Université Catholique de Louvain
École Polytechnique de Louvain
Pôle d'Ingénierie Mathématique (INMA)

and

Massachusett's Institute of Technology
Laboratory for Information and Decision Systems

On the Policy Iteration algorithm for PageRank Optimization

Romain Hollanders

Supervisors : Vincent Blondel
Raphaël Jungers

June 2010

I want to thank

My supervisor, Professor Vincent Blondel,
for his support and especially for giving me the opportunity to go to MIT,

Raphael Jungers, his post-doctoral fellow,
for his exceptional follow-up and his insightful comments,

Professor John Tsitsiklis,
for inviting me to MIT and for offering valuable advice,

Nathalie Ponet, Emmanuelle Brun and Debbie Deng,
for their effective assistance on procedures that have preceded my departure,

And of course, all those who supported me throughout this work, either by their readings, their advice or simply their encouragement.

Contents

Introduction **1**

1 Theoretical background **3**

1.1 Basic notions and "guide problem" 3

1.1.1 PageRank 3

1.1.2 The PageRank Optimization by Edge Selection Problem 5

1.1.3 Markov Decision Processes 5

1.1.4 Solving Markov Decision Processes 7

1.1.5 The Stochastic Shortest Path Problem 9

1.1.6 Expected first return and hitting times 9

1.2 Literature review on the complexity of solving MDP's 10

1.2.1 On the Simplex method 11

1.2.2 On general Markov Decision Processes 11

1.2.3 On Linear Programming 12

1.2.4 On Value Iteration, discounted case 12

1.2.5 On Policy Iteration 12

2 Algorithms to solve the Max-PageRank problem **15**

2.1 An approximation algorithm based on Interval Matrices 15

2.2 Max-PageRank as a Stochastic Shortest Path problem 16

2.2.1 A first simple SSP formulation 16

2.2.2 A refined SSP formulation 18

2.2.3 Including damping to the problem 20

2.3 A polynomial time algorithm based on Linear Programming 21

2.4 An iterative algorithm based on Policy Iteration 22

3 Experimental results **25**
3.1 Experimental comparison of the algorithms 25
3.1.1 Influence of the graph's size 27
3.1.2 Influence of the number of fragile edges and of the graph's structure . . 27
3.1.3 Influence of damping 29
3.2 Degree of approximation of IM 29
3.3 Experimental performances of PRI 29
3.3.1 Influence of the problem's size 30
3.3.2 Influence of the number of fragile edges 31

4 Results on PageRank Iteration **33**
4.1 Theoretical polynomial upper bound on damped PRI complexity 33
4.1.1 Towards the polynomial upper bound 33
4.1.2 Related questions 35
4.2 Miscellaneous results 36
4.2.1 Properties about MDP's 36
4.2.2 Bounds on the first hitting times 37
4.3 Towards new bounds on undamped PRI complexity 42
4.3.1 Explored- and open- tracks about PRI's general behavior 42
4.3.2 Using and adapting Tseng's result 46
4.4 From undamped to damped Max-PageRank : an approach 47
4.4.1 Bound on the minimum change of value when changing policy 48
4.4.2 Bound on the maximum change of value when increasing damping . . . 50

Conclusion **57**

A Calculation precisions for the minimum value gap **59**

B Numerical bounds **61**

C Implementation of the algorithms **63**

Bibliography **71**

Introduction

In search engines, it is critical to be able to sort web-pages according to their relative importance. This can be done by considering the web as a graph in which the web-pages are nodes and the web-links are directed edges between the nodes. We can sort the nodes according to their importance using their *PageRank*. The PageRank of a node in a directed graph can be seen as the average portion of time spent at this node during an infinite random walk. Even though it has mainly been used for search engines, the notion of PageRank has also many other applications [Ber05].

Sometimes, it happens that the manipulated graph is not fixed and can be modified in some ways. In particular, we may have that a given subset of edges are *fragile*, meaning that the edges of that set can be either present or absent. In this context, optimizing the PageRank of a page depending on which fragile edges are active or not is a natural concern already investigated by [AL04, KND08, IT09, CJB09]. In this document, we mainly focus on this particular aspect. This problem has several real-life applications. For example, a web-master could be interested in increasing the PageRank of one of its web-pages by determining which links under his control he should activate and which ones he should not [KND08]. Another typical example is when some links of the graph can disappear and then reappear uncontrollably. Typically, these links can be broken because the server is down or because of traffic problems. In that case, we still may want to be able to estimate the importance of a node by computing its minimum and maximum PageRank, depending on the presence or absence of the fragile links [IT09].

These applications can both be formulated as a single PageRank optimization problem that we are interested to solve efficiently (i.e., in polynomial time). A first naive approach for that could be for instance to consider and evaluate every possible configuration of the fragile edges, and to choose which one of these configurations gives rise to the maximum (resp. minimum) PageRank for a target node that we initially select. Of course, this method has exponential complexity since the number of possible configurations grows exponentially. In response to this, [IT09] mentioned the lack of efficient methods to solve this problem and they suggested an approximative but efficient method, using *interval matrices*.

Another approach was proposed by [CJB09] to tackle the problem. In their work, they formulate the PageRank optimization problem as a *Stochastic Shortest Path* problem, which is a particular case of *Markov Decision Process*. It is well known that this class of problems can be solved in polynomial time, using linear programming [Ber96]. Therefore, the authors were able to propose an exact and efficient method to solve the original problem.

Unfortunately, the linear programming approach does not catch all of the problem's specificity into account and therefore, it does not behave so well in practice. Consequently, many other algorithms have been suggested in the literature to achieve better performance [SB98]. The *Policy Iteration* method is probably one of the most famous and the most studied among them. The reason is that, even though very few results are known about its theoretical complexity,

it behaves particularly well in practice. In addition, the algorithm is outrageously simple to formulate but it gives rise to unexpectedly complex behaviors. For those reasons, Csáji et al. also proposed an adaptation of Policy Iteration to solve the PageRank optimization problem in [CJB09] ; they call it PageRank Iteration.

In the present work, we mainly study the performances of PageRank Iteration, both experimentally and theoretically. In the experimental part, we compare it with the two other algorithms respectively based on interval matrices and linear programming. We also study its performances on diverse random instances. In the theoretical part, we first prove that PageRank Iteration converges in polynomial time in the special case in which the random walk can be restarted with a certain probability that we refer to as *damping*. We then investigate the remaining case in which there is no damping and propose a number of approaches to find new bounds on the complexity of PageRank Iteration in that case.

More precisely, the first chapter is devoted to provide the reader with the necessary background notions on PageRank, Markov Decision Processes and Stochastic Shortest Path problems. It also summarizes what we know about these concepts from the literature. Chapter two describes the three algorithms that have been proposed to solve the PageRank optimization problem. Then, the comparison of these algorithms and an experimental study of PageRank Iteration is proposed in the third chapter. Finally, a fourth chapter presents what we discovered about the theoretical behaviors of the latter algorithm.

Please note that the two introduction chapters are mainly inspired from [CJB09] since this paper is the starting point of our research. To a smaller extent, they also take inspiration from [MS99], [LDK95], [Ber05] and [SB98]. We sometimes partly reproduced some sequences of these articles when we felt it was appropriate.

Chapter 1

Theoretical background

This introduction aims to provide the reader with the necessary background that we will need during our study and the results that have already been published in the literature and on which our research is based.

1.1 Basic notions and "guide problem"

In this section, we review the concepts that we will need throughout our analysis. We also formalize the problem on which we will concentrate our study, namely the PageRank optimization problem [CJB09]. Therefore, we first give the necessary notions on graphs and on PageRank (in section 1.1.1) to be able to clearly state the problem (in section 1.1.2). Then we define the Stochastic Shortest Path problem, which is a special case of Markov Decision Process, since the most interesting algorithms to solve the PageRank optimization problem are based on their theory. So, section 1.1.3 gives the necessary background on general Markov Decision Processes, section 1.1.4 presents some well known methods that solve that class of problems and section 1.1.5 introduces the Stochastic Shortest Path problem. Finally, we will define the expected first return and first hitting times which are key concepts to link the PageRank optimization problem to the Stochastic Shortest Path problem (section 1.1.6).

1.1.1 PageRank

We consider a *directed graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V} = \{1, ..., n\}$ is a set of vertices and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges. We denote the *adjacency matrix* of $\mathcal{G}$ by A such that $a_{i,j} = 1$ if and only if the edge (i, j) from i to j exists and $a_{i,j} = 0$ otherwise. We also refer to the out-degree of a node i by $\deg(i)$. For simplicity, we first assume that the graph $\mathcal{G}$ is strongly connected so that A becomes irreducible. We will see later what happens when we release that constraint.

We then define a random walk on the graph as follows : if we are on a node i, we jump in the next step to an out-neighbor j of i with probability $1/\deg(i)$. This walking pattern defines a Markov chain on the nodes of the graph with *transition matrix*

$$P \triangleq D_A^{-1} A, \tag{1.1}$$

where $D_A \triangleq \text{diag}(A\mathbb{1})$, $\mathbb{1} = [1, ..., 1]^T$ and diag is an operator that creates a diagonal matrix from a vector. Here, D_A is a diagonal matrix that contains the out-degrees of the nodes on its

diagonal. The transition matrix P is row stochastic and represents an infinite uniform random walk on the graph.

The *PageRank vector* of the graph, denoted by π, is defined as the stationary distribution of the previously described Markov chain and can be seen as the proportion of time spend in each node when walking randomly for an infinite time. More precisely, it is defined by

$$P^T\pi = \pi,$$

where π is non-negative and $\pi^T\mathbb{1} = 1$. Since P is an irreducible stochastic matrix, we know from the Perron-Frobenius theorem that π exists and is unique. Note that π_i can also be interpreted as the importance of node i. The PageRank vector therefore defines a linear order on the nodes.

Let us now consider the more general case where we do not assume that $\mathcal{G}$ is strongly connected. In that case, we can have several connected components that have to be treated independently. However, the main problem is that it can generate nodes that do not have any out-going edges. Such nodes are usually referred to as *dangling* nodes and induce the reducibility of A which is the reason why we have to deal with them. There are many ways to deal with dangling nodes, several being proposed by [Ber05]. We can indeed :

- delete them because they do not influence any other node (although this can create new dangling nodes) ;
- add them only at the end of the PageRank computation ;
- add a self-link and adjust the PageRank at the end of its computation ;
- connect all of them to a unique ideal node that only contains a self loop ;
- connect them to all other nodes of the graph ;

This last solution is usually the most popular and can be interpreted as restarting the random walk once we reach a dangling node. Below, we will consider that we already treated dangling nodes using one of the mentioned solutions and that all nodes have at least one out-going edge accordingly.

We could now define a Markov chain similarly to (1.1) but the stationary distribution of this chain might not be unique. To solve this problem, we define the PageRank vector π of $\mathcal{G}$ as the stationary distribution of the *Google matrix* defined as

$$G \triangleq (1-c)P + c\mathbb{1}z^T$$

where z is a positive *personalization* (or *teleportation*) vector satisfying $z^T\mathbb{1} = 1$ and $0 \leq c \leq 1$ is a *damping* constant. The damping constant and the personalization vector can be respectively seen as the probability to restart the random walk at each time step and the probability distribution for choosing the new starting node. Typically, in practice, values between 0.1 and 0.15 are applied for c and the uniform vector $(1/n)\mathbb{1}$ is chosen for z.

Note that the PageRank vector can be approximated by the *power method* by

$$x_{n+1} = Gx_n,$$

where x_0 is an arbitrary stochastic vector. Besides, its exact value can be obtained by solving a linear system [Ber05] :

$$\pi = c(I - (1-c)P^T)^{-1}z,$$

where I is an $n \times n$ identity matrix. Assuming positive values of the damping constant c and P being stochastic, one can conclude that the matrix $I - (1-c)P^T$ is strongly diagonally dominant and therefore, it is invertible.

1.1.2 The PageRank Optimization by Edge Selection Problem

The problem that motivated and guided our research can be described as follows. We dispose of a subset of edges that are called *fragile*, and we wish to determine the maximum (or minimum) PageRank that a specific node can have. More precisely, we are given a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a target node $v \in \mathcal{V}$ for which we want to maximize (or minimize) the PageRank, and a subset $\mathcal{F} \subseteq \mathcal{E}$ of edges over which we have control, namely the set of fragile edges we can choose to activate or not. We call $\mathcal{F}^+ \subseteq \mathcal{F}$ the subset of edges that are turned "on" (or *activated*) and $\mathcal{F}^- = \mathcal{F} \backslash \mathcal{F}^+$ the ones that are turned "off" (or *deactivated*), the edges in $\mathcal{E} \backslash \mathcal{F}$ remaining fixed.

So, the problem we are addressing consists in choosing the right subset $\mathcal{F}^+ \subseteq \mathcal{F}$ such that the PageRank of node v is maximal (or minimal). In short, the problem can be stated as follow [CJB09] :

THE MAX-PAGERANK PROBLEM

Instance :	A digraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a node $v \in \mathcal{V}$ and a set of controllable edges $\mathcal{F} \subseteq \mathcal{E}$.
Optional :	A damping constant $0 \leq c \leq 1$ and a stochastic personalization vector z.
Task :	Compute the maximum possible PageRank of v by selecting edges in $\mathcal{F}$ and provide a configuration of edges in $\mathcal{F}$ for which the maximum is taken.

The Min-PageRank problem can be stated similarly. In addition, the algorithms to solve it can be obtained with a straightforward modification from the ones we will use for the Max-PageRank problem as well.

It is also interesting to see that it is possible to solve the problem using a simple brute force approach since the number of different configurations of the fragile edges is finite. If we have f fragile edges, there will be 2^f possible configurations, each of them needing to solve a linear system in order to evaluate the corresponding PageRank of the target node v. The solution of the linear system can possibly be computed in $O(n^{2.376})$ using the Coppersmith-Winograd algorithm[1] [CW90], for a total complexity of $O(n^{2.376}2^f)$. We will see later that it is possible to achieve polynomial complexity to solve this problem, which is far more efficient than this somewhat simplistic exponential complexity.

1.1.3 Markov Decision Processes

We now give an introduction to *Markov Decision Processes* (MDP's) since the goal of this work is mainly to study the performances of some existing algorithms to solve them. We will focus on a special class of MDP's known as *Stochastic Shortest Path* (SSP) problems, especially in the context of the Max-PageRank problem which, as we will see, can be formulated as a particular SSP problem. We start by introducing MDP's in their general form and will talk about methods to solve them as well as SSP right afterwards.

A Markov Decision Process describes the dynamics of an *agent* interacting with a stochastic *environment*. It is defined by a tuple $(\mathcal{S}, \mathcal{U}, \mathcal{P}, \mathcal{C})$, where :

- $\mathcal{S} = \{1, 2, ...\}$ is the finite set of *states* in which the environment can be,
- $\mathcal{U}$ is the finite set of possible *actions* and $\mathcal{U}_s \subseteq \mathcal{U}$ the set of actions available to the agent in state s,

[1] In practice however, one should prefer other methods like for example the power method.

- $\mathcal{P}$ is the table of *transition probabilities* between states where $\mathcal{P}^u_{s,s'}$ is the probability of going from state s to state s' starting from s and taking action $u \in \mathcal{U}_s$,
- $\mathcal{C}$ is the *cost function* where $\mathcal{C}^u_{s,s'}$ is the cost incurred by the agent when taking action u in state s and arriving at state s'.

We usually denote the size of the state space $\mathcal{S}$ by N and the maximum number of actions per state $\max_s |\mathcal{U}(s)|$ by M, where $|\cdot|$ is the cardinality of the corresponding set. Of course, the transition probabilities are non negative and they sum to one, namely for all state s, s' and action $u \in \mathcal{U}_s$, $\mathcal{P}^u_{s,s'} \geq 0$ and

$$\sum_{s' \in \mathcal{S}} \mathcal{P}^u_{s,s'} = 1.$$

From an initial state, the Markov Decision Process describes the subsequent evolution of the system state over a (possibly infinite) sequence of times referred to as *stages*. In this work, we will exclusively focus on the *infinite-horizon* case in which the sequence of stages is infinite. The qualifier "Markov" stands for the Markov property which states that the present state contains all the necessary information to determine the next one.

The agent's *expected return* is the sum of the expected costs $\sum_{t=0}^{\infty} \gamma^t c_t$ incurred during the infinite process, where c_t is the cost paid at stage t and $0 \leq \gamma \leq 1$ is the *discount factor* that makes future costs less penalizing than immediate costs. Having a discount factor strictly less than 1 is meaningful since it allows to avoid infinite returns. As we will see later on, research has already produced many results to treat that case. However, under certain conditions, the undiscounted case (i.e., with $\gamma = 1$) can make sense. We will see this for instance with the Stochastic Shortest Path Problem which roughly corresponds to that particular case. For the time being however, we will suppose that we are in the discounted case (where $0 \leq \gamma < 1$) which will temporarily free us from certain conditions that would otherwise be necessary.

The agent's goal is to maximize his return through the best possible choice of actions. In this context, we define a *policy* μ as a mapping from states to actions that gives for any state $s \in \mathcal{S}$, the probability distribution for taking any action in $\mathcal{U}_s$. It can be seen as the strategy used by the agent to interact with his environment. We say that a policy is *deterministic* when the probability of taking an action u in a state s is either 0 or 1. In the end, the goal is to find the optimal policy μ^* that would maximize the agent's expected return (wrt. the chosen discount rate).

Another useful quantity to characterize MDP's are *value* or *costs-to-go* functions. The cost-to-go function of a policy μ gives us the expected total costs that we incur starting from a state and following μ thereafter. That is, for all states s,

$$J^\mu_s \triangleq \lim_{k \to \infty} E_\mu \left[\sum_{t=0}^{k-1} \mathcal{C}^{u_t}_{s_t, s_{t+1}} \;\middle|\; s_0 = s \right],$$

where s_t and u_t are random variables representing the state and the action taken at stage t, respectively. Action u_t is chosen by the policy μ and state s_{t+1} is chosen by the probability distribution $\mathcal{P}$, both using the starting state s_t.

We say that μ_1 *dominates* μ_2 if

$$J^{\mu_1}_s \leq J^{\mu_2}_s \tag{1.2}$$

for all states $s \in \mathcal{S}$ and we write $\mu_1 \succeq \mu_2$. We will also write $\mu_1 \succ \mu_2$ if the inequality in (1.2) is strict for at least one state s. We say that a policy is optimal if it dominates all other policies. There may be more than one optimal policy but all of them share the same unique

value function J^*. In that case, J^* is the unique solution of the *Bellman equation* $TJ^* = J^*$, where T is the *Bellman operator* that, for all states s, is defined by

$$TJ_s \triangleq \min_{u \in \mathcal{U}_s} \sum_{s'=1}^{N} \mathcal{P}^u_{s,s'} \left[\mathcal{C}^u_{s,s'} + \gamma J_{s'} \right]. \tag{1.3}$$

The Bellman operator of a policy μ is defined similarly for all state s as

$$T_\mu J_s \triangleq \sum_{u \in \mathcal{U}_s} \mu(s,u) \sum_{s'=1}^{N} \mathcal{P}^u_{s,s'} \left[\mathcal{C}^u_{s,s'} + \gamma J_{s'} \right], \tag{1.4}$$

where $\mu(s,u)$ is the probability of taking action u in state s according to policy μ. This definition is of course also valid for deterministic policies since they are special cases of the randomized ones. Again, we have that $T_\mu J^\mu = J^\mu$ which implies that

$$\lim_{k \to \infty} T^k_\mu J = J^\mu, \qquad \lim_{k \to \infty} T^k J = J^*.$$

Moreover, we know that operators T and T_μ are monotone and contracting with respect to a weighted maximum norm [Ber96]. Therefore, we can say that a policy μ is optimal if and only if $T_\mu J^* = TJ^* = J^*$.

Finally, given a policy μ, we define the *modification set* $\mathcal{T}^\mu \subset \mathcal{S} \times \mathcal{U}$ as the set of all pairs (s,u) such that a policy μ' obtained by changing the action of μ by u in state s improves the cost-to-go of s, i.e., $J^{\mu'}_s < J^\mu_s$. Furthermore, [MS99] shows that in that case, μ' strictly dominates μ. So, intuitively, $\mathcal{T}^\mu$ can be seen as the set of all single modifications of μ that would result in a better policy μ'. Let $\texttt{states}(\mathcal{T}^\mu) = \{s : (s,u) \in \mathcal{T}^\mu\}$ be the set of states that appear in $\mathcal{T}^\mu$. We say that a modification set $\mathcal{T}^\mu$ is well defined if each state s appears at most once. Clearly, if $|\mathcal{U}_s| \leq 2$ for all states s, $\mathcal{T}^\mu$ is always well defined.

1.1.4 Solving Markov Decision Processes

Many methods are known for solving Markov Decision Processes [SB98]. Yet, three main methods have been dominating the literature, namely *Linear Programming*, *Policy Iteration* and *Value Iteration*. Here are the outlines of how they work and behave.

The problem of computing the optimal cost-to-go function can be formulated as a Linear Program (LP) [D'E63] and can therefore be computed in polynomial time. More precisely, the optimal cost-to-go function $J^*_1, ..., J^*_N$ is the solution of the following optimization problem in the variables $x_1, ..., x_N$:

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{N} x_i \\ \text{subject to} \quad & x_i \leq \sum_{j=1}^{N} \mathcal{P}^u_{i,j} \left[\mathcal{C}^u_{i,j} + \gamma\, x_j \right] \end{aligned}$$

for all states i and for all action $u \in \mathcal{U}_i$. Even though LP is a general method that does not specifically deal with the special structure of MDP's, it is the first proof that shows that MDP's can be solved in polynomial time and the only one known when treating undiscounted problems. We will further discuss that question in section 1.2.

The Policy Iteration (PI) method is an iterative algorithm due to Howard (1960). Starting from an arbitrary policy, its principle is to alternate between a value determination step in which the current policy (that is a priori not yet optimal) is evaluated, and a policy improvement step in which we try to improve the current policy based on its evaluation. Several strategies are possible for the improvement step and that is why we will rather consider the three instead of two following steps for each iteration :

1. Evaluate the value J^{μ_i} of the current policy μ_i and use the result to determine $\mathcal{T}^{\mu_i}$. Let $\texttt{evaluate}(\mu_i)$ return $\mathcal{T}^{\mu_i}$.
2. Select a subset U of pairs (s, u) in $\mathcal{T}^{\mu_i}$ that represents the modifications that will be applied to μ_i. This step can be seen as the strategy for improving the current policy. Let $\texttt{select}(\mathcal{T}^{\mu_i})$ choose the subset U according to the chosen strategy. Note that any choice for the set U would result in an improvement of the policy and therefore, if $\mathcal{S}$ and $\mathcal{U}$ are finite sets, PI always converges in finitely many iterations to the optimum [MS99].
3. Modify the current policy according to the set U. Therefore, we define $\texttt{modify}(\mu_i, U)$ that returns the improved policy μ_{i+1} that is identical to μ_i for all states that are not in $\texttt{states}(U)$ and is such that $\mu_{i+1}(s) = u$ for all $(s, u) \in U$.

The resulting algorithm is given by Algorithm 1.

Algorithm 1 Policy Iteration

Require: An arbitrary policy μ_0, $\mathcal{T}^{\mu_0} \neq \emptyset$, $i = 0$.
Ensure: The optimal policy μ^*.

1: **while** $\mathcal{T}^{\mu_i} \neq \emptyset$ **do**
2: $\quad \mathcal{T}^{\mu_i} = \texttt{evaluate}(\mu_i)$.
3: $\quad U = \texttt{select}(\mathcal{T}^{\mu_i})$.
4: $\quad \mu_{i+1} = \texttt{modify}(\mu_i, U)$.
5: $\quad i \leftarrow i + 1$.
6: **end while**
7: **return** μ_i.

