$ and $P_3 := \langle s, w \rangle$ are all 'relevant paths', the graph consists of relevant paths. However, condition (ii) is *not* satisfied. The path $P_2 = \langle s, v, w \rangle$ is relevant, with v, w-subpath $P_2' = \langle v, w \rangle$. Also $P_1 := \langle s, x, v \rangle$ is a relevant x, v path. But the concatenation of P_1 and P_2' is $\langle s, x, v, w \rangle$ and this is *not* a relevant path.

For relevant paths within *additive* range (as in Definition 4.1.1 (i)), we prove that property (ii) of Definition 4.2.1 does not necessarily hold.

Example 4.2.1 (Counterexample, additive range)**.** Let G be a graph consisting of the vertices s, v, w, x, y, edge costs as in Figure 4.2 It holds that $P = \langle s, x, v, w \rangle$ is a relevant s, w-path, with v lying on P. Also, $P_1 = \langle s, v \rangle$ is a relevant s, v path of cost $1 + \alpha$ and the v, w-subpath $P' = \langle v, w \rangle$ of P also has cost $1 + \alpha$, where $\alpha > 0$. However, the concatenation of P_1 and P' is *not* relevant, since it has cost $c(P_1) + c(P') = 2 + 2\alpha > 2 + \alpha = c(P) + \alpha$.

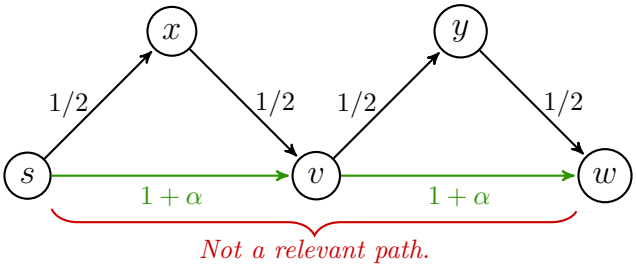


Figure 4.2: Example graph. Path $P_1 = \langle s, v \rangle$ is relevant and $P' = \langle v, w \rangle$ is a subpath of a relevant s, w-path, but their concatenation $\langle s, v, w \rangle$ is not a relevant path.

For the relative range notion of 'relevance' as in Definition 4.1.1, we will prove that property (i) of the subpath optimality condtion does *not* necessarily hold: subpaths (starting at s) of relevant paths (starting at s) need not to be relevant.

Example 4.2.2 (Counterexample, multiplicative range)**.** Consider a graph consisting of 4 vertices s, v, w, x. Let there be directed edges (s, v) of cost $c(s, v) = 12$, (s, x) of cost $c(s, x) = 5$, (x, v) of cost $c(x, v) = 4$ and (v, w) of cost $c(v, w) = 6$. Let $\alpha = 0.2$.

[1] A graph is constructed from relevant paths if its edge set is the union of edge sets of relevant paths.

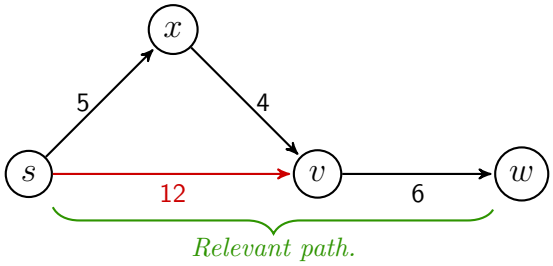


Figure 4.3: Example graph, with $\alpha = 0.2$.

The path $P = \langle s, v, w \rangle$ is relevant, since $18 = \delta(s, w) \cdot (1 + \alpha) = 15 \cdot (1 + \alpha)$. But the subpath $P' = \langle s, v \rangle$ does *not* satisfy $c(P') = 12 \leq 9 \cdot (1 + \alpha) = \delta(s, v) \cdot (1 + \alpha)$. Therefore we see that subpaths (starting at s) of relevant paths are not necessarily relevant: property (ii) of Definition 4.2.1 is not satisfied.

However, we will see that there exists a notion of 'restricted relative range' relevance, such that both conditions of the subpath optimality condition are satisfied. To this end, we seek a notion of *almost tight* edges, such that the graph of almost tight edges with respect to $\delta(s, \cdot)$ for some vertex $s \in V$ consists *exactly* of all relevant paths (with this 'restricted relative range' version of relevance). A natural definition of an 'almost tight edge' is the following:

Definition 4.2.2 (*Almost tight edge – proposed*)**.** We define an edge (v, w) to be *almost tight* with respect to $\delta(s, \cdot)$ if

$$c(v, w) + \delta(s, v) \cdot (1 + \alpha) \leq \delta(s, w) \cdot (1 + \alpha).$$

It holds that the graph of almost tight edges with respect to $\delta(s, \cdot)$ for some vertex $s \in V$ contains exactly all relevant paths (as we will later prove in Theorem 4.2.1), with the following 'restricted relative range' definition of a *relevant* path.

Definition 4.2.3 (*Relevant path – restricted relative range*)**.** We define a path $P = \langle s, \ldots, w \rangle$ to be relevant if for each subpath $P' = \langle x, \ldots, y \rangle$ of P it holds that

$$c(P') + \delta(s, x) \cdot (1 + \alpha) \leq \delta(s, y) \cdot (1 + \alpha).$$

This definition of *relevance* makes sense: when concatenating paths, we always remain 'within range'. If an x, y-path in the relevant path graph (that contains all relevant paths starting at s) is concatenated with a relevant s, x-path, the result must remain a relevant path.

An interpretation for companies: companies want to send tax over a path that is 'within range' in total, but in each step they do not want to lose too much money, and when concatenating tax routes the total tax must remain 'within range'.

Note that for this definition of 'relevance', condition (i) of Definition 4.2.1 is satisfied (this is an easy check for the reader). Condition (ii) of Definition 4.2.1 is also satisfied, although this requires some more consideration. We will prove that, if G does not contain cycles of cost 0, there exists a directed acyclic graph G' that contains exactly *all* relevant paths starting from some vertex $s \in V$ (i.e. each path in G' is a relevant path and each relevant path in G is a path in G'), just as we did for shortest paths. It then follows that both conditions of Definition 4.2.1 hold. We first formulate the problem of this section.

Problem 3 (Restricted relative range path problem)**.** For any pair $(s, v) \in V \times V$ of vertices, count all relevant paths (as defined in Definition 4.2.3) from s to v.

We try an approach similar to the approach we used in the strictly-shortest path case. Fix a vertex $s \in V$. We construct the graph $G' = (V', E')$ that consists of all almost tight edges with respect to $\delta(s, \cdot)$, i.e. let

$$V' := \{v \in V \mid v \text{ is reachable from } s\},$$

and

$$E' := \{(v, w) \in E \ : \ (v, w) \text{ is almost tight with respect to } \delta(s, \cdot)\}.$$

In the following theorem we prove that this graph consists *exactly* of all relevant paths that start at s.

Theorem 4.2.1. *A path P starting at s is contained in $G' = (V', E')$ if and only if P is a relevant path starting at s in G according to Definition* 4.2.3.

Proof. "$\Longleftarrow$": Suppose P is a relevant path starting at s in G (according to Definition 4.2.3). Let (x, y) be an edge of P. This is also a subpath of P so it holds that

$$c(x, y) + \delta(s, x) \cdot (1 + \alpha) \leq \delta(s, y) \cdot (1 + \alpha),$$

i.e. (x, y) is an almost tight edge with respect to $\delta(s, \cdot)$. So all edges of P are almost tight, i.e. P is contained in $G' = (V', E')$.

"$\Longrightarrow$": Suppose a path $P = \langle s, \ldots, w\rangle$ starting at s is contained in $G' = (V', E')$. Let $P' = \langle u_i, \ldots, u_j\rangle$ be an arbitrary subpath of P. Then

$$\begin{aligned} c(P') &= \sum_{t=i}^{j-1} c(u_t, u_{t+1}) \leq \sum_{t=i}^{j-1} \left(\delta(s, u_{t+1}) - \delta(s, u_t)\right)(1 + \alpha) \\ &\leq \left(\delta(s, u_j) - \delta(s, u_i)\right) \cdot (1 + \alpha), \end{aligned}$$

i.e. $c(P') + \delta(s, u_i) \cdot (1 + \alpha) \leq \delta(s, u_j) \cdot (1 + \alpha)$. Therefore we conclude that P is a relevant path in G. □

Note that we can construct $G' = (V', E')$ efficiently (just calculate the $\delta(s, \cdot)$-values with Dijkstra's algorithm and construct G'). The graph G' will consist exactly of all relevant paths (according to Definition 4.2.3 starting at s, so now we need to count all paths from G'. Note that G' may contain cycles. For example, consider a graph containing cycles G in which all edge costs are 0 which contains a cycle that is reachable from s. Then all edges on this cycle are (almost) tight with respect to $\delta(s, \cdot)$, i.e. G' contains a cycle.

Lemma 4.2.2. *Suppose G' contains a cycle, then all edges on this cycle will have cost* 0.

Proof. Let $C = \langle v = v_1, \ldots, v_j = v\rangle$ be a cycle in G'. Let P be a s, v-path in G'. Then P is relevant and P concatenated with C is also a relevant s, v-path (since it is a s, v-path contained in G'). Therefore it holds that for the subpath C,

$$c(C) + \delta(s, v) \cdot (1 + \alpha) \leq \delta(s, v) \cdot (1 + \alpha),$$

i.e. $c(C) \leq 0$. Since we do not have edge costs of value smaller than 0, we see that $c(C) = 0$ and hence that all edges on the cycle have cost 0. □

Therefore, if we allow no edges of cost 0, the graph G' of almost tight edges is acyclic. Now we can count all relevant paths efficiently with Algorithm 5 of Chapter 2. Hence we solved Problem 3.

Example 4.2.3 (Example, restricted relative range)**.** Consider a graph $G = (V, E)$ consisting of 6 vertices s, x, v, y, w. Let there be directed edges with edge costs as in Figure 4.4.

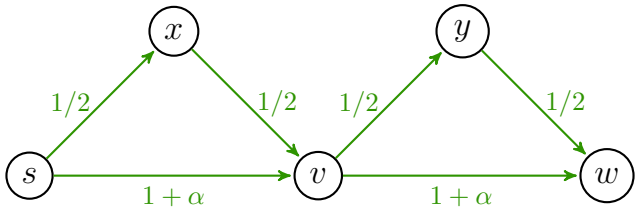


Figure 4.4: Example graph $G = (V, E)$. Note that *all* paths starting from s are 'relevant' paths in the sense of Definition 4.2.3, while not all paths are shortest paths. Note that *all* edges in G are 'almost tight' with respect to $\delta(s, \cdot)$.

Not all paths in G are shortest paths. For example, the path $P = \langle s, v, w \rangle$ of cost $c(P) = (1 + \alpha) \cdot 2 = (1 + \alpha) \cdot \delta(s, w)$ is *not* a shortest path, for $\alpha > 0$. However, it is a 'relevant' path starting from s. This is a simple check left to the reader.

Remark 4.2.2. Note that it is not sensible to use the same trick with 'restricted range' relevance for the *additive notion of relevance*: if we defined 'almost tight edges' with respect to the additive restricted range notion analogously (to the relative restricted range notion of 'almost tight edges'), we would get that an edge (v, w) is almost tight with respect to $\delta(s, \cdot)$ if

$$c(v, w) + \delta(s, v) + \alpha \leq \delta(s, w) + \alpha,$$

and by subtracting both sides of the equation by α we see that we would have defined *tight* edges, so that we do not find any paths within range that are not shortest paths.

Remark 4.2.3. Note that 'relevant' paths within restricted relative range (according to Definition 4.2.3) have the property that each x, y-subpath of an s, w-subpath is 'within multiplicative range' (according to Definition 4.1.1). To see this, suppose $P = \langle s, \ldots, w \rangle$ is a path that is relevant (according to Definition 4.2.3). Suppose that $P' = \langle x, \ldots, y \rangle$ is a subpath of P. Then

$$c(P') + \delta(s, x) \cdot (1 + \alpha) \leq \delta(s, y) \cdot (1 + \alpha),$$

which means that

$$c(P') \leq (\delta(s, y) - \delta(s, x)) \cdot (1 + \alpha) \leq \delta(x, y) \cdot (1 + \alpha),$$

by the triangle inequality. We conclude that P' is a 'within range' path.

Conversely, if every x, y-subpath of an s, w-subpath P is 'within range' (according to Definition 4.1.1 (ii)), then the path P needs not to be 'relevant' within restricted relative range. Consider the example graph from Remark 4.2.1, and consider the path $\langle s, v, w \rangle$. All subpaths are within range, but for the subpath $\langle v, w \rangle$ it does *not* hold that

$$11.5 = c(\langle v, w \rangle) + \delta(s, v) \cdot (1 + \alpha) \leq \delta(s, w) \cdot (1 + \alpha) = 11,$$

so the condition of Definition 4.2.3 is *stronger* than the condition that 'all subpaths must be within range'.

4.3 Restricted relative range notion for multiplicative edge reliabilities

Now we would like to use the approach of the last section in graphs with multiplicative edge reliabilities. Let $G = (V, E)$ be a directed graph with a reliability function $r : E \to (0, 1]$. (If we have a graph with edge reliabilities of 0, then we simply remove these edges). In this section we are not only interested in the maximum reliability paths, but also in paths that are *almost*

of maximum reliability. We give a notion of a 'relevant path', an 'almost tight edge' and an idea to count all relevant paths efficiently.

Note that, if $c = -\log \circ\, r$, we can find all relevant paths for the additive edge cost function c. We write δ_c for the shortest distance function with respect to the costs c and we write δ for the maximum reliability distance function with respect to the reliabilities r. We called an s, v-path P relevant if for each subpath $P' = \langle x, \dots, y \rangle$ of P it holds that

$$c(P') + \delta_c(s, x) \cdot (1 + \alpha) \leq \delta_c(s, y) \cdot (1 + \alpha).$$

Now we apply on both sides $e^{(-\cdot)}$ (which is a strictly decreasing function $\mathbb{R}_{\geq 0} \to (0, 1]$ with inverse $-\log(\cdot)$, to get

$$\begin{aligned} & c(P') + \delta_c(s, x) \cdot (1 + \alpha) && \leq \delta_c(s, y) \cdot (1 + \alpha). \\ \iff\ & e^{-(-\log(r(P')) + \delta_c(s,x)(1+\alpha))} && \geq e^{-\delta_c(s,y)\cdot(1+\alpha)} \\ \iff\ & r(P') \cdot e^{-\delta_c(s,x)(1+\alpha)} && \geq e^{-\delta_c(s,y)(1+\alpha)} \\ \iff\ & r(P') \cdot \delta(s, x)^{1+\alpha} && \geq \delta(s, y)^{1+\alpha}. \end{aligned}$$

This will be our definition of relevance.

Definition 4.3.1 (*Relevant path, multiplicative reliabilities – proposed*)**.** We define a path $P = \langle s, \dots, w \rangle$ to be relevant if for each subpath $P' = \langle x, \dots, y \rangle$ of P it holds that

$$r(P') \cdot \delta(s, x)^{1+\alpha} \geq \delta(s, y)^{1+\alpha}.$$

Similar, we give a definition for almost tight edges, derived in the same way. We start with the definition of an almost tight edge in the additive sense with respect to $c = -\log \circ\, r$, and we derive a suitable definition of an almost tight edge with respect to the reliability function r:

$$\begin{aligned} & c(v, w) + \delta_c(s, v) \cdot (1 + \alpha) && \leq \delta_c(s, w) \cdot (1 + \alpha). \\ \iff\ & e^{-(-\log(r(v,w)) + \delta_c(s,v)(1+\alpha))} && \geq e^{-\delta_c(s,w)\cdot(1+\alpha)} \\ \iff\ & r(v, w) \cdot e^{-\delta_c(s,v)(1+\alpha)} && \geq e^{-\delta_c(s,w)(1+\alpha)} \\ \iff\ & r(v, w) \cdot \delta(s, v)^{1+\alpha} && \geq \delta(s, w)^{1+\alpha}. \end{aligned}$$

This results in the following definition.

Definition 4.3.2 (*Almost tight edge – proposed*)**.** We define an edge (v, w) to be *almost tight* with respect to $\delta(s, \cdot)$ if

$$r(v, w) \cdot \delta(s, v)^{1+\alpha} \geq \delta(s, w)^{1+\alpha}.$$

With these two definitions, an approach similar to the strictly-shortest-path-approach can work.

Fix a vertex $s \in V$. We construct the graph $G' = (V', E')$ that consists of all almost tight edges with respect to $\delta(s, \cdot)$, i.e. let

$$V' := \{v \in V \mid v \text{ is reachable from } s\},$$

and

$$E' := \{(v, w) \in E \ : \ (v, w) \text{ is almost tight with respect to } \delta(s, \cdot)\}.$$

In the following theorem we prove that this graph is exactly consisting of all relevant paths that start at s.

Theorem 4.3.1. *It holds that:*

(i) A path P starting at s is contained in $G' = (V', E')$ if and only if P is a relevant path starting at s in G (according to Definition 4.3.1*).*

(ii) Suppose G' contains a cycle, then all edges on this cycle will have reliability 1.

Proof. Use the cost function $c = -\log \circ\ r$. For this cost-function, Theorem 4.2.1 (for proving (i)) and Theorem 4.2.2 (for proving (ii)) hold. Now apply $e^{-(\cdot)}$ again to get the desired results (as written out above, this is how we obtained the definitions of relevant paths and almost tightness in the multiplicative case). For proving (ii), we use that $-\log(1) = 0$. □

Observe that if we allow no edges of reliability 1, the graph G' of almost tight edges is acyclic. Then we can count all relevant paths efficiently with Algorithm 5.

Remark 4.3.1. Note that the analogous 'additive' respectively 'multiplicative' within restricted range definition (analogous to Definition 4.3.1) for a relevant path makes no sense if we use multiplicative edge costs. These definitions would state that a s, w-path is relevant if for each x, y-subpath P' of P it holds that

$$r(P') \cdot \delta(s,x) - \alpha \geq \delta(s,y) - \alpha \quad \text{resp.} \quad r(P') \cdot \delta(s,x) \cdot (1-\alpha) \geq \delta(s,y) \cdot (1-\alpha),$$

and in both cases we would find only *strictly* maximum reliability paths.

4.4 Results

The notion of '*restricted within range*' paths obtained in this thesis is easy to implement: one only needs to replace the definition of a 'tight edge with respect to $\delta(s,\cdot)$' by the definition of an '*almost* tight edge with respect to $\delta(s,\cdot)$' in the maximum reliability path (resp. shortest path) algorithm and one easily computes the *restricted within range* path graph containing all relevant paths starting at some vertex s. Nevertheless, for our purpose the notion is not very useful, by the following reasons.

(i) We allow edge reliabilities of 1 (on the first edge of a path, see Section 2.4) therefore in the CPB-network there are pairs s, v with $\delta(s,v) = 1$. For those pairs, 'relevant' paths in the sense of Definition 4.3.1 are only strictly maximum reliability paths.

(ii) If, for some $s, v \in V$, it holds that $\delta(s,v) \approx 1$ (for example $\delta(s,v) = 1-\varepsilon$, with ε as in 2.4.1), a very large α is needed to compute relevant paths in the sense of Definition 4.3.1 within a given range $[a,b]$ (for $0 < a < b \leq 1$).

(iii) The interpretation of multiplicative restricted relative range paths (in the sense of Definition 4.3.1) is not clear. The CPB asked for paths within an additive or a multiplicative range, and not for paths that are within range with 'taking powers' in the sense of Definition 4.3.1.

(iv) Brute-force computation of 'almost shortest paths' using depth-first search can be done fast enough, when adjusting the penalty ε% that is levied on each edge after the first edge in a shortest paths (see Chapter 5).

However, it is still interesting to test the notion of 'restricted within range' paths. We do this by considering tax routes from NLD to USA. We assume in all computations in this section that the penalty ε in the sense of Section 2.4.1, equals $\varepsilon = 10^{-12}$. The distance of a maximum reliability path NLD–USA is $0.83627906 - 10^{-12}$, and a maximum reliability NLD–USA-path is for example NLD – DEU – USA. There are 38 maximum reliability paths (which are maximum reliability paths of *shortest length*) from NLD to USA. We first draw the graph containing all *strictly* maximum reliability NLD–USA-paths.[2]

[2]In the CPB-data, all maximum reliability NLD–USA-paths of shortest length are indirect routes. However, the direct route NLD–USA is actually equally as profitable as the indirect routes produced with the CPB-data. This has to do with the construction of the tax-distances by the CPB: in conduit situations some assumptions are made, and that is why the tax-distances in conduit situations are not exact [RL14].

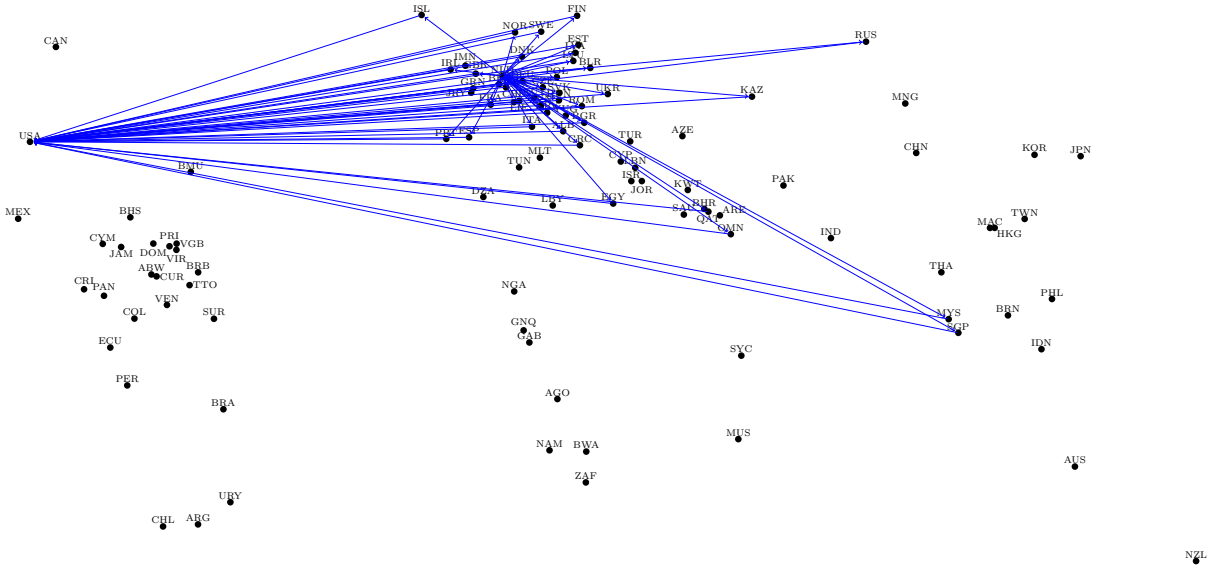


Figure 4.5: The graph containing all *maximum reliability* paths from NLD to USA. Since we added a small penalty ε in the sense of Section 2.4.1, these paths are exactly all maximum reliability paths from NLD to USA *of shortest length*.

We will calculate all 'relevant' paths (in the sense of Definition 4.3.1) that are at most 5% more expensive for companies than the maximum reliability path, i.e. we calculate all relevant paths in the interval $[X \cdot (1 - 0.5), X]$, where

$$X = \delta(\text{NLD, USA}) \approx 0.83627906,$$

i.e. we calculate all relevant paths within the interval $[0.79446511, 0.83627906]$. By solving

$$0.83627906^{1+\alpha} = 0.79446511,$$

we find $\alpha = \log(0.79446511)/\log(0.83627906) - 1 \approx 0.28688661$. We find 871 *relevant* paths. We draw the *relevant* path graph containing all *relevant* NLD,USA-paths in the sense of Definition 4.3.1.

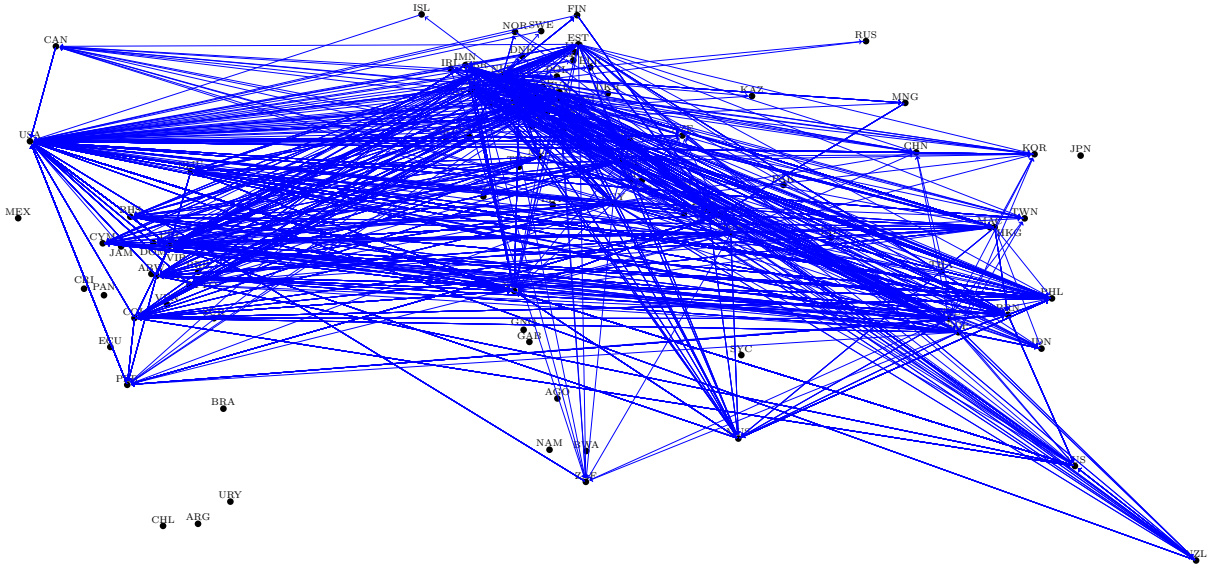


Figure 4.6: The graph containing all *restricted relative range* paths from NLD to USA (i.e. all relevant paths in the sense of Definition 4.3.1) that are at most 5% more expensive than the strictly maximum reliability path.

Here we end the discussion of 'restricted' within range paths. Constructing the relevant path graph and counting relevant paths can be done very fast, within seconds. However, finding or counting *all* (not necessarily relevant according to Definition 4.3.1) paths that are within range of 5% the maximum reliability paths takes *very* long. The computer can think for days, since there are a lot of reliability $1-\varepsilon$-edges (with ε as in Section 2.4.1) and paths along those edges remain 'within range'. Even if we increase the penalty ε (as in Section 2.4.1) the algorithm does not run fast. For example, the experiment from Section 5.4.1 takes around 10 minutes.

In the next chapter, we will try to compute all 'within range' paths. First we try a brute-force depth-first search approach. After that we shortly consider *Yen's algorithm*, an algorithm to compute the first K shortest paths from one vertex to another vertex in a graph.

Chapter 5

Computing within range paths

In this chapter we try to compute all 'within range' paths, paths that have reliability (or cost) *almost* as large as the maximum reliability path (or cost *almost* as small as the shortest path). We will try two approaches: a brute-force depth-first search and *Yen*'s algorithm: an algorithm to find the K shortest simple s,t-paths in a graph.

5.1 Brute-force computation of within range paths

To compute *all* s,t-paths in a graph $G=(V,E)$, one can use depth-first search. Each path is found in time $\mathcal{O}(|V|+|E|)$, but there may be exponentially many paths, so we cannot list them in polynomial time. Even *counting* s,t-paths we cannot do in polynomial time (unless G is a *directed acyclic graph*), as this is $\#P$-hard (see Example 1.3.5).

When we calculate brute force within range paths for the CPB, we use depth-first search (see Chapter 1). Algorithm 10 is an example of a depth-first search to find all simple s,t-paths in a graph.

In the CPB-network we use Algorithm 10 on the graph G_s with source country s (where the edge reliabilities are as in Section 2.4) with some adaptations. Suppose we want to find s,t-paths with reliability at least X, and that we already know the distances $\delta_s(\cdot,\cdot)$ in the graph G_s.

1. We remove all edges (u,v) with $r(u,v)<X$
2. During the search, we store (and keep track of) the reliability of $P=\langle s,\ldots,v\rangle$. As soon as $r(P)\cdot\delta_s(v,t)<X$ we do not search deeper and continue at the next-to-last vertex of P.
3. The graph G_s contains *many* reliability 1 edges (around 2500 edges of reliability 1, depending on the starting vertex). Furthermore, there are in total 6214 pairs s,t with $s\neq t$ such that $\delta_s(s,t)=1$, which means that for $100\cdot 6214/(107\cdot 108)=53.77\%$ of the s,t-pairs companies can send money without paying taxes from s to t.

   Since the graph contains many edges with reliability 1, there are *many* within range paths even if we search for paths within a small range of the shortest path. Therefore when computing within range paths we will increase the penalty ε from Section 2.4.1. For each experiment we do brute-force (in the Results section), we will indicate the size of the 'penalty' ε (in the sense of Section 2.4.1).
4. We could also put a maximum on the depth of the search (a maximum on the number of conduit countries), this speeds up the procedure. However, in the experiments of this chapter we did not use this approach (although using the approach of Section 5.4.1 already implicitely puts a maximum on the number of conduit countries).

When considering within range paths we can also compute (weighted) betweenness centrality. The variables σ_{st} and $\sigma_{st}(u)$ now denote the number of within range paths from $s\in V$ to $t\in V$

respectively the number of within range paths from s to t passing through $u \in V$, where $s \neq u \neq t$. The definition of the betweenness is then the same as in Definition 2.3.1 (but using the new 'within range' σ-values).

Input: Directed graph $G = (V, E)$, source vertex s, sink t.
Output: All s, t-paths.

Initialize: v is *white* for every $v \in V$, temporary path $P := \langle s \rangle$, List $L = \emptyset$.

DFS-visit(s)

Procedure DFS-visit(u)
    Color vertex u green
    foreach neighbour v of u **do**
        if $v = t$ **then**
            append v to P
            add (a copy of) P to List L
            remove v from (the original) P
        end
    end

    foreach neighbour $v \neq t$ of u **do**
        if Color(v) = white **then**
            Append v to P
            DFS-visit(v)
            Color v white
            Remove v from P
        end
    end
end

return List L

Algorithm 10: Depth-first search to find all s, t-paths in a graph. Note that, by adapting the first loop, one could also find all paths from s to *each* $v \in V$, by keeping lists L_v for each $v \in V$ and adding P to L_v in this loop.

In the section 'Results' of this Chapter, Section 5.4, we will give results and interpretations of some brute-force experiments in the CPB-network.

5.2 The additive K-th shortest path problem

Another idea for computing within range paths is using *Yen's algorithm* [Yen71]. This is an algorithm for computing the K shortest simple s, t-paths in a graph. We can use this for computing within range paths: increase K by one iteratively, until the cost of the K-th shortest path exceeds the range.

In this section we will look at the K-th shortest path problem, with [Yen71] as reference. We will formulate the K-th shortest path problem in the additive way. Later we make adaptations to use the algorithm for finding maximum reliability paths. The setting is as follows.

Let $G = (V, E)$ be a directed graph, and $c : E \to \mathbb{R}$ be a cost function[1]. If $P = \langle u_1, \ldots, u_k \rangle$ is a path, then we define the cost of P as $c(P) := \sum_{i=1}^{k-1} c(u_i, u_{i+1})$. We assume that G has no cycles of negative cost. In this section we aim to solve the following problem.

Problem 4 (K-th shortest path problem)**.** Fix two vertices $s, t \in V$. Find the K shortest *simple* paths from vertex s to vertex t in G.

One — not very efficient, but easy to understand — way of solving this problem is found by Pollack [Pol61].

Algorithm 5.2.1 (*Pollack's Algorithm*)**.** Suppose that we have found the $K-1$ shortest paths. Then we set in each of the 1st, 2nd, $\ldots$, $(K-1)$-st shortest paths the distance of one edge to infinity and we solve a shortest-path problem for each such case (i.e. if the first $(K-1)$ shortest paths coincidentally all have size a (by *size* we mean the number of vertices in the path), then we need to solve a^{K-1} shortest path problems). The shortest of all the resulting paths is our K-th shortest path.

This method has the clear disadvantage that we need to solve a large amount of shortest-path problems: the number of shortest path problems increases exponentially with K. (As we have seen: if the first $(K-1)$ shortest paths all have size a, then we are required to solve a^{K-1} shortest path problems). Therefore this method is only of use for small values of K. In general, Pollack's Algorithm is 'computationally overburdening' (as Yen notes in [Yen71]). Therefore we will consider another, much more efficient algorithm found by Yen [Yen71]: *Yen's algorithm.* First we introduce some notation.

(i) We denote the vertices of G by $(1), \ldots, (n)$ and we let vertex (1) be the source vertex and (n) be the sink vertex.

(ii) We define size(P) to be the number of *vertices* on the path P.

(iii) For $k = 1, \ldots, K$, let $P^k = \langle (1), (2^k), (3^k), \ldots, ((\text{size}(P^k) - 1)^k), (n) \rangle$ be the k-th shortest path from (1) to (n), where $(2^k), (3^k), \ldots, ((\text{size}(P^k) - 1)^k)$ denote respectively the 2nd, 3rd, ..., next-to-last vertex of the k-th shortest path.

(iv) Suppose $k \in 2, \ldots, K$. Suppose $i \in \{1, 2, \ldots \text{size}(P^{k-1}) - 1\}$. We define a *deviation* from P^{k-1} at the i-th vertex (vertex (i^{k-1})) to be a shortest path that coincides with P^{k-1} from vertex (1) to (i^{k-1}), then 'deviates' to a vertex that differs from all the $(i+1)$-th vertices of the P^j that have the same subpath from the 1st to the i-th vertex as P^{k-1}, with $j = 1, \ldots, k-1$, and reaches (n) by a shortest path without passing any vertex that is already passed in the first part of the path (the $\langle (1), \ldots, (i^{k-1}) \rangle$-part). By construction this is a simple path.

Now we are ready to give a description of the algorithm. After that, we prove that the algorithm is correct and we prove a bound on the running time.

Algorithm 5.2.2 (*Yen's Algorithm*)**.** We will describe first the initialization step of the algorithm. After that we describe what the algorithm does at the k-th iteration.

(1.) First we determine P^1 by using an efficient shortest path algorithm. If all edge costs are nonnegative, we use Dijkstra's algorithm. Otherwise we can use the Bellman-Ford algorithm (note that G does not contain cycles of negative length by assumption). For

[1] We did not cover the Bellman-Ford Algorithm here, which is needed when negative edge costs are allowed. The Bellman-Ford Algorithm (with running time $\mathcal{O}(|V||E|)$ just relaxes all edges $n-1$ times, such that it is certain that all edges are relaxed along shortest paths. Possibly one post-processing step can be added to determine negative cost cycles: if the tentative distances change after all edges are relaxed for the n-th time, the graph contains a negative cycle.

these algorithms, we refer to [Schä13]. If we have K or more paths of this shortest length, we are done. Otherwise we choose an arbitrary shortest path and we store it in List A. The rest of the shortest paths we store in List B (if there was only one path then List B still remains empty). During the execution of this algorithm, List A is the list of k-shortest-paths, and List B is the list of candidates for $(k+1)$-shortest paths.

(k.) We will compute P^k, assuming that we already know $P^1, \ldots, P^{k-1}$. Now, for $i = 1, 2, \ldots, \text{size}(P^{k-1}) - 1$, we carry out the following three steps:

(a.) Look if the subpath of the first i vertices of P^{k-1} is the same as the subpath of the first i vertices of $P^1, \ldots, P^{k-2}$. We set $c((i^{k-1}), ((i+1)^j)) := \infty$ for all $j = 1, \ldots, k-1$ for which this is the case, where we bear in mind that (i^{k-1}) is the i-th vertex of the path P^{k-1}, and $((i+1)^j)$ is the $(i+1)$-th vertex of the path P^j. We need to store the original values of the costs because we will reset them after the whole iteration. Proceed to the next step.

