Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Computer Science and Information Theory

Simulation of quantum walks on a classical computer

Scientific Students' Association Report

Author:

Viktória Nemkin

Advisor:

dr. Katalin Friedl

2021

Contents

Notes

Kivonat

Az utóbbi években egyre nagyobb figyelem összpontosul a kvantuminformatikára. Olyan globális vállalatok, mint az IBM, a Google, a Microsoft és az Amazon jelentős összegeket fektetnek kutatásba, hardver- és szoftverfejlesztésekbe ezen a területen, míg az Európai Unió és Magyarország számos olyan programot indított, melyek a kvantuminformatikai kutatások fellendítését célozzák meg.

A jelenlegi kvantumszámítógépekben elérhető qubitek (kvantumbitek) mennyisége még csekély, de sokan úgy vélik hogy a jövőben ez a szám növekedni fog. Az első olyan, a gyakorlatban is hasznos kvantumalgoritmusok, amiket ezeken a processzorokon futtatni tudunk majd, várhatóan azok lesznek, melyek takarékosan bánnak a rendelkezésre álló qubitekkel. A kvantumséta, mely a klasszikus véletlen bolyongás általánosítása kvantumos esetben, pontosan ilyen algoritmus. Mivel a qubit igénye a gráf csúcsszámában logaritmikus, így ez egy érdekes módszernek ígérkezik akár a közeljövőre nézve is. A kvantumséták erejét bizonyítja, hogy a Grover keresés (mely több kvantumalgoritmus alapját képezi) is értelmezhető ezek egy speciális fajtájának.

Dolgozatomban leírom a kvantumséták matematikai alapjait, részletezve a megvalósítás szempontjából fontos pontokat, melyek a szakirodalomban kisebb hangsúllyal szerepelnek. Ismertetem az általam írt szimulátor program architekturális felépítését és működését, továbbá a futtatott szimulációim eredményeit.

A szimulátor programot Python 3 nyelven írtam, a Stratégia tervezési minta alapján. A szakirodalomban tipikusan használt gráfokat beépítetten támogatja, melyek kombinálásával tetszőlegesen bonyolult reguláris gráf előállítható, ez az előállítás képezi a kvantumséta alapját is. A szoftver reguláris gráfokon történő kvantum és klasszikus séták szimulációját teszi lehetővé, az eredményekről pedig egy részletes report fájlt generál. A kvantumos séták esetében a séta tulajdonságai a valószínűségek generálásához felhasznált érmétől is függnek, melyet többféleképpen is lehet definiálni. A program beépítetten tartalmazza az Hadamard-, a Grover- és a Fourier-érméket, de felépítéséből adódóan könnyen bővíthető tetszőleges érmével is.

Szimulációim segítségével összehasonlítottam a klasszikus és a kvantum séták viselkedését, továbbá kimutattam az elméleti szakirodalom alapján elvárt kvantumos jellegzetességeket, az Hadamard-séta ballisztikus természetét és a kvantumséták ciklikus tulajdonságát.

Abstract

In recent years, there has been an increasing focus on quantum informatics. Influential global companies such as IBM, Google, Microsoft, and Amazon have invested significant amounts into studying and developing hardware and software for this sector, while the European Union and Hungary have launched several programs to accelerate quantum research.

Current technology is yet to produce a significant number of qubits (quantum bits) in a quantum processor, but many believe the amount will increase over the years. The first practical quantum algorithms to be run on these processors are likely to be the ones that use qubits sparingly. Quantum walking, the generalized version of classical random walking, is exactly this kind of algorithm. The number of qubits required to run a quantum walk on a graph is logarithmic in the number of vertices, making it a promising technique for the near future. Furthermore, Grover's search algorithm (a basis for many quantum algorithms) can be viewed as a special case of quantum walks, which illustrates the potential power of this method.

In my dissertation, I present the mathematical framework for quantum walks, detailing the points critical for implementation, which are given less emphasis in the literature. I describe the architecture and capabilities of the simulator program I have written and the conclusions of the simulations I have run.

I developed the software using Python 3, based on the Strategy design pattern. It supports graphs commonly found in the literature while also providing a method for combining them, facilitating experimentation on several kinds of regular graphs. This composition is also the foundation of the quantum walk. It can simulate classical and quantum walks on the same graphs and produce a report file detailing the results. In the quantum case, the characteristics of the walk are also dependent on the type of coin used to generate the probabilities, which can be defined in several ways. The program includes the Hadamard, Grover, and Fourier coins and can easily be extended with others.

Running several simulations, I compared the behavior of classical and quantum walks and demonstrated the quantum characteristics expected from the theoretical literature, the ballistic nature of the Hadamard walk, and the cyclic property of quantum walks.

Chapter 1

Introduction

> The following chapter contains the analysis of the TDK dissertation topic, historic background, motivation for the task: why it is important, relevant, useful, etc, the solutions currently available and then compare and contrast with the student's solution. Then, it concludes with the structure of the dissertation, describing the remaining chapters.

The following sections are based on [6].

1.1 Classical random walks on graphs

Classical random walks are used to describe stochastic processes and many real life sciences rely on these methods. For example stock price movement prediction in finance, natural language processing algorithms, describing brownian motion in engineering physics, and various applications in biology and bioinformatics, such as DNA evolution models, population dynamics, modeling disease outbreaks, epidemic modeling ...etc. There are also algorithms, where stochastic processes are not directly present, but introduced as a way to deal with large quantities of data. Most notably, Google's Page Rank algorithm utilizes classical random walks on the large sized graph of the internet, to score documents on how good of an information source they are or how well they link to other information sources. Other algorithms include Markov chain Monte Carlo, for sampling from a probability distribution, difficult to directly model. Instead, they construct a stohastic process which's equilibrium distribution is the desired one and sample this by repeatedly recording states from the chain. The Metropolis Hashtings algorithm is similar to this, and is used to approximate the distribution or compute an integral (e.g expected value).

> Itt az lenne a célom hogy eladjam a véletlen sétákat, hogy ezek nagyon fontos algoritmusok. A Google PageRank egy nagyon jó példa.

Random walks are effective in link prediction and recommendation system, computer vision, semi-supervised learning, network embedding, and complex social network analysis. There are also some literature illustrating the applications of random walks on graphs, text analysis, science of science, and knowledge discovery. Random walks can speed up deterministic algorithms by introducing randomization, for example finding specific subgraphs in graphs.

In conclusion, there are many important applications for classical random walks.

1.2 From classical to quantum

The scalable quantum computer is a topical issue so that approaches of quantum computation are popular topics nowadays. Quantum walks are the corresponding part of classical random walks in quantum mechanics. The main difference between them is that quantum walks don't converge to some limiting distributions. Due to the quantum interference, quantum walks can spread significantly faster or slower than classical random walks.

There are many properties of quantum walks that make them differ from their classical counterparts. While in classical walks, in N steps the walks reaches a distance of $O(\sqrt{N})$, in quantum, it is a lot faster, $O(N)$. Quantum walks have inherent interferences, where some amplitudes can amplify, while others can be destrutive to each other and diminish. This makes them behave very differently from classical random walks and a good reason to study them.

Quantum walks are often used to accelerate classical algorithms. It can be used to decision trees, search problems, and element distinctness.

There are many algorithms based on Quantum Walks. Two types of models are present: continuous time and discrete time walks. Quantum decision tree algorithm is one of the continuous time quantum walk based algorithm. In this algorithm, you systematically explore the decision tree as a graph. The decision tree nodes are quantum states in a Hilbert space.

Based on Quantum Walks

- Quantum Decision Tree

- Quantum PageRank

- Grover Search Algorithm

1.3 Applications

- Collaborative Filtering

- Recommender System

- Link Prediction

- Computer Vision

- Semi-supervised

- Learning

- Network Embedding

- Element Distinctness

At the current state it is not viable to run these algorithms on quantum computers (if you don't have acces to google's or IBM's machinery, however even with simulation quite a few properties can be discovered and understood). I'm hoping to gain access to a viable computer to try out the quantum version eventually.

There is no real randomness in classical algorithms (pseudo random only), however quantum computers can create in the physical sense true randomness, which is an exciting opportunity for all randomization based algorithms.

Randomized algorithms are often simpler and easier to program than exact algorithms.

In randomized algorithms there is a probability of error, however if that probability is small enough the improvement in memory and speed may be worth it and the algorithm can be repeated to decrease the possibility of error (if the error probability is less than $\frac{1}{2}$).

Chapter 2

Classical random walks

Before introducing quantum computation and specifically quantum walks, I first overview classical random walks, based on the book Probability and Computing, written by Michael Mitzenmacher and Eli Upfal [4].

A *random walk* is a stochastic process modeled by a particular type of Markov chain. While a variety of Markov chains exist, in this work, I use the following definition exclusively.

Definition 2.1 (Markov chain). A discrete time stochastic process $X_0, X_1, X_2, \ldots$ on a finite state space A is a Markov chain if it has the Markov property:

$$P(X_k = a_k \mid X_{k-1} = a_{k-1}, \ldots, X_0 = a_0) = P(X_k = a_k \mid X_{k-1} = a_{k-1}) \quad \forall a_0, \ldots, a_k \in A.$$

Without loss of generality, we can assume, that $A = \{0, 1, \ldots, n\}$.

If the Markov chain is homogenous (time-invariant), the probability of moving from state $i \in A$ to state $j \in A$ is independent of k, and thus can be shortened the following way:

$$P(X_k = j \mid X_{k-1} = i) = p_{j \leftarrow i} = p_{j,i} \quad \forall k \in \mathbb{Z}^+.$$

Where $p_{j,i}$ is called the *transition probability* between states i and j.