Since the number of different possible policies is finite, Policy Iteration converges to the optimal policy in a finite number of iterations. Note that the `select` strategy should be defined a priori. Depending on the chosen strategy, the efficiency of the resulting algorithm may vary as we will see in section 1.2. Here are a few well known strategies :

- taking $U = (s, u) \in \mathcal{T}^{\mu_i}$ where the index of state s is minimal, is known as the Smallest-Index strategy ;
- taking $U = (s, u) \in \mathcal{T}^{\mu_i}$ such that $J_s^{\mu_i}$ is minimal results in the Most-Negative-Reduced-Cost (MNRC) strategy ;
- taking $U = \mathcal{T}^{\mu_i}$ results in the so called Greedy strategy.

Of course, many other strategies can be defined. The most popular among them is probably the Greedy one and hence, it will be our main focus in this work.

One drawback of PI is that the evaluation step requires to solve a linear system, which can be inefficient. Unlike Policy Iteration that works on improving the policy, the Value Iteration method tries to improve the value function. It starts from an arbitrary value and then iterates using the simple relation $J \leftarrow TJ$ which does not require to solve any system. As we saw

earlier, the process converges to the optimal value function J^*. Yet, there is no guarantee that the algorithm will converge in a finite number of iterations and a stopping criterion is therefore needed. However, it is possible to identify the optimal policy after a given finite number of iterations [Tse90]. Note that the value of J at a given iteration does not always correspond to the value of an existing policy. More results on the complexity of VI will be discussed in section 1.2.

1.1.5 The Stochastic Shortest Path Problem

We now introduce the Stochastic Shortest Path (SSP) problem since our main goal is to study how well it can be solved using the PI algorithm. In particular, we are going to formulate the Max-PageRank problem as an SSP problem and study the theoretical and experimental behavior of PI in that case.

A Stochastic Shortest Path problem is a generalization of the classical deterministic Shortest path problem [Ber96]. It differs from the latter in that the transition probabilities are non-deterministic but can be controlled in some way. The goal is to find a control policy that would enable us to reach a target node from a given starting node with probability one while minimizing the expected cumulative encountered costs, which also depend on the applied policy.

SSP problems are one of the most important particular cases of undiscounted MDP's. Such problems have a finite state- and action space and they require an absorbing, cost-free terminal state to be well defined. SSP's are of high practical importance since they arise in many real-world applications such as in several operational research problems (e.g. routing and resource allocation problems) [CJB09].

An SSP problem can be formulated as an MDP with state space $\mathcal{S} = \{1, 2, ..., N\}$, where $\tau \triangleq N$ is a special state called the *target* or *termination* state, action space $\mathcal{U}$, transition-probability function $\mathcal{P}$ and cost function $\mathcal{C}$. The terminal state is absorbing and cost-free, so for any action $u \in \mathcal{U}_\tau$, $\mathcal{P}^u_{\tau,\tau} = 1$ and $\mathcal{C}^u_{\tau,\tau} = 0$. The problem is to find a policy μ that reaches the state τ with probability one while minimizing the expected cumulative costs, namely minimizing J^μ_s for all states s.

Depending on the chosen policy, it can happen that not every state (sometimes even not the termination state) is accessible from others. To handle such events, we make the distinction between a policy that enables each node to reach τ with probability one, which is called *proper*, and a policy that does not, which is called *improper*. If we apply a proper policy, we do get a finite horizon problem. However this horizon may be random and will depend on the applied policy.

In the special case of SSP, we also have that every optimal policy shares the same optimal value function J^* provided that (1) there exists at least one proper policy and (2) every improper policy yields infinite cost for at least one initial state. In that case, J^* is the unique solution of the Bellman optimality equation $TJ^* = J^*$. To solve that equation, we can for instance use one of the above mentioned algorithms (section 1.1.4) which are guaranteed to converge [Ber96]. Note that operators T and T^μ, respectively defined by (1.3) and (1.4), are still monotone but they only are contractions if all allowed control policies are proper.

1.1.6 Expected first return and hitting times

A last tool that will be used during our study is the concept of *expected first return* and *expected first hitting time*. We define the expected first *return* time as the average portion of

time between two consecutive visits of a same given node. More precisely, let $(X_0, X_1, ...)$ be a homogeneous Markov chain on a finite set of states $\mathcal{S}$, the expected first return time of a state $s \in \mathcal{S}$ is given by

$$\phi_s \triangleq E[\min\{t \geq 1 : X_t = s\}|X_0 = s].$$

This concept is closely related to both the Max-PageRank and the SSP problem. It will therefore help us to relate one to the other. Indeed, suppose that we take the graph on which we considered the random walk in the Max-PageRank problem and identify it to a Markov chain. It is easy to see that, provided that the chain is irreducible, PageRank and expected first return time share an inverse relation such that

$$\pi_s = \frac{1}{\phi_s}.$$

It follows that maximizing the PageRank of s is equivalent to minimizing its expected first return time. Note that if s is *recurrent* (i.e., the stationary probability of being in s is non zeros), its expected first return time is finite and π_s is well defined. If s is *transient* instead (i.e., there is no recurrent state s' such that $\mathcal{P}^{\mu}_{s',s} > 0$), we need the convention that $1/\infty = 0$ since the expected first return time of a transient state is ∞.

Similarly, in the frame of the SSP problem, we define the expected first *hitting* time which is the average portion of time needed to reach the target state τ from any given state s (not only τ in opposition to the first *return* time) and is denoted by

$$\varphi_s \triangleq E[\min\{t \geq 0 : X_t = \tau\}|X_0 = s].$$

Note that for the target state τ, expected first return and hitting times are equal. In our study, we will especially work with first hitting times even though first return times are more directly linked to PageRank.

It is also interesting to see that in the graph corresponding to the Max-PageRank problem (in which v is the equivalent for τ), we can simply compute the expected first hitting times of the nodes by

$$\varphi = (I - \overline{P})^{-1}\mathbb{1}, \tag{1.5}$$

where $\overline{P}$ is obtained from the transition matrix P by setting the v^{th} column to zero. More precisely, $\overline{P} = P \cdot \text{diag}(\mathbb{1} - e_v)$, where e_v is the v^{th} n dimensional canonical basis vector. Note that in the damped case ($c > 0$), we use the Google matrix G instead of P. To be convinced that equation (1.5) is true, one can see that it can be reformulated as $\varphi = \overline{P}\varphi + \mathbb{1}$. This last formulation simply says : *the first hitting time of a node i is the same as the weighted sum of the hitting times of its neighbors plus the cost of the displacement to reach them, which is one.* The reason why we set the v^{th} column to zero comes from the fact that the terminal node has a first hitting time of zero. Note that if node v can be reached from all others, then $I - \overline{P}$ is invertible.

1.2 Literature review on the complexity of solving MDP's

We now give an overview of the main existing results on the complexity of solving MDP's, on which our research has been built. In particular, we start with a recent result on the simplex algorithm that has improved in many ways knowledge about the complexity of MDP's (section 1.2.1). Then, we summarize some results on MDP's (section 1.2.2), as well as on the three main methods we already presented to solve them, namely Linear Programming (section 1.2.3), Value Iteration (section 1.2.4) and especially Policy Iteration (section 1.2.5) which is our main focus in this work.

1.2.1 On the Simplex method

There have been some recent breakthroughs on the complexity of solving MDP's with a fixed discount rate. A number of them are based on the fact that the classical simplex method with the Most-Negative-Reduced-Cost (MNRC) pivoting rule is a strongly polynomial time algorithm for solving MDP's [Ye10]. By strongly polynomial time algorithm, we mean an algorithm that solves an MDP in a number of arithmetic operations polynomial in N and M, respectively the number of states and the maximum number of actions per state. Note that the behavior of the simplex is quite similar to that of PI and the same strategies are thus appropriate for both. Here is a summary of what we currently know about simplex for solving LP problems or MDP's.

The debate is still open on whether or not there exists a pivoting rule such that the simplex method has polynomial complexity for solving a general linear program. Several rules have been proposed but many of them have given an exponential number of iterations on carefully built examples, among which the MNRC pivoting rule.

Meanwhile, the simplex method with the MNRC pivoting rule has been shown to be strongly polynomial for solving discounted MDP's with fixed discount rate $0 \leq \gamma < 1$ [Ye10]. Surprisingly, the same simplex algorithm can generate exponential complexity under the same working conditions if we use the Smallest-Index pivoting rule instead [LDK95].

In addition, [Ye10] shows that the MNRC simplex method is equivalent to the Policy Iteration method with the same rule used to choose the next updated nodes at each iteration (we will refer to this version of PI by MNRC PI). It also mentions that the Greedy version of PI (or simply Greedy PI) is at least as good as the MNRC version. Therefore, the Greedy Policy Iteration algorithm converges in strongly polynomial time for MDP's with fixed discount rate.

From the above, and using the fact that MDP's can be solved using linear programming, one can claim that MDP's with fixed discount rate $0 \leq \gamma < 1$ are in some sense easier to solve than general linear programs. This is not trivial since that very same question has been discussed for years : it was unknown whether MDP's could be equivalent to general linear programs or if it was possible to use their special structure to build more efficient algorithms than the regular LP methods to solve them. Note that if the opposite had been shown, we could have been able to transpose MDP-specific methods such as PI to solve LP problems.

1.2.2 On general Markov Decision Processes

There is no known algorithm that can solve any general MDP in strongly polynomial time. However, MDP's can be solved in weakly polynomial time using linear programming, meaning that the complexity also depends on the size L of the input (i.e., the maximum number of bits required to represent the problem's data). This dependence is due to the fact that no strongly polynomial time algorithm is known for solving LP problems yet. If such an algorithm was to be found, MDP's could be solved in strongly polynomial time as well. However, note that LP is a general technique and methods for solving such problems do not take the special structure of MDP's into account.

We can also overcome the dependence in L with discounted MDP's if the discount factor $0 \leq \gamma < 1$ is not an input of the problem. This can indeed be achieved using Greedy PI, which converges in a number of iterations that is bounded by a polynomial in N, M and $1/(1-\gamma)$, as shown in [Ye10] with its result on the simplex method described earlier.

Despite the above, the big question of knowing whether the undiscounted case (which in our case is essentially SSP) can also be solved in strongly polynomial time still remains. A priori, a solution cannot easily be deduced from the arguments used for the discounted case. Confirming this, one can actually see that all undiscounted MDP's are not equally difficult and that the efficiency of the algorithms used to solve them strongly depends on their structure. To illustrate that, here are two particular instances for which results have been produced. The first one is easier to solve than most undiscounted MDP's, the other one is harder.

- In a case such that all transition probabilities are deterministic (i.e., $p_{i,j} \in \{0, 1\}$ for all i, j), it is possible to find a strongly polynomial time algorithm to solve the corresponding MDP [PT87].
- Solving a general SSP problem with greedy PI can yield an exponential number of iterations for particular instances [Fea10]. This contrasts with the fact that the very same PI method has strongly polynomial complexity when solving discounted MDP's with fixed discount rate.

1.2.3 On Linear Programming

As mentioned earlier, linear programs can be solved in weakly polynomial time in the number of variables n and constraints m, and the size of the input L. The algorithm proposed by [Gon88][2] has been shown to run in $O(n^3L)$ provided that the number of constraints $m = O(n)$. In practice, the simplex algorithm can be used and works well on sufficiently small instances, even if the worse case complexity is exponential. Further research could lead to a pivoting rule that would provide polynomial complexity. However the question whether there exists a strongly polynomial algorithm to solve LP problems for now remains open.

1.2.4 On Value Iteration, discounted case

Value Iteration converges to the optimum [BT91] and we can identify the optimal solution after a finite number of steps. Moreover, the number of iterations is bounded by a polynomial in the size L of the inputs and in the ratio $1/(1-\gamma)$, or more precisely by $O(N \log(N\delta)/(1-\gamma))$, where δ is the accuracy in the problem data (i.e., the smallest integer such that any problem data multiplied by δ is an integer). This result comes from [Tse90] and uses the fact that the Bellman operator T defined by (1.3) is a contraction of modulus $\gamma < 1$ such that $\|T(x)-x^*\| \leq \gamma\|x-x^*\|$. As a consequence, VI converges in weakly (or pseudo) polynomial time for a fixed discount rate $0 \leq \gamma < 1$.

Furthermore, [LDK95] shows that the dependence in $1/(1-\gamma)$ can actually be achieved. They found an example for which the number of iterations K is bounded below by $\frac{1}{2}\log\left(\frac{1}{1-\gamma}\right)\frac{1}{1-\gamma}$.

1.2.5 On Policy Iteration

Each step of Policy Iteration generates an improved policy that strictly dominates every previously generated policy. Together with the fact that there are at most M^N distinct policies, this implies that PI always converges to the optimal solution in at most M^N iterations. Furthermore, each iteration has two steps :

[2] In practice, one would prefer to use the $O(n^{3.5}L)$ algorithm from [Kar84] since it behaves usually better. Yet, the theoretical complexity of Gonzaga's algorithm shows at least that the problem of solving a linear program is intrinsically an $O(n^3L)$ problem.

- a value determination step of the present policy, which roughly consists in solving a linear system and can be solved for example using the Coppersmith-Winograd inversion algorithm in $O(N^{2.376})$ arithmetic operations[3] ;
- a policy improvement step that creates the new improved policy and takes $O(MN^2)$ arithmetic operations.

Consequently, we have that one step of PI runs in $O(N^{2.376} + MN^2)$. Together with the upper bound on the number of iterations, we get a first exponential upper bound on the complexity of PI : $O((N^{2.376} + MN^2)M^N)$.

This complexity bound may seem unrealistic since Policy Iteration usually works well in practice and takes far less iterations. However, it is not always the case as shown by [MC90]. Their example achieves 2^N iterations with the Smallest-Index criterion for PI.

That example tells us one important thing : the selection criterion for the next states to update is essential. In our case, we will especially study the Greedy version of PI. A first result that applies to Greedy PI and that was proposed by Mansour and Singh reduces the needed number of iterations to $O(M^N/N)$. The question is now : can we do better?

Several interesting results have been found on PI when working with discounted MDP's ($0 \leq \gamma < 0$). One of the most remarkable of them states that the number of iterations K of MNRC PI is at most :

$$\frac{N^2(M-1)}{1-\gamma} \log\left(\frac{N^2}{1-\gamma}\right). \tag{1.6}$$

This breakthrough result from [Ye10] first shows that MNRC PI converges in strongly polynomial time for a fixed value of the discount rate. Furthermore, as mentioned earlier, Greedy PI is at least as good as MNRC PI which means that the same bound on the number of iterations can be used. To our knowledge, this is the first strongly polynomial result for bounding the complexity of algorithms that solve MDP's. However, the question if the dependence in $1/(1-\gamma)$ can be dropped is non trivial and remains open.

An alternative weakly polynomial result on Greedy PI for discounted MDP's can also be derived from Tseng's and Puterman's works [Tse90], [Put94]. Puterman stated that Greedy PI converges at least as fast as VI in terms of their number of iterations while Tseng limited the number of iterations of the latter by $O(N\log(N\delta)/(1-\gamma))$. Those two results put together lead to the same $O(N\log(N\delta)/(1-\gamma))$ bound for PI which is weakly polynomial since $N\log(N\delta)$ is polynomial in the binary input size L.

Another important result from [Tse90] shows that the Bellman operator T is also a contraction for undiscounted MDP's ($\gamma = 1$) with respect to a certain weighted norm such that $\|T(x) - x^*\|^\omega \leq \alpha\|x - x^*\|^\omega$, where $\alpha < 1$ is the contraction factor. This enables us to use the same result from Tseng and Puterman that has been found for Greedy PI with discount rate, except that the contraction factor α (which is the equivalent for γ) is now fixed by the problem instance. However, it can be shown that $1-\alpha \geq \eta^{2r}$, where η is the smallest transition probability and r is the diameter of the unweighted graph associated with the MDP for any policy. Applying Tseng and Puterman, the resulting bound on the number of iterations of PI is given by $O\left(\frac{N\log(N\delta)}{\eta^{2r}}\right)$. This bound means that if (1) the smallest transition probability is not exponentially small with respect to the problem's input and (2) that the diameter is bounded by a constant, we have that PI converges in weakly polynomial time for undiscounted MDP's.

[3]Note that, in practice, we never inverse the system matrix to solve a linear system. We can make use of the Power method instead, for instance.

For the general SSP problem, which is roughly equivalent to the undiscounted case, [Fea10] found an example where Greedy PI achieves 2^N iterations which gives us an exponential lower bound for the complexity of PI in that case. However, the instance used by Fearnley to show his result makes use of some exponentially small transition probabilities. This kind of examples could therefore not exist if such transition probabilities were prohibited.

Chapter 2

Algorithms to solve the Max-PageRank problem

In this chapter, we present three algorithms that were designed to solve the Max-PageRank problem. The first one is an approximation algorithm that runs in polynomial time. Its theory is based on interval matrices. To be able to create an algorithm that gives us an exact solution, we next formulate the Max-PageRank problem as an SSP problem and describe the necessary steps of the construction. Using this formulation we are able to build two other algorithms :

- one based on linear programming that runs in (weakly) polynomial time ;
- the other based on the Policy Iteration method that seems to be more efficient in practice than the two preceding ones, but for which we have few theoretical results concerning the complexity.

For the sake of simplicity, we assume that the graph on which we make the random walk has no multiple edges.

2.1 An approximation algorithm based on Interval Matrices

A first approach to solve the Max-PageRank (resp. Min-PageRank) problem was proposed By [IT09]. At that time, they where mentioning the lack of polynomial time algorithms for solving the problem. Therefore, they proposed an approximation algorithm, using interval matrices and linear programming.

Broadly, their solution proposes to consider all Google matrices G for all possible policies and then to take the most extreme values for each component of the matrices. More precisely, they define a matrix interval $I_G = [\underline{G}, \overline{G}]$ such that for any possible Google matrix G and component $(i, j) \in \mathcal{V} \times \mathcal{V}$, we have $\underline{G}_{i,j} \leq G_{i,j} \leq \overline{G}_{i,j}$. With that matrix interval, they are able to define a polytope that contains the PageRank vectors corresponding to all possible Google matrices. Then, using linear programming component by component on that polytope, they obtain a vector whose components bound above (resp. below) the components of any PageRank vector corresponding to a possible Google matrix. Thus, if we only maximize (resp. minimize) along the dimension corresponding to the target node, we of course get a bound on the maximum (resp. minimum) achievable PageRank of that node.

This solution has a bright side. The algorithm indeed runs in (weakly) polynomial time and the size of the linear problem does not depend on the number of fragile edges. Moreover, the

time needed to build the polytope with the interval matrices is negligible compared to the time needed to solve the linear program. Therefore, we can consider that the general complexity is also independent from the number of fragile edges. All in all, since there are n variables and $O(n)$ constraints, the algorithm runs in $O(n^3L)$ arithmetic operation if we use the algorithm describes in [Gon88] for solving the linear problem for instance.

On the other hand, the solution computed by the algorithm only approximates the exact solution and the relative error between the exact and the computed solutions grows linearly with the number of fragile edges of the instance. We will therefore introduce other algorithms that can compute the exact solution of the Max-PageRank problem with better or comparable execution times.

2.2 Max-PageRank as a Stochastic Shortest Path problem

In this section, we will formulate the Max-PageRank problem as a Stochastic Shortest Path problem. We will draw this section from [CJB09] that first introduced this formulation. Since we already saw that we can efficiently solve SSP (i.e., in polynomial time, see section 1.1.4), this formulation will lead to efficient algorithms for the Max-PageRank problem as well.

To keep things simple, we first make the two following assumptions :

(AD) *Dangling Nodes Assumption* : Each node of the graph has at least one fixed out-going edge. This ensures that there are no dangling nodes under any possible configuration of the fragile edges (i.e., there are no nodes with only fragile out-going edges).

(AR) *Reachability Assumption* : There is at least one configuration of the fragile edges for which the target node v can be reached with positive probability from any other nodes. This assumption is required to have a well defined PageRank for at least one configuration. In our SSP formulation, it will be equivalent to requiring that there is at least one proper policy, which statement can be checked in polynomial time [Ber96].

These assumptions are not restrictive and we show below how to drop them, in particular by adding damping to the problem. In that case, the Markov chain becomes irreducible for any configuration of the fragile edges and thus, all policies are proper and (AR) is always satisfied.

Using these assumptions, we first give a simple SSP formulation for the Max-PageRank problem. It is simple in the sense that it is quite straightforward but, as we will see, it does not lead to a polynomial time algorithm yet. This will need a refinement of the formulation that will meet this requirement. Finally, we will see how this last formulation can be adapted if we add damping to the problem, while maintaining its properties.

2.2.1 A first simple SSP formulation

First of all, let's remind the intuition about the notion of PageRank that links it to the concept of expected first return time. Actually, the PageRank of node v could be interpreted in two different ways :

- either as the average portion of time spent in v by an infinite random walk ;
- or as the inverse of the average time between two consecutive visits of v (which time is called the expected first return time of v).

So, we are interested in activating the best subset of fragile links in order to maximize the PageRank of v, or minimize its expected first return time. This last minimization problem will correspond to an SSP problem where the states correspond to the nodes of the graph and the actions to the set of fragile edges we decide to activate in each node, among other things. Figure 2.1 illustrates how this SSP formulation affects the original graph.

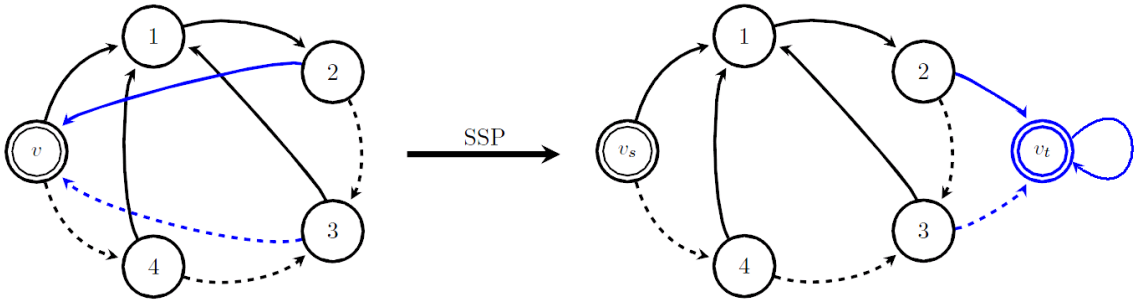

Figure 2.1: An example showing how the SSP formulation in which we wish to minimize the first return time of the target node v affects the original graph. Node v is split in two : the starting node v_s and the termination node v_t. The dashed edges correspond to the fragile links. The question is : which fragile edges should we activate in order to join v_t from v_s as fast as possible, following the original random walk ?