(b.) Find a shortest path from (i^{k-1}) to (n) with a shortest path algorithm, where we only consider vertices not contained in the subpath $\langle (1), \ldots, (i^{k-1}) \rangle$ of P^{k-1}, which we denote by R_i^k. The resulting shortest path we denote by S_i^k. If there is more than one candidate for S_i^k, we choose one arbitrarily. Proceed to the next step.

(c.) Find a 'deviation' from P^{k-1} at the i-th vertex by concatenating the paths R_i^k and S_i^k. Add A_i^k to List B, if it is not already contained in List B. We only need to store the $K - k + 1$ shortest paths in List B.

We choose one of the paths in List B of minimum length, we call this path P^k and we move it to List A. The rest of the paths in List B remain there. We set the edge costs to their original values proceed to iteration $k+1$.

We now prove that the algorithm is correct and we give a bound on the running time.

Theorem 5.2.1. *Algorithm 5.2.2 solves the K-th shortest path problem in time $\mathcal{O}(Kn(m + n\log n))$ for graphs with nonnegative edge costs and in time $\mathcal{O}(Kn^2m)$ if we also allow negative edge costs.*

Proof. First we prove that the algorithm terminates in time $\mathcal{O}(Kn(m+n\log n))$ for graphs with nonnegative edge costs and in time $\mathcal{O}(Kn^2m)$. We examine the total time that the algorithm is in each of the three steps (a.),(b.) and (c.).

(a.) The comparisons of the subpaths R_i^k of all the paths $P^1, \ldots, P^{k-1}$ can be done in total time $\mathcal{O}(K \cdot n)$, which is negligible compared to the time needed in (b.).

(b.) In each iteration we solve a shortest path problem on a subgraph of G. If the edge-costs are nonnegative, we solve this with Dijkstra's algorithm in time at most $\mathcal{O}(m + n\log n)$. If we also allow negative edge costs, we solve this in time at most $\mathcal{O}(mn)$. This gives a total time of $\mathcal{O}(Kn \cdot (m + n\log n))$ respectively $\mathcal{O}(Kn^2m)$.

(c.) In this step we concatenate two paths and store the resulting path in List B, provided that it will belong to the $K-k+1$ shortest paths in List B. This can be done in time negliglible compared to the time used in step (b.).

We conclude that the running time (in the case of non-negative edge costs) of the algorithm is bounded by $\mathcal{O}(Kn(m + n\log n))$, and in case we allow negative edge costs $\mathcal{O}(Kn(mn)) = \mathcal{O}(Kn^2m)$. Therefore the algorithm ends in the claimed running time.

It remains to prove that the algorithm is correct. For some $k = 2, \ldots, K$, suppose that we know $P^1, \ldots, P^{k-1}$. We want to compute a k-th shortest path P^k. Note that this path must be a deviation of P^{k-1} (since P^{k-1} starts at (1) and the k-th shortest path can not have

Input: directed graph $G = (V, E)$, cost function $c : e \to \mathbb{R}$, natural number K, source vertex (1), sink vertex (n).
Output: K shortest simple paths $P^1, \ldots, P^K$ of non-decr. length from (1) to (n).

Initialize: List A $= \{P^1\}$ (found with a shortest path algorithm), List B $= \emptyset$.
for $k = 2, \ldots, K$ **do**
  for $i = 1, \ldots, \text{size}(P^{k-1}) - 1$ **do**
    - Let (i^{k-1}) be the i-th vertex of P^{k-1}.
    - Let R_i^k the subpath of the first i vertices of P^{k-1}.
    - **if** R_i^k is the same as the subpath of P^j consisting of the first i vertices **then** set $c((i^j), ((i+1)^j)) := \infty$, **for** $j = 1, \ldots k-1$.
    - Compute a shortest path from (i^{k-1}) to (n) not crossing other vertices from R_i^k using a shortest path algorithm.
    - Concatenate this path with R_i^k.
    - Store the resulting path in List B (provided that it is not already stored in List B before and that it will belong to the $K - k + 1$ shortest paths in List B).
  end
  - Restore the original graph G (by restoring the values of c).
  - Move the path with the smallest cost from List B to List A. This path is P^k.
  - **if** this path does not exist **then break**
end
return List A $= \{P^1, \ldots, P^K\}$.

Algorithm 11: The K-th shortest path algorithm found by Yen in [Yen71] to solve Problem 4.

all vertices in common with a path in List A). Hence, for computing P^k it is only necessary to look at the vertices of P^{k-1} for a shortest deviation of P^{k-1} at this vertex, and then find from all these shortest deviations the one of shortest length, which will be P^k. Yen's algorithm (Algorithm 5.2.2) exactly does this. Therefore the algorithm is correct. □

Remark 5.2.1. Algorithm 5.2.2 uses $\mathcal{O}(n^2 + Kn)$ storage space.

Proof. We need $\mathcal{O}(n^2)$ storage space to store all the c-values during the execution of the algorithm. Also we need at most Kn adresses to store the entries of List A and List B (where we note that, at the k-th iteration of the algorithm, we only store the $K - k + 1$ shortest paths in List B and then we move the shortest of them to List A. Therefore List A and List B will together contain no more than K paths. □

Example 5.2.1. We consider an example. Suppose that we would like to find $K = 3$ shortest paths in the following graph.

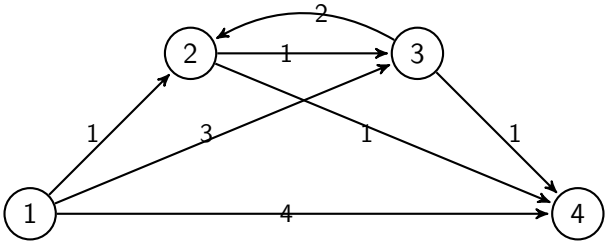


Figure 5.1: An example graph. We would like to find the $K = 3$ shortest paths from vertex (1) to vertex (4).

We begin with the iteration $k = 1$. In this iteration we compute a shortest path from vertex (1) to vertex (4) with Dijkstra's Algorithm. Note that all edge costs are nonnegative.

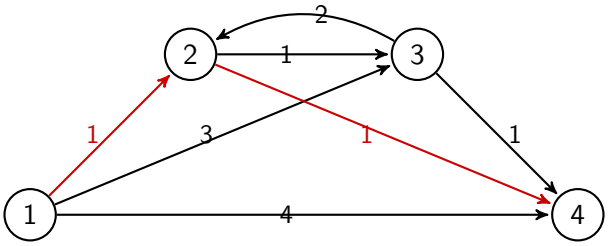


Figure 5.2: The first iteration. We compute a shortest path (red) with Dijkstra's algorithm. This path P^1 has cost 2.

Now we store List A $= \{P^1\} = \{\langle(1),(2),(4)\rangle\}$, the red path in the above picture, and we continue to iteration $k = 2$. We consider all vertices on the path P^1 except the end vertex in order.

(1) First we consider vertex (1) and we remove the edge leaving (1) in P^1 from our graph. Then we compute a shortest path from (1) to (4) (blue in the image below) using Dijkstra's algorithm. We store this path in List B.

(2) Next we consider vertex (2) and we remove the edge leaving (2) in P^1 from our graph. Then we compute a shortest path from (2) to (4) not crossing (1) with Dijkstra's Algorithm and we concatenate it with the subpath $\langle(1),(2)\rangle$ of P^1. The resulting path (green in the figure below) we store in List B.

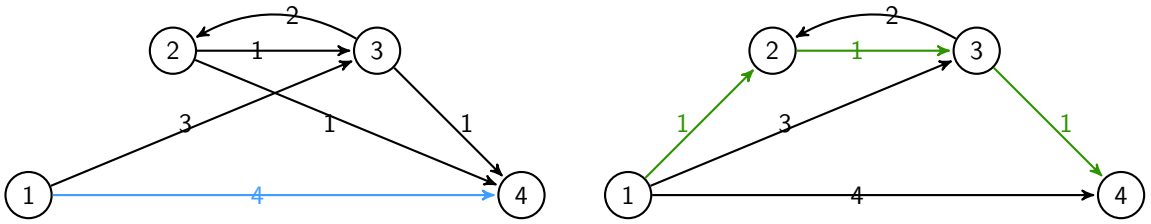


Figure 5.3: Left: Begin of the iteration $k = 2$. First, $i = 1$. We compute a shortest path (blue) with Dijkstra's algorithm. This path has cost 4. Right: Now, $i = 2$. We compute the shortest path from (2) to (4) not crossing (1) with Dijkstra's algorithm. Then we concatenate it with the path $\langle(1),(2)\rangle$. The resulting path (green) has cost 3.

Our List B consists of the green and the blue path, i.e.:

$$\text{List B} = \{\langle(1),(2),(3),(4)\rangle,\ \langle(1),(4)\rangle\}.$$

We choose the one of lowest cost (the green path $\langle(1),(2),(3),(4)\rangle$, of cost 3) to be P^2 and we move it to List A. Note that

$$\text{List A} = \{P^1,\ P^2\} = \{\langle(1),(2),(4)\rangle,\ \langle(1),(2),(3),(4)\rangle\}.$$

And List B now only consists of the path $\langle(1),(4)\rangle$. We continue to iteration $k = 3$. We consider all vertices on the path P^2 except the end point in order.

(1) We first note that paths P^1 and P^2 have vertex (1) as 1st vertex. Therefore we remove the edges leaving (1) in P^1 in P^2 from our graph. Then we compute a shortest path from (1) to (4) (blue in the image below) using Dijkstra's algorithm. This path is already contained in List B, therefore we do not need to store it. (see the image below, left part). We restore the edges in the graph.

(2) We note that paths P^1 and P^2 have vertex (2) as 2nd vertex. Therefore we remove the edges leaving (2) in P^1 and P^2 from our graph. Then we compute a shortest path from (2) to (4) not crossing (1). This path does not exist. We continue to the next iteration (see the image below, right part).

(3) Path P^1 does not have a 3rd vertex, and path P^2 has (3) as 3rd vertex. We remove edge $\langle(3),(4)\rangle$ from the graph and we compute a path from (3) to (4) without crossing the first 3 vertices of P^2. This path does not exist. (This step is not depicted in one of the figures)

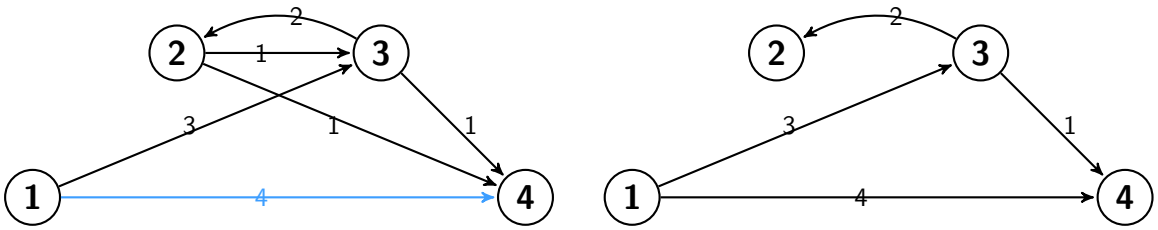


Figure 5.4: Left: Begin of the iteration k=3. First, $i = 1$. We compute a shortest path (blue) with Dijkstra's algorithm. This path has cost 4. We store it in List B. Right: Now, $i = 2$. We delete the edges leaving (2) from P^1 and P^2. Then we compute the shortest path from (2) to (4) not crossing (1) with Dijkstra's algorithm. This path does not exist.

Now, List B only consists of the path $\langle(1),(4)\rangle$. Therefore this will be our P^3 and we move it to List A. We now have computed $K = 3$ shortest paths:

$$\begin{aligned} P^1 &= \langle(1),(2),(4)\rangle && \text{of cost 2,} \\ P^2 &= \langle(1),(2),(3),(4)\rangle && \text{of cost 3,} \\ P^3 &= \langle(1),(4)\rangle && \text{of cost 4,} \end{aligned}$$

as desired.

5.3 The K-th maximum reliability path problem.

In this section we adapt Yen's algorithm to the maximum reliability setting with multiplicative edge reliabilities. Let $G = (V, E)$ be a directed graph, and $r : E \to (0, 1]$ be a reliability function. If $P = \langle u_1, \ldots, u_k\rangle$ is a path, then we define the reliability of P as $r(P) := \prod_{i=1}^{k-1} r(u_i, u_{i+1})$. In this section we aim to solve the following problem.

Problem 5 (K-th maximum reliability path problem)**.** Fix two vertices $s, t \in V$. Find the K maximum reliability simple paths from vertex s to vertex t in G.

We adapt Yen's algorithm [Yen71]. This adaptation is straightforward. We only need to use the multiplicative version of Dijkstra's algorithm for finding a maximum reliability path in a graph, and we need to scan List B each time for a maximum reliability path. For the rest the algorithm is the same (and the proofs are too). Therefore Problem 5 is easily solved.

5.4 Results

The K-th shortest path algorithm was shortly tested on the CPB-network, but computing only 2000 NLD – USA-paths was done not very fast (it took around 2 minutes, for only this one pair of countries). The K-shortest path algorithm computes a new path in $\mathcal{O}(n \cdot (m + n \log n))$, since it uses n shortest path algorithms to find one new path.

However, the CPB-network contains many reliability 1-edges and hence many paths are within range paths, so depth-first search finds a new within range path very fast. The problem is that there are *many* within range paths. Since depth-first search finds new paths fast, the original idea of using the K-th shortest path algorithm for computing within range paths will not be investigated further in this section. This section contains some selected *within range* experiments.

5.4.1 Sensible idea for computing within range paths

In the case of within range-paths, we could multiply each edge reliability (except for the first edge) by $(1-\alpha)^{(1/j)} > 0$, for j a natural number and $\alpha > 0$ a real number. Then we compute all paths of reliability at least $\delta(s,t) \cdot (1-\alpha)$, for all pairs s, t, with the original distances $\delta(\cdot,\cdot)$. By following this procedure, it holds that:

1. All shortest paths of length at most $j+1$ remain 'within range paths'.
2. All paths of length 1 with reliability *at least* $\delta \cdot (1-\alpha)$ are computed.
3. All paths of length k with reliability *at least*

$$\delta \cdot \frac{(1-\alpha)}{(1-\alpha)^{\frac{k-1}{j}}},$$

for $k = 1, \ldots, j+1$, are computed.

We will try this approach in the CPB-network.

Example 5.4.1. If we set $\alpha = 0.05$ and $j = 3$, then

$$\begin{aligned}
\frac{(1-\alpha)}{(1-\alpha)^0} &= 1-\alpha &&= 0.95\\
\frac{(1-\alpha)}{(1-\alpha)^{\frac{1}{3}}} &= (1-\alpha)^{\frac{2}{3}} &&\approx 0.96638253\\
\frac{(1-\alpha)}{(1-\alpha)^{\frac{2}{3}}} &= (1-\alpha)^{\frac{1}{3}} &&\approx 0.98304757\\
\frac{(1-\alpha)}{(1-\alpha)^{\frac{3}{3}}} &= \frac{(1-\alpha)}{(1-\alpha)} &&= 1.
\end{aligned}$$

This means that all paths of length 1 with reliability at most 5% less than the reliability of the maximum reliability path are computed. All paths of length 2 with reliability at most $100 \cdot (1 - 0.099638) = 3.3617\%$ less than the reliability of the maximum reliability path are computed, etc. The paths of length 4 that are computed are only strictly maximum reliability paths.

If we do this experiment on the CPB-network, we get the following results. We find 155,724,338 within range paths (in total). The top 10 of countries is as in the following table.

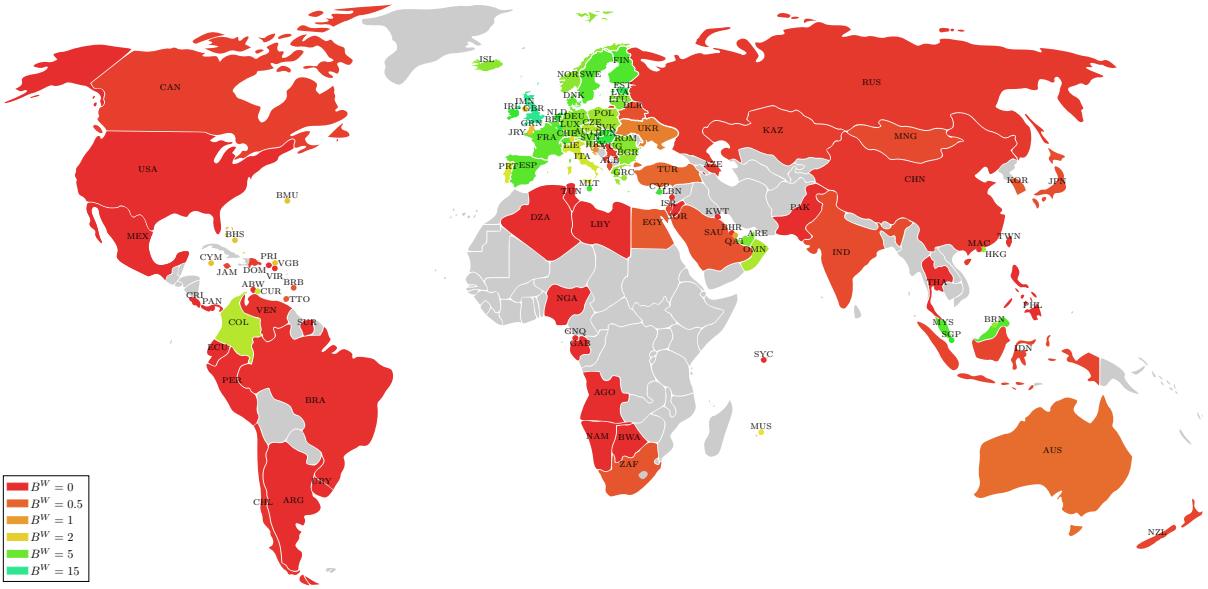


Figure 5.5: The within range experiment of Section 5.4.1, with $\alpha = 0.05$ and $j = 3$. Total number of paths: 155,724,338.

Note that the betweenness values are generally larger than the betweenness values where we only consider *strictly* shortest paths. This makes sense: if we allow a small range, a lot of paths are generated. Many of them pass through conduit countries. Furthermore, we note that countries in the intersection of the strongly connected components (see Chapter 3) have a high betweenness if we allow a small range.

Position	Country u	$B^W(u)$
1	GBR	15.29057
2	EST	10.94572
3	LUX	10.53155
4	HUN	9.75630
5	NLD	9.43637
6	SGP	8.67798
7	SVK	8.51093
8	IRL	8.46756
9	CYP	8.40420
10	MLT	8.06937

Position	Country u	$B(u)$
1	GBR	17.45276
2	CYP	12.91591
3	NLD	12.88430
4	MLT	11.40787
5	EST	11.21784
6	HUN	11.13852
7	SGP	10.51320
8	MYS	8.62411
9	SVK	8.36982
10	LUX	8.32703

Table 5.1: The 10 countries with the highest *weighted*, respectively *unweighted* betweenness centrality values with the experiment as described in Section 5.4.1.

We evaluate also the edge betweenness (weighted and unweighted). Those values tend to be somewhat smaller than in the strictly maximum reliability path case. When considering within range paths, more paths are computed and the flow is divided over more edges: the edge flows tend to get smaller. A remarkable fact is that the edge USA–NLD is the edge with the second-largest (weighted) edge flow in this experiment. Note that the edge CHN-USA is not in the top 5 of edges anymore, while the direct route CHN–USA *is* a maximum reliability path[2]. When allowing a large range, many (non-direct) within range CHN–USA-paths are found and the flow from CHN to USA is equally spread over all those paths. Hence, the edge flow on the edge CHN–USA decreases.

[2]USA and CHN are the two countries with the largest GDP.

Pos. (weight)	Edge (u, v)	$B^W[(u, v)]$
1	USA – GBR	2.35891
2	USA – NLD	1.80192
3	IDN – GBR	1.73166
4	EST – CHN	1.71722
5	LUX – CHN	1.71170

Pos. (unweight.)	Edge (u, v)	$B[(u, v)]$
1	NLD – EGY	1.41745
2	BRB – GBR	1.32511
3	IDN – GBR	1.02019
4	ZAF – CYP	0.99798
5	TWN – SGP	0.98332

Table 5.2: The 5 edges with the highest (weighted/unweighted) edge betweenness (flow) with the experiment as described in Section 5.4.1.

In the next section we will use a slightly different approach for computing the within range paths.

5.4.2 Additive and multiplicative within range paths

Another approach is also possible. We *first* increase the penalty ε in the sense of Section 2.4.1, to reduce the number of within range paths (otherwise all paths via edges of reliability 1 are within range), so that the depth-first search finishes faster. *In the resulting network*[3], we compute all s, t-paths P that are within additive range $r(P) \geq \delta(s, t) - \alpha$, or multiplicative range $r(P) \geq (1 - \alpha) \cdot \delta(s, t)$. It might be justified to add a larger penalty ε (in the sense of Section 2.4.1) *before* calculating the distances: companies make costs when sending money through an additional conduit country. These costs can be reflected in the distances.

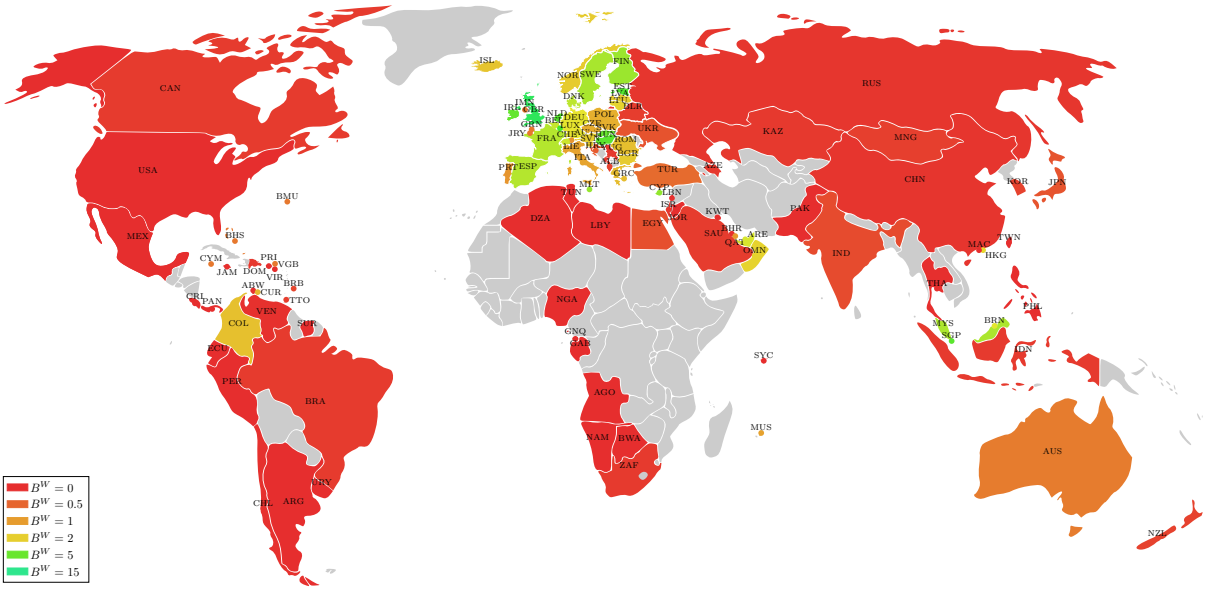


Figure 5.6: Within additive range paths, with $\alpha = 0.005$ and $\varepsilon = 0.005$. This is the version of within range that the CPB used in their report [RL14]. Total number of paths: 2,324,679.

Note that maximum reliability paths do not have to be paths-within range with this approach, since ε (the penalty from Section 2.4.1) is not very small. The CPB used the approach of this section in [RL14], but only for one version of within range (within additive range, with $\varepsilon = 0.005$ and $\alpha = 0.005$). In this section we perform some experiments within additive and multiplicative range, using different values for α and ε. For paths within *additive* range, we get the following results.

[3]The distances are now computed in the resulting network.

Position	Country u	$B^W(u)$
1	GBR	14.28458
2	EST	9.79146
3	LUX	9.61353
4	HUN	8.78620
5	NLD	8.68236

(a) $\alpha = 0.005$, $\varepsilon = 0.0025$. Number of paths: 135,217,133

Position	Country u	$B^W(u)$
1	GBR	11.75287
2	LUX	7.29783
3	EST	6.50612
4	NLD	6.23390
5	SGP	5.96253

(b) $\alpha = 0.005$, $\varepsilon = 0.005$. Number of paths: 2,324,679.

Position	Country u	$B^W(u)$
1	GBR	14.26947
2	EST	9.78262
3	LUX	9.60152
4	HUN	8.77161
5	NLD	8.66874

(c) $\alpha = 0.01$, $\varepsilon = 0.005$. Number of paths: 136,301,035.

Position	Country u	$B^W(u)$
1	GBR	11.67190
2	LUX	7.22259
3	EST	6.36515
4	NLD	6.19838
5	HUN	5.85822

(d) $\alpha = 0.01$, $\varepsilon = 0.01$. Number of paths: 2,312,028.

Position	Country u	$B^W(u)$
1	GBR	14.29684
2	EST	9.95862
3	LUX	9.73517
4	HUN	8.79714
5	NLD	8.59876

(e) $\alpha = 0.05$, $\varepsilon = 0.025$. Number of paths: 169,365,696.

Position	Country u	$B^W(u)$
1	GBR	10.44494
2	LUX	6.56641
3	EST	5.80208
4	NLD	5.55616
5	SGP	5.47544

(f) $\alpha = 0.05$, $\varepsilon = 0.05$. Number of paths: 2,175,747.

Table 5.3: Experiment CPB within range. Range α is *additive*.

We performed the same experiments also within *multiplicative* range instead of additive range. The rankings do not differ much. The ratio of α to ε is more important than the size of ε and α, and this ratio is also more important than the choice between 'additive or multiplicative' within range paths. There are many reliability 1 paths in the CPB-network (when not taking into account a penalty ε), and when the ratio of α to ε is large, a lot of these paths are considered 'within range'.

The number of paths 'within range' that is found by the algorithm is larger in the additive case than in the multiplicative case. This is because the range is larger: in the within multiplicative range case, the range is at least $(1-\alpha)\cdot\delta(s,t) \geq \delta(s,t)-\alpha$, and the latter is the lower bound of the reliability of paths within range when we consider paths witin additive range. We observe that the rankings (top-5) of the countries are quite similar.

Position	Country u	$B^W(u)$
1	GBR	14.25050
2	EST	9.76523
3	LUX	9.58995
4	HUN	8.75584
5	NLD	8.65784

(a) $\alpha = 0.005$, $\varepsilon = 0.0025$. Number of paths: 78,221,169.

Position	Country u	$B^W(u)$
1	GBR	11.72503
2	LUX	7.25856
3	EST	6.43782
4	NLD	6.20201
5	HUN	5.91037

(b) $\alpha = 0.005$, $\varepsilon = 0.005$. Number of paths: 2,321,755.

Position	Country u	$B^W(u)$
1	GBR	14.19071
2	EST	9.71982
3	LUX	9.55687
4	HUN	8.70211
5	NLD	8.61654

(c) $\alpha = 0.01$, $\varepsilon = 0.005$. Number of paths: 78,011,715.

Position	Country u	$B^W(u)$
1	GBR	11.61612
2	LUX	7.21788
3	EST	6.34867
4	NLD	6.19902
5	SGP	5.85855

(d) $\alpha = 0.01$, $\varepsilon = 0.01$. Number of paths: 2,310,052.

Position	Country u	$B^W(u)$
1	GBR	14.19289
2	EST	9.88632
3	LUX	9.70464
4	HUN	8.67981
5	NLD	8.55359

(e) $\alpha = 0.05$, $\varepsilon = 0.025$. Number of paths: 75,644,968.

Position	Country u	$B^W(u)$
1	GBR	10.36691
2	LUX	6.57623
3	EST	5.75759
4	NLD	5.51548
5	SGP	5.44295

(f) $\alpha = 0.05$, $\varepsilon = 0.05$. Number of paths: 2,109,729.

Table 5.4: Experiment CPB within range. Range α is *multiplicative*.

In the last experiment of this section, we consider within multiplicative range paths with $\alpha = 0.01$ and $\varepsilon = 0.0033333$. The depth-first search takes around 97 minutes to calculate the paths (the other brute-force experiments of this section take at most 20 minutes). The number of within range paths is now 2,540,053,489, and the ranking, according to (weighted) betweenness, is as follows.

Position	Country u	$B^W(u)$
1	GBR	16.86922
2	EST	12.75982
3	LUX	11.70391
4	HUN	11.55565
5	NLD	10.93764
6	CYP	10.55827
7	MLT	10.33779
8	SVK	10.10477
9	SGP	9.78214
10	IRL	9.63753

Position	Country u	$B(u)$
1	GBR	19.82831
2	CYP	15.62582
3	NLD	14.88468
4	MLT	14.17225
5	EST	13.95693
6	HUN	13.61274
7	SGP	12.17747
8	SVK	10.60266
9	MYS	10.36432
10	LUX	10.34881

Table 5.5: $\alpha = 0.01$, $\varepsilon = 0.0033333$. Number of paths: 2,540,053,489.

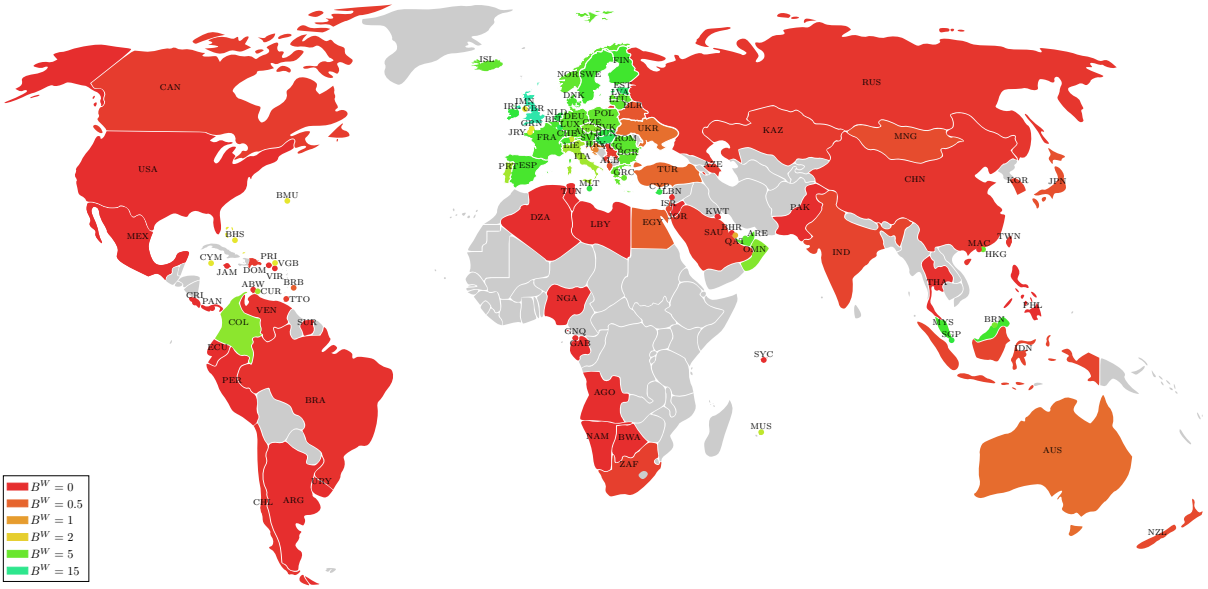


Figure 5.7: Within multiplicative range paths, with $\alpha = 0.01$ and $\varepsilon = 0.0033333$. Total number of paths: 2,540,053,489.

The betweenness values are larger than in any other experiment we did before: the ratio of α to ε is large, so *many* within range paths are computed, passing through conduit countries (a larger share of the paths is non-direct and passes through many conduit countries).

In general the rankings of the within range experiments are similar to the ranking where only strictly shortest paths are taken into account (see Chapter 2), although countries in the top-10 are sometimes in different positions. Great Britain ranks first in every experiment we did.

Here we end the discussion of within range paths. In the next section we will consider the following problem: suppose a country can change the tax rate on k of its outgoing edges. Which k edges (and which tax-rates) must this country choose, in order to maximize the total amount of money that companies will send *through* this country (as a conduit country)?

Chapter 6

Betweenness: maximizing the betweenness of one node

Let $G = (V, E)$ be a directed graph, with additive edge costs $c : E \to \mathbb{R}_{\geq 0}$, such that G contains no cycles of cost 0, also after setting the edge cost of the outgoing edges of one node $u \in V$ to 0. Some (trivial) examples of graphs G that fulfill this condition are:

- A directed graph G with *positive* edge costs $c : E \to \mathbb{R}_{>0}$.
- A directed graph G with *positive* edge costs $c : E \to \mathbb{R}_{>0}$ except for the outgoing edges of the node $u \in V$, those edge costs may also be zero.
- An *acyclic* directed graph G with nonnegative edge costs $c : E \to \mathbb{R}_{\geq 0}$.

The vertices in V in our graph represent 'countries'. Suppose we choose one country u in our network and we write $N^+(u)$ for the set of outneighbours[1] of u in G. The country u can change a tax treaty with *one* other country $v \in N^+(u)$: a country can change the edge cost of one outgoing edge. Which country v would be the 'best' to choose by u? We will investigate this question, where we look at two different objectives:

(i) Maximize the flow through country u.
(ii) Maximize the tax country u receives as a conduit country.

For objective (i) we will prove that it is enough to choose a country v such that the betweenness of u is maximized. Objective (ii) requires more consideration, as we will see. We will also look at the case that not *one*, but k treaty partners can be chosen – with k an integer and $1 \leq k \leq |N^+(u)| \leq n - 1$. This chapter will be about objective (i). In the next chapter we try to solve the question with objective (ii) in mind.

6.1 Betweenness: decreasing edge costs

As we have seen in Section 2.3.2, the total *flow* that passes *through* a country (as a conduit country) equals the weighted betweenness centrality value of this country. In this section we prove an important property of (weighted) betweenness centrality: if we decrease an edge cost of an outgoing edge of a node, the (weighted) betweenness centrality of this node can only increase or stay the same.

Lemma 6.1.1. *Suppose the edge cost of one edge $(u, v) \in E$ is decreased. Then the betweenness centrality $B(u)$ of u increases or stays the same. In other words:*

$$B_{before}(u) \leq B_{after}(u),$$

[1] The set of outneighbours $N^+(u)$ of u consists of all $v \in V$ such that there is an edge $(u, v) \in E$.

where $B_{before}(u)$ (resp. $B_{after}(u)$) refers to the betweenness centrality of u before (resp. after) lowering the edge cost of (u, v).

Proof. It holds that $B(u) = \sum_{s \neq u \neq v} B_{s,t}(u)$. Fix $s, t \in V$. If an edge cost (u, v) is lowered, what will be the effect on $B_{s,t}(u)$? Note that

$$B_{s,t}(u) = \frac{\sigma_{st}(u)}{\sigma_{st}} = \frac{\sigma_{st}(u)}{\sigma_{st}(u^-) + \sigma_{st}(u)}, \tag{6.1}$$

where $\sigma_{st}(u^-)$ denotes the number of shortest paths from s to t *not* passing through vertex u. There are two possibilities.

(1.) Decreasing the edge cost of edge (u, v) lowers the distance $\delta(s, t)$. That means that *all* shortest paths from s to t pass through edge (u, v) and therefore through vertex u. This implies that $B_{s,t_{\text{after}}}(u) = 1 \geq B_{s,t_{\text{before}}}(u)$.