The *transition matrix* $\mathbf{P}$ is formed by the transition probabilities.

$$\mathbf{P}[j, i] = p_{j,i}$$

It follows, that for each column in $\mathbf{P}$, the sum is 1.

$$\sum_{j=0}^{n} \mathbf{P}[j, i] = 1 \quad \forall i \in \{0, \ldots, n\}$$

Let the *probability distribution* of the process in the t-th step be π_t. Then, π_t can be computed from the starting distribution π_0 using $\mathbf{P}$.

$$\pi_t = \mathbf{P}^t \pi_0$$

The *stationary distribution* (π) of the Markov chain is a distribution that does not change with a transition, i.e. $\pi = \mathbf{P}\pi$.

Markov chains can be represented using graphs. A directed, weighted graph $G(V, E)$ with weight function $w : E \rightarrow [0, 1]$ represents a Markov chain, if $V = A$ and $w(i, j) = \mathbf{P}[j, i]$. If $\mathbf{P}[j, i] = 0$, then $\{i, j\} \notin E$.

A random walk on graph G starts from $X_0 = a_0$ and visits the vertices of the graph according to the states of the Markov-chain: $X_1 = a_1, X_2 = a_2, \dots$.

Frequently studied characteristics of random walks are *hitting time* [6] and *mixing time* [4]. Informally, hitting time describes how quickly can a vertex be reached from another vertex in the graph, while mixing time expresses how fast the walk reaches the stationary distribution, where the starting vertex can no longer be identified.

Definition 2.2 (Hitting time). Let $h_{j,i}$ be the expected number of steps before node j is visited in a random walk starting from node i. Then, $h_{j,i}$ is given by the following recursive formula:

$$h_{j,i} = \begin{cases} 1 + \sum_{k \in A} p_{j,k} h_{k,i} & \text{if } i \neq j \\ 0 & \text{if } i = j \end{cases}$$

Definition 2.3 (Mixing time). The smallest time index of the Markov chain, where the total variational distance between the current and the stationary distribution is not greater than a given ε. This measure still depends on the starting distrubiton π_0, so we take the maximum over all of the possible π_0 distributions.

$$m(\varepsilon) = \max_{\pi_0}\{\min\{t : \sum_{j=0}^{n} |\pi_t[j] - \pi[j]| \leq \varepsilon\}\}$$

Chapter 3

Quantum computing

Algorithms in quantum computing are derived from the postulates of quantum mechanics. These fundamental rules define how a quantum computer operates, and therefore they are essential for any discourse on quantum algorithms.

3.1 The postulates of quantum mechanics

This introduction is based on the following books: Quantum Computing and Communications by Sándor Imre and Ferenc Balázs [1], Quantum Computing by Mika Hirvensalo [2] and Quantum Walks and Search Algorithms by Renato Portugal [5].

Postulate I. State space

The state of an isolated physical system can be described using a unit length *state vector* in a Hilbert space (i.e. complex linear vector space), or *state space*, equipped with an inner product.

Definition 3.1 (Qubit). A state vector in the 2 dimensional Hilbert space (H_2) is a qubit. The base vectors in this space are

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \text{ and } |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

A generic qubit is written in the form

$$a|0\rangle + b|1\rangle = \begin{pmatrix} a \\ b \end{pmatrix}$$

where $a, b \in \mathbb{C}$ and $|a|^2 + |b|^2 = 1$.

Definition 3.2 (Superposition). A quantum system is said to be in *superpotion*, if its state vector is a linear combination of multiple basis states.

For example $a\left|0\right\rangle + b\left|1\right\rangle$ is in a superposition of the basis states $\left|0\right\rangle$ and $\left|1\right\rangle$, with probability amplitudes a and b.

Postulate II. Evolution

The time evolution of an isolated physical system is described using unitary transformation, which depends only on the starting and finishing time of the evolution.

A *quantum algorithm* is a sequence of unitary operators applied to an initial state.

Definition 3.3 (Unitary matrix). $\mathbf{U}$ is unitary if $\mathbf{U}^\dagger = \mathbf{U}^{-1}$.

Cite this!

The following definitions are equivalent:

1. $\mathbf{U}$'s rows form an orthonormal basis of $\mathbb{C}^n$.

2. $\mathbf{U}$'s columns form an orthonormal basis of $\mathbb{C}^n$.

3. $\mathbf{U}$ is an isometry: it is injective and preserves length.

4. $\mathbf{U}$ preserves the inner product.

Postulate III. Measurement

A quantum measurement is defined by a set of measurement operators $\{\mathbf{M}_m\}$, where m stands for the possible results of the measurement. The probability of measuring m if the system is in state $\left|v\right\rangle$ is

$$P(m|\left|v\right\rangle) = \left\langle v\right| \mathbf{M}_m^\dagger \mathbf{M}_m \left|v\right\rangle.$$

The state of the system after measuring m is then

$$\left|v'\right\rangle = \frac{\mathbf{M}_m \left|v\right\rangle}{\sqrt{\left\langle v\right| \mathbf{M}_m^\dagger \mathbf{M}_m \left|v\right\rangle}}.$$

The set of measurement operators have to satisfy the following *completeness relation*:

$$\sum_m \mathbf{M}_m^\dagger \mathbf{M}_m = I,$$

due to

$$1 = \sum_m P(m|\left|v\right\rangle) = \sum_m \left\langle v\right| \mathbf{M}_m^\dagger \mathbf{M}_m \left|v\right\rangle.$$

Projective measurement

To distinguish a set of orthonormal states $\{|\varphi_m\rangle\}$, the corresponding measurement operators can be produced as $\mathbf{P}_m = |\varphi_m\rangle\langle\varphi_m|$, with the following properties.

Property 3.1 ($\mathbf{P}_m$ is self adjoint).

$$\mathbf{P}_m^\dagger = \mathbf{P}_m$$

Since

$$\mathbf{P}_m^\dagger = (|\varphi_m\rangle\langle\varphi_m|)^\dagger = \langle\varphi_m|^\dagger\,|\varphi_m\rangle^\dagger = |\varphi_m\rangle\langle\varphi_m| = \mathbf{P}_m.$$

Property 3.2 ($\mathbf{P}_m$ is a projection).

$$\mathbf{P}_m\mathbf{P}_m = \mathbf{P}_m$$

Since

$$\mathbf{P}_m\mathbf{P}_m = (|\varphi_m\rangle\langle\varphi_m|)(|\varphi_m\rangle\langle\varphi_m|) = |\varphi_m\rangle(\langle\varphi_m|\varphi_m\rangle)\langle\varphi_m| = |\varphi_m\rangle\,1\,\langle\varphi_m| = |\varphi_m\rangle\langle\varphi_m| = \mathbf{P}_m.$$

Property 3.3 (The $\mathbf{P}_m$ are orthogonal).

$$m \neq n \Rightarrow \mathbf{P}_m\mathbf{P}_n = \mathbf{0}$$

Since

$$\mathbf{P}_m\mathbf{P}_n = (|\varphi_m\rangle\langle\varphi_m|)(|\varphi_n\rangle\langle\varphi_n|) = |\varphi_m\rangle(\langle\varphi_m|\varphi_n\rangle)\langle\varphi_n| = |\varphi_m\rangle\,0\,\langle\varphi_n| = \mathbf{0}.$$

From these properties follows, that the probability of measuring m in case of a projective measurement is

$$P(m|\,|v\rangle) = \langle v|\,\mathbf{P}_m^\dagger\mathbf{P}_m\,|v\rangle = \langle v|\,\mathbf{P}_m\mathbf{P}_m\,|v\rangle = \langle v|\,\mathbf{P}_m\,|v\rangle = \langle v|\varphi_m\rangle\langle\varphi_m|v\rangle = |\,\langle\varphi_m|v\rangle\,|^2.$$

For example, the value of a qubit can be any unit length vector in H_2, however when we measure it, we will receive one of the base vectors of H_2. For $a\,|0\rangle + b\,|1\rangle$ we measure 0 with probability $|a|^2$ and 1 with probability $|b|^2$.

Postulate IV. Composite systems

The state space of an isolated composite physical system is the *tensor product* of the state spaces of the individual components. The current state vector of the composite system is the *tensor product* of the current state vectors of the individual systems.

If $V_1, \ldots, V_n$ are the state spaces of the individual systems, then $V_1 \otimes \cdots \otimes V_n$ is the composite state space and for $|v_1\rangle \in V_1, \ldots, |v_n\rangle \in V_n$ state vectors, $|v_1\rangle \otimes \cdots \otimes |v_n\rangle = |v_1, \ldots, v_n\rangle$ is the state vector of the composite system.

Definition 3.4 (Tensor product). The tensor product $\mathbf{A} \otimes \mathbf{B}$ of matrix $\mathbf{A}_{(r \times s)}$ and matrix $\mathbf{B}_{(t \times u)}$ is of size $(rt \times su)$ and is defined as follows:

$$\text{For } \mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1s} \\ a_{21} & a_{22} & \dots & a_{2s} \\ \vdots & \vdots & \ddots & \vdots \\ a_{r1} & a_{r2} & \ddots & a_{rs} \end{pmatrix}, \text{ and } \mathbf{B} = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1u} \\ b_{21} & b_{22} & \dots & b_{2u} \\ \vdots & \vdots & \ddots & \vdots \\ b_{t1} & b_{t2} & \ddots & b_{tu} \end{pmatrix}$$

$$\mathbf{A} \otimes \mathbf{B} = \begin{pmatrix} a_{11}\mathbf{B} & a_{12}\mathbf{B} & \dots & a_{1s}\mathbf{B} \\ a_{21}\mathbf{B} & a_{22}\mathbf{B} & \dots & a_{2s}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{r1}\mathbf{B} & a_{r2}\mathbf{B} & \ddots & a_{rs}\mathbf{B} \end{pmatrix}$$

and has the following properties:

Cite this!