More precisely, let's consider an instance of the Max-PageRank problem. We will build an associated SSP problem as follows :

- The states of the SSP are the nodes of the graph except for the target node v that we split in two nodes v_s and v_t and that we respectively call the *starting* and the *termination* node. To the starting node v_s, we give all the out-going links of v and to the termination node v_t, we give all the incoming links and one self-loop link. That way, $\tau = v_t$ is an absorbing termination state, required in any SSP instance.
- An action in state s is to select a subset of fragile edges $\mathcal{F}_s^+ \subseteq \mathcal{F}_s$ that we activate, while all other fragile edges $\mathcal{F}_s^- = \mathcal{F}_s \backslash \mathcal{F}_s^+$ going out from s are deactivated, $\mathcal{F}_s \subseteq \mathcal{F}$ being the set of fragile edges going out from s. So, we define the set $\mathcal{U}_s$ of possible actions in state s by $\mathcal{U}_s = \{\mathcal{F}_s\}$ meaning that $\mathcal{U}_s$ is the set of all possible subsets of $\mathcal{F}_s$. Each element of $\mathcal{U}_s$ corresponds to a feasible subset of activated fragile edges in s.
- Suppose that we are in state s where there are $a_s \geq 1$ fixed out-going edges and $b_s(u) \geq 0$ fragile out-going edges for a chosen action $u \in \mathcal{U}_s$. Then, for any state s' that is accessible from s using a fixed or an activated fragile link, the transition probabilities are given by

$$\mathcal{P}_{s,s'}^u = \frac{1}{a_s + b_s(u)}.$$

  If there is no such edge between s and s', then $\mathcal{P}_{s,s'}^u = 0$.
- The action of going from state s to state s' using action u results in an immediate-cost of 1, except if we are already at the terminal state τ where we get a cost of 0 because we do not want the total cost to be influenced. Therefore, the immediate-cost function is

$$\mathcal{C}_{s,s'}^u = \begin{cases} 0 & \text{if } s = v_t \\ 1 & \text{otherwise,} \end{cases}$$

  for all states s, s' and action u.

In this formulation of the problem, a policy μ is deterministic and defines a possible configuration of the fragile links for the whole problem. Note that performing an action can be seen as performing a step in the random walk. So, the expected cumulative cost of starting from state v_s until we reach state v_t is equal to the expected number of steps between two visits of node v in our original random walk. This number of steps is exactly the expected first return time (and also the first hitting time) of node v. Therefore, the v_s component of the cost-to-go function $J^\mu_{v_s}$ corresponds exactly to this expected first hitting time of v_s under policy μ and

$$\pi^\mu_v = \frac{1}{\varphi^\mu_{v_s}} = \frac{1}{J^\mu_{v_s}},$$

where μ designates the particular configuration of the graph. Thus, the maximal PageRank of v is given by $1/J^*_{v_s}$ which can be obtained by solving this new SSP problem.

Unfortunately, it is known that MDP's can be solved in a number of step which is polynomial in the number of states N, the maximum number of actions per state M, and the binary input size L. The number of states is simply $n+1$, where n is the number of vertices in the original graph, so it is polynomial in the problem's size. However, the maximum number of actions per state does not have a polynomial size : it is exponential in the maximum number of fragile edges leaving a node, which is $O(n)$. More precisely, if we have maximum m_f fragile links leaving a node, we have 2^{m_f} possible actions to take in the corresponding state, so $M = 2^{m_f} = O(2^n)$. While this is obviously not good enough, one can observe that if the maximum number of fragile links leaving a node was bounded by a constant k, then we could have a solution that is polynomial in the size n of the problem (since the maximum number of actions per state becomes a constant 2^k). This motivates the next refined solution in which we are going to reduce the maximum number of actions per state to two, while only slightly increasing the number of states.

2.2.2 A refined SSP formulation

We now refine the above formulation such that the number of actions per state is at most two. In order to do that, we introduce an auxiliary state f_{ij} if node i was linked to node j by a fragile edge in the original graph. For each fragile edge (i,j), the idea is to move the decision whether we activate it or not, from state[1] i to the auxiliary state f_{ij}. This idea is illustrated on Figure 2.2.

Assuming that node i has a_i fixed out-going edges and b_i fragile out-going edges (independently from their activation state), we define the refined action space, transition probability function and cost function as follows :

- In state i there is only one available action u that brings us to state j with probability

$$\mathcal{P}^u_{i,j} = \begin{cases} \frac{1}{a_i+b_i} & \text{if } (i,j) \in \mathcal{E}\backslash\mathcal{F} \text{ or if } j = f_{il}, \\ 0 & \text{otherwise} \end{cases}$$

  for all states i, j, l, where $\mathcal{E}\backslash\mathcal{F}$ is the set of fixed edges of the original graph. This means that we go from state i to state j with always the same probability, no matter the configuration of the fragile edges.

[1] We sometimes mix up notation for states and nodes since they refer to equivalent objects. So by state i, we here mean the state corresponding to node i in the SSP instance. Usually, we will use the term "node" when referring to the graph problem and its associated concepts and the term "state" when referring to the SSP problem and its associated concepts.

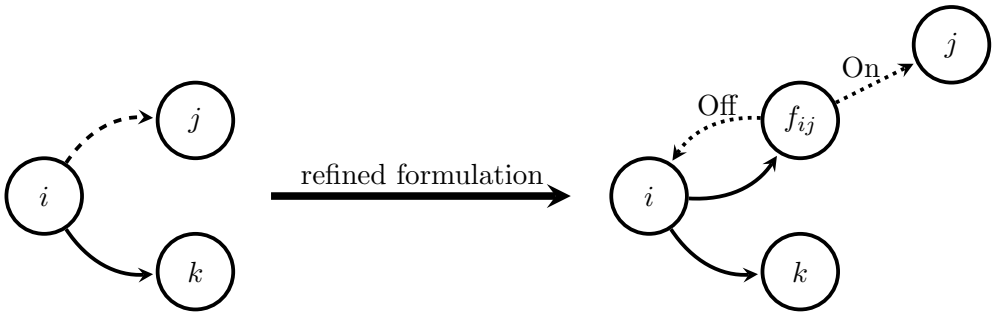

Figure 2.2: An example showing how a new auxiliary state f_{ij} is introduced to replace the fragile edge (i, j). No more action is needed in state i and the action whether we activate the fragile link or not is moved to the auxiliary state f_{ij}.

- In state f_{ij} we have two available actions : "On" or "Off". If we choose "On", we go to state j with probability one. Otherwise, we go back to state i with probability one. This is the key modification towards the maximum-two-actions-per-state formulation we are looking for. It transfers the 2^{b_i} possible decisions from state i towards b_i auxiliary 2-decisions states. In addition, [CJB09] shows that, for a given policy μ, this new formulation does not affect the transition probabilities between the original nodes.

- However, it does change the value function J^μ because it is possible to visit deactivated auxiliary states (i.e., auxiliary states for which the chosen action was "Off") several times before visiting another state. Visiting such a deactivated auxiliary state should be a fictive action since it was not possible to go along a deactivated fragile edge in the original problem. Furthermore, going along an activated fragile edge now takes two steps instead of one originally. A simple way to solve these problems is to change the immediate-cost function such that

$$\mathcal{C}^u_{i,j} = \begin{cases} 0 & \text{if } i = v_t \text{ or } j = f_{il} \text{ or } u = \text{"Off"} \\ 1 & \text{otherwise} \end{cases}$$

  for all states i, j, l and action u. That way, we only count the steps of the original random walk. The value function J^μ remains unchanged and it still corresponds to the first hitting times of the states.

With this refined formulation we can again compute $J^*_{v_s}$, where J^* is the minimum expected first hitting time that we can achieve with a suitable policy. Then, the maximum PageRank of node v comes simply from $\pi_v = 1/J^*_{v_s}$. Note that the minimum PageRank can also be found in a straightforward way from this formulation, namely by taking the new immediate-cost function $\hat{\mathcal{C}}$ such that $\hat{\mathcal{C}}^u_{i,j} = -\mathcal{C}^u_{i,j}$ for all states i, j and action u. Then, if the optimal cost-to-go function of this modified SSP is $\hat{J}^*$, the minimum PageRank of node v is given by $1/|\hat{J}^*_{v_s}|$.

This formulation now has $N = n + f + 1$ states, where n is the number of nodes in the original graph and f is the number of fragile edges. Furthermore, the maximum number of actions per state is now $M = 2$, as required. Therefore, we have a proof that the Max-PageRank problem can be solved in (weakly) polynomial time (if we assume (AD) and (AR)) using linear programming, since N and M are polynomial with respect to the problem's size n.

Finally, we can drop the dangling node assumption (AD). There are many ways to deal with dangling nodes, as we mentioned in section 1.1.1. The idea is to choose one of these ways and

to apply it to the problem before the optimization. However, in our case, we can also have problems with nodes that only have fragile out-going links since they are latent dangling nodes. Such nodes are called fragile nodes and can be handled in a similar way as the usual dangling nodes. Details can be found in [CJB09].

2.2.3 Including damping to the problem

Some possible graphs can sometimes be ill-conditioned for the Max-PageRank problem, e.g. they violate the (AR) constraint. This is one reason why it is often useful to add some damping to the problem, in addition to modeling purposes. Here we describe a construction for including damping in our SSP formulation described above.

For recall, damping can be seen as the probability c to restart the random walk at each step, where c is called the damping constant $\in [0, 1]$. The new starting node is chosen according to a stochastic personalization vector z

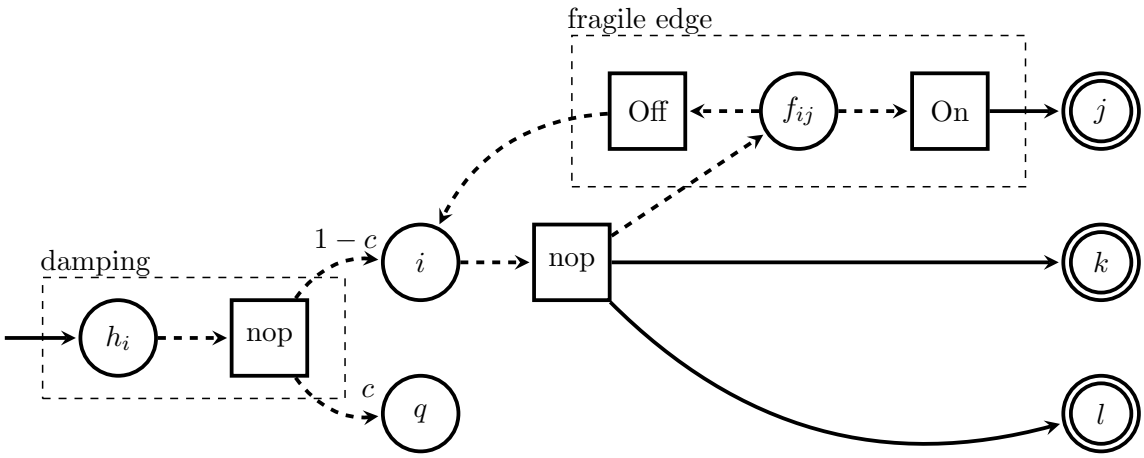

Figure 2.3: An illustration of damping : the structure of a node of the original digraph. Circles represent the states and boxes represent the actions. The double circles hide the same kind of structure but for another node of the graph. The diminutive "nop" stands for "no operation", meaning that the only available action is to do nothing. State q is the teleportation state from which we go to every other state with respect to the personalization vector z. The dashed links help determining zero-cost events.

In order to modify our formulation, we introduce a new global auxiliary state q, called the "teleportation" state, where any state goes with probability c when we chose to restart the random walk. To be able to take the effect of damping into account in each step, we also create an auxiliary state h_i "before" each (non auxiliary) state i. Each action that leads to i in the previous formulation now leads to h_i. In h_i, we only have one available action that leads us to state i with probability $1 - c$ and to state q with probability c, except for the target state v_t for which h_{v_t} leads with probability one to v_t. So, it is in state h_i that we handle the effect of damping on state i. In the teleportation state q, we also have one available action that brings us to any other state h_i with probability $z(i)$ if $i \neq v_s, v_t$ and to state h_{v_t} with probability $z(v)$. All other transition probabilities from q are zero, including the probability to go from q to h_{v_s}.

Regarding the immediate-cost function, it is clear that we should not count the steps when we move through a state h_i and therefore, $\mathcal{C}^u_{h_i,i} = \mathcal{C}^u_{h_i,q} = 0$ where u is the unique action available in h_i (which is to do nothing). On the other hand, we should count the step when we move out

from the teleportation state, so $\mathcal{C}^u_{q,h_i} = 1$ for all states $i \neq v$. An example of this construction is shown on Figure 2.3.

In terms of complexity, this formulation has $N = 2n + f + 2$ states and still maximum $M = 2$ actions per state. This means that we can again solve the SSP problem in polynomial time using linear programming.

2.3 A polynomial time algorithm based on Linear Programming

From our Max-PageRank problem, we constructed an equivalent Markov Decision Process, more precisely a Stochastic Shortest Path problem. We already saw that such problems can be solved in polynomial time using linear programming. Here we present the linear programs that correspond to our problem with and without damping.

First, in the undamped case, the optimal cost-to-go function solves the following linear program in variables x_i and x_{ij} :

$$\begin{aligned} \text{maximize} \quad & \sum_{i\in\mathcal{V}} x_i + \sum_{(i,j)\in\mathcal{F}} x_{ij} \\ \text{subject to} \quad & x_{ij} \leq x_i, \qquad x_{ij} \leq x_j + 1, \\ & x_i \leq \frac{1}{\deg(i)} \left[\sum_{(i,j)\in\mathcal{E}\setminus\mathcal{F}} (x_j + 1) + \sum_{(i,j)\in\mathcal{F}} x_{ij} \right] \end{aligned} \tag{2.1}$$

for all $i \in \mathcal{V}\backslash\{v_t\}$ and $(i,j) \in \mathcal{F}$, where x_i is the cost to go of state i, x_{ij} relates to the auxiliary state of the fragile edge (i,j) and $\deg(i)$ corresponds to the out-degree of node i including both fixed and fragile edges (independently of their configuration). Note that this formulation requires the target node to be split into the starting and termination nodes beforehand. Also note that x_{v_t} is here fixed to zero. Having $O(n+f)$ variables and constraints, we can solve this linear program in $O((n+f)^3 L)$ using the algorithm algorithm from [Gon88].

When damping is part of the input, the problem becomes a little bit more complex :

$$\begin{aligned} \text{maximize} \quad & \sum_{i\in\mathcal{V}} (x_i + \hat{x}_i) + \sum_{(i,j)\in\mathcal{F}} (x_{ij} + \hat{x}_{ij}) \\ \text{subject to} \quad & x_{ij} \leq x_i, \qquad x_{ij} \leq \hat{x}_j + 1, \\ & \hat{x}_i \leq (1-c)x_j + cx_q, \qquad x_q \leq \sum_{i\in\mathcal{V}} \hat{z}_i(\hat{x}_i + 1), \\ & x_i \leq \frac{1}{\deg(i)} \left[\sum_{(i,j)\in\mathcal{E}\setminus\mathcal{F}} (\hat{x}_j + 1) + \sum_{(i,j)\in\mathcal{F}} x_{ij} \right] \end{aligned} \tag{2.2}$$

for all $i \in \mathcal{V}\backslash\{v_t\}$ and $(i,j) \in \mathcal{F}$, where $\hat{z}_i = \mathcal{P}^u_{q,h_i}$, $\hat{x}_i$ corresponds to the cost-to-go of state h_i and x_q is the value of the teleportation state q. All other notation are the same as in (2.1). Again, we fix x_{v_t} and $\hat{x}_{v_t}$ to zero. Note that in this damped case, the costs-to-go corresponding to the original Max-PageRank problem are given by the $\hat{x}_i$ variables, instead of the x_i variables. Indeed, the costs-to-go of the h_i states are the ones that take damping into account.

Again, we have $O(n+f)$ variables and constraints so we can solve this linear program in $O((n+f)^3 L)$, i.e in weakly polynomial time, using the algorithm from [Gon88]. Note that in practice,

we often use fixed values for the damping constant and the personalization vector (typically, $c = 0.1$ or 0.15 and $z = 1/n\ \mathbb{1}$). In addition, the elements of the transition probability function $\mathcal{P}$ are rational with n as the highest possible denominator and the elements of the immediate-cost function are binary, so they all can be represented using $O(\log n)$ bits. Therefore, the binary input size L can be bounded by a polynomial in n and we get a strongly polynomial time algorithm.

2.4 An iterative algorithm based on Policy Iteration

Even though it achieves polynomial time complexity, solving Markov Decision Processes using linear programming can be quite inefficient in practice since it does not take the special structure of such problems into account. Therefore, we often prefer to use the Policy Iteration method instead. As we already saw, this iterative method is designed for MDP's and is really effective in practice, although its theoretical complexity remains quite unknown. Here we propose to adapt the Greedy Policy Iteration method to the above described SSP problem. We will refer to this algorithm by PageRank Iteration.

As for the classical PI method (described in section 1.1.4), our algorithm starts from an arbitrary policy μ_0, e.g. a configuration in which each fragile link is activated. Then, at each iteration k, it performs the three general steps from Algorithm 1 as follows (these steps will be simplified afterwards) :

- It first evaluates the current policy μ_k, i.e., the expected first hitting times of each node, with respect to the configuration of the fragile edges. To do that, it simply solves a linear system as indicated in equation (1.5). This enables us to determine the modification set $\mathcal{T}^{\mu_k}$ as defined in section 1.1.3. This set can be seen as the biggest possible set of fragile edges, such that modifying the state ("On" or "Off") of any subset of these edges would result in a better policy (i.e., a policy that dominates μ_k).
- The second step consists in applying a strategy to select a subset U_k of edges that are in the modification set $\mathcal{T}^{\mu_k}$ and for which we want to *switch* the state ("On" to "Off" or vice versa). In our case, we chose to work with the Greedy Policy Iteration method, so we always choose $U_k = \mathcal{T}^{\mu_k}$. The idea of this strategy is to change everything that seems to have a favorable impact (from the current point of view that depends on the current policy). This avoids us the trouble of having to choose arbitrarily which edges we would like to switch.
- It finally applies the changes in $U_k = \mathcal{T}^{\mu_k}$ by modifying the state of all fragile edges in the set.

This formulation is a strict adaptation of the definition of PI that we gave in section 1.1.4 and it will be useful to our later analysis of the algorithm. Yet, in the case of the Max-PageRank problem, it can be further simplified. Let us define a configuration F_k as the subset of fragile edges that are active in iteration k. This set can be interpreted as the current policy. Assuming for the moment that there is no damping in the problem, the greedy strategy tells us to update the configuration in the following way : a fragile link from node i to node j should be activated if and only if $\varphi_i^{F_k} \geq \varphi_j^{F_k} + 1$, where φ^{F_k} is the first hitting time vector in iteration k corresponding to a transition matrix P_k defined according to the configuration F_k. This follows from Bellman's equation. Intuitively, activating a fragile edge (i, j) is interesting as soon as node j is closer to the target node than i, even after a displacement of cost 1.

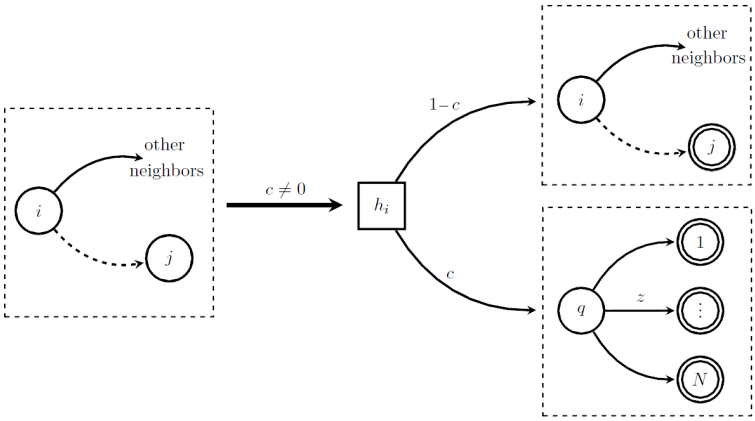

Figure 2.4: When there is no damping, we activate the fragile edge (i, j) if it is better to be in j than in i even at the cost of a displacement. If we add damping, we have to take two situations into account : (1) the "probability $1 - c$" scenario in which we do not restart the random walk and that is similar to the undamped case, and (2) the "probability c" scenario in which we do restart the random walk and that interacts with all the nodes of the graph.

When damping is added to the problem, the above criterion has to be modified. We want to know under which condition a fragile edge (i, j) has to be activated. Let us consider the representation described in section 2.2.3. We know that with probability $1 - c$, damping will not operate and we will go from state h_i to state i, in which the criterion is the same as in the undamped case. On the other hand, we go with probability c to the teleportation state q where a new starting state is chosen according to the personalization vector z (see Figure 2.4). Those two cases put together, we obtain that a fragile edge (i, j) should be activated if and only if

$$\begin{aligned} \varphi_i^{F_k} &\geq (1-c)(\varphi_j^{F_k}+1) + c\left(\sum_{l\in\mathcal{V}} z(l)(\varphi_l^{F_k}+1)\right) \\ &= \varphi_j^{F_k} + (1-c) + c\left(z^T\varphi^{F_k}+1\right) \\ &= \varphi_j^{F_k} + cz^T\varphi^{F_k} + 1. \end{aligned}$$

Note that this criterion with $c = 0$ simply brings us back to the previous criterion without damping. Putting everything together, we finally obtain the PageRank Iteration algorithm (Algorithm 2).

Algorithm 2 PageRank Iteration

Require: A digraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a target node $v \in \mathcal{V}$ and a set of fragile links $\mathcal{F} \subseteq \mathcal{E}$.
Ensure: The optimal configuration of fragile links F_k.

1: $k := 0$.
2: $F_0 = \mathcal{F}$.
3: **repeat**
4: $\quad \varphi^{F_k} := (I - \overline{P}_k)^{-1}\mathbb{1}$.
5: $\quad F_{k+1} := \{(i, j) \in \mathcal{F} : \varphi_i^{F_k} \geq \varphi_j^{F_k} + cz^T\varphi^{F_k} + 1\}$.
6: $\quad k \leftarrow k + 1$.
7: **until** $F_{k-1} = F_k$
8: **return** F_k.

Also notice that from the optimal configuration F_k, it is straightforward to recover the optimal policy μ^* and therefore, it is also easy to find the first hitting times from (1.5).

In the subsequent chapters, we address both the experimental and the theoretical complexity of PageRank Iteration.

Chapter 3

Experimental results

This chapter aims at evaluating the performances of the three algorithms proposed in chapter 2, namely Interval Matrices (IM), Linear Programming (LP) and especially PageRank Iteration (PRI). We start by comparing these methods on their respective execution times. Then, we study the precision of IM and end up with a more precise analysis of PRI.

3.1 Experimental comparison of the algorithms

In this first section, our goal is to compare the performance of the three algorithms proposed in chapter 2 in terms of their execution times. We have therefore designed a number of tests on many different graphs. We considered graph instances characterized by the three following parameters :

- the number of nodes n ;
- the number of fragile edges f ;
- the type of graph, i.e., its general structure.

Since we want to be able to generate graphs automatically from those parameters only, the generation has to be done in a random way. Different methods to generate random graphs will be proposed later. Note that, our graph instances being generated randomly, we do not try every possible case and so, we do not a priori catch all the possibly observable behaviors of the algorithms. To observe a particular phenomenon, instances should be designed independently.

The construction of a test graph is done according to the three following steps :

- we choose n and construct the node set $\mathcal{V} = \{1, ..., n\}$ of the graph ;
- we choose the type of graph and use the corresponding construction method to generate the edge set $\mathcal{E}$ of the graph ;
- we choose $f \leq m$ and select randomly the f among m edges that will be fragile, and as such define the set $\mathcal{F}$ of fragile edges.