(2.) Decreasing the edge cost of edge (u, v) leaves the distance $\delta(s, t)$ the same. We make a distinction between two possible cases.

  (i) After the lowering of the edge cost (u, v), no shortest paths pass through this edge. Then also no shortest paths could pass through edge (u, v) before the decreasing of the edge cost, since for every path P containing (u, v) it holds that $c(P_{\text{before}}) \geq c(P_{\text{after}})$. Therefore we see that $B_{s,t_{\text{after}}}(u) = B_{s,t_{\text{before}}}(u)$.

  (ii) After the lowering of the edge cost (u, v), at least one shortest path passes through this edge. All shortest paths not passing through (u, v) remain shortest paths (since the distance remains unchanged). Therefore, $\sigma_{st}(u^-)$ stays the same, while $\sigma_{st}(u)$ increases. With the above equality (6.1) we see that $B_{s,t_{\text{after}}}(u) > B_{s,t_{\text{before}}}(u)$.

We conclude that for each $s, t \in V$ it holds that $B_{s,t_{\text{after}}}(u) \geq B_{s,t_{\text{before}}}(u)$. Therefore

$$B_{\text{before}}(u) = \sum_{\substack{s,t,u \in V \\ s \neq u \neq t}} B_{s,t_{\text{before}}}(u) \leq \sum_{\substack{s,t,u \in V \\ s \neq u \neq t}} B_{s,t_{\text{after}}}(u) = B_{\text{after}}(u),$$

which concludes the proof. □

Corollary 6.1.2. *Suppose the edge cost on one edge $(u, v) \in E$ is* decreased. *Then the* weighted *betweenness centrality $B^W(u)$ of u increases or stays the same. In other words:*

$$B^W_{before}(u) \leq B^W_{after}(u).$$

Proof. This follows from the proof of Lemma 6.1.1, since $B_{s,t_{\text{after}}}(u) \geq B_{s,t_{\text{before}}}(u)$ for all $s, t \in V$ (where $s \neq t$) and all weights $w_{s,t}$ are nonnegative. Therefore we conclude that

$$B^W_{\text{before}}(u) = \sum_{\substack{s,t,u \in V \\ s \neq u \neq t}} w_{s,t} B_{s,t_{\text{before}}}(u) \leq \sum_{\substack{s,t,u \in V \\ s \neq u \neq t}} w_{s,t} B_{s,t_{\text{after}}}(u) = B^W_{\text{after}}(u),$$

as desired. □

Corollary 6.1.3. *Suppose the edge cost on one edge $(u, v) \in E$ is* increased. *Then for each $s, t \in V$ it holds that $B_{s,t_{before}}(u) \geq B_{s,t_{after}}(u)$ and hence we have for the (weighted) betweenness centrality that*

$$B^W_{before}(u) \geq B^W_{after}(u).$$

Proof. This follows directly from Lemma 6.1.1 and Corollary 6.1.2. We write $c_1 := c(u,v)_{\text{before}}$ for the initial edge cost of (u,v), and $c_2 := c(u,v)_{\text{after}}$ for the edge cost of (u,v) *after* increasing.

If the initial edge cost of (u,v) was c_2 and we *decreased* the edge cost from c_2 to c_1 then $B_{s,t}(u)$ could only increase (Lemma 6.1.2). This gives

$$B_{s,t}^{c_1}(u) \geq B_{s,t}^{c_2}(u),$$

where $B_{s,t}^{c_1}(u)$ respectively $B_{s,t}^{c_2}(u)$ denote the value $B_{s,t}(u)$ where edge (u,v) has cost c_1 respectively c_2. This gives the desired result for $B_{s,t}(u)$. The result for the (weighted) betweenness centrality follows (similarly as in Corollary 6.1.2), since $B_{s,t_{\text{before}}}(u) \geq B_{s,t_{\text{after}}}(u)$ for all $s,t \in V$ (where $s \neq t$) and all weights $w_{s,t}$ are nonnegative. □

We conclude that the betweenness centrality of a node can only *increase* when *decreasing* edge costs of outgoing edges of this node. Furthermore, increasing edge costs of outgoing edges does not increase the betweenness centrality. We will use these properties of the betweenness centrality throughout this chapter.

6.2 Maximizing betweenness: interpretation

We remember from Section 2.3.2 that the s,t-flow that passes through u, when equally divided along the shortest s,t-paths, equals

$$f_{st}(u) = w_{s,t}B_{s,t}(u) = w_{s,t}\frac{\sigma_{st}(u)}{\sigma_{st}},$$

where the total s,t-flow value $w_{s,t}$ is a given positive value. To maximize $f_{s,t}$ for *one pair* s,t, it is required to maximize $B_{s,t}(u) = w_{s,t}\sigma_{st}(u)/\sigma(st)$. To maximize the total flow

$$f(u) = \sum_{\substack{s,t\in V\\ s\neq u\neq t}} f_{st}(u) = \sum_{\substack{s,t\in V,\\ s\neq u\neq t}} w_{s,t}B_{s,t}(u) = B^W(u),$$

one needs to maximize the weighted betweenness centrality of u. Hence, we are considering the betweenness values as flow-values.

There is also another, possibly more intuitive interpretation of betweenness (see [FS11]): we interpret each pair $s,t \in V$ as 'communicating nodes'. We assume that a communicating path between s and t is selected uniformly at random among all shortest s,t-paths. The probability that node u 'detects' communication s,t is therefore $\sigma_{st}(u)/\sigma_{st}$. The selection s,t as communicating pair is also uniformly at random. Therefore the probability that u detects an arbitrary communication is

$$\frac{1}{(n-1)\cdot(n-2)} \sum_{\substack{s,t\in V,\\ s\neq u\neq t}} \frac{\sigma_{st}(u)}{\sigma_{st}} = \frac{B(u)}{(n-1)\cdot(n-2)},$$

which is proportional to $B(u)$. We see that betweenness can also be phrased in terms of probability. Our problem can thus be rephrased as follows: By lowering at most k edge costs of outgoing edges of $u \in V$, how can we maximize the probability that u detects an arbitrary s,t-communication?

6.3 Maximizing the s,t-flow through a country: *one* fixed pair

Suppose we want to maximize $B_{s,t}(u)$ for one fixed pair s,t (with $s,t \in V$, $s \neq u \neq t$) and $u \in V \setminus \{s,t\}$, by lowering the edge cost of k edges $(u,v_1),\ldots,(u,v_k)$ (with k an integer and $1 \leq k \leq |N^+(u)| \leq n-1$). Maximizing $B_{s,t}(u)$ has two different interpretations, as we have seen in the previous section. By maximizing $B_{s,t}(u)$, we maximize:

(i) the s,t-flow that passes *through* vertex u,
(ii) the probability that vertex u 'detects' communication s,t.

We begin by noting that if we decrease the cost of an edge, then we can as well set its cost to 0. This follows from the proof of Lemma 6.1.1: $B_{s,t}(u)$ can only increase (or remain the same) while lowering an edge cost. Increasing an edge cost does not help to increase $B_{s,t}(u)$ (Corollary 6.1.3). Hence the goal is to maximize $B_{s,t}(u)$ by setting k costs of outgoing edges of u to 0.

Algorithm 6.3.1 (Maximizing $B_{s,t}(u)$)**.** Suppose that we want to maximize $B_{s,t}(u)$ while setting at most k outgoing edges of u to zero. Consider the following algorithm.

1. If there is an edge (u,v), such that

$$c(u,v) > 0 \text{ and } \delta(s,u) + \delta(v,t) < \delta(s,t), \tag{6.2}$$

   then we set the cost of this edge to zero. We can choose (and set to zero) the other $k-1$ edges (u,v_i) $(v_i \in N^+(u))$ arbitrarily.

2. Else, for each $v \in N^+(u)$ such that

$$c(u,v) > 0 \text{ and } \delta(s,u) + \delta(v,t) = \delta(s,t), \tag{6.3}$$

   we compute σ_{vt}. Among the vertices $v \in N^+(u)$ satisfying (6.3) we choose the k vertices for which σ_{vt} is the largest and we set these $c(u,v) := 0$ for these k vertices v. If there are only l vertices with $l < k$ satisfying (6.3) then we choose the remaining $k-l$ vertices arbitrarily.

This procedure maximizes $B_{s,t}(u)$ by setting at most k outgoing edges of u to zero and runs in time bounded by $\mathcal{O}(n^3)$.

Proof. We investigate both steps of the procedure, by making a case distinction.

1. If there is an edge (u,v), such that by setting it to zero we strictly decrease $\delta(s,t)$ (note that this is precisely the case if (6.2) holds) then we set the cost of this edge to zero and we get $f_{st}(u) = w_{st}B_{s,t}(u) = w_{st}$, since all shortest paths then pass through this edge, and hence all shortest s,t-paths pass through u. Now $B_{s,t}(u) = 1$ (which is maximal) so f_{st} is maximal. Therefore we can choose (and set to zero) the other $k-1$ edges (u,v_i) $(v_i \in N^+(u))$ arbitrarily.

2. Else, the distance $\delta(s,t)$ stays the same when setting the cost of an edge $e = (u,v)$ to zero, for all $v \in N^+(u)$. We distinguish two cases.

   (a) if $c(u,v) = 0$ then we cannot lower the edge cost. Also, if $\delta(s,u)+\delta(v,t) > \delta(s,t)$ then decreasing the edge cost of (u,v) has no impact on $B_{s,t}(u)$ since all shortest paths remain shortest paths and there are no new shortest paths created while lowering the edge cost.

   (b) For each $v \in N^+(u)$ such that (6.3) holds, setting this edge to zero adds $\sigma_{su} \cdot \sigma_{vt}$ shortest paths passing through u. Therefore we compute for each edge (u,v) with $v \in N^+(u)$ satisfying (6.3) the quantity

$$g(v) := \sigma_{su} \cdot \sigma_{vt}.$$

   Then we choose the k vertices v satisfying (6.3) with the biggest $g(v)$, or equivalently (and more easily) with the biggest $g(v)/\sigma_{su} = \sigma_{vt}$. Taking these k vertices will increase $B_{s,t}(u)$ the most, since it will increase the number of shortest s,t-paths

passing through u the most, while the number of shortest s,t-paths *not* passing through u remains the same. If there are only l with $l < k$ vertices satisfying (6.3) then we can choose the remaining $k - l$ vertices arbitrarily, since all other outgoing edges of u satisfy (a) and therefore lowering their costs leaves $B_{s,t}(u)$ the same.

Hence, the above procedure is correct. Moreover, the running time of the algorithm is dominated by calculating the distances $\delta(\cdot,\cdot)$ using Floyd-Warshall. Therefore the running time of the algorithm is $\mathcal{O}(n^3)$. □

We now know how to maximize the s,t-flow that passes through one vertex $u \in V$. In the next section we will consider the more interesting problem of how to maximize the *total* flow that passes through a vertex.

6.4 Maximizing the *total* flow through a country: all pairs

The total flow that passes through a vertex u is

$$f(u) = \sum_{\substack{s,t,u \in V \\ s \neq u \neq t}} f_{st}(u) = \sum_{\substack{s,t,u \in V \\ s \neq u \neq t}} w_{s,t} B_{s,t}(u) = B^W(u),$$

which is the (weighted) betweenness centrality of u. We want to maximize $f(u)$, by lowering the edge cost of one edge $(u,v) \in E$, and later by lowering the edge costs of k edges $(u,v_1),\ldots,(u,v_k)$ in E.

Hence, we are required to maximize $B^W(u)$. To do this, we must set one edge (u,v) to zero, as follows from Corollary 6.1.2. We have at most $|N^+(u)|$ possible choices for v. Therefore, is sufficient to calculate for each outgoing edge (u,v) of u the weighted betweenness for our graph G with $c(u,v) = 0$. Then we choose the edge (u,v) for which the weighted betweenness is maximal. Note that this is a polynomial time algorithm (in n), since it involves at most $|N^+(u)| \leq n-1$ betweenness-calculations (and a betweenness-calculation can be done in time $\mathcal{O}(mn + n^2 \log n)$, see Theorem 2.3.3).

Now, suppose that we are given an integer k with $1 \leq k \leq |N^+(u)|$. It is allowed to choose k vertices $v_1,\ldots v_k$, and to lower the edge costs of $(u,v_1),\ldots (u,v_k)$. Which k vertices do we need to choose? We could try all subsets of size k of $N^+(u)$, and compute betweenness when the costs of all edges in the subset are set to zero. This will cost time $\binom{|N^+(u)|}{k}$ multiplied by $\mathcal{O}(mn + n^2 \log n)$, the time for one betweenness-calculation.

If k is constant (not depending on n) this approach will work and give a polynomial time algorithm. In particular, for graphs in which vertex u has a *constant* outdegree $|N^+(u)|$, the approach works in polynomial time. However, if $k \leq |N^+(u)| \leq n-1$ is *not* constant then trying all subsets of size k of $N^+(u)$ will not give us a polynomial algorithm. Perhaps a polynomial time greedy algorithm would work? It turns out that this is not the case: in fact, we will prove that Problem 6 is *NP*-hard. We begin by stating the problem formally.

Problem 6 (Maximizing the betweenness of one node (*Maximizing Betweenness*))**.** Given a directed graph $G = (V,E)$, a vertex $u \in V$ and additive edge costs $c : E \to \mathbb{R}_{\geq 0}$. What is the maximum betweenness $B^W(u)$ that can be obtained by setting the cost of at most k outgoing edges of u to zero?

First we assume that all weights are equal, so that we need to maximize $B(u)$ instead of $B^W(u)$. We consider a greedy approach.

Algorithm 6.4.1 (Greedy approach to *Maximizing Betweenness* (Problem 6))**.** Let $B_0(u)$ be the betweennes $B(u)$ of u in G. We perform the following steps:

(i) First, we choose the edge (u, v_1) with $v_1 \in N^+(u)$ such that the betweenness $B_{\text{after}}(u)$ of u after setting this edge cost to zero is maximized. We set the cost of this edge to zero and we set $B_1(u) := B_{\text{after}}(u)$.

(ii) Suppose $(u, v_1), \ldots, (u, v_{j-1})$ are chosen. Then we choose the edge (u, v_j) with $v_j \in N^+(u)$ such that the betweennes $B_{\text{after}}(u)$ of u after setting this edge cost to zero is maximized. We set the cost of this edge to zero and we set $B_j(u) := B_{\text{after}}(u)$.

Unfortunately, the proposed greedy algorithm does not always produce an optimal result, as the next example shows.

Example 6.4.1. Consider the following graph:

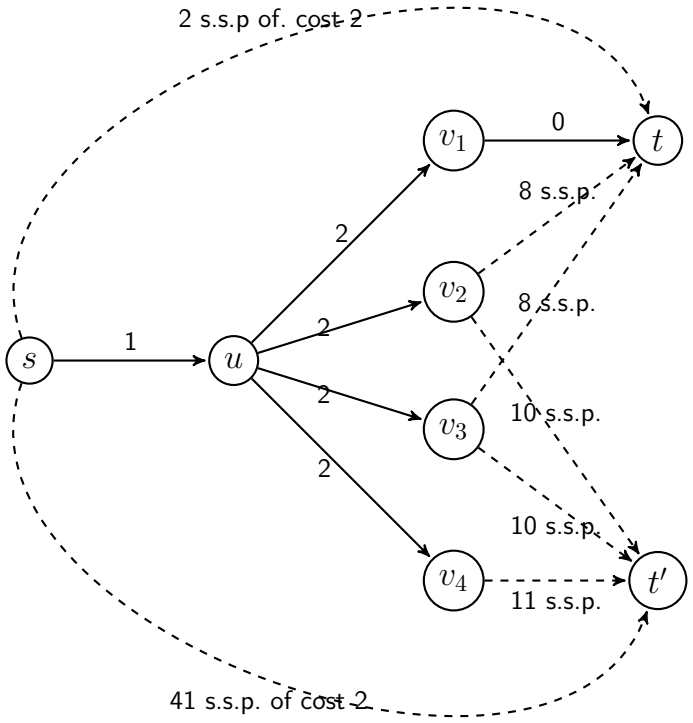


Figure 6.1: Example graph $G = (V, E)$, with $k = 2$. The abbrevation 's.s.p.' stands for 'simple shortest paths'. If no cost of a path is mentioned, then the cost of this path is 1. Dashed lines stand for (an amount of) simple shortest paths, non-dashed lines are edges. The betweenness of u is maximized by setting the costs of (u, v_2) and (u, v_3) to zero, while the greedy algorithm would give (u, v_1) and (u, v_4).

Let $G = (V, E)$ be a graph where V consists of at least $T = \{s, v_1, v_2, v_3, v_4, t, t'\} \subset V$ and more vertices edges as are required to satisfy the following properties of $G = (V, E)$:

(i) There are 2 simple shortest paths of cost 2 from s to t not passing any vertex in T.

(ii) There are 41 simple shortest paths of cost 2 from s to t' not passing through any vertex in T.

(iii) There is one edge (v_1, t) of cost 2.

(iv) There are 8 simple shortest paths of cost 1 from v_2 to t not passing through any vertex in T.

(v) There are 10 simple shortest paths of cost 1 from v_2 to t' not passing through any vertex in T.

(vi) There are 8 simple shortest paths of cost 1 from v_3 to t not passing through any vertex in T.

(vii) There are 10 simple shortest paths of cost 1 from v_3 to t' not passing through any vertex in T.

(viii) There are 11 simple shortest paths of cost 1 from v_4 to t' not passing through any vertex in T.

(ix) There is an edge (s, u) of cost 1, and there are edges (u, v_i) of cost 2 for $i = 1, \ldots, 4$.

Furthermore, let $k = 2$. Initially, setting edge (u, v_1) to 0 increases $B(u)$ by 1, setting edge (u, v_2) to zero increases $B(u)$ by $8/10+11/51 < 1$, setting edge (u, v_3) to zero increases $B(u)$ by $8/10+11/51 < 1$ and setting edge (u, v_4) to zero increases $B(u)$ by $11/52$. In a greedy approach, we would pick (u, v_1) first and set this edge to zero. After that we would pick (u, v_4) to get a total increase of the betweenness of $1 + 11/52$. But it can be easily verified that picking (u, v_2) and (u, v_3) gives a higher increase in betweenness: $16/18 + 20/61 > 1 + 11/52$.

Example 6.4.1 shows that our proposed greedy approach does not work. But we can even prove a stronger result: *Maximizing Betweenness* (Problem 6) is *NP*-hard. We will do a reduction from the *Maximum Coverage Problem*.

Problem 7 (Maximum Coverage Problem)**.** Given a universe $\mathcal{U} = \{e_1, \ldots, e_N\}$, a collection of subsets of this universe $S = S_1, \ldots, S_t$ and a number k, find a subcollection of sets $S' \subseteq S$ such that $|S'| < k$ and the total number of covered elements $|\cup_{S_i \in S'} S_i|$ is maximal.

The Maximum Coverage Problem is *NP*-hard [Fei98]. This we will use to prove that *Maximizing Betweenness* is also *NP*-hard.

Theorem 6.4.1. Maximizing Betweenness *(Problem* 6*) is NP-hard.*

Proof. We use a reduction from the *Maximum Coverage Problem*. Given a universe $\mathcal{U} = \{e_1, \ldots, e_N\}$, a collection of subsets of this universe $S = S_1, \ldots, S_t$ and a number k, we construct an instance of Problem 6 as follows:

Let there be vertices $s, u, v_1, \ldots, v_t$ (so for every set $S_i \in S$ there is a vertex v_i) and let there also be a vertex for each element in $\mathcal{U}$:

$$V = \{s, u, v_1, \ldots, v_t\} \cup (\cup_{e_i \in \mathcal{U}} e_i) .$$

Let there be directed edges as follows:

- There is an edge (s, u) of cost 1.
- For every v_i, where $i = 1, \ldots, m$, there is an edge (u, v_i) of cost 2.
- For every $x \in \mathcal{U}$, and for every $i \in \{1, \ldots, m\}$ there is an edge (v_i, x) if and only if $x \in S_i$. This edge has cost 1.
- For every $x \in \mathcal{U}$, there is an edge (s, x) of cost 3.

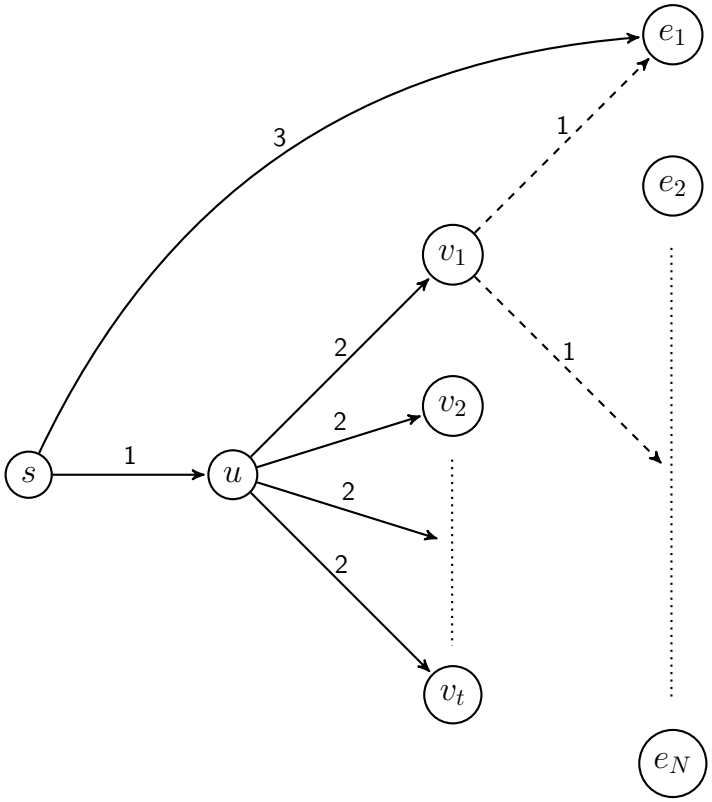


Figure 6.2: Illustration of the instance created in the reduction of *Maximum Coverage* to *Maximizing Betweenness* (Problem 6).

Firstly, we note that the reduction can be done in polynomial time: $\mathcal{O}(t \cdot N)$. Also we observe that, before lowering any edge cost, the (unweighted) betweenness of u is $B(u) = t$. This is because all paths from s to each v_i pass only through u and all shortest paths from s to x, for $x \in \mathcal{U}$, do *not* pass through u. For every other pair (source, sink) (where the source is not s or u and the sink is not u), there are no paths passing u.

By setting the edge cost of edges $(u, v_{i_1}), \ldots, (u, v_{i_k})$ to zero all shortest paths from s to x, for each $x \in S_{i_1} \cup \ldots \cup S_{i_k}$ suddenly pass through u. For other sink-source-pairs the shortest paths do not change. Therefore the betweenness of u is $B(u) = t + |S_{i_1} \cup \ldots \cup S_{i_k}|$ implying that the betweenness of u increases exactly with $|S_{i_1} \cup \ldots \cup S_{i_k}|$.

Now, for $i_1, \ldots, i_k \in \{1, \ldots, t\}$, it holds that $S_{i_1}, \ldots, S_{i_k}$ is a maximum cover if and only if $v_{i_1}, \ldots, v_{i_k}$ solve Problem 6.

"$\Longrightarrow$". Suppose $S_{i_1}, \ldots, S_{i_k}$ is a maximum cover. Then $|S_{i_1} \cup \ldots \cup S_{i_k}|$ is maximal, hence $B(u)$ increases maximally by setting the costs of $(u, v_{i_1}), \ldots, (u, v_{i_k})$ to zero. Therefore $v_{i_1}, \ldots, v_{i_k}$ solve Problem 6.

"$\Longleftarrow$". Suppose $v_{i_1}, \ldots, v_{i_l}$ solve Problem 6. That means that $B(u)$ increases maximally by setting the costs of $(u, v_{i_1}), \ldots, (u, v_{i_k})$ to zero. Since the betweenness of u increases exactly with $|S_{i_1} \cup \ldots \cup S_{i_k}|$, this implies that $|S_{i_1} \cup \ldots \cup S_{i_k}|$ is maximal, hence $S_{i_1}, \ldots, S_{i_k}$ is a maximum cover. □

Now that we know that *Maximizing Betweenness* (Problem 6) is *NP*-hard, we will look for an approximation algorithm. The bound on the maximum error the greedy algorithm produces (the increase in betweenness given by the greedy algorithm compared to the optimal increase in betweenness) can be estimated, in a similar way as it can be done for *Maximum Coverage* [Hoc97]. We prove two auxiliary lemmas.

Lemma 6.4.2. *Let $G = (V, E)$, $u \in V$ with betweenness $B(u)$. Suppose we first set the cost of r edges $(u, v_1), \ldots, (u, v_r)$ to zero. We denote the betweenness of u after setting these edges to zero*

by $B(u)_r$. Suppose we subsequently set the edge cost of an edge (u, v') to zero. Let the resulting betweenness be $B(u)'_r$. Then, the gain in betweenness that would be obtained by setting (u, v') to zero in the first step (in the initial graph G) is at least $B(u)'_r - B(u)_r$, i.e.

$$B(u)' - B(u) \geq B(u)'_r - B(u)_r, \tag{6.4}$$

where $B(u)'$ denotes the betweenness of u in the original graph, only with the edge cost of (u, v') set to zero.

Proof. Take one s, t-pair. We will first prove inequality (6.4) for $B_{s,t}$ instead of B. We will use four distance functions: the original distances $\delta(\cdot, \cdot)$ in G, the distances $\delta'(\cdot, \cdot)$ in G with when only the cost of (u, v') is set to zero, the distances $\delta_r(\cdot, \cdot)$ in G after the edge costs of the given r edges are set to zero, and the distances $\delta'_r(\cdot, \cdot)$ in G after the edge costs of the r edges and of (u, v') are set to zero. We begin by noting that

$$\delta(s, u) = \delta'(s, u) = \delta_r(s, u) = \delta'_r(s, u)$$

and

$$\delta(v', t) = \delta'(v', t) \text{ and } \delta_r(v', t) = \delta'_r(v', t).$$

If $\delta_r(v', t) < \delta(v', t)$ then all shortest paths, after setting the r edge costs to zero, from v' to t pass through u. Therefore

$$\delta_r(s, t) \leq \delta_r(s, u) + \delta_r(u, t) < \delta_r(s, u) + \delta_r(v', t), \tag{6.5}$$

where the last inequality holds because after setting $c(u, v')$ to zero, no zero cycles can arise (by assumption); therefore the shortest v', u-path cannot be of cost zero and hence $\delta_r(v', t) = \delta_r(v', u) + \delta_r(u, t) > \delta_r(u, t)$. In the situation of (6.5) it holds that $B(u)'_r = B(u)_r$ (setting $c(u, v')$ to zero creates no extra shortest paths), so that 6.4 is immediately satisfied. Therefore we assume from now on that

$$\delta(v', t) = \delta'(v', t) = \delta_r(v', t) = \delta'_r(v', t).$$

We distinguish a few cases.

1. If $\delta(s, t) > \delta(s, u) + \delta(v', t)$ then all paths will pass through u after setting edge cost (u, v') to zero. This means that $B_{s,t}(u)'_r = B_{s,t}(u)' = 1$. The inequality (6.4) follows because $B_{s,t}(u)_r \geq B_{s,t}(u)$ by the proof of Theorem 6.1.1.

2. If $\delta(s, t) = \delta(s, u) + \delta(v', t)$ then only part of the paths will pass through u after setting edge cost (u, v') to zero.

   - If $\delta_r(s, t) < \delta(s, t)$ then

     $$\delta_r(s, t) < \delta(s, t) = \delta(s, u) + \delta(v', t) = \delta_r(s, u) + \delta_r(v', t),$$

     so setting the edge cost of (u, v') to zero after the given r edges are set to zero will have no effect on the betweenness, i.e. $B_{s,t}(u)'_r - B_{s,t}(u)_r = 0$ and the inequality (6.4) is satisfied.

   - If $\delta_r(s, t) = \delta(s, t)$ then the number of shortest paths through u increases by some number t (which is the number of shortest paths from s to t via (u, v')) by setting (u, v') to zero, both in the original case and after setting r edges to zero. Therefore we have

     $$B_{s,t}(u)' = \frac{B_{s,t}(u) \cdot \sigma_{st} + t}{t + \sigma_{st}},$$

which gives

$$\begin{aligned} B_{s,t}(u)' - B_{s,t}(u) &= \frac{\sigma_{st} B_{s,t}(u) + t}{t + \sigma_{st}} - B_{s,t}(u) \\ &= \frac{\sigma_{st} B_{s,t}(u) + t - t B_{s,t}(u) - \sigma_{st} B_{s,t}(u)}{t + \sigma_{st}} \\ &= \frac{t - t B_{s,t}(u)}{t + \sigma_{st}} = \frac{t(1 - B_{s,t}(u))}{t + \sigma_{st}} \end{aligned} \tag{6.6}$$

and similarly

$$B_{s,t}(u)'_r = \frac{\sigma_{r_{st}} B_{s,t}(u)_r + t}{t + \sigma_{r_{st}}},$$

so

$$B_{s,t}(u)'_r - B_{s,t}(u)_r = \frac{t - t B_{s,t}(u)_r}{t + \sigma_{r_{st}}} = \frac{t(1 - B_{s,t}(u)_r)}{t + \sigma_{r_{st}}}. \tag{6.7}$$

Now, by noting that $(1 - B_{s,t}(u)_r) \leq (1 - B_{s,t}(u))$ and that $\sigma_{r_{s,t}} \geq \sigma_{s,t}$ (since $\delta_r(s,t) = \delta(s,t)$), by comparing (6.6) and (6.7), the desired inequality (6.4) follows.

- $\delta_r(s,t) > \delta(s,t)$ is not possible, the distance from s to t cannot get bigger while setting edge costs to zero.

3. If $\delta(s,t) < \delta(s,u) + \delta(v',t)$, then setting edge cost (u,v') to zero will have no effect on the betweenness in any step of the algorithm, so $B_{s,t}(u)' - B_{s,t}(u) = B_{s,t}(u)'_r - B_{s,t}(u)_r = 0$ and the inequality (6.4) is satisfied.

We conclude that for each s,t-pair it holds that

$$B_{s,t}(u)' - B_{s,t}(u) \geq B_{s,t}(u)'_r - B_{s,t}(u)_r,$$

therefore this inequality (inequality (6.4)) also holds for the betweenness $B(u)$, since the betweenness is the sum of $B_{s,t}$-values: $B(u) = \sum_{s \neq u \neq t} B_{s,t}(u)$. □

Corollary 6.4.3. *Lemma* 6.4.2 *also holds for the* weighted *betweenness centrality* $B^W(u)$ *instead of the unweighted betweenness centrality* $B(u)$.

Proof. Note that we proved Lemma 6.4.2 per s,t-pair. Therefore the inequality also holds for $B^W(u)$ instead of $B(u)$ (all weights are nonnegative). □

Lemma 6.4.4. *It holds that*

$$B_j(u) - B_{j-1}(u) \geq \frac{OPT - B_{j-1}(u)}{k}, \quad \text{for } j = 1, \ldots, k,$$

where OPT is the maximum betweenness centrality of u *that can possibly be obtained by setting the edge cost of* k *outgoing edges of* u *to zero and* $B_j(u)$ *is as in Algorithm* 6.4.1.

Proof. Suppose we have completed $j-1$ steps of the greedy algorithm. After completing 0 steps, the maximum betweenness that can possibly be obtained by setting the edge cost of k outgoing edges of u to zero is OPT. After $j-1$ edges are set to 0, we can still obtain a betweenness $\geq$ OPT by setting the optimal k edges in the original setting to zero. Hence, by setting at most k extra edges to zero, we can get an increase in betweenness of $\text{OPT} - B_{j-1}(u)$. This means that there must be at least one step in the process of setting the k edges to zero where the betweenness rises with at least

$$\frac{\text{OPT} - B_{j-1}(u)}{k}.$$

Setting the edge of this step to zero immediately (before setting other edges to zero) can only give a larger rise in betweenness by Lemma 6.4.2. Since the j-th greedy step gives the highest rise in betweenness in the situation of $B_{j-1}(u)$, it holds that

$$B_j(u) - B_{j-1}(u) \geq \frac{\text{OPT} - B_{j-1}(u)}{k},$$

which is the claim of the lemma. □

Lemma 6.4.5. *It holds that*

$$B_j(u) - B_0(u) \geq \left(1 - \left(1 - \frac{1}{k}\right)^j\right)(OPT - B_0(u)), \quad \text{for } j = 1, \ldots, k.$$

Proof. We prove the result by induction. Vertex u is fixed, so we write B_j instead of $B_j(u)$ to simplify notation.

(1.) For $j = 1$ the result holds by the previous lemma: $B_1(u) - B_0 \geq (\text{OPT} - B_0)/k$.

(j.) Suppose the result holds for $j - 1$. We prove that the result also holds for j.

$$\begin{aligned}
B_j - B_0 &= (B_{j-1} - B_0) + (B_j - B_{j-1}) \\
&\geq (B_{j-1} - B_0) + \frac{\text{OPT} - B_{j-1}}{k} \\
&= \left(1 - \frac{1}{k}\right)(B_{j-1} - B_0) + \frac{\text{OPT} - B_0}{k} \\
&\geq \left(1 - \frac{1}{k}\right)\left(1 - \left(1 - \frac{1}{k}\right)^{j-1}\right)(\text{OPT} - B_0) + \frac{\text{OPT} - B_0}{k} \\
&= \left(1 - \left(1 - \frac{1}{k}\right)^j\right)(\text{OPT} - B_0),
\end{aligned}$$

which gives the desired result. The first inequality was proven in Lemma 6.4.4 and the second inequality holds by the induction hypothesis.

We conclude that the result holds by induction. □

Now the approximation result follows easily.

Theorem 6.4.6. *It holds that*

$$B_k - B_0 \geq \left(1 - \left(1 - \frac{1}{k}\right)^k\right)(OPT - B_0) > \left(1 - \frac{1}{e}\right)(OPT - B_0) > 0.632 \cdot (OPT - B_0),$$

where B_k is the betweenness centrality of u after the kth step of the greedy algorithm (Algorithm 6.4.1*).*

Proof. We apply Lemma 6.4.5, with $j = k$. This gives the first inequality. The second inequality follows because $1 - (1 - 1/k)^k$ is decreasing with

$$\lim_{k \to \infty} 1 - \left(1 - \frac{1}{k}\right)^k = 1 - \frac{1}{e}.$$

Hence it holds that

$$1 - \left(1 - \frac{1}{k}\right)^k > 1 - \frac{1}{e} > 0.632,$$

which proves the theorem. □

The approximation ratio of Theorem 6.4.6 is at *least* $1-1/e$, but for small k the approximation ratio is better. If we write $\alpha_k := 1-(1-1/k)^k$, then we have as $\alpha_1 = 1$, $\alpha_2 = 0.75$, $\alpha_3 \approx 0.703$. For $k = 1$ the greedy algorithm (trivially) achieves an optimal increase in betweenness.

Theorem 6.4.7. *The approximation ratio of Theorem* 6.4.6 *is tight.*

Proof. We give an outline of the example from [HP98][2]. Let B be a $(k+1) \times k$-matrix, such that the i,j-th entry contains $b_{i,j}$ elements, where:

(0.) Row 0:

$$b_{0,j} = \begin{cases} (k-1)^{k-1}, & \text{if } j = 1, \\ (k-2)(k-1)^{k-2}, & \text{if } 2 \leq j \leq k. \end{cases}$$

(1.) Row 1:

$$b_{1,j} = \begin{cases} 0, & \text{if } j = 1, \\ (k-1)^{k-2}, & \text{if } 2 \leq j \leq k. \end{cases}$$

(i.) Row i:

$$b_{i,j} = k^{i-2}(k-1)^{k-i}, \quad 2 \leq i \leq k,\ 1 \leq j \leq k.$$

Let C_i be the set consisting of all elements of the i-th column of B, where $0 \leq i \leq k$. Furthermore, let R_j be the set consisting of all elements of the j-th column of B, where $1 \leq j \leq k$. Let

$$S := \{C_0, C_1, \ldots, C_k\} \cup \{R_1, \ldots, R_k\}.$$

Let U be the set consisting of all elements of B. Now, construct an instance of Problem 6 as in Theorem 6.4.1 (the reduction to *Maximum Coverage*).