Property 3.4 (Associativity).

$$(\mathbf{A} \otimes \mathbf{B}) \otimes \mathbf{C} = \mathbf{A} \otimes (\mathbf{B} \otimes \mathbf{C})$$

Property 3.5 (Mixed product property). If the corresponding matrices are compatible, then

$$(\mathbf{A} \otimes \mathbf{B})(\mathbf{C} \otimes \mathbf{D}) = (\mathbf{A}\mathbf{C}) \otimes (\mathbf{B}\mathbf{D}),$$

and as an immediate consequence, we obtain

$$(\mathbf{A} \otimes \mathbf{I})(\mathbf{I} \otimes \mathbf{B}) = \mathbf{A} \otimes \mathbf{B}.$$

Definition 3.5 (Quantum register). The composite system of n qubits is a *quantum register*, having the composite state space

$$H_2^{\otimes n} = H_2 \otimes H_2 \otimes \dots \otimes H_2$$

and for $|q_{n-1}\rangle \in H_2, \dots, |q_0\rangle \in H_2$ individual state vectors, the composite state vector is

$$|q_{n-1}\rangle \otimes \dots \otimes |q_0\rangle = |q_{n-1}, \dots, q_0\rangle .$$

Definition 3.6 (Entangled state). Any state consisting of multiple qubits, that is not decomposable, i.e. that can not be written in the form of a composite system of qubits is *entangled.*

For example, the state $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ is entangled, since it can not be written in the form

$$(a_0 |0\rangle + a_1 |1\rangle) \otimes (b_0 |0\rangle + b_1 |1\rangle) = a_0 b_0 |00\rangle + a_0 b_1 |01\rangle + a_1 b_0 |10\rangle + a_1 b_1 |11\rangle$$

since that would require from $a_0 b_0 = a_1 b_1 = \frac{1}{\sqrt{2}}$, for all coefficients to be non-zero and from $a_0 b_1 = a_1 b_0 = 0$ for either a_0 or b_1 and either a_1 or b_0 to be zero, which is a contradiction.

Chapter 4

Quantum walks

In classical random walks, the walker moves from the current vertex via one of its outgoing edges, chosen randomly, weighted by the edge weights. This random choice can be interpreted as a (generalized) coin toss.

To formulate a quantum version of graph walking, we define the quantum coin, which will replace the classical concept of randomness with quantum superposition.

4.1 Formulating the Quantum coin

I used Renato Portugal's Quantum Walks and Search Algorithms [5] book as a reference for the different types of coins I present in this section.

A *quantum coin* is a quantum system, which behaves according to the postulates of quantum mechanics. It has a current state, represented by a state vector in a Hilbert space and a unitary time evolution operator, describing a coin toss.

After tossing the coin, the resulting coin state chooses the next step of the quantum walker. If there are d outgoing edges to choose from, then the coin's state space must have d orthonormal basis states, each corresponding to one of the possible edges. If the current state is one of the basis states, then the walker moves in that direction. However, in the quantum world, the coin can also be in a superposition, consisting of multiple basis states. This means that the walker will simultaneously move in all corresponding directions and occupy more than one vertex at the same time, resulting in the walker spreading over the graph in a superposition.

For the d dimensional coin state, the corresponding coin flip operator is a $(d \times d)$ dimensional unitary matrix. Based on what the transition operator is, multiple types of coins can be defined. The following ones are typically used in quantum walks.

4.1.1 Hadamard coin

The Hadamard coin is the most commonly used quantum coin. It is defined by the Hadamard-matrix as a transition operator:

$$\mathbf{H} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}.$$

If the starting coin state is $|0\rangle$, then flipping the coin once results in the following state:

$$\mathbf{H}\,|0\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}\begin{pmatrix} 1 \\ 0 \end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}}\,|0\rangle + \frac{1}{\sqrt{2}}\,|1\rangle.$$

If we measured the above coin, the probability of measuring 0 is

$$P(0 \mid \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)) = \left|\frac{1}{\sqrt{2}}\right|^2 = \frac{1}{2}.$$

Similarly, if the starting coin state is $|1\rangle$, then flipping the coin once results in the following state:

$$\mathbf{H}\,|1\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}\begin{pmatrix} 0 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ -1 \end{pmatrix} = \frac{1}{\sqrt{2}}\,|0\rangle - \frac{1}{\sqrt{2}}\,|1\rangle.$$

The probability of measuring 1 here is similarly

$$P(1 \mid \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)) = \left|-\frac{1}{\sqrt{2}}\right|^2 = \frac{1}{2}.$$

An unexpected feature of this coin comes from the fact, that the Hadamard-matrix is Hermitian (self-adjoint), i.e. $\mathbf{H}^\dagger = \mathbf{H}$, while also unitary, i.e. $\mathbf{H}^\dagger = \mathbf{H}^{-1}$, which results in $\mathbf{H}^{-1} = \mathbf{H}$, thus $\mathbf{H}\mathbf{H} = \mathbf{I}$. This means, that after flipping the coin twice without measuring it, it will return the coin state to its origin. For example, starting from $|0\rangle$:

$$\mathbf{H}^2\,|0\rangle = \mathbf{H}\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = \frac{1}{2}(|0\rangle + |1\rangle + |0\rangle - |1\rangle) = |0\rangle.$$

After the second flip, the probability of measuring $|0\rangle$ is 1, due to the destructive interference between the two $|1\rangle$ probability amplitudes, demonstrating a significant contrast between classical and quantum walks.

Definition 4.1 (2^n dimensional Hadamard-coin). A 2^n dimensional Hadamard-coin operator can be created by taking the tensor product of the 2 dimensional Hadamard-coin n times: $\mathbf{H}^{\otimes n}$.

4.1.2 Grover coin

The Grover coin originates from Grover's search algorithm, where it is applied as the diffusion operator.

Let $|D\rangle$ be the following state:

$$|D\rangle = \mathbf{H}^{\otimes n} |0\rangle = \frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i\rangle .$$

Using $|D\rangle$, the Grover coin is the following unitary matrix:

$$\mathbf{G} = 2 |D\rangle \langle D| - \mathbf{I}.$$

If $N = 2^n$, then $\mathbf{G}$ unrolls to the following representation:

$$\mathbf{G} = \begin{pmatrix} \frac{2}{N} - 1 & \frac{2}{N} & \cdots & \frac{2}{N} \\ \frac{2}{N} & \frac{2}{N} - 1 & \cdots & \frac{2}{N} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{2}{N} & \frac{2}{N} & \ddots & \frac{2}{N} - 1 \end{pmatrix}.$$

4.1.3   Fourier coin

In contrast to the Hadamard and Grover coins, the Fourier coin can be of any size, not just a power of 2. A size N Fourier-coin, F_N is defined by the matrix of the Quantum Fourier Transform:

$$\mathbf{F}[k, l] = \frac{1}{\sqrt{N}} \omega^{kl}$$

where ω is the N-th root of unity,

$$\omega = e^{\frac{2\pi i}{N}}.$$

$\mathbf{F}$ unrolls to the following representation:

$$\mathbf{F} = \frac{1}{\sqrt{N}} \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{N-1} & \omega^{2(N-1)} & \cdots & \omega^{(N-1)(N-1)} \end{pmatrix}$$

4.2   Quantum walks on the line

Kempe introduces quantum walks from a physicist's perspective in [3] using a particle characterised by its position on the line $|x\rangle$ and its spin state $|s\rangle$.

4.2.1 State space

Spin state

The spin state is in H_2 with the basis states spin up and down:

$$|\uparrow\rangle = |0\rangle \,,$$
$$|\downarrow\rangle = |1\rangle \,.$$

The spin state vector is then given by:

$$|s\rangle = s_0 \,|\uparrow\rangle + s_1 \,|\downarrow\rangle \,.$$

Position state

At the start of the walk the particle is at the origin $|0\rangle$ and the walking lasts for N steps. The position state is in $H_{(2N+1)}$ with the following basis vectors corresponding to the possible positions on the line.

$$\{|-N\rangle\,, |-(N-1)\rangle\,, \ldots, |-1\rangle\,, |0\rangle\,, |1\rangle\,, \ldots, |N-1\rangle\,, |N\rangle\}$$

I index the basis states using negative numbers to match the labels on the axis.

The position state vector is then given by:

$$|x\rangle = \sum_{i=-N}^{N} x_i \,|i\rangle \,.$$

Composite state

The composite state of the system, according to [PostulateIV] is then

$$|x\rangle \otimes |s\rangle \,.$$

4.2.2 Evolution

The particle travels on the line based on its current spin state:

- If the current spin state is $|0\rangle$, the particle moves to the left, i.e. from position $|i\rangle$ the particle travels to position $|i-1\rangle$.

- If the current spin state is $|1\rangle$, the particle moves to the right, i.e. from position $|i\rangle$ the particle travels to position $|i+1\rangle$.

This step is realised with the unitary matrix **S** which operates on the complete state of the system, $|x\rangle \otimes |s\rangle$ and is assembled from a left and a right shift operator acting on $|x\rangle$ and another operator for checking $|s\rangle$ compiled using tensor product.

Definition 4.2 (Left shift operator). To move from position $|i\rangle$ to the left $(|i-1\rangle)$ the position vector is multiplied with the following **L** matrix, containing 1's above the diagonal. To keep **S** unitary, an unused transition must be added in the lower left corner.