Regarding the type of (random) graph, we considered four types of structures. The main reason why we did not consider only one type is to be able to observe the prospective impact of the graph's structure on the performances of our algorithms. We also did this in reaction to the tests made in [IT09] to test the algorithm described in section 2.1. In their work, the

authors limited their study to one particular graph construction type which we do not think is appropriate since it is arbitrary and it cannot be generalized for any size of graph. Here, we consider the four following types of random graphs, among which the one used by [IT09] :

- **Erdös-Renyi Graphs**. To generate these graphs, we use the following rule : we first fix an arbitrary probability p. Then for each couple of nodes i, j, we put the edge (i, j) in $\mathcal{E}$ with probability p. Note that we can choose p in order to get close to a given number of edges m (at least in terms of probabilities).
- **Ishii and Tempo-like Graphs**. For those graphs, we tried to follow the construction used in [IT09]. Since it cannot be generalized for any size of graphs, we only considered the instance containing $n = 150$ nodes. Here is the way it is constructed : we give the first 5 nodes a high incoming degree (between 75 and 135), a medium high degree for the next 100 nodes (between 5 and 25 incoming edges), and an incoming degree of 2 for the last 45 nodes. Note that in their tests, Ishii and Tempo constrained the total number of edges to $m = 2559$.
- **Power Law Graphs**. Those graphs are meant to be quite similar to real web graphs. The way of constructing them is explained for example in [MST04]. The idea is roughly that the in-degrees of the nodes follow a power law. That way, some high degree nodes can appear while other nodes can have very low degree. In our tests, we usually bound the minimum degree of the nodes in order to obtain better conditioned graphs (i.e., graphs that are not too far from complying with the Reachability Assumption (AR) explained in section 2.2).
- **Web Graphs** (graphs generated from the Web itself). Those are generated from the web itself using a *surfer* which starts from a given web-site and then follows links in a random way[1]. Even though it is based on real data, this approach has some drawbacks. Since it is costly to generate large graphs with that method, we often end up with clusters of web pages that come from the same web site and that all link to one another. Therefore, it is difficult to obtain a representative part of the real web structure. Another consequence is that we never get similar data when starting from completely different web pages. So, all generated graphs are quite different which can lead to big variations in the tests results.

Figure 3.1 illustrates the structure of each of the above describes types of graphs.

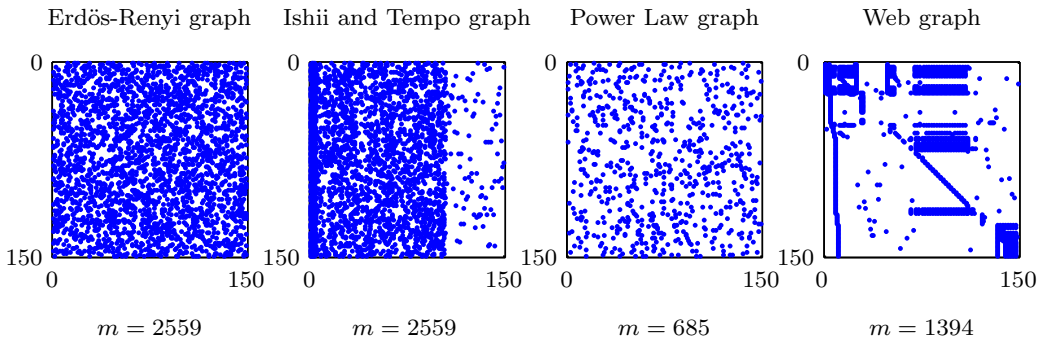

Figure 3.1: An example for the adjacency matrix of the four types of random graphs we will consider.

In the present tests, we only measure the execution times of the algorithms. For better reliability, we do each test on 10 different graphs generated with the same parameters. We then take

[1]Such a surfer can be found at *http://www.mathworks.com/moler/ncm/surfer.m*.

the average result on those 10 tests. For each generated graph, we choose the target node v to be the node of index 1, without loss of generality since the graphs are random. Note that we here concentrate on the Max-PageRank problem, performances for the Min-PageRank problem being similar.

Under those conditions, we will make the three following comparison tests between the three algorithms :

- a comparison on size-varying Power Law graphs, with $f = n$ in each test and fixed positive damping constant ;
- a comparison on all four types of fixed size ($n = 150$) graphs where the number of fragile edges f varies and where the damping constant is fixed ;
- a comparison with and without damping on fixed size ($n = 150$) Erdös-Renyi graphs where the number of fragile edges f varies.

The first test studies the variation of the execution times in function of the size of the problem. The second one has two purposes : studying the performances of the algorithms with respect to the number of fragile edges and observing the influence of the graph's structure. The last test essentially compares the situations with and without damping.

3.1.1 Influence of the graph's size

We would like to know which method is the most efficient for a given problem's size. Therefore, we compared them to one another on Power Law instances for sizes going from $n = 10$ to $n = 200$. We choose the number of fragile edges to vary in the same way as n in each test. Resulting execution times are to be seen on Figure 3.2.

The reason why we used Power Law graphs here is because they are closest to the kinds of graphs we might encounter in practice with the Max-PageRank problem, which is essentially designed for web data. So, we will favor the use of these graphs in our tests whenever possible. However, note that these kinds of graphs are usually bad choices when working on undamped problems because they hardly respect the required Reachability Assumption (AR).

Whatever the size of the problem (at least in the considered range), it seems on the figure that the PageRank Iteration method performs far better than the two other algorithms. Even though it is guaranteed to run in polynomial time, Linear Programming is the least efficient of the three methods. This confirms that it is only useful for theoretical purposes. Interval Matrices method lies somewhere in between but remains an approximation.

One can note that the curves are quite smooth and regular. Therefore, we can assume that the same kind of (relative) performances will be observed for large graphs as well. Furthermore, we can see that the theoretically predicted complexities of the LP and IM methods (the thin black lines on the figure) seem to fit almost perfectly to the curves. Yet, let's not forget that these curves are built on the average of several similar randomized tests and hence are not representative of all possible situation that can happen in practice.

3.1.2 Influence of the number of fragile edges and of the graph's structure

With this test, we try to reproduce the observation made by Ishii and Tempo in their work in which they study the influence of the number of fragile edges on the complexity. This time however, we include the LP and the PRI methods in the analysis. We also push the analysis

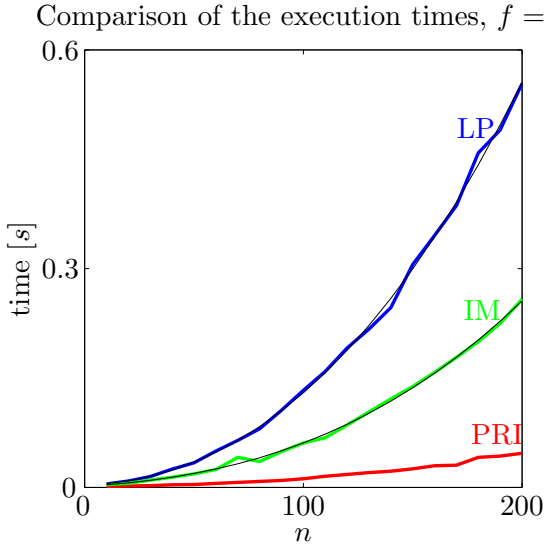

Figure 3.2: Clearly, the PageRank Iteration method seems to be the most efficient of the three in practice. The thin black curves along the blue LP and the green IM curves correspond to the ones predicted theoretically. Recall that the IM method only provides an approximation of the solution.

further by considering the four above described types of random graphs and a larger range of f-values going from 0 to 200. Figure 3.3 presents the obtained results.

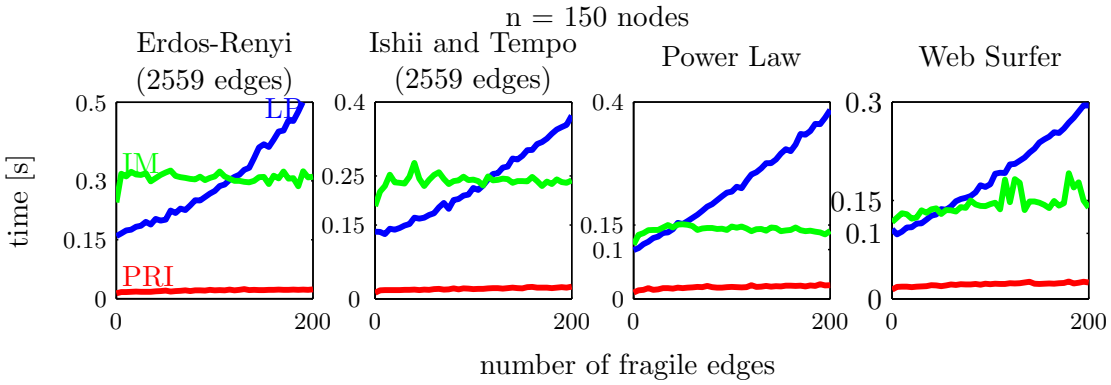

Figure 3.3: The four types of graphs seem to produce similar comparisons between the methods. As predicted from the theory, IM seems not to depend on the number of fragile edges. Note that for Power Law and surfer graphs, the number of edges varies from test to test, which is why it is not indicated on the figure.

As predicted from the theory, the execution time of IM seems not to depend on the number of fragile edges, at least for sufficiently high values of f. For lower values of f, it is likely that the construction of the polytope described in section 2.1 becomes more costly than solving the linear program. This could explain the small dependence in f of the complexity observed in that case. We also can see that for low values of f, the LP method behaves better than IM.

Furthermore, the influence of the type of graphs seems to be negligible, although the execution times seem to be slightly shorter for Power Law and Web graphs. This is probably due to the fact that these two types of graphs have fewer edges than the two others in our tests.

3.1.3 Influence of damping

The main goal of this last test is to compare an identical situation with and without damping. The test is done with Erdös-Renyi graphs since it is easier to generate graphs that respect the Reachability Assumption (AR) that way. Figure 3.4 makes the desired comparison. As we can see, the situation is pretty much the same with or without damping, at least for the chosen value of c.

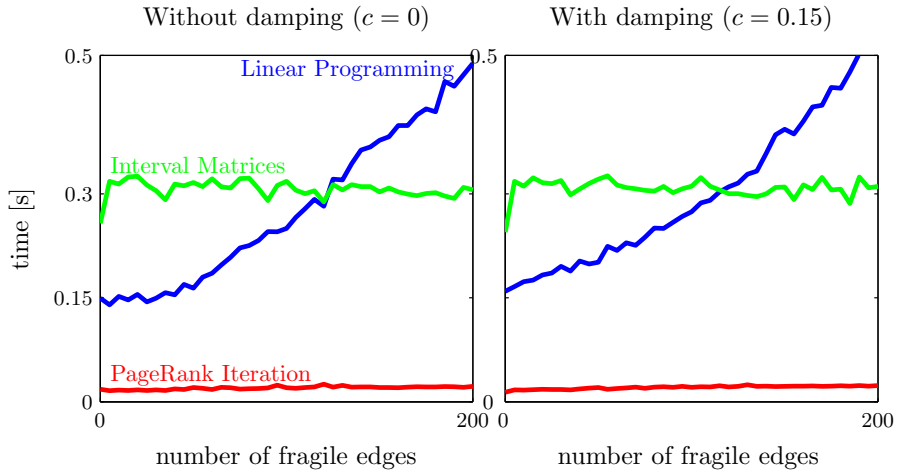

Figure 3.4: The average behavior of the methods does not change when we add damping to the problem.

3.2 Degree of approximation of IM

In this section, we briefly study the quality of the solution given by Ishii and Tempo's algorithm, IM. More precisely, we check their statement according to which the error of the approximation grows linearly with the number of fragile edges. We also extend their test to the four types of graphs described in the previous section.

To evaluate this error, we used the PageRank values obtained from PRI as a reference for the solution. We then measure the relative error committed by IM from

$$e_{\text{rel}} = \frac{\|\pi^{(\text{IM})} - \pi^{(\text{PRI})}\|}{\|\pi^{(\text{PRI})}\|},$$

where $\|\cdot\|$ denotes the euclidean norm. The result for varying values of f and with the four types of graphs is represented on Figure 3.5.

On the figure, we can indeed verify the linear dependence of e_{rel} in the number of fragile edges f. It is also possible to see that this relative error does not depend on the size n of the manipulated graphs.

Somewhat expectedly, notice that the best behaving graph structure in terms of relative errors is the one proposed by Ishii and Tempo. On the contrary, the web graph structure has quite jumpy relative errors, and the IM method seems to be unsuitable in that case.

3.3 Experimental performances of PRI

We now concentrate on the performances of the PRI algorithm and try to display its average behavior. This time, we expand our evaluation scope to three criteria :

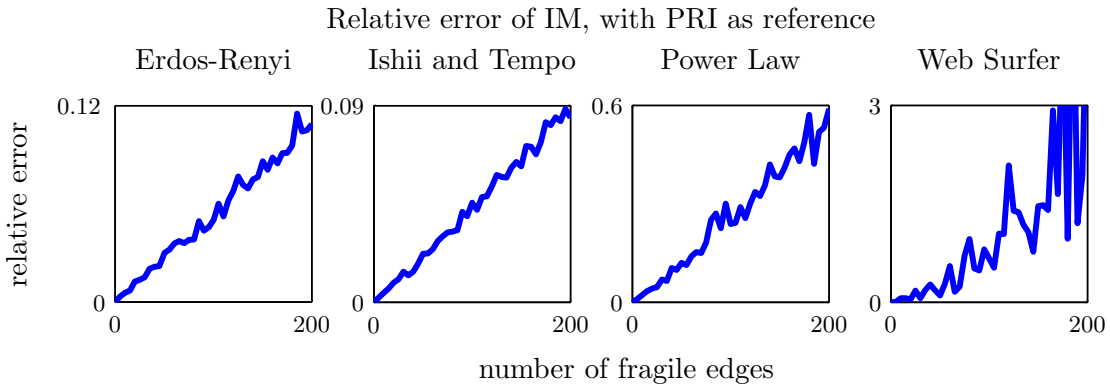

Figure 3.5: Relative error committed by the IM algorithm grows linearly with the number of fragile edges. The best behaving graph structure is the one proposed by [IT09].

- the execution time ;
- the mean number of iterations ;
- the maximum number of iterations.

The number of iterations of PRI is essential. Indeed, we already saw in section 1.2.5 that one iteration of PRI runs in polynomial time. The only remaining question is the number of iterations needed to converge for which we do not have a polynomial upper bound yet.

Two tests are planned to figure out (1) the dependence on the problem's size n and (2) the dependence on the number of fragile edges f. Both tests will be executed on Power Law random graphs with damping constant $c = 0.15$.

3.3.1 Influence of the problem's size

Figure 3.6 presents the performances of PRI with respect to the number of nodes $n \in [10, 500]$, with f proportional to n such that $f = n$.

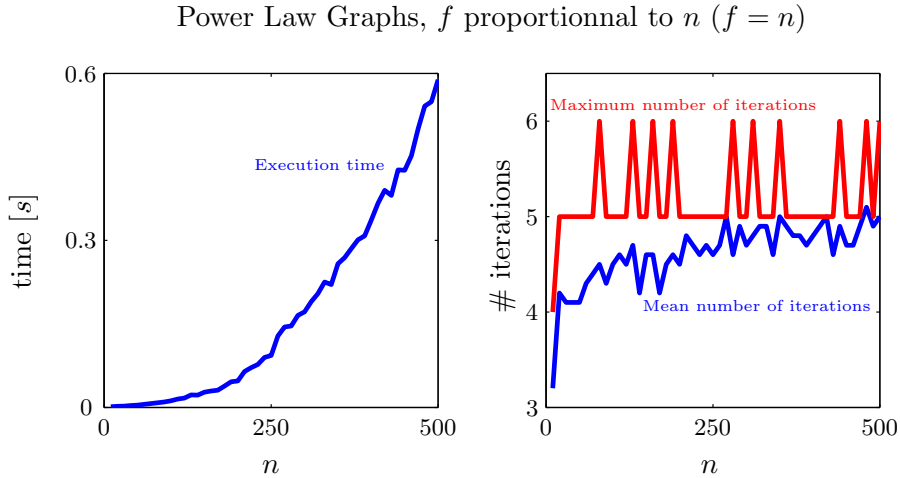

Figure 3.6: The execution time grows according to a polynomial in n. Meanwhile, the number of iterations grows less than linearly.

As we can see, the complexity dependence on n looks far from the existing exponential bounds we have in theory. On the contrary, the number of iterations seems to grow linearly, logarithmically or even maybe constantly with n.

3.3.2 Influence of the number of fragile edges

Figure 3.6 presents the performances of PRI with respect to the number of fragile edges $f \in [0, 500]$.

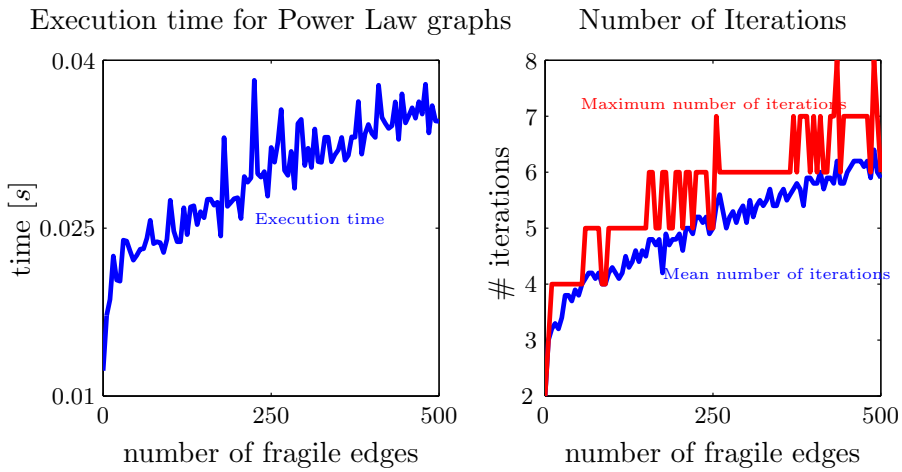

Figure 3.7: Both the execution time and the number of iterations grow when f grows, but less than linearly.

Again, even though both the execution time and the number of iterations increase when f increases, the dependence is nothing like exponential. If we kept increasing f, we would actually observe a peak for the execution time (and the number of iterations) followed by a decrease as f approaches the total number of edges m.

The conclusion we can draw from these tests is that the PageRank Iteration algorithm behaves well in practice and seems to achieve polynomial complexity. However, this used to be said also about the simplex for linear programs. The question whether the theoretical bounds on complexity can be reconciled with the experimental results is addressed in the next chapter.

Chapter 4

Results on PageRank Iteration

This chapter is devoted to the presentation of our research results. We start by describing our main result which bounds the complexity of the damped version of PRI by a polynomial in the problem's input. We then present a number of results which, for most of them, have proven useful. Next, using these results, we propose some approaches to tackle the problem of bounding the complexity of PRI when damping is absent from the problem. Finally, in a last section, we develop one of these approaches which roughly suggests to formulate an undamped problem as an equivalent damped problem.

4.1 Theoretical polynomial upper bound on damped PRI complexity

In this section, we present a first weakly polynomial bound (i.e., polynomial in the problem's size and the binary input size) on the complexity of the PageRank Iteration algorithm in case of a fixed non-zero damping constant. As we will see, this bound essentially derives from Tseng's work. It is our main contribution. We will also compare our bound to the exponential one proposed by [Fea10] and use this comparison to draw some conclusions about Stochastic Shortest Path problems.

4.1.1 Towards the polynomial upper bound

Here is a reminder of the necessary notions that lead to our result. First remember that we are working with a Markov Decision Process instance defined by a quadruple $(\mathcal{S},\mathcal{U},\mathcal{P},\mathcal{C})$ as explained in section 1.1.3. We initially assume that we are in a discounted case with discount factor $0 \leq \gamma < 1$. We saw in section 1.1.3 that in that case, the Bellman operator T defined by (1.3) is a contraction with respect to the L^∞-norm such that

$$\|T(x) - x^*\| \leq \gamma \|x - x^*\|. \tag{4.1}$$

Consequently, the Value Iteration (VI) algorithm converges to the optimal solution in a maximum number of iterations bounded by

$$O\left(\frac{N \log(N\delta)}{1-\gamma}\right) \tag{4.2}$$

where N is the number of states and δ is the accuracy of the problem, meaning that δ is the smallest positive integer for which $\delta\gamma$ and $\delta\mathcal{P}^u_{s,s'}$ are integers and $\mathcal{C}^u_{s,s'} \leq \delta$ for all states $s, s' \in \mathcal{S}$ and actions $u \in \mathcal{U}$ [Tse90]. Note that $N \log(N\delta)$ is polynomial in the binary size of the input, L, and therefore, the above bound is weakly polynomial. Furthermore, we know [Put94] that Policy Iteration (PI) converges at least as fast as VI so the same bound can be applied to it.

Let us now assume that we are in the undiscounted case ($\gamma = 1$), and more precisely in the case of a Stochastic Shortest Path problem corresponding to a Max-PageRank problem (see section 1.1.5). In that case, we know that the following contraction relation holds :

$$\|T(x) - x^*\|^\omega \leq \alpha \|x - x^*\|^\omega, \tag{4.3}$$

where

$$\alpha \triangleq \frac{1 - \eta^{2r-1}}{1 - \eta^{2r}} < 1 \tag{4.4}$$

is the contraction ratio, $\|\cdot\|^\omega$ is a scaled L^∞-norm, η is the smallest non-zero transition probability and r is the diameter of the graph associated with the SSP problem [Tse90]. More precisely, we define $d^\mu_{s,s'}$ as the minimum number of states that have to be visited before having a chance to reach state s' when starting from state s and following policy μ (it can be seen as the minimum distance from s to s' when choosing policy μ). Then the diameter is simply given by

$$r \triangleq \max_{s,s',\mu} d^\mu_{s,s'}.$$

Having that, we can apply the bound given by (4.2) for the undiscounted case as well but with the fixed contraction ratio α instead of the discount factor γ. Furthermore, Tseng shows that $1 - \alpha \geq \eta^{2r}$ and therefore, (4.2) becomes

$$O(N \log(N\delta)\eta^{-2r}). \tag{4.5}$$

Let us now try to apply these results to the PageRank Iteration (PRI) algorithm. As we already saw, PRI is built as a particular case of PI. The only noticeable difference is that it does not check every possible action in each state : it optimizes each fragile link separately. However, as the refined SSP formulation of section 2.2.2 demonstrates, we are allowed to do so since we are able to isolate the decisions on each fragile edge in an independent 2-choices auxiliary state. Hence, we can apply the same bound given by (4.5) on the number of iterations of PRI.

Our solution is driven by using the particular values of η and r that we have in case of a Max-PageRank problem with fixed non-zero damping constant c. It is indeed easy to see that in case of damping, each node is directly connected to every other with non-zero probability, using the formulation of section 2.2.3. So, independently from the policy μ, the longest distance $d^\mu_{s,s'}$ between two arbitrary states s and s' is at most 2 : (1) one transition to go from s to the auxiliary teleportation state q and (2) one transition to go from q to s'. That way, we obtain a diameter $r = 2$ and a smallest non-zero probability $\eta \geq \min\{c, 1/n\}$. Therefore, since we also have $N = n + 1$, we can bound the number of iterations of PRI by $O(n \log(n\delta)(1/c + n)^4)$.

In addition, we can further improve this result by discarding the auxiliary teleportation state q (and the h_s states) and modifying the transition probabilities accordingly. Doing so, the influence of damping is directly contained in the transitions between states. More precisely, suppose that we start in the simple or the refined formulation (sections 2.2.1 and 2.2.2). Let us analyze the transition from a state s to another state s' due to damping. Previously, when damping was operating, we went to the teleportation state q with probability c and then to state s' with probability $z(s')$, the whole displacement having a cost of one. The idea is that

we could have gone directly from s to s' with probability $cz(s')$. This would eventually give us a new transition probability function $\hat{\mathcal{P}}$ such that

$$\hat{\mathcal{P}}^u_{s,s'} = (1-c)\mathcal{P}^u_{s,s'} + cz(s')$$

for all states s, s' and action u, where $\mathcal{P}$ is the transition probability function of the simple or the refined formulation. By doing this, we give a non-zero probability for any transition and therefore, the diameter becomes $r = 1$. Assuming that the personalization vector is given by $z = (1/n)\mathbb{1}$, the smallest transition probability is now $\eta = c/n$ (except for the trivial case of a complete graph with no fragile edges for which $\eta = 1/n$). Altogether, we can bound the number of iterations of PRI by :

$$O\left(n \log(n\delta) \left(\frac{n}{c}\right)^2\right). \tag{4.6}$$

We believe this result to be a significant contribution to solve the Max-PageRank problem. Since c is fixed beforehand, we could further simplify this expression. However, we will keep this dependence, because we will need it later.