(i) An optimal solution to Problem 6 is $\{C_1, \ldots, C_k\}$. Then all elements of B are covered, so we will get a maximal betweenness value in the reduction of Theorem 6.4.1.
(ii) However, one can prove that the greedy algorithm can select $\{R_k, R_{k-1}, \ldots, R_1\}$ as solution.
(iii) Furthermore, one can prove that total number of elements in rows 1 through k is exactly $1-(1-1/k)^k$ times the total number of elements in B.

Note that $1-(1-1/k)^k$ is exactly the approximation ratio of Theorem 6.4.6. Therefore the approximation ratio of this example is tight. □

Even a stronger result holds.

Theorem 6.4.8. *Let $B_0(u)$ be the initial betweenness value of u. There is no polynomial time algorithm to compute a feasible solution (i.e. there is no polynomial time algorithm to find k outgoing edges of u) for Problem* 6 *(*Maximizing Betweenness*) that always outputs a solution with betweenness $B_{SOL}(u)$ such that*

$$B_{SOL}(u) - B_0(u) \geq \alpha \cdot (OPT - B_0(u)),$$

with $1 \geq \alpha > 1-1/e$, unless $NP \subseteq DTIME(n^{\mathcal{O}(\log\log n)})$.

[2]The authors of [HP98] took this example from [DJS93].

Proof. Suppose for the sake of contradiction that there exists a polynomial time algorithm that outputs a solution with betweenness $B_{\text{SOL}}(u)$ such that $B_{\text{SOL}}(u) - B_0(u) \geq \alpha \cdot (\text{OPT} - B_0(u))$ with $1 \geq \alpha > 1 - 1/e$. We apply this algorithm to the instance of Problem 6 used in the reduction from *Maximum Coverage* in Theorem 6.4.1. This means that the resulting betweenness value $B_{\text{SOL}}(u)$ of the algorithm will satisfy

$$B_{\text{SOL}}(u) - B_0(u) \geq \alpha(\text{OPT} - B_0) = \alpha \cdot ((\text{OPT}_{\text{maxcover}} + t) - t) = \alpha \cdot \text{OPT}_{\text{maxcover}}. \tag{6.8}$$

Let $v_{s_1}, \ldots, v_{s_k}$ be the vertices that are set to zero in the obtained solution with betweenness $B_{\text{SOL}}(u)$. Then it holds that

$$B_{\text{SOL}}(u) = t + |S_{s_1} \cup \ldots \cup S_{s_k}| = B_0(u) + |S_{s_1} \cup \ldots \cup S_{s_k}|.$$

Combining this with (6.8) gives

$$|S_{s_1} \cup \ldots \cup S_{s_k}| = B_{\text{SOL}}(u) - B_0(u) \geq \alpha \cdot \text{OPT}_{\text{maxcover}},$$

where $\alpha > 1-1/e$. That means that *Maximum Coverage* can be approximated within a factor α, where $\alpha > 1 - 1/e$. This is impossible, as is proven in [Fei98]. We arrive at a contradiction. □

6.5 Submodular set functions

In this section we will prove the approximation ratio from Theorem 6.4.8 using the theory from the literature about *submodular set functions*. First we will define these functions (see the definitions in [Schr04] and [NWF78]). Subsequently, we will use the approximation algorithm for submodular monotone functions from [NWF78] to re-prove Theorem 6.4.8.

Definition 6.5.1 (Submodular set function)**.** A *submodular set function* is a *set* function $S \to \mathbb{R}$ (that is, a function $\mathcal{P}(S) \to \mathbb{R}$, where $\mathcal{P}(S)$ is the set of al subsets of S) that satisfies the condition

$$f(T) + f(U) \geq f(T \cap U) + f(T \cup U), \tag{6.9}$$

for all subsets T, U of S.

Theorem 6.5.1. *Let $G = (V, E)$ be a directed graph, $u \in V$ a vertex and $c : E \to \mathbb{R}_{\geq 0}$ additive edge costs. Let S be the set of all outgoing edges of u. The function $f : \mathcal{P}(S) \to \mathbb{R}$, given by*

$$f(U) = B_U^W(u) - B_0^W(u),$$

(where $B_U^W(u)$ denotes the (weighted) betweenness centrality of u in G, but with the edge cost of all edges in U set to zero, and $B_0^W(u)$ denotes the initial weighted betweenness centrality of u in G) is a submodular set function $S \to \mathbb{R}$.

Proof. This follows from Lemma 6.4.2. Let $U, T \subset S$ be sets of outgoing edges of u. We will prove that

$$f(T) - f(T \cap U) \geq f(T \cup U) - f(U), \tag{6.10}$$

establishing (6.9). First, define $f' : \mathcal{P}(S) \to \mathbb{R}$ by

$$f'(U) := B_U^W(u) = f(u) + B_0^W(u).$$

We will prove (6.10) for the function f' instead of f. It immediately follows that the equation (6.10) then also holds for f.

We first (before applying Lemma 6.4.2) set the edge costs of the outgoing edges in $T \cap U$ to zero. Let R be the set of edges in $U \setminus (T \cap U)$ and let r be $|R|$. Let Z be the set of edges

in $T \setminus (T \cap U)$, where we write $Z = \{z_1, \ldots, z_{|Z|}\}$. Lemma 6.4.2 gives us (by applying it with the r edges from R) that

$$f'(\{z_1\} \cup (T \cap U)) - f'(T \cap U) \geq f'(\{z_1\} \cup U) - f'(U).$$

Suppose we have proven for $Z' = \{z_1, \ldots, z_{j-1}\}$ (where $2 \leq j \leq |Z|$) that

$$f'(Z' \cup (T \cap U)) - f'(T \cap U) \geq f'(Z' \cup U) - f'(U). \tag{6.11}$$

Then, applying Lemma 6.4.2 gives

$$f'(\{z_j\} \cup Z' \cup (T \cap U)) - f'(Z' \cup (T \cap U)) \geq f'(\{z_j\} \cup Z' \cup U) - f'(U \cup Z'). \tag{6.12}$$

Adding inequalities (6.11) and (6.12) yields

$$f'(\{z_j\} \cup Z' \cup (T \cap U)) - f'(T \cap U) \geq f'(\{z_j\} \cup Z' \cup U) - f'(U). \tag{6.13}$$

Therefore it holds by induction that

$$f'(Z \cup (T \cap U)) - f'(T \cap U) \geq f'(Z \cup U) - f'(U),$$

i.e.

$$f'(T) - f'(T \cap U) \geq f'(T \cup U) - f'(U),$$

which yields equation (6.10) (since $f' = f + C$, with C a constant), as was needed to prove. □

Definition 6.5.2 (Monotone nondecreasing set function)**.** A set function f on S is called *monotone nondecreasing* if $f(U) \leq f(T)$ for all subsets U, T with $U \subseteq T \subseteq S$.

Lemma 6.5.2. *The function f from Theorem* 6.5.1 *is a monotone nondecreasing set function.*

Proof. Follows directly from a repeated application of Corollary 6.1.2. □

Problem 8 (Maximizing a submodular nondecreasing set function)**.** Let f be a submodular nondecreasing set function $S \to \mathbb{R}$. Let $k \leq |S|$ be a natural number. We would like to find a $T \subseteq S$ with $|T| = k$ such that $f(T)$ is as large as possible.

Algorithm 6.5.1 (Greedy approach for maximizing a submodular nondecreasing set function)**.** To approximate a solution to Problem 8, one could perform the following steps.

(i) First, we choose $e_1 \in S$ such that $f(e_1)$ is maximal.
(ii) Suppose $e_1, \ldots, e_{j-1} \in S$ are chosen. Then we choose an element $e_j \subseteq S$ such that $f(e_j \cup \{e_1, \ldots, e_{j-1}\})$ is maximized.

Theorem 6.5.3. *Algorithm* 6.5.1 *outputs a set U for which $f(U)$ is $\geq 1 - 1/e$ times as big as the optimal value.*

Proof. See [NWF78]. □

Corollary 6.5.4. *The greedy algorithm (Algorithm* 6.4.1*) to maximize the betweenness gives a solution in which the rise in betweenness is at least $1-1/e$ times the optimal rise in betweenness.*

Proof. This follows from Theorem 6.5.3. □

We see that by using the general theory for submodular set functions, one can prove that the greedy algorithm (Algorithm 6.4.1) to solve Problem 6 yields a set of edges that give a rise in betweenness of at least $(1 - 1/e)$ times the optimal rise in betweenness.

6.6 Results

In this section we will test the greedy algorithm in the case that we want to improve the betweenness of the Netherlands. First we make an adaptation to the network to make the experiment more relevant.

Countries can only partly control the tax rate that must be paid when money is sent along their outgoing edges[3]. Therefore, we split the outgoing edge costs of the Netherlands in two parts, with an auxiliary node in between. The first part will be the reliability based on the tax rate that the Netherlands controls, and the second part will be the 'remaining' part: the part that the Netherlands does *not* control. Remember from Section 2.4 that the edge reliabilities depend on whether a country is the source country, the starting vertex of a path. Since we compute the betweenness of the Netherlands (and hence consider paths that have the Netherlands neither as source, nor as sink vertex), we do only need to adapt the $|V| - 1 = 107$ graphs from Section 2.4 in which the Netherlands is *not* a source country. Note that for all these graphs (cf. Section 2.4), the outgoing edge reliabilities of NLD are the same.

For every country i, $i \neq \text{NLD}$, we add an auxiliary node i_A to the network, and we replace the edge (NLD, i) by two edges (NLD, i_A) and (i_A, i). Let t_{ci} be the tax of the part of (the original) edge (NLD, i) that The Netherlands controls. The reliability of edge (NLD, i_A), we define as $1 - t_{ci}$.

Let q_i be the reliability of the original edge (NLD, i). Since we multiplied each edge (except for edges with tail the source of G_i), but NLD does not appear as source vertex in our computations in this section) by $(1 - \varepsilon)$ (see Section 2.4.1), it holds that $q_i \leq (1 - \varepsilon)(1 - t_{ci})$. We define reliability q_i' as follows

$$q_i = (1 - t_{ci}) \cdot q_i',$$

so $q_i' \leq (1 - \varepsilon)$. We set $r(i_A, i) := q_i'$, which implies that $r(\text{NLD}, i_A) \cdot r(i_A, i) = q_i$.

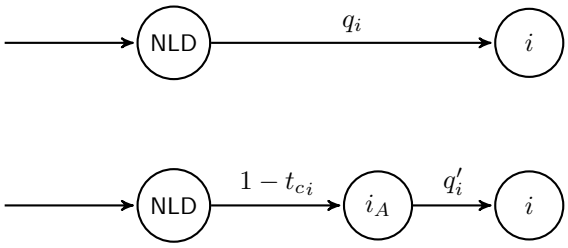


Figure 6.3: We add an auxiliary node. Above: the network before the auxiliary node is added. Below: the adaptation to the network.

Note that the resulting networks (with the outgoing edges of the Netherlands split into two parts) contain no cycles of reliability 1, since the original networks contained no cycles of reliability 1. Furthermore, in the resulting graphs, there are no cycles of reliability 1, *even if* we change the edge reliability of edges (NLD, i_A) from $1 - t_{ci}$ to 1. To see this, note that the only edges of reliability 1 in G_j, the graph of source country j (where $j \neq \text{NLD}$), might be (j, u) (for $u \in V$), and edges (NLD, i_A) get reliability 1, but edge (i_A, i) does *not* have reliability 1, since $r(i_A, i) = q_i' \leq (1 - \varepsilon)$.

The betweenness of The Netherlands is initially 6.27001. The weighted betweenness is initially 2.63225. When The Netherlands is a conduit country in our network, there are 59 countries i to which The Netherlands has an edge (with the adaptation as above) (NLD, i_A) of

[3]The country controls the withholding tax on dividends, but not the tax that must be paid in another country when money enters this country.

reliability unequal to one (i.e. nonzero outgoing tax rate t_{ci}). These countries are:

$$\begin{aligned}\mathcal{S} := \{&\text{DZA}, \text{AGO}, \text{ARG}, \text{ABW}, \text{AUS}, \text{AZE}, \text{BHS}, \text{BHR}, \text{BMU}, \text{BWA}, \text{BRA}, \text{BRN},\\ &\text{CAN}, \text{CYM}, \text{CHL}, \text{CHN}, \text{COL}, \text{CRI}, \text{CUR}, \text{DOM}, \text{ECU}, \text{GNQ}, \text{GAB}, \text{GRN},\\ &\text{IND}, \text{IDN}, \text{IMN}, \text{ISR}, \text{JAM}, \text{JRY}, \text{JOR}, \text{KOR}, \text{LBN}, \text{LBY}, \text{LIE}, \text{MAC},\\ &\text{MUS}, \text{MEX}, \text{NAM}, \text{NZL}, \text{NGA}, \text{PAK}, \text{PAN}, \text{PER}, \text{PHL}, \text{PRI}, \text{SAU}, \text{YUG},\\ &\text{SYC}, \text{ZAF}, \text{SUR}, \text{TWN}, \text{THA}, \text{TTO}, \text{TUR}, \text{ARE}, \text{URY}, \text{VIR}, \text{VGB}\}.\end{aligned}$$

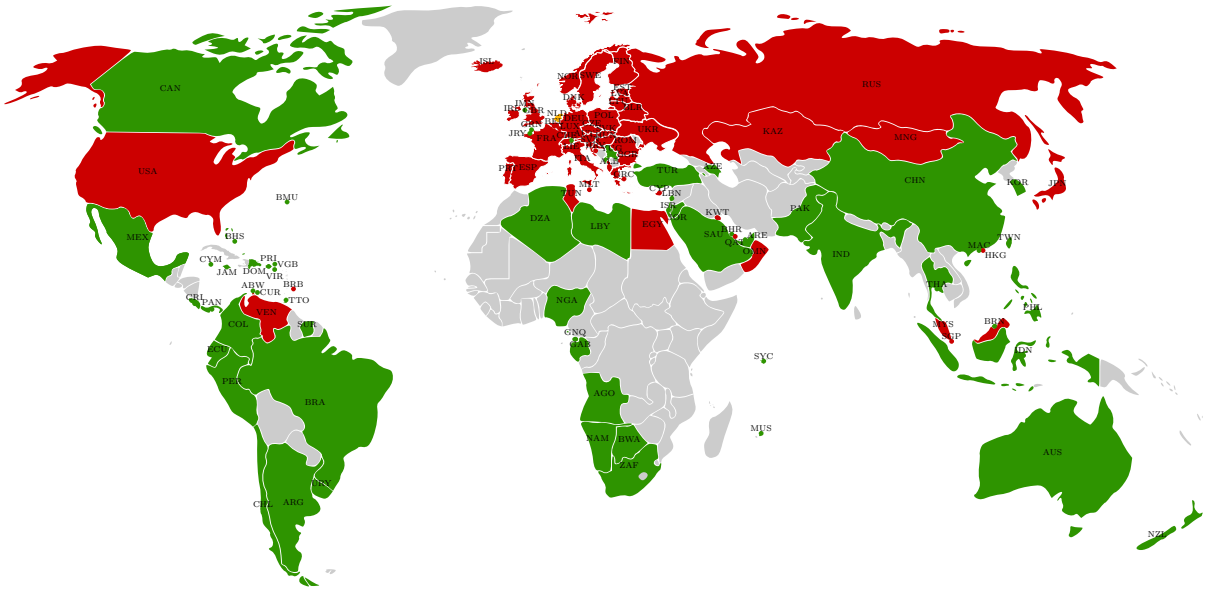


Figure 6.4: Green: the countries i to which the Netherlands has a non-zero *outgoing* tax t_{ci}. Red: the countries to which the Netherlands has *zero* as its *outgoing* tax rate.

We will test the greedy algorithm (Algorithm 6.4.1) in our network.

Algorithm 6.6.1 (Greedy algorithm 6.4.1 applied to the example of this section)**.** We iteratively set the reliability of edges (NLD, i_A) (where $i \in \mathcal{S}$) to 1 that increases the betweenness the most.

(i) First, we choose the edge (NLD, i_{1A}) with $i_1 \in \mathcal{S}$ such that the betweenness $B_{\text{after}}(\text{NLD})$ of NLD after setting this edge reliability to 1 is maximized. We set the reliability of this edge to 1 and we set $B_1(\text{NLD}) := B_{\text{after}}(\text{NLD})$.

(ii) Suppose $(\text{NLD}, i_{1A}), \ldots, (\text{NLD}, i_{j-1A})$ are chosen. Then we choose the edge (NLD, i_{jA}) with $i_j \in \mathcal{S}$ such that the betweennes $B_{\text{after}}(\text{NLD})$ of NLD after setting this edge reliability to 1 is maximized. We set the reliability of this edge to 1 and we set $B_j(\text{NLD}) := B_{\text{after}}(\text{NLD})$.

Here the 'betweenness centrality' is defined as in Section 2.4.3, since we have one different graph for every source vertex. This same procedure we can also apply to the *weighted* betweenness centrality.

Setting the outgoing tax to VIR[4], to zero (hence setting $r(\text{NLD}, \text{VIR}_A) := 1$) raises the unweighted betweenness the most, to 7.13537. To increase the weighted betweenness the most, one must set $r(\text{NLD}, \text{IND}_A) := 1$, giving the Netherlands a weighted betweenness of 8.26804. The following table shows the first ten steps in the greedy algorithm. The first line contains the

[4]The names of the countries corresponding to the three-letter codes used throughout the thesis, can be found in the Appendix A.1. VIR stands for: Virgin Islands (U.S.).

starting weighted betweenness (resp. unweighted betweenness) of NLD, before setting reliabilities of edges (NLD, i_A) to 1. After that, one-by-one edge reliabilities of edges (NLD, i_A) are set to 1 according to Algorithm 6.6.1.

Table 6.1: The first ten steps of Algorithm 6.6.1, for increasing the *weighted* betweenness centrality $B^W(\text{NLD})$ as well as the unweighted betweenness centrality $B(\text{NLD})$ of NLD.

Step	Quality α_k	Country	$B^W(\text{NLD})$	Country	$B(\text{NLD})$
			2.63225		6.27001
1	1.000	IND	8.26804	VIR	7.13537
2	.750	CHN	11.24343	BRA	7.99066
3	.703	BRA	13.98996	SUR	8.83870
4	.683	ARG	14.91397	ARG	9.66944
5	.672	AUS	15.64832	BHR	10.49580
6	.665	PAK	16.28557	IND	11.31937
7	.660	NGA	16.79352	NAM/(PAK)	12.14145
8	.656	PHL	17.25844	PAK/(NAM)	12.96354
9	.654	PER	17.61796	PER	13.78229
10	.651	SAU	17.96748	PRI	14.57842
		...	...	...	...
59	1.000	ALL	20.30425	ALL	21.87247

To increase the weighted betweenness, we first set $r(\text{NLD}, \text{IND}_A) := 1$. After that, we additionally set $r(\text{NLD}, \text{CHN}_A) := 1$ and we continue as in Table 6.1. To increase the unweighted betweenness, we first set $r(\text{NLD}, \text{VIR}_A) := 1$. After that, we additionally set $r(\text{NLD}, \text{BRA}_A) := 1$ and we continue as in Table 6.1. We observe that by setting the outgoing tax rates to India and China to zero, the Netherlands achieves a weighted betweenness centrality that is more than 50% of the maximum weighted betweenness ever achievable for the Netherlands by decreasing outgoing tax rates.

The second column of Table 6.1 contains a 'quality guarantee' α_k of the greedy solution, where it holds that

$$B_k(\text{NLD}) - B_0(\text{NLD}) \geq \alpha_k(\text{OPT} - B_0(\text{NLD})),$$

i.e. the total rise in betweenness that the k-th step of the greedy algorithm achieves, is *at least* α_k times the optimal increase in betweenness by setting k edge reliabilities of edges of the form (NLD, i_A) to 1. This follows from Theorem 6.4.6.

How precise is our approximation algorithm? We tested this for $k = 1, 2, 3$. We compute the maximal betweenness by setting edge reliabilities of k edges of the form (NLD, i_A) (with i_A such that $i \in \mathcal{S}$) to 1, by testing all subsets of $\mathcal{S}$ of size k. There are $\binom{59}{k}$ of such subsets. For $k = 1, 2, 3$ the optimal solution *exactly* corresponds with the solution given by the greedy algorithm, for *both* the weighted and the unweighted betweenness centrality. The greedy algorithm seems to be very useful in practice.

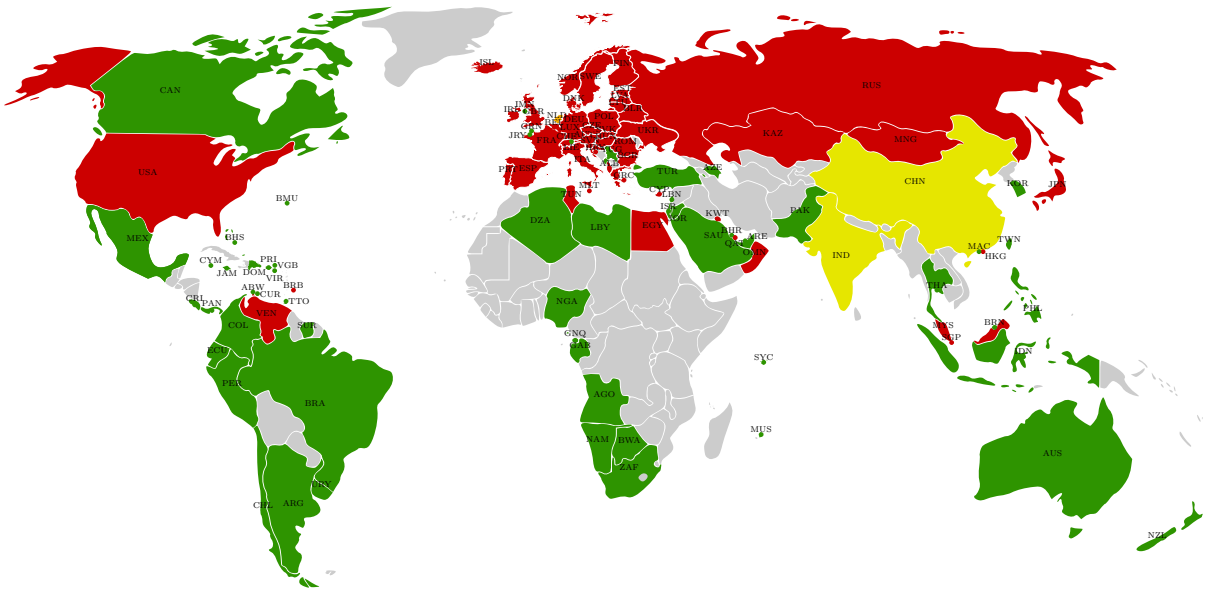


Figure 6.5: By setting the outgoing dividend tax rates to India and China to zero, the Netherlands achieves a weighted betweenness centrality that is more than 50% of the maximum weighted betweenness ever achievable for the Netherlands by decreasing outgoing tax rates. Green: the other countries (except India and China) to which the Netherlands has a non-zero *outgoing* tax rate.

Chapter 7

Maximizing tax revenues

The objective of this (short) chapter is maximizing the total tax a country receives as a conduit country. We will look at this problem from a theoretical perspective. This (short) chapter contains no *results* section: we will not apply the insights on the CPB-network (that consists of 108 graphs G_s).

In reality, the taxes that conduit countries receive when companies send money through them, are often not high (although this differs from country to country). The Netherlands, for example (see [RL13]), collects little tax from being a conduit country and might be more interested in creating and attracting jobs in the financial sector by increasing the amount of money that companies send through it (Chapter 6), than in raising (or maximizing) conduit taxes.

We formulate a 'tax problem' in *one* graph $G = (V, E)$. We will be concerned with *additive* edge costs and we will not make the translation to edge reliabilities.

Problem 9 (*Tax Problem*)**.** Let $G = (V, E)$ be a directed graph with additive edge costs $c : E \to \mathbb{Z}_{\geq 0}$, such that G does *not* contain cycles, even if we set the outgoing edges of a given vertex $u \in V$ to zero. The *total tax* that a vertex u receives is defined as

$$T_{\text{total}}(u) := \sum_{\substack{s,t \in V \\ s \neq u \neq t}} \sum_{v \in N^+(u)} f_{st}[(u,v)] \cdot c(u,v) = \sum_{\substack{s,t \in V \\ s \neq u \neq t}} \frac{w_{s,t}}{\sigma_{st}} \sum_{v \in N^+(u)} \sigma_{st}[(u,v)] c(u,v), \tag{7.1}$$

where $\sigma_{s,t}[(u,v)]$ is the number of shortest paths from s to t passing through edge (u,v). Suppose we are allowed to change the edge cost of, respectively

(i) only *one* outgoing edge of u,
(ii) k outgoing edges of u, with $1 \leq k \leq N^+(u) \leq |V|$,
(iii) all outgoing edges of u.

How must we set these edge costs (to nonnegative integers) to maximize the total tax $T_{\text{total}}(u)$ that country u receives?

Note that this 'total tax' is defined as the sum over all pairs $s, t \in V$ with $s \neq u \neq t$ of the tax received by u over the pair s, t. The tax received by u over the pair s, t we define as the sum over all outgoing edges (u, v) of u of the edge flow $f_{s,t}[(u, v)]$ multiplied by the 'tax rate' $c(u, v)$ on edge (u, v).

Observe that if there is an s, t-pair in G for which u is an intermediate country on *all* s, t paths, then there is an outneighbour v' of u for which u is an intermediate country on all s, v'-paths. We can give (u, v') an arbitrary high edge cost M, so that the obtained tax is at least $w_{u,v'} \cdot M$. This means that the tax that vertex u receives can become arbitrarily high. Therefore we assume from now on:

Assumption 7.0.1. For every s,t-pair in G for which u is an intermediate country on an s,t-path, there also exists an s,t-path not passing through u.

Remark 7.0.1. Note that we assumed that all edge costs are *nonnegative integers*. Suppose we would allow all numbers in $\mathbb{R}_{\geq 0}$ (such that G does not contain zero-cost cycles even after setting outgoing edges of u to zero). Then the *Tax Problem* has no solution. Consider the following graph.

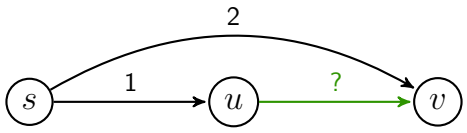


Figure 7.1: Example graph $G = (V, E)$. Suppose we allow nonnegative real numbers as edge costs. How do we need to set $c(u, v)$ to maximize tax?

In the above picture, setting $c(u, v) := 1$ yields a tax $T_{\text{total}}(u)$ of $1/2$. There are two shortest s, v-paths (of cost 1) and half of them pass through (u, v) with cost 1. Therefore the tax received 'through edge (u, v)' is $1 \cdot 1/2$ and (u, v) is the only outgoing edge of u. However, if we set $c(u, v) := 1 - \varepsilon$, then there is only *one* shortest s, v-path. The total tax received by vertex u will be $1 - \varepsilon$. As ε can be arbitrarily small, there is no maximum obtainable total tax, only an upperbound (of 1) that can be approximated arbitrarily close. Note that edge costs in $\mathbb{Q}_{\geq 0}$ do not help to solve this problem, as we can set $\varepsilon := 1/n$ and we still have the problem that there is no maximum obtainable tax.

To circumvent this problem, we assume in this chapter that all edge costs are *nonnegative integers*. The results from this chapter can be generalized to solve the *Tax Problem* in case the edge costs are not (nonnegative) integers, but are contained in $c \cdot \mathbb{Z}_{\geq 0}$, with $c > 0$ a real number. Note that we always assume that G has *no cycles of cost* 0, even after setting the outgoing edge costs of u to 0. We do this to be able to count paths efficiently (cf. Chapter 2).

In the following sections we will try to solve Problem 9 (i), (ii) and (iii).[1]

7.1 Tax problem: changing one edge cost

Suppose that we are allowed to change the edge cost of *one* outgoing edge of a vertex u. Consider first the case that we are *given* one particular outgoing edge of u. What edge cost do we need to give it?

Theorem 7.1.1. *Let $G = (V, E)$ be a directed graph with edge costs $c : E \to \mathbb{Z}_{\geq 0}$, such that G does not contain cycles of cost 0, even after setting the outgoing edges of a given vertex $u \in V$ to 0. Let $v \in N^+(u)$ be one* given *vertex. Suppose that it is allowed to change the edge cost of this particular edge (u, v). The aim is to change this edge cost in such a way that the resulting tax $T_{total}(u)$, as defined in (7.1), is maximized. To this end, we compute for each s,t-pair for which (u, v) lies on a path from s to t, with $s \neq u \neq t$, the value*

$$R_{s,t} := \delta_{E \setminus \{(u,v)\}}(s, t) - \delta(s, u) - \delta(v, t),$$

where $\delta_{E \setminus \{(u,v)\}}(s, t)$ stands for the distance from s to t in $G = (V, E \setminus \{(u, v)\})$. Note that in Assumption 7.0.1 we assumed that those $R^i_{s,t}$ values are finite if (but not only if) $\delta(s, u)$ and $\delta(v, t)$ are finite. If $\delta(s, u)$, $\delta(v, t)$ or $\delta_{E \setminus \{(u,v)\}}(s, t)$ is infinitely large, we define $R_{s,t}$ to be 0. To maximize the tax, one only needs to check, for all $s, t \in V$ with $s \neq u \neq t$, the nonnegative values among

[1] We will assume that the weights $w_{s,t}$ are all equal (and hence without loss of generality equal to 1), but the results that we will prove hold for arbitrary positive weights $w_{s,t}$.

$$c(u,v) = R_{s,t} + 1, \quad \textit{or} \quad c(u,v) = R_{s,t}, \quad \textit{or} \quad c(u,v) = R_{s,t} - 1.$$

If $\delta(s,u)$, $\delta(v,t)$ *or* $\delta_{E\setminus\{(u,v)\}}(s,t)$ *is infinitely large, we define* $R_{s,t}$ *to be* 0. *That is, only* $3\cdot|\{s,t \in V : s \neq u \neq t\}| = 3\cdot(n-1)\cdot(n-2) \leq 3n^2$ *edge costs need to be checked for* (u,v).

Proof. Let $M \in \mathbb{Z}_{\geq 0}$ be an edge cost such that $c(u,v) = M$ maximizes $T_{\text{total}}(u)$. If $M = R_{s,t} + 1$, $M = R_{s,t}$, or $M = R_{s,t} - 1$ for some pair s,t, then we are done. Therefore, suppose that this is not the case.

(1.) Suppose that there is a pair $s,t \in V$ for which $M < R_{s,t} - 1$. Then s_1, t_1 can be chosen such that R_{s_1,t_1} is the smallest R-value for which $R-1$ is *larger* than M, i.e.

$$M < R_{s_1,t_1} - 1,$$

and for all s,t such that $R_{s,t} - 1 > M$ we have $R_{s,t} \geq R_{s_1,t_1}$. Note that u will receive no tax via vertex v[2] on all routes s,t for which $R_{s,t} < M$. However, u will receive a tax of $M < R_{s_1,t_1} - 1$ on all routes $R_{s,t}$ for which $R_{s,t} - 1 > M$ and (u,v) lies on the shortest path from s to t. But resetting $c(u,v) := R_{s_1,t_1} - 1$ can only possibly *increase* the tax that u receives via vertex v, while keeping the tax that u receives via all other outgoing vertices equal. To see this, note that exactly all shortest paths through u remain shortest paths when changing $c(u,v)$ from M to $R_{s_1,t_1} - 1$. The shortest paths through (u,v) remain shortest paths by construction (since by changing the edge cost from M to $R_{s_1,t_1} - 1$, the edge cost of (u,v) remains below the smallest R-value above M). The shortest paths through u *not* crossing (u,v) also remain shortest paths if we increase the edge cost of (u,v) (increasing $c(u,v)$ could only possibly increase the distances in G, but paths not crossing (u,v) already have a given cost). In the computation of $T_{total}(u)$ all variables remain the same, except the cost $c(u,v)$. This cost rises from M to $R_{s_1,t_1} - 1$, yielding a possibly higher tax.

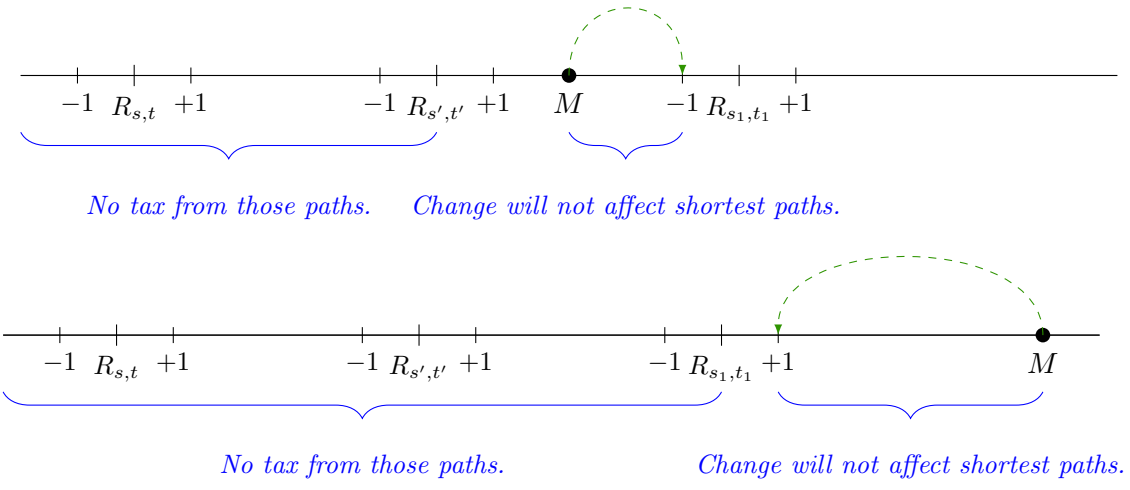


Figure 7.2: Case (1.) (above) and (2.) (below) of the proof of Theorem 7.1.1.

[2]Naturally, we define the tax that u receives 'via' vertex v (with $v \in N^+(u)$) as

$$\sum_{s,t \in V} \frac{w_{s,t}}{\sigma_{s,t}} \sigma_{st}((u,v)) \cdot c(u,v).$$

This is exactly the contribution that edge (u,v) makes to $T_{total}(u)$, see (7.1).

(2.) Suppose that there does *not* exist a pair $R_{s,t}$ for which $M < R_{s,t} - 1$. Then it holds for all pairs s, t that $R_{s,t} + 1 < M$. Let R_{s_1,t_1} be the largest such pair, i.e.

$$M > R_{s_1,t_1} + 1,$$

and for all s, t such that $R_{s,t} + 1 < M$ we have $R_{s,t} \leq R_{s_1,t_1}$. Note that u does not obtain any tax via v, as $M > R_{s,t}$ for each pair s, t. By resetting $M := R_{s_1,t_1} + 1$ we will still have that $M > R_{s,t}$ for all s, t-pairs (so u will receive no tax via v), while all shortest paths remain unchanged. So the tax that u receives via all other vertices remains the same. Therefore we can set $M := R_{s_1,t_1} + 1$.

We conclude that there always exists some pair s, t for which we can set $c(u, v) = R_{s,t} - 1$, $c(u, v) = R_{s,t}$ or $c(u, v) = R_{s,t} + 1$ to maximize the tax. Therefore the theorem is true. □

The above theorem suggests a polynomial time algorithm for calculating the edge cost of one given edge to maximize $T_{\text{total}}(u)$.