> Előrehivatkozni a permutációmátrixos tételre.

$$\mathbf{L} = |N\rangle \langle -N| + \sum_{i=-(N-1)}^{N} |i-1\rangle \langle i| = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ 0 & & & \ddots & 1 \\ 1 & 0 & \cdots & & 0 \end{pmatrix} \tag{4.1}$$

For a given basis vector $|j\rangle$ only one of the summands in **L** is non-zero, where $i = j$, resulting in the required shift being performed.

$$\mathbf{L}|j\rangle = |j-1\rangle \langle j|j\rangle = |j-1\rangle$$

Definition 4.3 (Right shift operator). To move from position $|i\rangle$ to the right $(|i+1\rangle)$ the position vector is multiplied with the following **R** matrix, containing 1's under the diagonal. To keep **S** unitary, an unused transition must be added in the top right corner.

> Előrehivatkozni a permutációmátrixos tételre.

$$\mathbf{R} = |-N\rangle \langle N| + \sum_{i=-N}^{N-1} |i+1\rangle \langle i| = \begin{pmatrix} 0 & & \cdots & 0 & 1 \\ 1 & \ddots & & & 0 \\ 0 & \ddots & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \\ 0 & \cdots & 0 & 1 & 0 \end{pmatrix} \tag{4.2}$$

For a given basis vector $|j\rangle$ only one of the summands in **R** is non-zero, where $i = j$, resulting in the required shift being performed.

$$\mathbf{R}|j\rangle = |j+1\rangle \langle j|j\rangle = |j+1\rangle$$

Shift operator

Using matrixes L and R operating on the position register $|x\rangle$ only, we construct a unitary operator S, which operates on the composite state of the system, $|x\rangle \otimes |s\rangle$, executing matrix **L** on $|x\rangle$ only when $|s\rangle = |0\rangle$ and matrix **R** only when $|s\rangle = |1\rangle$.

$$\mathbf{S} = \mathbf{L} \otimes |0\rangle \langle 0| + \mathbf{R} \otimes |1\rangle \langle 1| \tag{4.3}$$

The execution logic is as follows:

$$\mathbf{S}(|x\rangle \otimes |s\rangle) =$$
$$(\mathbf{L} \otimes |0\rangle \langle 0| + \mathbf{R} \otimes |1\rangle \langle 1|)(|x\rangle \otimes |s\rangle) =$$
$$(\mathbf{L} \otimes |0\rangle \langle 0|)(|x\rangle \otimes |s\rangle) + (\mathbf{R} \otimes |1\rangle \langle 1|)(|x\rangle \otimes |s\rangle) = \ldots$$

using [TensorMixedProduct]:

$$\cdots = \mathbf{L}|x\rangle \otimes (|0\rangle \langle 0|s\rangle) + \mathbf{R}|x\rangle \otimes (|1\rangle \langle 1|s\rangle) =$$
$$|x-1\rangle \otimes s_0 |0\rangle + |x+1\rangle \otimes s_1 |1\rangle =$$
$$s_0 |x-1, 0\rangle + s_1 |x+1, 1\rangle.$$

- If the spin state was $|s\rangle = |0\rangle$ at the beginning, then $s_0 = 1$ and $s_1 = 0$, which means that the resulting system state is $|x-1, 0\rangle$, which means that the particle shifted to the left, as designed.

- If the spin state was $|s\rangle = |1\rangle$ at the beginning, then $s_0 = 0$ and $s_1 = 1$, which means that the resulting system state is $|x+1, 1\rangle$, which means that the particle shifted to the right, also as intended.

Furthermore, the spin state can be any mixed state $s_0 |0\rangle + s_1 |1\rangle$ as well. In this case the particle will shift *both* to the left and to the right, at the same time. When measured, the particle can be found in position $|x-1\rangle$ with probability $|s_0|^2$ and in position $|x+1\rangle$ with probability $|s_1|^2$.

In quantum graph walks, the walker can simultaneously explore multiple parallel paths in the graph, at the same time. With good design, this behaviour can be used to search the graph faster than in classical random graph walks.

Coin operator

To inject the quantum superposition into the walk, the particle's spin state is transformed using any 2 dimensional unitary matrix between shift operations. The Hadamard, Grover and Fourier coins mentioned earlier are commonly used as coin operators.

For any C operator on the coin register, the unitary transform for the composite system is defined as follows:

$$\hat{\mathbf{C}} = \mathbf{I} \otimes \mathbf{C}$$

since the coin operator does not modify the position register.

Evolution operator

Combining the shift operator and the coin operator together, we obtain the following evolution operator, defining one step of the quantum walk on the line. The step consists of flipping the coin once, then applying the shifting the walker's position accordingly, as follows:

$$\mathbf{U} = \mathbf{S}\hat{\mathbf{C}} = \mathbf{S}(\mathbf{I} \otimes \mathbf{C})$$

4.2.3   Measurement

To measure the probability of the particle being at position $|i\rangle$, the projective measurement operator acting on $|x\rangle$ is defined as $\mathbf{P_i} = |i\rangle\langle i|$, in accordance with [PostulateIIIProjective].

Since the coin register need not be measured, we apply the identity operator on it, using $\mathbf{P_i} \otimes \mathbf{I}$ on the complete system to measure the particle's current position.

The probability of finding the particle in position i is:

$$P(i|\,|x\rangle) = \langle x, s|\,\mathbf{P_i} \otimes \mathbf{I}\,|x, s\rangle = \dots$$

using [TensorMixedProduct]:

$$\dots = \langle x|\,\mathbf{P_i}\,|x\rangle\,\langle s|\,\mathbf{I}\,|s\rangle = \langle x|\,\mathbf{P_i}\,|x\rangle\,1 = \langle x|\,\mathbf{P_i}\,|x\rangle = \langle x|i\rangle\,\langle i|x\rangle = |\,\langle i|x\rangle\,|^2 = |x_i|^2$$

4.3   Generalization of Quantum Walks

After presenting quantum walking on the line, I review and extend two approaches to generalize it in this chapter.

1. **Use multiple two-dimensional coins**: In [5], Renato Portugal shows the generalization of quantum walks on a line to a two-dimensional grid, using a method with 2 two-dimensional coins. In this work, I prove how his method reduces to effectively two synchronous independent walks on the x and y axes. Then I improve his technique by generalizing to arbitrarily large dimensional grids in a more memory-efficient way than what would naturally follow from his description.

2. **Use a single higher dimensional coin**: In [5], Renato Portugal describes the generalization of a quantum walk on a line to an arbitrary undirected graph and gives the necessary condition for creating the unitary transition matrix without proof. In this work, I generalize to directed graphs and give proof of the generalization of the condition using directed graphs.

4.3.1   Generalization using multiple independent 2 dimensional coins

TODO

4.3.1.1 Quantum Walk on a 2D grid

In [5] defines the following method:

Let the position state of the walker be $|x, y\rangle$ and the two coins $|c_x\rangle$, acting on the x coordinate and $|c_y\rangle$, acting on the y coordinate of the walker.

The shift operator moves the walker on the grid diagonally, according to the current state of the two coins, described by

$$\mathbf{S} |x, y\rangle |c_x\rangle |c_y\rangle = |x + (-1)^{c_x}, y + (-1)^{c_y}\rangle |c_x\rangle |c_y\rangle, \tag{4.4}$$

and the coin operator leaves the position state in place while flipping both coins at the same time, described by

$$\hat{\mathbf{C}} = \mathbf{I} \otimes \mathbf{C_4} = \mathbf{I} \otimes (\mathbf{C_2} \otimes \mathbf{C_2}). \tag{4.5}$$

Issues with this method

In 4.4, we can see how the matrix $\mathbf{S}$ will quickly increase in size, as further dimensions are added to the equation. In 2D, if the walker takes N steps, the size of $\mathbf{S}$ is

$$(2N + 1)^2 (2N + 1)^2 2^2 2^2 = 16(2N + 1)^4 = O(N^4).$$

To increase the dimension count, one would naturally append more coordinates to the composite position state and add further coins, for example in 3D $\mathbf{S}$ would become

$$\mathbf{S} |x, y, z\rangle |c_x\rangle |c_y\rangle |c_z\rangle = |x + (-1)^{c_x}, y + (-1)^{c_y}, z + (-1)^{c_z}\rangle |c_x\rangle |c_y\rangle |c_z\rangle,$$

For a dimension count d, the size of $\mathbf{S}$ is exponential in d.

$$((2N + 1)^2)^d (2^2)^d = (4(2N + 1)^2)^d = O(N^{2d}).$$

My improvements

Since in 4.4 the coordinates of the walker are updated independently by the separate coins, I was able to disassemble $\mathbf{S}$ into smaller matrices, using the properties of the tensor product.

To do this, I first needed to define $\mathbf{S}$ in 2D explicitly (as opposed to the implicit definition in 4.4, stating only how $\mathbf{S}$ updates the state of the system). I will be using the matrices $\mathbf{L}$ defined by 4.1 and $\mathbf{R}$ defined by 4.2.

Notice that $\mathbf{R}$ increases the walker's coordinates on the line, while $\mathbf{L}$ decreases them. This means, that on the y axis $\mathbf{R}$ acts by moving the walker up, and $\mathbf{L}$ acts by moving the walker down.