4.1.2 Related questions

Having formulated this polynomial upper bound for solving Max-PageRank problems, it is worth investigating if a general SSP problem could be reduced to this new class of problems. It is unfortunately not possible since [Fea10] found an exponential lower bound for general SSP problems. However, Fearnley's result is based on a counterexample which makes use of exponentially small transition probabilities (namely $\frac{1}{(10n+4)2^n}$), and therefore, $\eta = O(1/(n2^n))$. It is thus coherent with Tseng's result which would give an exponential upper bound in that case. Yet, the question remains open if we impose the transition probabilities to be polynomial in n, as it would invalidate Fearnley's example. More research could therefore potentially lead to polynomial complexity of SSP under this condition.

Another interesting question would be to try to adapt the result from [Ye10], i.e., the strongly polynomial upper bound on discounted PI given by (1.6), to PRI which is undiscounted. For instance, would it be possible to replace the discount factor γ by the contraction factor α defined by (4.4) like we previously did ? To validate this, we see two possible approaches :

- We could show that Ye's bound is a consequence of the fact that the Bellman operator T is a contraction. In other words, we could show that if T is a contraction with ratio α, then Ye's bound can be applied with $\gamma = \alpha$. The above result was indeed a direct consequence of that property applied to Tseng's bound for discounted problems.

- Another way of showing the same thing would be to show that if T is a contraction of ratio α, then we can formulate an equivalent discounted MDP problem such that $\gamma = \alpha$. This discounted problem could then be solved using PI with a number of iterations bounded by (1.6). This would however be surprising as it would imply that all SSP problems could be formulated as discounted MDP problems...

All in all, it is very likely that the above weakly polynomial bound on the number of iterations of PRI can be further improved since it does not yet reconcile with the experimental results obtained in section 3.3.

Finally, a last interesting question would be to see whether the undamped Max-PageRank problem could be somehow reduced to the damped Max-PageRank. This would enable us to use the same complexity results in the undamped case. This question will be our main focus from now on. We will address it after describing a number of relevant results on SSP and Max-PageRank.

4.2 Miscellaneous results

In this section, we give a number of results that essentially emerged from our research. Most of them could be a key to new discoveries. Some of them will also be used in the next sections during our study of the undamped case of PRI. We here start with several general properties on Markov Decision Processes. Then, we concentrate on the Max-PageRank problem by giving some bounds on the first hitting times of the nodes.

4.2.1 Properties about MDP's

Let us start with several interesting properties about MDP's in general. These will be useful later in this chapter and could lead to new discoveries regarding MDP's, SSPs and in particular the Max-PageRank problem.

First an interesting property was proposed by [MS99]. To formulate it, Let us remind the notion of modification set $\mathcal{T}^\mu$ of a policy μ which is the set of all modifications of the policy μ that independently result in its improvement. More precisely, $\mathcal{T}^\mu$ is the set of all state-action pairs (s, u) such that replacing $\mu(s)$ by u gives a policy that dominates μ. We also define the set $\texttt{states}(\mathcal{T}^\mu)$ of the states at which an improvement is possible, i.e., $s \in \texttt{states}(\mathcal{T}^\mu) \Leftrightarrow \exists u \in \mathcal{U}_s$ such that $(s, u) \in \mathcal{T}^\mu$. Also, let $\mu_0, \mu_1, ..., \mu_K = \mu^*$ be the sequence of policies tried by the PI algorithm, μ_k being the policy evaluated at iteration k. Then we have the following proposition :

Proposition 4.1. *There are no integers, $i, j, i < j$, such that $\texttt{states}(\mathcal{T}^{\mu_i}) \subseteq \texttt{states}(\mathcal{T}^{\mu_j})$ [MS99].*

Roughly, the idea of this result is that the same subset of modifications cannot be considered twice by PI. The proof can be found in the work of [MS99]. Note that the authors already made use of this property to obtain the result saying that that PI converges in at most $O(M^N/N)$ iterations, where N is the number of states of the MDP and M is the maximum number of actions per state. However, the result did not reveal all its secrets yet and that's why we include it here. We will also apply it with the Max-PageRank problem later on.

We now give two properties that will lead to an important result for our analysis. These properties are quite straightforward but they express interesting behaviors of MDP's.

Property 4.2. *Let μ_1 and μ_2 be two different policies. If $J_s^{\mu_1} > J_s^{\mu_2}$ for some state s, then $J_i^{\mu_1} \geq J_i^{\mu_2}$ for all states i, where J^μ is the value function under policy μ.*

This property expresses that a sufficient condition for a policy to dominate another is to give strictly better value in at least one node. This fact is a direct consequence of Bellman's equation. It also has the following consequence :

Corollary 4.3. *It is possible to define a partial order on the policies such that $\mu_j \succeq \mu_i$ iff $i < j$.*

Furthermore, using a lemma from [MS99], we also obtain the following property :

Property 4.4. *If several changes $U \subseteq \mathcal{T}^{\mu_1}$ to a given policy μ_1 give a strictly better (resp. worse) policy $\mu_2 \succ \mu_1$ (resp. $\mu_2 \prec \mu_1$), then a subset $\tilde{U} \subseteq U$ of these changes also gives a better (resp. worse) policy $\tilde{\mu}_2 \succeq \mu_1$ (resp. $\tilde{\mu}_2 \preceq \mu_1$).*

Together, these two properties lead to the following result :

Proposition 4.5. *It is possible to define a partial order $\mu_1 \preceq \mu_2 \preceq ... \preceq \mu_K$ on the policies such that for any consecutive policies μ_k and μ_{k+1}, there is exactly one state s such that $\mu_k(s) \neq \mu_{k+1}(s)$.*

Proof. We know from Corollary 4.3 that we may order the policies in such a way that $\mu_k \preceq \mu_{k+1}$ for all k. Suppose that μ_k differs from μ_{k+1} in $a > 1$ states.

- Then we are able to go from policy μ_k to policy μ_{k+1} applying a independent changes to μ_k. Let us denote the set of changes by $U = \{(s_1, u_1), ..., (s_a, u_a)\}$. From Property 4.4, we know that by applying a subset of the changes in U, we will get a better policy. So, Let us only apply the first change in U to μ_k. Doing this, we obtain a new policy $\tilde{\mu}_k$ that dominates μ_k and that is different from it in exactly one state.
- We are also able to go from policy μ_{k+1} to policy μ_k applying the a inverse independent changes in U to μ_{k+1}. From Property 4.4, suppose that we apply the $a-1$ last inverse changes to μ_{k+1}, then we again obtain policy $\tilde{\mu}_k$ which is this time dominated by μ_{k+1} such that $\tilde{\mu}_k \preceq \mu_{k+1}$.

Using this procedure, we have generated a new policy $\tilde{\mu}_k$ that is positioned between μ_k and μ_{k+1} in the initially defined order such that $\mu_k \preceq \tilde{\mu}_k \preceq \mu_{k+1}$. If the domination relation between these three policies is strict, then μ_k and μ_{k+1} are not consecutive which is absurd. However, if the domination relation was not strict, we simply can replace μ_{k+1} by $\tilde{\mu}_k$ since it respects the desired order between policies and it differs from μ_k in only one state. The same argument can be applied iteratively if necessary. Therefore, it is always possible to achieve the desired order. □

This proposition will be applied to the Max-PageRank problem later in this chapter.

4.2.2 Bounds on the first hitting times

We now concentrate specifically on the Max-PageRank problem. In this section, we give two different bounds on the expected first hitting times that the states can have. The first one is essentially useful in case of damping while the second one is based on the worst case for both the damped and undamped case. Note that in this section, we assume that the personalization vector $z = (1/n)\mathbb{1}$.

A bound in case of damping

First of all, Let us remember from section 1.1.6 that the first hitting times are given by

$$\varphi = (I - \overline{G})^{-1}\mathbb{1}, \tag{4.7}$$

where $\overline{G}$ is obtained from the Google matrix G by setting the v^{th} column to zero. Moreover, since G is stochastic and all its elements are strictly positive (for $c > 0$), $I - G$ is strictly diagonally dominant and therefore, it is invertible. We can thus write the following converging development :

$$(I - \overline{G})^{-1} = I + \overline{G} + \overline{G}^2 + ... = \sum_{k=0}^{\infty} \overline{G}^k. \tag{4.8}$$

Furthermore, the next proposition bounds the powers of G in a convenient way.

Proposition 4.6. *For all $i \in \mathcal{V}, j \in \mathcal{V}\backslash\{v\}$, and for all $k \geq 1$, we have*

$$\frac{c}{n}\left(c - \frac{c}{n}\right)^{k-1} \;\leq\; [\overline{G}^k]_{i,j} \;\leq\; \left(1 - (n-1)\frac{c}{n}\right)\left(1 - \frac{c}{n}\right)^{k-1}, \tag{4.9}$$

and $[\overline{G}^k]_{i,j} = 0$ if $j = v$.

Proof. Beforehand, it is important to see that the following bounds hold on the values of $\overline{G}$:

$$\frac{c}{n} \leq [\overline{G}]_{ij} \leq (1-c) + \frac{c}{n} = 1 - (n-1)\frac{c}{n} \tag{4.10}$$

for all $j \neq v$, and $[\overline{G}]_{i,v} = 0$ by definition of $\overline{G}$. This comes from the fact that all elements in the Google matrix G are at least c/n and G is stochastic. From the same facts, and since we imposed a column of G to 0 to obtain $\overline{G}$, we also have

$$(n-1)\frac{c}{n} \leq \sum_i [\overline{G}]_{ij} \leq 1 - \frac{c}{n}. \tag{4.11}$$

These bounds are a consequence from the way we constructed $\overline{G}$ and are easily verified. We are now going to prove (4.9) using an inductive strategy. We will proceed in two times : first we will prove the upper bound and then the lower bound in a similar way.

Upper bound. We first prove that the result is true for $k = 1$ and then that it is true for k if it is true for $k - 1$.

- $k = 1$: From equation (4.10), we verify that $[\overline{G}]_{ij} \leq 1 - (n-1)\frac{c}{n}$ if $j \neq v$ and $[\overline{G}]_{iv} = 0$.
- $k - 1 \Rightarrow k$: Our induction hypothesis tells us that $[\overline{G}^{k-1}]_{i,j} \leq \left(1 - (n-1)\frac{c}{n}\right)\left(1 - \frac{c}{n}\right)^{k-2}$ for all $i \in \mathcal{V}, j \in \mathcal{V}\backslash\{v\}$ and that $[\overline{G}^{k-1}]_{iv} = 0$ for all $i \in \mathcal{V}$. We have the following chain of consequences :

$$\begin{aligned}
[\overline{G}^k]_{i,j} &= [\overline{G}]_{i,:} \cdot [\overline{G}^{k-1}]_{:,j} \\
&\leq [\overline{G}]_{i,:} \cdot \left(\max_i \left([\overline{G}^{k-1}]_{i,j}\right) \cdot \mathbb{1}\right) \\
&= \max_i \left([\overline{G}^{k-1}]_{i,j}\right) \cdot \sum_j [\overline{G}]_{i,j} \\
&\leq \begin{cases} \left(1 - (n-1)\frac{c}{n}\right)\left(1 - \frac{c}{n}\right)^{k-2}\left(1 - \frac{c}{n}\right) & \text{if } j \neq v \\ 0 & \text{if } j = v \end{cases} \\
[\overline{G}^k]_{i,j} &\leq \begin{cases} \left(1 - (n-1)\frac{c}{n}\right)\left(1 - \frac{c}{n}\right)^{k-1} & \text{if } j \neq v \\ 0 & \text{if } j = v, \end{cases}
\end{aligned}$$

where ":" designates the whole row or column. The different steps are essentially using the induction hypothesis as well as the bound defined by (4.11).

Lower bound. We use the same kind of reasoning.

- $k = 1$: Again, we use equation (4.10) to verify that $\frac{c}{n} \leq [\overline{G}]_{ij}$ if $j \neq v$ and $[\overline{G}]_{iv} = 0$.
- $k-1 \Rightarrow k$: Our induction hypothesis tells us that $\frac{c}{n}\left(c - \frac{c}{n}\right)^{k-2} \leq [\overline{G}^{k-1}]_{i,j}$ for all $i \in \mathcal{V}, j \in \mathcal{V}\backslash\{v\}$ and that $[\overline{G}^{k-1}]_{iv} = 0$ for all $i \in \mathcal{V}$. We have the following chain of consequences :

$$\begin{aligned}
[\overline{G}^{k}]_{i,j} &= [\overline{G}]_{i,:} \cdot [\overline{G}^{k-1}]_{:,j} \\
&\geq [\overline{G}]_{i,:} \cdot \left(\min_i \left([\overline{G}^{k-1}]_{i,j} \right) \cdot \mathbb{1} \right) \\
&\geq \begin{cases} \left(\frac{c}{n}\right)\left(c - \frac{c}{n}\right)^{k-2}\left(c - \frac{c}{n}\right) & \text{if } j \neq v \\ 0 & \text{if } j = v \end{cases} \\
[\overline{G}^{k}]_{i,j} &\geq \begin{cases} \left(\frac{c}{n}\right)\left(c - \frac{c}{n}\right)^{k-1} & \text{if } j \neq v \\ 0 & \text{if } j = v. \end{cases}
\end{aligned}$$

□

We will now use this result to bound $(I - \overline{G})^{-1}$. Indeed, we can rewrite equation (4.8) as follows :

$$[(I - \overline{G})^{-1}]_{ij} = [I]_{ij} + \sum_{k=1}^{\infty} [\overline{G}^{k}]_{ij}$$

and then, using (4.9) :

$$[I]_{ij} + \frac{c}{n}\sum_{k=0}^{\infty}\left(c - \frac{c}{n}\right)^{k} \leq [(I - \overline{G})^{-1}]_{ij} \leq [I]_{ij} + \left(1 - (n-1)\frac{c}{n}\right)\sum_{k=0}^{\infty}\left(1 - \frac{c}{n}\right)^{k} \tag{4.12}$$

if $j \neq v$. Moreover, it is well known that

$$\sum_{k=0}^{\infty} a^{k} = \frac{1}{1-a} \Leftrightarrow a < 1.$$

Since, $0 \leq c - \frac{c}{n} \leq 1 - \frac{c}{n} < 1$, we can further develop (4.12) and obtain :

$$\frac{c}{n - nc + c} + 1 \leq [(I - \overline{G})^{-1}]_{ii} \leq \frac{n - nc + c}{c} + 1$$

if $i \neq v$, and

$$\frac{c}{n - nc + c} \leq [(I - \overline{G})^{-1}]_{ij} \leq \frac{n - nc + c}{c}$$

if $j \neq v$. We also have that $[(I - \overline{G})^{-1}]_{vv} = 1$ and $[(I - \overline{G})^{-1}]_{iv} = 0$ when $i \neq v$. Finally, substituting all these in (4.7), we obtain the desired bound on the first hitting times :

$$\frac{c}{n - nc + c}(n-1) + 1 \leq \varphi_s^{\mu} \leq \frac{n - nc + c}{c}(n-1) + 1$$

for any state s and policy μ. Notice for example that the bound is tight in the case of unit damping ($c = 1$) since we have $n \leq \varphi_s^{\mu} \leq n$.

A bound based on the worst case

For the second bound, we are going to build a graph for which we will show that it achieves the highest possible first hitting times possible. For simplicity, we assume that we are in the undamped case but the worst case remains exactly the same when we add damping to the problem. We also suppose that we are dealing with simple graphs in which there are no self-loops nor multiple edges. Note that adding such a self-loop always results in an increase of the first hitting times.

In this "worst case"[1] composed of n nodes, we wish to maximize the first hitting time of a node w. Intuitively, we have to ensure that it is "as hard as possible" for w to reach the target node v. For that, it is also necessary that the other nodes also have a first hitting time "as high as possible". This can be achieved by minimizing the probability of going "towards" v.

Concretely, we start by defining an order on the nodes such that $\varphi_v \geq \varphi_1 \geq \varphi_2 \geq ... \geq \varphi_{n-1} \geq 0$. Then, if we want node $n-1$ to have a first hitting time as high as possible, we have to connect it to all the other nodes in $\{1, ..., n-2\}$ that have a (strictly) higher first hitting time. We also should connect it to the target node v and make $n-1$ the only node from which v can be reached directly. A similar, recursive argument holds for the other nodes. So, node k should be linked to all the nodes in $\{1, ..., k-1\}$ that have (strictly) higher first hitting times. It should also be connected to at least one node that has better first hitting time, namely $k+1$ which is the worst node in that category. This construction generates the desired worst case as shown on Figure 4.1 for $n = 5$.

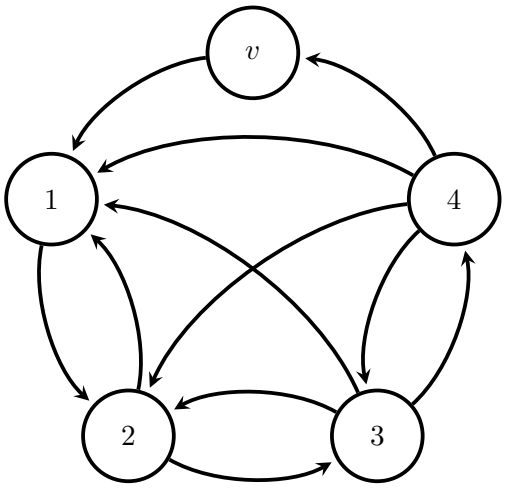

Figure 4.1: An illustration of the worst possible graph in terms of its first hitting times. This solution can also be found using the PRI algorithm. Note that we considered here that there where no self-loops to keep the graph simple. However, since such a loop always increases the first hitting times, adding it would result in an even worse case.

Notice that this problem could be formulated as a particular instance of a Min-PageRank problem (Figure 4.2) in which v is the node for which we want to minimize the PageRank and the dotted edges are the fragile links. Note that we imposed at least one cycle going through each node to make sure the graph is strongly connected. To get the solution, we can for instance use the PRI algorithm. The optimal solution (i.e., the worst possible graph) is given in Figure 4.1 for $n = 5$.

[1] We will refer to that particular case by "worst case" throughout this chapter.

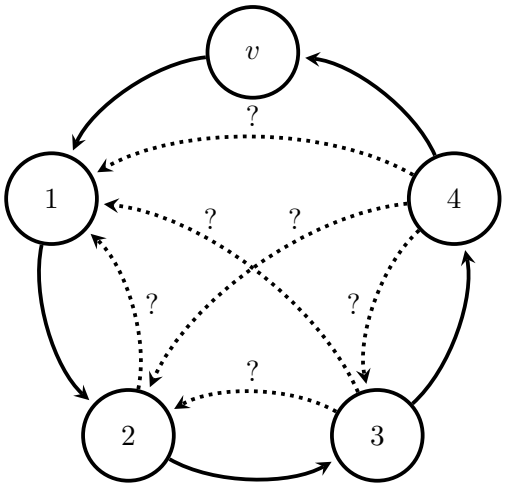

Figure 4.2: Illustration of the worst possible graph with $n = 5$. The only possible way to go from v to v is to go through all other nodes at least once. Besides, all possible node have a certain probability to come back to node 1 which is the farthest node from the destination v. Dotted edges are links for which it is not obvious whether activating them would give a worse first hitting time at node v. They can be seen as fragile edges and their activation state can be determined using PRI on the corresponding Min-PageRank problem

Under this configuration, v is clearly the node with highest first hitting time. Yet, it is not the only node having a high first hitting time. Indeed, from the above recursive argument, it is quite easy to see that it is not possible to find another n-nodes graph for which the k^{th} highest first hitting time is higher than the k^{th} highest first hitting time in this graph, and this for any value of $k \leq n$. Finally, it is not hard to see that the worst case remains exactly the same if we add damping to the problem, since damping has the same restarting effect on any graph. Therefore, it is only the undamped component that dictates whether a graph will have high first hitting times compared to another.

Unfortunately, it is not easy to find a closed form for the first hitting time of v with respect to n. Consequently, we simulated the worst graph for several values of n and observed a neat exponential dependence, as illustrated on Figure 4.3. This shows that it is possible to generate exponentially high first hitting times with respect to the size of the problem.

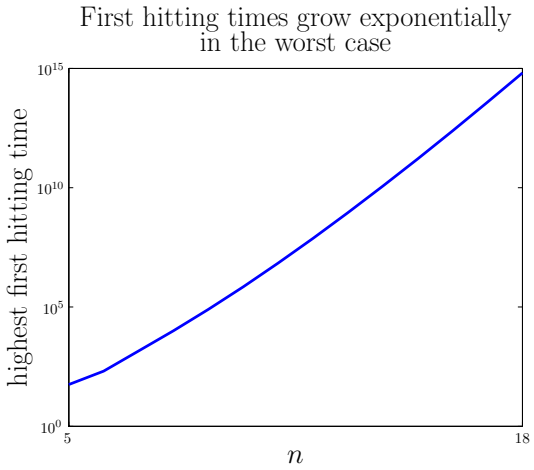

Figure 4.3: The first hitting time of node v grows at least exponentially with the number of nodes in the worst case.

4.3 Towards new bounds on undamped PRI complexity

We here discuss several ideas that share the same objective : finding new bounds, ideally polynomial bounds, on the complexity of PRI in the damped case. We classify these ideas in two categories :

- those that work directly on structural properties of PRI and on its general behavior ;
- those that use Tseng's bound as starting point, and try to adapt it to the undamped case.

Then, in the last section, we will further develop one of these ideas.

4.3.1 Explored- and open- tracks about PRI's general behavior

Proving that the undamped PRI algorithm runs in polynomial time is far from trivial. However, there are properties that could lead to such a proof, provided they are true. In this section, we describe and study these properties. Some of them will show to be false while others, even if not proven, still have potential for leading to new bounds on undamped PRI complexity. In the latter case, we try to give some hints for future developments.

Is it true that the "closest" fragile edge to the target node will be correctly switched at the first iteration ?

First, we give some precisions about what we mean by "the closest fragile edge to the target node". We can actually define several different distance criteria. For example we can say that the distance between the fragile edge (i, j) and the target node v is given by $d_f \triangleq (\varphi_i^* + \varphi_j^*)/2$. Considering the above, the closest fragile edge v should therefore be correctly switched at iteration 1. The intuition behind this hypothesis is that the closest edge to the target node is the least influenced by the other fragile edges. So, it should be the first to "know" how to be switched (using PRI). Unfortunately, the hypothesis does not work with the above distance criterion as shown by the counterexample on Figure 4.4.