Remark 7.1.1. To maximize $T_{\text{total}}(u)$ in polynomial time by changing the edge cost of one *given* edge we perform the following steps.

(i) First we want to compute the values $R_{s,t}$ for each pair s, t for which u is an intermediate node on a path from s to t, but we can as well compute $R_{s,t}$ for *all* pairs s, t. Using Floyd Warshall this can be done in $\mathcal{O}(n^3)$ (or in our CPB-network that consists of n graphs in $\mathcal{O}(n^4)$).

(ii) Try, for all s, t-pairs, all values $c(u, v) := R_{s,t} - 1$, $c(u, v) := R_{s,t}$ and $c(u, v) := R_{s,t} + 1$ and compute the resulting tax. Note that, given all edge costs, the tax T_{total} can be computed in the same time as a normal betweenness computation, by using Brandes' equation. Equation (2.6) becomes

$$T_{s,\bullet}(u) = \sum_{v\,:\,u \in P_s(v)} \frac{c(u,v) \cdot w_{s,v} \cdot \sigma_{su}}{\sigma_{sv}} \cdot \left(1 + \frac{B^W_{s,\bullet}(v)}{w_{s,v}}\right),$$

and by summing the values $T_{s,\bullet}$ over all $s \in V \setminus \{u\}$, we obtain the tax $T_{\text{total}}(u)$. This step (at most $3n^2$ betweenness computations) can be done in time bounded by $\mathcal{O}(3n^2 \cdot n(m + n \log n)) = \mathcal{O}(n^5)$.

(iii) The value of $c(u, v)$ that gives maximum $T_{\text{total}}(u)$ now maximizes $T_{\text{total}}(u)$, as seen in Theorem 7.1.1.

We conclude that we can maximize $T_{\text{total}}(u)$ in polynomial time by changing the edge cost of one *given* edge in time bounded by $\mathcal{O}(n^5)$.

Corollary 7.1.2. *If we are allowed to change the cost of one* arbitrary *edge (u, v) with $v \in N^+(u)$, we just run the above procedure $N^+(u) \leq |V| = n$ times (for each outgoing edge of u one time) and take the edge (with corresponding edge cost) that maximizes $T_{total}(u)$ the most. Therefore, we can maximize the total tax in polynomial time by changing the edge cost of one arbitrary outgoing edge of u in time $\mathcal{O}(n \cdot n^5) = \mathcal{O}(n^6)$.*

We conclude that Problem 9 (*i*), the *Tax Problem* with one outgoing edge, can be solved in polynomial time, as desired.

7.2 Tax problem: changing k edge costs

In this section we consider Problem 9 (*ii*), the *Tax Problem* of changing k edge costs. This problem is a bit similar to Problem 6 (*Maximizing Betweenness*) of the Chapter 6: it is *NP*-hard, via a reduction to *Maximum Coverage*.

Theorem 7.2.1. *The* Tax Problem *of changing at most k edge costs, Problem* 9 (*ii*)*, is NP-hard.*

Proof. We use a reduction from the *Maximum Coverage Problem* (Problem 7). Given a universe $\mathcal{U} = \{e_1, \ldots, e_N\}$, a collection of subsets of this universe $S = S_1, \ldots, S_t$ and a number k, we construct an instance of Problem 9 (*ii*) as follows:

Let there be vertices $s, u, v_1, \ldots, v_t$ (so for every set $S_i \in S$ there is a vertex v_i) and let there also be a vertex for each element in $\mathcal{U}$:

$$V = \{s, u, v_1, \ldots, v_t\} \cup (\cup_{e_i \in \mathcal{U}} e_i)\,.$$

Let there be directed edges as follows:

- There is an edge (s, u) of cost 0.
- For every v_i, where $i = 1, \ldots, m$, there is an edge (u, v_i) of cost 0.
- For every v_i, where $i = 1, \ldots, m$, there is an edge (s, v_i) of cost $2Nk + 1$.
- For every $x \in \mathcal{U}$, and for every $i \in \{1, \ldots, m\}$ there is an edge (v_i, x) if and only if $x \in S_i$. This edge has cost 0.
- For every $x \in \mathcal{U}$ there is an edge (s, x). This edge has cost $2Nk + 1$.

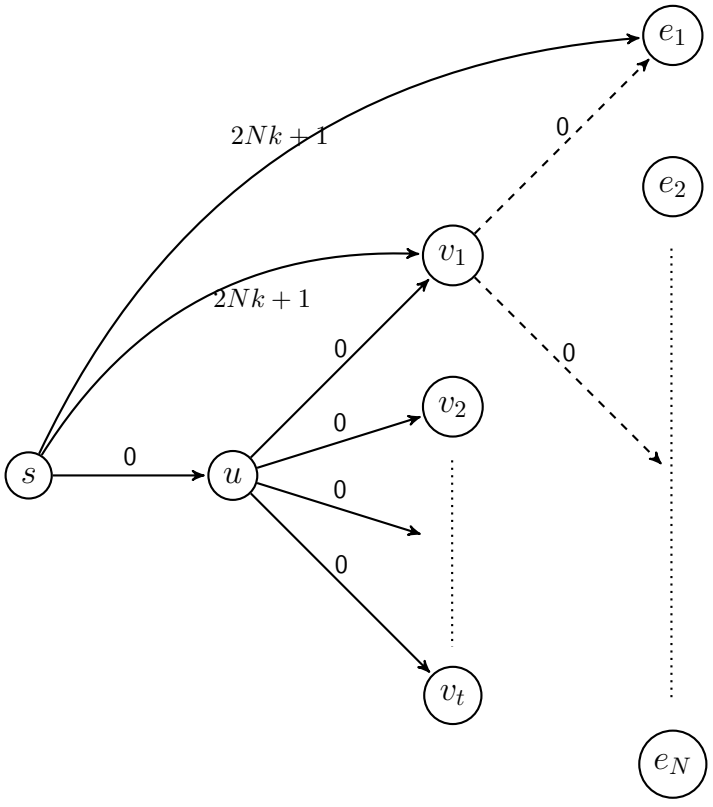


Figure 7.3: Illustration of the instance created in the reduction of *Maximum Coverage* to the *Tax Problem* of changing k outgoing edge costs (Problem 9 (*ii*)).

Before changing outgoing edge costs of vertex u, the total tax that u receives is 0. To maximize tax, we want to set k edge costs $c(u, v_{i_1}), \ldots, c(u, v_{i_k})$ to $2Nk$ so that $|S_{i_1} \cup \ldots \cup S_{i_k}|$ is maximal. To see this, note that setting k edge costs $c(u, v_{i_1}), \ldots, c(u, v_{i_k})$ to $2Nk$ gives a tax of *exactly*

$$T_1 := 2Nk^2 + 2Nk \cdot |S_{i_1} \cup \ldots \cup S_{i_k}|. \tag{7.2}$$

For all pairs $s, v_{i_1}, \ldots, s, v_{i_k}$, the tax that u receives via this pair is $2Nk$, which gives a combined tax via these pairs of $2Nk^2$. For each $x \in S_{i_1} \cup \ldots \cup S_{i_k}$, the tax that u receives via x is $2Nk$, giving a total tax of $2Nk \cdot |S_{i_1} \cup \ldots \cup S_{i_k}|$ for these s, x-pairs. We conclude that the combined tax received by u is as in (7.2). In contrast, setting a edges (with $a \geq 1$) to $2Nk+1$ and b edges to $2Nk$, with $a + b = k$, gives (at most) a tax of T_2, where

$$\begin{aligned} T_2 &\leq \frac{2Nk+1}{2} \cdot a + 2Nkb + (2Nk+1) \cdot |S_{i_1} \cup \ldots \cup S_{i_k}| \\ &< (Nk+1) \cdot a + 2Nkb + N + 2Nk \cdot |S_{i_1} \cup \ldots \cup S_{i_k}| \\ &\leq Nka + a + N + 2Nkb + 2Nk \cdot |S_{i_1} \cup \ldots \cup S_{i_k}| \\ &< 2Nka + 2Nkb + 2Nk \cdot |S_{i_1} \cup \ldots \cup S_{i_k}| \leq 2Nk^2 + 2Nk \cdot |S_{i_1} \cup \ldots \cup S_{i_k}| \leq T_1, \end{aligned}$$

where we assume without loss of generality that $k \geq 2$ and $N \geq 3$ (so that $a < k \leq Nk/2$ and $N \leq Nk/2$). This is a strictly smaller tax than T_1. Also, an optimal solution will contain no edge costs with a value smaller than $2Nk$ (since then we could as well give all edge costs that are assigned a value smaller than $2Nk$ the value $2Nk$ to strictly increase tax).

It follows that the tax is maximized if and only if we have set k outgoing edge costs of u to $2Nk$, where we have chosen these k outgoing edges such that $|S_{i_1} \cup \ldots \cup S_{i_k}|$ is maximal. We conclude that $S_{i_1}, \ldots, S_{i_k}$ is an optimal solution for *Maximum Coverage* if and only if setting the edge costs on $(u, v_{i_1}), \ldots, (u, v_{i_k})$ to $2Nk$ maximizes the tax in the created instance of Problem 9 (ii).

This gives a reduction from Problem 9(ii) to the *Maximum Coverage Problem* (Problem 7). Note that this reduction can be done in time $\mathcal{O}(t \cdot N)$, which is polynomial in the size of the instance. Therefore Problem 9 (ii) is *NP*-hard. □

Will a greedy approach similar to Algorithm 6.4.1, the greedy algorithm for *Maximizing Betweenness* (Problem 6), give good results in the case of maximizing taxes? First, we choose the outgoing edge (u, v_i) that maximizes tax and we set it to a value maximizing the tax. Then we continue iteratively as in Algorithm 6.4.1. Unfortunately, the greedy approach is a *very* bad approach, as we see in the following example.

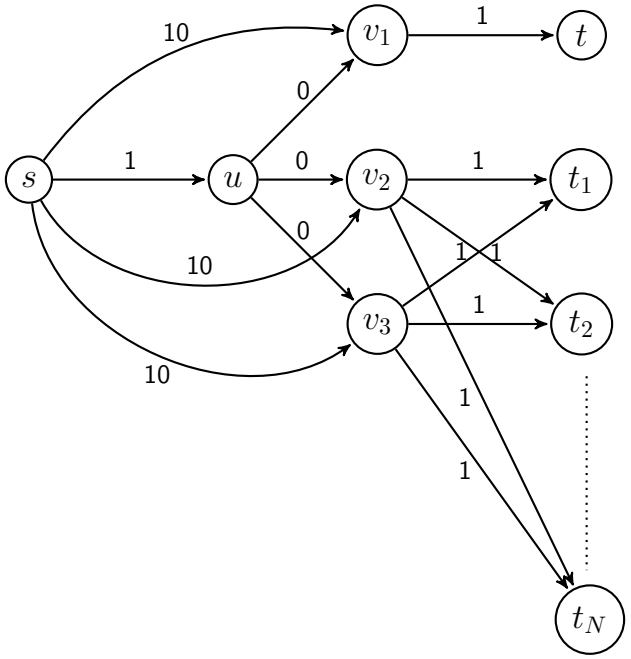


Figure 7.4: Example graph $G = (V, E)$, with $k = 2$. The greedy approach is not at all optimal here.

Example 7.2.1. Consider the graph of Figure 7.4. The tax that u receives in the initial position is zero. The greedy approach will first set edge (u, v_1) to 8, increasing the tax u receives both on routes s, v_1 and s, t by by 8. This leads to a total increase of tax of $8 + 8 = 16$. After that, the greedy approach will either set (u, v_2) or (u, v_3) to 8, giving an additional tax increase of 8, so the total received tax is $16 + 8 = 24$. But it is easy to verify that setting both (u, v_2) and (u, v_3) to 8 will give a total tax of $2 \cdot 8 + N \cdot 8$. Therefore the optimal solution can be arbitrarily many times larger than the greedy solution, since

$$\frac{2 \cdot 8 + N \cdot 8}{24} \xrightarrow{N \to \infty} \infty.$$

The greedy approach, which gave good results in the betweenness-maximizing problem, does not help here at all.

Here we conclude this section: the *Tax Problem* of changing k edge costs (Problem 9 (ii)) is *NP*-hard, and the greedy approach analogous to Algorithm 6.4.1 can give arbitrarily bad results.

7.3 Maximizing the tax over one pair s, t

In this section we make a (minor) start with considering Problem 9 (iii). We will maximize the tax obtained by u over *one* s, t-pair: we are allowed to change the edge costs of *all* outgoing edges of one node u, in order to maximize

$$\sum_{v \in N^+(u)} f_{st}[(u, v)] \cdot c(u, v) = \frac{w_{s,t}}{\sigma_{st}} \sum_{v \in N^+(u)} \sigma_{st}[(u, v)] c(u, v).$$

Since $w_{s,t}$ is a positive constant, we can assume that it is 1 in solving this problem.

Algorithm 7.3.1 (Maximizing the tax over *one* s, t-pair)**.** We compute the minimum distance of a path from s to t not crossing u, which we denote by $\delta_{\{V \setminus \{u\}\}}$. We compute, for each vertex $v \in N^+(u)$, the value

$$h(v) := \delta_{\{V \setminus \{u\}}(s, t) - \delta(s, u) - \delta(v_i, t).$$

We make a case distinction.

(1.) If

$$\max\{h(v) : v \in N^+(u)\} \leq 0,$$

then no positive tax can be obtained and we stop the procedure.

(2.) Else:

(i.) We calculate the value

$$C_1 := \max\{h(v) : v \in N^+(u)\} - 1,$$

and we store the edges (u, v) for which $h(v) - 1$ equals this maximum. We denote the set of $v \in N^+(u)$ with $c(u, v) = h(v)$ by U.

(ii.) For all vertices $v \in U$, compute the number of v, t-paths and calculate the quantity

$$C_2 = \frac{\sum_{v \in U} \sigma_{st}[(u, v)] \cdot c(u, v)}{\sigma_{st}(u^-) + \sum_{v \in U} \sigma_{st}[(u, v)]}. \tag{7.3}$$

Now, the maximum tax that can be obtained is $\max\{C_1, C_2\}$. If $C_1 > C_2$, we set $c(u, v) := C_1$ for an arbitrary vertex with $h(v) - 1 = C_1$. The other edge costs $c(u, v)$ we assign a value larger than $h(v)$. If $C_2 > C_1$, we set $c(u, v) := h(u)$ for all $v \in U$. The other edge costs $c(u, v)$ we assign a value larger than $h(v)$.

The procedure returns the edge costs and the resulting tax value.

Theorem 7.3.1. *Algorithm* 7.3.1 *correctly returns the edge costs that maximize the tax that* u *receives over the pair* s, t *in* $\mathcal{O}(n^3)$.

Proof. We examine all steps of the algorithm.

(1.) If

$$\max\{\delta_{\{V\setminus\{u\}\}}(s,t) - \delta(s,u) - \delta(v',t) \quad : v' \in N^+(u)\} \leq 0,$$

then no positive tax can be obtained: no paths will pass through any outgoing edge of (u, v), regardless of how we set the edge cost. We can quit the procedure.

(i.) What is the maximum tax that can be obtained by making the distance $\delta(s, t)$ *strictly shorter* than $\delta_{\{V\setminus\{u\}\}}(s, t)$? To compute this, we calculate the value

$$C_1 := \max\{\delta_{\{V\setminus\{u\}\}}(s,t) - \delta(s,u) - \delta(v',t) \ : v' \in N^+(u)\} - 1.$$

Choose the vertex v' for which this maximum is obtained. Setting the edge cost of (u, v') to C_1 will yield a tax of C_1, since all shortest paths from s to t will pass through this edge. The other edge costs must be set to a higher value than C_1. In this case we obtain a total tax over pair s, t of C_1.

(ii.) Could we obtain a higher tax than C_1 by *equalizing* the distance[3]? For all vertices $v \in U$, the set of vertices with $h(v) = C_1 + 1$ (which is the largest possible edge cost of an outgoing edge of u so that u receives some tax via this edge), we compute the number of v, t-paths and calculate the quantity (using that $\sigma_{st}[(u, v)] = \sigma_{su} \cdot \sigma_{vt}$ for $v \in U$ if we set $c(u, v) := C_1 + 1$)

$$C_2 = \frac{\sum_{v\in U} \sigma_{st}[(u,v)] \cdot c(u,v)}{\sigma_{st}(u^-) + \sum_{v\in U} \sigma_{st}[(u,v)]} = \frac{\sum_{v\in U} \sigma_{st}[(u,v)] \cdot (C_1+1)}{\sigma_{st}(u^-) + \sum_{v\in U} \sigma_{st}[(u,v)]}. \tag{7.4}$$

We could only get a tax value *larger than* C_1 if there are many shortest path passing through u via edges (u, v) with $v \in U$: there must be so many of these paths that (7.4) yields a value in the interval $(C_1, C_1 + 1)$. Note that setting an outgoing edge to C_1 or a smaller value can never help by achieving a tax inside the interval $(C_1, C_1 + 1)$. Also note that we can, for computing (7.4) set as well *all* costs of outgoing edges of u to $C_1 + 1$. Edges (u, v) with $v \notin U$ will in this case not contribute to the sum: no shortest paths through u will pass to them.

Now, the maximum tax that can be obtained is the maximum tax that results from one of the above two situations. Therefore Algorithm 7.3.1 computes the correct edge costs to maximize the tax obtainable by u over the pair s, t.

The running time of Algorithm 7.3.1 is dominated by the calculation of the distances, which can be done with Floyd-Warshall in time $\mathcal{O}(n^3)$. We conclude that the algorithm terminates in $\mathcal{O}(n^3)$. □

The last problem we would like to consider is maximizing the *total* tax vertex u receives over *all* s, t-pairs (Problem 9 (*iii*)) where it is allowed to change *all* outgoing edge costs of u. Unfortunately, we did not find a solution for this problem, and there is doubt that this is an easy problem. However, a proof that this probem is *NP*-hard was still not found. This is an interesting question for further research.

With these notes we end our short theoretical expedition to an invented problem, the *tax*-problem. In the next section we will return to the CPB-network: we will compute the *Shapley-value based betweenness centrality*, an alternative measure for betweenness centrality.

[3]By *equalizing* we mean: setting the outgoing edge costs of u such that $\delta(s, t) = \delta_{\{V\setminus\{u\}\}}(s, t)$.

Chapter 8

Shapley-value based betweenness centrality

In this chapter we give an alternative approach of betweenness centrality, based on the Shapley value. This chapter is based on a recent article by P. L. Szczepánski, T. Michalak and T. Rahwan (see [SMR12]), with some minor additions to compute the Shapley-value based *weighted* betweenness centrality. First we study the theory and then we test the Shapley-value based betweenness centrality on the CPB-network of 108 countries.

8.1 Shapley-value based betweenness centrality: an algorithm

The *Shapley-value based betweenness centrality* is a special way of measuring the importance of a vertex in a network. The original betweenness centrality measures importance of an individual vertex. How severe are the consequences for the possibility to communicate between vertices in the network if this particular node fails? It is argued (see [SMR12]) that the original betweenness centrality is not an adequate measure for many applications, since in practice many nodes can fail simultanuously.

Example 8.1.1. Consider the following graph (example taken from [SMR12]).

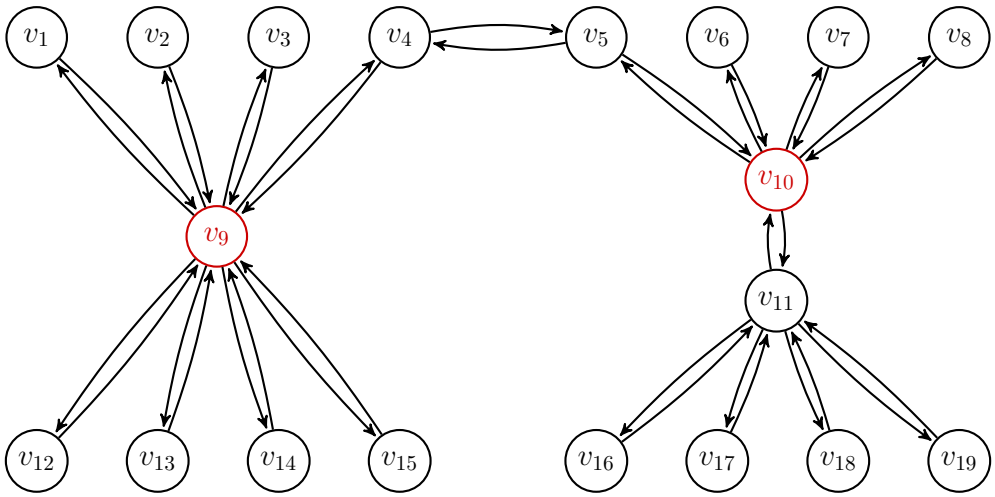


Figure 8.1: Example graph $G = (V, E)$ illustrating the difference between betweenness centrality and *the Shapley-value based* betweenness centrality. We assume that for all $e \in E$, the reliability $r(e) := 1/2$. The betweenness centralities of nodes v_9 and v_{10} are equal, while the Shapley-value based betweenness centralities are different.

When calculating betweenness centrality, we find that $B_G(v_9) = 98 = B_G(v_{10})$. This is not accurate for measuring the connectivity between nodes in the network, since the failure of vertex v_9 has more disastrous consequences for this connectivity than the failure of vertex v_{10}. For example, if v_9 fails, the vertices v_{12}, v_{13}, v_{14} and v_{15} cannot communicate with each other, while if v_{10} fails, the vertices v_{16}, v_{17}, v_{18} and v_{19} are still able to communicate with each other. The Shapley-value based betweenness centrality $SH_B : V \to \mathbb{R}$ (which will be defined below) is able to reflect this difference between v_9 and v_{10}. It holds that $SH_B(v_9) = 18.2$, while $SH_B(v_{10}) = 16.0833$.

The Shapley-value based betweenness centrality is a measure based on the importance of a vertex *as a member* of all possible subsets of vertices in G. In order to define and compute the Shapley-value based betweenness, we first define *group betweenness centrality*. Group betweenness centrality measures the importance of subset of vertices (a 'group' of vertices) in a graph, and was first defined by M.G. Everett and S.P. Borgatti (see [EB99]).

Definition 8.1.1 (Group betweenness centrality)**.** For a subset $S \subseteq V$, we define the *group betweenness centrality* of S as

$$B_G(S) = \sum_{\substack{s \notin S, \\ t \notin S}} \frac{\sigma_{st}(S)}{\sigma_{st}}.$$

Similarly we define

$$B_G^W(S) = \sum_{\substack{s \notin S, \\ t \notin S}} \frac{w_{s,t} \cdot \sigma_{st}(S)}{\sigma_{st}},$$

to be the *weighted group betweenness centrality*.

We write $\pi \in \Pi(V)$ for a permutation of the vertices V. Furthermore, we denote by $C_\pi(v) \subseteq V$ the subset of V that consists of all predecessesors of vertex v in π, i.e. if $\pi(v')$ is the location of vertex v' in π, then

$$C_\pi(v) = \{v' \in \pi \ : \ \pi(v') < \pi(v)\}.$$

Now we can define the Shapley-value based betweenness centrality.

Definition 8.1.2 (Shapley-value based betweenness centrality)**.** The *Shapley-value based betweenness centrality* is

$$SH_B(v) := \frac{1}{|V|!} \sum_{\pi \in \Pi} \left[B_G(C_\pi(v) \cup \{v\}) - B_G(C_\pi(v)) \right].$$

This is the average marginal contribution of vertex v to $C_\pi(i)$ over all permutations $\pi \in \Pi$. It can be rewritten as

$$SH_B(v) = \sum_{S \subseteq V \setminus \{v\}} \frac{|S|!(|V| - |S| - 1)!}{|V|!} \left[B_G(S \cup \{v\}) - B_G(S) \right]. \tag{8.1}$$

Similarly, the Shapley-value based *weighted* betweenness centrality can be defined by replacing the betweenness with the weighted betweenness in the above formulas. Therefore the Shapley-value based *weighted* betweenness cenrality can be written as

$$SH_{B^W}(v) = \sum_{S \subseteq V \setminus \{v\}} \frac{|S|!(|A| - |S| - 1)!}{|A|!} \left[B_G^W(S \cup \{v\}) - B_G^W(S) \right]. \tag{8.2}$$

We would like to calculate the Shapley-value based betweenness centrality efficiently. We could of course analyse all $2^{|V|}$ sets S, but this will not result in a polynomial time algorithm. Therefore we will use a trick. For each group $C_\pi(v)$ we consider the average marginal contribution a path P through v has to the group betweenness centrality $B_G(C_\pi(v) \cup \{v\}) - B_G(C_\pi(v))$.

(+) A path P in $\mathcal{P}_{s,t}(v)$ (the collection of paths from s to t through v, with $s \neq v \neq t$) has a *positive* contribution to $C_\pi(v)$ exactly if this path does not contain a vertex of $C_\pi(v)$. This contribution will equal $1/\sigma_{st}$, and in the weighted betweenness case $w_{s,t}/\sigma_{st}$. More formally, the path P has a positive contribution to $C_\pi(v)$ if and only if $C_\pi(v) \cap V(P) = \emptyset$, where $V(P)$ is the set of vertices contained in the path P.

We introduce a Bernoulli random variable $B^+_{v,P}$ which indicates whether vertex v contributes positively through path P to set $C_\pi(v)$. Then we have

$$\mathbb{E}\left[B^+_{v,P}\right] = \mathbb{P}\left[C_\pi(v) \cap V(P) = \emptyset\right].$$

This is the probability that all vertices in $|V(P)| \setminus \{v\}$ are not contained in $C_\pi(v)$,[1] i.e. this is the probability that v precedes all other vertices from the path P in a random permutation π of V. This probability is exactly $1/|V(P)|$, as we will prove in an auxiliary lemma.

Lemma 8.1.1. *Let $v \in V$. It holds that $\mathbb{P}[\forall\, v' \in V(P) \setminus \{v\} \ : \ \pi(v') > \pi(v)] = 1/|V(P)|$.*

Proof. We simply count the permutations $\pi \in \Pi(V)$ that satisfy $\forall\, v' \in V(P) \setminus \{v\} \ : \ \pi(v') > \pi(v)$. We first choose $|V(P)|$ positions in the sequence of all elements from $|V|$. There are

$$\binom{|V|}{|V(P)|}$$

posibilities. In the last $|V(P)| - 1$ positions of the chosen positions, we place all elements from $V(P) \setminus \{v\}$, and in the first position of the chosen positions, we place vertex v. The number of possibilities is $(V(P)| - 1)!$. Finally, we can arrange the remaining elements (i.e. the elements in $V \setminus V(P)$) of V in $(|V| - |V(P)|)!$ possible ways. Summarizing, the number of permutations $\pi \in \Pi(V)$ that satisfy $\pi(v') > \pi(v)$ for all $v' \in V(P) \setminus \{v\}$, is

$$\binom{|V|}{|V(P)|} \cdot (|V(P)| - 1)! \cdot (|V| - |V(P)|)! = \frac{|V|! \cdot (|V(P)| - 1)!}{|V(P)|!} = \frac{|V|!}{|V(P)|}.$$

Since there are $|V|!$ possible permutations of $|V|$, it follows that

$$\mathbb{P}[\forall\, v' \in V(P) \setminus \{v\} \ : \ \pi(v') > \pi(v)] = \frac{1}{|V(P)|},$$

as desired. □

It follows that

$$\mathbb{E}\left[B^+_{v,P}\right] = \mathbb{P}\left[C_\pi(v) \cap V(P) = \emptyset\right] = \frac{1}{|V(P)|},$$

for a path $P \in \mathcal{P}_{s,t}(v)$ (with $s \neq v \neq t$).

(–) It is also possible that a path P has *negative* contribution to the term corresponding to $C_\pi(v)$ in the Shapley value. This happens if path P starts or ends with v and contains already a vertex from $C_\pi(v)$, i.e. the path already contributes with a factor $1/\sigma_{sv}$ or $1/\sigma_{vt}$ (in the weighted betweenness case $w_{s,v}/\sigma_{sv}$ or $w_{v,t}/\sigma_{vt}$) to the group betweenness centrality. By adding v, the set $C_\pi(v) \cup \{v\}$ will contain an endpoint of the path, which means that the contribution to the group betweenness centrality of this path after adding v will be zero. Therefore, adding v has a negative effect. The negative contribution of v to $C_\pi(v)$ through path P is $-1/\sigma_{sv}$ resp. $-1/\sigma_{vt}$ (in the weighted betweenness case $-w_{s,v}/\sigma_{sv}$ resp. $-w_{v,t}/\sigma_{vt}$).

[1] Note that by definition $v \notin C_\pi(v)$.

We will consider the probability that a path P *starting or ending* at v has a negative contribution to the term corresponding to $C_\pi(v)$ in the Shapley-value based betweenness centrality, by first considering the probability of the complementary event: the event that a path starting or ending at v has a *neutral* contribution to $C_\pi(v)$.

- Suppose a path P starts or ends at v. Then there is a probability vertex v has a *neutral* contribution to $C_\pi(v)$ through path P, i.e. no contribution at all. This is the case:
  (i) if the other endpoint, the endpoint of P that is not v, is in $C_\pi(v)$. This happens with probability $1/2$.
  (ii) or if the path P does not contain any vertex of $C_\pi(v)$. This happens with probability $1/|V(P)|$.

  Since both events are disjoint[2], the probability that P makes a neutral contribution to $C_\pi(v)$ equals

$$\mathbb{P}\left[\text{endpoint other than } v \ \in C_\pi(v) \text{ or } C_\pi(v) \cap V(P) = \emptyset\right] = \frac{1}{2} + \frac{1}{V|P|}.$$

(–) The probability that v has a *negative* contribution to $C_\pi(v)$ through path P is therefore

$$\begin{aligned}\mathbb{E}\left[B^-_{v,P}\right] &= 1 - \mathbb{P}\left[\text{path } P \text{ has a negative contribution to } C_\pi(v)\right] \\ &= 1 - \left(\frac{1}{2} - \frac{1}{|V(P)|}\right) \\ &= \frac{1}{2} - \frac{1}{|V(P)|}.\end{aligned}$$

Now we can compute the Shapley-value based betweenness centrality more easily. We have

$$\begin{aligned}SH_B(v) = &\sum_{\substack{s,t\in V\\ s\neq v\neq t}} \sum_{P\in\mathcal{P}_{s,t}(v)} \frac{1}{\sigma_{st}} \mathbb{E}\left[B^+_{v,P}\right] + \sum_{\substack{s\in V\\ s\neq v}} \sum_{P\in\mathcal{P}_{s,v}} \left(-\frac{1}{\sigma_{sv}}\right) \mathbb{E}\left[B^-_{v,P}\right] \\ &+ \sum_{\substack{t\in V\\ t\neq v}} \sum_{P\in\mathcal{P}_{v,t}} \left(-\frac{1}{\sigma_{vt}}\right) \mathbb{E}\left[B^-_{v,P}\right] \\ = &\sum_{\substack{s,t\in V\\ s\neq v\neq t}} \sum_{P\in\mathcal{P}_{s,t}(v)} \frac{1}{\sigma_{st}} \frac{1}{|V(P)|} + \sum_{\substack{s\in V\\ s\neq v}} \sum_{P\in\mathcal{P}_{s,v}} \left(-\frac{1}{\sigma_{sv}}\right) \left(\frac{1}{2} - \frac{1}{|V(P)|}\right) \\ &+ \sum_{\substack{t\in V\\ t\neq v}} \sum_{P\in\mathcal{P}_{v,t}} \left(-\frac{1}{\sigma_{vt}}\right) \left(\frac{1}{2} - \frac{1}{|V(P)|}\right).\end{aligned}$$

[2]Both events cannot occur simultanuously.

Simplifying this equality gives

$$\begin{aligned} SH_B(v) &= \sum_{\substack{s,t\in V\\ s\neq v\neq t}} \sum_{P\in\mathcal{P}_{s,t}(v)} \frac{1}{\sigma_{st}\cdot|V(P)|} + \sum_{\substack{s\in V\\ s\neq v}} \left(\sum_{P\in\mathcal{P}_{s,v}} \frac{1}{\sigma_{sv}\cdot|V(P)|} - \sum_{P\in\mathcal{P}_{s,v}} \frac{1}{\sigma_{sv}}\cdot\frac{1}{2} \right) \\ &\quad + \sum_{\substack{t\in V\\ t\neq v}} \left(\sum_{P\in\mathcal{P}_{v,t}} \frac{1}{\sigma_{vt}\cdot|V(P)|} - \sum_{P\in\mathcal{P}_{v,t}} \frac{1}{\sigma_{vt}}\cdot\frac{1}{2} \right) \\ &= \sum_{\substack{s,t\in V\\ s\neq v\neq t}} \sum_{P\in\mathcal{P}_{s,t}(v)} \frac{1}{\sigma_{st}\cdot|V(P)|} + \sum_{\substack{s\in V\\ s\neq v}} \left(\sum_{P\in\mathcal{P}_{s,v}} \frac{1}{\sigma_{sv}\cdot|V(P)|} - \frac{1}{2} \right) \\ &\quad + \sum_{\substack{t\in V\\ t\neq v}} \left(\sum_{P\in\mathcal{P}_{v,t}} \frac{1}{\sigma_{vt}\cdot|V(P)|} - \frac{1}{2} \right) \end{aligned} \tag{8.3}$$

We define

$$\Upsilon_{st} = \sum_{P\in\mathcal{P}_{s,t}} \frac{1}{|V(P)|} \quad \text{and } \Upsilon_{st}(v) = \sum_{\substack{P\in\mathcal{P}_{s,t}(v)\\ v\neq s\neq t}} \frac{1}{|V(P)|}.$$

Using these definitions, we can again simplify the expression for the Shapley-value based betweenness centrality (8.3):

$$\begin{aligned} SH_B(v) &= \sum_{\substack{s,t\in V\\ s\neq v\neq t}} \frac{\Upsilon_{st}(v)}{\sigma_{st}} + \sum_{\substack{s\in V\\ s\neq v}} \left(\frac{\Upsilon_{sv}}{\sigma_{sv}} - \frac{1}{2} \right) + \sum_{\substack{t\in V\\ t\neq v}} \left(\frac{\Upsilon_{vt}}{\sigma_{vt}} - \frac{1}{2} \right) \\ &= \sum_{\substack{s\in V\\ s\neq v}} \left(\sum_{\substack{t\in V\\ t\neq v}} \frac{\Upsilon_{st}(v)}{\sigma_{st}} + \frac{\Upsilon_{sv}}{\sigma_{sv}} - \frac{1}{2} \right) + \sum_{\substack{t\in V\\ t\neq v}} \left(\frac{\Upsilon_{vt}}{\sigma_{vt}} - \frac{1}{2} \right) \\ &= \sum_{\substack{s\in V\\ s\neq v}} \left(\sum_{\substack{t\in V\\ t\neq v}} \frac{\Upsilon_{st}(v)}{\sigma_{st}} + \frac{\Upsilon_{sv}}{\sigma_{sv}} - 1 \right) + \sum_{\substack{t\in V\\ t\neq v}} \frac{\Upsilon_{vt}}{\sigma_{vt}} \end{aligned} \tag{8.4}$$

We will try to calculate this value[3]. To do this, we define

$$d_{s,\bullet}(v) := \sum_{t\in V} \frac{\Upsilon_{st}(v)}{\sigma_{st}}.$$

We will try to compute $d_{s,\bullet}(v)$ using a recurrence equation similar to Brandes' equation (2.3.1). To this end, we first define an array T_{st}. For $i = 1, \ldots, |V|$, we define $T_{st}[i]$ as the number of shortest paths from s to t that contain exactly i vertices. On these arrays we define a number of operations, where we use that the array T_{st} uniquely determines a polynomial $W_{st} := \sum_{i=1}^{|V|} T_{st}[i]X^i \in \mathbb{Z}[X]$.

