Probabilistic Transitive Closure of Fuzzy Cognitive Maps: Algorithm Enhancement and an Application to Work-Integrated Learning

Masoomeh Akbari

Thesis submitted to the Faculty of Science in partial fulfillment of the requirements
for the degree of
Master of Science Mathematics and Statistics[1]

Department of Mathematics and Statistics
Faculty of Science
University of Ottawa

[1]The M.Sc. program is a joint program with Carleton University, administered by the Ottawa-Carleton Institute of Mathematics and Statistics

Abstract

A *fuzzy cognitive map* (FCM) is made up of factors and direct impacts. In graph theory, a bipolar weighted digraph is used to model an FCM; its vertices represent the factors, and the arcs represent the direct impacts. Each direct impact is either positive or negative, and is assigned a weight; in the model considered in this thesis, each weight is interpreted as the probability of the impact. A directed walk from factor F to factor F' is interpreted as an indirect impact of F on F'. The *probabilistic transitive closure* (PTC) of an FCM (or bipolar weighted digraph) is a bipolar weighted digraph with the same set of factors, but with arcs corresponding to the indirect impacts in the given FCM.

Fuzzy cognitive maps can be used to represent structured knowledge in diverse fields, which include science, engineering, and the social sciences. In [P. Niesink, K. Poulin, M. Šajna, Computing transitive closure of bipolar weighted digraphs, *Discrete Appl. Math.* **161** (2013), 217–243], it was shown that the transitive closure provides valuable new information for its corresponding FCM. In particular, it gives the total impact of each factor on each other factor, which includes both direct and indirect impacts. Furthermore, several algorithms were developed to compute the transitive closure of an FCM. Unfortunately, computing the PTC of an FCM is computationally hard and the implemented algorithms are not successful for large FCMs. Hence, the Reduction-Recovery Algorithm was proposed to make other (direct) algorithms more efficient. However, this algorithm has never been implemented before.

In this thesis, we code the Reduction-Recovery Algorithm and compare its running time with the existing software. Also, we propose a new enhancement on the existing PTC algorithms, which we call the Separation-Reduction Algorithm. In particular, we state and prove a new theorem that describes how to reduce the input digraph to smaller components by using a separating vertex. In the application part of the thesis, we show how the PTC of an FCM can be used to compare different standpoints on the issue of work-integrated learning.

Dedication

I would like to dedicate my thesis with love and gratitude to:

- My sweet and loving parents, who are the heroes of my life. No matter how far I am from home, their days and nights' prayer gives me the strength and determination to fulfill my dreams.
- My respected supervisor, Prof. Šajna, for her great support and continuous encouragement.

Acknowledgement

Words simply cannot express my sincere appreciation to my supervisor Prof. Šajna, whose expertise, valuable guidance, and extraordinary support helped me accomplish this thesis. She is not only my academic supervisor but also a great friend. I can remember times when I felt discouraged and unsure about my work. It was her endless inspiration and support that enabled me to overcome the entire problem. Studying at a graduate level was a dream for me that came true because of Prof. Šajna.

The internship opportunity that I had with Shopify was a great chance for learning and developing my programming skills. I wish to express my sincere gratitude to Mitacs for financial support, and to Shopify for financial support and hospitality.

Finally, my heartfelt thanks to my caring, loving, and supportive family, especially my beloved parents, who are the reason that I keep going and overcoming struggles and hardships.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Fuzzy cognitive maps (FCMs) can serve as effective tools to study problems in various fields. They can be used to represent structured knowledge in science, engineering, and the social sciences. FCMs make it easier to organize the information related to a problem. Hence, researchers are interested in using FCMs for their modeling. FCMs can be used to model complex system in education, economics, communication network, and management. For example, in [3], FCMs were used to understand the cognitive mechanisms that affect errors happening during the performance of drug management activities. FCMs were also used to represent traditional knowledge on determinants of diabetes in first nations communities in Canada [8]. In [2], FCMs were used in medical decision-support systems (MDSSs) to assist physicians in decision-making. These are just a few examples of hundreds of applications of FCMs in the real world.

An FCM consists of factors and direct impacts, which are relationships between pairs of factors. Each direct impact is assigned a weight in $[0, 1]$ as well as a sign (positive or negative). In graph theory, a bipolar weighted digraph is used to model an FCM; its vertices represent the factors, and the arcs represent the direct impacts. In [16], it was shown that transitive closure (TC) of an FCM gives additional valuable information about the corresponding FCM. Hence, two models were introduced and developed for defining the TC of an FCM, the fuzzy model and the probabilistic model. The latter model is more suitable for practical applications and we also consider it in this thesis.

In the probabilistic model, the weight of each impact is interpreted as the probability of that impact. Moreover, a directed walk from factor F to factor F' is interpreted as an indirect impact of F on F'. In the probabilistic model, the TC of a bipolar weighted digraph is defined as a bipolar weighted digraph with the same set of factors (vertices), but with arcs corresponding to the indirect impacts in the given bipolar weighted digraph. The weight of an arc (F, F') is the probability of the event that the FCM has a directed walk from F to F' of the specified sign (where the sign

of a walk is the product of the signs of its arcs).

In [16], several algorithms were proposed for the computation of the PTC of a bipolar weighted digraph, namely, the Complete State Enumeration (CSE) Algorithm, the Principle of Inclusion-Exclusion (PIE), the bundles version of the PIE algorithm, and the Boolean Algebra Algorithm (both the exact version and the approximative version). These algorithms were coded in [15] and compared; of these algorithms, the Boolean Algebra approach was usually the most successful.

Furthermore, the Reduction-Recovery Algorithm was proposed as an algorithmic enhancement for the computation of PTC. The Reduction-Recovery Algorithm reduces the input digraph recursively by deleting vertices of indegree at most one and outdegree at most one, and then recovers the PTC of the input digraph from the PTC of the fully reduced digraph. However, the Reduction-Recovery Algorithm as described in [16] does not specify the form of recursion, and it has never been coded before.

In Chapter 4 of this thesis, we present two pseudocode versions of the Reduction-Recovery Algorithm. Moreover, we implement these two approaches in Python and compare them. We also provide the comparison of the running times of our implementation of the Reduction-Recovery Algorithm with the existing software, namely ProbTC [13], and show that the enhanced code is at least as efficient, and often more efficient, than ProbTC.

In Chapter 5, we use the idea of the reduction of the input digraph to propose a new enhancement of the algorithms for computing PTC. We state and prove a theorem that shows how one can use a separating vertex to reduce the input digraph into smaller components, and how to recover the PTC of the original digraph from the PTCs of the smaller components. This procedure may be applied recursively. We also provide a pseudocode algorithm for this new enhancement, which we call the Separation-Reduction Algorithm.

In Chapter 6, as an application of FCMs and PTC, we provide insightful analysis into the determinants of success and well-being of a student intern at Shopify. For this purpose, we asked each group of participants (the student group, team group, and mentor group) to create its FCM during a group mapping session. For each FCM, the net causal effect of every determinant on success and well-being was obtained from its PTC. Then the net causal effects were compared across the three FCMs in two different ways to identify strong, weak, and controversial determinants. This work demonstrates how FCMs can be used to represent different standpoints on an issue (in this case) of success and well-being of a Shopify student intern, and how these different opinions can be compared via probabilistic transitive closure. As far as we know, this is the first study ever to address work-integrated learning using cognitive mapping.

Chapter 2

Prerequisites

2.1 Graphs

In this section, we review the basic notions of graphs. These notions are taken from [4].

Definition 2.1.1. A *graph* G is an ordered pair (V, E) with an incidence function ψ_G which has the following properties:

- $V \neq \emptyset$,
- $V \cap E = \emptyset$, and
- $\psi_G : E \to \{\{u, v\} : u, v \in V\}$.

The elements of V are called *vertices*, and the elements of E are called *edges*, so $V = V(G)$ is the *vertex set*, and $E = E(G)$ is the *edge set* of G, and the function ψ_G associates an unordered pair of vertices to each edge.

A graph G is called an *empty graph* if its edge set is empty; that is $E(G) = \emptyset$. Otherwise the graph is *non-empty*.

Let $G = (V, E)$ be a graph, and $u, v \in V$, $e \in E$ such that $\psi_G(e) = \{u, v\}$.

- Vertices u and v are *joined* by e, and they are called the *ends* of e.
- Vertices u and v are said to be *incident* with e, and e is said to be *incident* with vertices u and v.
- Vertices u and v are called *adjacent*.
- If $u = v$, then e is called a *loop*.
- If $u \neq v$, then e is called a *link*.

For simplicity, we write uv instead of $\{u, v\}$. Distinct links with the same ends are called *parallel edges*. A graph without loops or parallel edges is called a *simple graph*. If e, f are two edges of G which are incident with a common vertex, then e and f are called *adjacent*.

Whenever possible without ambiguity, we write $e = uv$ instead of $\psi_G(e) = uv$ (for example, in simple graphs, or when we do not need to distinguish between parallel edges). In other words, in these cases, the incidence function may be omitted.

Definition 2.1.2. The *degree* of a vertex u in a graph G, denoted by $d_G(v)$, is defined as the number of edges of G that are incident with u, except that each loop is counted twice.

Definition 2.1.3. Let G be a graph with at least two vertices. The operation of deleting a vertex u, together with all edges incident with u, from G is called *vertex deletion*, and the graph obtained by this operation is denoted by $G - u$.

Definition 2.1.4. A graph H is a *subgraph* of a graph G (denoted by $H \subseteq G$), if it satisfies the following properties:

- The vertex set of H is a subset of the vertex set of G, that is, $V(H) \subseteq V(G)$.
- The edge set of H is a subset of the edge set of G, that is, $E(H) \subseteq E(G)$.
- The incidence function of H, ψ_H, is the restriction of the incidence function of G, ψ_G, to the edge set of H.

The *union* of graphs G_1 and G_2 is the graph $G_1 \cup G_2$ with $V(G_1) \cup V(G_2)$ as the vertex set, and $E(G_1) \cup E(G_2)$ as the edge set. Furthermore, if G_1 and G_2 have no vertex in common (that is, the vertex sets of G_1 and G_2 are disjoint), then $G_1 \cup G_2$ is a *disjoint union*, and is denoted by $G_1 \dot{\cup} G_2$. The *intersection* of graphs G_1 and G_2 is the graph $G_1 \cap G_2$ with $V(G_1) \cap V(G_2)$ as the vertex set, and $E(G_1) \cap E(G_2)$ as the edge set.

Definition 2.1.5. Let $U \subseteq V(G)$. The *induced subgraph* $G[U]$ is the subgraph of G with vertex set U and edge set

$$E(G[U]) = \{uv \in E(G) : u, v \in U\}.$$

Definition 2.1.6. A *walk* in a graph G is a sequence $W = u_0 e_1 u_1 \dots u_{k-1} e_k u_k$ such that:

- $u_0, u_1, \dots, u_k$ are in V,
- $e_1, e_2, \dots, e_k$ are in E, and
- $e_i = u_{i-1} u_i$, for $1 \leq i \leq k$.

If $u_0 = u$ and $u_k = v$, then W *connects* u to v, and we refer to W as a *(u,v)-walk*. If $u = v$ and $k \geq 1$, then W is called a *closed walk*.

The walk W is called a *path* if the vertices $u_0, u_1, \ldots, u_k$ are pairwise distinct; in other words, no two vertices in the sequence are the same. The walk W is called a *cycle* if it is closed, the vertices $u_0, u_1, \ldots, u_{k-1}$ are pairwise distinct, and the edges $e_1, e_2, \ldots, e_k$ are pairwise distinct.

Definition 2.1.7. A graph G is *connected* if for all $u, v \in V(G)$, there is a (u, v)-path (or equivalently, a (u, v)-walk) in G.

A graph which is a disjoint union of two graphs is called *disconnected*. Any graph G can be represented uniquely as a disjoint union of connected graphs, which are called the *connected components* (or simply *components*) of G. The number of connected components of G is denoted by $c(G)$.

2.2 Directed Graphs

In this section, we review the basic notions of directed graphs, which are taken from [4].

Definition 2.2.1. A *directed graph* (*digraph*) D is an ordered pair (V, A) with an incidence function ψ_D which has the following properties:

- $V \neq \emptyset$,
- $V \cap A = \emptyset$, and
- $\psi_D : A \to \{(u, v) : u, v \in V\}$.

The elements of V are called *vertices*, and the elements of A are called *arcs*, so $V = V(D)$ is the *vertex set*, and $A = A(D)$ is the *arc set* of D, and the function ψ_D associates an ordered pair of vertices to each arc.

A digraph D is called an *empty digraph* if its arc set is empty; that is $A(D) = \emptyset$. Otherwise the digraph is *non-empty*.

Let $D = (V, A)$ be a digraph, and $u, v \in V$, $a \in A$ such that $\psi_D(a) = (u, v)$. The following terminology will be used.

- Vertex u is *joined* to vertex v by a.
- Vertex u is called the *tail,* and vertex v is called the *head* of the arc a.
- If $u = v$, then a is called a *directed loop*.

- We say that u *dominates* v.
- The *in-neighbors* of u are the vertices that dominate u.
- The *out-neighbors* of u are the vertices which are dominated by u.
- The number of arcs with head u is the *indegree* of u.
- The number of arcs with tail u is the *outdegree* of u.

Distinct arcs with the same tail and the same head are called *parallel arcs*. Whenever possible without ambiguity, we write $a = (u, v)$ instead of $\psi_D(a) = (u, v)$.

Definition 2.2.2. A graph G which is obtained from a digraph D by replacing each arc (u, v) with an edge uv is called the *underlying undirected* graph of D, and denoted by $G(D)$.

Definition 2.2.3. A digraph D' is a *subdigraph* of a digraph D (denoted by $D' \subseteq D$), if it satisfies the following properties:

- The vertex set of D' is a subset of the vertex set of D, that is, $V(D') \subseteq V(D)$.
- The arc set of D' is a subset of the arc set of D, that is, $A(D') \subseteq A(D)$.
- The incidence function of D', $\psi_{D'}$, is the restriction of the incidence function of D, ψ_D, to the arc set of D'.

A digraph $D = (V, A)$ is said to be *strongly connected* if for any two vertices $u, v \in V$ there exists both a directed (u, v)-walk and a directed (v, u)-walk. A *strong component* of a digraph D is a maximal strongly connected subdigraph of D. A digraph D is *weakly connected* if its underlying undirected graph is connected.

The *union* of digraphs D_1 and D_2 is the digraph $D_1 \cup D_2$ with $V(D_1) \cup V(D_2)$ as the vertex set, and $A(D_1) \cup A(D_2)$ as the arc set. Furthermore, if D_1 and D_2 have no vertex in common (that is, the vertex sets of D_1 and D_2 are disjoint), then $D_1 \cup D_2$ is a *disjoint union*, and is denoted by $D_1 \dot{\cup} D_2$. The *intersection* of digraphs D_1 and D_2 is the digraph $D_1 \cap D_2$ with $V(D_1) \cap V(D_2)$ as the vertex set, and $A(D_1) \cap A(D_2)$ as the arc set.

Definition 2.2.4. Let $U \subseteq V(D)$. The *induced subdigraph* $D[U]$ is the subdigraph of D with vertex set U, and arc set

$$A(D[U]) = \{a \in A(D) : a = (u, v), u, v \in U\}.$$

Definition 2.2.5. A *directed walk* in a digraph D is a sequence $W = u_0 a_1 u_1 \dots u_{k-1} a_k u_k$ such that:

- $u_0, u_1, \ldots, u_k$ are in V,
- $a_1, a_2, \ldots, a_k$ are in A, and
- $a_i = (u_{i-1}, u_i)$, for $1 \leq i \leq k$.

If $u_0 = u$ and $u_k = v$, then W *connects* u to v, and we refer to W as a *directed (u,v)-walk*. If $u = v$ and $k \geq 1$, then W is called a *directed closed walk*.

The directed walk W is called a *directed path* if the vertices $u_0, u_1, \ldots, u_k$ are pairwise distinct; in other words, no two vertices in the sequence are the same. The directed walk W is called a *directed cycle* if it is closed and the vertices $u_0, u_1, \ldots, u_{k-1}$ are pairwise distinct.

2.3 Probability

In this section, we recall the notion of a probability space.

Definition 2.3.1. A *finite probability space* is a triple $(\mathcal{D}, \mathcal{E}, P)$ satisfying the following assumptions.

- $\mathcal{D}$ is a finite set, called the *sample set*.
- $\mathcal{E}$ is a subset of $2^{\mathcal{D}}$ such that:
  - $\mathcal{D} \in \mathcal{E}$,
  - $\mathcal{E}$ is closed under complements, and
  - $\mathcal{E}$ is closed under unions.

  The elements of $\mathcal{E}$ are called *events*.
- $P : \mathcal{E} \to [0, 1]$ is a probability function satisfying
  - $P(\mathcal{D}) = 1$; and
  - for any mutually disjoint events $A_1, A_2, \ldots, A_k \in \mathcal{E}$, we have

$$P(\cup_{i=1}^{k} A_i) = \sum_{i=1}^{k} P(A_i).$$

2.4 Fuzzy Cognitive Maps

Fuzzy cognitive maps (FCMs) can be used to represent structured knowledge in various fields of science, engineering, and the social sciences. In [8], for example, FCMs were used to represent traditional knowledge on determinants of diabetes in first nations communities in Canada. Many more applications of FCMs can be found in [1, 2, 3, 7, 10, 11, 12, 17, 18, 19, 23, 24, 25, 26, 27]. In this thesis we use FCMs to study the issue of work-integrated learning for Shopify interns.

A *fuzzy cognitive map* is made up of factors and direct impacts, which are relationships between pairs of factors. A factor F can have either a direct positive or a direct negative impact on factor F' (or on itself). Each direct impact is assigned a weight in the interval $[0, 1]$.

In [16], as well as in this thesis, the following assumptions on the interpretation of a direct impact are made. A direct positive impact of factor F on factor F' means either

- (First interpretation) F causes F', and not F causes not F', or
- (Second interpretation) an increase in F causes an increase in F', and a decrease in F causes a decrease in F'.

Similarly, a direct negative impact of factor F on factor F' means either

- (First interpretation) F causes not F', and not F causes F', or
- (Second interpretation) an increase in F causes a decrease in F', and a decrease in F causes an increase in F'.

Figure 2.1 represents a simple FCM: ovals represent the factors, and black and red arrows represent the direct positive and direct negative impacts, respectively. The weight of each impact is given as the number closest to the corresponding direct impact.

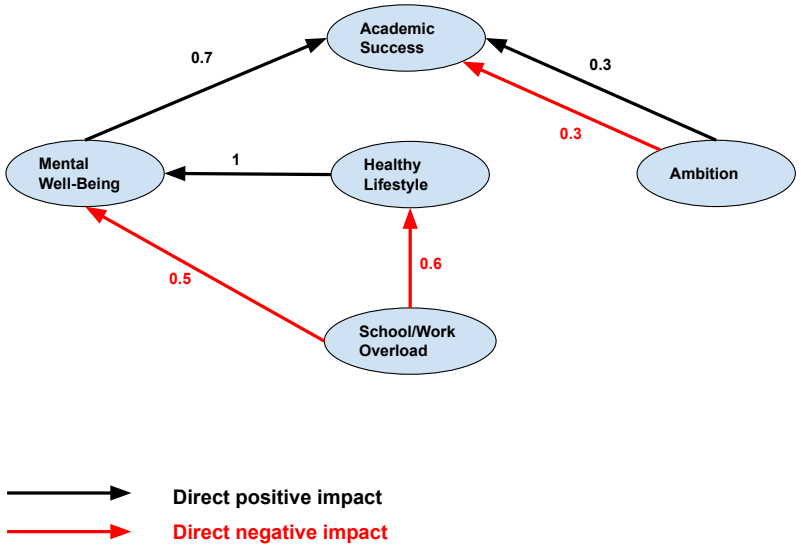

Figure 2.1: A simple example of a fuzzy cognitive map (FCM)

Chapter 3

Previous Work

3.1 Bipolar Weighted Digraphs and Bipolar Random Digraphs

In this section, we review the notions of bipolar weighted digraphs and bipolar random digraphs. These notions were introduced in [16]. Throughout this thesis, we denote the set $\{-1, 1\}$ by Ω.

Definition 3.1.1. A digraph $D = (V, A)$ with incident function ψ is said to be a *bipolar digraph* if it has the following properties:

- $A \subseteq V^2 \times \Omega$, and
- $\psi(u, v, \sigma) = (u, v)$, for all $u, v \in V$ and $\sigma \in \Omega$.

Figure 3.1 represents a bipolar digraph: black and red arrows represent the positive and negative arcs, respectively.

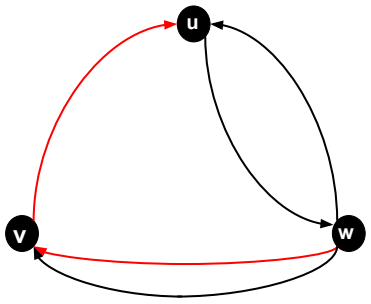

Figure 3.1: A bipolar digraph

In a bipolar digraph (V, A), an arc a is an ordered triple (u, v, σ), and the *sign* of this arc is defined as $\text{sign}(a) = \sigma$, where $\sigma \in \Omega$. The *sign* of a directed walk $W = u_0a_1u_1a_2u_2 \dots u_{k-1}a_ku_k$ in a bipolar digraph (V, A) is defined as the product of the signs of the arcs in its sequence. In other words, $\text{sign}(W) = \text{sign}(a_1)\text{sign}(a_2) \dots \text{sign}(a_k)$. Furthermore, a directed (u, v)-walk of sign σ is called a directed (u, v, σ)-walk.

Let $W' = ua_1u_1 \dots u_{m-1}a_mv$ and $W'' = va'_1u'_1 \dots u'_{n-1}a'_nz$ be a directed (u, v)-walk and a directed (v, z)-walk, respectively. The *concatenation* of W' and W'' is a directed (u, z)-walk $W'W'' = ua_1u_1 \dots a_mva'_1 \dots u'_{n-1}a'_nz$. It can be easily seen that $\text{sign}(W'W'') = \text{sign}(W')\text{sign}(W'')$.

Definition 3.1.2. A *bipolar weighted digraph* $D = (V, A, w)$ is a bipolar digraph (V, A) with a weight function $w : A \to [0, 1]$.

Figure 3.2 represents a bipolar weighted digraph. Black and red arrows represent the positive and negative arcs, respectively. The weight of each arc is given as the number closest to that arc.

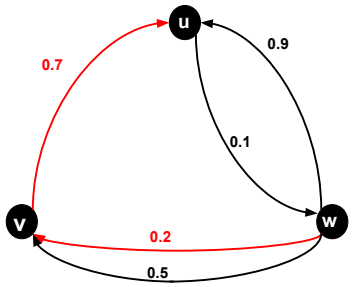

Figure 3.2: A bipolar weighted digraph

A fuzzy cognitive map (FCM) can be modeled by a bipolar weighted digraph; its factors are represented by the vertices, and the direct impacts are represented by the arcs of the bipolar weighted digraph. The direct positive and direct negative impacts are represented by the arcs of positive and negative signs, respectively.

Let D be a bipolar weighted digraph modeling an FCM. We say that factor F *impacts* factor F' in the FCM if there is a directed (F, F')-walk W in D. In other words, a directed walk W from factor F to factor F' is interpreted as an (indirect) impact of F on F'. If $\text{sign}(W) = 1$, then this impact is said to be positive, and if $\text{sign}(W) = -1$, then this impact is negative.

The following two assumptions regarding an FCM justify the definition of the sign of a directed walk.

1. A direct positive impact of factor F on factor F' means that F causes F' and not F causes not F', or that an increase in F causes an increase in F', and a

decrease in F causes a decrease in F'. A direct negative impact of factor F on factor F' means that F causes not F', and not F causes F', or (in the second interpretation) that an increase in F causes a decrease in F', and a decrease in F causes an increase in F'.

2. The relation "impact" is transitive. That is, if F has an impact on F', and F' has an impact on F'', then F has an impact on F''. To determine the sign of the impact of F on F'', we use first assumption. More precisely, suppose F, F', and F'' are three factors of an FCM.

   - If F has a positive impact on F' and F' has a positive impact on F'', then F has a positive impact on F''.
   - If F has a negative impact on F' and F' has a negative impact on F'', then F has a positive impact on F''.
   - If F has a positive impact on F' and F' has a negative impact on F'', or vice-versa, then F has a negative impact on F''.

Definition 3.1.3. A *bipolar random digraph* R is an ordered pair (V, p) which has the following properties:

- V is a finite non-empty set of vertices, and
- $p : V^2 \times \Omega \to [0, 1]$.

To R we associate a probability space $(\mathcal{D}, \mathcal{E}, P)$ as follows.