When the coins are in the state $|c_x, c_y\rangle = |0, 0\rangle$, the walker moves up and to the right. This movement is captured by $\mathbf{S}_{0,0}$, acting on the position state:

$$\mathbf{S}_{0,0} = \mathbf{R} \otimes \mathbf{R}$$

The other 3 $\mathbf{S}_{c_x, c_y}$ matrices are defined similarly:

$$\mathbf{S}_{0,1} = \mathbf{R} \otimes \mathbf{L}$$
$$\mathbf{S}_{1,0} = \mathbf{L} \otimes \mathbf{R}$$
$$\mathbf{S}_{1,1} = \mathbf{L} \otimes \mathbf{L}$$

Then, I assemble $\mathbf{S}$ using $\mathbf{S}_{0,0}$, $\mathbf{S}_{0,1}$, $\mathbf{S}_{1,0}$ and $\mathbf{S}_{1,1}$ the following way: I want

- $\mathbf{S}_{0,0}$ to act only when $|c_x, c_y\rangle = |0, 0\rangle$,

- $\mathbf{S}_{0,1}$ to act only when $|c_x, c_y\rangle = |0, 1\rangle$,

- $\mathbf{S}_{1,0}$ to act only when $|c_x, c_y\rangle = |1, 0\rangle$, and finally

- $\mathbf{S}_{1,1}$ to act only when $|c_x, c_y\rangle = |1, 1\rangle$.

Using the same method as in 4.3 I arrive at:

$$\mathbf{S} = \mathbf{S}_{0,0} \otimes |0, 0\rangle \langle 0, 0| +$$
$$\mathbf{S}_{0,1} \otimes |0, 1\rangle \langle 0, 1| +$$
$$\mathbf{S}_{1,0} \otimes |1, 0\rangle \langle 1, 0| +$$
$$\mathbf{S}_{1,1} \otimes |1, 1\rangle \langle 1, 1|$$

After substituting the $\mathbf{S}_{c_x, c_y}$ matrices in:

$$\mathbf{S} = (\mathbf{R} \otimes \mathbf{R}) \otimes |0, 0\rangle \langle 0, 0| +$$
$$(\mathbf{R} \otimes \mathbf{L}) \otimes |0, 1\rangle \langle 0, 1| +$$
$$(\mathbf{L} \otimes \mathbf{R}) \otimes |1, 0\rangle \langle 1, 0| +$$
$$(\mathbf{L} \otimes \mathbf{L}) \otimes |1, 1\rangle \langle 1, 1|$$

Let us name the coin state $|0\rangle$ heads and the coin state $|1\rangle$ tails. Then, using the following equalities amd introducing matrices $\mathbf{H}$ and $\mathbf{T}$, as shorthands:

$$|0,0\rangle\langle 0,0| = (|0\rangle\langle 0|)\otimes(|0\rangle\langle 0|) = \mathbf{H}\otimes\mathbf{H}$$
$$|0,1\rangle\langle 0,1| = (|0\rangle\langle 0|)\otimes(|1\rangle\langle 1|) = \mathbf{H}\otimes\mathbf{T}$$
$$|1,0\rangle\langle 1,0| = (|1\rangle\langle 1|)\otimes(|0\rangle\langle 0|) = \mathbf{T}\otimes\mathbf{H}$$
$$|1,1\rangle\langle 1,1| = (|1\rangle\langle 1|)\otimes(|1\rangle\langle 1|) = \mathbf{T}\otimes\mathbf{T}$$

I arrive at:

$$\mathbf{S} = (\mathbf{R}\otimes\mathbf{R})\otimes(\mathbf{H}\otimes\mathbf{H})+$$
$$(\mathbf{R}\otimes\mathbf{L})\otimes(\mathbf{H}\otimes\mathbf{T})+$$
$$(\mathbf{L}\otimes\mathbf{R})\otimes(\mathbf{T}\otimes\mathbf{H})+$$
$$(\mathbf{L}\otimes\mathbf{L})\otimes(\mathbf{T}\otimes\mathbf{T})$$

Then, using [TensorMixedProduct] I inflate the equation with (appropriately sized) $\mathbf{I}$ matrices:

$$\mathbf{S} = ((\mathbf{R}\otimes\mathbf{I})(\mathbf{I}\otimes\mathbf{R}))\otimes((\mathbf{H}\otimes\mathbf{I})(\mathbf{I}\otimes\mathbf{H}))+$$
$$((\mathbf{R}\otimes\mathbf{I})(\mathbf{I}\otimes\mathbf{L}))\otimes((\mathbf{H}\otimes\mathbf{I})(\mathbf{I}\otimes\mathbf{T}))+$$
$$((\mathbf{L}\otimes\mathbf{I})(\mathbf{I}\otimes\mathbf{R}))\otimes((\mathbf{T}\otimes\mathbf{I})(\mathbf{I}\otimes\mathbf{H}))+$$
$$((\mathbf{L}\otimes\mathbf{I})(\mathbf{I}\otimes\mathbf{L}))\otimes((\mathbf{T}\otimes\mathbf{I})(\mathbf{I}\otimes\mathbf{T}))$$

At this point, I introduce a few aliases to make the equation more manageable. Notice, how for example $\mathbf{I}\otimes\mathbf{R}$ is acting on the 2D position state, but only updating the y coordinate to move the walker upward. This gives a way to naturally define $\mathbf{S}_{\text{up}} = \mathbf{I}\otimes\mathbf{R}$, and similarly:

$$\mathbf{S}_{\text{right}} = \mathbf{R}\otimes\mathbf{I}$$
$$\mathbf{S}_{\text{left}} = \mathbf{L}\otimes\mathbf{I}$$
$$\mathbf{S}_{\text{up}} = \mathbf{I}\otimes\mathbf{R}$$
$$\mathbf{S}_{\text{down}} = \mathbf{I}\otimes\mathbf{L}$$

At the same time, for example $\mathbf{I}\otimes\mathbf{H}$ is acting on the composite coin state, but only checking if the second coin's state is heads, which is the coin for the y axis. This gives a way to naturally define $\mathbf{H}_y = \mathbf{I}\otimes\mathbf{H}$, and similarly:

$$\mathbf{H}_x = \mathbf{H}\otimes\mathbf{I}$$
$$\mathbf{T}_x = \mathbf{T}\otimes\mathbf{I}$$
$$\mathbf{H}_y = \mathbf{I}\otimes\mathbf{H}$$
$$\mathbf{T}_y = \mathbf{I}\otimes\mathbf{T}$$

Substituting all of the aliases:

$$\mathbf{S} = (\mathbf{S}_{\text{right}}\mathbf{S}_{\text{up}}) \otimes (\mathbf{H}_x\mathbf{H}_y)+$$
$$(\mathbf{S}_{\text{right}}\mathbf{S}_{\text{down}}) \otimes (\mathbf{H}_x\mathbf{T}_y)+$$
$$(\mathbf{S}_{\text{left}}\mathbf{S}_{\text{up}}) \otimes (\mathbf{T}_x\mathbf{H}_y)+$$
$$(\mathbf{S}_{\text{left}}\mathbf{S}_{\text{down}}) \otimes (\mathbf{T}_x\mathbf{T}_y)$$

Then, using [TensorMixedProduct] again, I arrive at:

$$\mathbf{S} = (\mathbf{S}_{\text{right}} \otimes \mathbf{H}_x)(\mathbf{S}_{\text{up}} \otimes \mathbf{H}_y)+$$
$$(\mathbf{S}_{\text{right}} \otimes \mathbf{H}_x)(\mathbf{S}_{\text{down}} \otimes \mathbf{T}_y)+$$
$$(\mathbf{S}_{\text{left}} \otimes \mathbf{T}_x)(\mathbf{S}_{\text{up}} \otimes \mathbf{H}_y)+$$
$$(\mathbf{S}_{\text{left}} \otimes \mathbf{T}_x)(\mathbf{S}_{\text{down}} \otimes \mathbf{T}_y)$$

Then using the distributive property of matrix multiplication with respect to matrix addition I finally arrive at:

$$\mathbf{S} = ((\mathbf{S}_{\text{right}} \otimes \mathbf{H}_x) + (\mathbf{S}_{\text{left}} \otimes \mathbf{T}_x))((\mathbf{S}_{\text{up}} \otimes \mathbf{H}_y) + (\mathbf{S}_{\text{down}} \otimes \mathbf{T}_y))$$

We can see from the equation above, that $\mathbf{S}$ is actually the product of two shift operators, one only acting on the x coordinate using the first coin's state, the other acting only on the y coordinate, according to the second coin's state.

$$\mathbf{S}_x = (\mathbf{S}_{\text{right}} \otimes \mathbf{H}_x) + (\mathbf{S}_{\text{left}} \otimes \mathbf{T}_x)$$
$$\mathbf{S}_y = (\mathbf{S}_{\text{up}} \otimes \mathbf{H}_y) + (\mathbf{S}_{\text{down}} \otimes \mathbf{T}_y)$$

$$\mathbf{S} = \mathbf{S}_x\mathbf{S}_y$$

TODO finish this

- Rács = két körnek a tenzorszorzata.

- Fel/le = mindig 1 sorral felette/alatta lévő (modulóval) van összekötve = n-el van elcsúsztatva.

- Függőlegesen számozva ugyanígy csak fordított elnevezések.

- Jobbra-balra = 1 távolság, kivéve sor végén, ahol a sor elejére ugrik = kis körök.

- Tenzorszorzás tulajdonságait előre kiszedni, műveletek, használatkor hivatkozni.

- Többsoros képleteknek csak 1 száma legyen.

4.3.2 Generalization using a single higher dimensional coin

Using this method, we generalize quantum walking to arbitrary directed graphs. To do so, we first generalize to regular graphs, then discuss how non-regular graphs can be regularized. This introduction is done following Renato Portugal's book [5].