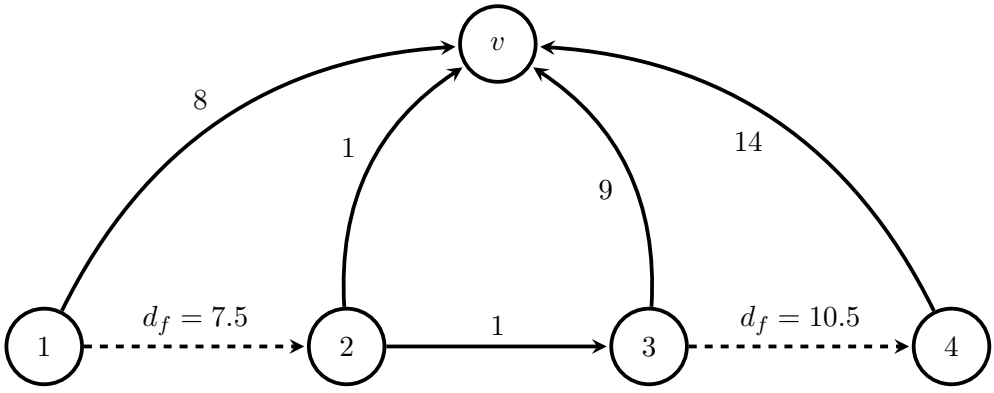

Figure 4.4: In that example, the fragile edge $(1, 2)$ should be switched correctly (in this case, activated) in the first iteration, following our hypothesis. However, if we use PRI on this instance with all fragile edges initially deactivated, we can see that $(1, 2)$ is activated only in the second iteration. Note that the values on the fixed edges represent the length of these edges, meaning that an edge of length l is actually hiding a chain that contains l nodes.

In this counterexample, starting from the configuration in which all fragile edges are deactivated, PRI first activates $(3, 4)$ at iteration 1 and then it activates $(1, 2)$ at iteration 2. This contrasts with our hypothesis : $(1, 2)$ should have been activated at iteration 1 since it is closer to the target node v. Therefore, we have to abandon the hypothesis, at least for this distance criterion.

The same counterexample also rejects the hypothesis for most of the natural distance criteria. Notice that, if it had been true, we could have considered the closest edge to v as fixed at its optimal state after one iteration. By repeating the same argument at the next iterations, we would have shown that PRI converges in at most f iterations[2].

Is it true that PRI cannot change its decisions on a given fragile edge more than a fixed number of times ?

Suppose we want to observe k switches to the same fragile edge when applying PRI (On - Off - On - ... k times). Is it always possible to find such an example ? It turns out that the answer is "Yes". For example, Figure 4.5 shows a graph in which edge (i, j) is switched three times.

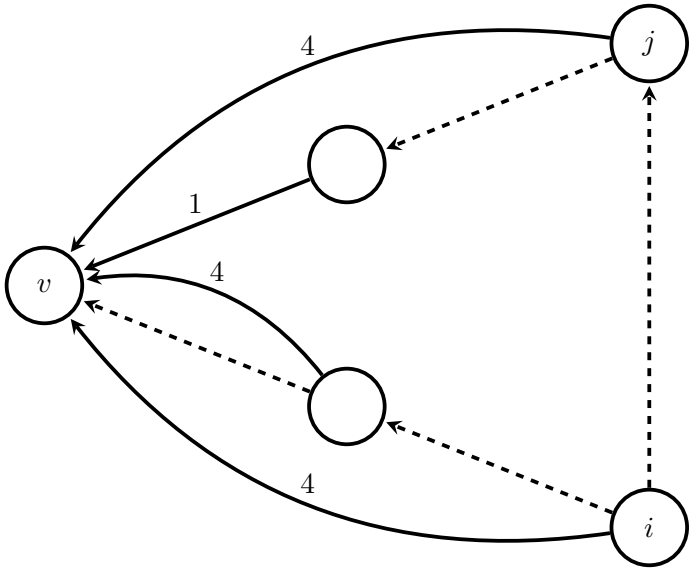

Figure 4.5: In this example, the dashed edges are fragile and have length 1. The weight on the fixed edges can be seen as their length, such that an edge of length l is hiding a chain that contains l nodes. We are here interested in the behavior of the fragile edge (i, j). Assuming that (i, j) is initially On and the other edges are Off, (i, j) will be switched Off, then On and then back Off, three switches in total.

The goal of this example is to alternate the state On or Off of the fragile edge (i, j). To do that, we constructed two branches, from j to v and from i to v, with respectively one and two aligned fragile edges. Initially, only (i, j) is active. The idea is that, at each iteration, one edge from each branch is activated (i.e., the one closest to v that is not active yet). When all the edges of a branch are active, the starting node of the corresponding branch receives a positive effect (i.e., i or j). We make sure that the branches that start in j have an odd number of fragile edges and the branches that start in i have an even number, so that i and j receive the positive effect alternatively. We also choose the weights of the fixed edges in such a way that

[2]In fact, to this number of iterations, we should add one iteration because PRI only stops one step after having found the optimal solution since it has to realize that the policy does not change anymore.

each positive effect produces a switch of (i, j). An example with three branches (and so, with four switches of (i, j)) is proposed in Figure 4.6. Of course, we could add as many branches as we want.

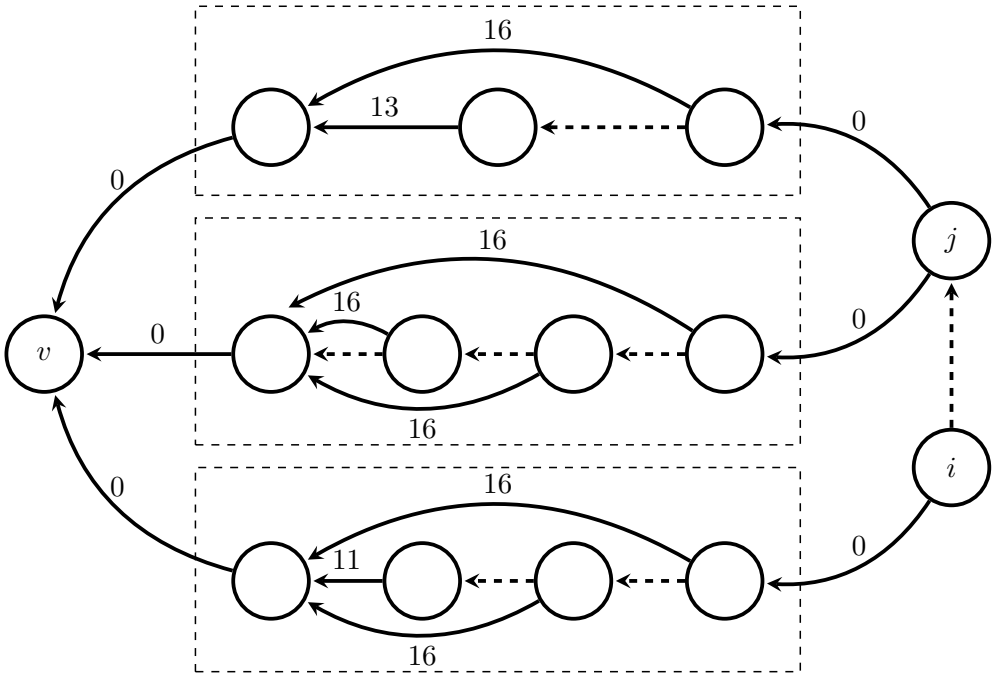

Figure 4.6: The three branches appear clearly here. Those with an odd number of fragile edges are starting from node j while the ones with an even number are starting from i. In this example, we added 0-weight edges to ensure the readability of the graph. Note that in this example, the weights are the lowest possible integers that lead to the switch of (i, j) at every iteration, at least with this approach.

Notice that the more switches we want, the more branches/fragile edges/nodes we need. For $k + 1$ switches for instance, we need k branches, $k(k - 1)/2 + 1$ fragile edges and a lot of nodes (which are mostly contained in the weights of the fixed edges). An interesting question to investigate would be the dependence between the size of the problem, n, and the number of possible switches on one (or more) fragile edge(s). Linked to this issue, another question would be whether it is possible to obtain an example that would have the same number of switches with less nodes and/or more needed iterations of PRI. It is maybe possible to bound the number of switches of each fragile edges with respect to n. This could probably give rise to new bounds on the number of iterations of PRI when damping is absent from the problem.

Is it true that PRI makes at least one final decision at each step ?

If it was, and if the final decision in question does not change during the process afterwards, then obviously there could not be more than f iterations. Unfortunately, we have not yet been able to prove this property in the general case. We have however proved it when there are only one or two fragile edges in the Max-PageRank instance. The case with $f = 1$ fragile edge is trivial since it is always solved in at most 1 iteration.

For $f = 2$, the reasoning is a little more interesting. We make use of proposition 4.1 from [MS99]. Applied to the Max-PageRank problem, this property states that if a subset of fragile edges is switched at a given iteration, then the same subset will never be switched all at once in

a subsequent iteration. Let A and B be the two fragile edges of the problem. By contradiction, we will try to find a feasible situation in which PRI does not make any final decision in at least one of its iterations. We will proceed in two steps. First we will enumerate all these situations. Then we will try to determine which of them are feasible using Proposition 4.1.

- Without loss of generality, since the activation and deactivation actions have a symmetric role and can be exchanged, we can always suppose that we start from a configuration in which all the fragile edges are Off, i.e., $F_0 = \{\}$ where F_k is the set of activated fragile edges at iteration k. First, Let us imagine that PRI only activates one fragile edge at iteration 1. Without loss of generality, we can also suppose that PRI chose to activate edge A. Then, the only possible way such that this modification is not a final decision is if the optimal solution is $F = \{B\}$. Now, suppose that the first choice of PRI was to activate both A and B. Then, the optimal solution can either be $F = \{A\}$ or $F = \{B\}$. In both cases, PRI made one final decision at iteration 1. If we put all these observations together, we obtain the three following sequences of iterations in which at least one iteration did not produce any final decision :

  1. $F_0 = \{\}, \underline{F_1 = \{A\}}, F_2 = \{B\}$;
  2. $F_0 = \{\}, \underline{F_1 = \{A\}}, F_2 = \{A, B\}, F_3 = \{B\}$;
  3. $F_0 = \{\}, F_1 = \{A, B\}, \underline{F_2 = \{A\}}, F_3 = \{B\}$.

- We will now decide for each of these sequences whether they are feasible or not. For that, we will use the modification sets $\mathcal{T}$ at each iteration and see if they always satisfy Proposition 4.1. As part of the Max-PageRank, we can see the set $\mathcal{T}^{\mu_k}$ as the set of all fragile edges that should be switched at iteration k. The set $\texttt{states}(\mathcal{T}^{\mu_k})$ designates the nodes at which the out-going fragile edges will be switched. We know that we can formulate the Max-PageRank problem as an SSP with maximum two actions per state, these actions corresponding to the activation choice on one fragile edge. Therefore, the state corresponding to a fragile edge and the fragile edge itself correspond to the same concept and we will abuse notation saying that $\texttt{states}(\mathcal{T}^{\mu_k})$ contains fragile edges. This gives us for each of the three proposed cases :

  1. $\texttt{states}(\mathcal{T}^{\mu_1}) = \{A\}$, $\texttt{states}(\mathcal{T}^{\mu_2}) = \{A, B\}$;
  2. $\texttt{states}(\mathcal{T}^{\mu_1}) = \{A\}$, $\texttt{states}(\mathcal{T}^{\mu_2}) = \{B\}$, $\texttt{states}(\mathcal{T}^{\mu_3}) = \{A\}$;
  3. $\texttt{states}(\mathcal{T}^{\mu_1}) = \{A, B\}$, $\texttt{states}(\mathcal{T}^{\mu_2}) = \{B\}$, $\texttt{states}(\mathcal{T}^{\mu_3}) = \{A, B\}$.

  As we can see we have the following contradictions in each case :

  1. $\texttt{states}(\mathcal{T}^{\mu_1}) \subseteq \texttt{states}(\mathcal{T}^{\mu_2})$;
  2. $\texttt{states}(\mathcal{T}^{\mu_1}) \subseteq \texttt{states}(\mathcal{T}^{\mu_3})$;
  3. $\texttt{states}(\mathcal{T}^{\mu_{1/2}}) \subseteq \texttt{states}(\mathcal{T}^{\mu_3})$.

This shows the property in case we have two fragile edges. Unfortunately, the same type of argument does not work anymore with more fragile edges since we cannot contradict all the possible situations. It could however be adapted or completed to this end. Note that proving this property in the general case could be an interesting challenge and would lead to a bound linear in n on the number of iterations of PRI, if true.

4.3.2 Using and adapting Tseng's result

Hereunder are some ideas to adapt Tseng's bound efficiently to the undamped Max-PageRank problem. As a reminder, Tseng stated that the number of iterations of PRI is bounded by $O(n\log(n\delta)\eta^{-2r})$ where $n\log(n\delta)$ is polynomial in the binary input size L.

A hypothesis on the diameter of the graph

Using Tseng's result, a first possible way to obtain a polynomial upper bound on the complexity of PRI is to assume that the diameter of the graph is bounded by a universal constant κ, i.e., independently from the graph's size n. With this assumption, we have that $\eta^{-2r} = \eta^{-2\kappa} = O(n^{2\kappa})$ for a Max-PageRank instance. Therefore, the number of iterations is bounded by $O(n\log(n\delta)n^{2\kappa})$ and PRI runs in weakly polynomial time. A detailed proof of that fact can be found in [CJB09].

Using the special structure of random graphs

One can wonder what Tseng's bound becomes when manipulating random graphs since such graphs often have special structure and properties. Here we analyze succinctly what happens for the average behavior of several graphs that are frequently used in the literature.

- **Erdös-Renyi random graphs.** According to [CL01], for sufficiently large graphs in which there is a giant component, we may consider that $\eta = O(1/(pn))$, where p is the probability of occurrence of an edge, and that $r = O(\log n/\log(pn))$. Therefore, PRI has a number of iterations bounded by a polynomial in $(pn)^{2\log n/\log(pn)}$ and in the binary input size L in that case.
- **Power Law random graphs.** Again, in the presence of a sufficiently large graph, we can consider that $\eta = O(1/\sqrt{n})$ using the construction described by [MST04] and that $r = O(\log n)$, [ACL04]. This leads to a bound for the complexity of PRI that is polynomial in $n^{\log n}$ and in L.
- **Expander random graphs.** In these graphs, every node has the same degree d fixed beforehand and therefore, $\eta = 1/d$. According to [VG09], it can be shown that the average diameter of expanders if given by $r = O(\log n)$ and hence, using Tseng, we have that the number of iterations of PRI is polynomial in $d^{2\log n}$ and in L.

From the above, we can see that even though none of the obtained bounds for PRI is polynomial, they are not purely exponential either. Furthermore, these bounds simply result from Tseng's bound, which is not tight.

Undamped Max-PageRank as a damped problem

One can observe that Tseng's bound is not applicable when there is no damping in the problem because the diameter of the problem's graph is not bound. However, an arbitrary small damping would be sufficient to have a graph with unit diameter. So, the question that we would like to address is to find the maximum damping constant that we can add to an arbitrary undamped Max-PageRank problem such that the optimal configuration of fragile edges obtained using PRI remains unchanged.

Beforehand, we should check whether such a damping constant always exists (i.e., a non-zero damping constant). Therefore, we first redefine the notation for the first hitting times as a function of the damping constant c :

Notation 4.7. *The expected first hitting time of a state s under policy μ with respect to the applied damping c is denoted by $\varphi_s^\mu(c)$.*

Of course, relation (4.7) still holds. Furthermore, we know that $\varphi_s^\mu(c)$ is continuous for all $c \in [0, 1]$. Starting from $c = 0$, we are interested to find out from which value of c the optimal policy could change. Using the continuity of φ with respect to c, we know that for all graph $\mathcal{G}$ of size n and for any set of f fragile edges, $\mathcal{F}$, there exists $\varepsilon > 0$ such that for all $c < \varepsilon$, the optimal policy will not change.

The goal is now to find a value (or at least a lower bound) for ε with respect to the problem's parameters. In section 4.4, we discuss an approach to tackle this problem.

4.4 From undamped to damped Max-PageRank : an approach

As suggested in the previous section, we now want to show that the solution obtained for the undamped case of our problem is equivalent to the one obtained for the damped case for sufficiently small values of the damping constant c. To show that, and to find the maximum acceptable value for c (denoted by ε), we here study an interesting approach based on the two following steps :

1. First, we wish to find a constant K such that for any state $s \in \mathcal{S}$ and for any policy μ' that has non optimal value at state s (i.e., $\varphi_s^{\mu'}(0) > \varphi_s^*(0)$), we have

$$\varphi_s^{\mu'}(0) - K \geq \varphi_s^*(0)$$

   where φ_s^* is the first hitting time of state s for an optimal policy. Here, K is supposed to measure how close can any suboptimal policy be from the optimum in terms of their associated first hitting times. In other words, we want to find $K > 0$ such that

$$K \leq K' \triangleq \min_{s,\mu',\varphi_s^{\mu'}(0)>\varphi_s^*(0)} \varphi_s^{\mu'}(0) - \varphi_s^*(0). \tag{4.13}$$

2. Then, we wish to determine a constant $\varepsilon > 0$ such that for all $c < \varepsilon$, we have :

$$\begin{aligned} \varphi_s^*(c) &\leq \varphi_s^*(0) + \varepsilon_1 \\ \varphi_s^{\mu'}(c) &\geq \varphi_s^{\mu'}(0) - \varepsilon_2 \end{aligned}$$

   for all states $s \in \mathcal{S}$ and for any policy μ' that is suboptimal in s, such that $\varepsilon_1 + \varepsilon_2 \leq K$. This implies that if we use a value of c that is inferior to ε, we do not change the functions φ enough to change the optimal policy.

The global idea is illustrated on Figure 4.7.

Unfortunately, as we will see, this approach does not produce the expected result since it leads to an exponentially small lower bound for ε. Consequently, the smallest transition probability $\eta = c/n$ is exponentially small and we do not obtain a polynomial bound on the number of iterations of PRI. However, the approach does generate a number of interesting ideas on PRI and the Max-PageRank problem, as we will see during the analysis.

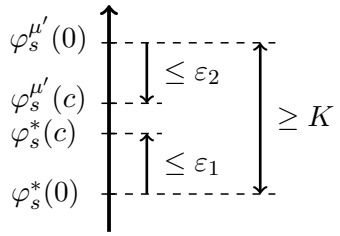

Figure 4.7: We want to find a value K such that we know that the *gap* between $\varphi_s^{\mu'}(0)$ and $\varphi_s^{\mu^*}(0)$ will always be at least K. Then, we want to measure the influence of adding some damping, on the first hitting times (ε_1 and ε_2). We conclude by saying that if we add sufficiently small damping such that the variation of the first hitting times is less than $\varepsilon_1 + \varepsilon_2 \leq K$, then the optimal policy cannot change since we do not modify the first hitting times enough for that and μ' remains non optimal.

4.4.1 Bound on the minimum change of value when changing policy

We here want to construct a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and a set of fragile edges $\mathcal{F} \subseteq \mathcal{E}$ without damping such that K' defined by (4.13) is as small as possible[3]. For that, we proceed in three steps.

1. We first show that, whatever the instance, the closest policy to the optimum, μ', always has exactly one misconfigured edge (compared with at least one of the optimal policies μ^*) that we will call (w, j). Since we do not care about all the other policies, this enables us to consider instances with only one fragile edge : (w, j).

2. Then we construct the case in which $\varphi_w^{\mu'}$ is closest to $\varphi_w^{\mu^*}$. We also evaluate how small $\varphi_w^{\mu'} - \varphi_w^{\mu^*}$ can be.

3. Finally, we evaluate the minimum influence that w can have on the other nodes of the graph. The smaller the influence, the closer these nodes can be from their optimal value.

Using these three steps, we are able to find a lower bound K on the value of K'.

We may consider only one fragile edge

We say that the closest policy to the optimum, μ', is the one such that

$$\min_{s, \varphi_s^{\mu'} > \varphi_s^{\mu^*}} \varphi_s^{\mu'} - \varphi_s^{\mu^*}$$

is minimal. Let us show that μ' has exactly one misconfigured fragile edge (w, j). This property simply comes from Proposition 4.5 applied to the Max-PageRank problem. Indeed, in that case, the proposition states that we are always able to order all the possible policies in such a way that two consecutive policies μ_k and μ_{k+1} satisfy $\varphi_s^{\mu_k} \geq \varphi_s^{\mu_{k+1}}$ for any state s and that μ_k and μ_{k+1} differ in exactly one fragile edge. Therefore, there exists a policy μ^* that dominates μ' (and is optimal since μ' is as close from optimal as possible) and that differs from μ' in exactly one fragile edge.

Using this result, we can always suppose that there is only one fragile edge. Any graph with more than one fragile edge can indeed be reduced to the same graph where all fragile edges

[3]the dependence of the first hitting times in c can be omitted here since we are only interested in the zero-damping case.

other than (w, j) become fixed edges according to policy μ^* (or μ'), giving the same result for the analysis (i.e., the same minimal difference between $\varphi_w^{\mu'}$ and $\varphi_w^{\mu^*}$). In addition, we can assume without loss of generality that (w, j) is optimally deactivated and so, (w, j) is activated under policy μ'.

Case construction and analysis

We now want to find an example with fragile edge (w, j) only, in which $\varphi_w^{\mu'} - \varphi_w^{\mu^*}$ is minimal. To achieve that, we should have a situation in which (1) w is connected to as many nodes as possible to reduce the influence that j has on w as much as possible, and in which (2) $\varphi_j^{\mu'}$ is as close as possible to the mean of the first hitting times of all the other neighbors of w.

First, to keep things simple, we will artificially make sure that every node other than w has its optimal value (or that they are independent from w with the same result). That way, w becomes even closer to its optimal value, yet without reaching it while (w, j) is active. Such a situation where every node except w has its optimal value can be constructed from any one-fragile-edge graph adding one auxiliary node that we call w'. In the construction, w' will play the role of w in its optimal state : it has all the same out-going edges as w except (w', j). Then, all the nodes that initially pointed towards w now point towards w' and therefore, they reach their optimal value since the subgraph with w' and without w is now optimal according to policy μ^*. Figure 4.8 illustrates that construction on a little example.

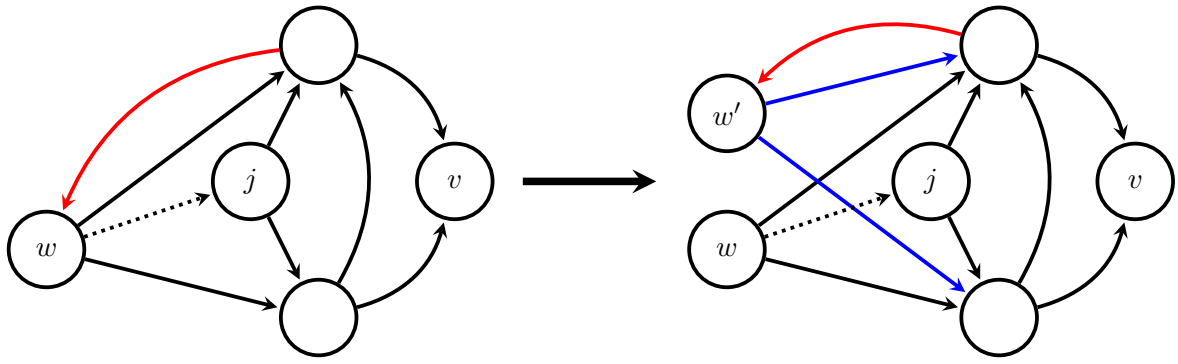

Figure 4.8: An example of construction to make sure that every node except w has optimal value. Therefore, we add an auxiliary node that plays the role of w in its optimal state. After the construction, the red edge goes to w' instead of w. We also add the blue edges out-going from w' to make it identical to w in the optimal configuration. Here, the dotted edges correspond to activated fragile edges.