- $\mathcal{D}$ is the sample set consisting of all bipolar digraphs (V, A) with $A \subseteq V^2 \times \Omega$,
- $\mathcal{E}$ is the event set consisting of all subsets of $\mathcal{D}$.
- The function $P : \mathcal{E} \to [0, 1]$ is defined as follows:

$$P(\{D_1, \ldots, D_k\}) = \sum_{i=1}^{k} \prod_{a \in A_i} p(a) \prod_{a \in V^2 \times \Omega - A_i} (1 - p(a)),$$

where $D_1, \ldots, D_k \in \mathcal{D}$ are such that for all $i \in \{1, 2, \ldots, k\}$, $D_i = (V, A_i)$ for some $A_i \subseteq V^2 \times \Omega$. In addition, we define $P(\emptyset) = 0$.

In Corollary 3.1.5, we show that $(\mathcal{D}, \mathcal{E}, P)$ indeed satisfies the requirements for a probability space. But first we need to prove the following lemma.

Lemma 3.1.4. *[16]*

(i) *If $E_1, E_2 \in \mathcal{E}$ are disjoint events, then $P(E_1 \cup E_2) = P(E_1) + P(E_2)$.*

(ii) *For any* $S \subseteq V^2 \times \Omega$,

$$\sum_{A \subseteq S} \prod_{a \in A} p(a) \prod_{a \in S - A} (1 - p(a)) = 1.$$

Proof: To prove **(i)**, assume $E_1 = \{D_1, \ldots, D_m\}$ and $E_2 = \{D_{m+1}, \ldots, D_n\}$ where $D_1, \ldots, D_m, D_{m+1}, \ldots, D_n$ are distinct digraphs in $\mathcal{D}$, and $D_i = (V, A_i)$ for all $i \in \{1, 2, \ldots, n\}$. Then, by Definition 3.1.3, we have

$$\begin{aligned}
P(E_1 \cup E_2) &= P(\{D_1, \ldots, D_m, D_{m+1}, \ldots, D_n\}) \\
&= \sum_{i=1}^{n} \prod_{a \in A_i} p(a) \prod_{a \in V^2 \times \Omega - A_i} (1 - p(a)) \\
&= \sum_{i=1}^{m} \prod_{a \in A_i} p(a) \prod_{a \in V^2 \times \Omega - A_i} (1 - p(a)) + \sum_{i=m+1}^{n} \prod_{a \in A_i} p(a) \prod_{a \in V^2 \times \Omega - A_i} (1 - p(a)) \\
&= P(\{D_1, \ldots, D_m\}) + P(\{D_{m+1}, \ldots, D_n\}) \\
&= P(E_1) + P(E_2).
\end{aligned}$$

To prove statement **(ii)**, we use induction on $|S|$. Let $P(n)$ be the statement that **(ii)** holds for all $S \subseteq V^2 \times \Omega$ such that $|S| = n$.

1. **Base Case:** For the case $|S| = 0$, we have

$$\begin{aligned}
\sum_{A \subseteq S} \prod_{a \in A} p(a) \prod_{a \in S - A} (1 - p(a)) &= \prod_{a \in \emptyset} p(a) \prod_{a \in \emptyset} (1 - p(a)) \\
&= 1 \cdot 1 = 1.
\end{aligned}$$

2. **Induction Step:** To prove $P(k)$ implies $P(k+1)$, for all $k \geq 0$, fix any $k \geq 0$, and assume $P(k)$ is true. Let a' be a fixed arc in S. We have

$$\begin{aligned}
&\sum_{A \subseteq S} \prod_{a \in A} p(a) \prod_{a \in S - A} (1 - p(a)) \\
&= \sum_{\substack{A \subseteq S \\ a' \in A}} \prod_{a \in A} p(a) \prod_{a \in S - A} (1 - p(a)) + \sum_{\substack{A \subseteq S \\ a' \notin A}} \prod_{a \in A} p(a) \prod_{a \in S - A} (1 - p(a)) \\
&= p(a') \sum_{A \subseteq S - \{a'\}} \prod_{a \in A} p(a) \prod_{a \in S - \{a'\} - A} (1 - p(a)) + \\
&\quad (1 - p(a')) \sum_{A \subseteq S - \{a'\}} \prod_{a \in A} p(a) \prod_{a \in S - \{a'\} - A} (1 - p(a)) \\
&= \Big(p(a') + 1 - p(a')\Big) \sum_{A \subseteq S - \{a'\}} \prod_{a \in A} p(a) \prod_{a \in S - \{a'\} - A} (1 - p(a))
\end{aligned}$$

$$= \sum_{A \subseteq S-\{a'\}} \prod_{a \in A} p(a) \prod_{a \in S-\{a'\}-A} (1-p(a)) = 1.$$

The last equality holds by the induction hypothesis as the size of $S - \{a'\}$ is k.

3. **Conclusion:** Since the equality holds for $P(0)$, and $P(k)$ implies $P(k+1)$, for all $k \geq 0$, we can conclude $P(n)$ holds for all $n \geq 0$.

∎

Corollary 3.1.5. *Let (V, p) be a bipolar random digraph, and $(\mathcal{D}, \mathcal{E}, P)$ the associated triple from Definition 3.1.3. Then $(\mathcal{D}, \mathcal{E}, P)$ is a finite probability space.*

Proof: We show that $(\mathcal{D}, \mathcal{E}, P)$ satisfies the requirements in Definition 2.3.1. Since V is finite and there are only finitely many subsets of $V^2 \times \Omega$, observe that $\mathcal{D}$ is a finite set. The set $\mathcal{E}$ also satisfies all required properties as shown below.

- The event set $\mathcal{E}$ consists of all subsets of $\mathcal{D}$, so $\mathcal{D} \in \mathcal{E}$.
- The complement of a set of bipolar digraphs is a set of bipolar digraphs. Hence, $\mathcal{E}$ is closed under complements.
- The event set $\mathcal{E}$ is also closed under unions as the union of a set of bipolar digraphs is a set of bipolar digraphs.

The following properties, which show that P is a probability measure will follow easily from Lemma 3.1.4, as shown below.

1. For any mutually disjoint events $E_1, E_2, \ldots, E_k \in \mathcal{E}$, we have

$$P(\cup_{i=1}^{k} E_i) = \sum_{i=1}^{k} P(E_i); \text{ and}$$

2. $P(\mathcal{D}) = 1$.

In Lemma 3.1.4**(i)**, we proved (1) for two disjoint events. We can easily extend it to any finite union of disjoint events. To prove $P(\mathcal{D}) = 1$, let $\mathcal{D} = \{D_1, D_2, \ldots, D_N\}$ where $D_i = (V, A_i)$, for all $i \in \{1, 2, \ldots, N\}$. Then we have

$$\begin{aligned} P(\mathcal{D}) &= P(\{D_1, D_2, \ldots, D_N\}) \\ &= \sum_{i=1}^{N} \prod_{a \in A_i} p(a) \prod_{a \in V^2 \times \Omega - A_i} (1-p(a)) \end{aligned}$$

$$= \sum_{A \subseteq V^2 \times \Omega} \prod_{a \in A} p(a) \prod_{a \in V^2 \times \Omega - A} (1 - p(a))$$
$$= 1.$$

The last equality follows from Lemma 3.1.4**(ii)**. ∎

We now explain the connection between the concepts of a bipolar random digraph and bipolar weighted digraph.

Let $R = (V, p)$ be a bipolar random digraph. The *support digraph* of R, denoted $\text{supp}(R)$, is the bipolar weighted digraph (V, A, p) where $A = \{a \in V^2 \times \Omega : p(a) > 0\}$. Conversely, a bipolar random digraph (V, p) can be associated to any bipolar weighted digraph (V, A, w) if we define the function $p : V^2 \times \Omega$ as follows:

$$p(u, v, \sigma) = \begin{cases} w(u, v, \sigma) & \text{if } (u, v, \sigma) \in A \text{ and } w(u, v, \sigma) > 0, \\ 0 & \text{otherwise.} \end{cases}$$

Therefore, in this sense, the concepts of a bipolar random digraph and a bipolar weighted digraph can be used interchangeably.

3.2 Probabilistic Transitive Closure

In this section, we review the concept of probabilistic transitive closure. This concept was introduced and developed in [16]. We start by defining the transitive closure of a digraph and a bipolar digraph, then we extend the definition to probabilistic transitive closure of a bipolar weighted digraph. We also explain its relevance to FCMs.

The *transitive closure* of a digraph $D = (V, A)$ is a digraph $D^* = (V^*, A^*)$ where $V^* = V$, and $(u, v) \in A^*$ if and only if D has a directed (u, v)-walk.

The *transitive closure* of a bipolar digraph $D = (V, A)$ is a bipolar digraph $D^* = (V^*, A^*)$ where $V^* = V$, and $(u, v, \sigma) \in A^*$ if and only if D has a directed (u, v, σ)-walk.

The question to ask now is, how do we define the transitive closure (TC) of a bipolar weighted digraph?

The weights of the arcs in the TC can be computed by two models, the fuzzy model and the probabilistic model. These two models were developed in [16]. For many applications, including our study in Chapter 6, the probabilistic model is more suitable because the arc weights can (or should) be interpreted as probabilities; however, the problem of finding TC within this model is computationally hard [16].

In the probabilistic model, the weight of an arc from factor F to factor F' in an FCM is interpreted as the probability that factor F has a direct impact of the

specified sign on factor F'. Thus a large weight shows that the probability that factor F has a direct impact of the specified sign on factor F' is high.

In the probabilistic TC (to be defined below) of a bipolar weighted digraph D, the weight of a positive/negative arc from F to F' (net effect of F on F') is the probability that D has a directed positive/negative walk from F to F', and is computed from the weights of all directed (F, F')-walks. Since the concept of a probability space is required, bipolar weighted digraphs are viewed as bipolar random digraphs in this context.

Definition 3.2.1. Let $R = (V, p)$ be a bipolar random digraph. The *probabilistic transitive closure* (PTC) of R is the bipolar random digraph (V, p^∞) such that for all $(s, t, \sigma) \in V^2 \times \Omega$ we have

$$p^\infty(s, t, \sigma) = P(E_{(s,t,\sigma)}),$$

where $E_{(s,t,\sigma)}$ is the set of all bipolar digraphs in the sample space that contain a directed (s, t, σ)-walk. In other words,

$$E_{(s,t,\sigma)} = \{D \in \mathcal{D} : D \text{ contains a directed } (s, t, \sigma)\text{-walk}\}.$$

Figure 3.3 represents a bipolar random digraph and its probabilistic transitive closure.

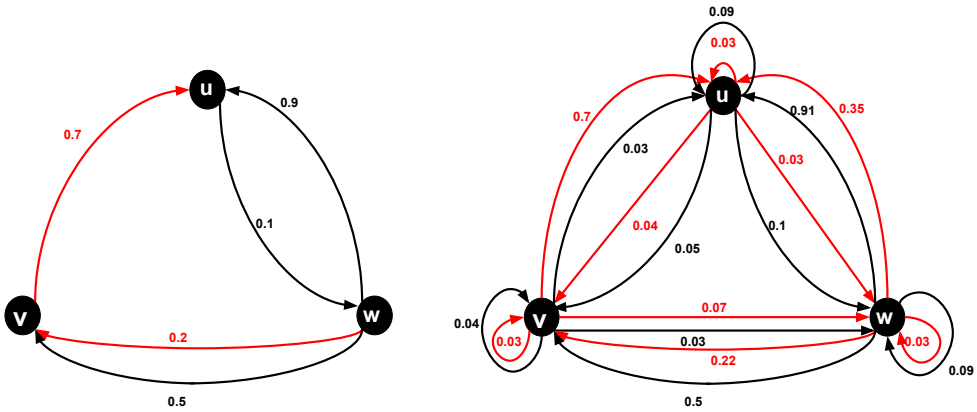

Figure 3.3: A bipolar weighted digraph (left) and its PTC (right)

The following lemma shows how to compute the probability (weight) of an arc in the PTC of a bipolar random digraph. It easily follows from Definition 3.2.1, and the definition of probability measure from Definition 3.1.3.

Lemma 3.2.2. *Let $R = (V, p)$ be a bipolar random digraph and $(s, t, \sigma) \in V^2 \times \Omega$. Then*

$$p^\infty(s, t, \sigma) = P(E_{(s,t,\sigma)}) = \sum_{D \in E_{(s,t,\sigma)}} P(\{D\}) = \sum_{D \in E_{(s,t,\sigma)}} \prod_{a \in A(D)} p(a) \prod_{a \in V^2 \times \Omega - A(D)} (1 - p(a)).$$

3.3 Summary of the Previous Work on Transitive Closure of Bipolar Weighted Digraphs

FCMs are used in various fields to represent structured knowledge. It was shown in [16] that TC of an FCM gives valuable new information, namely, it provides the total impacts of each factor on each other factor with weights; interrelated factors (factors in a strong component wherein each factor impacts all other factors); and double impacts (factors with both a positive and negative impact on another factor; in other words, parallel arcs). In [16], the notion of a bipolar weighted digraph was introduced, and the necessary assumptions were considered to show how a bipolar weighted digraph can be used to model an FCM. Furthermore, two models were proposed for defining transitive closure of a bipolar weighted digraph, the fuzzy model and the probabilistic model. In both models, a bipolar weighted digraph is used to represent an FCM, but in each model there is a different interpretation of the weight of an arc. In the fuzzy model, the weight of an arc is viewed as the degree of truth. However, in the probabilistic model, the weight of an arc is interpreted as the probability that the corresponding impact is manifested.

The mathematical theory behind the probabilistic model, as well as the fuzzy model, was developed in [16]. Furthermore, these two models were compared and several algorithms were proposed for the computation of TC. The proposed algorithm for computing the fuzzy transitive closure is based on the well-known Roy-Warshall Algorithm, and is of polynomiyal complexity. The probabilistic transitive closure, on the other hand, is closely related to the network reliability problem called s,t-connectedness, which is known to be computationally hard [5]. Hence we may assume that PTC is computationally hard as well. For that reason several algorithms for PTC were described in [16], namely, the Complete State Enumeration (CSE) Algorithm, the Principle of Inclusion-Exclusion (PIE), the bundles version of the PIE algorithm, the Boolean Algebra Algorithm (both the approximative version and exact version), and the Reduction-Recovery Algorithm. The running times of the first four of these algorithms were compared and the results of comparison were also provided; of these algorithms, the boolean algebra approach turned out to be generally the most efficient.

We give a brief description of each of these five main algorithms below. All of these algorithms take a bipolar weighted digraph $D = (V, A, p)$ as input, and return its PTC as output.

- **Complete State Enumeration (CSE) Algorithm:** First of all, in this algorithm we need to determine whether or not the input bipolar weighted digraph contains a directed (s, t, σ)-walk, for each $(s, t, \sigma) \in V^2 \times \Omega$. For this purpose the following algorithm, which is a variation of the Roy-Warshall Algorithm, was developed.

  The CSE algorithm starts by generating all subdigraphs of D, for each $(s, t, \sigma) \in$

Algorithm 1 Directed Walks

Input: support digraph $D = (V, A, p)$
Output: array w, where $w(s,t,\sigma) = 1$ if D contains an (s,t,σ)-walk, and 0 otherwise.

```
1: for all (s,t,σ) ∈ V² × Ω do
2:     if (s,t,σ) ∈ A then
3:         w(s,t,σ) := 1 else w(s,t,σ) := 0
4:     end if
5: end for
6: for all k ∈ V and i ∈ {1,2} do
7:     for all s,t ∈ V do
8:         if w(s,k,1)w(k,t,1) = 1 or w(s,k,−1)w(k,t,−1) = 1 then
9:             w(s,t,1) := 1
10:        end if
11:        if w(s,k,1)w(k,t,−1) = 1 or w(s,k,−1)w(k,t,1) = 1 then
12:            w(s,t,−1) := 1
13:        end if
14:    end for
15:    return w
16: end for
```

$V^2 \times \Omega$. Then it applies Algorithm 1 to check for each subdigraph whether or not it contains a directed (s,t,σ)-walk. At the end, it computes the probability $p^\infty(s,t,\sigma)$ using Lemma 3.2.2.

- **Principle of Inclusion-Exclusion (PIE) Algorithm:** First, let us define some terms that we need to know before speaking about the PIE algorithm.
  - Let $W = u_0a_1u_1 \dots u_{k-1}a_ku_k$ be a directed walk. A directed walk of the form $W' = u_ia_{i+1}u_{i+1} \dots a_ju_j$ is said to be a *subwalk* of W if $0 \le i \le j \le k$, for some integers i, j.
  - An (s,t,σ)-walk W' is said to be *contained* in an (s,t,σ)-walk W if it is obtained from W by deleting some subwalks of W that are closed walks.
  - A directed (s,t,σ)-walk that does not properly contain any directed (s,t,σ)-walk is called a *minimal* directed (s,t,σ)-walk.
  - Let D be the support digraph of a bipolar random digraph $R = (V,p)$, and W be a directed walk in D. The set $\{a \in A(W) : p(a) < 1\}$ is called the *uncertain arc set* of W and denoted by $A^*(W)$.
  - Let D be the support digraph of a bipolar random digraph $R = (V,p)$, and W and W^* be directed walks in D. We say W^* is a *representative* of the directed walk W if $A^*(W^*) \subseteq A^*(W)$.

The PIE algorithm starts by generating the set of representatives of all minimal (s,t,σ)-walks for all $(s,t,\sigma) \in A$, where A is the arc set of the bipolar weighted digraph. Then it uses the Principle of Inclusion-Exclusion to compute the probability $p^{\infty}(s,t,\sigma)$ of the union of the events associated with these walks.

- **PIE Algorithm — Bundles Version:** The bundles version was meant to be an enhancement of the PIE Algorithm, but in practice, it did not perform any better [16].

- **Boolean Algebra Algorithm:** As mentioned, in the PIE Algorithm the probability of the union of the events associated with the set of representatives of all minimal (s,t,σ)-walks is computed using the Principle of Inclusion-Exclusion. However, in the Boolean Algebra Algorithm, a different method is used for computing the probability of the union of these events. This method simplifies the calculation of the union of dependent events by converting it into a union of disjoint events. Let us see how this method works.

  Let $R = (V,p)$ be a bipolar random digraph and $D = (V,A,p)$ its support digraph. Assume $\mathcal{W}_{rep} = \{W_j : j = 1,2,\ldots,k\}$ is a set of representatives of all minimal directed (s,t,σ)-walks for a fixed $(s,t,\sigma) \in V^2 \times \Omega$. We define the events E_j, for $j = 1,2,\ldots,k$, as follows:

$$E_j = \{D \in \mathcal{D} : A^*(W_j) \subseteq A(D)\}.$$

  For all $a \in A$, define $X_a = \{D \in \mathcal{D} : a \in A(D)\}$ so we have $E_j = \bigcap_{a \in A^*(W_j)} X_a$. Define the event $E^*_{(s,t,\sigma)} = \bigcup_{j=1}^{k} E_j$, that is,

$$E^*_{(s,t,\sigma)} = \{D \in \mathcal{D} : A^*(W_i) \subseteq A(D) \text{ for some } i = 1,\ldots,k\}.$$

  Now, we can write $E^*_{(s,t,\sigma)}$ as follows:

$$E^*_{(s,t,\sigma)} = \bigcup_{j=1}^{k} E_j = \bigcup_{j=1}^{k} \Big(\bigcap_{a \in A^*(W_j)} X_a \Big).$$

  We can use DeMorgan's Laws to "isolate" event E_1:

$$E_1 \cup E_2 \cup \ldots \cup E_k = E_1 \dot{\cup} (\bar{E}_1 \cap (E_2 \cup \ldots \cup E_k)).$$

  Note that the complement of an event E is shown with the symbol $\bar{E}$. By repeating this procedure and "isolating" one event after another we can write $\bigcup_{j=1}^{k} E_j$ as a disjoint union,

$$E^*_{(s,t,\sigma)} = E'_1 \dot{\cup} E'_2 \dot{\cup} \ldots \dot{\cup} E'_m.$$

Then

$$\begin{aligned} P(E^*_{(s,t,\sigma)}) &= P(E'_1 \dot\cup E'_2 \dot\cup \ldots \dot\cup E'_m) \\ &= P(E'_1) + P(E'_2) + \ldots + P(E'_m) \\ &= \sum_{i=1}^{m} \prod_{a\in A'} p(a) \prod_{a\in A''} (1-p(a)), \end{aligned}$$

where each $E'_i = \Big(\bigcap_{a\in A'} X_a\Big) \bigcap \Big(\bigcap_{a\in A''} \bar{X}_a\Big)$ for some disjoint sets $A', A'' \subseteq V^2 \times \Omega$.

We know $E_1 = \bigcap_{a\in A^*(W_1)} X_a$, so we have $\bar{E}_1 = \bigcup_{a\in A^*(W_1)} \bar{X}_a$, and therefore

$$\begin{aligned} E_1 \dot\cup \Big(\bar{E}_1 \cap (\bigcup_{j=2}^{k} E_j)\Big) &= E_1 \dot\cup \Big((\bigcup_{a\in A^*(W_1)} \bar{X}_a) \cap (\bigcup_{j=2}^{k} E_j)\Big) \\ &= E_1 \dot\cup \bigcup_{a\in A^*(W_1)} \bigcup_{j=2}^{k} (\bar{X}_a \cap E_j). \end{aligned}$$

As can be seen, each "isolation" increases the number of conjunctive clauses. Consequently, the efficiency of this algorithm depends on the number of conjunctive clauses generated. The process of "isolating" eventually terminates as each new generated conjunctive clause is independent from the previous isolated clauses, and thus no event is generated twice.

In the exact approach of the Boolean Algebra Algorithm, events $E_1, E_2, \ldots, E_k$ are sorted in the increasing order of their length (that is, the number of literals X_a in the event). In each step of "isolation", the complement of a literal in the isolated clause is added to each of these clauses. Therefore, we will have a similar ordering in the new clauses. Isolating the events in this specific order accelerates the process of "isolation".

Now let us see how the Boolean Algebra Algorithm can be modified into an approximation algorithm.

Let (s,t,σ) be a fixed triple. After the i-th step, the event $E^*_{(s,t,\sigma)}$ has been expressed as follows:

$$E^*_{(s,t,\sigma)} = E_1^{(i)} \dot\cup E_2^{(i)} \dot\cup \ldots E_i^{(i)} \dot\cup \big(E_{i+1}^{(i)} \cup \ldots \cup E_{k_i}^{(i)}\big).$$

We know that after the i-th step, the first i events have been isolated, so if $P(E_1^{(i)}) + P(E_2^{(i)}) + \ldots + P(E_i^{(i)})$ is taken as an approximation for $P(E^*_{(s,t,\sigma)})$, then we have the following upper bound for the error r_i:

$$r_i = P(E^*_{(s,t,\sigma)}) - \Big(P(E_1^{(i)}) + P(E_2^{(i)}) + \ldots + P(E_i^{(i)})\Big)$$

$$\begin{aligned} &= P\big(E_{i+1}^{(i)} \cup \ldots \cup E_{k_i}^{(i)}\big) \\ &\leq P(E_{i+1}^{(i)}) + \ldots + P(E_{k_i}^{(i)}). \end{aligned}$$

The sequence $(r_i)_i$ of errors satisfies the recursion $r_{i+1} = r_i - P(E_{i+1}^{(i+1)})$, where $E_{i+1}^{(i+1)}$ is the event isolated in the $(i+1)$-st step. Since this sequence is decreasing and also the exact version of the Boolean Algebra Algorithm terminates in a finite number of steps, we conclude that $r_m = 0$ for some integer m.

In the approximative version of the algorithm, we choose a desirable error ϵ and the computation of $P(E_{(s,t,\sigma)}^*)$ will be terminated as soon as r_i falls below ϵ. We can accelerate this process by isolating, at each step i, the event $E_i^{(i)}$ among $P(E_i^{(i-1)}), P(E_{i+1}^{(i-1)}), \ldots, P(E_{k_{i-1}}^{(i-1)})$ such that $P(E_i^{(i)})$ is as large as possible.

- **Reduction-Recovery Algorithm:** The Reduction-Recovery Algorithm reduces a digraph recursively by deleting vertices of indegree at most one and outdegree at most one. Once the reduction is completed, the transitive closure of the reduced digraph is computed directly; that is, using one of the other algorithms. At the end, the transitive closure of the original digraph is recovered from the transitive closure of the reduced digraph. We will give detailed information on this algorithm in Chapter 4 of the thesis.

3.4 Previous Implementations of the Algorithms for Computing the Transitive Closure of Bipolar Weighted Digraphs

A software package, called FuzzyTC, was implemented in Python by J. Morzaria and M. Šajna, for computing the fuzzy transitive closure (FTC) of an FCM [14]. This software is based on the algorithm for FTC which was described in [16]. Furthermore, several of the algorithms mentioned in Section 3.3, namely, the Complete State Enumeration (CSE) Algorithm, the Principle of Inclusion-Exclusion (PIE), the bundles version of the PIE algorithm, and the Boolean Algebra Algorithm, were coded in C by P. Niesink, K. Poulin, and M. Šajna for practical use [15]. This software package, called ProbClosure, was used to obtain the results in paper [8]. In addition, the implementation of these algorithms were compared, and the Boolean Algebra approach turned out to be the most efficient. Hence it was subsequently re-coded by J. Morzaria and M. Šajna, in a software package called ProbTC [13]. The Reduction-Recovery Algorithm was also described in [16], but it has never been coded before. In this thesis, we wrote two specific forms of pseudocode for the Reduction-Recovery Algorithm, and we implemented them in Python. In both codes, the existing ProbTC code [13] is used to compute the transitive closure of the reduced digraph.