(i.) $T_{st}^{\rightarrow}$ and $T_{st}^{\leftarrow}$, shifting right and shifting left, increases resp. decreases the indices of all values in the array by one. This can be done in time $\mathcal{O}(|V|)$.

[3]Note that this equation is slightly different from equation (10) of the paper [SMR12]. The paper is about undirected graphs, but an adaptation is given for directed graphs (see the paper, at the end of Section 4.3). However, probably this adaptation does not take into account the possibility that a path *starts* at v. Then a path can also have a negative contribution. Therefore we derived a slightly different formula.

(ii.) $||T_{st}||$ computes $\sum_{i=1}^{|V|} T_{st}[i]/i$. The running time is $\mathcal{O}(|V|)$.

(iii.) $T_{sv} \oplus T_{su}$ adds the polynomials W_{sv} and W_{su}. This can be done in time $\mathcal{O}(|V|)$. We will write $\bigoplus$ for a sum of a series of polynomials.

(iv.) $T_{sv} \otimes T_{vt}$ multiplies the polynomials W_{su} and W_{vt}. With a fast multiplication algorithm (see [CLR01]) this can be done in $\mathcal{O}(|V| \log |V|)$.

(v.) $T_{sv} \oslash T_{vt}$ divides the polynomials W_{sv} and W_{vt}. This can be done in $\mathcal{O}(|V| \log |V|)$.

(vi.) $T_{sv} \div k$ divides every value in the array T_{sv} by the real value k. The running time is $\mathcal{O}(|V|)$.

We observe that[4]

$$T_{sv} = \bigoplus_{u \in P_s(v)} T_{su}^{\rightarrow}. \tag{8.5}$$

By construction it holds that $||T_{st}|| = \curlywedge_{st}$. Also, for $v \in V$ with $v \neq s \neq t$, we define $T_{st}(v)[i]$ to be the number of shortest paths from s to t that pass through v and that contain exactly i vertices ($i = 1, \ldots, |V|$). We note that

$$T_{st}(v) = (T_{sv} \otimes T_{vt})^{\leftarrow} = T_{sv} \otimes T_{vt}^{\leftarrow}.$$

This holds because every path from s to v can be extended by a path from v to t and this operation is in fact the multiplication of the polynomials W_{sv} and W_{vt}. Because the vertex v is counted twice, one needs to decrease all indices of the resulting array by one. By writing

$$D_{s,\bullet}(v) = \bigoplus_{t \in V} \frac{T_{st}(v)}{\sigma_{st}},$$

we are now able to derive an equation similar to Brandes' equation (2.3.1). The proof of this equation is not given in the paper (see [SMR12]), but we prove it here.

Lemma 8.1.2 (Analogon to Brandes' equation)**.** *It holds that*

$$D_{s,\bullet}(v) = \bigoplus_{w \,:\, v \in P_s(w)} \left(\frac{T_{sv}^{\rightarrow}}{\sigma_{sw}} \oplus T_{sv} \otimes (D_{s,\bullet}(w) \oslash T_{sw}^{\leftarrow}) \right), \tag{8.6}$$

similar to Brandes' equation (2.3.1)*.*

Proof. In the proof we denote by $T_{st}(e) = T_{st}[(v, w)]$ the array with as i-th entry the number of shortest paths from s to t that pass through edge $e = (v, w) \in E$ that contain exactly i vertices, with $i = 1, \ldots, |V|$. Then it holds that

$$D_{s,\bullet}(v) = \bigoplus_{t \in V} \frac{T_{st}(v)}{\sigma_{st}} = \bigoplus_{t \in V} \bigoplus_{w \,:\, v \in P_s(w)} \frac{T_{st}[(v, w)]}{\sigma_{st}} = \bigoplus_{w \,:\, v \in P_s(w)} \bigoplus_{t \in V} \frac{T_{st}[(v, w)]}{\sigma_{st}}. \tag{8.7}$$

Let $w \neq t$ be a vertex with $v \in P_s(w)$. It holds that $T_{st}[(v, w)] = T_{sv} \otimes T_{wt}$ and that

$$T_{st}(w) = (T_{sw} \otimes T_{wt})^{\leftarrow} = T_{wt} \otimes T_{sw}^{\leftarrow}.$$

Hence, the following equality holds:

$$T_{st}[(v, w)] = T_{sv} \otimes T_{wt} = T_{sv} \otimes (T_{st}(w) \oslash T_{sw}^{\leftarrow}),$$

[4]Recall that $P_s(v) = \{u \in V \,:\, (u, v) \in E \text{ is a tight edge with respect to } \delta(s, \cdot)\}$ is the *set of predecessors of* v along shortest paths starting at s, see (2.1).

for $w \neq t$ a vertex with $v \in P_s(w)$. We conclude that

$$\frac{T_{st}[(v,w)]}{\sigma_{st}} = \begin{cases} T_{sv} \otimes \left(\frac{T_{st}(w)}{\sigma_{st}} \oslash T_{sw}^{\leftarrow} \right) & \text{if } t \neq w \\ \frac{T_{sv}^{\rightarrow}}{\sigma_{sw}} & \text{if } t = w. \end{cases}$$

Inserting these values in equation (8.7), the equation for $D_{s,\bullet}(v)$, gives

$$\begin{aligned} D_{s,\bullet}(v) &= \bigoplus_{w\,:\,v \in P_s(w)} \bigoplus_{t \in V} \frac{T_{st}[(v,w)]}{\sigma_{st}} = \bigoplus_{w\,:\,v \in P_s(w)} \left(\frac{T_{sv}^{\rightarrow}}{\sigma_{sw}} \oplus \bigoplus_{t \in V \setminus \{w\}} \left(T_{sv} \otimes \left(\frac{T_{st}(w)}{\sigma_{st}} \oslash T_{sw}^{\leftarrow} \right) \right) \right) \\ &= \bigoplus_{w\,:\,v \in P_s(w)} \left(\frac{T_{sv}^{\rightarrow}}{\sigma_{sw}} \oplus T_{sv} \otimes \left(\left(\bigoplus_{t \in V \setminus \{w\}} \frac{T_{st}(w)}{\sigma_{st}} \right) \oslash T_{sw}^{\leftarrow} \right) \right) \\ &= \bigoplus_{w\,:\,v \in P_s(w)} \left(\frac{T_{sv}^{\rightarrow}}{\sigma_{sw}} \oplus T_{sv} \otimes (D_{s,\bullet}(w) \oslash T_{sw}^{\leftarrow}) \right), \end{aligned}$$

which is the desired equation. □

We use Brandes' equation by noting that

$$\begin{aligned} SV_B(v) &= \sum_{\substack{s \in V \\ s \neq v}} \left(\sum_{\substack{t \in V \\ t \neq v}} \frac{\Upsilon_{st}(v)}{\sigma_{st}} + \frac{\Upsilon_{sv}}{\sigma_{sv}} - 1 \right) + \sum_{\substack{t \in V \\ t \neq v}} \frac{\Upsilon_{vt}}{\sigma_{vt}} \\ &= \sum_{\substack{s \in V \\ s \neq v}} \left(||D_{s,\bullet}(v)|| + \frac{||T_{sv}||}{\sigma_{sv}} - 1 \right) + \sum_{\substack{t \in V \\ t \neq v}} \frac{||T_{vt}||}{\sigma_{vt}}. \end{aligned}$$

We are now able to give an algorithm for computing the Shapley betweenness centrality SH_B For each source $s \in V$, an array SH_s is computed. Summing $\bigoplus_{s \in V} SH_s$ gives an array with the Shapley-value betweenness centrality SH_B for each vertex $v \in V$.

Input: Graph $G = (V, E)$ with edge reliabilities $r : E \to (0, 1)$ or costs $c : E \to \mathbb{R}_{>0}$.
Output: For each $v \in V$, the value $SH_B(v)$.

Initialize: array $SH_B[\,]$ consisting of $|V|$ zeroes,
foreach $s \in V$ **do**
    Compute topologically sorted shortest path *DAG* G' rooted at s.
    Apply Algorithm 13 to compute $SH_s[\,]$.
    $SH_B := SH_B \oplus SH_s$.
end
return SH_B.

Algorithm 12: Algorithm to efficiently compute SH_B.

Theorem 8.1.3. *Algorithm* 12 *computes the Shapley-value based betweenness centrality in time bounded by* $\mathcal{O}(|V|^2|E|\log|V|)$*, while* $\mathcal{O}(|V|^2)$ *storage space is required.*

Proof. Observe that an execution of Algorithm 13 takes time $\mathcal{O}(|E| \cdot |V|\log|V|)$. Note that an execution of Dijkstra's algorithm can be done in time $\mathcal{O}(|V||E| + |V|\log|V|) \subseteq \mathcal{O}(|E| \cdot |V|\log|V|)$. It follows that the total running time of Algorithm 12 is bounded by $\mathcal{O}(|V| \cdot |E||V|\log|V|)$.The storage space is bounded by $\mathcal{O}(|V|^2)$. The largest objects that are stored are arrays of size $|V|^2$. □

Input: Top. sorted shortest/max. rel. path *DAG* $G = (V, E)$ w.r.t. source vertex s.
Output: For each $u \in V$, the value SH_s.

Initialize: $\sigma_s[s] = 1$, $\sigma_s[v] = 0$ for all $v \in V \setminus \{s\}$, array $SH_s[\,]$ consisting of $|V|$ zeroes, empty list $P[v]$ for all $v \in V$, array $D_s[\,][\,]$ consisting $|V| \times |V|$ zeroes, array $T_s[\,][\,]$ consisting of $|V| \times |V|$ zeroes, except for $T_s[s][1] := 1$.
foreach $x \in V$ in topological order **do**
  foreach child $y \in V$ of x **do**
    $\sigma_s[y] := \sigma_s[y] + \sigma_s[x]$
    append x to $P[y]$
    $T_s[y] := T_s[y] \oplus T_s^{\rightarrow}[x]$
  end
end
foreach $v \in V$ in reverse topological order **do**
  foreach $u \in V$ in $P[v]$ **do**
    $D_s[u] := D_s[u] \oplus \frac{T_s^{\rightarrow}[u]}{\sigma_s[v]} \oplus T_s[u] \otimes (D_s[v] \oslash T_s^{\leftarrow}[v])$
  end
  if $v \neq s$ **do**
    $SH_s[v] := ||D_s[v]|| + \frac{||T_s[v]||}{\sigma_s[v]} - 1$
    $SH_s[s] := SH_s[s] + \frac{||T_s[v]||}{\sigma_s[v]}$ //Small adaptation to the algorithm in [SMR12].
  end
end
return SH_s

Algorithm 13: Algorithm to efficiently compute SH_s. Executing this algorithm for all $s \in V$ and summing $\bigoplus_{s \in V} SH_s$ gives an array with the Shapley-value based betweenness centrality SH_B.

Example 8.1.2. We return to Example 8.1.1, the example from the beginning of this section. With the above algorithm, we are now able to compute the Shapley-value based betweenness centrality for each vertex in the example graph G.

Vertex	Betweenness	Shapley
v_9	98	18.2
v_{10}	98	16.08333
v_5	81	10.33333
v_4	80	10.33333
v_{11}	62	9.25
v_1, v_2, v_3	0	-4.47857
v_{12}, v_{13}, v_{14}, v_{15}	0	-4.47857
v_6, v_7, v_8	0	-4.55
v_{16}, v_{17}, v_{18}, v_{19}	0	-4.8

Remark 8.1.1. With some minor modifications, one is able to adapt the algorithm for computing the Shapley-value based *weighted* betweenness centrality. First we note that equation (8.4)

becomes

$$SH_{B^W}(v) = \sum_{\substack{s,t\in V\\ s\neq v\neq t}} \frac{w_{s,t}\cdot \Upsilon_{st}(v)}{\sigma_{st}} + \sum_{\substack{s\in V\\ s\neq v}} \left(\frac{w_{s,v}\cdot \Upsilon_{sv}}{\sigma_{sv}} - \frac{w_{s,v}}{2}\right) + \sum_{\substack{t\in V\\ t\neq v}} \left(\frac{w_{v,t}\cdot \Upsilon_{vt}}{\sigma_{vt}} - \frac{w_{v,t}}{2}\right)$$

$$= \sum_{\substack{s\in V\\ s\neq v}} \left(\sum_{\substack{t\in V\\ t\neq v}} \frac{w_{s,t}\cdot \Upsilon_{st}(v)}{\sigma_{st}} + \frac{w_{s,v}\cdot \Upsilon_{sv}}{\sigma_{sv}} - \frac{w_{s,v}}{2} \right) + \sum_{\substack{t\in V\\ t\neq v}} \left(\frac{w_{v,t}\cdot \Upsilon_{vt}}{\sigma_{vt}} - \frac{w_{v,t}}{2}\right). \quad (8.8)$$

For the weighted Brandes' recurrence relation, we get (by just putting weights in the proof of Lemma 8.6):

$$D^W_{s,\bullet}(v) = \bigoplus_{t\in V} \frac{w_{s,t}\cdot T_{st}(v)}{\sigma_{st}} = \bigoplus_{w\,:\,v\in P_s(w)} \left(\frac{w_{s,w}\cdot T^{\rightarrow}_{sv}}{\sigma_{sw}} \oplus T_{sv} \otimes (D^W_{s,\bullet}(w) \oslash T^{\leftarrow}_{sw}) \right).$$

Hence, for computing the Shapley-value based *weighted* betweenness centrality, the last loop in Algorithm 13 needs to be adjusted as follows.

```
...
foreach v ∈ V in reverse topological order do
    foreach u ∈ V in P[v] do
        D_s^W[u] := D_s^W[u] ⊕ (w_{s,v}·T_s^→[u] / σ_s[v]) ⊕ T_s[u] ⊗ (D_s^W[v] ⊘ T_s^←[v])
    end
    if v ≠ s do
        SH_s^W[v] := ||D_s^W[v]|| + (w_{s,v}||T_s[v]||| / σ_s[v]) − w_{s,v}/2
        SH_s^W[s] := SH_s^W[s] + (w_{s,v}·||T_s[v]||| / σ_s[v]) − w_{s,v}/2   //Adaptation to [SMR12].
    end
end
return SH_s^W
```

Algorithm 14: Adaptations to Algorithm 13 in order to compute the Shapley-value based *weighted* betweenness centrality. The rest of Algorithms 12 and 13 remains the same, except that we have SH_{B^W} instead of SH_B, SH^W_s instead of SH_s and D^W_s instead of D_s.

With the above adaptations we compute the Shapley-value based *weighted* betweenness centrality in the same time as the Shapley-value based betweenness centrality (see Theorem 8.1.3).

In the next section we will apply these algorithms to the CPB-network of 108 vertices to compute the Shapley-value based (weighted) betweenness centrality in this network.

8.2 Results

We compute the Shapley-value based (weigthed and unweighted) betweenness centrality for all countries in the CPB-network. First we note that the CPB-network consists not just of one graph but of 108 different graphs, one for each 'source' country (see Section 2.4). We already observed that the definition of (weighted) betweenness still makes sense in the CPB-network (see Section 2.4.3). Analogously, this observation also holds for the *group* betweenness centrality and hence for the *Shapley-value based* betweenness centrality. Since the algorithm of the previous section seperately computes the contributions made by each source vertex to the Shapley-value based (weighted) betweenness centrality, we can use that algorithm for computing the Shapley-value based (weighted) betweenness in our network of countries without modifications.

Before computing the Shapley-value based betweenness centrality, it might be interesting to compute

$$T := \bigoplus_{\substack{s,t \in V: \\ s \neq t}} T_{st},$$

which is the total number of shortest paths in the CPB-network stored with their number of vertices (i.e. $T[i]$ is the is the number of shortest paths that contain exactly i vertices) in an array, and which we can efficiently compute using the algorithm from the previous section. We find that

$$\begin{aligned} T[1] &= 0, \\ T[2] &= 3844, \\ T[3] &= 52333, \\ T[4] &= 8280, \\ T[5] &= 263, \\ T[i] &= 0 \quad \text{for } 5 < i \leq 108. \end{aligned}$$

The total number of shortest paths in the CPB-network is 64720. Of this number, $T[2] = 3844$ paths contain 2 vertices: those are the direct paths. There are $T[3] = 52333$ paths that consist of 3 vertices and hence have exactly *one* intermediary conduit country. Note that there are *no* shortest paths that contain more than 5 vertices, i.e. that travel via more than 3 conduit countries, where we assume in all computations that a small penalty is added (see Section 2.4.1), so that we only look at maximum reliability paths of minimum *length*.

We continue with computing the Shapley-value based *weighted* betweenness centrality.

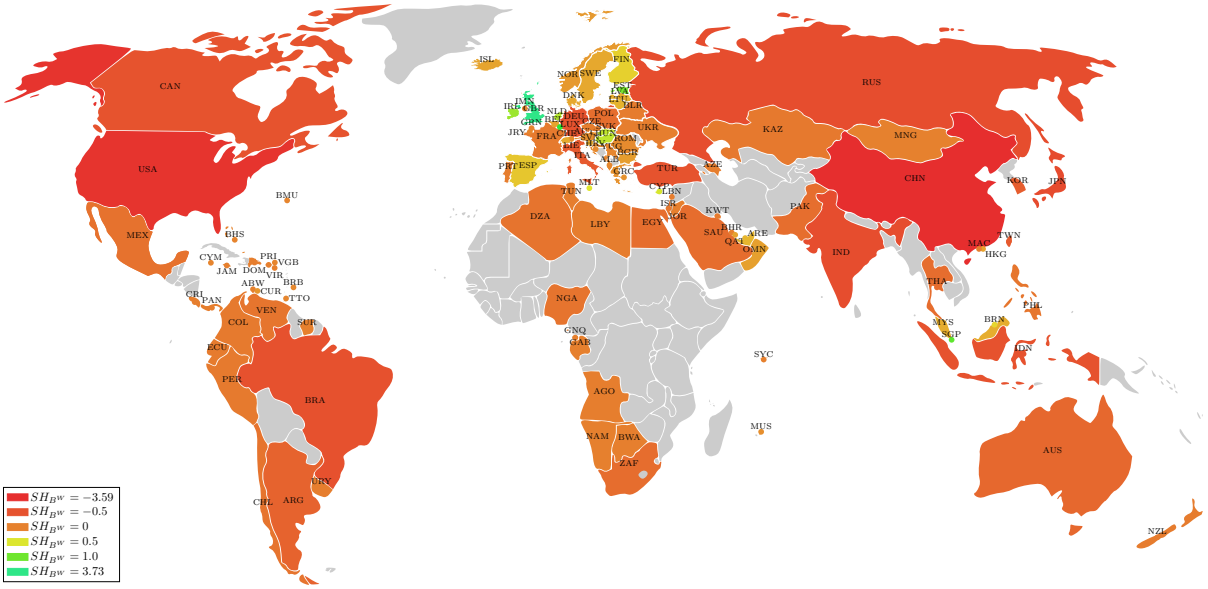


Figure 8.2: The countries sorted according to their *weighted* Shapley-value based betweenness centrality SH_{Bw}.

Position	Country	$B^W(u)$
1	GBR	12.80779
2	LUX	6.96023
3	SGP	4.23889
4	EST	2.92012
5	NLD	2.63225
6	IRL	2.57268
7	HUN	2.11862
8	ESP	2.03112
9	SVK	2.00355
10	CYP	1.67547
11	MLT	1.48476
12	FRA	1.31831
13	FIN	1.22550
14	BRN	1.18916
15	MYS	1.04301

Position	Country u	$SH_{B^W}(u)$
1	GBR	3.72715
2	LUX	2.29288
3	SGP	1.29130
4	EST	0.95515
5	IRL	0.79181
6	NLD	0.69373
7	HUN	0.63404
8	SVK	0.61776
9	CYP	0.52248
10	MLT	0.48568
11	BRN	0.38671
12	FIN	0.36042
13	ESP	0.31411
14	ARE	0.24205
15	LTU	0.23502

The ranking of the countries according to their Shapley-value based *weighted* betweenness centralities is similar to the ranking according to weighted betweenness centrality, as can be seen in the above table. Next we compare the *unweighted* betweenness centrality with the Shapley-value based unweighted betweenness centrality. Just as we did in Section 2.4.3, we multiply the Shapley-value based *unweighted* betweenness centrality by the constant $c := 1/(108 \cdot 1.07)$ to be able to compare it with the *weighted* Shapley-value based betweenness centrality.

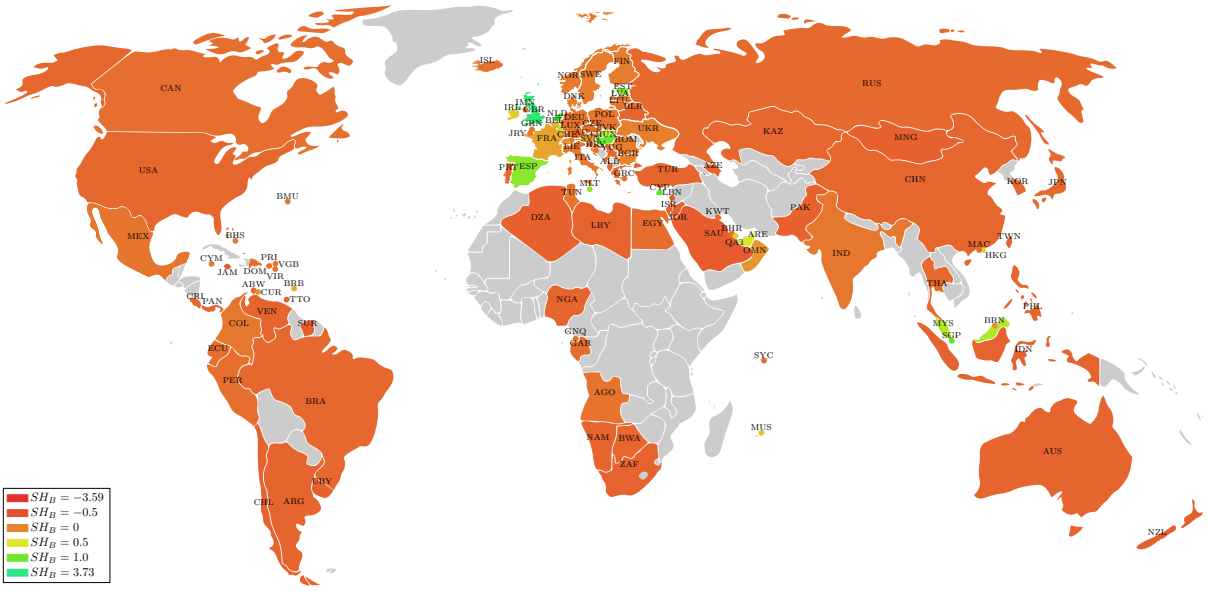


Figure 8.3: The countries sorted according to their *unweighted* Shapley-value based betweenness centrality SH_B.

Position	Country u	$B(u)$
1	GBR	10.61932
2	NLD	6.27001
3	SGP	4.80919
4	CYP	4.60625
5	HUN	3.78398
6	ESP	3.51100
7	EST	3.38580
8	MLT	2.97467
9	LUX	2.70852
10	MYS	2.52015
11	SVK	2.50322
12	QAT	2.29019
13	ARE	2.11230
14	BRB	1.79871
15	HKG	1.71729

Position	Country u	$SH_B(u)$
1	GBR	3.28099
2	NLD	1.81936
3	SGP	1.39736
4	CYP	1.34307
5	HUN	1.08319
6	EST	0.96612
7	ESP	0.87226
8	MLT	0.85857
9	LUX	0.70726
10	MYS	0.69411
11	SVK	0.64792
12	ARE	0.50794
13	HKG	0.35632
14	QAT	0.35157
15	IRL	0.32686

Here the similarities are even more clear: the top-5 of countries sorted according to both unweighted centrality measures is exactly the same. What is the most notable difference between the results for the Shapley-value based betweenness centrality and the 'usual' betweenness centrality? For this we look to the last 10 countries sorted according to their Shapley-value weighted (resp. unweighted) betweenness centrality.

Pos.	Country (u)	$SH_{B^W}(u)$	$B^W(u)$
98	ITA	-0.38507	(0.35983)
99	TUR	-0.40353	(0.09493)
100	CAN	-0.41714	(0.04265)
101	IDN	-0.49466	(0.01599)
102	DEU	-0.52411	(0.49025)
103	BRA	-0.58740	(0.00537)
104	RUS	-0.88385	(0.07895)
105	IND	-0.90967	(0.11221)
106	JPN	-1.12314	(0.13997)
107	USA	-3.07953	(0.00000)
108	CHN	-3.58992	(0.02164)

Pos.	Country u	$SH_B(u)$	$B(u)$
98	ISR	-0.29193	(0.00686)
99	ZAF	-0.29705	(0.01575)
100	MNG	-0.30084	(0.00882)
101	URY	-0.30114	(0.00000)
102	PAK	-0.30594	(0.00074)
103	DZA	-0.30749	(0.00000)
104	NZL	-0.31621	(0.00456)
105	SUR	-0.32306	(0.00000)
106	SAU	-0.34983	(0.07512)
107	JAM	-0.38724	(0.00865)
108	LBN	-0.38984	(0.00000)

The list of the 10 countries with the lowest Shapley-value based *weighted* betweenness centrality contains (almost exclusively) very large economies, such as the United States, China, Japan and Germany. This is not a surprise, since in computing the Shapley-value based *weighted* betweenness centrality of a country, a path starting or ending at that country can have a *negative* contribution. If this country has a large economy, then the weight (see Section 2.4.2) of this path will be large and therefore the negative contribution of this path will be large. Large economies are more likely to be a source or a destination country and not a conduit country. The Shapley-value based weighted betweenness centrality reflects this.

Conclusions

In this section we shortly restate the most important *experimental* conclusions.

(1.) The Netherlands ranks quite high, they are 5th in the ranking according to weighted betweenness centrality. This seems to give some evidence for the news headers that the Netherlands is an attractive tax country for companies.

(2.) Great Britain (GBR) is the most central country in the network. Great Britain has a substantially higher (weighted) betweenness centrality, as well as a substantially higher Shapley-value based betweenness centrality, than any other country in the network.

(3.) There is one big group of 64 countries, with the following property: for *any* country in this group, companies can send dividends to *any* other country *without* paying taxes.

(4.) If we contract this group (adding four countries that also have zero-tax routes to many countries) into one big 'super node', this node gets a *very high* betweenness centrality: this indicates that countries not in the group *very often* use countries in the group as conduit countries to send money through.

(5.) There always is a 'most profitable' route for companies passing through *at most* 3 conduit countries.

(6.) If the Netherlands wants to improve its role as a conduit country (measured with weighted betweenness centrality), it is a good idea to set the outgoing tax rate to India to zero, and after that to set the the tax rates to China and Brazil to zero.

(7.) If the Netherlands sets it outgoing dividend tax rates to India and China to 0%, the amount of money that companies send through The Netherlands is more than 50% of the amount that can be achieved *in total* (by setting the outgoing dividend tax rates to *every* country to 0%).

(8.) The Shapley-value based betweenness centrality applied on our network of countries gives similar results as the original betweenness centrality. However, this measure can differentiate more between vertices that have a low betweenness centrality.

The thesis also provided interesting mathematical insights. For a short summary of the mathematical contributions of this thesis, the reader is referred to the introduction. One question posed in this thesis remains open: is the *Tax Problem* when *all outgoing edge costs* of a node may be assigned a value, Problem 9 (*iii*), *NP*-hard? This is an interesting question for further research.

Appendix A

Data

This appendix contains the data in tables and pictures. In the first section some general data about the CPB-network is provided. The second section contains the experimental results for strictly maximum reliability paths. In the last section of this appendix, larger versions of some illustrations (world maps) used throughout the thesis are given.

A.1 The countries in the CPB-dataset

The first column contains the three-letter codes of the countries used throughout the thesis. The full names of the countries are given in the second column. The third column contains the *standard* Corporate Income Tax rate that is applicable in a country, the fourth column contains the *standard* tax on dividends leaving the country. When money is sent from one country to another country, the tax rates may be lower when this agreed in a bilateral tax treaty, or when a unilateral *tax-relief method* (to prevent double taxation) is used. For more information about the tax-rates and the construction of the tax-distances, see [RL14]. In this project the tax-distances provided by the CPB are considered as given facts. The fifth column contains the number of (bilateral or multilateral) tax-treaties that a country has signed with other countries. The last column contains the GDP (Gross Domestic Product) of a country, but normalized such that the sum of the GDPs of the countries in our dataset is 100.

Table A.1: The countries in the CPB-dataset.

Country	Full name	CIT	Div-tax	#treaties	GDP weight
ALB	Albania	10	10	26	0.0330
DZA	Algeria	25	15	23	0.3444
AGO	Angola	35	10	0	0.1619
ARG	Argentina	35	35	14	0.9378
ABW	Aruba	28	10	1	0.0031
AUS	Australia	30	30	40	1.2251
AUT	Austria	25	25	66	0.4531
AZE	Azerbaijan	20	10	29	0.1221
BHS	Bahamas	0	0	0	0.0140
BHR	Bahrain	46	0	10	0.0418
BRB	Barbados	25	15	23	0.0089
BLR	Belarus	18	12	44	0.1853
BEL	Belgium	33	25	70	0.5304
BMU	Bermuda	0	0	0	0.0070
BWA	Botswana	22	7.5	8	0.0398
BRA	Brazil	15	15	35	2.9725

Continued on the next page

Table A.1 The CPB-dataset – continued from the previous page

Country	Full name	CIT	Div-tax	#treaties	GDP weight
BRN	Brunei Darussalam	20	0	1	0.0274
BGR	Bulgaria	10	5	50	0.1310
CAN	Canada	15	25	75	1.8786
CYM	Cayman Islands	0	0	0	0.0028
CHL	Chile	20	35	24	0.4045
CHN	China	25	10	61	15.6574
COL	Colombia	25	0	4	0.6346
CRI	Costa Rica	30	15	1	0.0742
HRV	Croatia	20	12	44	0.0989
CUR	Curacao	27.5	0	0	0.0035
CYP	Cyprus	10	0	35	0.0298
CZE	Czech Republic	19	35	66	0.3622
DNK	Denmark	25	27	61	0.2652
DOM	Dominican Republic	29	10	1	0.1247
ECU	Ecuador	22	0	10	0.1935
EGY	Egypt	25	0	23	0.6814
GNQ	Equatorial Guinea	35	25	1	0.0243
EST	Estonia	21	0	36	0.0367
FIN	Finland	24.5	24.5	59	0.2492
FRA	France	33.3	30	80	2.8447
GAB	Gabon	35	15	4	0.0322
DEU	Germany	30.9	25	71	4.0354
GRC	Greece	26	10	42	0.3494
GRN	Guernsey	0	0	0	0.0034
HKG	HongKong	16.5	0	14	0.4662
HUN	Hungary	19	0	47	0.2468
ISL	Iceland	20	18	38	0.0162
IND	India	30	0	40	5.9116
IDN	Indonesia	25	20	52	1.5359
IRL	Ireland	12.5	20	53	0.2426
IMN	Isle of Man	0	0	0	0.0051
ISR	Israel	25	20	43	0.3139
ITA	Italy	27.5	20	69	2.3132
JAM	Jamaica	33.33	33.33	15	0.0318
JPN	Japan	25.5	20	47	5.8408
JRY	Jersey	0	0	0	0.0063
JOR	Jordan	30	0	13	0.0488
KAZ	Kazakhstan	20	15	35	0.2925
KOR	Korea Republic	22	20	67	2.0363
KWT	Kuwait	15	15	40	0.1905
LVA	Latvia	15	10	45	0.0470
LBN	Lebanon	15	10	13	0.0797
LBY	Libya	20	0	1	0.0976
LIE	Liechtenstein	12.5	0	3	0.0040
LTU	Lithuania	15	15	44	0.08200
LUX	Luxembourg	21	15	57	0.0533
MAC	Macao	12	0	0	0.0586
MYS	Malaysia	25	0	34	0.6292

Continued on the next page

Table A.1 The CPB-dataset – continued from the previous page

Country	Full name	CIT	Div-tax	#treaties	GDP weight
MLT	Malta	35	0	38	0.0142
MUS	Mauritius	15	0	15	0.0255
MEX	Mexico	30	0	36	2.2201
MNG	Mongolia	25	20	27	0.0192
NAM	Namibia	34	10	9	0.0211
NLD	Netherlands	25	15	74	0.8923
NZL	New Zealand	28	30	36	0.1666
NGA	Nigeria	30	10	11	0.5656
NOR	Norway	28	25	64	0.3498
OMN	Oman	12	0	8	0.1137
PAK	Pakistan	35	10	31	0.6505
PAN	Panama	25	17	14	0.0720
PER	Peru	30	4.1	3	0.4122
PHL	Philippines	30	15	29	0.5355
POL	Poland	19	19	64	1.0108
PRT	Portugal	25	25	53	0.3112
PRI	Puerto Rico	30	10	0	0.0805
QAT	Qatar	10	7	36	0.2372
ROM	Romania	16	16	66	0.3451
RUS	Russian Federation	20	15	59	3.1725
SAU	Saudi Arabia	50	5	18	1.1444
YUG	Serbia and Montenegro	15	20	42	0.0994
SYC	Seychelles	33	15	12	0.0029
SGP	Singapore	17	0	40	0.4121
SVK	Slovak Republic	23	0	42	0.1665
SVN	Slovenia	17	15	46	0.0731
ZAF	South Africa	28	15	55	0.7351
ESP	Spain	30	21	71	1.7805
SUR	Suriname	36	25	1	0.0085
SWE	Sweden	22	30	67	0.4959
CHE	Switzerland	24	35	71	0.4587
TWN	Taiwan Province	17	20	19	1.1402
THA	Thailand	20	10	34	0.8227
TTO	Trinidad and Tobago	25	10	16	0.0337
TUN	Tunisia	30	0	26	0.1330
TUR	Turkey	20	15	59	1.4180
UKR	Ukraine	19	15	56	0.4230
ARE	Untd Arab Emirates	0	0	21	0.3425
GBR	United Kingdom	24	0	51	2.9490
USA	United States	35	30	54	19.7921
URY	Uruguay	25	7	6	0.0679
VEN	Venezuela	34	34	28	0.5072
VIR	Virgin Islands U.S.	38.5	11	0	0.0020
VGB	Virgin Islands U.K.	0	0	0	0.0006

A.2 Data, strict paths

Table A.2: Strictly shortest paths, with a penalty of 10^{-20}. Total number of paths: 64720. Sorted according to betweenness. The number $\#P$ is the number of paths passing *through* a country (as a conduit country). The weighted (respectively unweighted) betweenness centrality is denoted by $B^W(u)$ (respectively $B(u)$). The Shapley-value based weighted betweenness centrality is denoted by $SH_B^W(u)$ and the Shapley-value unweighted betweenness centrality by $SH_B(u)$.