We can now consider that w is the only state that has not yet its optimal value. According to the observations made earlier, we should connect w to every other node except v (j being connected to w with a fragile edge), in order to achieve our goal. Let us call $\mathcal{S}' = \mathcal{S} \backslash \{w, j, v\}$ the set of fixed neighbors of w (i.e., j excluded). We then have :

$$\begin{aligned} \varphi_w^{\mu'} &= \overline{\varphi}_{\mathcal{S}' \cup \{j\}}^{\mu'} + 1 = \overline{\varphi}_{\mathcal{S}' \cup \{j\}}^{\mu^*} + 1, \\ \varphi_w^{\mu^*} &= \overline{\varphi}_{\mathcal{S}'}^{\mu'} + 1 \qquad = \overline{\varphi}_{\mathcal{S}'}^{\mu^*} + 1, \end{aligned}$$

where $\overline{\varphi}_{\mathcal{S}}^{\mu}$ is the mean of the first hitting times of all states in the set $\mathcal{S}$, according to policy μ. Here, we have the second equalities because we assumed that all states other than w are at their optimal value, even with policy μ'. So, our goal is to minimize (or more precisely to find a lower bound on) the gap $\varphi_w^{\mu'} - \varphi_w^{\mu^*}$ which is equivalent to $\overline{\varphi}_{\mathcal{S}' \cup \{j\}}^{\mu'} - \overline{\varphi}_{\mathcal{S}'}^{\mu'}$. We can see that

the higher the denominators of the elements of $\varphi^{\mu'}$ are, the lower this quantity can be. This is why we are going to analyze these denominators next.

In this case, we can restrict our analysis to the subgraph in which w is replaced by w', as indicated by the above construction. Note that we will here make several shortcuts in the calculations and give only the main steps. The complete reasoning can be found in appendix A. First, recall that φ is given by (1.5). So, the denominator of the elements of φ will be dictated by $(I - \overline{P})^{-1}$. It is also well known that if $I - \overline{P}$ is invertible, then we can write its inverse as

$$(I - \overline{P})^{-1} = \frac{1}{\det(I - \overline{P})} \operatorname{Adj}(I - \overline{P}),$$

where $\operatorname{Adj}(A)$ is the adjoint of matrix A as defined for example in [Doo08]. The important argument here comes from the following theorem from [Had93] :

Theorem 4.8 (Hadamard's maximum determinant problem)**.** *Let A be an n by n matrix such that $|a_{ij}| \leq 1$ for all i, j. Then the following inequality is true :*

$$|\det(A)| \leq n^{n/2}.$$

Using this theorem, we are able to bound the determinant of $I - \overline{P}$. We use this to deduce an exponential upper bound of order $O(n^n)$ on the denominators of the first hitting times. Introducing this result into the above developments, we finally obtain :

$$\varphi_w^{\mu'} - \varphi_w^{\mu^*} \geq K = O(n^{-n}).$$

As said, the complete calculation details that lead to that bound can be found in appendix A. Note that we made the analysis assuming that (w, j) was optimally deactivated. Yet, we could have done exactly the same reasoning if we had assumed the opposite.

Minimum influence and conclusion

Now that we have a bound on how close w can be from its optimal value, we should evaluate its influence on the other nodes of the graph (we may suppose that we are back in the initial graph where all nodes may possibly depend on w). It is easy to see that the value gap $\varphi_s^{\mu'} - \varphi_s^{\mu^*}$ is minimal when s is "as far as possible" from w. And the weighted distance between s and w is bounded below by η^r which is bounded below by $O(n^{-n})$. So, the very smallest possible gap between a suboptimal value and an optimal value of a given node is also bounded below by $O(n^{-n})$.

Of course, unfortunately, this bound is exponential and will produce an exponential bound with Tseng's result as well. Better bounds could be found even though we believe that it is possible to build examples that would achieve the exponentially small gap.

4.4.2 Bound on the maximum change of value when increasing damping

For the second step of our approach, we want to find ε such that for any damping constant $c < \varepsilon$, we have $\varphi_s^{\mu}(0) - \varphi_s^{\mu}(c) \in [-\varepsilon_1, \varepsilon_2]$ for any state $s \in \mathcal{S}$ and for any policy μ, such that $\varepsilon_1 + \varepsilon_2 \leq K$. Clearly, ε_1 measures the maximum *increase* of φ when we add damping to the problem while ε_2 measures its maximum *decrease*. Since the maximum decrease can be much more significant than the maximum increase, we mainly focus on how to bound the former one in this section.

To bound ε_2, we will use the worst case that we analyzed in section 4.2.2. The idea is to show that, when we apply a damping constant c to the problem, this particular case gives rise to the highest possible decrease of value. The intuition is quite simple : in that example, the first hitting times without damping are very high since it is extremely difficult (as difficult as possible) to go towards the target node v because the majority of the edges are going "backwards". But if we add damping to the problem, then we always have a probability c/n to immediately go to the target node v. This will result in a drastic decrease of the first hitting times, i.e., the highest decrease possible. And it is exactly this decrease rate that we want to determine.

To find an upper bound on this decrease, we will essentially proceed in three steps :

- First, we determine an upper bound on the first hitting times of the worst case, without damping yet.
- Then, we propose a favorable version of damping were the personalization vector z is such that all nodes go to v anytime that damping operates, i.e., $z_v = 1$. With this version of damping, the decrease of the first hitting times is even higher. We then bound the obtained decrease.
- Finally, we argue that such a case leads to the worst possible decrease of the first hitting times.

Then, we determine a bound on the maximal increase of the first hitting times with a simple argument. Altogether, this will give us a lower bound for the desired value of ε.

At last, we will also see that the worst case described in section 4.2.2 actually achieves an exponential decrease when we add damping to the problem. This will also give us an exponentially small upper bound for ε.

Bound on the first hitting times in the worst case

We first want to determine the first hitting time of node v in the worst case, with respect to the number of nodes n. Unfortunately, it is difficult to obtain a closed form for φ_v in function of n. However, it is easy to verify numerically that

$$\varphi_v \leq \frac{2n!}{n-1}.$$

Less precise bounds can also be obtained analytically but they require many calculations that we will not perform here. Of course, this bound is also valid for all the other nodes of the worst graph, since $\varphi_v \geq \varphi_i$ for all $i \in \mathcal{V}$. Besides, surprisingly, all nodes have very close first hitting times, even the nodes that are closer to the target node. This is due to the fact that even if we start at the closest node to v, we have a fair chance of going back towards the other nodes and only a small chance to actually reach v.

Note that the quality of the bound can be observed in appendix B.

A favorable version of damping

For now, we assume that the worst graph described in section 4.2.2 is indeed the one that will lead to the highest decrease of the first hitting times when we add damping. Then, we consider a particular personalization vector z in which all the nodes are connected to only v

when damping operates, i.e., $z_v = 1$ and $z_i = 0$ for all nodes $i \neq v$. This version of damping, that we will refer to as *favorable*, ensures an even greater decrease of the first hitting times, and more precisely of φ_v since v is the most critical node. Figure 4.9 illustrates how the worst case looks like in that case, with $n = 4$.

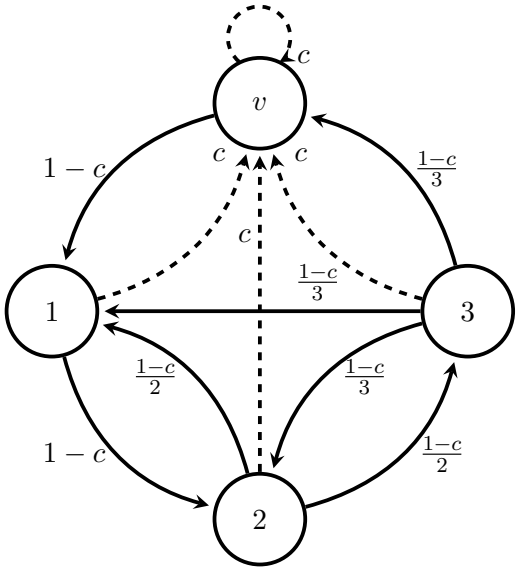

Figure 4.9: The worst case with $n = 4$ when we add favorable damping. Here, the dashed edges represent the links that appeared due to damping. The coefficients on the edges are their transition probabilities.

Note that $\varphi_v(c)$ is always inferior to $1/c$. To understand this, suppose that n is high. If there is no damping, the first hitting time of v is enormous, almost infinite. Once we add damping, however, we have probability c to go directly to v. So, in absence of damping, we continue the almost infinite random walk ; as soon as damping kicks in, we finish the random walk in one step. Therefore, the average time spent on the graph before such an event happens is simply $1/c$. Furthermore, when n is small, we have a better probability of reaching v before damping operates. Consequently, $\varphi_v(c)$ is always inferior to $1/c$ and reaches it asymptotically when n tends to infinity.

We now want to find an upper bound $B_n(c)$ on the decrease $\varphi_s^\mu(0) - \varphi_s^\mu(c)$, knowing already that $\varphi_s^\mu(0) \leq \frac{2n!}{n-1} \triangleq b_n$. We also would like our bound to be unbiased such that $B_n(0) = 0$. To do that, we propose the following approximation for $\varphi_s^\mu(c)$:

$$\varphi_s^\mu(c) \approx \frac{\sqrt{b_n} - \sqrt{b_n}^{-1}}{c\sqrt{b_n} + \sqrt{b_n}^{-1}} + 1.$$

This function is built using the principle of hyperbolic tangent and it converges to $1/c$ when n tends to infinity. It is possible to check the quality of this approximation numerically. Besides, observe that it verifies $\varphi_s^\mu(0) = b_n$. So, in the end, we obtain the following upper bound on ε_2 :

$$\varepsilon_2 \leq B_n(c) = \frac{cb_n^2 - cb_n}{cb_n + 1}.$$

The quality of this bound is briefly discussed in appendix B. Note that it is not a problem to use approximations here since we only try to exhibit the exponential behavior of $B_n(c)$.

Bound on the decrease of the first hitting times

Let us now justify why this worst case with favorable damping generates the highest decrease of the first hitting times when adding damping to the problem. We first show that the favorable version of damping provokes a decrease to the first hitting times of all nodes of any graph when c increases. Then, we give two criteria for an important decrease.

Proposition 4.9. *Let P be the transition matrix for a Max-PageRank problem and let $\overline{P}$ be the same matrix in which the v^{th} column has been set to zero. Then, if $I - \overline{P}$ is invertible, we have, in case of favorable damping, that*

$$\frac{\mathrm{d}\varphi(c)}{\mathrm{d}c} \leq 0$$

for any $c \in [0, 1]$, where $[\mathrm{d}\varphi(c)/\mathrm{d}c]_i$ is the derivative of $\varphi_i(c)$.

Proof. We know that the first hitting times are given by expression (4.7). In case of favorable damping, this expression can be reformulated as

$$\varphi(c) = (I - (1-c)\overline{P})^{-1}\mathbb{1}.$$

We assumed that $I - \overline{P}$ is invertible, and therefore $\varphi(c)$ is derivable with respect to c for any $c \geq 0$ and

$$\frac{\mathrm{d}\varphi(c)}{\mathrm{d}c} = -(I - (1-c)\overline{P})^{-1}\ \overline{P}\ (I - (1-c)\overline{P})^{-1}\mathbb{1}.$$

Moreover, since $I - (1-c)\overline{P}$ is invertible for any $c \geq 0$, we can apply the following converging development :

$$(I - (1-c)\overline{P})^{-1} = \sum_{k=0}^{\infty}(1-c)^k\overline{P}^k.$$

Obviously, if $c \leq 1$, we have $[(1-c)^k\overline{P}^k]_{ij} \geq 0$ for any k and for any $i, j \in \mathcal{V}$ because all the elements of $\overline{P}$ are non negative. Thus, we also have that $[(I-(1-c)\overline{P})^{-1}]_{ij} \geq 0$ for all $i, j \in \mathcal{V}$ and therefore

$$\left[\frac{\mathrm{d}\varphi(c)}{\mathrm{d}c}\right]_i \leq 0$$

for all $i \in \mathcal{V}$ and for any $0 \leq c \leq 1$. □

This proposition states that in case of favorable damping, increasing c always results in a decrease of the first hitting times at all nodes, and this irrespective of the instance's graph.

Now we give two criteria that dictate in what case there is a high decrease when applying favorable damping. These will give us the necessary intuition about why the worst case with favorable damping provokes the highest possible decrease.

First, Let us observe a node s and let us assume that the first hitting times of all the other nodes are fixed and will not move when we add damping, i.e., $\varphi_i(0) = \varphi_i(c)$ for all $i \neq s$. Then, the decrease of φ_s when we go from damping $c = 0$ to damping $c > 0$ is largest when $\varphi_s(0)$ is as high as possible. Indeed, using the favorable damping version and the above hypothesis :

$$\begin{aligned}\varphi_s(0) - \varphi_s(c) &= \varphi_s(0)\ -\ [(1-c)\varphi_s(0) + c] \\ &= c\ (\varphi_s(0) - 1).\end{aligned}$$

So, we deduce that $\varphi_s(0) - \varphi_s(c)$ is maximal when $\varphi_s(0)$ is maximal. However, we here fixed the first hitting times of the nodes other than s. Yet, if we remove that hypothesis, we saw that the first hitting times of the neighbors of s would also decrease, provoking an even larger decrease of φ_s and so

$$\varphi_s(0) - \varphi_s(c) \geq c\ (\varphi_s(0) - 1).$$

This fact can be seen as a first criterion to achieve a large decrease.

On the other hand, we just said that φ_s decreases even more when its neighbors experience a high decrease too. So $\varphi_s(0) - \varphi_s(c)$ will be even higher if $\varphi_i(0) - \varphi_i(c)$ is high, for all neighbors i of s. This can be seen as a second criterion for high decrease. Furthermore, these two criteria are complementary and they are the only ones that manage the decrease of $\varphi_s(0)$.

Let us now consider the worst case. In this case, we already saw that the first hitting times of all nodes are as high as possible (i.e., the k^{th} highest first hitting time in the worst graph is higher than any k^{th} highest first hitting time in any other graph). So, we already maximize the first criterion. But then, all the nodes experience a high decrease, which also affects the first hitting times of the neighbors, from the second criterion, and so on recursively. Therefore, the decrease of all the first hitting times should be as high as possible. This argument sounds convincing, even though it is not perfectly rigorous.

Bound on the increase of the first hitting times and final bound on ε

The maximum increase of the first hitting times is less critic than their decrease. The worst possible undamped situation that can happen is illustrated on Figure 4.10.

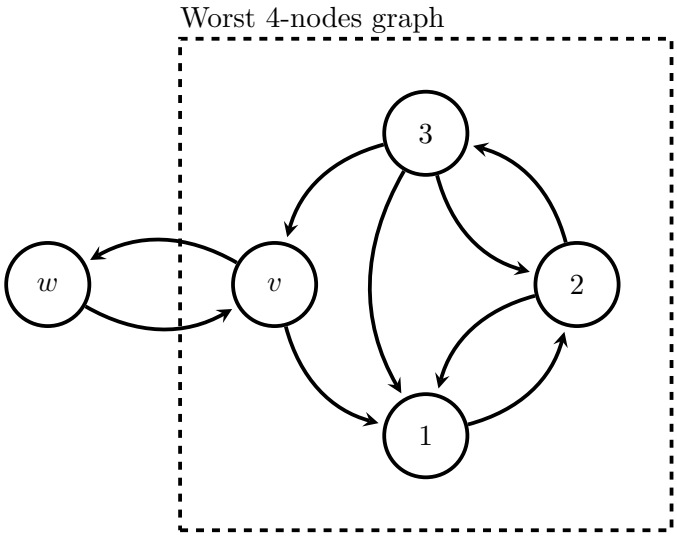

Figure 4.10: Illustrates the worst possible case for the maximum increase of the first hitting times, with $n = 5$. When there is no damping, $\varphi_w = 1$. But then, when damping is added to the problem, w has a chance to reach the graph with highest possible first hitting times.

In this case, the node w starts with a unit first hitting time when there is no damping. Then, as soon as damping appears, it becomes connected to the worst graph with almost probability c and its first hitting time grows extremely fast. Of course, since the first hitting times of the nodes of the worst graph are bounded by $b_n = \frac{2n!}{n-1}$ whatever the value of c, the increase of the first hitting time of w is always bounded by cb_n. Therefore, we have for sure that $\varepsilon_1 \leq cb_n$.

This development completes our analysis of ε_1 and ε_2 and we have :

$$\varphi_s^\mu(0) - \varphi_s^\mu(c) \in \left[-cb_n \ , \ \frac{cb_n^2 - cb_n}{cb_n + 1}\right].$$

Now, we would like to determine ε such that for all $c \leq \varepsilon$, we have $\varepsilon_1 + \varepsilon_2 \leq K$. To do that, we just have to isolate c in the latter expression. Doing so, we obtain the following bound on ε :

$$\varepsilon \geq \frac{K}{2b_n^2 - Kb_n} = O\left(\frac{K}{B^2}\right)$$

which is exponentially small. This is of course not a convenient bound when applied to our bound on the complexity of damped PRI from section 4.1 since the bound we found there depends on $1/c$ which is here exponential. So, our developments did not generate an appropriate bound but it does not mean that such a bound does not exist yet. This question is dealt with in the next point.

Actual exponential decrease of the first hitting times

In the above, we were able to determine exponentially high upper bounds on ε_1 and ε_2. The question is now : do examples achieving an exponential decrease of $\varphi_s^\mu(0)$ really exist ? And the answer is "Yes" : the above worst case achieves such a decrease, even with the classical version of damping in which the personalization vector is $z = (1/n)\mathbb{1}$. We can check this fact numerically, as Figure 4.11 illustrates.

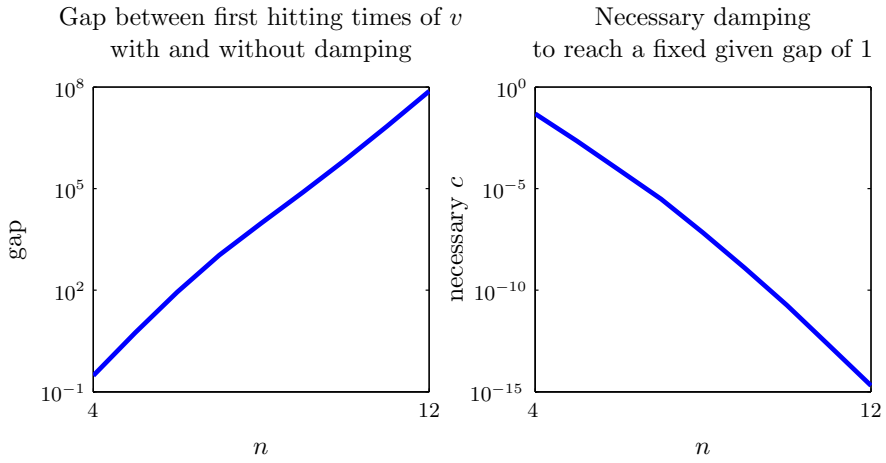

Figure 4.11: The worst case is an example of a graph that achieve an exponential decrease of the first hitting times (notice the logarithmic scale of the y-axis) when we add damping to the problem in the classical way. In the left figure we can see that the gap between $\varphi_v(0)$ and $\varphi_v(0.1)$ is actually growing exponentially with n. The right figure displays the highest acceptable damping constant that can be imposed to obtain a unit gap, with respect to n. We see that this constant decreases exponentially when n grows.

This example simply shows that our approach will always generate inappropriate (i.e., exponentially small) lower bounds for ε. However, this does not mean that it is not possible to find polynomial lower bounds on ε at all yet. Indeed, the two steps of our approach are unrelated and the bounds we found in both steps do not refer to the same kinds of examples. So, we should be looking for an example that behaves as badly as possible for both the first and the second step combined. Hence, it is still possible that such a case would give us a polynomially small value for ε.

As a conclusion for this section, our approach is probably not the one that will lead to the desired result. It has however produced some interesting developments that could maybe be reproduced in other situations.

Conclusions

In the present work, we have focused on the problem of optimizing the PageRank of a given node when a subset of edges are fragile. This problem has already generated much interest [AL04, KND08, IT09, CJB09]. Previously, the only known efficient algorithm to solve the problem was approximative [IT09]. In response to this, we have seen that the PageRank optimization problem could be formulated as a Stochastic Shortest Path problem which is a special case of undiscounted Markov Decision Processes. Using this formulation, the authors of [CJB09] were able to formulate an efficient and exact algorithm using linear programming. They also proposed an iterative algorithm (PageRank Iteration) based on the well known Policy Iteration method and they claimed that this method has good properties to solve the PageRank optimization problem.

Following their advice, we have tried to study the performances of the so called PageRank Iteration algorithm. We have first generalized the method to the damped case in which we add a certain probability of restarting the random walk process. Then, we have compared the three methods that have been proposed and we have validated the fact that PageRank Iteration, by far, exhibits the best behavior among the three. We also experimentally highlighted the complexity of the latter algorithm which seems to be polynomial in practice ; its number of iterations appears to be linear or even logarithmic in the problem's size. Yet, the main part of our work aimed to study the theoretical behavior of PageRank Iteration. Our main result is a polynomial upper bound on the complexity of PageRank Iteration in the damped case of the problem. In our attempts to generalize this result to the undamped case, we have also come across a number of interesting properties of all kinds as well as several open questions.

Among these open questions, we think that some are promising and could potentially lead towards new key results on PageRank Iteration and on the general Policy Iteration algorithm, if they were to be solved. For instance, it would be interesting to investigate if a general Stochastic Shortest Path problem can be reduced to a PageRank optimization problem. In that case, all the results we found for the latter could be applied to the former, including the polynomial bound for Policy Iteration with damping. However, an additional assumption is necessary since we have seen that this single hypothesis has been invalidated [Fea10]. For example, we can impose that all the transition probabilities are polynomial in the problem's size.

The other key question that remains is to find a polynomial bound on the complexity of PageRank Iteration without damping. We see three promising approaches to tackle that problem :

- bound the number of switches of a fragile edge made by the algorithm, with respect to the problem's size ;
- show that PageRank Iteration always makes at least one final decision at each iteration, i.e., a decision that will not be changed until the algorithm has converged ;

- find an equivalent damped problem that would give the same optimal configuration of the fragile edges if solved with PageRank Iteration. We could then apply our polynomial bound, provided that the equivalent damped problem has a polynomially small damping constant.

This last approach was discussed in detail in our work. Remark that the undamped problem should intuitively be easier to solve than the damped one, although the latter does have the property that the diameter of its associated graph is bounded by a (unit) constant. Therefore, we have a strong feeling that a polynomial bound should exist on the complexity of undamped PageRank Iteration, even without reducing it to the damped case. The problem is open.

Appendix A

Calculation precisions for the minimum value gap

This appendix is a complement of information to what has been done in section 4.4.1. We here detail a little bit more the calculations needed in this section. First of all, let us remember that

- we are in a case in which there is only one fragile edge (w, j) ;
- (w, j) is optimally deactivated ;
- μ' is the non-optimal policy in which (w, j) is activated ;
- every node other than w has optimal value ;
- w is linked to every other nodes, except v and we refer to the set of its neighbors by $\mathcal{S}' \cup \{j\}$.