Chapter 4

The Reduction-Recovery Algorithm: Python Code and a Comparison of Running Times

4.1 Introduction

Finding the probabilistic transitive closure (PTC) of a digraph is computationally hard; for this reason, paper [16] introduced a number of different algorithms. One of them was the Reduction-Recovery Algorithm. The mathematical basis and pseudocode version of the Reduction-Recovery Algorithm were developed in [16]. In particular, it was shown how to recursively reduce the digraph and recover the PTC of the input digraph from the PTC of the reduced digraph. However, the algorithm in [16] does not specify the form of recursion, and it has never been coded. In this chapter, we wrote two specific forms of pseudocode for the Reduction-Recovery Algorithm, and we implemented them in Python.

In Section 4.2, we speak about the mathematical basis behind the Reduction-Recovery Algorithm, and we describe the two forms of pseudocode in full detail. In Section 4.3, we compare our implementation of the Reduction-Recovery Algorithm with the existing software, namely ProbTC [13], and show that the enhanced code is at least as efficient, and often more efficient, than ProbTC.

4.2 The Reduction-Recovery Algorithm

The Reduction-Recovery Algorithm reduces a digraph recursively by deleting vertices of indegree at most one and outdegree at most one. Once the reduction is completed, the transitive closure of the reduced digraph is computed directly. At the end, the transitive closure of the original digraph is recovered from the transitive closure of

the reduced digraph.

The mathematical theory behind this algorithm was developed in [16], and is stated in the following lemma. The proof can be found in [16].

Lemma 4.2.1. [16] *Let $R = (V, p)$ be a bipolar random digraph and $D = (V, A, p)$ its support digraph. Assume that the underlying undirected graph of D is connected. Let $u \in V$ be a vertex with $indeg_D(u) \leq 1$ and $outdeg_D(u) \leq 1$. Then the following hold.*

1. *If $indeg_D(u) = 1$ and $outdeg_D(u) = 0$, let $v_1 \in V$ be such that $(v_1, u, \sigma_1) \in A$. Define a bipolar random digraph $R' = (V', p')$ as follows:*

   $V' = V - \{u\}$,
   $p'(s, t, \sigma) = p(s, t, \sigma)$ *for all* $(s, t, \sigma) \in V'^2 \times \Omega$.

   Let $R^\infty = (V, p^\infty)$ and $R'^\infty = (V', p'^\infty)$ be the PTCs of R and R', respectively. Then:

   $p^\infty(s, t, \sigma) = p'^\infty(s, t, \sigma)$ *for all* $(s, t, \sigma) \in V'^2 \times \Omega$,
   $p^\infty(s, u, \sigma) = p'^\infty(s, v_1, \sigma\sigma_1)p(v_1, u, \sigma_1)$ *for all* $s \in V - \{u, v_1\}, \sigma \in \Omega$,
   $p^\infty(v_1, u, \sigma_1) = p(v_1, u, \sigma_1)$,
   $p^\infty(v_1, u, -\sigma_1) = p'^\infty(v_1, v_1, -1)p(v_1, u, \sigma_1)$,
   $p^\infty(u, t, \sigma) = 0$ *for all* $t \in V, \sigma \in \Omega$.

2. *If $indeg_D(u) = 0$ and $outdeg_D(u) = 1$, let $v_2 \in V$ be such that $(u, v_2, \sigma_2) \in A$. Define a bipolar random digraph $R' = (V', p')$ as follows:*

   $V' = V - \{u\}$,
   $p'(s, t, \sigma) = p(s, t, \sigma)$ *for all* $(s, t, \sigma) \in V'^2 \times \Omega$.

   Let $R^\infty = (V, p^\infty)$ and $R'^\infty = (V', p'^\infty)$ be the PTCs of R and R', respectively. Then:

   $p^\infty(s, t, \sigma) = p'^\infty(s, t, \sigma)$ *for all* $(s, t, \sigma) \in V'^2 \times \Omega$,
   $p^\infty(s, u, \sigma) = 0$ *for all* $s \in V, \sigma \in \Omega$,
   $p^\infty(u, t, \sigma) = p(u, v_2, \sigma_2)p'^\infty(v_2, t, \sigma\sigma_2)$ *for all* $t \in V - \{u, v_2\}, \sigma \in \Omega$,
   $p^\infty(u, v_2, \sigma_2) = p(u, v_2, \sigma_2)$,
   $p^\infty(u, v_2, -\sigma_2) = p(u, v_2, \sigma_2)p'^\infty(v_2, v_2, -1)$,

3. *If $indeg_D(u) = outdeg_D(u) = 1$, let $v_1, v_2 \in V$ be such that $(v_1, u, \sigma_1), (u, v_2, \sigma_2) \in A$. In addition, assume that there exists no directed $(u, u, -1)$-walk in D. Define a bipolar random digraph $R' = (V', p')$ as follows:*

$V' = V - \{u\}$,
$p'(s,t,\sigma) = p(s,t,\sigma)$ *for all* $(s,t,\sigma) \in V'^2 \times \Omega$ *such that* $(s,t,\sigma) \neq (v_1,v_2,\sigma_1\sigma_2)$,
$p'(v_1,v_2,\sigma_1\sigma_2) = p(v_1,v_2,\sigma_1\sigma_2) + p(v_1,u,\sigma_1)p(u,v_2,\sigma_2) -$
$p(v_1,v_2,\sigma_1\sigma_2)p(v_1,u,\sigma_1)p(u,v_2,\sigma_2)$.

Let $R^\infty = (V,p^\infty)$ *and* $R'^\infty = (V',p'^\infty)$ *be the PTCs of* R *and* R', *respectively. Then:*

$p^\infty(s,t,\sigma) = p'^\infty(s,t,\sigma)$ *for all* $(s,t,\sigma) \in V'^2 \times \Omega$,
$p^\infty(v_1,u,\sigma_1) = p(v_1,u,\sigma_1)$,
$p^\infty(v_1,u,-\sigma_1) = p'^\infty(v_1,v_1,-1)p(v_1,u,\sigma_1)$,
$p^\infty(s,u,\sigma) = p'^\infty(s,v_1,\sigma\sigma_1)p(v_1,u,\sigma_1)$ *for all* $s \in V - \{u,v_1\}, \sigma \in \Omega$,
$p^\infty(u,t,\sigma) = p(u,v_2,\sigma_2)p'^\infty(v_2,t,\sigma\sigma_2)$ *for all* $t \in V - \{u,v_2\}, \sigma \in \Omega$,
$p^\infty(u,v_2,\sigma_2) = p(u,v_2,\sigma_2)$,
$p^\infty(u,v_2,-\sigma_2) = p(u,v_2,\sigma_2)p'^\infty(v_2,v_2,-1)$,
$p^\infty(u,u,-1) = 0$,
$p^\infty(u,u,1) = p(u,v_2,\sigma_2)p'^\infty(v_2,v_1,\sigma_1\sigma_2)p(v_1,u,\sigma_1)$ *if* $v_1 \neq v_2$,
$p^\infty(u,u,1) = p(u,v_2,\sigma_2)p(v_1,u,\sigma_1)$ *if* $v_1 = v_2$.

We use two approaches to implement the Reduction-Recovery Algorithm. Both approaches are implemented in Python 3, and the existing ProbTC code [13] was used to compute the transitive closure of the reduced digraph. The first approach, which we call SRR-PTC, is a "by-hand" recursion algorithm. In this approach we used a stack and saved only the data really needed. The second approach, which we call RRR-PTC, uses Python's built-in recursion. On one hand, RRR-PTC may consume more memory as for each call it passes a copy of the object so it saves more data than needed; on the other hand, we expect that a built-in recursion is implemented more efficiently than a "by-hand" recursion. Hence, we implemented both to be able to compare.

Algorithms 2 – 3 and 3 – 4 represent the pseudocode versions of SRR-PTC and RRR-PTC, respectively. The part of the code common to SRR-PTC and RRR-PTC is called Recovery, and is presented as Algorithm 3. The Python code for SRR-PTC and RRR-PTC can be found in Appendices A and B, respectively.

In Algorithms 2 – 4. the type of a vertex is defined as follows:

- a vertex with indegree 1 and outdegree 0 is called of *type 1*,
- a vertex with indegree 0 and outdegree 1 is called of *type 2*,
- a vertex with indegree 1 and outdegree 1 is called of *type 3*.

In both SRR-PTC and RRR-PTC, we are assuming that there is a function type(u, D) that determines the type of vertex u in a digraph D.

4.2.1 The SRR-PTC Algorithm

This algorithm takes as input a bipolar weighted digraph $D = (V, A, p)$, and returns its PTC. It starts by sequentially removing vertices of type 1 or 2, and if that is not possible, vertices of type 3. After removal, each vertex (as well as the additional data that will be needed for recovery) is put on stack. When no further vertices can be removed, the PTC of the fully reduced digraph is computed directly. Next, the removed vertices are taken off the stack one after the other, and the PTCs of these "intermediate" digraphs are recovered using the Recovery Algorithm (Algorithm 3). The PTC of the input digraph is the last recovered PTC.

4.2.2 The Recovery Algorithm

This algorithm takes the PTC of the reduced digraph and the removed vertex (with the additional saved information) as inputs, and returns the PTC of the recovered digraph as output. First, this algorithm puts the removed vertex back to the reduced digraph. Then it computes the PTC of the recovered digraph from the PTC of the reduced digraph according to Lemma 4.2.1.

The Recovery Algorithm is used in both SRR-PTC and RRR-PTC, so it is written here as a separate procedure, but corresponds to lines 311 to 408 in the SRR-PTC code (see Appendix A), as well as to lines 289 to 421 in the RRR-PTC code (see Appendix B).

4.2.3 The RRR-PTC Algorithm

This algorithm takes a bipolar weighted digraph $D = (V, A, p)$ as input, and returns its PTC by using Python's built-in recursion. It starts by deleting a vertex of type 1 or 2, and if that is not possible, a vertex of type 3. After removing such a vertex, the algorithm calls itself recursively. If no vertex can be removed, the PTC of the digraph is computed directly, using ProbTC. Finally, the PTC of the input digraph is recovered using the Recovery Algorithm (Algorithm 3).

Algorithm 2 SRR-PTC

Input: support digraph $D = (V, A, p)$ of the bipolar random digraph $R = (V, p)$
Output: PTC $R^{\infty} = (V, p^{\infty})$ of R

```
 1:
 2: procedure SRR-PTC (V, A, p)
 3:     stack := ∅
 4:     while |V| > 2 do
 5:         save-vertex-type3 := None
 6:         for all u ∈ V do
 7:             if type(u, D) = 1 or type(u, D) = 2 then
 8:                 if type(u, D) = 1 then
 9:                     determine v_1 ∈ V and σ_1 ∈ Ω such that (v_1, u, σ_1) ∈ A
10:                     in-neighbor := (v_1, σ_1 · p(v_1, u, σ_1))
11:                     put (u, 1, in-neighbor) on stack
12:                     A := A − {(v_1, u, σ_1)}
13:                 else if type(u, D) = 2 then
14:                     determine v_2 ∈ V and σ_2 ∈ Ω such that (u, v_2, σ_2) ∈ A
15:                     out-neighbor := (v_2, σ_2 · p(u, v_2, σ_2))
16:                     put (u, 2, out-neighbor) on stack
17:                     A := A − {(u, v_2, σ_2)}
18:                 end if
19:                 V := V − {u}
20:                 break
21:                 #Break out of the " for all u ∈ V" loop.
22:                 #If possible, we reduce a vertex of type 1 or 2; otherwise, we look for a
    vertex of type 3.
23:             else if type(u, D) = 3 then
24:                 save-vertex-type3 := u
25:             end if
26:         end for
27:         #The case with no vertex of type 1 or 2, but with a vertex of type 3.
28:         #Check the type of the last vertex.
29:         if type(u, D) ∈ {0, 3} and save-vertex-type3 ≠ None then
```

```
            u := save-vertex-type3
            determine v_1 ∈ V and σ_1 ∈ Ω such that (v_1, u, σ_1) ∈ A
            in-neighbor := (v_1, σ_1 · p(v_1, u, σ_1))
            determine v_2 ∈ V and σ_2 ∈ Ω such that (u, v_2, σ_2) ∈ A
            out-neighbor := (v_2, σ_2 · p(u, v_2, σ_2))
            put (u, 3, (in-neighbor, out-neighbor)) on stack
            A := (A − {(v_1, u, σ_1), (u, v_2, σ_2)}) ∪ {(v_1, v_2, σ_1σ_2)}
            V := V − {u}
            p(v_1, v_2, σ_1σ_2) := p(v_1, v_2, σ_1σ_2) + p(v_1, u, σ_1)p(u, v_2, σ_2)−
                                   p(v_1, v_2, σ_1σ_2)p(v_1, u, σ_1)p(u, v_2, σ_2)
        else if type(u, D) = 0 and save-vertex-type3 = None then
            break
            #Break out of the " while |V| > 2" loop.
            #No vertices of type 1, 2 or 3 – reduction is complete.
        end if
    end while
    #Call ProbTC on the reduced digraph.
    (V, p^∞) := ProbTC(V, A, p)
    #Start recovery.
    while stack ≠ ∅ do
        pop stored from stack
        R^∞ := Recovery (stored, V, p^∞)
    end while
end procedure
```

Algorithm 3 Recovery

procedure Recovery $(stored, V, p^\infty)$
  $(u, typ, neighbors) := stored$
  $V := V \cup \{u\}$
  if $typ = 1$ **then**
    $(v_1, p) := neighbors$
    #Recall $p = \sigma_1 \cdot p(v_1, u, \sigma_1)$, where $p(v_1, u, \sigma_1)$ was the weight of the deleted arc.
    if $p > 0$ **then**
      $\sigma_1 := 1$ **else** $\sigma_1 := -1$
    end if
    $p := |p|$
    $p^\infty(v_1, u, \sigma_1) := p$
    $p^\infty(v_1, u, -\sigma_1) := p^\infty(v_1, v_1, -1) \cdot p$
    for all $s \in V - \{u, v_1\}$ and $\sigma \in \Omega$ **do**
      $p^\infty(s, u, \sigma) := p^\infty(s, v_1, \sigma\sigma_1) \cdot p$
    end for
    for all $t \in V$ and $\sigma \in \Omega$ **do**
      $p^\infty(u, t, \sigma) := 0$
    end for
  else if $typ = 2$ **then**
    $(v_2, p) := neighbors$
    #Recall $p = \sigma_2 \cdot p(u, v_2, \sigma_2)$, where $p(u, v_2, \sigma_2)$ was the weight of the deleted arc.
    if $p > 0$ **then**
      $\sigma_2 := 1$ **else** $\sigma_2 := -1$
    end if
    $p := |p|$
    $p^\infty(u, v_2, \sigma_2) := p$
    $p^\infty(u, v_2, -\sigma_2) := p \cdot p^\infty(v_2, v_2, -1)$
    for all $s \in V$ and $\sigma \in \Omega$ **do**
      $p^\infty(s, u, \sigma) := 0$
    end for
    for all $t \in V - \{u, v_2\}$ and $\sigma \in \Omega$ **do**
      $p^\infty(u, t, \sigma) := p \cdot p^\infty(v_2, t, \sigma\sigma_2)$
    end for

34: **else if** $typ = 3$ **then**
35: $((v_1, p_1), (v_2, p_2)) := neighbors$
36: #Recall $p_1 = \sigma_1 \cdot p(v_1, u, \sigma_1)$ and $p_2 = \sigma_2 \cdot p(u, v_2, \sigma_2)$ where $p(v_1, u, \sigma_1)$ and $p(u, v_2, \sigma_2)$ were the weights of the deleted arcs.
37: **if** $p_1 > 0$ **then**
38: $\sigma_1 := 1$ **else** $\sigma_1 := -1$
39: **end if**
40: $p_1 := |p_1|$
41: $p^\infty(v_1, u, \sigma_1) := p_1$
42: $p^\infty(v_1, u, -\sigma_1) := p^\infty(v_1, v_1, -1) \cdot p_1$
43: **for all** $s \in V - \{u, v_1\}$ and $\sigma \in \Omega$ **do**
44: $p^\infty(s, u, \sigma) := p^\infty(s, v_1, \sigma\sigma_1) \cdot p_1$
45: **end for**
46: **if** $p_2 > 0$ **then**
47: $\sigma_2 := 1$ **else** $\sigma_2 := -1$
48: **end if**
49: $p_2 := |p_2|$
50: $p^\infty(u, v_2, \sigma_2) := p_2$
51: $p^\infty(u, v_2, -\sigma_2) := p_2 \cdot p^\infty(v_2, v_2, -1)$
52: **for all** $t \in V - \{u, v_2\}$ and $\sigma \in \Omega$ **do**
53: $p^\infty(u, t, \sigma) := p_2 \cdot p^\infty(v_2, t, \sigma\sigma_2)$
54: **end for**
55: $p^\infty(u, u, -1) := 0$
56: **if** $v_1 = v_2$ **then**
57: $p^\infty(u, u, 1) := p_1 \cdot p_2$
58: **else**
59: $p^\infty(u, u, 1) := p_1 \cdot p_2 \cdot p^\infty(v_2, v_1, \sigma_1\sigma_2)$
60: **end if**
61: **end if**
62: **return** (V, p^∞)
63: **end procedure**

Algorithm 4 RRR-PTC

Input: support digraph $D = (V, A, p)$ of the bipolar random digraph $R = (V, p)$
Output: PTC of (V, p^∞) of R

```
 1:
 2: procedure RRR-PTC (V, A, p)
 3:     Flag := False
 4:     save-vertex-type3 := None
 5:     for all u ∈ V  do
 6:         if  type(u, D) = 1 then
 7:             Flag := True
 8:             determine v_1 ∈ V and σ_1 ∈ Ω such that (v_1, u, σ_1) ∈ A
 9:             stored := (u, 1, σ_1 · p(v_1, u, σ_1))
10:             A := A − {(v_1, u, σ_1)}
11:             V := V − {u}
12:         else if type(u, D) = 2 then
13:             Flag := True
14:             determine v_2 ∈ V and σ_2 ∈ Ω such that (u, v_2, σ_2) ∈ A
15:             stored := (u, 2, σ_2 · p(u, v_2, σ_2))
16:             A := A − {(u, v_2, σ_2)}
17:             V := V − {u}
18:         else if type(u, D) = 3 then
19:             save-vertex-type3 := u
20:         end if
21:         #Reducing digraph by using vertex of type 1 or 2.
22:         if Flag = True then
23:             #Recursively compute PTC, using further reduction if possible.
24:             (V, p^∞) := RRR-PTC(V, A, p)
25:             #Recover PTC of the input digraph.
26:             (V, p^∞) := Recovery(stored, V, p^∞)
27:             return (V, p^∞)
28:             break
29:             #Break out of the " for all u ∈ V" loop.
30:         end if
31:     end for
```

```
    if type(u, D) ∈ {0, 3} and save-vertex-type 3 ≠ None then
        u := save-vertex-type3
        determine v_1 ∈ V and σ_1 ∈ Ω such that (v_1, u, σ_1) ∈ A
        determine v_2 ∈ V and σ_2 ∈ Ω such that (u, v_2, σ_2) ∈ A
        stored := (u, 3, ((σ_1 · p(v_1, u, σ_1)), (σ_2 · p(u, v_2, σ_2))))
        A := (A − {(v_1, u, σ_1), (u, v_2, σ_2)}) ∪ {(v_1, v_2, σ_1σ_2)}
        V := V − {u}
        p(v_1, v_2, σ_1σ_2) := p(v_1, v_2, σ_1σ_2) + p(v_1, u, σ_1)p(u, v_2, σ_2)−
                               p(v_1, v_2, σ_1σ_2)p(v_1, u, σ_1)p(u, v_2, σ_2)
        #Recursively compute PTC, using further reduction if possible.
        (V, p^∞) := RRR-PTC(V, A, p)
        #Recover PTC of the input digraph.
        (V, p^∞) := Recovery(stored, V, p^∞)
        return (V, p^∞)
    else
        #Reduction is complete, call ProbTC.
        (V, p^∞) := ProbTC(V, A, p)
        return (V, p^∞)
    end if
end procedure
```

4.3 Comparison of Running Times

In this section, we compare the running times of the three algorithms, namely, ProbTC, RRR-PTC, and SRR-PTC. We chose a wide range of examples — digraphs with different numbers of vertices and arcs — for comparison. The input digraphs were either reducible (digraphs with removable vertices) or irreducible (digraphs without removable vertices). The running times of the tested examples are collected in Tables 4.1 – 4.4, and the digraphs in each table are sorted in a non-increasing order of the number of arcs. In Tables 4.1, 4.3, and 4.4, the exact version of ProbTC was used. In Table 4.2, however, the approximative version of ProbTC was used since the exact version took too long to finish for some of the examples. We used the following abbreviations in all tables:

- Name of the Example → Name
- Approximation → Approx
  (maximum allowable error for the approximative version of ProbTC)
- Number of Vertices → #Vertices
- Number of Arcs → #Arcs
- Arc-Density → Arc-Density
  Note that arc density is computed as
  $$\frac{\text{number of arcs}}{\text{maximum number of arcs in a bipolar digraph}} = \frac{\#\text{Arcs}}{2 \cdot n^2}.$$
- Number of Removed Vertices of Type 1 or 2 → #RV(1 or 2)
- Number of Removed Vertices of Type 3 → #RV(3)
- Total Number of Removed Vertices → Total #RV
- RRR-PTC Running Time → RRR-Run-Time
- SRR-PTC Running Time → SRR-Run-Time
- Time-Ratio of SRR-PTC to RRR-PTC→ TR (SRR to RRR)
- ProbTC Running Time → ProbTC Run-Time
- Time-Ratio of SRR-PTC to ProbTC → TR (SRR to ProbTC)

Our code was run on a MacBook Pro computer (manufactured in 2017) with the following specifications:

Processor: 2.8 GHz Quad-core Intel Core i7

Memory: 16 GB 2133 MHz LPDDR3

4.3.1 Comparing SRR-PTC and RRR-PTC (Table 4.1)

Table 4.1 shows the results of our comparison of the running times of SRR-PTC and RRR-PTC on twenty examples. In both algorithms we used the exact version of ProbTC to find the PTC of the reduced digraph. Among the twenty examples, we chose both reducible digraphs and irreducible digraphs. The last column represents the ratios of running times of SRR-PTC to RRR-PTC. As can be seen in this column, these two programs have almost the same running times, and in most of the tested cases, SRR-PTC is slightly faster.

Name	#Vertices	#Arcs	Arc-Density	#RV(1 or 2)	#RV(3)	Total #RV	RRR-Run-Time	SRR-Run-Time	TR (SRR to RRR)
1-elderin	15	29	0.064	1	3	4	0:00:00.031020	0:00:00.025597	0.825
2-10.B.2	10	26	0.13	0	0	0	0:07:45.384032	0:08:08.262306	1.049
3-10.B.3	10	22	0.11	0	1	1	0:00:00.299667	0:00:00.263881	0.881
4-10.B.9	10	21	0.105	0	0	0	0:00:00.012994	0:00:00.012423	0.956
5-M	7	20	0.204	0	0	0	0:00:01.311654	0:00:01.362355	1.039
6-10.B.5	10	19	0.095	1	0	1	0:00:09.482438	0:00:09.365680	0.988
7-10.A.2	10	19	0.095	1	4	5	0:00:00.016385	0:00:00.015032	0.917
8-10.B.1	10	18	0.09	1	3	4	0:00:00.011532	0:00:00.011234	0.974
9-ManyWalks	9	18	0.111	0	0	0	0:12:33.941023	0:12:22.198902	0.984
10-10.B.7	10	17	0.085	3	1	4	0:00:00.006020	0:00:00.006394	1.062
11-10.A.6	10	15	0.075	1	1	2	0:00:00.021741	0:00:00.018773	0.863
12-10.A.3	10	14	0.07	2	1	3	0:00:00.005127	0:00:00.004730	0.922
13-10.A.10	10	12	0.06	4	0	4	0:00:00.002815	0:00:00.003002	1.066
14-10.A.7	10	11	0.055	1	4	5	0:00:00.006564	0:00:00.006475	0.986
15-10.A.9	10	9	0.045	6	0	6	0:00:00.001174	0:00:00.001135	0.967
16-BoolGood4	7	9	0.099	1	0	1	0:00:00.002027	0:00:00.001972	0.973
17-10.A.4	10	7	0.035	7	0	7	0:00:00.000616	0:00:00.000552	0.896
18-10.A.8	10	6	0.03	2	1	3	0:00:00.002756	0:00:00.002449	0.889
19-10.A.5	10	5	0.025	4	1	5	0:00:00.001000	0:00:00.001165	1.165
20-keven1	3	5	0.278	0	0	0	0:00:00.000675	0:00:00.000606	0.898

Table 4.1: Comparison of running times for SRR-PTC and RRR-PTC (using the exact version of ProbTC)

4.3.2 Comparing SRR-PTC and ProbTC using the approximative version of ProbTC (Table 4.2)

In Table 4.1 for most of the examples, SRR-PTC is slightly faster than RRR-PTC. Hence, we decided to consider SRR-PTC as the new enhancement for comparison with ProbTC. Table 4.2 represents the results of timing of these two programs. We used the approximative version of ProbTC to find the PTC of the reduced digraph, as well as in ProbTC itself. Note that the second column shows the maximum error that was used for the approximative version of ProbTC. The last column represents the ratios of running times of SRR-PTC to ProbTC. We observe that the larger the number of removed vertices, the smaller the ratio. In other words, SRR-PTC runs faster than ProbTC on examples that have many removable vertices. Furthermore, these two programs have almost the same running times for irreducible digraphs.