Country u	$\#P$	$B^W(u)$	$SH_{B^W}(u)$	$B(u)$	$SH_B(u)$
GBR	5621	12.80779	3.72715	10.61932	3.28099
LUX	2289	6.96023	2.29288	2.70852	0.70726
SGP	3316	4.23889	1.29130	4.80919	1.39736
EST	2720	2.92012	0.95515	3.38580	0.96612
NLD	3958	2.63225	0.69373	6.27001	1.81936
IRL	1744	2.57268	0.79181	1.57323	0.32686
HUN	3465	2.11862	0.63404	3.78398	1.08319
ESP	2679	2.03112	0.31411	3.51100	0.87226
SVK	2283	2.00355	0.61776	2.50322	0.64792
CYP	3786	1.67547	0.52248	4.60625	1.34307
MLT	3406	1.48476	0.48568	2.97467	0.85857
FRA	967	1.31831	-0.05901	1.14540	0.16051
FIN	588	1.22550	0.36042	0.45762	-0.03914
BRN	1292	1.18916	0.38671	0.75912	0.07469
MYS	2594	1.04301	0.20262	2.52015	0.69411
ARE	2006	1.01731	0.24205	2.11230	0.50794
CHE	1095	0.88451	0.19009	0.70979	0.07399
HKG	1853	0.87090	0.15707	1.71729	0.35632
SWE	550	0.84262	0.18058	0.49584	-0.03043
QAT	1758	0.78951	0.15847	2.29019	0.35157
LTU	456	0.76250	0.23502	0.20685	-0.11793
BEL	1213	0.71595	0.15541	1.22915	0.24021
DNK	539	0.68814	0.17303	0.46278	-0.04000
LVA	518	0.65222	0.20588	0.23798	-0.10631
SAU	58	0.62263	-0.19372	0.07512	-0.34983
SVN	333	0.61497	0.18751	0.13134	-0.14974
NOR	463	0.57897	0.11319	0.49520	-0.04951
OMN	1105	0.54205	0.14475	0.93497	0.09532
AUT	1205	0.51871	0.06629	0.75577	0.09021
ISL	286	0.49601	0.16051	0.11220	-0.15759
BGR	984	0.49183	0.12873	0.57719	0.00370
DEU	362	0.49025	-0.52411	0.21323	-0.12235
GRC	275	0.48274	0.07982	0.11444	-0.16558
ROM	837	0.41432	0.04734	0.53385	-0.01975
ITA	506	0.35983	-0.38507	0.48066	-0.07300
UKR	745	0.31899	-0.02421	0.94415	-0.01157
COL	424	0.30222	-0.06299	0.32581	-0.09085
CUR	1238	0.29067	0.09423	0.67772	0.06645
MUS	974	0.22089	0.06399	1.54819	0.29208
POL	470	0.21355	-0.18197	0.22187	-0.11161
CZE	304	0.20870	-0.02057	0.14359	-0.16197
HRV	186	0.19209	0.03638	0.26776	-0.17166

Continued on the next page

Table A.2 Strictly shortest paths – continued from the previous page

Country u	$\#P$	$B^W(u)$	$SH_{B^W}(u)$	$B(u)$	$SH_B(u)$
LIE	880	0.17617	0.05747	0.40682	-0.06662
YUG	37	0.15960	0.02628	0.01398	-0.23764
JPN	92	0.13997	-1.12314	0.09419	-0.23153
ALB	124	0.12166	0.03019	0.04468	-0.24510
BRB	456	0.11823	0.03201	1.79871	0.17923
IND	154	0.11221	-0.90967	0.17494	-0.08820
TTO	127	0.10507	0.02271	0.19404	-0.28885
PRT	99	0.10215	-0.03534	0.03447	-0.20138
TUR	38	0.09493	-0.40353	0.01100	-0.26748
AUS	19	0.09282	-0.23910	0.03682	-0.27480
KOR	29	0.09215	-0.34361	0.00865	-0.23365
RUS	53	0.07895	-0.88385	0.03411	-0.22954
ZAF	23	0.06798	-0.18426	0.01575	-0.29705
VGB	776	0.06412	0.02116	0.31777	-0.10551
CYM	776	0.06408	0.02048	0.31777	-0.10551
GRN	776	0.06407	0.02029	0.31777	-0.10551
IMN	776	0.06405	0.01977	0.31777	-0.10551
JRY	776	0.06403	0.01939	0.31777	-0.10551
BMU	776	0.06401	0.01918	0.31777	-0.10551
BHS	776	0.06390	0.01700	0.31777	-0.10551
KAZ	40	0.05557	-0.06807	0.00849	-0.25245
EGY	152	0.04814	-0.17714	0.10838	-0.19044
BLR	65	0.04591	-0.04304	0.02064	-0.26650
CAN	102	0.04265	-0.41714	0.14262	-0.18375
AZE	48	0.04174	-0.02429	0.01094	-0.26245
MNG	19	0.02614	0.00225	0.00882	-0.30084
ISR	24	0.02538	-0.09317	0.00686	-0.29193
CHN	24	0.02164	-3.58992	0.00731	-0.24924
NZL	3	0.02076	-0.04159	0.00456	-0.31621
IDN	18	0.01599	-0.49466	0.00334	-0.29094
ABW	84	0.01192	0.00295	0.02648	-0.26808
CHL	6	0.01062	-0.11955	0.00605	-0.29148
JAM	1	0.00832	-0.00968	0.00865	-0.38724
VEN	11	0.00696	-0.11324	0.00322	-0.27228
BRA	23	0.00537	-0.58740	0.02913	-0.23908
PER	4	0.00406	-0.06522	0.00657	-0.16799
THA	5	0.00084	-0.19017	0.00219	-0.21416
DOM	4	0.00072	-0.02038	0.00074	-0.16489
NAM	4	0.00072	-0.00482	0.00074	-0.26657
NGA	4	0.00072	-0.12177	0.00074	-0.26873
PAK	4	0.00072	-0.17867	0.00074	-0.30594
PHL	4	0.00072	-0.10608	0.00074	-0.25359
PRI	4	0.00072	-0.01307	0.00074	-0.16489
TWN	14	0.00068	-0.37092	0.00287	-0.29182
BHR	1	0.00000	-0.00975	0.00026	-0.16505
DZA	0	0.00000	-0.11676	0.00000	-0.30749
AGO	0	0.00000	-0.02461	0.00000	-0.12475
ARG	0	0.00000	-0.28383	0.00000	-0.27186

Continued on the next page

Table A.2 Strictly shortest paths – continued from the previous page

Country u	$\#P$	$B^W(u)$	$SH_{B^W}(u)$	$B(u)$	$SH_B(u)$
BWA	0	0.00000	-0.01250	0.00000	-0.25456
CRI	0	0.00000	-0.02409	0.00000	-0.23292
ECU	0	0.00000	-0.05188	0.00000	-0.16658
GNQ	0	0.00000	-0.00369	0.00000	-0.12331
GAB	0	0.00000	-0.00609	0.00000	-0.19139
JOR	0	0.00000	-0.01583	0.00000	-0.25384
KWT	0	0.00000	-0.05114	0.00000	-0.22139
LBN	0	0.00000	-0.03489	0.00000	-0.38984
LBY	0	0.00000	-0.03489	0.00000	-0.28138
MAC	0	0.00000	-0.01786	0.00000	-0.20624
MEX	0	0.00000	-0.14382	0.00000	-0.10096
PAN	0	0.00000	-0.02280	0.00000	-0.25312
SYC	0	0.00000	-0.00086	0.00000	-0.24879
SUR	0	0.00000	-0.00316	0.00000	-0.32306
TUN	0	0.00000	-0.02027	0.00000	-0.11538
USA	0	0.00000	-3.07953	0.00000	-0.24446
URY	0	0.00000	-0.02440	0.00000	-0.30114
VIR	0	0.00000	-0.00053	0.00000	-0.18100

A.3 Figures

The next few sections contain pictures from the text, but printed in landscape mode. Some countries are very small, and the reader will be better able to see them on the larger figures. The following pictures are included:

(A.3.1) An overview of the countries in the CPB-*dataset*.

(A.3.2) The countries depicted according to their *weighted* betweenness centrality.

(A.3.3) The countries depicted according to their *weighted* betweenness centrality *and* the 20 edges with the highest edge flow.

(A.3.4) The countries depicted according to their *unweighted* betweenness centrality.

(A.3.5) The countries depicted according to their *unweighted* betweenness centrality *and* the 20 edges with the highest edge betweenness centrality.

(A.3.6) The intersection $\cap_{s\in V}\mathrm{SCC}_0(s)$ of the strongly connected components of the subgraphs of zero–tax edges of the graphs G_s. Companies can send dividends from any green country to any other green country without paying taxes, possibly via a route through conduit countries.

(A.3.7) Within range paths: the experiment of Section 5.4.1, with $\alpha = 0.05$ and $j = 3$. Total number of paths: 155,724,338.

(A.3.8) Within additive range paths, with $\alpha = 0.005$ and $\varepsilon = 0.005$. This is the version of within range that the CPB used in their report [RL14]. Total number of paths: 2,324,679.

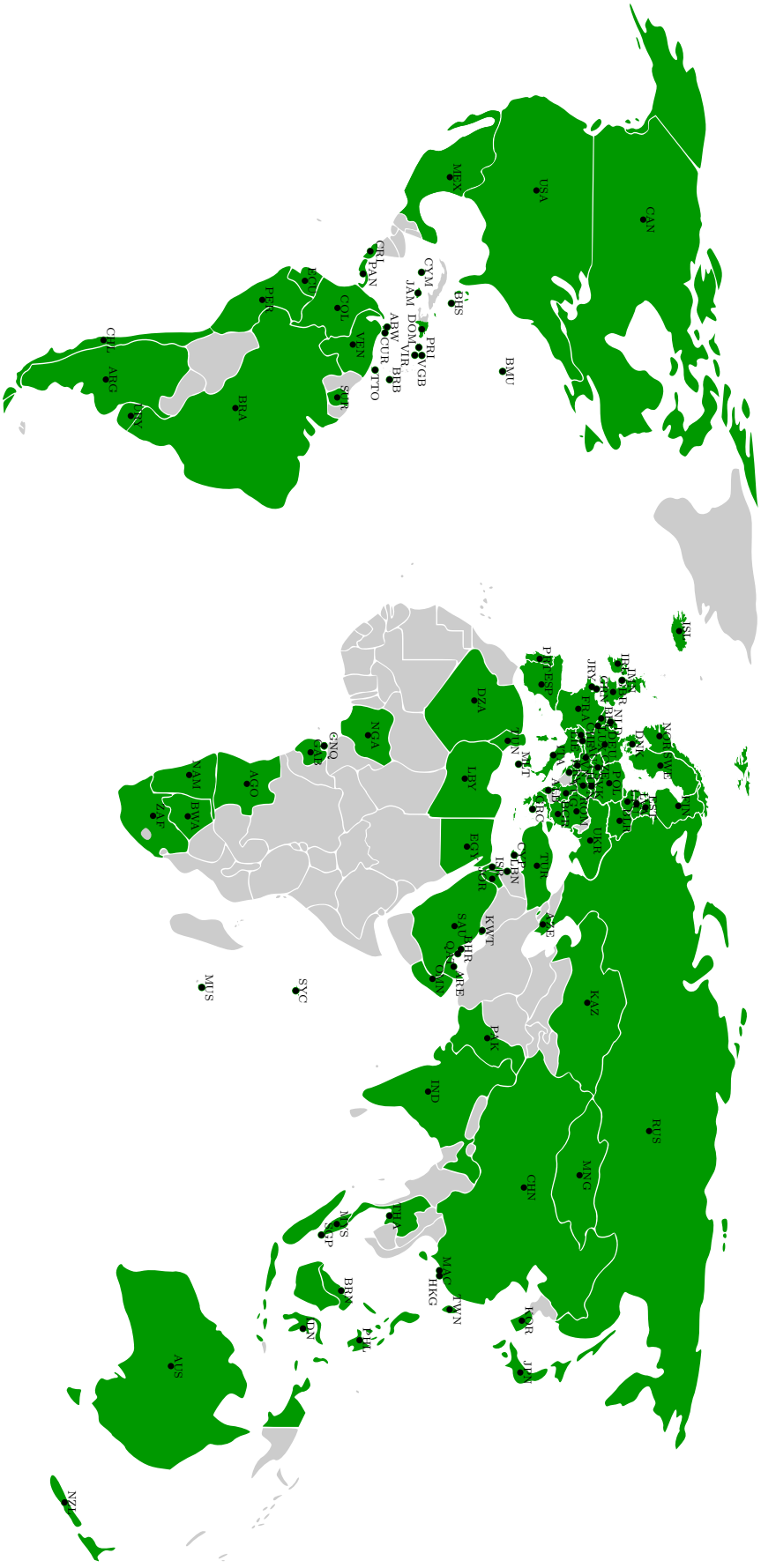
(A.3.9) Within multiplicative range paths, with $\alpha = 0.01$ and $\varepsilon = 0.0033333$. Total number of paths: 2,540,053,489.

(A.3.10) By setting the outgoing dividend tax rates to India and China to zero, the Netherlands achieves a weighted betweenness centrality that is more than 50% of the maximum weighted betweenness ever achievable for the Netherlands by decreasing outgoing tax rates. Green in the illustration: the other countries (except India and China) to which the Netherlands has a non-zero *outgoing* tax rate.

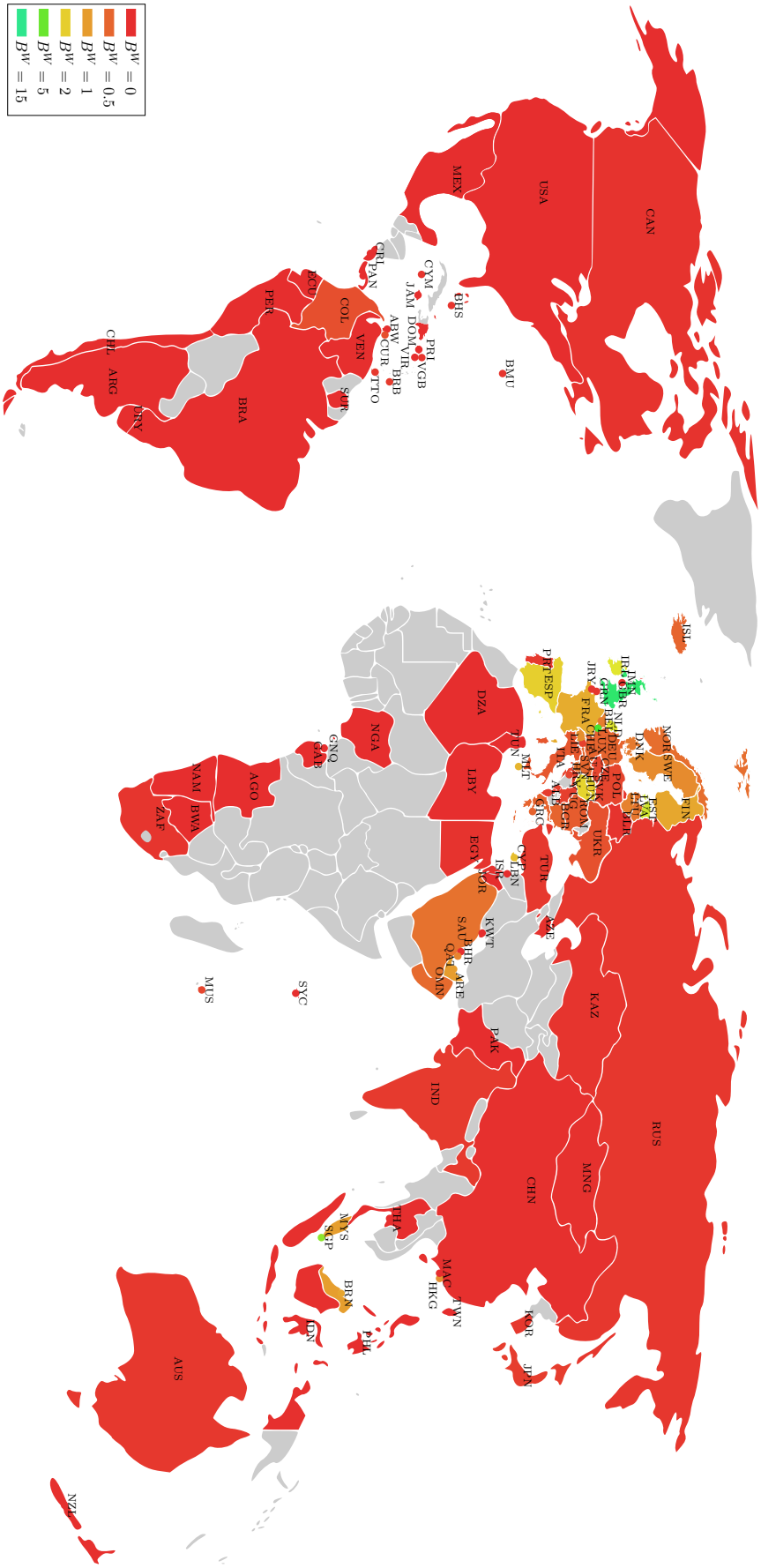
(A.3.11) The countries depicted according to their *Shapley*-value based *weighted* betweenness centrality.

(A.3.12) The countries depicted according to their *Shapley*-value based *unweighted* betweenness centrality.

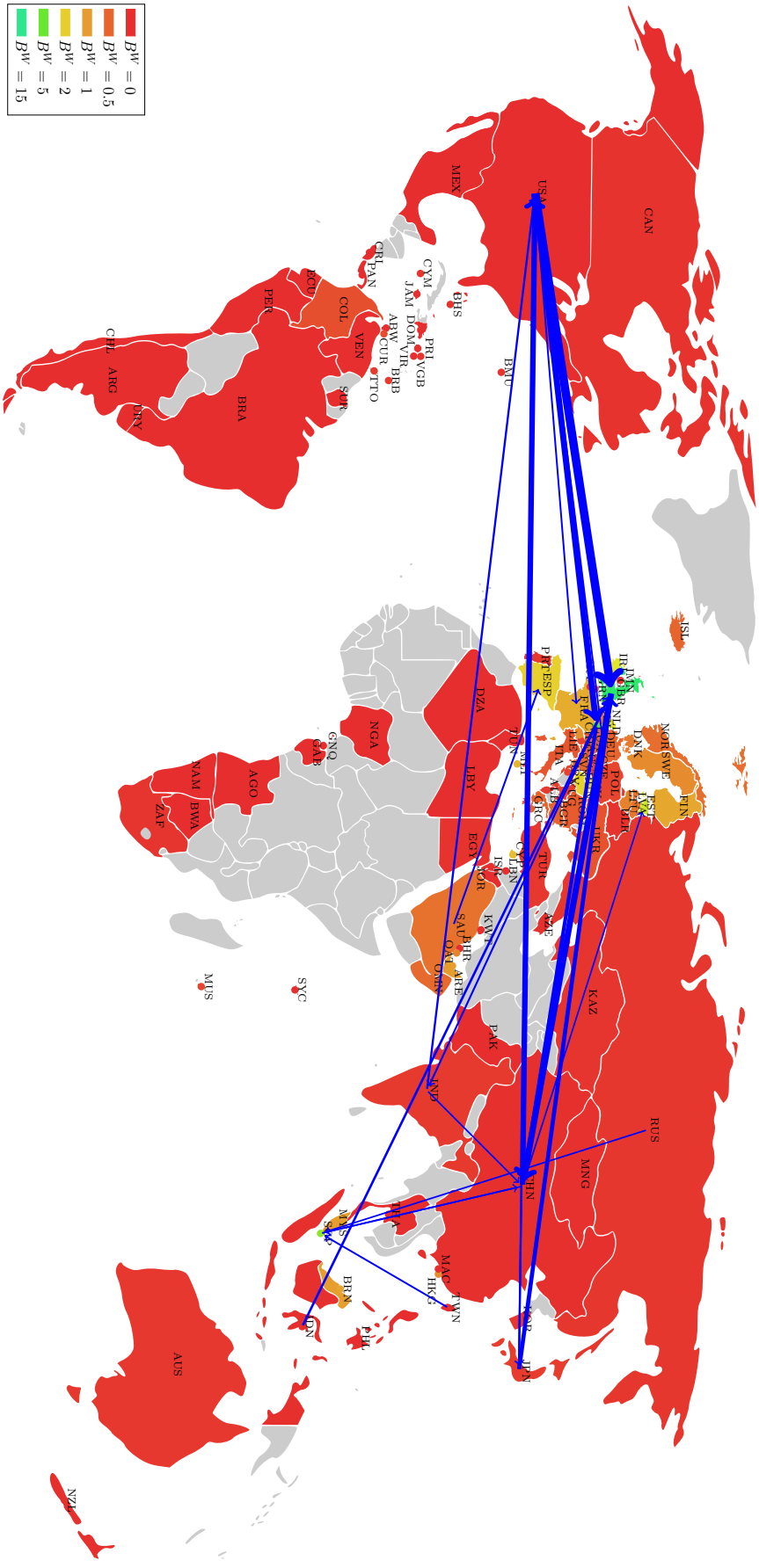
A.3.1 The CPB-dataset



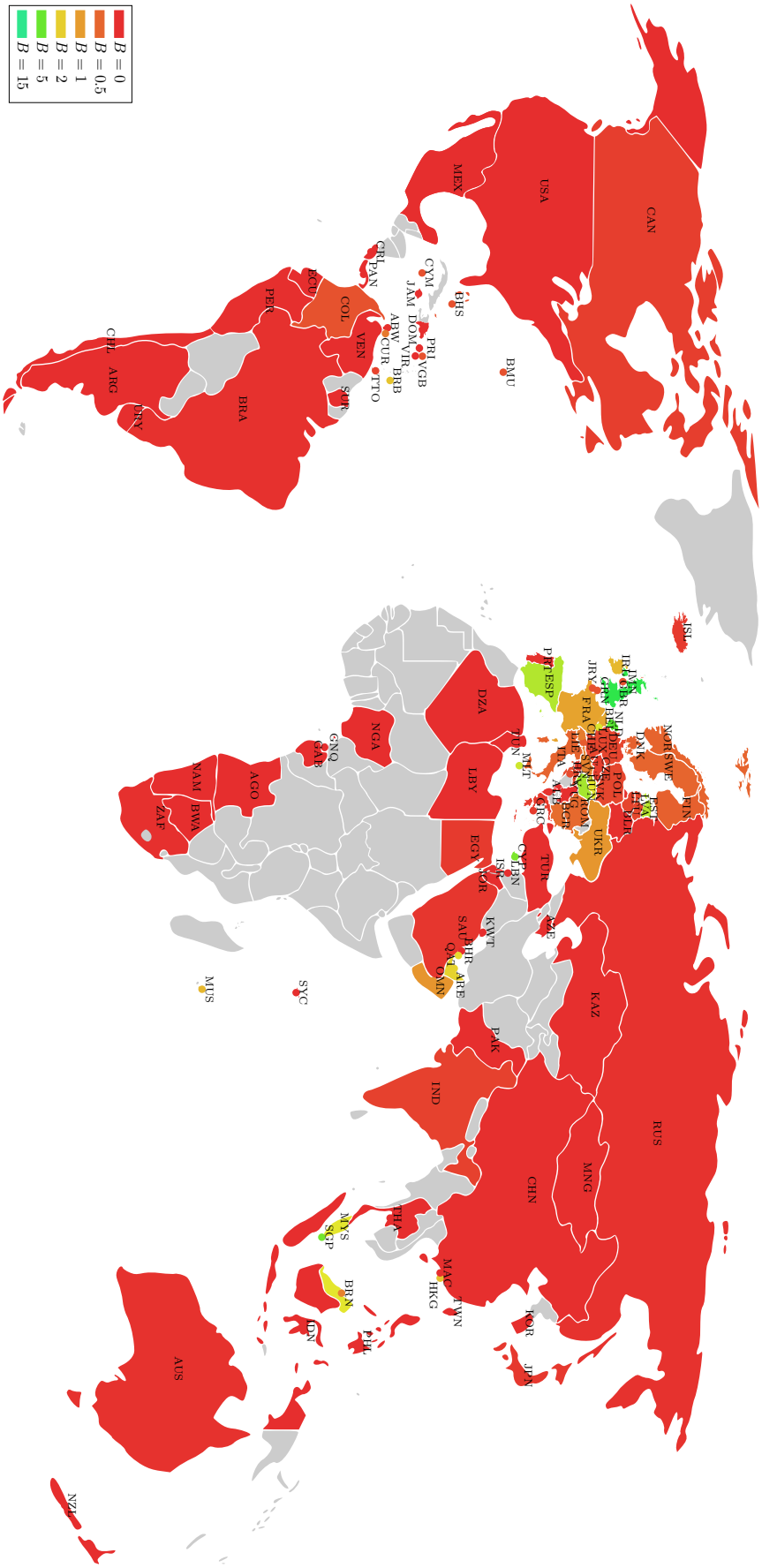
A.3.2 Strict paths, weighted betweenness centrality



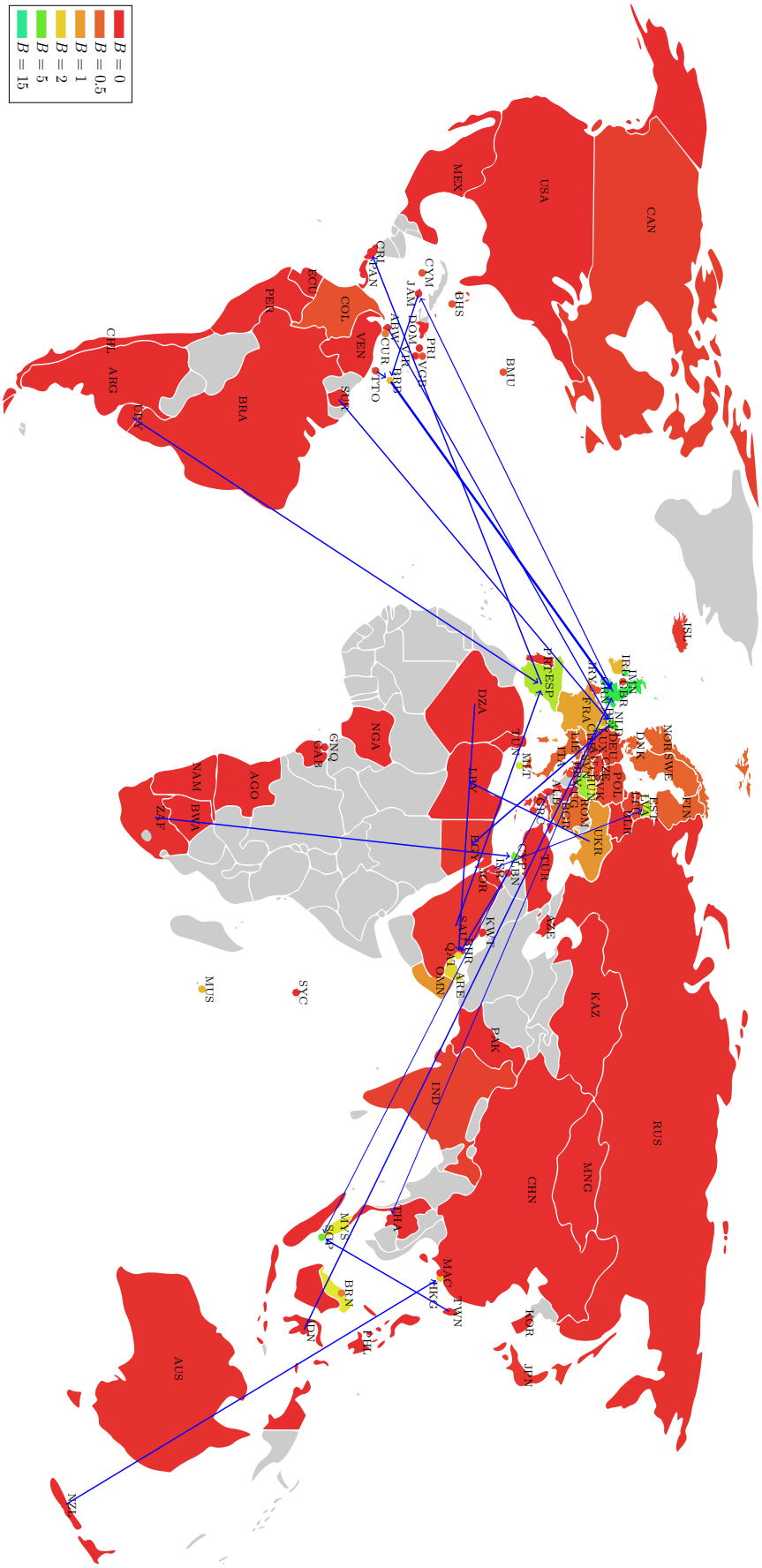
A.3.3 Strict paths, weighted betweenness centrality including edges



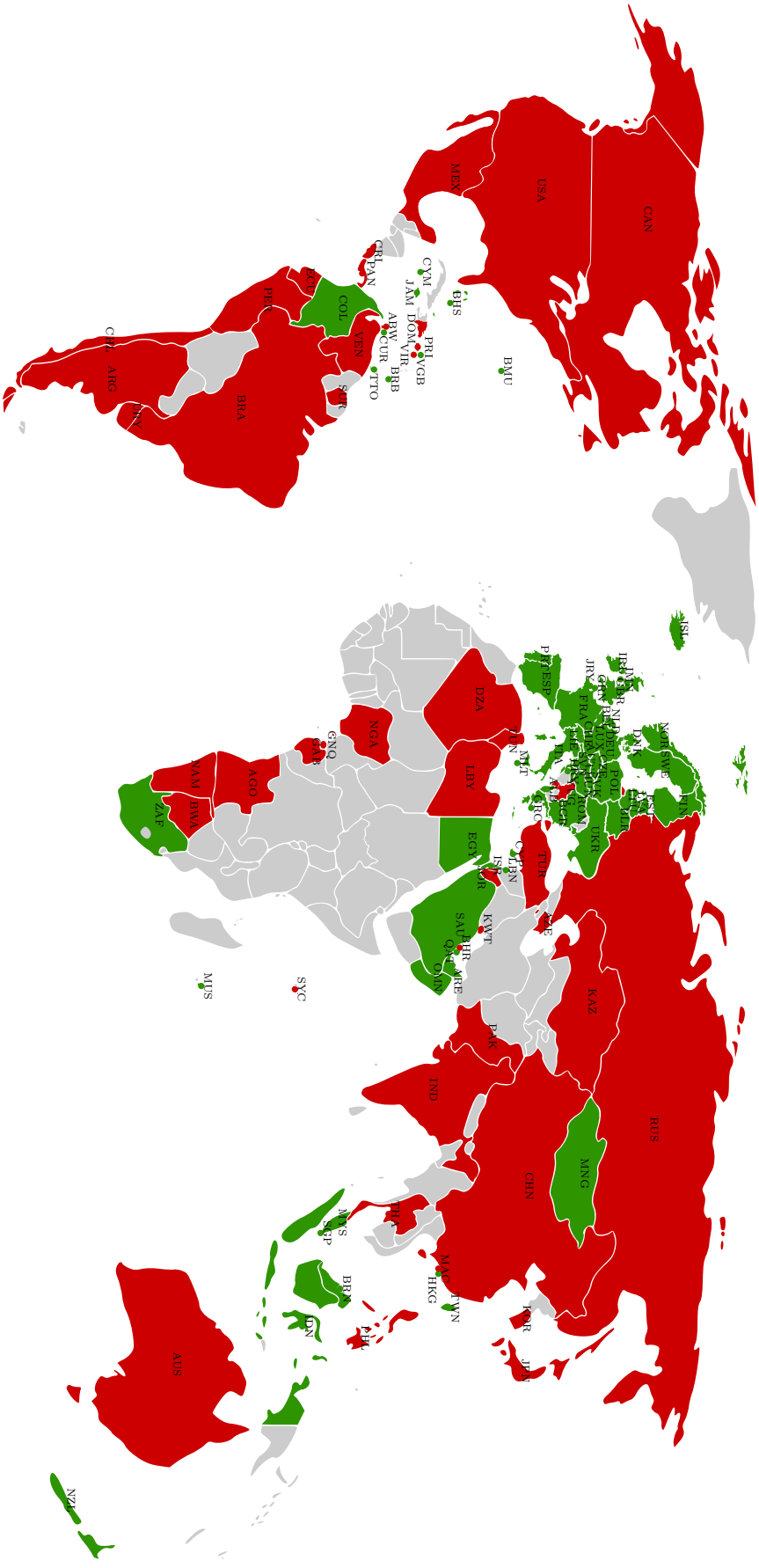
A.3.4 Strict paths, unweighted betweenness centrality



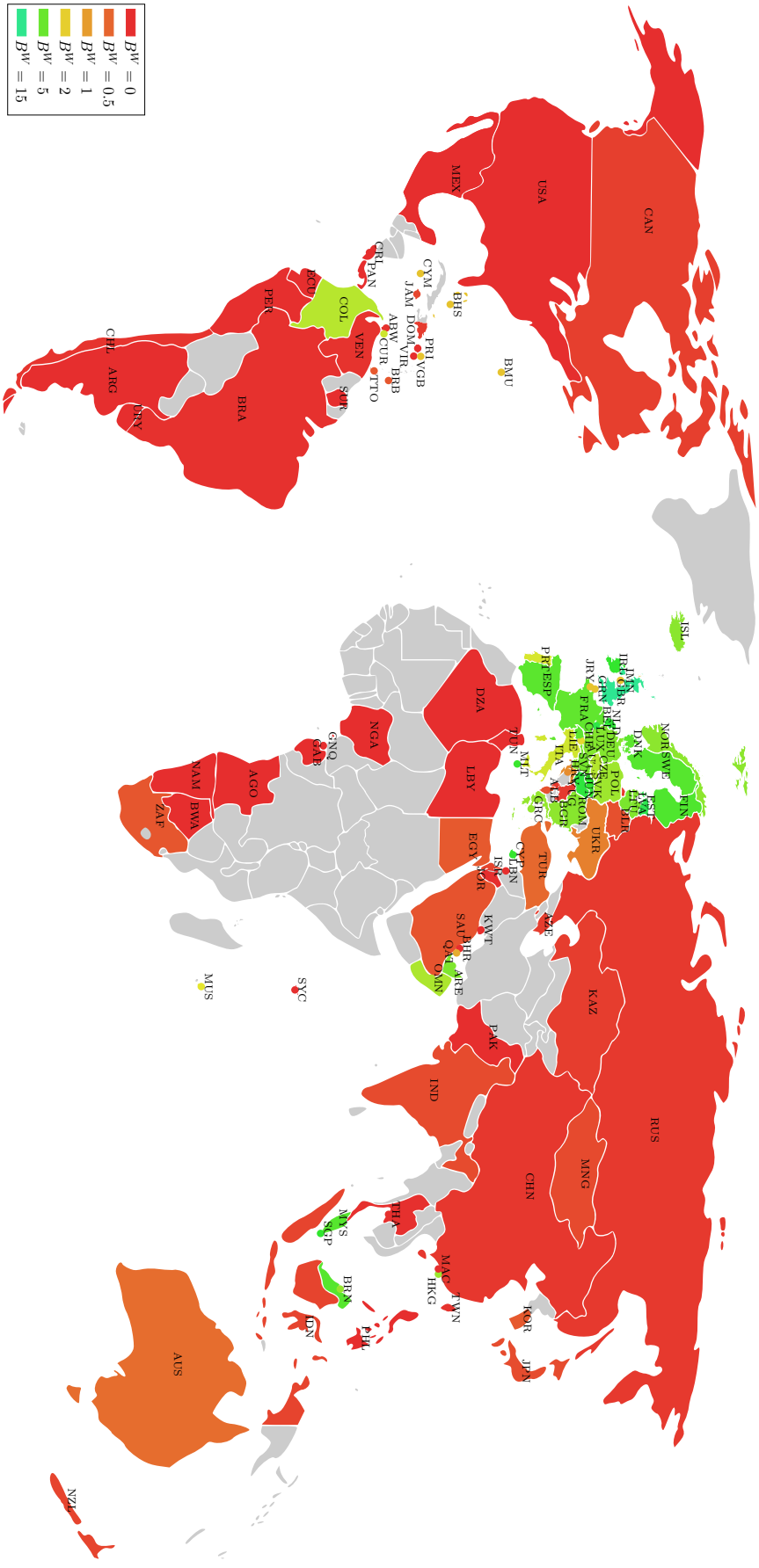
A.3.5 Strict paths, unweighted betweenness centrality including edges