Knowing that, we wish to find the situation, compatible with the above, that minimizes $\overline{\varphi}^{\mu'}_{S'\cup\{j\}} - \overline{\varphi}^{\mu'}_{S'}$. Note that according to the above hypothesis, we know for sure that $\overline{\varphi}^{\mu'}_{S'\cup\{j\}} - \overline{\varphi}^{\mu'}_{S'} > 0$. This expression can be further developed :

$$\begin{aligned}
\overline{\varphi}^{\mu'}_{\mathcal{S}'\cup\{j\}} - \overline{\varphi}^{\mu'}_{\mathcal{S}'} &= \left(\frac{n-3}{n-2}\,\overline{\varphi}^{\mu'}_{\mathcal{S}'} + \frac{1}{n-2}\,\varphi^{\mu'}_{j}\right) - \overline{\varphi}^{\mu'}_{\mathcal{S}'} \\
&= \frac{1}{n-2}\left(\varphi^{\mu'}_{j} - \overline{\varphi}^{\mu'}_{\mathcal{S}'}\right) \\
&= \frac{1}{n-2}\left(\varphi^{\mu'}_{j} - \frac{1}{n-3}\,\sum_{s\in\mathcal{S}'}\varphi^{\mu'}_{s}\right).
\end{aligned}$$

since the cardinality of $\mathcal{S}'$ is $n-3$. From this last expression, it appears that the gap we want to minimize can be bounded below by $\frac{1}{(n-2)(n-3)D}$ where D is the highest possible denominator of the first hitting times. So, if we find an upper bound on D, we also find the desired result. For that, let us remind that

$$\varphi = (I - \overline{P})^{-1}\mathbb{1}.$$

Assuming that $I - \overline{P}$ is invertible, it is well known that

$$(I - \overline{P})^{-1} = \frac{1}{\det(I - \overline{P})}\,\mathrm{Adj}(I - \overline{P}),$$

where $\mathrm{Adj}(A)$ is the adjoint of matrix A [Doo08]. More precisely, the adjoint matrix is defined as

$$\mathrm{Adj}(A) \triangleq \mathrm{Cof}(A)^T,$$

where $\mathrm{Cof}(A)$ is the cofactor matrix of A such that

$$[\mathrm{Cof}(A)]_{ij} \triangleq (-1)^{i+j}[\mathrm{Min}(A)]_{ij},$$

where $\mathrm{Min}(A)$ is the minor of A and $[\mathrm{Min}(A)]_{ij}$ is the determinant of a matrix obtained from A by deleting its i^{th} row and j^{th} column.

It is quite easy to see that the denominator of the elements of $\mathrm{Min}(I - \overline{P})$ is at most n^{n-1} since all the elements of $I - \overline{P}$ are rationals with denominator at most n. Hence, so is the denominator of the elements of $Adj(I - \overline{P})$. Then, using theorem (4.8), we obtain the following bound on the denominators of the first hitting times :

$$D \leq n^{n/2} n^{n-1}.$$

This finally leads to the desired bound, which becomes

$$\overline{\varphi}^{\mu'}_{S' \cup \{j\}} - \overline{\varphi}^{\mu'}_{S'} \geq \frac{1}{(n-2)(n-3)n^{3n/2-1}} = O(n^{-n}) = K.$$

Of course, this bound is exponential and is probably not the tightest one. However, since the result obtained in section 4.4.2 is also exponential, it does not really matter.

Appendix B

Numerical bounds

In section 4.4.2, we have bounded a number of expression asserting that it was possible to verify the bounds numerically. Here, we give some figures that give an idea of the quality of the proposed bounds.

First, we discussed an absolute upper bound on the first hitting times. For that, we have built a worst case and we said that the first hitting times of the worst case were always inferior to $\frac{2n!}{n-1}$. On Figure B.1, we compare this bound to the actual maximum first hitting times.

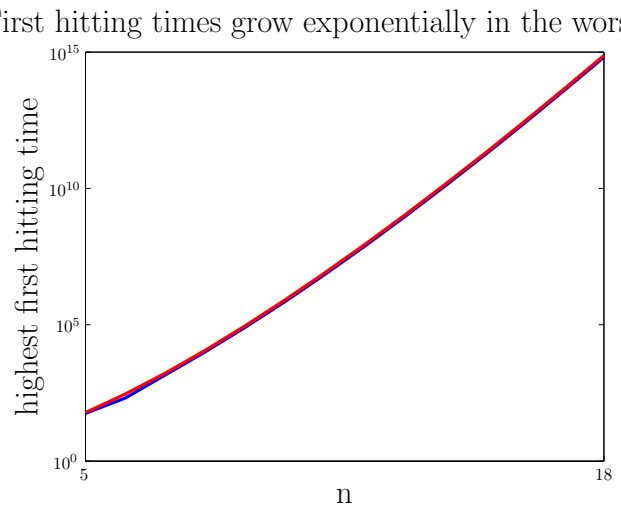

Figure B.1: In blue : the actual maximum first hitting time with respect to n in the worst case. In red : our upper bound on this maximum first hitting time. One can see that the bound is quite close to the real maximum but is always superior to it.

The blue curve represents the reality while the red one is our bound. Clearly, the bound seems to fit almost perfectly to the blue curve, while still remaining superior to it.

Later, we have tried to bound the maximum decrease of the first hitting times $\varphi_s(0) - \varphi_s(c)$ when damping is added to the problem. Therefore, we have used the above bound for $\varphi_s(0)$ as well as an approximation for $\varphi_s(c)$. Combined, we obtained the following bound on the maximum decrease :

$$\varphi_s(0) - \varphi_s(c) \leq \frac{cb_n^2 - cb_n}{cb_n + 1}$$

where $b_n \triangleq \frac{2n!}{n-1}$ corresponds to the upper bound on $\varphi_s(0)$. Figure B.2 enables us to check once again the quality of the proposed bound.

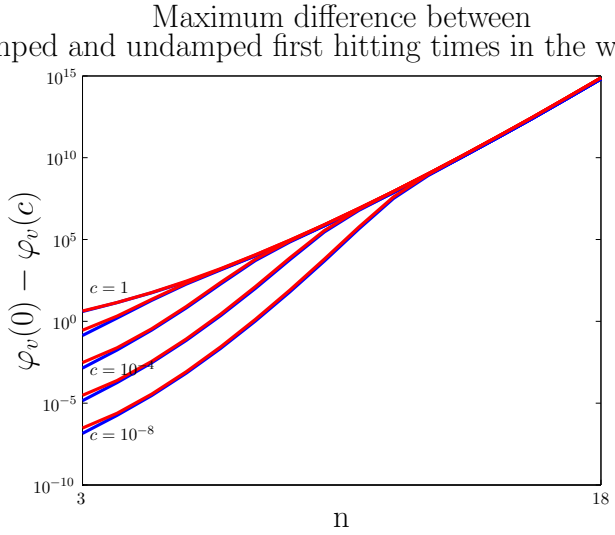

Figure B.2: The figure illustrates the maximum decrease of the first hitting times with respect to n when we add some damping to the problem. The different curves correspond to different values of the damping constant. The blue curve represents the actual decrease while the the red curve is our upper bound on that decrease. As one can see, the bound stays close but strictly superior to the actual maximum decrease.

As we can see, the error between reality (blue curve) and bound (red curve) is derisory, at least with an logarithmic scale.

Appendix C

Implementation of the algorithms

We have implemented the three algorithms mentioned in this work using Matlab[1] in order to test their behavior on several situations and to generate our experimental result. In this appendix, we give our implementation for the Linear Programming method (LP), the PageRank Iteration algorithm (PRI) and the method based on Interval Matrices (IM) as well as the necessary auxiliary function.

Note that we have used the glpk solver[2] to solve the linear programs that appear in the the IM and LP algorithms.

```
%% The Linear Programming algorithm with damping
%
% Calculates the PageRank of node v (default : v = 1). Also includes
% damping (default : c = 0, z = ones(n,1)/n). The Max-PageRank problem is
% solved if minOrMax = 1, and the Min-PageRank is solved if minOrMax = -1.
% The used algorithm is is described in the article "PageRank optimization
% by edge selection" by B.C. Csáji, R.M. Jungers and V.D. Blondel
%
% PR is the desired bound on the PageRank of v and Eopt is the optimal
% configuration of fragile edges that achives this PageRank value
function [PR, Eopt] = PRLinearProgramming(E, F, v, minOrMax, c, z)

n  = size(E,1);
nn = n+1;

if nargin ≤ 5; z = ones(n,1)/n; end
if nargin ≤ 4; c = 0;           end
if nargin ≤ 3; minOrMax = 1;    end
if nargin == 2; v = 1;           end

% Handling dangling nodes (because they may cause problems to the algorithm)
danglingI = find(sum(E-F,2) == 0);
E(danglingI,:) =  ones(length(danglingI),n);
E(danglingI, danglingI) = E(danglingI, danglingI) - eye(size(danglingI,1));
F(danglingI,:) = zeros(length(danglingI),n);

f  = sum(sum(F));
```

[1] The MathWorks : http://www.mathworks.fr/
[2] http://www.gnu.org/software/glpk/

```
% Conversion of the PageRank problem into a Stochastic Shortest Path
% problem
E = [E(:,1:v-1) zeros(n,1) E(:,v+1:n) E(:,v) ; zeros(1,nn)];
F = [F(:,1:v-1) zeros(n,1) F(:,v+1:n) F(:,v) ; zeros(1,nn)];

% Indices of fragile edges
FF = find(F);
I = mod(FF,nn);
J = ceil(FF/nn);
J(J>v) = J(J>v)-1;

% First constraint
%   x_ij - x_i ≤ 0   for all i,j s.t. i≠vt and (i,j) is fragile
AA = zeros(n, f);
AA(I'+(0:n:n*(f-1))) = 1;
AA = [-AA' zeros(f,1)];

% Second constraint
%   x_ij - xx_j ≤ 1   for all i,j s.t. i≠vt and (i,j) is fragile
BB = zeros(n, f);
BB(J'+(0:n:n*(f-1))) = 1;
BB = -BB';
BB = [BB(:,1:v-1) zeros(f,1) BB(:,v:n)];

% Additional constraints x_vt = 0 and xx_vt = 0
CC = [zeros(2,n) [-1 ; 1]];

% Third constraint
%   deg(i) x_i - sum_{(i,j) in E-F} x_j - sum_{(i,j) in F} x_ij ≤
%   sum_{(i,j) in E-F} 1   for all i≠vt
deg = sum(E,2);
deg = deg*ones(1,nn);
DD = [deg.*eye(nn) zeros(nn,f+nn+1)] - [zeros(nn) E-F zeros(nn,f+1)] + ...
     [zeros(nn,2*nn) AA' zeros(nn,1)];

% Fourth constraint
%   xx_i - (1-c)x_i - c x_q ≤ 0   for all i≠vt

% Fifth constraint
%   x_q - sum{i} z_i xx_i ≤ 1

% Construction of the system
A = [AA zeros(f,nn) eye(f) zeros(f,1) ;
     zeros(f,nn) BB eye(f) zeros(f,1) ;
     CC zeros(2,nn+f+1) ;
     zeros(2,nn) CC zeros(2,f+1) ;
     DD ;
     -(1-c)*eye(n) zeros(n,1) eye(n) zeros(n,f+1) -c*ones(n,1) ;
     zeros(1,nn+1) -z(1:n-1)' -z(n) zeros(1,f) 1];
C = ones(2*nn+f+1,1);
B = [zeros(f,1) ; minOrMax*ones(f,1) ; zeros(4,1) ; minOrMax*sum(E-F,2) ; ...
     zeros(n,1) ; minOrMax*1];

% Uncomment to change solver
ctype   = char('U'*ones(1, 2*(nn+f+2)));
vartype = char('C'*ones(1, 2*nn+f+1));

sense = -1;

xmax = minOrMax*glpk(C,A,B,[],[],ctype, vartype, sense);
```

```
% Track back the optimal configuration of the fragile edges
a = xmax(nn+1:2*nn);
b = xmax(2*nn+1:end-1);
E(FF) = 0;
J = ceil(FF/nn);
epsilon = 1e-8;
E(FF(a(J)+1 ≥ b-epsilon & a(J)+1 ≤ b+epsilon)) = 1;

Eopt = [E(1:n,1:v-1) E(1:n,nn) E(1:n,v+1:n)];

% Compute the PageRank of the nodes for that configuration
D = diag(Eopt*ones(n,1));
P = (inv(D)*Eopt)';
G = (1-c)*P + c*z*ones(1,n);
A = G-eye(n);
A = [A(1:n-1,:) ; ones(1,n)];
b = [zeros(n-1,1) ; 1];
PR = A\b;

end
```

```
% Conputes max PageRank for all nodes.
function allPR = PRLinearProgrammingAll(E, F, minOrMax, c, z)

n = size(E,1);

if nargin ≤ 4; z = ones(n,1)/n; end
if nargin ≤ 3; c = 0;           end
if nargin == 2; minOrMax = 1;    end

allPR = zeros(n,1);

for i = 1:n
    if n > 300 && mod(i,10) == 0
        fprintf([num2str(i) '/' num2str(n) '\n'])
    end
    PR = PRLinearProgramming(E, F, i, minOrMax, c, z);
    allPR(i) = PR(i);
end

end
```

```
%% PageRank Iteration algorithm with damping
%
% Calculates the PageRank of node v (default¬: v = 1). Damping is also
% included in the algorithm (default¬: c = 0, z = ones(n,1)/n). The
% Max-PageRank problem is solved if minOrMax = 1, and the Min-PageRank is
% solved if minOrMax = -1.
% The used algorithm is iterative and is described in the article "PageRank
% optimization by edge selection" by B.C. Csáji, R.M. Jungers and V.D.
% Blondel
%
% PR is the desired bound on the PageRank of v and Eopt is the optimal
% configuration of fragile edges that achives this PageRank value. k is the
% number of iterations taken by the algorithm.
```

```
%
% Warning : v must be attainable from any other node including v itself
function [PR, Eopt, k] = PRIteration(E, F, v, minOrMax, c, z)

n = size(E,1);

if nargin ≤ 5; z = ones(n,1)/n; end
if nargin ≤ 4; c = 0;           end
if nargin ≤ 3; minOrMax = 1;    end
if nargin == 2; v = 1;           end

% Handling dangling nodes (because they may cause problems to the algorithm)
danglingI = find(sum(E-F,2) == 0);
E(danglingI,:) =  ones(length(danglingI),n);
E(danglingI, danglingI) = E(danglingI, danglingI) - eye(size(danglingI,1));
F(danglingI,:) = zeros(length(danglingI),n);

k = 0;

Fk  = F;
Hkk = zeros(n,1);
Hk  = Hkk+1;

% Find the (i,j) coordinates of the fragile edges in Fk
If = find(F);
I = mod(mod(If,n)+n-1,n)+1;
J = ceil(If/n);

% Iteration

while norm(Hk - Hkk) > 1e-10

    % Compute the new first hitting time
    Hkk = Hk;
    Hk  = firstHittingTime(E, F, Fk, v, c, z);

    k = k+1;

    % Special case is needed for fragile edges that end in v : those edges
    % must consider that they arrive at destination and therefor, the first
    % hitting time of v must be 0 (this is not the case when an edge starts
    % from v because it has not returned to v yet)
    HHk = Hk;
    HHk(v) = 0;

    % Update : Take a fragile edge only if it is obviously advantageous to
    % take it (wrt the first hitting times Hk)
    update = (minOrMax*Hk(I) < minOrMax*((1-c)*HHk(J)+c*z'*HHk+1));

    Fk = F;
    Fk(If(update)) = 0;

    if isnan(0*sum(Hk))
        fprintf(['PRI algorithm did not converge. The considered graph'...
                 'could be ill conditioned.\n\n'])
        PR = NaN*zeros(n,1);
        Eopt = zeros(n);
        return;
    end
```

```
end

Eopt = E-F+Fk;
PR = 1./Hk;

end
```

```
% Conputes Max-PageRank for all nodes. k is a vector containing the number of
% iteration needed to compute each maximum PageRank.
function [allPR, k] = PRIterationAll(E, F, minOrMax, c, z)

n = size(E,1);

if nargin <= 4; z = ones(n,1)/n; end
if nargin <= 3; c = 0;           end
if nargin == 2; minOrMax = 1;    end

allPR = zeros(n,1);

k = zeros(n,1);
for i = 1:n
    [PR, Eopt, k(i)] = PRIteration(E, F, i, minOrMax, c, z);
    allPR(i) = PR(i);
end

end
```

```
% Computes bounds on the possible PageRank value of v using interval matrices.
% The damping constant can also determined. The personalization vector is fixed
% to z = ones(n,1)/n.
function [PRmin, PRmax, A, b] = PRIntervalMatrices(E, F, v, c)

if nargin <= 3; c = 0; end
if nargin == 2; v = 1; end

n = size(E,1);

[A, b, ctype, vartype] = constructIM(E, F, c);

if v < n
    c = zeros(n-1,1);
    c(v) = 1;
else
    c = ones(n-1,1);
end

[foo, PRmin] = glpk(c,A,b,[],[],ctype, vartype,  1);
[foo, PRmax] = glpk(c,A,b,[],[],ctype, vartype, -1);

if v == n
    PRmin = 1 - PRmin(n);
    PRmax = 1 - PRmax(n);
end
PRmin = PRmax;

end
```

```
% Computes bounds on the possible PageRank value of all the nodes using
% interval matrices. The damping constant can also determined. The
% personalization vector is fixed to z = ones(n,1)/n.
function [PRmin, PRmax, A, b] = PRIntervalMatricesAll(E, F, c)

if nargin == 2; c = 0; end

n = size(E,1);

[A, b, ctype, vartype] = constructIM(E, F, c);

PRmin = zeros(n,1);
PRmax = zeros(n,1);

for i = 1:n-1

    c = zeros(n-1,1);
    c(i) = 1;

    [foo, PRmin(i)] = glpk(c,A,b,[],[],ctype, vartype,  1);
    [foo, PRmax(i)] = glpk(c,A,b,[],[],ctype, vartype, -1);

end

c = ones(n-1,1);

[foo, PRmin(n)] = glpk(c,A,b,[],[],ctype, vartype, -1);
[foo, PRmax(n)] = glpk(c,A,b,[],[],ctype, vartype,  1);

PRmin(n) = 1 - PRmin(n);
PRmax(n) = 1 - PRmax(n);

end
```

```
function [A, b, ctype, vartype] = constructIM(E, F, c)

n = size(E,1);

% Handling dangling nodes (because they may cause problems to the algorithm)
danglingI = find(sum(E-F,2) == 0);
E(danglingI,:) =  ones(length(danglingI),n);
E(danglingI, danglingI) = E(danglingI, danglingI) - eye(size(danglingI,1));
F(danglingI,:) = zeros(length(danglingI),n);

degE = sum(E,2)*ones(1,n);
degF = sum(F,2)*ones(1,n);
AA = (E ./ degE)';

Amax = (degE.*E - degF.*E + F)';
Amax_ind = find(Amax);
Amax(Amax_ind) = 1./Amax(Amax_ind);

Amin = AA;
Amin(F'==1) = 0;
```

```
Mmax = (1-c)*Amax + c/n*ones(n);
Mmin = (1-c)*Amin + c/n*ones(n);

Mc    = (Mmax + Mmin)/2;
Delta = (Mmax - Mmin)/2;

Bmax = Mc + Delta - eye(n);
Bmin = Mc - Delta - eye(n);

EE = [eye(n-1) ; -ones(1,n-1)];
g  = [zeros(n-1,1) ; 1];

% Construction of the system
A = [-Bmax*EE ;  Bmin*EE ; ones(1,n-1) ; -eye(n-1)   ];
b = [ Bmax*g  ; -Bmin*g  ; 1           ; zeros(n-1,1)];

A(abs(A) < 1e-15) = 0;
b(abs(b) < 1e-15) = 0;

ctype   = char('U'*ones(3*n, 1));
vartype = char('C'*ones(n-1, 1));

end
```

```
% Calculates the mean first hitting time Hk of node v by all the nodes i of
% the graph constituted by edges E-F+Fk
function Hk = firstHittingTime(E, F, Fk, v, c, z)

n = size(E,1);
one = ones(n,1);

if nargin <= 5; z = ones(n,1)/n; end
if nargin <= 4; c = 0;           end

% Adjacency matrix of the considered graph
Pk = E - F + Fk;
% Degrees matrix of the nodes
Dk = sum(Pk,2)*one';
% Define node v as the target node
Pk(:,v) = 0;

% Matrix to handle damping
Qk = one*z';
Qk(:,v) = 0;

% Compute the first hitting time
Hk = (eye(n) - (1-c)*(Pk ./ Dk) - c*Qk) \ ((1-c)*one + c*one*z'*one);

end
```

Bibliography

[ACL04] R. Andersen, F. Chung, and L. Lu. Drawing power law graphs. *Proceedings of the 12th Symposium on Graph Drawing (GD'04)*, 2004.

[AL04] K. Avrachenkov and N. Litvak. Decomposition of the google pagerank and optimal linking strategy. *Technical report, INRIA*, 2004.

[Ber96] D. P. Bertsekas. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, Massachusetts, 1996.

[Ber05] P. Berkhin. A survey on pagerank computing. *Internet Mathematics*, 2(1):73-120, 2005.

[BT91] D. P. Bertsekas and J. N. Tsitsiklis. An analysis of stochastic shortest path problems. *Mathematics of Operations Research*, 16(3):580-595, 1991.

[CJB09] B. C. Csáji, R. M. Jungers, and V. D. Blondel. Pagerank optimization by edge selection. *(Submitted)*, 2009.

[CL01] F. Chung and L. Lu. The diameter of random sparse graphs. *Advances in Applied Math*, 26:257–279, 2001.

[CW90] D. Coppersmith and S Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251-280, 1990.

[D'E63] F. D'Epenoux. *A probabilistic production and inventory problem*, volume 10:98-108. Management Science, 1963.

[Doo08] P. Van Dooren. *Algorithmique numérique (INMA2710 course notes)*. Ecole Polytechnique de Louvain, Université Catholique de Louvain (UCL), 2008.

[Fea10] J. Fearnley. Exponential lower bounds for policy iteration. *CoRR, abs/1003.3418*, 2010.

[Gon88] C. C. Gonzaga. *Progress in Mathematical Programming: Interior-Point and Related Methods*. Springer-Verlag, 1988.

[Had93] J. Hadamard. Résolution d'une question relative aux déterminants. *Bull. Sci. Math*, 28:240-246, 1893.

[IT09] H. Ishii and R. Tempo. Computing the pagerank variation for fragile web data. *SICE J. of Control, Measurement, and System Integration*, 2(1):1-9, 2009.

[Kar84] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373-395, 1984.

[KND08] C. De Kerchove, L. Ninove, and P. Van Dooren. Maximizing pagerank via outlinks. *Linear Algebra and its Applications*, 429:1254-1276, 2008.

[LDK95] M. L. Littman, T. L. Dean, and L. P. Kaelbling. On the complexity of solving markov decision problems. *In Proc. of the Eleventh International Conference on Uncertainty in Artificial Intelligence*, pages 394–402, 1995.

[MC90] M. Melekepoglou and A. Condon. On the complexity of the policy iteration algorithm for stochastic games. *Operations Research Society of America (ORSA) Journal on Computing*, Vol. 6, No. 2, 1990.

[MS99] Y. Mansour and S. Singh. On the complexity of policy iteration. *Fifteenth Conference on Uncertainty in Artificial Intelligence*, 1999.

[MST04] M. Mihail, A. Saberi, and P. Tetali. Random walks with lookahead in power law random graphs. *Internet Mathematics*, 2004.

[PT87] C. H. Papadimitriou and J. N. Tsitsiklis. The complexity of markov decision processes. *Mathematics of Operations Research*, 12(3):441-450, 1987.

[Put94] M. L. Puterman. *Markov decision processes.* John Wiley & Sons, 1994.

[SB98] R. S. Sutton and A. G. Barto. *Reinforcement learning.* The MIT Press, 1998.

[Tse90] P. Tseng. Solving h-horizon, stationary markov decision problems in time proportional to log(h). *Operations Research Letters*, 9(4):287-297, 1990.

[VG09] R. O'Donnell V. Guruswami. Intensive intro to complexity theory : Expander graphs. 2009.

[Ye10] Y. Ye. The simplex method is strongly polynomial for the markov decision problem with a fixed discount rate. *(Submitted)*, 2010.