Name	Approx	#Vertices	#Arcs	Arc-Density	#RV(1 or 2)	#RV(3)	Total#RV	ProbTC Run-Time	SRR-Run-Time	TR(SRR to ProbTC)
1-Stabin	0.01	18	54	0.083	1	1	2	1:43:06.179684	1:04:29.246843	0.625
2-TkFCMsimple	0.01	30	52	0.029	4	6	10	8:59:45.456462	0:03:58.946188	0.007
3-TkFCMsimple2	0.01	30	52	0.029	4	6	10	7:36:04.272437	0:03:57.465063	0.009
4- TKFCM.alt	0.01	19	36	0.0499	1	5	6	2:14:00.757245	0:00:02.614393	0.000325
5- Huntrin	0.0001	14	39	0.099	0	2	2	3:38:31.838466	0:02:01.235871	0.009
6- Mentor-Matrix.txt	0.00001	17	35	0.061	0	1	1	0:05:54.670614	0:03:56.243885	0.666
7-10.C.9	10^{-10}	10	32	0.16	0	0	0	0:25:58.944815	0:26:00.379332	1.001
8-10.C.8	10^{-10}	10	32	0.16	0	0	0	0:07:29.044671	0:07:28.231681	0.998
9-10.C.1	10^{-10}	10	30	0.15	0	0	0	5:33:39.396641	5:18:11.915523	0.954
10-10.C.7	10^{-10}	10	30	0.15	0	0	0	0:00:27.885149	0:00:28.262943	1.014
11-10.C.6	10^{-10}	10	28	0.14	0	0	0	0:01:38.742281	0:01:38.510270	0.998
12-Rodeo.Bool	10^{-10}	9	27	0.17	0	0	0	0:18:32.018059	0:18:34.203478	1.002
13-AKELDIN	10^{-10}	10	26	0.13	0	0	0	0:26:46.335050	0:26:45.598073	1.0
14-10.C.10	10^{-10}	10	26	0.13	0	1	1	0:06:42.383428	0:01:17.519272	0.193
15-10.B.2	10^{-10}	10	26	0.13	0	0	0	0:00:08.706291	0:00:09.135587	1.049
16-10.B.8	10^{-10}	10	24	0.12	0	0	0	0:19:41.758567	0:19:30.774396	0.991
17-RMatrix10a	10^{-10}	10	24	0.12	0	1	1	0:00:30.393441	0:00:18.749014	0.617
18-Small-3	10^{-10}	5	22	0.44	0	0	0	0:00:01.637985	0:00:01.552435	0.948
19-Small-4	10^{-10}	5	21	0.42	0	0	0	0:00:00.449130	0:00:00.389854	0.868
20-10.B.9	10^{-10}	10	21	0.105	0	0	0	0:00:00.015765	0:00:00.014304	0.907
21-10.B.6	10^{-10}	10	20	0.1	0	0	0	0:00:00.026670	0:00:00.023610	0.885
22-M	10^{-10}	7	20	0.204	0	0	0	0:00:00.637423	0:00:00.583702	0.916
23-Small2A	10^{-10}	5	18	0.36	0	0	0	0:00:03.459186	0:00:03.369688	0.974
23-10.B.10	10^{-10}	10	18	0.09	0	0	0	0:00:01.124631	0:00:01.050556	0.934
24-Many.walk	10^{-10}	9	18	0.111	0	0	0	0:00:19.519073	0:00:19.352776	0.991
25-EG5	10^{-10}	5	16	0.32	0	0	0	0:01:21.087043	0:01:20.755856	0.996
26-Keven1	10^{-10}	3	5	0.278	0	0	0	0:00:00.000662	0:00:00.000932	1.408

Table 4.2: Comparison of running times for SRR-PTC and ProbTC (using the approximative version of ProbTC)

4.3.3 Comparing SRR-PTC and ProbTC on irreducible examples (Table 4.3)

In Table 4.3, we present the results of timing of SRR-PTC and ProbTC on 13 examples. All these 13 examples are irreducible digraphs and were selected from examples in Table 4.2. In Table 4.3, we used the exact version of ProbTC instead of the approximative version. The last two columns represent the ratio of running times of SRR-PTC to ProbTC in the exact version and approximative version of ProbTC, respectively. These two columns point out that no matter which version of ProbTC is used, SRR-PTC and ProbTC have almost the same running times for irreducible digraphs. To put it differently, SRR-PTC does not take longer than ProbTC for irreducible examples.

4.3.4 Comparing SRR-PTC and ProbTC on reducible examples (Table 4.4)

Table 4.4 represents our comparison of SRR-PTC and ProbTC on reducible digraphs. The last column gives the ratios of running times of SRR-PTC to ProbTC in the exact version of ProbTC. The data in the last column confirms that SRR-PTC is faster than ProbTC for reducible digraphs. We observe that, roughly speaking, the more vertices removed, the smaller the ratio.

Name	#Vertices	#Arcs	Arc-Density	#RV(1 or 2)	#RV(3)	Total#RV	ProbTC Run-Time	SRR-Run-Time	TR(SRR to ProbTC)	Approx Ratio
1-10.C.7	10	30	0.15	0	0	0	10:35:35.877375	10:34:36.436314	0.998	1.014
2-10.C.6	10	28	0.14	0	0	0	15:37:15.001267	15:44:10.044212	1.007	0.998
3-Rodeo.Bool	9	27	0.17	0	0	0	7:23:09.881704	7:29:10.526042	1.014	1.002
4-10.B.2	10	26	0.13	0	0	0	0:07:48.727737	0:07:36.522041	0.974	1.049
5-Small-3	5	22	0.44	0	0	0	7:59:05.963184	8:13:50.300545	1.031	0.948
6-Small-4	5	21	0.42	0	0	0	0:13:58.846904	0:13:57.286213	0.998	0.868
7-10.B.9	10	21	0.105	0	0	0	0:00:00.013363	0:00:00.011885	0.889	0.907
8-M	7	20	0.204	0	0	0	0:00:01.388660	0:00:01.400180	1.008	0.916
9-10.B.6	10	20	0.1	0	0	0	0:00:00.041055	0:00:00.033373	0.813	0.885
10-ManyWalks	9	18	0.111	0	0	0	0:12:26.054415	0:12:24.664902	0.998	0.991
11-10.B.10	10	18	0.09	0	0	0	0:00:01.703048	0:00:01.751324	1.028	0.934
12-EG5	5	16	0.32	0	0	0	0:41:18.876403	0:41:29.264245	1.004	0.996
13-keven1	3	5	0.278	0	0	0	0:00:00.000661	0:00:00.000619	0.937	1.408

Table 4.3: Comparison of running times for SRR-PTC and ProbTC on irreducible examples (using the exact version of ProbTC)

Name	#Vertices	#Arcs	Arc-Density	#RV(1 or 2)	#RV(3)	Total #RV	ProbTC Run-Time	SRR-Run-Time	TR(SRR to ProbTC)
1-Western.Full	31	46	0.024	11	6	17	0:00:28.652988	0:00:00.480606	0.017
2-elderin	15	29	0.064	1	3	4	0:00:02.542396	0:00:00.023575	0.009
3-10.B.3	10	22	0.11	0	1	1	0:00:00.301837	0:00:00.272007	0.901
4-10.B.4	10	20	0.1	0	2	2	0:00:00.046524	0:00:00.014922	0.321
5-10.B.5	10	19	0.095	1	0	1	0:00:09.017166	0:00:08.766976	0.972
6-17-10.A.2	10	19	0.095	1	4	5	0:00:00.110724	0:00:00.012789	0.116
7-10.B.1	10	18	0.09	1	3	4	0:00:00.025112	0:00:00.009052	0.36
8-10.B.7	10	17	0.085	3	1	4	0:00:00.011205	0:00:00.005810	0.519
9-10.B.7	10	17	0.085	3	1	4	0:00:00.011129	0:00:00.006257	0.562
10-10.A.6	10	15	0.075	1	1	2	0:00:00.026739	0:00:00.020122	0.753
11-10.A.3	10	14	0.07	2	1	3	0:00:00.008377	0:00:00.004843	0.578
12-10.A.10	10	12	0.06	4	0	4	0:00:00.008584	0:00:00.002828	0.329
13-BoolGood4	7	9	0.092	1	0	1	0:00:00.002814	0:00:00.002300	0.817
14-10.A.9	10	9	0.045	6	0	6	0:00:00.006550	0:00:00.001063	0.162
15-10.A.4	10	7	0.035	7	0	7	0:00:00.006151	0:00:00.000609	0.099
16-10.A.8	10	6	0.03	2	1	3	0:00:00.005991	0:00:00.002606	0.435
17-10.A.5	10	5	0.025	4	1	5	0:00:00.005146	0:00:00.001063	0.207

Table 4.4: Comparison of running times for SRR-PTC and ProbTC on reducible examples (using the exact version of ProbTC)

4.4 Conclusion

The Reduction-Recovery Algorithm was developed to reduce the running time of PTC algorithms. As shown in this chapter, it was implemented in the form of two programs, SRR-PTC and RRR-PTC. Our testing shows that the SRR-PTC program runs faster than ProbTC for reducible digraphs. Furthermore, it has almost the same running times as ProbTC for irreducible examples. Hence, the enhanced algorithm is worth using for finding the PTC regardless of the type of the input digraph. Also, we observe that SRR-PTC and RRR-PTC, our two implementations of the Reduction-Recovery Algorithm, have practically the same running time.

Chapter 5

Algorithm Enhancement: the Separation-Reduction Algorithm

In Chapter 4, we realized that the reduction of the input digraph makes the computation of the probabilistic transitive closure (PTC) faster. In particular, we saw that the reduction of the input digraph by deleting vertices of indegree at most one and outdegree at most one makes the computation of PTC more efficient. In this chapter, we propose a new enhancement for the computation of PTC. We state and prove a theorem that explains how to obtain the PTC of an FCM from the PTCs of smaller FCMs by using a separating vertex. Furthermore, we develop an algorithm, called Separation-Reduction, based on this theorem, and provide a pseudocode version for it.

5.1 Preliminaries

In this section we review all the essential graph terminology and tools that we need to know before proving the main theorem. We start by defining a separating vertex and separation in graphs and digraphs, and then we extend these definitions to bipolar weighted digraphs.

Definition 5.1.1. A *decomposition* of a graph $G = (V, E)$ is a set $\{G_1, \ldots, G_k\}$ of subgraphs of G such that

(1) $G = G_1 \cup G_2 \cup \ldots \cup G_k$ and

(2) $\{E(G_1), \ldots, E(G_k)\}$ is a partition of $E(G)$.

Definition 5.1.2. A vertex u in a graph $G = (V, E)$ with at least two vertices is called a *cut vertex* if $c(G - u) > c(G)$; that is, the graph $G - u$ has more connected components than G.

Definition 5.1.3. Let $G = (V, E)$ be a graph and $u \in V$. Then u is called a *separating vertex* of G if G has a decomposition $\{G_1, G_2\}$ such that G_1 and G_2 are non-empty subgraphs of G with just vertex u in common; that is, $G_1 \cap G_2 = (\{u\}, \emptyset)$. If this is the case, then $\{G_1, G_2\}$ is called a *separation* of G (with respect to vertex u).

The following lemma shows the connection between the notions of a cut vertex and a separating vertex in a graph G.

Lemma 5.1.4. *Let $G = (V, E)$ be a connected graph and $u \in V$ such that there is no loop incident with u. Then u is a cut vertex of G if and only if u is a separating vertex of G.*

Proof: Assume u is a cut vertex of G, so $c(G - u) > c(G)$; that is, the graph $G - u$ has at least two connected components. Let G_1 be one connected component of $G - u$, and G_2 the union of the remaining connected components, so $G - u = G_1 \cup G_2$. Define G'_i for $i = 1, 2$, as

$$G'_i = G[V'_i] \text{ where } V'_i = V(G_i) \cup \{u\}.$$

Then $\{G'_1, G'_2\}$ is a decomposition of G such that G'_1 and G'_2 have just vertex u in common. The graph G is connected so G'_1 and G'_2 are connected as well, and since each has at least two vertices, they are non-empty. Hence, $\{G'_1, G'_2\}$ is a separation, and by Definition 5.1.3, u is a separating vertex of G.

Assume u is a separating vertex of G. By Definition 5.1.3, G has a decomposition $\{G_1, G_2\}$ such that G_1 and G_2 are non-empty subgraphs of G, and $G_1 \cap G_2 = (\{u\}, \emptyset)$. Since G_1 and G_2 are non-empty subgraphs, each has at least one edge. By assumption, this edge cannot be a loop incident with u; hence there exist $v_1 \in V(G_1)$ and $v_2 \in V(G_2)$ with $v_1 \neq u$ and $v_2 \neq u$. We know G is connected so there exists a (v_1, v_2)-path in G. Since G_1 and G_2 have only vertex u in common, any such path must contain vertex u. In $G - u$, there is no (v_1, v_2)-path, so $G - u$ is disconnected. Therefore $c(G - u) > c(G)$, which implies that u is a cut vertex.

∎

We now extend the above definitions to digraphs.

Definition 5.1.5. A *decomposition* of a digraph $D = (V, A)$ is a set $\{D_1, \ldots, D_k\}$ of subdigraphs of D such that

(1) $D = D_1 \cup D_2 \cup \ldots \cup D_k$ and

(2) $\{A(D_1), \ldots, A(D_k)\}$ is a partition of $A(D)$.

Definition 5.1.6. Let $D = (V, A)$ be a digraph and $u \in V$. Then u is called a *separating vertex* of D if D has a decomposition $\{D_1, D_2\}$ such that D_1 and D_2 are non-empty subdigraphs of D with just vertex u in common; that is, $D_1 \cap D_2 = (\{u\}, \emptyset)$. If this is the case, then $\{D_1, D_2\}$ is called a *separation* of D (with respect to vertex u).

In the following lemma, we explain that a separating vertex of the underlying undirected graph G of a digraph D is also a separating vertex of D, and vice-versa.

Lemma 5.1.7. *Let $D = (V, A)$ be a digraph, $G = (V, E)$ its underlying undirected graph, and $u \in V$. Then u is a separating vertex of D if and only if u is a separating vertex of G.*

Proof: Assume u is a separating vertex of D, so there is a decomposition $\{D_1, D_2\}$ of D such that

(1) $V(D_1) \cap V(D_2) = \{u\}$ and

(2) $A(D_i) \neq \emptyset$ for $i = 1, 2$.

For $i = 1, 2$, let G_i be the underlying undirected graph of the digraph D_i. Since there exists a one-to-one correspondence between the edges in G_i and the arcs of D_i, we can conclude that $\{G_1, G_2\}$ is a decomposition of G,

(1′) $V(G_1) \cap V(G_2) = \{u\}$, and

(2′) $E(G_i) \neq \emptyset$ for $i = 1, 2$.

With these properties, $\{G_1, G_2\}$ is a separation of G with respect to u, so u is a separating vertex of G.

Conversely, assume u is a separating vertex of G, so there is a decomposition $\{G_1, G_2\}$ of G such that (1′) and (2′) hold. For $i = 1, 2$, we define $D_i = (V_i, A_i)$ where

- $V(D_i) = V(G_i)$ and
- $A(D_i) = \{a \in A(D) : a = (u, v),\ u, v \in V(D_i)\}$;

that is, $D_i = D[V(G_i)]$. With such a definition for D_i we can easily conclude that G_i is the underlying undirected graph of D_i. From the one-to-one correspondence of the edges in G_i with the arcs of D_i we can conclude $\{D_1, D_2\}$ is a decomposition of D satisfying (1) and (2) above. With these properties, $\{D_1, D_2\}$ is a separation of D with respect to u, so u is a separating vertex of D. ∎

Definition 5.1.8. Let $D = (V, A, p)$ be a bipolar weighted digraph and $u \in V$. Then u is called a *separating vertex* of D if u is a separating vertex of (V, A). In addition, assume $\{(V_1, A_1), (V_2, A_2)\}$ is a separation of (V, A) and for $i = 1, 2$, define $p_i : A_i \to [0, 1]$ as follows:

$$p_i(a) = p(a) \quad \text{for all} \quad a \in A_i.$$

Then $\{D_1, D_2\}$, where $D_i = (V_i, A_i, p_i)$ for $i = 1, 2$, will be called a *separation* of D.

5.2 Main Theorem

The following theorem forms the basis for the reduction of a bipolar weighted digraph using a separating vertex. It states that if $\{D_1, D_2\}$ is a separation of a bipolar weighted digraph D satisfying certain conditions, then we can recover the probabilistic transitive closure of D from the probabilistic transitive closures of the bipolar weighted digraphs D_1 and D_2. The PTCs of D_1 and D_2 can be computed directly by using one of the existing algorithms or recursively using a further reduction. This theorem is the basis for the Separation-Reduction Algorithm (Algorithm 5 in Section 5.3).

Theorem 5.2.1. *Let $R = (V, p)$ be a bipolar random digraph and $D = (V, A, p)$ its support digraph. Assume that the underlying undirected graph of D is connected. Let $\{D_1, D_2\}$ be a separation of D with respect to a separating vertex u, and assume $(u, u, 1), (u, u, -1) \notin A$. For $i = 1, 2$, let $R_i = (V_i, p_i)$ be the bipolar random digraph corresponding to D_i, and $R_i^\infty = (V_i, p_i^\infty)$ the PTC of R_i.*

Let $s, t \in V$, $\sigma \in \Omega$, and $i, j \in \{1, 2\}$ be such that $i \neq j$.

(1) If $s, t \in V_i$ and $p_j^\infty(u, u, -1) = 0$, then

$$p^\infty(s, t, \sigma) = p_i^\infty(s, t, \sigma).$$

(2) If $s \in V_i - \{u\}$ and $t \in V_j - \{u\}$, and there exists $\sigma_s \in \Omega$ such that $p_i^\infty(s, u, -\sigma_s) = 0$, then

$$p^\infty(s, t, \sigma) = p_i^\infty(s, u, \sigma_s) p_j^\infty(u, t, \sigma\sigma_s).$$

(3) If $s \in V_i - \{u\}$ and $t \in V_j - \{u\}$, and there exists $\sigma_t \in \Omega$ such that $p_j^\infty(u, t, -\sigma_t) = 0$, then

$$p^\infty(s, t, \sigma) = p_i^\infty(s, u, \sigma\sigma_t) p_j^\infty(u, t, \sigma_t).$$

Proof: Let $(\mathcal{D}, \mathcal{E}, P)$ and $(\mathcal{D}_i, \mathcal{E}_i, P_i)$, for $i = 1, 2$, be the probability spaces associated with R and R_i, respectively. In this proof, for clarity and convenience, we shall write $P(V, B)$ to mean $P(\{(V, B)\})$, for any $B \subseteq V^2 \times \Omega$.

Let $U = V^2 \times \Omega$, and for $i = 1, 2$, let $U_i = V_i^2 \times \Omega$, $U_i' = U_i - \{a^+, a^-\}$, where $a^+ = (u, u, 1)$ and $a^- = (u, u, -1)$, and $V_i' = V_i - \{u\}$. Then

$$U = U_1' \dot\cup U_2' \dot\cup U_3 \quad \text{where} \quad U_3 = (V_1' \times V_2' \times \Omega) \cup (V_2' \times V_1' \times \Omega) \cup \{a^+, a^-\}.$$

Note that $a^+, a^- \in U_3$.

First, we make some preliminary observations to assist with the main calculation.

(a) Since D has no arc with one endpoint in V_1' and the other in V_2', and no loop at u, we have $A \cap U_3 = \emptyset$. So for all $B \subseteq A$ we have $B = B_1 \dot\cup B_2$ for some $B_1 \subseteq U_1'$ and $B_2 \subseteq U_2'$. Then

$$U - B = U - (B_1 \cup B_2) = U_3 \dot\cup (U_1' - B_1) \dot\cup (U_2' - B_2).$$

(b) For $i = 1, 2$, let $A_i = A(D_i)$ so that $D_i = (V_i, A_i, p_i|_{A_i})$. Using Definition 5.1.8, we have

$$p_i(a) = p(a) \quad \text{for all} \quad a \in A_i.$$

For all $a \in U_i - A_i$, including $a \in \{a^+, a^-\}$, we have $p_i(a) = 0 = p(a)$. Hence, $p_i(a) = p(a)$ for all $a \in U_i$.

(c) For any $s, t \in V_i$, and $\sigma \in \Omega$, using Lemma 3.2.2, we obtain

$$\begin{aligned}
p_i^\infty(s,t,\sigma) &= \sum_{\substack{B_i \subseteq U_i \\ (V_i, B_i) \text{ has an } (s,t,\sigma)\text{-walk}}} P_i(V_i, B_i) \\
&= \sum_{\substack{B_i \subseteq U_i \\ (V_i, B_i) \text{ has an } (s,t,\sigma)\text{-walk}}} \prod_{a \in B_i} p_i(a) \prod_{a \in U_i - B_i} (1 - p_i(a)) \\
&= \sum_{\substack{B_i \subseteq U_i \\ (V_i, B_i) \text{ has an } (s,t,\sigma)\text{-walk}}} \prod_{a \in B_i} p(a) \prod_{a \in U_i - B_i} (1 - p(a)) \\
&= \sum_{\substack{B_i \subseteq U_i' \\ (V_i, B_i) \text{ has an } (s,t,\sigma)\text{-walk}}} \prod_{a \in B_i} p(a) \prod_{a \in U_i' - B_i} (1 - p(a)).
\end{aligned}$$

Note that the second to last equality follows from **(b)**, and the last one from $p(a^-) = p(a^+) = 0$.

We shall now prove Statement (1)–(3) of the theorem.

(1) Take any $s, t \in V_1$, and $\sigma \in \Omega$, and assume that $p_2^\infty(u, u, -1) = 0$.

Claim 1. Let $B \subseteq A$ and $B = B_1 \dot\cup B_2$, where $B_1 \subseteq U_1'$ and $B_2 \subseteq U_2'$. Then (V, B) has an (s, t, σ)-walk if and only if (V_1, B_1) has an (s, t, σ)-walk.

Proof. Let W be any (s, t, σ)-walk in (V, B). There are two cases to consider:

(i) The walk W does not traverse u. We know $\{D_1, D_2\}$ is a separation of D with respect to vertex u. Since the starting point and end point of W are both in D_1, and W does not traverse u, the walk W is fully in (V_1, B_1). Therefore (V_1, B_1) has an (s, t, σ)-walk.

(ii) The walk W traverses u. Therefore W can be written as a concatenation $W = W_1 W_2 \dots W_{2k+1}$, where

$$\begin{gathered} W_1 \text{ is an } (s,u)\text{-walk in } (V_1, B_1), \\ W_2 \text{ is a } (u,u)\text{-walk in } (V_2, B_2), \\ \vdots \\ W_{2i} \text{ is a } (u,u)\text{-walk in } (V_2, B_2), \\ W_{2i+1} \text{ is } (u,u)\text{-walk in } (V_1, B_1), \\ \vdots \\ W_{2k+1} \text{ is a } (u,t)\text{-walk in } (V_1, B_1). \end{gathered}$$

Define W_1' as the concatenation of walks $W_1, W_3, \dots, W_{2k+1}$, which are all in (V_1, B_1), and W_2' as the concatenation of walks $W_2, W_4, \dots, W_{2k}$, which are all in (V_2, B_2). Consequently, W_1' is an (s,t)-walk in (V_1, B_1) and W_2' is a (u,u)-walk in (V_2, B_2). Then $\text{sign}(W_1') = \text{sign}(W_1)\text{sign}(W_3)\dots\text{sign}(W_{2k+1})$, and $\text{sign}(W_2') = \text{sign}(W_2)\text{sign}(W_4)\dots\text{sign}(W_{2k})$. Hence,

$$\text{sign}(W) = \prod_{i=1}^{2k+1} \text{sign}(W_i) = \text{sign}(W_1')\text{sign}(W_2').$$

We have $\text{sign}(W) = \sigma$ and by assumption we know that $p_2^\infty(u, u, -1) = 0$. That means $\text{sign}(W_2') = 1$. Therefore the sign of W_1' is σ, so W_1' is an (s, t, σ)-walk in (V_1, B_1).