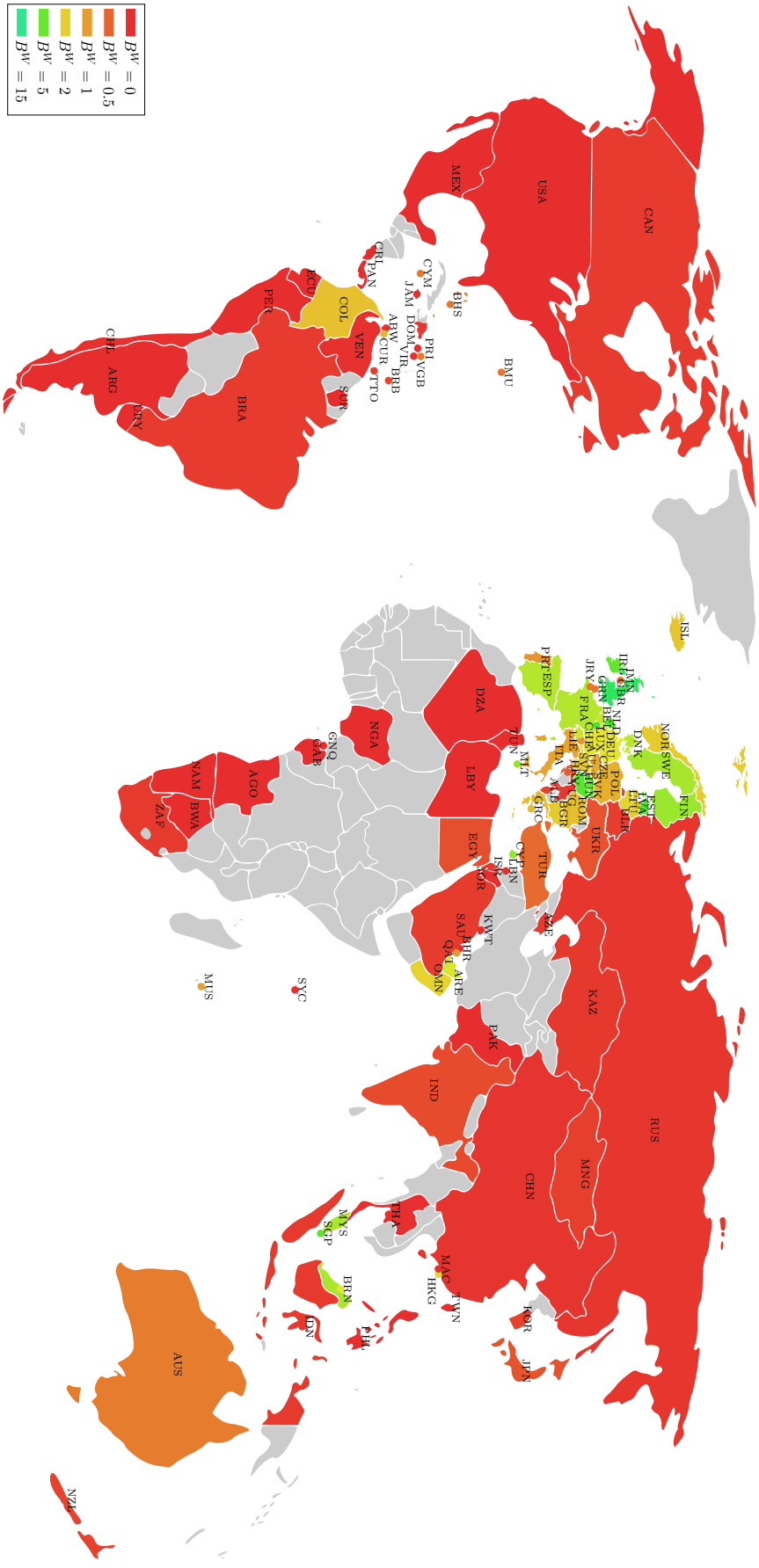
A.3.6 Intersection of the strongly connected components



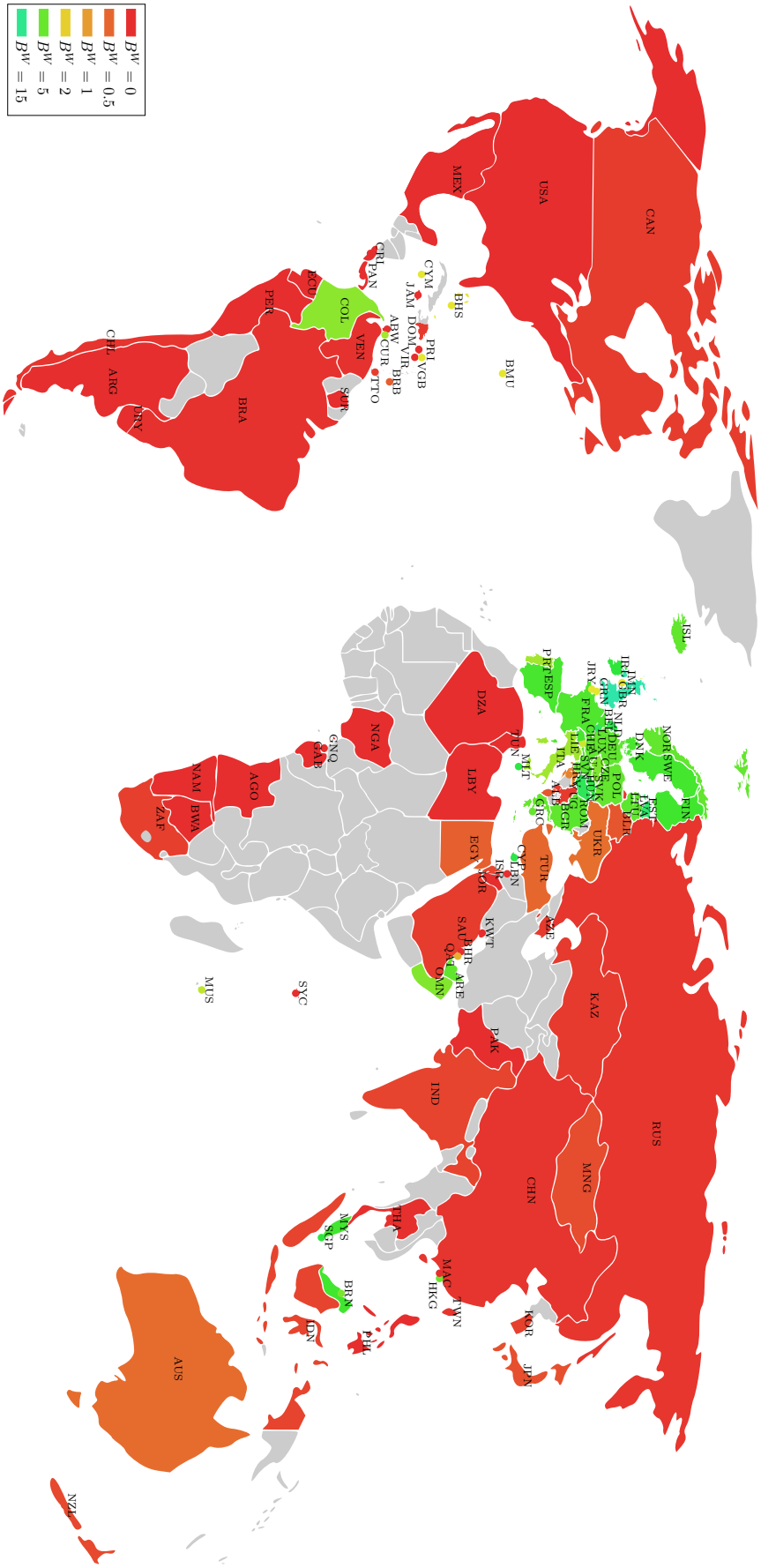
A.3.7 Within range paths: the experiment of Section 5.4.1



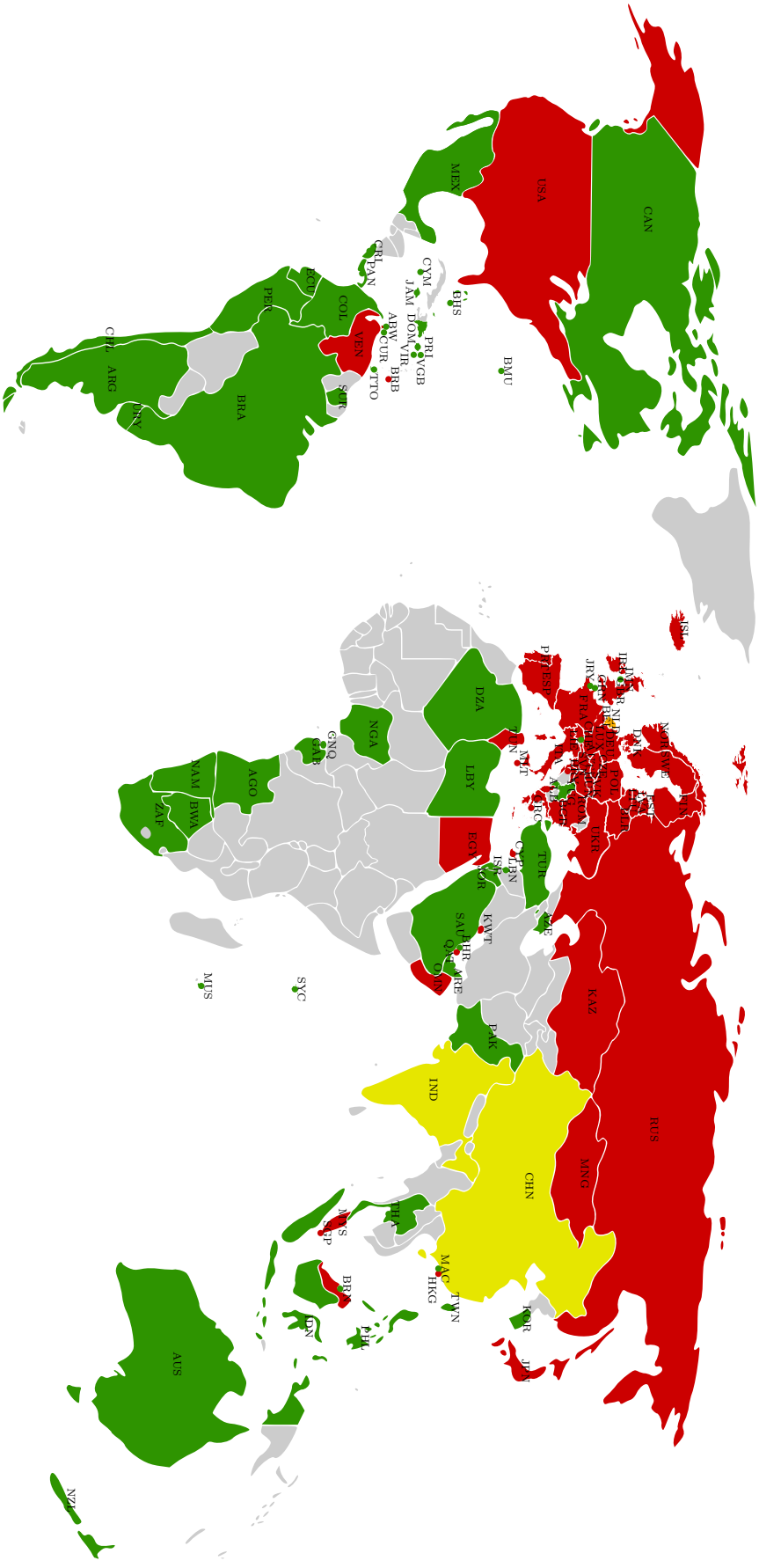
A.3.8 Within additive range paths, with $\alpha = 0.005$ and $\varepsilon = 0.005$



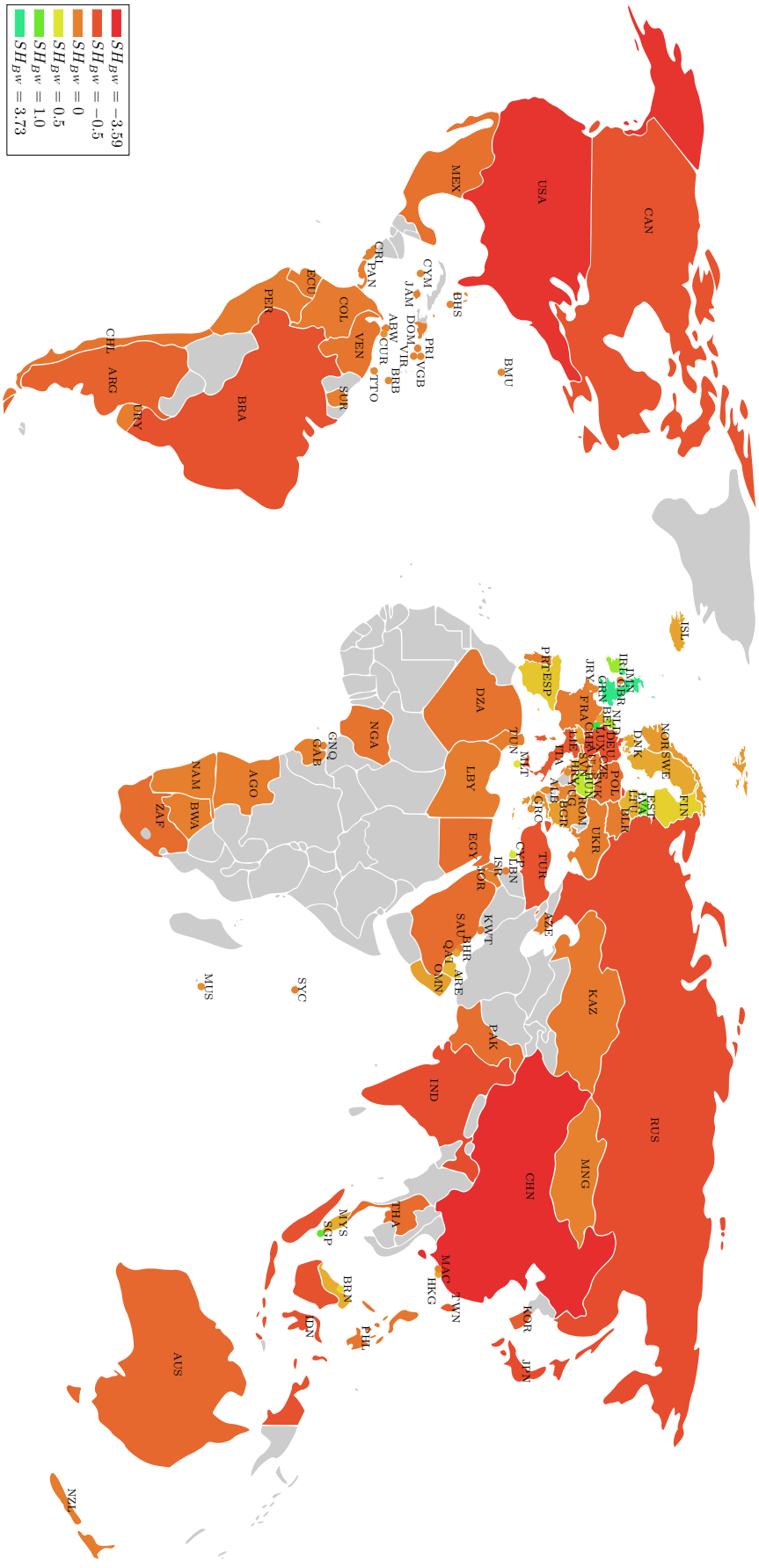
A.3.9 Within multiplicative range paths, with $\alpha = 0.01$ and $\varepsilon = 0.0033333$



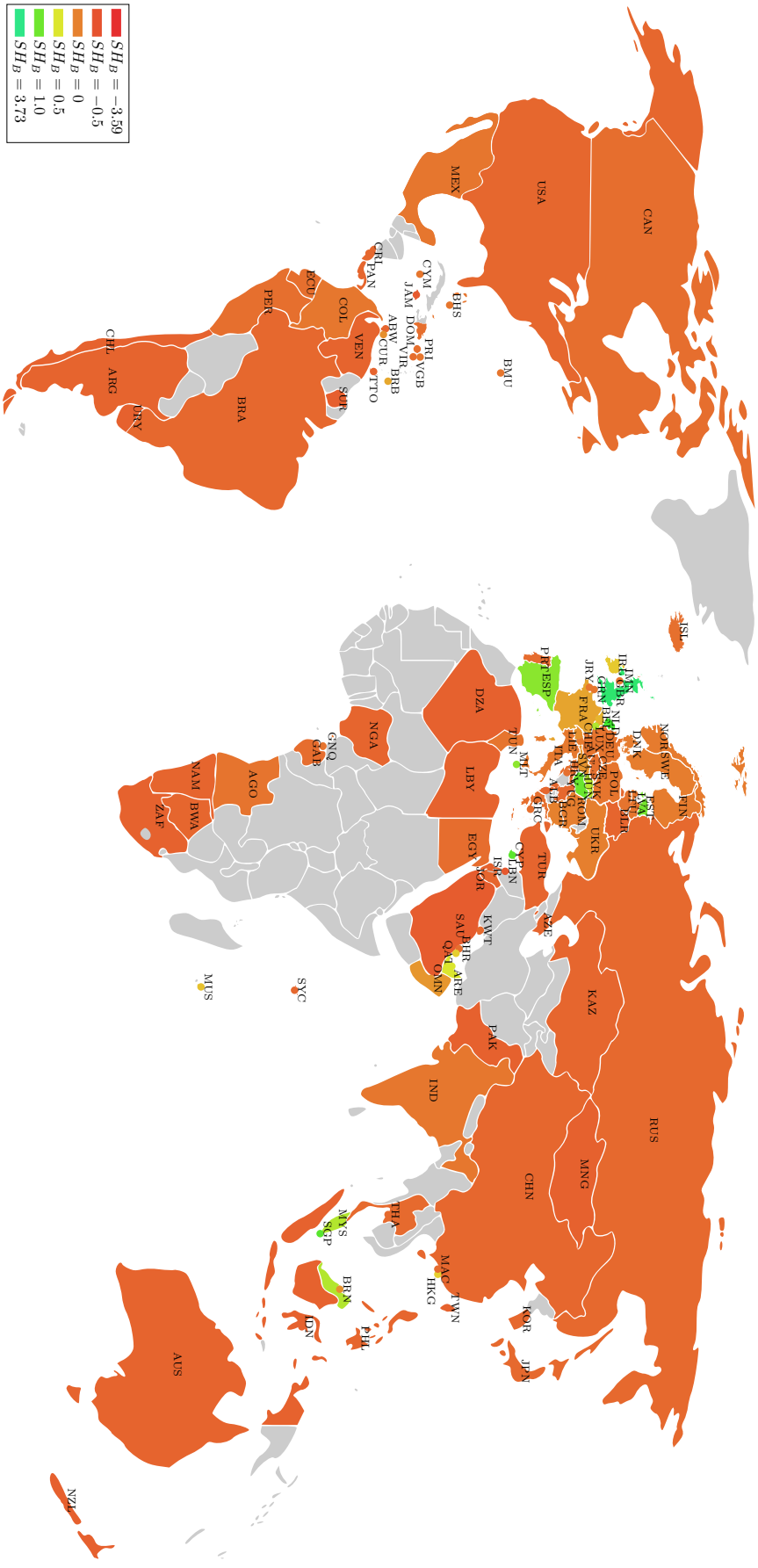
A.3.10 Maximizing betweenness: the experiment of Chapter 6



A.3.11 Strict paths, Shapley-value based weighted betweenness centrality



A.3.12 Strict paths, Shapley-value based unweighted betweenness centrality

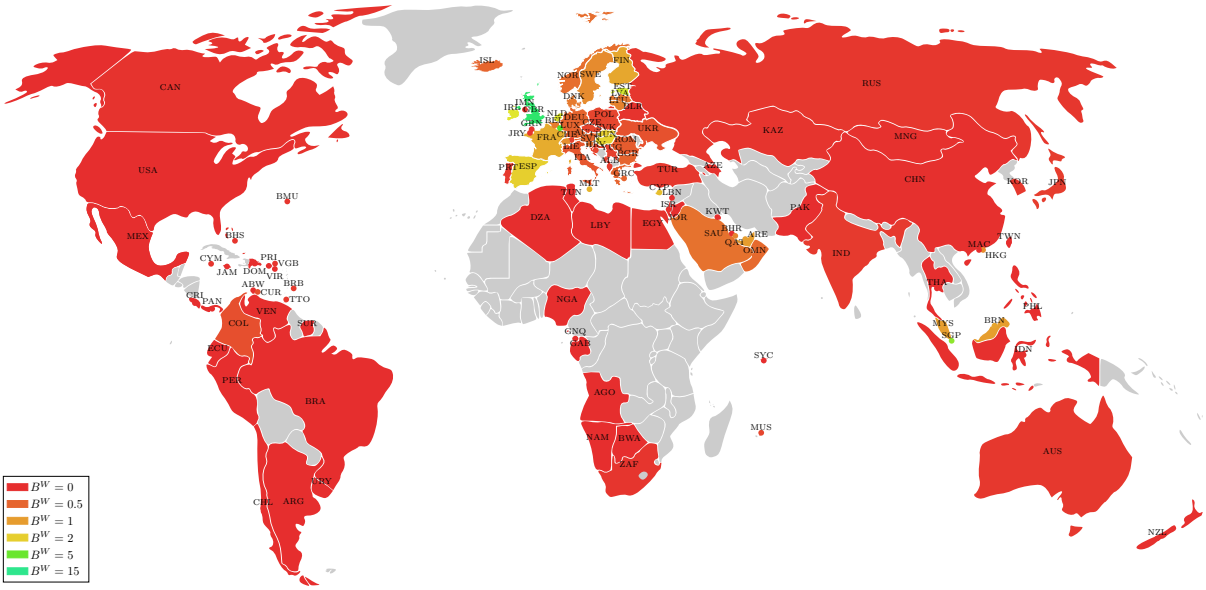


Populaire samenvatting

Is Nederland een belastingparadijs? Nieuwsberichten van de laatste jaren geven inderdaad die indruk. '*Nederland belastingparadijs voor veel multinationals*' [Waa11], '*Nederland is een aantrekkelijk belastingland*' [NOS14], '*Nederlandse meesters in belastingontwijking*' [GM11], zijn titels van nieuwsberichten die deze richting op wijzen.

Om te onderzoeken of Nederland inderdaad een belastingparadijs is, heeft het CPB (Centraal Planbureau) een onderzoek gedaan (zie [RL14] en meer in het bijzonder [RL13]). Multinationale ondernemingen gebruiken Nederland voornamelijk als tussenstation om winsten doorheen te sluizen op een route van een land naar een ander land. Zo beschouwd is Nederland geen *belastingparadijs* (een bestemmingsland waar het geld bewaard wordt, een *tax haven*), maar een *doorsluisland*.

In deze scriptie bekijken we algoritmes voor de netwerkanalyse van bilaterale belastingverdragen vanuit een wiskundig perspectief. Ook zullen we de algoritmes experimenteel toepassen om de rol van Nederland en andere landen als *doorsluislanden* te bekijken, gebruik makend van (belasting)gegevens van 108 landen (of jurisdictiegebieden) die ons door het CPB ter beschikking zijn gesteld.



Figuur B.1: De landen die meer centraal in het netwerk zijn hebben een hogere gewogen betweenness-centraliteit B^W.

Om te onderzoeken welke landen de meest belangrijke doorsluislanden zijn, gebruiken we een instrument om de centraliteit van een punt (een land) in een netwerk te meten: *betweenness-centraliteit*. Wanneer we de betweenness-centraliteit van een land berekenen, bekijken we *alle* 'meest voordelige' belastingroutes voor bedrijven. Op welk deel van deze 'meest voordelige' routes komt ons land voor als doorsluisland? Dit geeft een maat voor de centraliteit van een land in het netwerk. We zullen in het bijzonder *gewogen* betweenness-centraliteit bekijken:

hier krijgen routes die als begin- of startpunt een 'belangrijk' land hebben (waarbij landen met een grote economie 'belangrijk' zijn), een groot gewicht. Deze routes tellen meer mee in de betweenness-centraliteit dan routes die als begin- en eindpunt minder relevante landen hebben.

De belangrijkste experimentele resultaten uit deze scriptie zijn de volgende:

(1.) Nederland staat vrij hoog in de ranglijst: vijfde, als we gewogen betweenness-centraliteit bekijken. Dit lijkt de beweringen uit het nieuws (dat Nederland een aantrekkelijk belastingland voor multinationals is) te ondersteunen.

(2.) Groot-Brittannië is met afstand het meest centrale land in het netwerk. Groot-Brittannië heeft een substantieel hogere betweenness-centraliteit (zowel gewogen als ongewogen) dan *ieder ander* land in het netwerk.

(3.) Er bestaat altijd een 'meest voordelige' route van een land naar een ander land via *ten hoogste drie* doorsluislanden.

(4.) Er is een grote groep landen (64 landen) met de volgende eigenschap: bedrijven kunnen hun winsten *zonder belasting* te betalen versturen van *ieder* land in deze groep naar *ieder ander* land in deze groep, soms over een route via doorsluislanden.

(5.) De ranglijsten blijven vrij stabiel als we ook paden bekijken die *bijna* het 'meest voordeligst' zijn.

(6.) Als Nederland haar rol als 'doorsluisland' wil vergroten en wil zorgen dat bedrijven meer geld door Nederland sluizen dan is het een goed idee om ten eerste de uitgaande dividendbelasting naar India op 0% te zetten en om vervolgens de uitgaande dividendbelasting naar China en Brazilië op 0% te zetten.

(7.) De op de Shapleywaarde gebaseerde betweenness-centraliteit geeft in ons netwerk resultaten (in de ranglijst) die lijken op de gewone betweenness-centraliteit.

(8.) De algoritmes zijn geprogrammeerd in Java en werken sneller dan de implementaties van het CPB. Als het CPB een vervolgonderzoek op [RL14] gaat doen, kunnen de algoritmes die geprogrammeerd zijn voor deze scriptie gebruikt worden.

De resultaten worden geïllustreerd aan de hand van gekleurde wereldkaarten, zoals die in Figuur $B.1$ en $B.2$

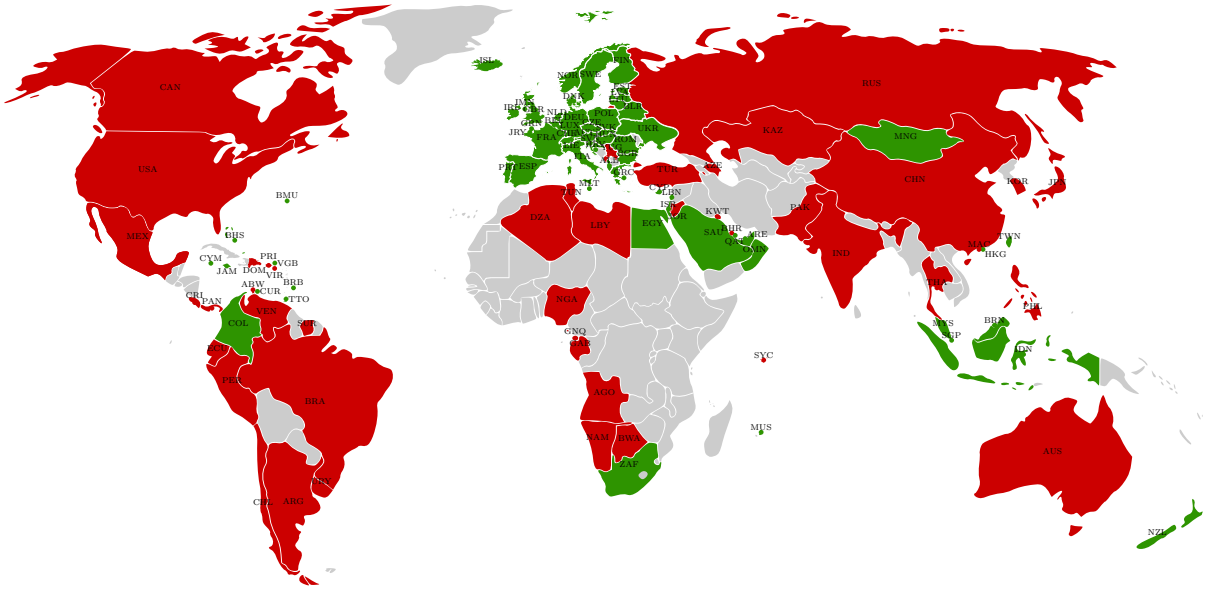
Deze scriptie heeft ook veel theoretische resultaten opgeleverd. Een korte samenvatting.

(1.) Stel dat een land u de geldstroom die bedrijven door hem heensturen wil maximaliseren. Hierbij mag u de uitgaande dividend-belasting naar k andere landen op 0% zetten. Welke k landen moet u dan kiezen? In deze scriptie hebben we bewezen dat dit een heel moeilijk probleem is: het is een zogenaamd 'NP-moeilijk probleem'. Dat betekent[1] dat zelfs snelle computers dit probleem niet efficient kunnen oplossen en heel veel rekentijd nodig hebben.

Wel vinden we een 'approximatie-algoritme' voor het zojuist beschreven probleem (het dividend-belastingverlagingprobleem). Een approximatie-algoritme is een procedure waarmee *snel* een 'goede' oplossing gevonden kan worden: deze oplossing is misschien niet optimaal, maar wel 'bijna'. We kunnen bewijzen dat de approximatieratio, de waarde die de berekende 'goede' oplossing geeft gedeeld door de waarde van de optimale oplossing, nooit kleiner is dan een bepaald getal, in ons geval $1 - 1/e$.

(2.) We bewijzen dat het moeilijk is om het aantal paden binnen een bepaald bereik van het 'meest voordelige pad' te tellen. We vinden een manier om het aantal 'gerestricteerde binnen-bereik-paden' te berekenen: dit kan een computer wel snel.

[1] Althans, aangenomen dat '$P \neq NP$'. Dit vermoeden wordt vaak aangenomen door wiskundigen, maar het is nog nooit bewezen, zie [Coo00].



Figuur B.2: Bedrijven kunnen hun winsten *zonder belasting* te betalen versturen van *ieder* groen land naar *ieder ander* groen land, soms over een route via doorsluislanden.

(3.) Het belastingnetwerk bevat veel verbindingen tussen landen waarover 0% tax geheven wordt. Dat betekent dat héél veel paden gekwalificeerd worden als 'meest voordelige' paden. Dat zorgt ervoor dat de computer er lang over doet om alle paden te berekenen en te tellen. In deze scriptie bekijken we verschillende manieren om deze 0-verbindingen te verwerken.

(4.) We formuleren een 'belastingprobleem'. Hoe hoog moet een land belasting heffen op uitgaande dividenden om de totale ontvangen belasting te maximaliseren? We bekijken dit probleem vanuit een theoretisch perspectief.

(5.) Tot slot vinden we nog een kleine onnauwkeurigheid in een recent artikel (2012) van P. L. Szczepánski, T. Michalak and T. Rahwan (zie [SMR12]): in dit artikel wordt een algoritme gegeven om de 'op de Shapley-waarde gebaseerde betweenness-centraliteit' te berekenen. Dit algoritme wordt gegeven voor ongerichte grafen (netwerken waarin de richting van beweging niet uitmaakt), maar er wordt ook een aanpassing gegeven voor gerichte grafen (hier maakt de richting van beweging wél uit). Deze aanpassing is echter niet helemaal correct (maar het is slechts een klein detail).

Dit was een korte samenvatting van de resultaten uit deze scriptie. Hopelijk vond u het interessant. Het was leuk om deze scriptie te maken: zowel theoretisch als praktisch hebben we interessante resultaten gevonden. Veel plezier met het lezen van de hele scriptie!

Bibliography

[Bra01] U. Brandes, A Faster Algorithm for Betweenness Centrality, *Journal of Mathematical Sociology*, 2001, 25(2), pp. 163–177.

[CLR01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, *Introduction to Algorithms*, Second edition, MIT Press, 2001.

[Coo00] S. Cook, The *P* versus *NP* problem, *Claymath millennium problem*, `http://www.claymath.org/millennium-problems`.

[DJS93] B. Dasgupta, R. Janardan, N. Sherwani, *On the Greedy Algorithm for a Covering Problem*, unpublished manuscript, February 1993.

[EB99] M. G. Everett and S. P. Borgatti, The centrality of groups and classes, *Journal of Mathematical Sociology*, 23, 1999, no. 3, pp. 181–201.

[Fei98] U. Feige, A Threshold of $\ln n$ for Approximating Set Cover, *J. of the ACM*, 1998, 45(5), pp. 634-652.

[Fre78] L. C. Freeman, Centrality in Social Networks: Conceptual Clarification, *Social Networks*, Elsevier, 1, 1978/79, pp. 215–239.

[FS11] M. Fink, J. Spoerhase, Maximum betweenness centrality: approximability and tractable cases, *WALCOM'11 Proceedings of the 5th international conference on WALCOM: algorithms and computation*, 2011.

[GJ79] M. R. Garey, D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, New York: W.H. Freeman, 1979.

[GM11] S. Goodley, D. Milmo, Dutch masters of tax avoidance, *The Guardian*, 19-10-2011.

[Hoc97] D. S. Hochbaum, *Approximation algorithms for NP-hard problems*, PWS Publishing Company, Boston, 1997.

[HP98] D.S. Hochbaum, A. Pathria, Analysis of the greedy approach in problems of maximum k-coverage, *Naval Research Quarterly* 45, 1998, pp. 615–627.

[LP97] Kumar N. Lalgudi, Marios C. Papaefthymiou, Computing strictly-second shortest paths, *Information Processing Letters*, Elsevier, Vol. 63, Issue 4, 1997, pp. 177–181.

[MS07] Miller, Sleaters, *Depth First Search and Strong Components*, Lecture Notes, Carnegie Melon University, 2007.

[New01] M. E. J. Newman, Scientific collaboration networks. II. Shortest paths, weighted networks, and centrality, *Physical Review E*, Vol. 64, No. 016132, 2001, APS.

[NOS14] *Nederland is een aantrekkelijk belastingland*, NOS, 6-11-2014.

[NWF78] G. L. Nemhauser, L. A. Wolsey and M. L. Fisher, An analysis of approximations for maximizing submodular set functions I, *Mathematical Programming* 14, 1978, pp. 265–294.

[Pol61] M. Pollack, The k-th Best Route Through a Network, *Opns. Res.*, Vol. 9, No. 4 (1961), pp. 578–580.

[RL13] Maarten van 't Riet, Arjan M. Lejour, *Nederland belastingparadijs? Nederland doorsluisland!*, CPB Policy Brief, 07-2013.

[RL14] Maarten van 't Riet, Arjan M. Lejour, *Ranking the Stars: Network Analysis of Bilateral Tax Treaties*, CPB Discussion Paper, 10-2014.

[Schä13] G. Schäfer, *Discrete Optimization*, Lecture Notes, Utrecht University, 2013-2014.

[Schr04] A. Schrijver, *Combinatorial Optimization: Polyhedra and Efficiency*, Springer-Verlag, 2004.

[Schr13] A. Schrijver, *A Course in Combinatorial Optimization*, Lecture Notes, 2013.

[SMR12] P. L. Szczepánski, T. Michalak, T. Rahwan, A New Approach to Betweenness Centrality Based on the Shapley Value, *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems – Volume 1*, 2012, pp. 239-246.

[Tar72] R. Tarjan, Depth-first search and lineair graph algorithms, *SIAM J. Comput.*, Vol. 1, No. 2, June 1972, pp. 146–160.

[Val79] Leslie G. Valiant, The complexity of enumeration and reliability problems, *SIAM J. Comput.* Vol. 8, No. 3, August 1979, pp. 410–421.

[Waa11] P. de Waard, Nederland belastingparadijs voor veel multinationals, *de Volkskrant*, 14-10-2011.

[Yen71] Jin Y. Yen, Finding the K shortest loopless paths in a network, *Management Science*, 1971, 17(11): pp. 712–716.