In a d-regular graph, the vertices have d neighbours, meaning the walker must choose from d possible directions at every step. Thus the coin is d dimensional. Using this idea as a starting point, we can reverse engineer the generalized walk from the walk on the line by starting from the evolution operator, which assumes nothing about the given graph or coin.

$$\mathbf{U} = \mathbf{S\hat{C}} = \mathbf{S}(\mathbf{I} \otimes \mathbf{C})$$

In this equation, $\mathbf{C}$ can be any d-dimensional unitary matrix. The definition of $\mathbf{S}$ requires more thought, since $\mathbf{S}$ has to encode the graph's structure, while also satisfying [PostulateII].

In 1 dimension, $\mathbf{S}$ was defined the following way:

$$\mathbf{S} = \mathbf{L} \otimes |0\rangle \langle 0| + \mathbf{R} \otimes |1\rangle \langle 1|.$$

$|0\rangle \langle 0|$ and $|1\rangle \langle 1|$ were matrices that checked the current state of the coin and "activated" the transition $\mathbf{L}$ or $\mathbf{R}$ accordingly. In d dimensions, the coin has d possible states

$$\{|0\rangle, |1\rangle, \ldots, |d-1\rangle\},$$

which means $\mathbf{S}$ will be constructed using d transition matrices, describing the graph's structure

$$\mathbf{S} = \mathbf{S}_0 |0\rangle \langle 0| + \mathbf{S}_1 |1\rangle \langle 1| + \cdots + \mathbf{S}_{d-1} |d-1\rangle \langle d-1|.$$

In the 1 dimensional case $\mathbf{L}$ and $\mathbf{R}$ described stepping to the left and to the right, which are the directed edges of the line graph and $\mathbf{L} + \mathbf{R}$ is the adjacency matrix of the line graph. To generalize this, $\mathbf{S}_0 + \mathbf{S}_1 + \cdots + \mathbf{S}_{d-1}$ is going to be the adjacency matrix of the d-regular graph.

The question is how do we construct the matrices $\mathbf{S}_0, \mathbf{S}_1, \ldots, \mathbf{S}_{d-1}$ from a given adjacency matrix of a d-regular graph? It turns out, that in order for $\mathbf{S}$ to satisfy [PostulateII], there are strict rules on how these S_i matrices can be defined.

It seems to be a well-known fact in the literature, that for undirected graphs, given a valid coloring of the edges the for all i, the S_i matrix contains the edges colored using ith color.

The coin operator can be defined as before, by applying the identity operator on the position register and any coin flip unitary matrix on the coin register.

To define the shift operator, let's introduce the edge chromatic number of a graph.

Definition 4.4. Edge coloring

An edge coloring of a graph is an assignment of colors to the edges of the graph, so that no two incident edges have the same color.

Definition 4.5. Edge chromatic number, $\chi'(G)$ is the minimum number of colors that can be used to correctly color the graph.

Theorem 4.1. Vizing's theorem

The edge chromatic number $\chi'(G)$ of graph G is $\Delta(G) \leq \chi'(G) \leq \Delta(G) + 1$. Graphs with $\chi'(G) = \Delta(G)$ are called Class 1 graphs and graphs with $\chi'(G) = \Delta(G) + 1$ are called Class 2 graphs.

Ez az a tétel amit nem találtam meg sehol, itt kellene írni hogy én bizonyítottam, de ismert eredmény.

Theorem 4.2. For Class 1, d-regular graphs, the coin sides correspond to the color sets of a correct edge coloring.

d-reguláris gráfra d oldalú érmével:

- A gráf egy adott csúcsából úgy döntjük el, hogy melyik következő csúcsba lépünk tovább, hogy feldobjuk az érmét és a kapott oldalhoz tartozó élen megyünk a következő csúcsba.

- Ehhez akkor tudni kell, hogy adott csúcsból nézve melyik hozzá tartozó él melyik érmeoldalhoz tartozik.

- Az érmeoldalakat számozzuk 0,...,$d-1$ -el.

- Minden csúcsban a hozzá tartozó d darab élet megszámozzuk 0,...,$d-1$-el.

- (Ha ezt lehet úgy, hogy minden élre az él két végéhez ugyanaz a szám tartozik akkor irányítatlan gráfként kezeljük, ha nem lehet akkor meg irányítottként és a kezdőpontjában az eredeti élhez rendelt számot kapja. ?)

- Ha egy adott csúcsban i-t dobunk az érmével akkor a hozzá tartozó i. sorszámú élen megyünk tovább.

- Ezt a mesét most mátrixszorzásokkal akarjuk felírni:

- Az érme legyen az $|i\rangle$ vektor $i = 0...,d-1$, az pedig hogy éppen melyik gráfcsúcsban vagyunk legyen a $|v\rangle$ vektor $v = 0,...,n-1$.

- Akkor az aktuális állapot (melyik oldala van felül az érmének és hol vagyok a gráfban) $|v\rangle \otimes |i\rangle$.

- Jelölje S_i azt a mátrixot, ahol akkor van az x. sor y. oszlopában 1-es, ha van a gráfban (y,x) él és az y csúcsban az él sorszáma i.

- Ekkor minden oszlopban csak 1 darab nem 0 elem van, mert az y csúcsnak csak 1 darab i. szomszédja van.

- Azt a lépést, hogy ha az érme k-t mutat akkor az adott csúcs k. szomszédjába megyek $S_k \otimes |k\rangle \langle k|$ -vel lehet leírni, mert ha ezzel balról szorzom a $|v\rangle \otimes |i\rangle$-t akkor ez csak akkor nem 0 ha $\langle k|i\rangle$ nem 0, vagyis ha $k = i$ és ilyenkor az S_k pont a v csúcs $i = k$. szomszédjába transzformálja $|v\rangle$-t.

- Akkor a teljes lépés mátrix $S = \sum\limits_{i=0}^{d-1} S_i \otimes |i\rangle \langle i|$.

- Ennek unitérnek kell lennie, mert kvantum.

- Ezért az oszlopok ortonormált bázist alkotnak. (Meg a sorok is, de az most nem kell.)

- Ezért az oszlopok skaláris szorzata 0 kell hogy legyen.

- Nincsenek negatív értékek az S_i mátrixokban és a $|k\rangle \langle k|$-kban sem és ezért az S-ben sem, ezért ez csak úgy lehet ha nincs két olyan különböző oszlop S-ben ahol ugyanannyiadik cellában van nem 0 érték.

- Kétféleképpen fordulhat elő, hogy S-ben két különböző oszlopban ugyanannyiadik cellában nem 0 érték: a két nem 0 elem vagy ugyanabból az $S_i \otimes |i\rangle \langle i|$ mátrixból jött az $S = \sum\limits_{i=0}^{d-1} S_i \otimes |i\rangle \langle i|$ összegben vagy 2 különbözőből.

- Különbözőből nem jöhetett, mert az $|i\rangle \langle i|$-vel való tenzorszorzás különböző sorokba fogja tenni az 1-eseket a különböző $S_i \otimes |i\rangle \langle i|$-kre.

- Ha azonosból jött akkor egy $S_i \otimes |i\rangle \langle i|$ mátrixban van két különböző oszlop ahol ugyanannyiadik cellában van nem 0 elem.

- Mivel $|i\rangle \langle i|$-ben csak 1 darab nem 0 elem van, ezért

- Akkor az S_i mátrixban van két különböző oszlop ahol ugyanannyiadik cellában van nem 0 elem.

És akkor ezek a lépések eddig végig "akkor és csak akkor" kapcsolatban voltak egymással.

Ez igaz?

Vagyis az jött ebből ki, hogy akkor és csak akkor unitér S ha az S_i mátrixokban nincs két különböző oszlop ahol ugyanannyiadik cellában van nem 0 elem. Vagyis minden sorban csak 1 darab nem 0 elem lehet. Közben pedig valahol felírtuk hogy minden oszlopban is csak 1 darab nem 0 elem lehet. És mivel d-reguláris a gráf ezért minden sorban és oszlopban van is 1 darab 1-es és ezért ezek permutációmátrixok lehetnek csak.

Az S_i pedig egy élszínosztályt ír le, mert ha az x. sor y. eleme 1-es akkor az egy él a gráfban és S_i-ben nincs másik él ami y-ból mutatna kifele vagy x-be mutatna befele. (Itt akkor kell hogy irányított legyen, mert az x. oszlopban vagy az y. sorban még lesz 1-es.)

És akkor ilyen módon csak azokra a gráfokra lehet csak sétát definiálni amik d színnel élszínezhetőek és amik pedig $d + 1$-el azokra pedig nem, azokkal a következő fejezet foglalkozik.

4.4   Quantum walks on Class 2 or non-regular graphs

4.4.1   Class 2

For graphs in class 2, we can not use the same coin-position notation, we have to use the arc notation and replace the simple graph by an associated symmetric digraph, whose underlying graph is the original graph. In this case, the walker steps on the arcs of the digraph. After those considerations, we are able to define formally coined quantum walks.

4.4.2   Make the graph regular

Add loops (and hope they don't ruin the edge chromatic number?).

4.4.3   Szegedy walk

Walk on the edges.

Chapter 5

Szimulátor szoftver

Itt nagyon kiemelni, hogy önlab1-ben csak a klasszikus bolyongás sikerült + az egyenesen a kvantumos, most meg már tudok bármilyen regulárison bolyongani.

5.1   Tervezési megfontolások

Az első fázisban a szomszédossági mátrixot generáltam le teljesen. Ez nagyon sok memóriát használt, az unitér mátrix meg mégtöbbet, nagyon rosszul skálázódott. Lecseréltem spare mátrixokra, az már egy fokkal jobb.