Conversely, it can be easily seen that if (V_1, B_1) has an (s, t, σ)-walk, then (V, B) has an (s, t, σ)-walk since (V_1, B_1) is a subdigraph of (V, B).

This completes the proof of Claim 1.

We use **(a)** and Claim 1 to justify the second equality below. Using Lemma 3.2.2, we have

$$\begin{aligned} & p^\infty(s,t,\sigma) \\ = & \sum_{\substack{B \subseteq V^2 \times \Omega \\ (V,B) \text{ has an } (s,t,\sigma)\text{-walk}}} P(V,B) \end{aligned}$$

$$
\begin{aligned}
&= \sum_{\substack{B_1 \subseteq U_1' \\ (V_1,B_1) \text{ has an } (s,t,\sigma)\text{-walk}}} \sum_{B_2 \subseteq U_2'} P(V, B_1 \cup B_2) \\
&= \sum_{\substack{B_1 \subseteq U_1' \\ (V_1,B_1) \text{ has an } (s,t,\sigma)\text{-walk}}} \sum_{B_2 \subseteq U_2'} \prod_{a \in B_1 \cup B_2} p(a) \prod_{a \in U - (B_1 \cup B_2)} (1 - p(a)) \\
&= \sum_{\substack{B_1 \subseteq U_1' \\ (V_1,B_1) \text{ has an } (s,t,\sigma)\text{-walk}}} \sum_{B_2 \subseteq U_2'} \prod_{a \in B_1} p(a) \prod_{a \in U_1' - B_1} (1 - p(a)) \cdot \\
&\qquad \cdot \prod_{a \in B_2} p(a) \prod_{a \in U_2' - B_2} (1 - p(a)) \cdot \prod_{a \in U_3} (1 - p(a)) \\
&= \left(\sum_{\substack{B_1 \subseteq U_1' \\ (V_1,B_1) \text{ has an} \\ (s,t,\sigma)\text{-walk}}} \prod_{a \in B_1} p(a) \prod_{a \in U_1' - B_1} (1 - p(a)) \right) \left(\sum_{B_2 \subseteq U_2'} \prod_{a \in B_2} p(a) \prod_{a \in U_2' - B_2} (1 - p(a)) \right) \\
&= \left(\sum_{\substack{B_1 \subseteq U_1' \\ (V_1,B_1) \text{ has an} \\ (s,t,\sigma)\text{-walk}}} \prod_{a \in B_1} p(a) \prod_{a \in U_1' - B_1} (1 - p(a)) \right) \left(\sum_{B_2 \subseteq U_2} \prod_{a \in B_2} p(a) \prod_{a \in U_2 - B_2} (1 - p(a)) \right) \\
&= p_1^{\infty}(s, t, \sigma).
\end{aligned}
$$

Note that the fifth equality is true as we know $p(a) = 0$ if $a \in U_3$, and the sixth equality holds since $p(a) = 0$ if $a \in U_2 - U_2'$. Finally, by using **(c)** and Lemma 3.1.4**(ii)**, respectively, we justify the last equality.

The proof for the case $s, t \in V_2$ is very similar.

(2) Take any $s \in V_1'$, $t \in V_2'$, $\sigma \in \Omega$, and assume there exists $\sigma_s \in \Omega$ such that $p_1^{\infty}(s, u, -\sigma_s) = 0$.

Claim 2. Let $B \subseteq A$ and $B = B_1 \dot{\cup} B_2$, where $B_1 \subseteq U_1'$ and $B_2 \subseteq U_2'$. Then (V, B) has an (s, t, σ)-walk if and only if (V_1, B_1) has an (s, u, σ_s)-walk and (V_2, B_2) has an $(u, t, \sigma\sigma_s)$-walk.

Proof. Suppose W is any (s, t, σ)-walk in (V, B). We know D is connected and $\{D_1, D_2\}$ is a separation of D with respect to vertex u, which means that $D_1 \cap D_2$ is a digraph with vertex set $\{u\}$. Hence every walk with starting point in D_1 and endpoint in D_2 passes through u. Therefore W can be written as a concatenation $W = W_1 W_2 \dots W_{2k}$, where

$$
\begin{gathered}
W_1 \text{ is an } (s,u)\text{-walk in } (V_1,B_1),\\
W_2 \text{ is a } (u,u)\text{-walk in } (V_2,B_2),\\
\vdots\\
W_{2i+1} \text{ is a } (u,u)\text{-walk in } (V_1,B_1),\\
W_{2i} \text{ is a } (u,u)\text{-walk in } (V_2,B_2),\\
\vdots\\
W_{2k} \text{ is a } (u,t)\text{-walk in } (V_2,B_2).
\end{gathered}
$$

Define W_1' as the concatenation of walks $W_1, W_3, \ldots, W_{2k-1}$, which are all in (V_1, B_1), and W_2' as the concatenation of walks $W_2, W_4, \ldots, W_{2k}$, which are all in (V_2, B_2). Consequently, W_1' is an (s,u)-walk in (V_1,B_1) and W_2' is a (u,t)-walk in (V_2,B_2). Then $\operatorname{sign}(W_1') = \operatorname{sign}(W_1)\operatorname{sign}(W_3)\ldots\operatorname{sign}(W_{2k-1})$, and $\operatorname{sign}(W_2') = \operatorname{sign}(W_2)\operatorname{sign}(W_4)\ldots\operatorname{sign}(W_{2k})$. Hence,

$$\operatorname{sign}(W) = \prod_{i=1}^{2k} \operatorname{sign}(W_i) = \operatorname{sign}(W_1')\operatorname{sign}(W_2').$$

By assumption, we know that $p_1^\infty(s,u,-\sigma_s) = 0$, so $\operatorname{sign}(W_1') = \sigma_s$. Also, we have $\operatorname{sign}(W) = \sigma$. Therefore $\operatorname{sign}(W_2') = \sigma\sigma_s$. We conclude that W_1' is an (s,u,σ_s)-walk in (V_1,B_1), and W_2' is a $(u,t,\sigma\sigma_s)$-walk in (V_2,B_2).

Conversely, it is easy to see that if there is an (s,u,σ_s)-walk in (V_1,B_1) and a $(u,t,\sigma\sigma_s)$-walk in (V_2,B_2), the concatenation of these two walks is an (s,t,σ)-walk in (V,B).

This completes the proof of Claim 2.

We use Claim 2 to justify the second equality below. Using Lemma 3.2.2, we have

$$
\begin{aligned}
& p^\infty(s,t,\sigma)\\
= & \sum_{\substack{B\subseteq U\\ (V,B)\text{ has an }(s,t,\sigma)\text{-walk}}} P(V,B)\\
= & \sum_{\substack{B_1\subseteq U_1'\\ (V_1,B_1)\text{ has an }(s,u,\sigma_s)\text{-walk}}} \sum_{\substack{B_2\subseteq U_2'\\ (V_2,B_2)\text{ has an }(u,t,\sigma\sigma_s)\text{-walk}}} P(V,B_1\cup B_2)\\
= & \sum_{\substack{B_1\subseteq U_1'\\ (V_1,B_1)\text{ has an }(s,u,\sigma_s)\text{-walk}}} \sum_{\substack{B_2\subseteq U_2'\\ (V_2,B_2)\text{ has an }(u,t,\sigma\sigma_s)\text{-walk}}} \prod_{a\in B_1\cup B_2} p(a) \prod_{a\in U-(B_1\cup B_2)} (1-p(a))
\end{aligned}
$$

$$= \sum_{\substack{B_1 \subseteq U_1' \\ (V_1,B_1) \text{ has an } (s,u,\sigma_s)\text{-walk}}} \sum_{\substack{B_2 \subseteq U_2' \\ (V_2,B_2) \text{ has an } (u,t,\sigma\sigma_s)\text{-walk}}} \prod_{a \in B_1} p(a) \prod_{a \in U_1' - B_1} (1-p(a)) \cdot$$
$$\cdot \prod_{a \in B_2} p(a) \prod_{a \in U_2' - B_2} (1-p(a)) \cdot \prod_{a \in U_3} (1-p(a))$$

$$= \left(\sum_{\substack{B_1 \subseteq U_1' \\ (V_1,B_1) \text{ has an } (s,u,\sigma_s)\text{-walk}}} \prod_{a \in B_1} p(a) \prod_{a \in U_1' - B_1} (1-p(a)) \right) \cdot$$
$$\cdot \left(\sum_{\substack{B_2 \subseteq U_2' \\ (V_2,B_2) \text{ has an } (u,t,\sigma\sigma_s)\text{-walk}}} \prod_{a \in B_2} p(a) \prod_{a \in U_2' - B_2} (1-p(a)) \right)$$

$$= p_1^\infty(s,u,\sigma_s) p_2^\infty(u,t,\sigma\sigma_s).$$

Note that for the second to last equality, we used the fact that $p(a) = 0$ for all $a \in U_3$, and for the last equality we used observation **(c)**.

A similar proof can be used for the case $s \in V_2'$, $t \in V_1'$.

(3) Take any $s \in V_i'$, $t \in V_j'$, for $i \neq j$, and $\sigma \in \Omega$, and assume there exists $\sigma_t \in \Omega$ such that $p_j^\infty(u,t,-\sigma_t) = 0$.

We then prove that $p^\infty(s,t,\sigma) = p_i^\infty(s,u,\sigma\sigma_t) p_j^\infty(u,t,\sigma_t)$ in a manner analogous to the proof of (2).

∎

5.3 Separation-Reduction Algorithm

We are now ready to describe an algorithm, Algorithm 5, based on Theorem 5.2.1 for a more efficient computation of PTC. This algorithm takes as input a bipolar weighted digraph $D = (V, A, p)$, and returns its PTC. For a separating vertex u of D and a separation $\{D_1, D_2\}$ with respect to vertex u, Algorithm 5 starts by computing the PTCs of D_1 and D_2. The PTCs of D_1 and D_2 can be computed directly by using one of the existing algorithms or recursively using a further reduction. Then, if the PTCs of D_1 and D_2 satisfy the conditions of Theorem 5.2.1, it starts recovering the PTC of D from the PTCs of D_1 and D_2. If, however, the conditions fail, then another separation is considered, and if no separation with respect to u satisfies the conditions of Theorem 5.2.1, then a different separating vertex is examined.

Note that algorithms for finding separating vertices are well known; for example, such an algorithm is described in Chapter 12 of [9]. (Note that in this source, separating vertices are called articulation points.)

Algorithm 5 Separation-Reduction Algorithm

Input: bipolar random digraph $R = (V, p)$ with support digraph $D = (V, A, p)$
Output: (V, p^∞) the transitive closure of (V, p)

Flag:=False
while D has a separating vertex u **do**
  for all separations $\{D_1, D_2\}$ of D **do**
    for $i = 1, 2$ **do**
      $(V_i, p_i) :=$ bipolar random digraph corresponding to D_i.
      $(V_i, {p_i}^\infty) :=$ transitive closure of (V_i, p_i)
    end for
    Condition:=False
    if $p_1^\infty(u, u, -1) = 0$ and $p_2^\infty(u, u, -1) = 0$ **then**
      Condition:=True
      for all $s, t \in V$ **do**
        if $s \in V_i - \{u\}$, and $t \in V_j - \{u\}$ for $i \neq j$, and there exists $\sigma_s \in \Omega$ such that $p_i^\infty(s, u, -\sigma_s) = 0$ **then**
          $p^\infty(s, t, \sigma) = p_i^\infty(s, u, \sigma_s) p_j^\infty(u, t, \sigma\sigma_s)$
        else if $s \in V_i - \{u\}$, and $t \in V_j - \{u\}$ for $i \neq j$, and there exists $\sigma_t \in \Omega$ such that $p_j^\infty(u, t, -\sigma_t) = 0$ **then**
          $p^\infty(s, t, \sigma) = p_i^\infty(s, u, \sigma\sigma_t) p_j^\infty(u, t, \sigma_t)$
        else if $s, t \in V_i$ **then**
          $p^\infty(s, t, \sigma) = p_i^\infty(s, t, \sigma)$
        else
          Condition:=False
          break #break out of "for all $s, t \in V$" loop.
        end if
      end for
    end if
    if Condition=True **then**
      Flag:=True
      break #break out of "for all separations $\{D_1, D_2\}$" loop.
    end if
  end for
  if Flag=True **then**
    break #break out of "while" loop.
  end if
end while
if Flag=True **then**
  return (V, p^∞)
else
  print ("no suitable separation")
end if

Chapter 6

A Study on Work-Integrated Learning at Shopify using Probabilistic Transitive Closure of Fuzzy Cognitive Maps

6.1 Introduction

A work-integrated learning program creates a bridge between academia and industry. It provides a situation wherein industries can use universities' knowledge, and universities can put the results of their research into practice. Shopify, a Canadian multinational e-commerce company, has created the DevDegree program for undergraduate students in computer science. This program is a work-integrated learning program that allows each student to study towards a B.Sc degree at a partner university and simultaneously work as an intern at Shopify. Students usually take three courses per semester at a partner university and spend 25 hours each week at Shopify. In the workplace, students are assigned mentors who help them improve their engineering skills and technical expertise. Throughout the program, students learn how to apply theory to problems in the real world.

A number of studies have been conducted on work-integrated learning in recent years. For example, the impact of work-integrated learning on employment readiness was studied in [22]. A report on the outcomes of a university forum that discussed various aspects of work-integrated learning programs was published in [20]. The results of a survey on work-integrated learning were analyzed at a forum of information and communication technology educational leaders in [21]. In all of these studies, data was collected using questionnaires and analyzed using various statistical methods.

The objective of this study is to provide further understanding of the determinants of success and well-being of a student intern via FCMs and probabilistic transi-

tive closure. As far as we know, this is the first study ever to address work-integrated learning using cognitive mapping.

6.2 Approach

Our main goal was to examine the determinants of success and well-being of a DevDegree student using probabilistic transitive closure of FCMs. First, we collected three FCMs which were created by the three groups (group of students, group of mentors, and the DevDegree team) during two mapping sessions. Then, the PTCs of the maps were calculated using existing software, ProbTC [13], and compared.

6.3 Data Collection

Two mapping sessions were held to collect the data. In the first session, the concepts of an FCM and PTC were explained, and the participants (students, members of DevDegree team, and mentors) were asked to each create their own FCM that best represents their view of the issue. They created their FCMs following the facilitators' instructions. First, they listed 10–20 factors which (in their opinion) have a direct or indirect impact on the success and well-being of a DevDegree student. One of the factors had to represent success and another one well-being. They were allowed to split these two factors further into several factors, for example, Academic Success and Work Success, and Physical Well-Being and Emotional Well-Being. They drew ovals and arcs (arrows) to represent factors and direct impacts, respectively. In particular, they joined factor F to factor F' by an arc if and only if they believed that factor F has a direct impact on factor F'. They were instructed to draw two parallel arcs from F to F' (one positive, one negative) if they believed that factor F has both a positive and a negative direct impact on factor F'. Furthermore, for each direct impact they chose the sign and the weight. For weights, they were instructed to use the following scale:

- weight 0 — no impact,
- weights 0.1, 0.2, 0.3 — mild impact,
- weights 0.4, 0.5, 0.6 — moderate impact,
- weights 0.7, 0.8, 0.9 — strong impact,
- weight 1 — very strong impact.

At the end of the first session, the participants submitted their individual maps and we collected them. Then we reviewed the individual maps, and compiled a (short)

list of factors that best represent all of the individual maps. The chosen factors were named as follows. (Alternative descriptions obtained from the individual maps are provided in brackets as needed.)

F1 — Academic Success (academic performance and standing)

F2 — Ambition (drive to succeed, challenge seeking)

F3 — Financial Security

F4 — Good Communication Skills (Intern)

F5 — Healthy Lifestyle (appropriate exercise, adequate sleep/rest, good nutrition, free time, recreational activities, meditation)

F6 — Knowledge and Skills (computer science knowledge, developer skills, domain knowledge)

F7 — Mental Well-Being (emotional well-being, happiness, lack of stress, lack of anxiety)

F8 — Peer Support at Shopify (friends within DevDegree)

F9 — Personal Support Network (friends, family, school support)

F10 — Physical Well-Being (good physical health)

F11 — Positive Personality Traits (confidence, self-awareness, positive attitude, maturity, curiosity, enthusiasm, self-motivation)

F12 — Positive Relationship with Mentor (good communication, trust, empowering, mentor competence, regular feedback)

F13 — Positive Team Environment (positive team dynamics/relationships, good communication, trust, social activities)

F14 — Positive Work Habits (good time management, diligence, participation, accountability)

F15 — School/Work Overload (unsuitable schedule, work/school-related pressure)

F16 — Well-Chosen Projects (meaningful, impactful, well-defined projects; appropriate level, clear expectations)

F17 — Work Success (high productivity, quality work, completion of training, successful performance evaluations)

Observe that the central issues (success and well-being) were split into four central outcomes: Academic Success, Work Success, Physical Well-Being, and Mental Well-Being.

During the second mapping session, each of the three groups was asked to construct a consensus map using the provided list of factors. They had to follow the same steps as they followed in the first session to create the consensus map, but in the second session they had to use the prescribed factors and discuss each direct impact, its weight, and its sign with other team members to reach an agreement.

The FCMs of the determinants of success and well-being created by the three groups are shown in Figures 6.1–6.3. Each factor is represented by an oval, and four central outcomes are shown in a different colour (not blue). Arcs (arrows) represent direct impacts. Black and red arcs represent positive and negative impacts, respectively. When a factor F has both a positive and a negative direct impact on a factor F', the figure shows two arcs from F to F', one positive and one negative. For example, in Figure 6.2, Ambition has both a positive and a negative direct impact on Academic Success. Each arc has a weight that is a real number between 0 and 1 (inclusive).

6.4 Computation

Each of the three FCMs was converted to a pair of weighted adjacency matrices, which served as input for the PTC code. We used ProbTC code [13] or the new SRR-PTC code (if ProbTC took a long time) to compute the probabilistic transitive closure of each FCM. The weighted adjacency matrices of each FCM and its PTC are shown in Tables 6.1–6.6. Note that the row and column labels correspond to the seventeen factors as labelled on Page 49. Tables 6.1, 6.3, and 6.5 represent the input matrices for the three FCMs. The black figures represents the weights of the positive impacts, and the red figures represents the weights of the negative impacts; if an entry has no red figure, then it is assumed to be 0. For clarity, a negative sign is included with each red figure. Each table representing a pair of input matrices is followed by a table representing the PTC of the corresponding FCM (Tables 6.2, 6.4, and 6.6), and output tables similarly encode both positive and negative impacts. In Tables 6.2, 6.4, and 6.6, each output figure was rounded to three decimals.

6.5 Discussion

The data describes, for each of the three groups of participants, the collective viewpoint of the group on the topic of work-integrated learning. The output of our PTC code shows the total impacts of all included factors on (among others) the four central outcomes: Academic Success, Work Success, Physical Well-Being, and Mental

6. A STUDY ON WORK-INTEGRATED LEARNING AT SHOPIFY USING PROBABILISTIC TRANSITIVE CLOSURE OF FUZZY COGNITIVE MAPS51

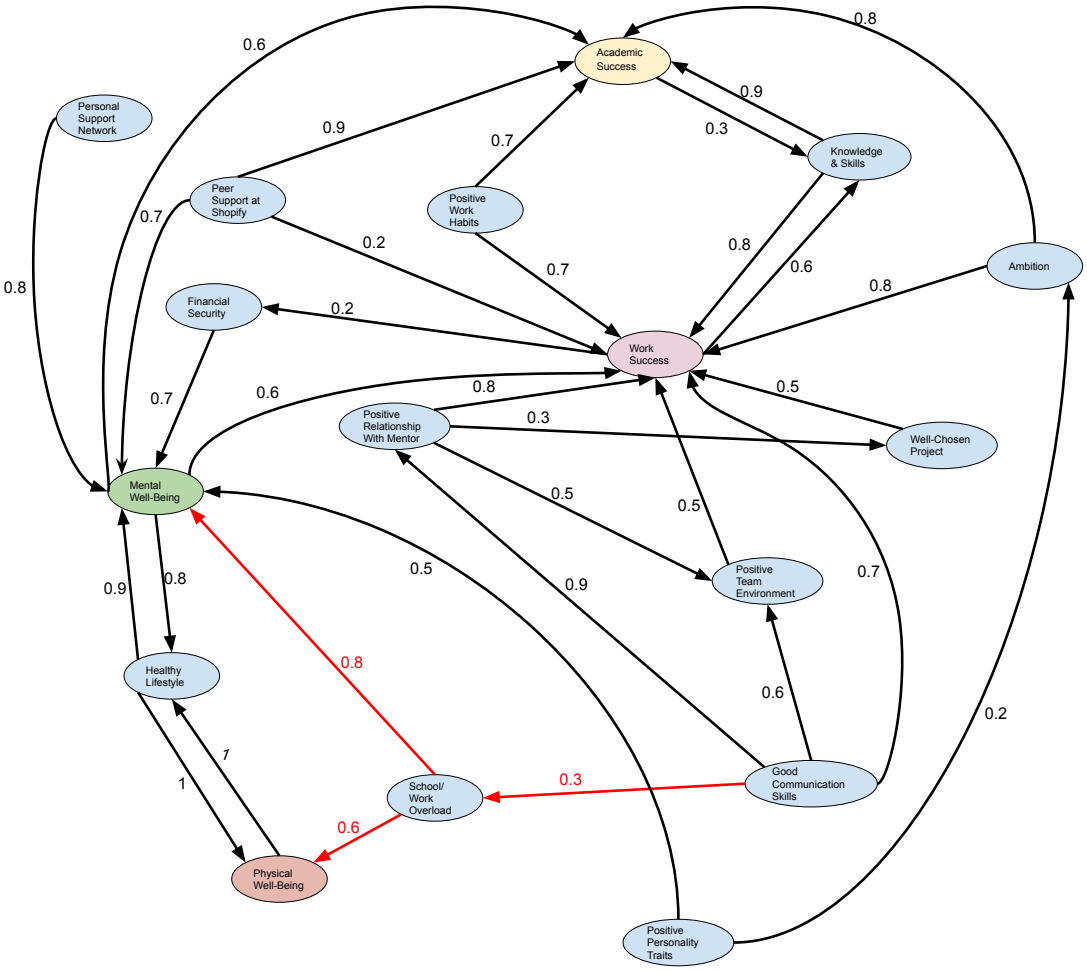

Figure 6.1: The FCM created by the group of student participants (Student FCM).

6. A STUDY ON WORK-INTEGRATED LEARNING AT SHOPIFY USING PROBABILISTIC TRANSITIVE CLOSURE OF FUZZY COGNITIVE MAPS52

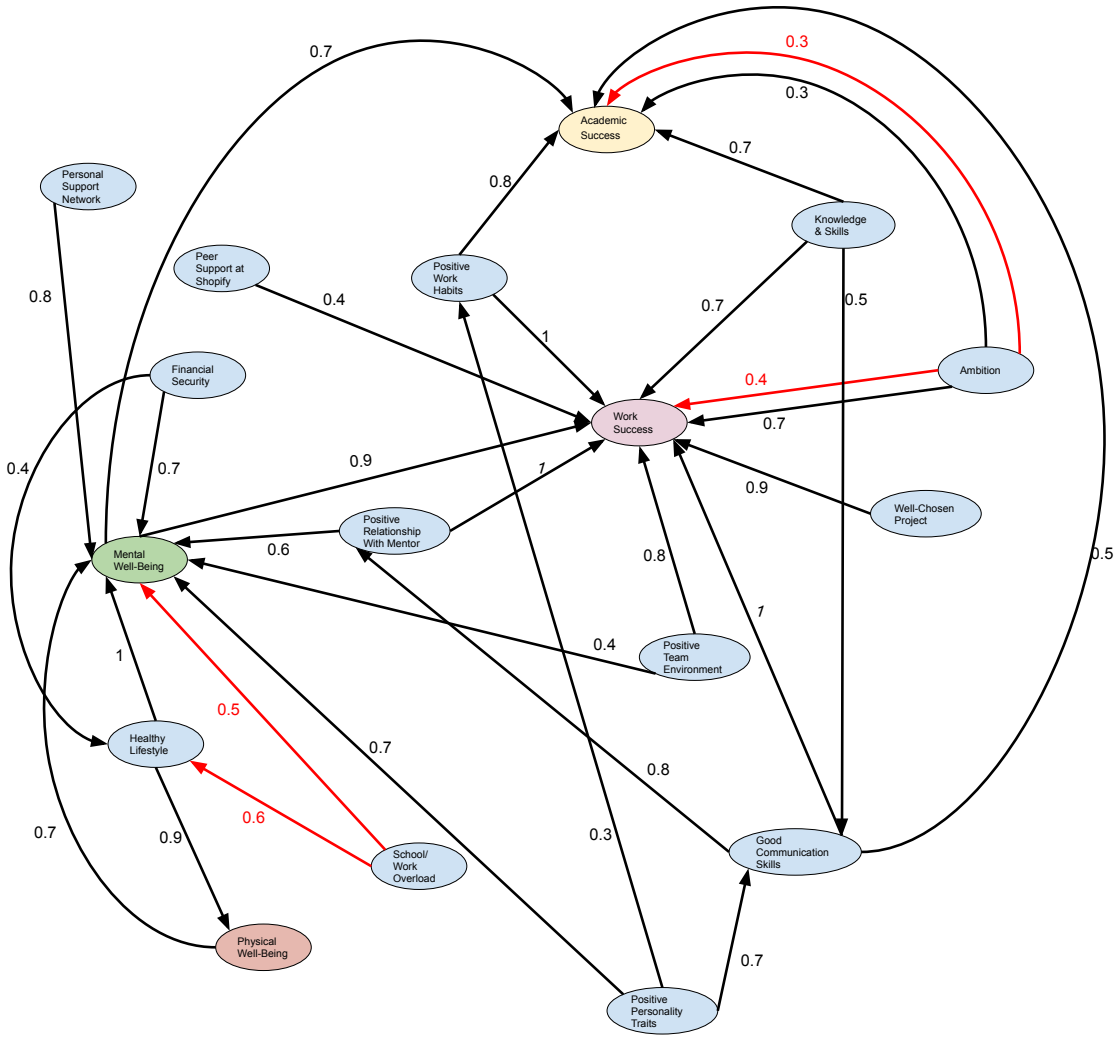

Figure 6.2: The FCM created by the DevDegree team (Team FCM).