Mérés összehasonlítás: Mátrix, Spare mátrix, Orákulum grafikon.

Ezután kipróbáltam a spare mátrixokat, de azok is picit sokak voltak, ugyan a gráfban sok a 0, de azért így is nagyon skálázódott.

A legjobb megoldás az úgynevezett gráf orákulum volt. A gráf orákulum lényege, hogy nem tároljuk a szomszédossági mátrixot, helyette biztosítunk egy függvényt, mely on-the-fly a gráf adott csúcsindexét odaadva neki visszaadja hogy annak mik a szomszédai. A coin kis méretű, azt lehet tárolni (max fokszám * max fokszám), ezért azzal nem bajlódtam, bár éppen lehetne.

Emellett a Stratégia tervezési mintát használatam.

TODO: https://en.wikipedia.org/wiki/Strategy_pattern, de valamilyen Design Patternes könyvből inkább.

Chapter 6

Architektúra

Ezt az önlabomból hoztam át, maga a keret nem sokat változott, az változott nagyot hogy a kvantumos szimulátor is működik már d reguláris gráfokra.

Ebben a fejezetben bemutatom az elkészült keretrendszert.

Programozási nyelvnek a Python 3-mat választottam. Ennek oka az, hogy nagyon sok data science-el kapcsolatos modulja van, mely nagyban megkönnyíti a különböző matematikai, algoritmuselméleti problémák feltárását, könnyen iterálhatunk a különböző prototípusokon. Emellett a szintaxisa rövid, tömör, lényegretörő programkódok megírását teszi lehetővé.

A forráskód három nagy részre bomlik:

- Gráfmodellek

- Szimulátorok

- Futtatás, konfiguráció, eredmények ábragenerátora

6.1 Gráfmodellek

A félév során sokféle gráfon futtattam szimulációs kísérleteket, melyek során több problémába ütköztem. Kezdetben úgy oldottam meg a szimulációkat, hogy a célgráfok szomszédossági mátrixait generáltam le, egyben a memóriában tartva azokat és a lépések során a megfelelő csúcshoz tartozó sorokat lekérdezve.

Ezzel a módszerrel több probléma is jelentkezett. Az első gondot az okozta, hogy a szomszédossági mátrix mérete a csúcsszám négyzetével arányos, ezért pár ezer csúcsú gráfot már nem tudtam a memóriában tartva szimulálni. A második probléma pedig az volt, hogy a szomszédossági mátrixos ábrázolás nagyon távol esett az emberi szempontból természetes ábrázolástól. A kvantumbolyongásos szimulációkat tipikusan nem véletlenszerű gráfokon szokták kipróbálni, hanem jól ismert struktúrával rendelkező gráfokon. Ilyen gráfok például a „súlyzók" vagy a ragasztott bináris fák.
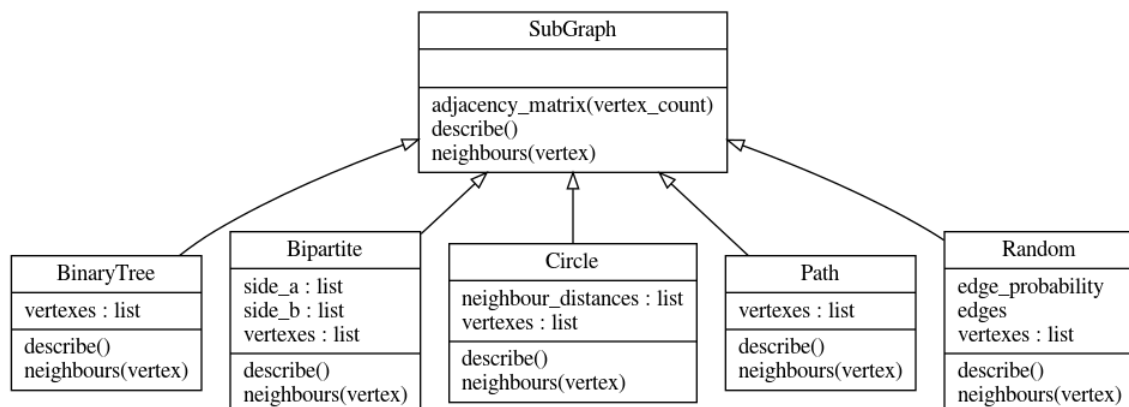
A súlyzó gráf két egyforma méretű kört tartalmaz, mindkét körből kiválasztva $k-k$ darab csúcsot, melyek teljes páros gráfot alkotnak (a súlyzó középső rúdját). A körökben pedig nem csak az egymás melletti csúcsok között fut él, hanem futhat él minden i. csúcs között is. A ragasztott bináris fában két egyforma méretű teljes bináris fa leveleit szembefordítjuk és a két oldali levelek közé egy teljes páros gráfot készítünk.

A fenti leírásból látható, hogy az ember számára természetes leírás a gráfokat ismert részgráfok kompozitjaként adja meg. A félév során olyan architektúrát alakítottam ki a szimulációkhoz, mely ezt a szemléletet támogatja. A szomszédossági mátrixos tárolási mód helyett pedig a szomszédossági orákulum megközelítést használva nagyban csökkent az alkalmazás memóriaigénye. Ennek a megközelítésnek a lényege, hogy az ismert struktúrájú gráfokra nem tárolok a memóriában szomszédossági információt, helyette biztosítok egy függvényt, amely a bemeneti paraméterként kapott csúcsindexre kiszámolja a vele szomszédos csúcsok indexeit.

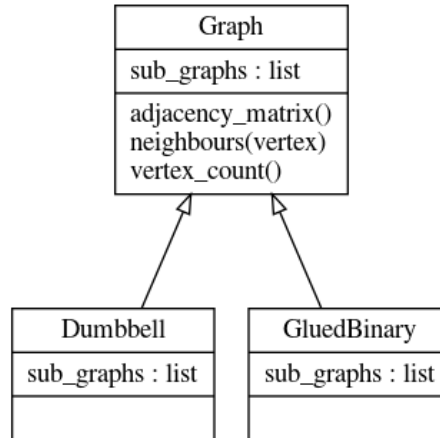A félév során a következő nevesített részgráfok szomszédossági orákulumját implementáltam:

- BinaryTree

- Bipartite

- Circle

- Path

- Random

Új: Hiperkocka

Ezen részgráfokból épülnek fel az alábbi kompozit gráfok:

- Dumbbell

- GluedBinary

Graph

sub_graphs : list

adjacency_matrix()
neighbours(vertex)
vertex_count()

Dumbbell

sub_graphs : list

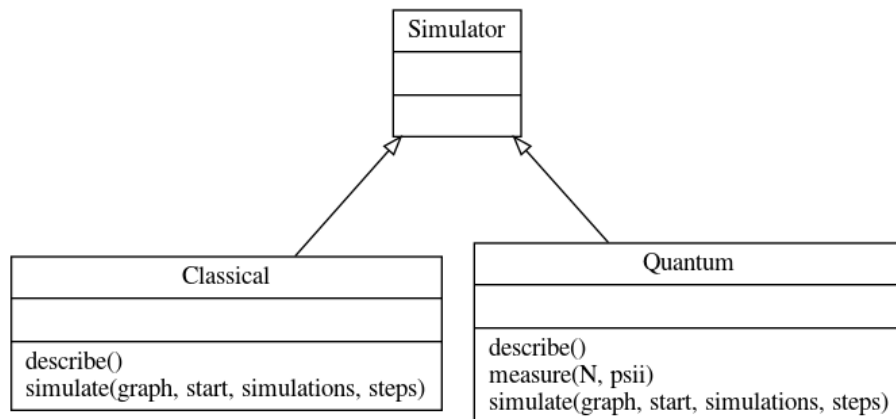GluedBinary

sub_graphs : list

6.2 Szimulátorok

A szimulátor osztályok közül a klasszikus tetszőleges kompozit gráfot tud fogadni.

Itt fontos átírni, hogy a kvantumszimulátor mostmár d-regulárist is tud!

A kvantumszimulátor jelenleg a kvantumbolyongás egy speciális esetét, az egyenesen való bolyongást képes kezelni, mely a 2-regularitása miatt egyszerűbben implementálható. Hosszú távú cél a k-reguláris, illetve az általános gráfokra kiterjeszteni ezt a szimulátort.

Simulator

Classical

describe()
simulate(graph, start, simulations, steps)

Quantum

describe()
measure(N, psii)
simulate(graph, start, simulations, steps)

6.3 Futtatás, konfiguráció, eredmények ábragenerátora

Itt van pár újdonság, pl. sajátértékek kiírása, stb.

A fenti osztályok segítségével egy olyan keretrendszert alakítottam ki, melyben nagyon gyorsan fel lehet 1-1 futtatást konfigurálni. A futtatás eredményeit egy összesített Latex dokumentumba gyűjti a program. Ez tartalmazza a beadott gráf részgráfjainak nevesített típusát, szomszédsági mátrixait, illetve a teljes gráf szomszédsági mátrixát, valamint a szimulációk eloszlási eredményeit. A következő fejezetben több ilyen ábrát is bemutatok.

Chapter 7

Other Quantum Walk Packages

- A kódok megnéztem: C++-ban írt, sok memóriamenedzselés és konkrét gráfok (csak 1d és 2d rács), sztochasztikus szimuláció, pythonban írt és bonyolult. - Van általános?