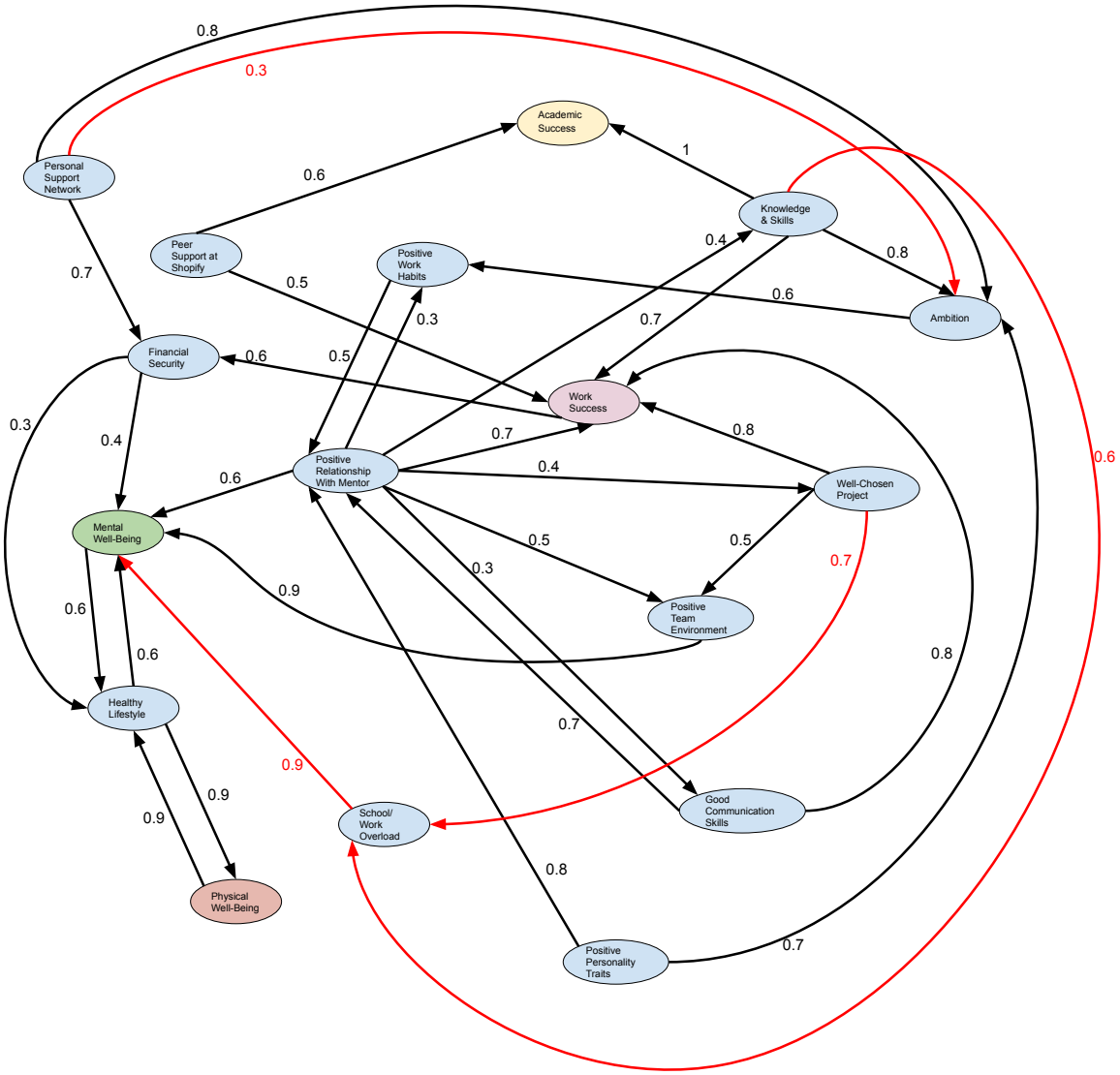

Figure 6.3: The FCM created by the group of mentor participants (Mentor FCM).

Well-Being.

Comparison of the PTCs of the three FCMs provides new insight on the four central outcomes from three different perspectives, student, mentor, and team. In Tables 6.7–6.10, we collected the total impacts on Work Success, Academic Success, Physical Well-Being, and Mental Well-Being, respectively, from each of the three PTCs. For each impact, we calculated the average across the three maps and sorted the table in a non-increasing order of the averages. Our observations are described in Section 6.5.1. A different analysis follows in Section 6.5.2: from each of the three PTCs, we extracted the three factors with the greatest impact on each of the four central outcomes (Tables 6.11–6.14).

	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15	F16	F17
F1-Academic Success	0	0	0	0	0	0.3	0	0	0	0	0	0	0	0	0	0	0
F2-Ambition	0.8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.8
F3-Financial Security	0	0	0	0	0	0	0.7	0	0	0	0	0	0	0	0	0	0
F4-Good Communication Skills	0	0	0	0	0	0	0	0	0	0	0	0.9	0.6	0	0	0	0.7
															-0.3		
F5-Healthy Lifestyle	0	0	0	0	0	0	0.9	0	0	1	0	0	0	0	0	0	0
F6-Knowledge and Skills	0.9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.8
F7-Mental Well-Being	0.6	0	0	0	0.8	0	0	0	0	0	0	0	0	0	0	0	0.6
F8-Peer Support at Shopify	0.9	0	0	0	0	0	0.7	0	0	0	0	0	0	0	0	0	0.2
F9-Personal Support Network	0	0	0	0	0	0	0.8	0	0	0	0	0	0	0	0	0	0
F10-Physical Well-Being	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
F11-Positive Personality Traits	0	0.2	0	0	0	0	0.5	0	0	0	0	0	0	0	0	0	0
F12-Positive Relation with Mentor	0	0	0	0	0	0	0	0	0	0	0	0	0.5	0	0	0.3	0.8
F13-Positive Team Environment	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.5
F14-Positive Work Habits	0.7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.7
F15-School/ Work Overload	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
							-0.8			-0.6							
F16-Well-Chosen Project	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.5
F17-Work Success	0	0	0.2	0	0	0.6	0	0	0	0	0	0	0	0	0	0	0

Table 6.1: Input matrix for Student FCM

	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15	F16	F17
F1-Academic Success	0.272	0	0.048	0	0.027	0.300	0.034	0	0	0.027	0	0	0	0	0	0	0.240
F2-Ambition	0.893	0	0.168	0	0.094	0.606	0.117	0	0	0.094	0	0	0	0	0	0	0.838
F3-Financial Security	0.511	0	0.092	0	0.560	0.333	0.700	0	0	0.560	0	0	0	0	0	0	0.460
F4-Good Communication Skills	0.622	0	0.192	0	0.334	0.603	0.369	0	0	0.334	0	0.900	0.780	0	0	0.270	0.960
															-0.3		
F5-Healthy Lifestyle	0.657	0	0.118	0	1	0.428	0.900	0	0	1	0	0	0	0	0	0	0.592
F6-Knowledge and Skills	0.907	0	0.160	0	0.090	0.621	0.112	0	0	0.090	0	0	0	0	0	0	0.800
F7-Mental Well-Being	0.730	0	0.132	0	0.800	0.475	0.746	0	0	0.800	0	0	0	0	0	0	0.658
F8-Peer Support at Shopify	0.956	0	0.128	0	0.573	0.512	0.716	0	0	0.573	0	0	0	0	0	0	0.639
F9-Personal Support Network	0.584	0	0.105	0	0.640	0.380	0.800	0	0	0.640	0	0	0	0	0	0	0.526
F10-Physical Well-Being	0.657	0	0.118	0	1	0.428	0.900	0	0	1	0	0	0	0	0	0	0.592
F11-Positive Personality Traits	0.477	0.200	0.088	0	0.409	0.318	0.512	0	0	0.409	0	0	0	0	0	0	0.441
F12-Positive Relationship with Mentor	0.505	0	0.175	0	0.098	0.532	0.122	0	0	0.098	0	0	0.500	0	0	0.300	0.873
F13-Positive Team Environment	0.289	0	0.100	0	0.056	0.305	0.070	0	0	0.056	0	0	0	0	0	0	0.500
F14-Positive Work Habits	0.822	0	0.150	0	0.084	0.544	0.105	0	0	0.084	0	0	0	0	0	0	0.750
F15-School/ Work Overload	-0.662	0	-0.119	0	-0.856	-0.431	-0.908	0	0	-0.856	0	0	0	0	0	0	-0.597
F16-Well-chosen Project	0.289	0	0.100	0	0.056	0.305	0.070	0	0	0.056	0	0	0	0	0	0	0.500
F17-Work Success	0.579	0	0.200	0	0.112	0.610	0.140	0	0	0.112	0	0	0	0	0	0	0.527

Table 6.2: PTC of Student FCM

6. A STUDY ON WORK-INTEGRATED LEARNING AT SHOPIFY USING PROBABILISTIC TRANSITIVE CLOSURE OF FUZZY COGNITIVE MAPS56

	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15	F16	F17
F1-Academic Success	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
F2-Ambition	0.3 -0.3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.7 -0.4
F3-Financial Security	0	0	0	0	0.4	0	0.7	0	0	0	0	0	0	0	0	0	0
F4-Good Communication Skills	0.5	0	0	0	0	0	0	0	0	0	0	0.8	0	0	0	0	1
F5-Healthy Lifestyle	0	0	0	0	0	0	1	0	0	0.9	0	0	0	0	0	0	0
F6-Knowledge and Skills	0.7	0	0	0.5	0	0	0	0	0	0	0	0	0	0	0	0	0.7
F7-Mental Well-Being	0.7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.9
F8-Peer Support at Shopify	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.4
F9-Personal Support Network	0	0	0	0	0	0	0.8	0	0	0	0	0	0	0	0	0	0
F10-Physical Well-Being	0	0	0	0	0	0	0.7	0	0	0	0	0	0	0	0	0	0
F11-Positive Personality Traits	0	0	0	0.7	0	0	0.7	0	0	0	0	0	0	0.3	0	0	0
F12-Positive Relationship with Mentor	0	0	0	0	0	0	0.6	0	0	0	0	0	0	0	0	0	1
F13-Positive Team Environment	0	0	0	0	0	0	0.4	0	0	0	0	0	0	0	0	0	0.8
F14-Positive Work Habits	0.8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
F15-School/ Work Overload	0	0	0	0	0 -0.6	0	0 -0.5	0	0	0	0	0	0	0	0	0	0
F16-Well-chosen Project	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.9
F17-Work Success	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 6.3: Input matrix for Team FCM

	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15	F16	F17
F1-Academic Success	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
F2-Ambition	0.3 -0.3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.7 -0.4
F3-Financial Security	0.574	0	0	0	0.4	0	0.82	0	0	0.36	0	0	0	0	0	0	0.738
F4-Good Communication Skills	0.668	0	0	0	0	0	0.48	0	0	0	0	0.8	0	0	0	0	1
F5-Healthy Lifestyle	0.7	0	0	0	0	0	1	0	0	0.9	0	0	0	0	0	0	0.9
F6-Knowledge and Skills	0.800	0	0	0.5	0	0	0.24	0	0	0	0	0.4	0	0	0	0	0.85
F7-Mental Well-Being	0.7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.9
F8-Peer Support at Shopify	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.4
F9-Personal Support Network	0.56	0	0	0	0	0	0.8	0	0	0	0	0	0	0	0	0	0.72
F10-Physical Well-Being	0.49	0	0	0	0	0	0.7	0	0	0	0	0	0	0	0	0	0.63
F11-Positive Personality Traits	0.775	0	0	0.7	0	0	0.801	0	0	0	0	0.56	0	0.3	0	0	0.922
F12-Positive Relationship with Mentor	0.42	0	0	0	0	0	0.6	0	0	0	0	0	0	0	0	0	1
F13-Positive Team Environment	0.28	0	0	0	0	0	0.4	0	0	0	0	0	0	0	0	0	0.872
F14-Positive Work Habits	0.8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
F15-School/ Work Overload	-0.56	0	0	0	-0.6	0	-0.8	0	0	-0.54	0	0	0	0	0	0	-0.72
F16-Well-chosen Project	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.9
F17-Work Success	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 6.4: PTC of Team FCM

	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15	F16	F17
F1-Academic Success	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
F2-Ambition	0	0	0	0	0	0	0	0	0	0	0	0	0	0.6	0	0	0
F3-Financial Security	0	0	0	0	0.3	0	0.4	0	0	0	0	0	0	0	0	0	0
F4-Good Communication Skills	0	0	0	0	0	0	0	0	0	0	0	0.7	0	0	0	0	0.8
F5-Healthy Lifestyle	0	0	0	0	0	0	0.6	0	0	0.9	0	0	0	0	0	0	0
F6-Knowledge and Skills	1	0.8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.7
															-0.6		
F7-Mental Well-Being	0	0	0	0	0.6	0	0	0	0	0	0	0	0	0	0	0	0
F8-Peer Support at Shopify	0.6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.5
F9-Personal Support Network	0	0.8	0.7	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		-0.3															
F10-Physical Well-Being	0	0	0	0	0.9	0	0	0	0	0	0	0	0	0	0	0	0
F11-Positive Personality Traits	0	0.7	0	0	0	0	0	0	0	0	0	0.8	0	0	0	0	0
F12-Positive Relationship with Mentor	0	0	0	0.3	0	0.4	0.6	0	0	0	0	0	0.5	0.3	0	0.4	0.7
F13-Positive Team Environment	0	0	0	0	0	0	0.9	0	0	0	0	0	0	0	0	0	0
F14-Positive Work Habits	0	0	0	0	0	0	0	0	0	0	0	0.5	0	0	0	0	0
F15-School/ Work Overload	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
							-0.9										
F16-Well-chosen Project	0	0	0	0	0	0	0	0	0	0	0	0	0.5	0	0	0	0.8
															-0.7		
F17-Work Success	0	0	0.6	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 6.5: Input matrix for Mentor FCM

	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15	F16	F17
F1-Academic Success	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
F2-Ambition	0.120	0.096	0.160	0.090	0.184	0.120	0.273	0	0	0.166	0	0.300	0.180	0.600	-0.136	0.120	0.267
F3-Financial Security	0	0	0	0	0.468	0	0.508	0	0	0.421	0	0	0	0	0	0	0
F4-Good Communication Skills	0.280	0.224	0.552	0.210	0.503	0.280	0.714	0	0	0.453	0	0.700	0.420	0.304	-0.317	0.280	0.919
F5-Healthy Lifestyle	0	0	0	0	0.878	0	0.600	0	0	0.900	0	0	0	0	0	0	0
F6-Knowledge and Skills	1	0.800	0.457	0.072	0.489	0.096	0.713	0	0	0.440	0	0.240	0.144	0.480	-0.627	0.096	0.761
F7-Mental Well-Being	0	0	0	0	0.600	0	0.360	0	0	0.540	0	0	0	0	0	0	0
F8-Peer Support at Shopify	0.600	0	0.300	0	0.140	0	0.152	0	0	0.126	0	0	0	0	0	0	0.500
F9-Personal Support Network	0.096	0.800	0.738	0.072	0.409	0.096	0.494	0	0	0.368	0	0.240	0.144	0.480	0.041	0.096	0.213
	-0.036	-0.3	-0.048	-0.027	-0.055	-0.036	-0.082			-0.05		-0.09	-0.054	-0.18	-0.109	-0.036	-0.08
F10-Physical Well-Being	0	0	0	0	0.900	0	0.540	0	0	0.810	0	0	0	0	0	0	0
F11-Positive Personality Traits	0.337	0.777	0.449	0.253	0.516	0.337	0.767	0	0	0.465	0	0.842	0.505	0.591	-0.381	0.337	0.748
F12-Positive Relationship with Mentor	0.400	0.320	0.533	0.300	0.613	0.400	0.911	0	0	0.552	0	0.382	0.600	0.434	-0.453	0.400	0.888
F13-Positive Team Environment	0	0	0	0	0.540	0	0.900	0	0	0.486	0	0	0	0	0	0	0
F14-Positive Work Habits	0.200	0.160	0.267	0.150	0.307	0.200	0.455	0	0	0.276	0	0.500	0.300	0.217	-0.226	0.200	0.444
F15-School/ Work Overload	0	0	0	0	-0.54	0	-0.9	0	0	-0.486	0	0	0	0	0	0	0
F16-Well-chosen Project	0	0	0.480	0	0.569	0	0.846	0	0	0.513	0	0	0.500	0	-0.7	0	0.800
F17-Work Success	0	0	0.600	0	0.281	0	0.305	0	0	0.253	0	0	0	0	0	0	0

Table 6.6: PTC of Mentor FCM

6.5.1 All impacts on the four key factors

In Tables 6.7–6.10, Columns 2–4 represent the weights of the total impacts on the specified central outcome in the Student PTC, Mentor PTC, and Team PTC, respectively. The last two columns represent the average and the standard deviation over the three PTCs. These two parameters were calculated to identify strong, weak, and controversial determinants. A small standard deviation for a particular impact means that the corresponding weights in all three PTCs are close to the average. In other words, there is an accord between the three groups' point of view. On the other hand, a large standard deviation for a particular impact means the opposite, so there is a disagreement between the three groups. In the discussion following below, we used the following descriptive scale for standard deviation:

- $[0, 0.1)$ — agreement,
- $[0.1, 0.2)$ — weak disagreement,
- $[0.2, 0.3)$ — moderate disagreement,
- $[0.3, 0.4)$ — strong disagreement, and
- 0.4 and up — very strong disagreement.

Tables 6.7–6.10 were sorted in a non-increasing order of the absolute values of the average weights of impact, and factors that occur twice in the table (because they have both a negative and a positive impact on the specified central outcome) are colour-coded. A discussion of each of Tables 6.7–6.10 follows below.

Impact on Work Success (Table 6.7): The PTCs of the three maps manifest a consensus among the three groups that Good Communication Skills, Positive Relationship with Mentor, and Knowledge and Skills have a strong (positive) impact on Work Success. As can be seen in Table 6.7, these three factors have the three highest average weights of impact, and three of the four lowest standard deviations, which confirms the agreement. Although Well-Chosen Project, Positive Work Habits, and Positive Personality Traits are the next three factors that have a high average weight of impact, their standard deviations indicate a moderate disagreement. On the other extreme of the table, the three factors that have the lowest absolute value of the average are Ambition (negative impact), Academic Success, and Personal Support Network (negative impact). While there is a moderate and weak disagreement about the (negative) impact of Ambition and (positive) impact of Academic Success, respectively, the standard deviation for Personal Support Network indicates a consensus. Put differently, the PTCs of all three groups' maps identified Personal Support Network as the determinant that has practically no negative impact on Work Success.

Impact on Academic Success (Table 6.8): An overall analysis of Table 6.8 shows Knowledge and Skills as the factor with the largest average weight of impact,

	Student Map	Mentor Map	Team Map	Average	Standard Deviation
F4-Good Communication Skills	0.960	0.919	1	0.960	0.040
F12-Positive Relationship with Mentor	0.873	0.888	1	0.920	0.069
F6-Knowledge and Skills	0.800	0.761	0.850	0.804	0.045
F16-Well-Chosen Project	0.500	0.800	0.900	0.733	0.208
F14-Positive Work Habits	0.750	0.444	1	0.732	0.278
F11-Positive Personality Traits	0.441	0.748	0.922	0.704	0.244
F2-Ambition	0.838	0.267	0.700	0.602	0.298
F7-Mental Well-Being	0.658	0	0.900	0.519	0.466
F8-Peer Support at Shopify	0.639	0.500	0.400	0.513	0.120
F5-Healthy Lifestyle	0.592	0	0.900	0.497	0.457
F9-Personal Support Network	0.526	0.213	0.720	0.486	0.256
F13-Positive Team Environment	0.500	0	0.872	0.458	0.439
F15-School/Work Overload	-0.597	0	-0.72	-0.439	0.385
F10-Physical Well-being	0.592	0	0.630	0.407	0.353
F3-Financial Security	0.460	0	0.738	0.399	0.373
F17-Work Success	0.527	0	0	0.176	0.304
F2-Ambition	0	0	-0.4	-0.133	0.231
F1-Academic Success	0.240	0	0	0.080	0.139
F9-Personal Support Network	0	-0.08	0	-0.027	0.046

Table 6.7: Comparison of all impacts on Work Success

as well as small standard deviation. Consequently, there is a weak disagreement on the strong positive impact of this factor on Academic Success. Although the impact of Positive Work Habits has the next highest average, its value indicates only a moderate positive impact, and its standard deviation suggests strong disagreement. The factors with the three smallest absolute values of the average weight of impact, as well as low standard deviation, are Well-Chosen Project, Academic Success, and Personal Support Network (negative impact). There is a weak disagreement that the first two of these three factors have mild impact, and there is an agreement that the Personal Support Network has almost no negative impact on Academic Success. Thus, all three groups considered the aforementioned factors as weak determinants of Academic Success.

Impact on Physical Well-Being (Table 6.9): In the PTCs of all three FCMs, Healthy Lifestyle is listed as having the strongest impact on Physical Well-Being. Its largest average weight of impact and small standard deviation confirms a consensus on its strong positive impact. School/Work Overload is the factor which has the next highest absolute value of the average weight of impact, and moderate standard deviation which indicates a moderate disagreement on its moderate negative impact. In the next line of the table, we observe that there is a strong disagreement about the moderate positive impact of Physical Well-Being on itself. The factors that have the lowest absolute value of the average weight of impact are Personal Support Network (negative impact) and Academic Success. The three groups consent that these factors have practically no impact on Physical Well-Being.

	Student Map	Mentor Map	Team Map	Average	Standard Deviation
F6-Knowledge and Skills	0.907	1	0.800	0.902	0.100
F14-Positive Work Habits	0.822	0.200	0.800	0.607	0.353
F11-Positive Personality Traits	0.477	0.337	0.775	0.530	0.224
F4-Good Communication Skills	0.622	0.280	0.668	0.523	0.212
F8-Peer Support at Shopify	0.956	0.600	0	0.519	0.483
F7-Mental Well-Being	0.730	0	0.700	0.477	0.413
F5-Healthy Lifestyle	0.657	0	0.700	0.452	0.392
F12-Positive Relationship with Mentors	0.505	0.400	0.420	0.442	0.056
F2-Ambition	0.893	0.120	0.300	0.438	0.404
F9-Personal Support Network	0.584	0.096	0.560	0.413	0.275
F15-School/Work Overload	-0.662	0	-0.56	-0.407	0.357
F10-Physical Well-being	0.657	0	0.490	0.382	0.341
F3-Financial Security	0.511	0	0.574	0.362	0.315
F17-Work Success	0.579	0	0	0.193	0.334
F13-Positive Team Environment	0.289	0	0.280	0.190	0.164
F2-Ambition	0	0	-0.3	-0.1	0.173
F16-Well-Chosen Project	0.289	0	0	0.096	0.167
F1-Academic Success	0.272	0	0	0.091	0.157
F9-Personal Support Network	0	-0.036	0	-0.012	0.021

Table 6.8: Comparison of all impacts on Academic Success

	Student Map	Mentor Map	Team Map	Average	Standard Deviation
F5-Healthy Lifestyle	1	0.900	0.900	0.933	0.058
F15-School/Work Overload	-0.856	-0.486	-0.54	-0.627	0.200
F10-Physical Well-being	1	0.810	0	0.603	0.531
F7-Mental Well-Being	0.800	0.540	0	0.447	0.408
F3-Financial Security	0.560	0.421	0.360	0.447	0.102
F9-Personal Support Network	0.640	0.368	0	0.336	0.321
F11-Positive Personality Traits	0.409	0.465	0	0.291	0.254
F4-Good Communication Skills	0.334	0.453	0	0.262	0.235
F8-Peer Support at Shopify	0.573	0.126	0	0.233	0.301
F12-Positive Relationship with Mentor	0.098	0.552	0	0.216	0.294
F16-Well-Chosen Project	0.056	0.513	0	0.190	0.281
F6-Knowledge and Skills	0.090	0.440	0	0.177	0.233
F13-Positive Team Environment	0.056	0.468	0	0.175	0.256
F17-Work Success	0.112	0.253	0	0.122	0.127
F14-Positive Work Habits	0.084	0.276	0	0.120	0.141
F2-Ambition	0.094	0.166	0	0.086	0.083
F9-Personal Support Network	0	-0.05	0	-0.017	0.029
F1-Academic Success	0.027	0	0	0.009	0.016

Table 6.9: Comparison of all impacts on Physical Well-Being

Impact on Mental Well-Being (Table 6.10): There is a consensus among the three groups about the strong negative impact of School/Work Overload on Mental Well-Being. Observe that this factor lies at the extreme top of the table. The next largest average weight of impact corresponds to Healthy Lifestyle, but there is a moderate disagreement on its impact. Personal Support Network (negative impact)

and Academic Success are the two factors which lie at the extreme bottom of the table, and the three groups agree these factors have almost no impact on Mental Well-Being.

	Student Map	Mentor Map	Team Map	Average	Standard Deviation
F15-School/Work Overload	-0.908	-0.9	-0.8	-0.869	0.060
F5-Healthy Lifestyle	0.900	0.600	1	0.833	0.208
F10-Physical Well-being	0.900	0.540	0.700	0.713	0.180
F9-Personal Support Network	0.800	0.494	0.800	0.698	0.177
F11-Positive Personality Traits	0.512	0.767	0.801	0.693	0.158
F3-Financial Security	0.700	0.508	0.820	0.676	0.157
F12-Positive Relationship with Mentor	0.122	0.911	0.600	0.544	0.397
F4-Good Communication Skills	0.369	0.714	0.480	0.521	0.176
F13-Positive Team Environment	0.070	0.900	0.400	0.457	0.418
F7-Mental Well-Being	0.746	0.360	0	0.369	0.373
F6-Knowledge and Skills	0.112	0.713	0.240	0.355	0.316
F16-Well-Chosen Project	0.070	0.846	0	0.305	0.470
F8-Peer Support at Shopify	0.716	0.152	0	0.289	0.377
F14-Positive Work Habits	0.105	0.455	0	0.187	0.238
F17-Work Success	0.140	0.305	0	0.148	0.153
F2-Ambition	0.117	0.273	0	0.130	0.137
F9-Personal Support Network	0	-0.082	0	-0.027	0.047
F1-Academic Success	0.034	0	0	0.011	0.019

Table 6.10: Comparison of all impacts on Mental Well-Being

6.5.2 The three greatest impacts

In Tables 6.11–6.14 below, the three factors that have the greatest impact on the specified central outcome from each group's outlook were sorted in a non-increasing order of the absolute values of the corresponding weights in their PTCs.

Impact on Work Success (Table 6.11): Observe that Good Communication Skills and Positive Relationship with Mentor are repeated three times in the table. This fact demonstrates these two factors have a very strong (positive) impact on Work Success, and confirms our analysis in Section 6.5.1. On the other hand, Ambition, Well-Chosen Project, and Positive Personality Traits each appear only once.

Student Map	Mentor Map	Team Map
F4-Good Communication Skills (0.96)	F4-Good Communication Skills (0.919)	F4-Good Communication Skills (1)
F12-Positive Relationship with Mentor (0.873)	F12-Positive Relationship with Mentor (0.888)	F12-Positive Relationship with Mentor (1)
F2-Ambition (0.838)	F16-Well-Chosen Project (0.8)	F11-Positive Personality Traits (0.922)

Table 6.11: The three strongest impacts on Work Success.

Impact on Academic Success (Table 6.12): Knowledge and Skills occurrs three times in the table which emphasizes the importance of its positive impact on Academic Success. Our analysis in Section 6.5.1 confirms its key impact. While Peer Support at Shopify and Positive Personality Traits are represented twice in the table, Ambition and Positive Work Habits are seen only once. As we observed in Table 6.8, Positive Work Habits has the second strongest average impact, but high standard deviation, so it makes sense that it is seen only once in Table 6.12.

Student Map	Mentor Map	Team Map
F8-Peer Support at Shopify (0.956)	F6-Knowledge and Skills (1)	F6-Knowledge and Skills (0.8)
F6-Knowledge and Skills (0.907)	F8-Peer Support at Shopify (0.6)	F14-Positive Work Habits (0.8)
F2-Ambition (0.893)	F11-Positive Personality Traits (0.775)	F11-Positive Personality Traits (0.922)

Table 6.12: The three strongest impacts on Academic Success.

Impact on Physical Well-Being (Table 6.13): In the PTCs of all three FCMs, Healthy Lifestyle is considered the strongest determinant, so it naturally shows up three times in the table. Also, this factor lies at the top of Table 6.9 in Section 6.5.1, which demonstrates its utmost importance. Physical Well-being and School/Work Overload are listed twice in Table 6.13, while Positive Relationship with Mentor and Financial Security occur only once. Our analysis in the previous section (Table 6.9) also shows that School/Work Overload and Physical Well-being are among the factors with the highest average weight of impact, but with moderate and very strong disagreement, respectively.

Student Map	Mentor Map	Team Map
F5-Healthy Lifestyle (1)	F5-Healthy Lifestyle (0.9)	F5-Healthy Lifestyle (0.9)
F10-Physical Well-being (1)	F10-Physical Well-being (0.81)	F15-School/Work Overload (-0.54)
F15-School/Work Overload (-0.856)	F12-Positive Relationship with Mentor (0.552)	F3-Financial Security (0.36)

Table 6.13: The three strongest impacts on Physical Well-being.

Impact on Mental Well-Being (Table 6.14): School/Work Overload appears three times in the table. This repetition and our analysis in Section 6.5.1 confirm the strong negative impact (with agreement) of School/Work Overload on Mental Well-Being. Healthy Lifestyle is seen twice, while each of Physical Well-being, Positive Relationship with Mentor, Positive Team Environment, Financial Security, Positive Personality Traits, and Personal Support Network appears only once in Table 6.14. We observed in Table 6.10 that Healthy Lifestyle has the second highest

average weight of impact, but its moderate impact in Mentor PTC results in a moderate disagreement. In other words, this lack of consensus lowers the importance of this factor.

Student Map	Mentor Map	Team Map
F15-School/Work Overload (-0.908)	F12-Positive Relationship with Mentor (0.91)	F5-Healthy Lifestyle (1)
F5-Healthy Lifestyle (0.9)	F13-Positive Team Environment (0.9)	F3-Financial Security (0.82)
F10-Physical Well-being (0.9)	F15-School/Work Overload (-0.9)	F11-Positive Personality Traits (0.801) F9-Personal Support Network(0.8) F15-School/Work Overload (-0.8)

Table 6.14: The three strongest impacts on Mental Well-being.

6.6 Conclusion

This work demonstrates how FCMs were used to represent different standpoints on the issue of success and well-being of a Shopify student intern, and how these different opinions were compared via probabilistic transitive closure.

Our analysis in Sections 6.5.1 and 6.5.2 reveals that the most important factors with a positive impact on Work Success are Good Communication Skills and Positive Relationship with Mentor. The three groups agree that Knowledge and Skills is the key determinant with the strongest positive impact on Academic Success. Healthy Lifestyle was identified as having the strongest positive impact on Physical Well-Being, while School/Work Overload stands out as having the strongest (negative) impact on Mental Well-Being.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In this thesis, we considered two algorithmic enhancements for the computation of the probabilistic transitive closure (PTC) of a fuzzy cognitive map (FCM). We also described an application of FCMs in work-integrated learning.

In Chapter 4, we implemented the Reduction-Recovery Algorithm, which was first developed in [16], with the goal of improving the efficiency of other, direct PTC algorithms. We implemented two versions, namely, SRR-PTC and RRR-PTC. Moreover, we compared the running times of these two programs and observed that SRR-PTC and RRR-PTC have practically the same running time, and in most of the tested cases, SRR-PTC was slightly faster. Hence, we used the SRR-PTC code for comparison with ProbTC. Our testing showed that the SRR-PTC program runs faster than ProbTC for reducible digraphs, and has almost the same running time for irreducible digraphs. Therefore, we recommend that the Reduction-Recovery Algorithm be used for finding PTC regardless of the type of the input digraph (once our code has been made available as part of the CIETmap software [6]).

In Chapter 5, we developed a new enhancement for the computation of PTC. We stated a theorem for the reduction of a bipolar weighted digraph using a separating vertex. We proved that if a separation of a bipolar weighted digraph D satisfies certain conditions, which were stated in the theorem, then we can recover the PTC of D from the PTCs of the bipolar weighted digraphs in the separation. Furthermore, using this theorem we developed an algorithm, called Separation-Reduction, and provided a pseudocode version for it. However, this algorithm has not been implemented yet.

In Chapter 6, we used FCMs to represent different perspectives on the issue of success and well-being of a Shopify student intern from three groups' point of view (the student group, team group, and mentor group). Then we compared these different opinions via PTC. Our analysis of the three PTCs revealed that Good Communication Skills and Positive Relationship with Mentor were the most important factors with

a positive impact on Work Success. The key determinant with the strongest positive impact on Academic Success was Knowledge and Skills. Healthy Lifestyle was listed as the strongest positive impact on Physical Well-being, and School/Work Overload had the strongest (negative) impact on Mental Well-Being.

7.2 Future Work

In future research, we envision developing a code for the Separation-Reduction Algorithm, and compare the running time of this new enhancement with the existing codes, namely, ProbTC and SRR-PTC. When coding the Separation-Reduction Algorithm, which is described in Chapter 5, we would need to decide how to compute the PTCs of the smaller components. We hereby propose that the PTCs of the smaller components be computed recursively using a further reduction by separation, if possible. At any step, if no suitable separation exists, then the PTC of the bipolar weighted digraph is computed using ProbTC or SRR-PTC.

Our Separation-Reduction Algorithm does not specify how to find separating vertices and all possible separations, however, algorithms for finding separating vertices are well known. For example, in Chapter 12 of [9], there is an algorithm describing how to find separating vertices and non-separable components of a graph using a depth-first spanning tree of the graph. Recall that a depth-first spanning tree of a graph can be found by using the well-known depth-first-search algorithm. In Chapter 5, we proved that a separating vertex of the underlying undirected graph G of a digraph D is also a separating vertex of D, and vice-versa. Hence, the algorithm for finding separating vertices of graphs can be easily extended to bipolar weighted digraphs. Having an algorithm that gives us the separating vertices and the corresponding non-separable components will allow us to find all possible separations with respect to a separating vertex.

Appendix A

SRR-PTC code

Note that in lines 307 and 308, the following methods from ProbTC are used.

- **genRepOfMinWalks:** It generates a set of representatives of all minimal directed (s, t, σ)-walks, for all (s, t, σ).
- **boolClosure:** It computes the probabilistic transitive closure of a bipolar weighted digraph using the Boolean Algebra method.

```
1
2  import copy
3  import pdb
4  import csv
5  import ProbTC #import ProbTC package
6
7  def arc_index_prob(n,A,B):
8      """From input matrix M, construct the matrices of positive (A)
       and negative (B) weights
9      when the input consists of ch (=1 or 2) matrices. Return the arc
       -> index map
10     and arc -> probability map."""
11
12     arcToIndexMap = {}
13     arcProbabilityList = {}
14     index = 0
15
16     for s in range(0, n):
17         for t in range(0, n):
18             if A[s][t] != 0:
19                 arc = s, t, 1
20                 arcToIndexMap[arc] = index
21                 arcProbabilityList[index] = A[s][t]
22                 index = index + 1
23             if B[s][t] != 0:
24                 arc = s, t, -1
```

```
                arcToIndexMap[arc] = index
                arcProbabilityList[index] = B[s][t]
                index = index + 1

    return arcToIndexMap, arcProbabilityList
#*************************************************************
def writePTransitiveClosureToFile(outfilename, n, A, B):
    """ Write output matrices A and B to a file."""

    fo = open(outfilename, "wb")
    fo.write(bytes("Positive Matrix: \n", 'UTF-8'))
    for i in range(n):
        for j in range(n):
            fo.write(bytes(str(round(A[i][j], 6)), 'UTF-8'))
            if j != n - 1:
                fo.write(bytes(",", 'UTF-8'))
        fo.write(bytes("\n", 'UTF-8'))
    fo.write(bytes("Negative Matrix: \n", 'UTF-8'))
    for i in range(n):
        for j in range(n):
            fo.write(bytes(str(round(B[i][j], 6)), 'UTF-8'))
            if j != n - 1:
                fo.write(bytes(",", 'UTF-8'))
        fo.write(bytes("\n", 'UTF-8'))
    fo.close()
#*************************************************************
def add_vertex(D,u):
    # add vertex u, but no new arcs, to digraph D
    n=len(D)+1
    N = [[0 for j in range(n)] for i in range(n)]

    for i in range(n):
        for j in range(n):
            if i < u and j < u:
                N[i][j] = D[i][j]

            elif i < u and j > u:
                N[i][j] = D[i][j-1]

            elif i > u and j < u:
                N[i][j] = D[i-1][j]

            elif i > u and j > u:
                N[i][j] = D[i-1][j-1]

    return N
#*************************************************************
def NegWalk_Existence(A,B):

```

```
74      # Warshall's Algorithm for the existence of a negative directed
        (s,t,sigma)-walk,
75      # for all (s,t,sigma)
76      # A: matrix of positive weights
77      # B: matrix of negative weights
78      # WP: 0-1 matrix of positive walks (existence only)
79      # WN: 0-1 matrix of negative walks (existence only)
80
81      n=len(A)
82      WP=[[0 for i in range(n)] for i in range(n)]
83      WN=[[0 for i in range(n)] for i in range(n)]
84
85      for s in range(n):
86          for t in range(n):
87              if A[s][t]!=0:
88                  WP[s][t]=1
89
90              if B[s][t]!=0:
91                  WN[s][t]=1
92
93      for k in range(n):
94          for i in range(2):
95               for s in range(n):
96                  for t in range(n):
97                      if (WP[s][k]==1 and WP[k][t]==1) or (WN[s][k]==1
        and WN[k][t]==1):
98                          WP[s][t]=1
99
100                     if (WP[s][k]==1 and WN[k][t]==1) or (WN[s][k]==1
        and WP[k][t]==1):
101                         WN[s][t]=1
102
103     return WN
104 #*************************************************************
105 def Ind(v,D):
106     # compute indegree of vertex v in the digraph with weighted
        adjacency matrix D
107     ind=0
108     for i in range(len(D)):
109         if D[i][v]!=0:
110             ind=ind+1
111     return(ind)
112 #*************************************************************
113 def Outd(v,D):
114     # compute outdegree of vertex v in the digraph with weighted
        adjacency matrix D
115     outd=0
116     for i in range(len(D)):
117         if D[v][i]!=0:
```

```
118             outd=outd+1
119     return(outd)
120 #*****************************************************************
121 def in_neighbour(u,A,B):
122 
123     # Find the last in-neighbour v1 of vertex u and the weight of
    arc (v1,u)
124     # in the digraph with weighted adjacency matrix D
125 
126     v1=None
127     for i in range(len(A)):
128 
129         if A[i][u]!=0:
130             v1=i
131             return v1,A[v1][u]
132             break
133         elif B[i][u]!=0:
134             v1=i
135             return v1,-B[v1][u]
136             break
137 #*****************************************************************
138 def out_neighbour(u,A,B):
139 
140     # Find the last out-neighbour v2 of vertex u and the weight of
    arc (u,v2)
141     # in the digraph with weighted adjacency matrix D
142 
143     v2=None
144 
145     for i in range(len(A)):
146         if A[u][i]!=0:
147             v2=i
148             return v2,A[u][v2]
149             break
150         elif B[u][i]!=0:
151             v2=i
152             return v2,-B[u][v2]
153             break
154 #*****************************************************************
155 def getInputMatrix(ch, M):
156 
157     # Convert the input matrix M into the weighted adjacency
    matrices of positive (A)
158     # and negative weight (B)
159 
160     if ch == 1:
161         n=len(M)
162         A = [[0 for i in range(n)] for i in range(n)]
163         B = [[0 for i in range(n)] for i in range(n)]
```

```
164         for s in range(n):
165             for t in range(n):
166                 if M[s][t] > 0:
167                     A[s][t] = M[s][t]
168 
169                 elif M[s][t] < 0:
170                     B[s][t] = -M[s][t]
171 
172     else:
173         n=len(M)//2
174         A = [[0 for i in range(n)] for i in range(n)]
175         B = [[0 for i in range(n)] for i in range(n)]
176         for s in range(n):
177             for t in range(n):
178                 if M[s][t] != 0:
179                     A[s][t] = M[s][t]
180 
181                 if M[s+n][t] != 0:
182                     B[s][t] = M[s+n][t]
183 
184     return A,B
185 #*****************************************************************
186 def Reduced_Digraph(A,B,u):
187 
188     # delete vertex u from the digraph with weighted adjacency
        matrices A (positive)
189     # and B (negative)
190 
191     A1 = []
192     for i, Ni in enumerate(A):
193         if i != u:
194             A1.append(Ni[:u]+Ni[u+1:])
195 
196     B1 = []
197     for i, Ni in enumerate(B):
198         if i != u:
199             B1.append(Ni[:u]+Ni[u+1:])
200 
201     return A1, B1
202 #*****************************************************************
203 def vertex_type(A,B,u):
204 
205     # Determine the type of vertex u in the digraph with weighted
        adjacency matrices
206     # A (positive) and B (negative)
207     # t=1: vertex of indegree 1 and outdegree 0
208     # t=2: vertex of indegree 0 and outdegree 1
209     # t=3: vertex of indegree 1 and outdegree 1 not in a negative
        cycle
```

```
210     # t=0: none of the above
211
212     Id1=Ind(u,A)
213     Od1=Outd(u,A)
214     Id2=Ind(u,B)
215     Od2=Outd(u,B)
216     t=0
217     Id=Id1+Id2
218     Od=Od1+Od2
219
220     if Id==1:
221         if Od==0:
222             t=1
223
224         elif Od==1:
225             WN=NegWalk_Existence(A,B)
226
227             if WN[u][u]==0:
228                 t=3
229
230     elif Id==0 and Od==1:
231         t=2
232
233     return t
234 #*****************************************************************
235 def Reduction(A,B):
236     # Deleting all vertices of types 1, 2, and 3 (giving priority to
        the first two)
237     # in a digraph with weighted adjacency matrices A (positive) and
        B (negative)
238     # Return the reduced digraph and the stack of deleted vertices
239     # (each vertex with additional info required to recover the
       transitive closure)
240
241     Stack=[]
242     Reduction_completed=False
243
244     while len(A)>2 and Reduction_completed==False :
245
246         save_vertex_type3=None
247         type_of_u=0
248
249         for u in range(len(A)):
250             type_of_u=vertex_type(A,B,u)
251
252             if type_of_u==1:
253                 vertex,prob=in_neighbour(u,A,B)
254                 Stack.append([u,1,[vertex,prob]])
255                 A,B = Reduced_Digraph(A,B,u)
```

```
256                 break
257             elif type_of_u==2:
258                 vertex,prob=out_neighbour(u,A,B)
259                 Stack.append([u,2,[vertex,prob]])
260                 A,B = Reduced_Digraph(A,B,u)
261                 break
262 
263             elif type_of_u==3: #save the vertex to delete only if
    there are no type 1 or 2 vertices
264                 save_vertex_type3=u
265 
266         if (type_of_u in {0,3}) and save_vertex_type3!=None:
267 
268             u=save_vertex_type3
269             type_of_u=3
270 
271             v1,prob1=in_neighbour(u,A,B)
272             v2,prob2=out_neighbour(u,A,B)
273             Stack.append([u,3,[v1,prob1],[v2,prob2]])
274 
275             if prob1>0 and prob2>0:
276 
277                 A[v1][v2]=A[v1][v2]+A[v1][u]*A[u][v2]-A[v1][v2]*A[v1
    ][u]*A[u][v2]
278 
279             elif prob1>0 and prob2<0:
280 
281                 B[v1][v2]=B[v1][v2]+A[v1][u]*B[u][v2]-B[v1][v2]*A[v1
    ][u]*B[u][v2]
282 
283             elif prob1<0 and prob2>0:
284 
285                 B[v1][v2]=B[v1][v2]+B[v1][u]*A[u][v2]-B[v1][v2]*B[v1
    ][u]*A[u][v2]
286 
287             elif prob1<0 and prob2<0:
288 
289                 A[v1][v2]=A[v1][v2]+B[v1][u]*B[u][v2]-A[v1][v2]*B[v1
    ][u]*B[u][v2]
290 
291             A,B = Reduced_Digraph(A,B,u)
292 
293         elif type_of_u==0 and save_vertex_type3==None:
294 
295             Reduction_completed=True
296 
297     return A,B,Stack
298 #*************************************************************
299 def SRR_PTC(A,B,epsilon):
```

```
300
301     A,B,Stack=Reduction(A,B)
302     # A and B are now the weighted adjacency matrices of the reduced
        digraph
303
304     # calling Prob_TC
305     n=len(A)
306     arcToIndexMap, arcProbabilityList = arc_index_prob(n,A,B)
307     c, d = genRepOfMinWalks(n, A, B)
308     A, B=boolClosure(n, c, d, arcToIndexMap, arcProbabilityList,
      epsilon)
309
310     # recovery algorithm
311     while Stack!=[]:
312         stored=Stack.pop()
313         u=stored[0] #stored vertex
314         A=add_vertex(A,u)
315         B=add_vertex(B,u)
316         if stored[1]==1: # type of stored vertex u
317
318             v=stored[2][0] # in-neighbour
319             p=stored[2][1] # weight of arc (v,u)
320             if p>0:
321                 A[v][u]=p
322                 B[v][u]=B[v][v]*p
323                 for s in range(len(A)):
324                     if s!=v and s!=u:
325                         A[s][u]=A[s][v]*p
326                         B[s][u]=B[s][v]*p
327             elif p<0:
328                 p=abs(p)
329                 B[v][u]=p
330                 A[v][u]=B[v][v]*p
331                 for s in range(len(A)):
332                     if s!=v and s!=u:
333                         A[s][u]=B[s][v]*p
334                         B[s][u]=A[s][v]*p
335
336         elif stored[1]==2: # type of stored vertex u
337
338             v=stored[2][0] # out-neighbour
339             p=stored[2][1] # weight of arc (v,u)
340             if p>0:
341                 A[u][v]=p
342                 B[u][v]=p*B[v][v]
343                 for t in range(len(A)):
344                     if t!=v and t!=u:
345                         A[u][t]=p*A[v][t]
346                         B[u][t]=p*B[v][t]
```

```
347             if p<0:
348                 p=abs(p)
349                 B[u][v]=p
350                 A[u][v]=p*B[v][v]
351                 for t in range(len(A)):
352                     if t!=v and t!=u:
353                         A[u][t]=p*B[v][t]
354                         B[u][t]=p*A[v][t]
355 
356         elif stored[1]==3: # type of stored vertex u
357 
358             v1=stored[2][0] # in-neighbour of u
359             p1=stored[2][1] # weight of arc (v1,u)
360             v2=stored[3][0] # out-neighbour of u
361             p2=stored[3][1] # weight of arc (u,v2)
362 
363 
364             if p1>0:
365                 A[v1][u]=p1
366                 B[v1][u]=B[v1][v1]*p1
367                 for s in range(len(A)):
368                     if s!=v1 and s!=u:
369                         A[s][u]=A[s][v1]*p1
370                         B[s][u]=B[s][v1]*p1
371             elif p1<0:
372                 p1=abs(p1)
373                 B[v1][u]=p1
374                 A[v1][u]=B[v1][v1]*p1
375                 for s in range(len(A)):
376                     if s!=v1 and s!=u:
377                         A[s][u]=B[s][v1]*p1
378                         B[s][u]=A[s][v1]*p1
379 
380             if p2>0:
381                 A[u][v2]=p2
382                 B[u][v2]=p2*B[v2][v2]
383                 for t in range(len(A)):
384                     if t!=v2 and t!=u:
385                         A[u][t]=p2*A[v2][t]
386                         B[u][t]=p2*B[v2][t]
387             if p2<0:
388                 p2=abs(p2)
389                 B[u][v2]=p2
390                 A[u][v2]=p2*B[v2][v2]
391                 for t in range(len(A)):
392                     if t!=v2 and t!=u:
393                         A[u][t]=p2*B[v2][t]
394                         B[u][t]=p2*A[v2][t]
395             if v1==v2:
```

```
396                 A[u][u]=abs(p1*p2)
397
398             else:
399                 m1=A[v2][v1]
400                 m2=B[v2][v1]
401
402                 if p1*p2>0:
403                     A[u][u]=abs(p1*p2*m1)
404
405                 elif p1*p2<0:
406                     A[u][u]=abs(p1*p2*m2)
407
408     return A,B
409 #*****************************************************************
410 """ main procedure """
411
412 print('\nCompute the probabilistic transitive closure of a bipolar
    weighted digraph.')
413 print('
    ------------------------------------------------------------------------
    ')
414 print('\nInput Options:\n')
415 print('1. Input comprises of a single matrix.')
416 print('2. Input comprises of two matrices.\n')
417 print('Enter your choice of input:',)
418 ch = int(input())
419 print('\nEnter the number of vertices:',)
420 n = int(input())
421 print('\nEnter the input file path:',)
422 filename = input()
423 #****************************
424 option = 1
425 while option == 1:
426     print('\nEnter the rounding difference for the input matrix. It
    should be a decimal between 0 (inclusive) and 0.5 (exclusive). 0
    results in no rounding.',)
427     delta = float(input())
428
429     """ create input matrices of arc weights """
430
431     input1 = []
432     with open(filename, 'r') as csvfile:
433         inputReader = csv.reader(csvfile, delimiter=',')
434         for row in inputReader:
435             input1.append(row)
436
437     for i in range(ch * n):
438         for j in range(n):
439             input1[i][j] = float(input1[i][j])
```

```
440             if input1[i][j] < delta-1:
441                 input1[i][j] = -1
442             elif input1[i][j] > -delta and input1[i][j] < delta:
443                 input1[i][j] = 0
444             elif input1[i][j] > 1-delta:
445                 input1[i][j] = 1
446
447     A,B=getInputMatrix(ch, input1)
448
449     again = 1
450     while again == 1:
451
452         print('\nEnter the output file path for transitive closure:'
    ,)
453         outfilename1 = input()
454         print('\nAlgorithm Options:\n')
455         print('1. Exact Boolean Algebra Approach.')
456         print('2. Approximative Boolean Algebra Approach (
    recommended for all but the smallest input matrices).\n')
457         print('Enter your choice of algorithm:',)
458         ch2 = int(input())
459
460         if ch2 == 2:
461             print('\nEnter required precision:',)
462             epsilon = float(input())
463         else:
464             epsilon = 0
465
466         """ compute transitive closure """
467
468         e,f=SRR_PTC(A,B,epsilon)
469         writePTransitiveClosureToFile(outfilename1,n, e, f)
470
471         print('\nEnter 1 if you want to compute transitive closure
    again for the same rounding difference.',)
472         again = int(input())
473
474     print('\nEnter 1 if you want to compute transitive closure for a
     different rounding difference.',)
475     option = int(input())
```

Appendix B

RRR-PTC code

Note that in lines 485 and 486, the following methods from ProbTC are used.