Ennek a fejezetnek az lenne a célja, hogy megnéztem sok más programot és egyik se olyan jó mint az enyém, talán kevesebb szöveg kellene és arra fókuszálni hogy miért nem jók ezek, nem hogy pont mit csinálnak, de csak beraktam amit a könyv ír róluk. Általánosan mindegyik bonyolult, C++-os, nem szép tervezési mintájú, vagy csak speciális gráfokra működik (pl. csak grid).

[5] gives a detailed description of currently available quantum walk packages.

7.1   QWalk

QWalk aims to simulate the coined quantum walk dynamics on one- and twodimensional lattices. The package is written in C and uses Gnuplot to plot the probability distribution. The user can choose the coin and the initial condition. There is an option to simulate decoherent dynamics based on broken links—also known as percolation [251, 280]. The links of the lattice can be broken at random at each step or the user can specify which edges will be missing during the evolution. QWalk allows the user to simulate quantum walks on any graph that is a subgraph of the two-dimensional lattice. Some plots in this section were made using QWalk. The package can be obtained from Mendeley: `https://data.mendeley.com/datasets/b93c846dhs/1,`

Ez ugyanaz: `https://github.com/QWalk/mainline`?

7.2   QwViz

QwViz aims at plotting graphics for visualizing the probability distribution of quantum walks on graphs. The package is written in C and uses OpenGL to generate two- or three-dimensional graphics. The user must enter the adjacency matrix of the graph, and the package simulates the dynamics of the coined model to calculate the probability distribution. By default, the walker starts at vertex 1 with the coin in uniform superposition. The initial location can be changed by the user. It is possible to specify marked vertices, which tell the package to use quantum-walk-based search procedures starting from a uniform superposition of all vertices and using the Grover coin on the

unmarked vertices and $(-I)$ on the marked vertices. The package can be obtained from
`https://data.mendeley.com/datasets/kj58zkdmt7/1`.

7.3 PyCTQW

PyCTQW aims to simulate large multi-particle continuous-time quantum walks using object-oriented Python and Fortran. The package takes advantage of modern HPC systems and runs using distributed memory. There are tools to visualize the probability distribution and tools for data analysis. The package can be obtained from `https://data.mendeley.com/datasets/wy29d79ds4/1`.

7.4 Hiperwalk

Hiperwalk (high-performance quantum walk) aims to simulate the quantum walk dynamics using high-performance computing (HPC). Hiperwalk uses OpenCL to run in parallel on accelerator cards, multicore CPU, or GPGPU. It is not required any knowledge of parallel programming, but the installation of the package dependencies is tricky, in special, OpenCL. Besides, Hiperwalk uses the Neblina programming language. In the CUSTOM option, the input is an initial state $|\Psi_0\rangle$ and a unitary operator U, which must be stored in two different files (only nonzero entries in order to take advantage of sparsity). Hiperwalk calculates $U^t |\Psi_0\rangle$ for integer t using HPC and saves the output in a file. There are extra commands for the coined and staggered models. The Hiperwalk manual has a detailed description of the installation steps and some examples of applications.

7.5 QSWalk

QSWalk is a Mathematica package that aims to simulate the time evolution of quantum stochastic walks on directed weighted graphs. The quantum stochastic walk is a generalization of the continuous-time quantum walk that includes the incoherent dynamics. The dynamic uses the Lindblad formalism for open quantum systems using density matrices. The package can be obtained from the Computer Physics Communications library.

7.6 QSWalk.jl

QSWalk.jl is a Julia package that aims to simulate the time evolution of quantum stochastic walks on directed weighted graphs. The authors claim that is faster than QSWalk when used in large networks. Besides, it can be used for nonmoralizing evolution, which means that the evolution takes place on a directed acyclic graph and does not change to an evolution on the corresponding moral graph. The package can be downloaded from GitHub `https://github.com/iitis/QSWalk.jl`.

Chapter 8

Results

8.1 Walk on line

8.2 Walk on Grid

8.3 Walk on Hypercube

Chapter 9

Conclusion

- Probléma a félév során: Python küzdés: referenciákat furán kezeli, fura szintaxis (C#-hoz képest), nehéz kezelni, indentáció a zárójelek helyett weird. Emiatt mátrixosan, matlabba átírtam (python fura referenciális hibái nem lesznek, a spare mátrixokkal egész jó a memhasználat).

- Popovics Viktória és Florin Christea érdeklődött, megtalálta.

- Python is segmented, everything is a module, no concise structure and interoperability.

- Youtube videók: nemk.in/quantum, illetve kód: github.com/nemkin/quantum.

Quantum: Joking about not understanding quantum. Studying the electromagnetic equations with physics teacher: You are looking at it right now. Simulation = look at it.

Acknowledgements

Ez nem kötelező, akár törölhető is. Ha a szerző szükségét érzi, itt lehet köszönetet nyilvánítani azoknak, akik hozzájárultak munkájukkal ahhoz, hogy a hallgató a szakdolgozatban vagy diplomamunkában leírt feladatokat sikeresen elvégezze. A konzulensnek való köszönetnyilvánítás sem kötelező, a konzulensnek hivatalosan is dolga, hogy a hallgatót konzultálja.

Bibliography

[1] Ferenc Balazs and Sandor Imre. *Quantum computing and communications: an engineering approach.* Wiley, Hoboken, N.J., 2013. ISBN 9781118725474.

[2] Mika Hirvensalo. *Quantum computing.* Springer, Berlin; New York, 2001. ISBN 9783540407041.

[3] J Kempe. Quantum random walks: An introductory overview. *Contemporary Physics*, 44(4):307–327, 2003.

[4] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis.* Cambridge University Press, New York, 2005. ISBN 9780521835404.

[5] Renato Portugal. *Quantum Walks and Search Algorithms.* Quantum Science and Technology. Springer, Cham, 2nd ed. 2018 edition, 2018. ISBN 9783319978130.

[6] Feng Xia, Jiaying Liu, Hansong Nie, Yonghao Fu, Liangtian Wan, and Xiangjie Kong. Random walks: A review of algorithms and applications. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 4(2):95–107, 2020.
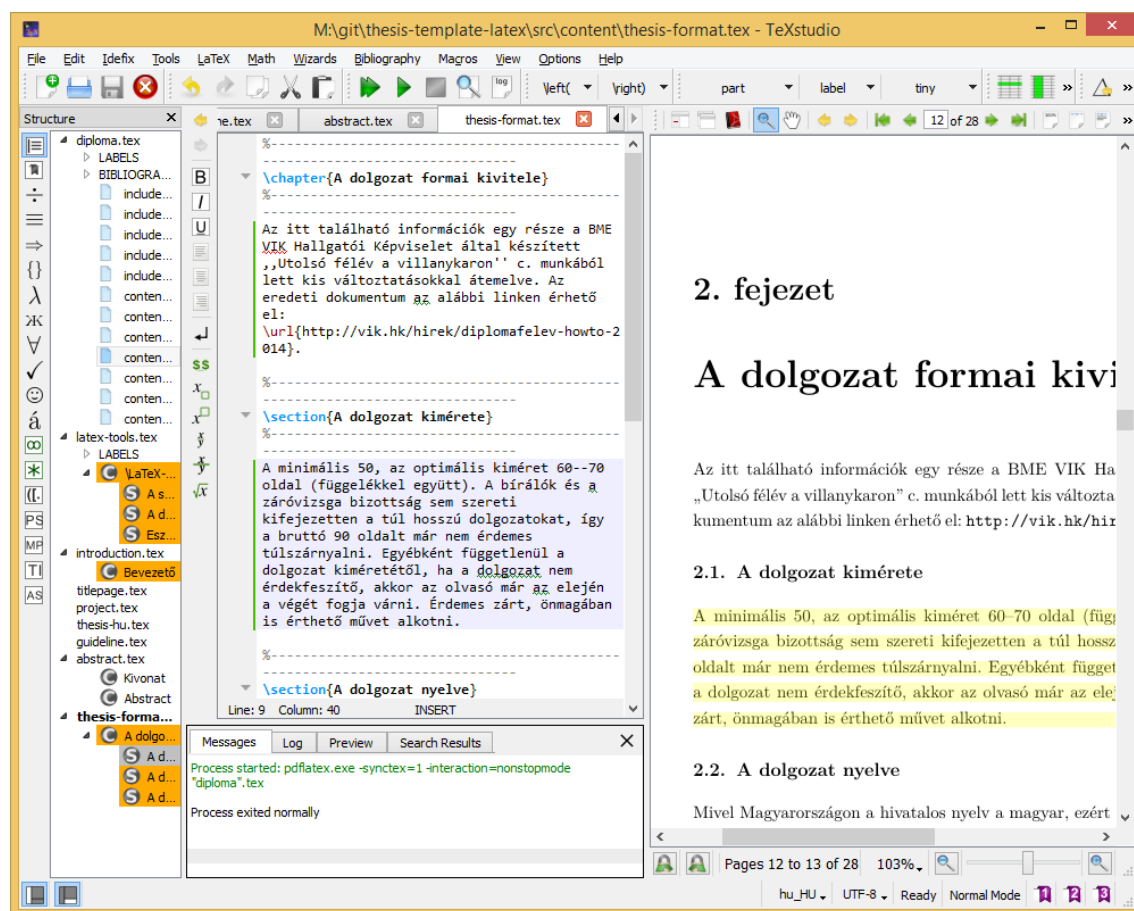
Appendix

A.1 Első függelék

Figure A.1.1: *A TeXstudio LaTeX-szerkesztő.*

A.2 Második függelék

A Pitagorasz-tételből levezetve

$$c^2 = a^2 + b^2 = 42.$$ (A.2.1)

$$c^2 = a^2 + b^2 = 42.$$ (A.2.1)