- **genRepOfMinWalks:** It generates a set of representatives of all minimal directed (s, t, σ)-walks, for all (s, t, σ).
- **boolClosure:** It computes the probabilistic transitive closure of a bipolar weighted digraph using the Boolean Algebra method.

```
1
2  import copy
3  import pdb
4  import csv
5  import ProbTC #import ProbTC package
6
7  def arc_index_prob(n,A,B):
8
9      """From input matrix M, construct the matrices of positive (A)
       and negative (B) weights
10     when the input consists of ch (=1 or 2) matrices. Return the arc
       -> index map
11     and arc -> probability map."""
12
13     arcToIndexMap = {}
14     arcProbabilityList = {}
15     index = 0
16
17     for s in range(0, n):
18         for t in range(0, n):
19             if A[s][t] != 0:
20
21                 arc = s, t, 1
22                 arcToIndexMap[arc] = index
23                 arcProbabilityList[index] = A[s][t]
24                 index = index + 1
```

```
25
26              if B[s][t] != 0:
27
28                  arc = s, t, -1
29                  arcToIndexMap[arc] = index
30                  arcProbabilityList[index] = B[s][t]
31                  index = index + 1
32
33      return arcToIndexMap, arcProbabilityList
34 #*************************************************************
35 def writePTransitiveClosureToFile(outfilename, n, A, B):
36
37      """ Write output matrices A and B to a file."""
38
39      fo = open(outfilename, "wb")
40      fo.write(bytes("Positive Matrix: \n", 'UTF-8'))
41
42      for i in range(n):
43          for j in range(n):
44              fo.write(bytes(str(round(A[i][j], 6)), 'UTF-8'))
45
46              if j != n - 1:
47
48                  fo.write(bytes(",", 'UTF-8'))
49          fo.write(bytes("\n", 'UTF-8'))
50
51      fo.write(bytes("Negative Matrix: \n", 'UTF-8'))
52
53      for i in range(n):
54          for j in range(n):
55
56              fo.write(bytes(str(round(B[i][j], 6)), 'UTF-8'))
57
58              if j != n - 1:
59
60                  fo.write(bytes(",", 'UTF-8'))
61          fo.write(bytes("\n", 'UTF-8'))
62      fo.close()
63 #*************************************************************
64 def add_vertex(D,u):
65
66      # add vertex u, but no new arcs, to digraph D
67      n=len(D)+1
68      N = [[0 for j in range(n)] for i in range(n)]
69
70      for i in range(n):
71          for j in range(n):
72              if i < u and j < u:
73                  N[i][j] = D[i][j]
```

```
74
75              elif i < u and j > u:
76                  N[i][j] = D[i][j-1]
77
78              elif i > u and j < u:
79                  N[i][j] = D[i-1][j]
80
81              elif i > u and j > u:
82                  N[i][j] = D[i-1][j-1]
83
84      return N
85 #*****************************************************************
86 def NegWalk_Existence(A,B):
87
88      # Warshall's Algorithm for the existence of a negative directed
        (s,t,sigma)-walk,
89      # for all (s,t,sigma)
90      # A: matrix of positive weights
91      # B: matrix of negative weights
92      # WP: 0-1 matrix of positive walks (existence only)
93      # WN: 0-1 matrix of negative walks (existence only)
94
95      n=len(A)
96      WP=[[0 for i in range(n)] for i in range(n)]
97      WN=[[0 for i in range(n)] for i in range(n)]
98
99      for s in range(n):
100         for t in range(n):
101             if A[s][t]!=0:
102                 WP[s][t]=1
103
104             if B[s][t]!=0:
105                 WN[s][t]=1
106
107     for k in range(n):
108         for i in range(2):
109              for s in range(n):
110                 for t in range(n):
111
112                     if (WP[s][k]==1 and WP[k][t]==1) or (WN[s][k]==1
        and WN[k][t]==1):
113                         WP[s][t]=1
114
115                     if (WP[s][k]==1 and WN[k][t]==1) or (WN[s][k]==1
        and WP[k][t]==1):
116                         WN[s][t]=1
117
118     return WN
119 #*****************************************************************
```

```
120 def Ind(v,D):
121     # compute indegree of vertex v in the digraph with weighted
    adjacency matrix D
122     ind=0
123     for i in range(len(D)):
124         if D[i][v]!=0:
125             ind=ind+1
126 
127     return(ind)
128 #*************************************************************
129 def Outd(v,D):
130     # compute outdegree of vertex v in the digraph with weighted
    adjacency matrix D
131     outd=0
132 
133     for i in range(len(D)):
134         if D[v][i]!=0:
135             outd=outd+1
136 
137     return(outd)
138 #*************************************************************
139 def out_neighbour(u,A,B):
140 
141     # Find the first out-neighbour v2 of vertex u and the weight of
    arc (u,v2)
142     # in the digraph with weighted adjacency matrices A (positive)
    and B (negative)
143     # Used when u has outdegree 1
144     v2=None
145 
146     for i in range(len(A)):
147         if A[u][i]!=0:
148             v2=i
149 
150             return v2,A[u][v2]
151             break
152 
153         elif B[u][i]!=0:
154             v2=i
155 
156             return v2,-B[u][v2]
157             break
158 #*************************************************************
159 def in_neighbour(u,A,B):
160 
161     # Find the first in-neighbour v1 of vertex u and the weight of
    arc (v1,u)
162     # in the digraph with weighted adjacency matrices A (positive)
    and B (negative)
```

```
163     # Used when u has indegree 1
164 
165     v1=None
166 
167     for i in range(len(A)):
168         if A[i][u]!=0:
169             v1=i
170 
171             return v1,A[v1][u]
172             break
173 
174         elif B[i][u]!=0:
175             v1=i
176 
177             return v1,-B[v1][u]
178             break
179 #****************************************************************
180 def Reduced_Digraph(A,B,u):
181 
182     # delete vertex u from the digraph with weighted adjacency
        matrices A (positive)
183     # and B (negative)
184     A1 = []
185 
186     for i, Ni in enumerate(A):
187         if i != u:
188             A1.append(Ni[:u]+Ni[u+1:])
189 
190     B1 = []
191 
192     for i, Ni in enumerate(B):
193         if i != u:
194             B1.append(Ni[:u]+Ni[u+1:])
195 
196     return A1, B1
197 #****************************************************************
198 def getInputMatrices(ch, M):
199 
200     # Convert the input matrix M into the weighted adjacency
        matrices of positive (A)
201     # and negative weight (B)
202 
203     if ch == 1:
204         n=len(M)
205         A = [[0 for i in range(n)] for i in range(n)]
206         B = [[0 for i in range(n)] for i in range(n)]
207 
208         for s in range(n):
209             for t in range(n):
```

```
210                 if M[s][t] > 0:
211                     A[s][t] = M[s][t]
212 
213                 elif M[s][t] < 0:
214                     B[s][t] = -M[s][t]
215 
216     else:
217 
218         n=len(M)//2
219         A = [[0 for i in range(n)] for i in range(n)]
220         B = [[0 for i in range(n)] for i in range(n)]
221 
222         for s in range(n):
223             for t in range(n):
224                 if M[s][t] != 0:
225 
226                     A[s][t] = M[s][t]
227 
228                 if M[s+n][t] != 0:
229 
230                     B[s][t] = M[s+n][t]
231 
232     return A,B
233 #*****************************************************************
234 def vertex_type(A,B,u):
235 
236     # Determine the type of vertex u in the digraph with weighted
    adjacency matrices
237     # A (positive) and B (negative)
238     # t=1: vertex of indegree 1 and outdegree 0
239     # t=2: vertex of indegree 0 and outdegree 1
240     # t=3: vertex of indegree 1 and outdegree 1 not in a negative
    cycle
241     # t=0: none of the above
242 
243     Id1=Ind(u,A)
244     Od1=Outd(u,A)
245     Id2=Ind(u,B)
246     Od2=Outd(u,B)
247     t=0
248     Id=Id1+Id2
249     Od=Od1+Od2
250 
251     if Id==1:
252         if Od==0:
253             t=1
254 
255         elif Od==1:
256             WN=NegWalk_Existence(A,B)
```

```
257
258             if WN[u][u]==0:
259                 t=3
260
261     elif Id==0 and Od==1:
262         t=2
263
264     return t
265 #*****************************************************************
266 def Modify_type3(u,A,B):
267
268     # modify the arc between v1 (in-neighbour of u) & v2 (out-
    neighbour of u)
269     # where u is a type 3 vertex
270
271     v1,p=in_neighbour(u,A,B)
272
273     v2,q=out_neighbour(u,A,B)
274
275     if p>0 and q>0:
276         A[v1][v2]=A[v1][v2]+A[v1][u]*A[u][v2]-A[v1][v2]*A[v1][u]*A[u
    ][v2]
277
278     elif p<0 and q>0:
279         B[v1][v2]=B[v1][v2]+B[v1][u]*A[u][v2]-B[v1][v2]*B[v1][u]*A[u
    ][v2]
280
281     elif p>0 and q<0:
282         B[v1][v2]=B[v1][v2]+A[v1][u]*B[u][v2]-B[v1][v2]*A[v1][u]*B[u
    ][v2]
283
284     elif p<0 and q<0:
285         A[v1][v2]=A[v1][v2]+B[v1][u]*B[u][v2]-A[v1][v2]*B[v1][u]*B[u
    ][v2]
286
287     return A,B
288 #*****************************************************************
289 def Recovery(A1,B1,u,stored):
290
291     A=add_vertex(A1,u)
292     B=add_vertex(B1,u)
293
294     if stored[0]==1: # vertex u is of type 1
295
296         v=stored[1][0]
297         p=stored[1][1]
298
299         if p>0:
300
```

```
301             A[v][u]=p
302             B[v][u]=B[v][v]*p
303 
304             for s in range(len(A)):
305 
306                 if s!=v and s!=u:
307 
308                     A[s][u]=A[s][v]*p
309                     B[s][u]=B[s][v]*p
310 
311         elif p<0:
312 
313             p=abs(p)
314             B[v][u]=p
315             A[v][u]=B[v][v]*p
316 
317             for s in range(len(A)):
318                 if s!=v and s!=u:
319 
320                     A[s][u]=B[s][v]*p
321                     B[s][u]=A[s][v]*p
322 
323     elif stored[0]==2: # vertex u is of type 2
324 
325         v=stored[1][0]
326         p=stored[1][1]
327 
328         if p>0:
329 
330             A[u][v]=p
331             B[u][v]=p*B[v][v]
332 
333             for t in range(len(A)):
334                 if t!=v and t!=u:
335 
336                     A[u][t]=p*A[v][t]
337                     B[u][t]=p*B[v][t]
338 
339         if p<0:
340 
341             p=abs(p)
342             B[u][v]=p
343             A[u][v]=p*B[v][v]
344 
345             for t in range(len(A)):
346                 if t!=v and t!=u:
347 
348                     A[u][t]=p*B[v][t]
349                     B[u][t]=p*A[v][t]
```

```
350
351     elif stored[0]==3: # vertex u is of type 3
352
353         v1=stored[1][0]
354         p1=stored[1][1]
355         v2=stored[2][0]
356         p2=stored[2][1]
357
358         if p1>0:
359
360             A[v1][u]=p1
361             B[v1][u]=B[v1][v1]*p1
362
363             for s in range(len(A)):
364                 if s!=v1 and s!=u:
365
366                     A[s][u]=A[s][v1]*p1
367                     B[s][u]=B[s][v1]*p1
368
369         elif p1<0:
370
371             p1=abs(p1)
372             B[v1][u]=p1
373             A[v1][u]=B[v1][v1]*p1
374
375             for s in range(len(A)):
376                 if s!=v1 and s!=u:
377
378                     A[s][u]=B[s][v1]*p1
379                     B[s][u]=A[s][v1]*p1
380
381         if p2>0:
382
383             A[u][v2]=p2
384             B[u][v2]=p2*B[v2][v2]
385
386             for t in range(len(A)):
387                 if t!=v2 and t!=u:
388
389                     A[u][t]=p2*A[v2][t]
390                     B[u][t]=p2*B[v2][t]
391
392         if p2<0:
393
394             p2=abs(p2)
395             B[u][v2]=p2
396             A[u][v2]=p2*B[v2][v2]
397
398             for t in range(len(A)):
```

```
399                 if t!=v2 and t!=u:
400
401                     A[u][t]=p2*B[v2][t]
402                     B[u][t]=p2*A[v2][t]
403         if v1==v2:
404
405             A[u][u]=abs(p1*p2)
406
407         else:
408
409             m1=A[v2][v1]
410             m2=B[v2][v1]
411
412             if p1*p2>0:
413
414                 A[u][u]=abs(p1*p2*m1)
415
416             elif p1*p2<0:
417
418                 A[u][u]=abs(p1*p2*m2)
419
420
421     return A,B
422 #*****************************************************************
423 def RRR_PTC(A,B,epsilon):
424
425     stored=[] # save type of the deleted vertex, its in-neighbour or
        out-neighbour,
426     # and the weight of the relevant arc(s)
427
428     flag=False
429     u=0
430     save_vertex_type3 = None
431
432     while flag==False and u<len(A):
433         t=vertex_type(A,B,u)
434         if t==1:
435
436             flag=True # delete u
437             v1,p=in_neighbour(u,A,B)
438             stored=[1,[v1,p]]  # save type of u, its in-neighbour,
    and the weight of arc (v1,u)
439             A,B=Reduced_Digraph(A,B,u)
440
441         elif t==2:
442
443             flag=True # delete u
444             v2,q=out_neighbour(u,A,B)
445             stored=[2,[v2,q]] # save type of u, its out-neighbour,
```

```
    and the weight of arc (u,v2)
446             A,B=Reduced_Digraph(A,B,u)
447
448         elif t==3:
449             save_vertex_type3 = u # temporarily save u
450
451         if flag: # vertex of type 1 or 2 has been deleted
452
453             A1,B1=RRR_PTC(A,B,epsilon) # Recursively compute TC,
    using further reduction if possible
454             A1,B1=Recovery(A1,B1,u,stored) # Recover TC of the input
     digraph
455
456             return A1,B1
457         else:
458
459             u=u+1
460
461     if u>=len(A):
462         if save_vertex_type3 is not None: # Delete the saved vertex
    of type3
463
464             u=save_vertex_type3
465             v1,p=in_neighbour(u,A,B)
466             v2,q=out_neighbour(u,A,B)
467             stored=[3,[v1,p],[v2,q]]
468
469             # save type of u, its in-neighbour v1 and the weight of
    arc (v1,u), and
470             # its out-neighbour v2 and the weight of arc (u,v2)
471
472             A,B=Modify_type3(u,A,B) # modify the arc between v1 & v2
473             A,B=Reduced_Digraph(A,B,u) # Delete vertex u
474             A1,B1=RRR_PTC(A,B,epsilon) # Recursively compute TC,
    using further reduction if possible
475             A1,B1=Recovery(A1,B1,u,stored) # Recover TC of the input
     digraph
476
477             return A1,B1
478
479         else:
480
481             # calling Prob_TC
482
483             n=len(A)
484             arcToIndexMap, arcProbabilityList = arc_index_prob(n,A,B
    )
485             c, d = genRepOfMinWalks(n, A, B)
486             A1,B1=boolClosure(n, c, d, arcToIndexMap,
```

```
    arcProbabilityList, epsilon)
487             return(A1,B1)
488 #*****************************************************************
489 """ main procedure """
490
491 print('\nCompute the probabilistic transitive closure of a bipolar
    weighted digraph.')
492 print('
    ------------------------------------------------------------------------
    ')
493 print('\nInput Options:\n')
494 print('1. Input comprises of a single matrix.')
495 print('2. Input comprises of two matrices.\n')
496 print('Enter your choice of input:',)
497 ch = int(input())
498 print('\nEnter the number of vertices:',)
499 n = int(input())
500 print('\nEnter the input file path:',)
501 filename = input()
502 #*****************************
503 option = 1
504 while option == 1:
505     print('\nEnter the rounding difference for the input matrix. It
    should be a decimal between 0 (inclusive) and 0.5 (exclusive). 0
    results in no rounding.',)
506     delta = float(input())
507
508     """ create input matrices of arc weights """
509
510     input1 = []
511     with open(filename, 'r') as csvfile:
512         inputReader = csv.reader(csvfile, delimiter=',')
513         for row in inputReader:
514             input1.append(row)
515
516     for i in range(ch * n):
517         for j in range(n):
518             input1[i][j] = float(input1[i][j])
519             if input1[i][j] < delta-1:
520                 input1[i][j] = -1
521             elif input1[i][j] > -delta and input1[i][j] < delta:
522                 input1[i][j] = 0
523             elif input1[i][j] > 1-delta:
524                 input1[i][j] = 1
525
526     A,B=getInputMatrices(ch, input1)
527
528     again = 1
529     while again == 1:
```

```
530
531         print('\nEnter the output file path for transitive closure:'
    ,)
532         outfilename1 = input()
533         print('\nAlgorithm Options:\n')
534         print('1. Exact Boolean Algebra Approach.')
535         print('2. Approximative Boolean Algebra Approach (
    recommended for all but the smallest input matrices).\n')
536         print('Enter your choice of algorithm:',)
537         ch2 = int(input())
538
539         if ch2 == 2:
540             print('\nEnter required precision:',)
541             epsilon = float(input())
542         else:
543             epsilon = 0
544
545         """ compute transitive closure """
546
547         e,f=RRR_PTC(A,B,epsilon)
548         writePTransitiveClosureToFile(outfilename1,n, e, f)
549
550         print('\nEnter 1 if you want to compute transitive closure
    again for the same rounding difference.',)
551         again = int(input())
552
553      print('\nEnter 1 if you want to compute transitive closure for a
       different rounding difference.',)
554      option = int(input())
```

Bibliography

[1] J. Aguilar, A survey about fuzzy cognitive maps papers, *International Journal of Computational Cognition* **3** (2005), 27–33.

[2] A. Amirkhani et al., A review of fuzzy cognitive maps in medicine: taxonomy, methods, and applications, *Computer Methods and Programs in Biomedicine* **142** (2017), 129–145.

[3] M. Bevilacqua et al., Application of fuzzy cognitive maps to drug administration risk management, *IFAC Proceedings* **46** (2013), 438–443.

[4] J. A. Bondy and U. S. R. Murty, *Graph Theory*, Springer, 2008.

[5] C. J. Colbourn, *The Combinatorics of Network Reliability*, Oxford University Press, 1987.

[6] https://ciet.org/home/technology/cietmap/.

[7] G. Espinosa-Paredes, A. Aũnez-Carrera, A. Vazquez-Rodriguez, E.-G. Espinosa-Martinez, Modeling of the high pressure core spray systems with fuzzy cognitive maps for operational transient analysis in nuclear power reactors, *Progress in Nuclear Energy* **51** (2009), 434–442.

[8] B. Giles, G. Haas, M. Šajna, C. S. Findlay, Exploring Aboriginal views of health using fuzzy cognitive maps and transitive closure: A case study of the determinants of diabetes, *Canadian J. Public Health* **99** (2008), 411–417.

[9] R. P. Grimaldi, *Discrete and Combinatorial Mathematics: an Applied Introduction*, Pearson/Addison-Wesley, 2004.

[10] S. Hossain and L. Brooks, Fuzzy cognitive map modeling educational software adoption, *Computers and Education* **51** (2008), 1569–1588.

[11] I. I. Kang and S. Lee, Using fuzzy cognitive map for the relationship management in airline service, *Expert Systems with Applications* **26** (2004), 545–555.

[12] U. Özesmi and S. L. Özesmi, Ecological models based on people's knowledge: a multi-step fuzzy cognitive mapping approach, *Ecological Modelling* **176** (2004), 43–64.

[13] J. Morzaria, M. Šajna, ProbTC (a software package for computing the probabilistic transitive closure of a fuzzy cognitive map; coded in Python), 2015.

[14] J. Morzaria, M. Šajna, FuzzyTC (a software package for computing the fuzzy transitive closure of a fuzzy cognitive map; coded in Python), 2014.

[15] P. Niesink, K. Poulin, M. Šajna, ProbClosure (a software package for computing the probabilistic closure of a fuzzy cognitive map; coded in C), 2007.

[16] P. Niesink, K. Poulin, M. Šajna, Computing transitive closure of bipolar weighted digraphs, *Discrete Appl. Math.* **161** (2013), 217–243.

[17] E. I. Papageorgiou, C. Stylios, P. Groumpos, An integrated two-level hierarchical decision making system based on fuzzy cognitive maps (FCMs), *IEEE Transactions on Biomedical Engineering* **50** (2003), 1326–1339.

[18] E. I. Papageorgiou, A. Markinos, T. Gemtos, Application of fuzzy cognitive maps for cotton yield management in precision farming, *Expert Systems with Applications* **36** (2010), 12399–12413.

[19] E. I. Papageorgiou, P. Spyridonos, D. Glotsos, C.D. Stylios, P.P. Groumpos, G. Nikiforidis, Brain tumor characterization using the soft computing technique of fuzzy cognitive maps, *Applied Soft Computing* **8** (2008), 820–828.

[20] C. Pilgrim, Work-integrated learning in ICT degrees, *Proceedings of the Thirteenth Australasian Computing Education Conference* **114**, 2011.

[21] C. Pilgrim and T. Koppi, Work integrated learning rationale and practices in Australian information and communications technology degrees, *Proceedings of the Fourteenth Australasian Computing Education Conference* **123**, 2012.

[22] C. Smith, S. Ferns, and L. Russel, The impact of work integrated learning on student work-readiness, *Final Report, Curtin University of Technology*, LSN Teaching Development Unit, 2014.

[23] W. Stach and L. Kurgan, Modeling software development project using fuzzy cognitive maps, in: *Proceedings of the 4th ASERC Workshop on Quantitative and Soft Software Engineering* (2004), 55–60.

[24] M. A. Styblinski and B. D. Meyer, Signal flow graphs vs. fuzzy cognitive maps in application to qualitative circuit analysis, *International Journal of Man–Machine Studies* **35** (1991), 175–186.

[25] C. D. Stylios and P. P. Groumpos, Modeling complex systems using fuzzy cognitive maps, *IEEE Transactions on Systems Man and Cybernetics, Part A-Systems and Humans* **34** (2004), 155–162.

[26] R. Taber, Knowledge processing with fuzzy cognitive maps, *Expert Systems with Applications* **2** (1991), 83–87.

[27] D. Yaman and S. Polat, A fuzzy cognitive map approach for effect-based operations: an illustrative case, *Information Sciences* **179** (2009), 382–403.