



M Ű E G Y E T E M 1 7 8 2

Budapesti Műszaki és Gazdaságtudományi Egyetem
Természettudományi Kar

OTDK DOLGOZAT

Koherens és inkohere ns hibák vizsgálata ismétléses kóddal kvantumszámítógépeken

Készítette:

Halmos Balázs Paszkál

Budapest II. Kerületi

II. Rákóczi Ferenc Gimnázium (12. évf.)

Konzulens:

Dr. Asbóth János

Egyetemi docens

BME Elméleti Fizika Tanszék

Absztrakt

A kvantumszámítógépeken, még ha nem is végzünk műveleteket, a kvantumbitek nincsenek biztonságban a zajtól, hibáktól, ezek ellen védekeznünk kell. Kutatásomban kétféle hibamodellt hasonlítok össze - nem tökéletes kiolvasások esetén - egy olyan kvantumszámítógép-memóriában, amely ismétléses kódot használ az információ védelmére. Az első modell, hogy az adatbiteken véletlen Pauli-X hibák történhetnek egy bizonyos valószínűséggel, a második, hogy az összes bitre egy koherens forgatás hat. Kutatásomban áramkörüi szintű numerikus szimulációval, a python qiskit szoftverfejlesztő készletével készített programommal, megmutatom, hogy a két hibamodell ugyanakkora veszélyt jelent az eltárolt információra, a kód méretének növelése csökkenti, az ismétlések számának növelése azonban növeli a hűtlenséget.

Tartalomjegyzék

1. Bevezetés a kvantuminformatikába	2
1.1. A kvantumbitek matematikai leírása	2
1.2. A kvantumáramkörök	4
1.3. A kvantumkapuk	4
1.4. A hibamodellek	6
1.5. Kvantumos hibajavítás	7
1.6. A hibajavítás hatékonyságának a meghatározása	9
2. A szimulációs program	10
3. A szimuláció eredményei	11
3.1. A két hibamodell ugyanakkora veszélyt jelent az eltárolt információra	11
3.2. A hibák és a kód méretének növelésével, a hibajavítás után, egyre többféle forgatási szög jelenik meg koherens hibák esetén	12
3.3. A kód méretének növelésével csökken a hűtlenség, bár néha nem szignifikánsan	17
3.4. A körök számának növelése növeli a hűtlenséget	19
4. A szimuláció futtatása	21
4.1. A program	21
4.2. A futtatási idők	21
5. Konklúzió	22

1. Bevezetés a kvantuminformatikába

A kvantumszámítógépek kutatása élénk napjainkban, hiszen segítségükkel olyan problémákat oldhatunk meg, amelyek klasszikus számítógéppel nem megoldhatóak belátható időn belül. Ilyen probléma például a hatalmas összetett számok prímtényezőkre felbontása. Ez azért nagyon fontos, mert jelenleg az egyik legnépszerűbb internetes titkosítási módszer az ezt a feladatot kihasználó RSA algoritmuson [7] alapul. A Shor-algoritmus [8] segítségével ez a probléma polinomiális idő alatt ($O(d^3)$, ahol d a számjegyek száma = gyorsan) megoldható a korábbi exponenciális idő ($O(\exp(d^{1/3}))$ = nagyon lassú) helyett, viszont jelenleg nincsenek olyan jó kvantumszámítógépek, amelyekkel ez kivitelezhető lenne - több és jobb minőségű kvantumbitekre van szükségünk. A Shor-algoritmus erőforrásigényéről a [2] cikkben tájékozódhatunk. A kvantumbitek minőségét szerencsére kvantum hibajavítással növelhetjük, vagyis a kvantumcsipeken lévő több zajos fizikai kvantumbit és hibajavítás segítségével lehet csinálni kevesebb jobb minőségű, úgynevezett logikai kvantumbitet, az [5] cikkben a 38. oldalon össze van gyűjtve több hibajavító eljárás.

1.1. A kvantumbitek matematikai leírása

A kvantumszámítógépek hatalmas előnyét a klasszikus számítógépekkel szemben a kvantumbitek adják, amik két kvantummechanikai jelenséget használnak ki: a *szuperpozíciót* és az *összefonódást*. A *szuperpozíció* azt jelenti, hogy a bitek nemcsak a 0 és az 1 állapotban lehetnek, hanem a kettő egy köztes állapotában is. Az *összefonódás* lehetővé teszi, hogy a kvantumbitek együttesen több kvantumállapotban lehetnek, mint bármilyen klasszikus vagy akár analóg számítógépen.

Ezeket a jelenségeket kihasználva készíthetünk a jelenlegieknél hatékonyabb algoritmusokat és számítógépeket. A kvantumbitek leírására új jelöléseket vezethetünk be.

Dirac-jelölés vektorokra [6]

- A *ket* egy oszlopvektort jelöl, pl.:

$$|\Psi\rangle = \begin{pmatrix} a \\ b \end{pmatrix}, \text{ ahol } a, b \in \mathbb{C}, \quad |a|^2 + |b|^2 = 1. \quad (1)$$

- A *bra* egy sorvektort jelöl, a *ket*ben lévő vektor *adjungáltját*. Ezt úgy kaphatjuk meg, hogy transzponáljuk a vektort és az értékeknek vesszük a komplex konjugáltját (a^*, b^*), pl.:

$$\langle\Psi| = (a^* \quad b^*), \text{ ahol } a, b \in \mathbb{C}, \quad |a|^2 + |b|^2 = 1. \quad (2)$$

- Az *belső szorzat* (*inner product*) a vektorok skaláris szorzatának az általánosítása. Két vektor *belső szorzata* egy (komplex)szám. A jelölése és a definíciója:

$$\langle a|b\rangle = (a_1^* \quad a_2^* \quad \dots \quad a_n^*) \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} = a_1^* b_1 + a_2^* b_2 + \dots + a_n^* b_n. \quad (3)$$

- *Külső szorzatot* (*outer product*) akkor kapunk ha megcseréljük a *bra* és a *ket* sorrendjét. Ekkor két vektor *külső szorzata* egy mátrix lesz.

$$|a\rangle\langle b| = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} (b_1^* \quad b_2^* \quad \dots \quad b_n^*) = \begin{pmatrix} a_1 b_1^* & a_1 b_2^* & \dots & a_1 b_n^* \\ a_2 b_1^* & a_2 b_2^* & \dots & a_2 b_n^* \\ \vdots & & \ddots & \vdots \\ a_n b_1^* & a_n b_2^* & \dots & a_n b_n^* \end{pmatrix}. \quad (4)$$

A *tenzorszorzat* egy művelet. Kiszámolható vektoroknak és mátrixoknak is a tenzorszorzata.

Vektorok tenzorszorzata növeli a vektor dimenzióit: két kétdimenziós vektor tenzorszorzata egy négydimenziós vektor:

$$|a\rangle \otimes |b\rangle = |ab\rangle = \begin{pmatrix} a_1 \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \\ a_2 \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} a_1 b_1 \\ a_1 b_2 \\ a_2 b_1 \\ a_2 b_2 \end{pmatrix}. \quad (5)$$

Mátrixok tenzorszorzata is növeli a mátrix dimenzióit:

$$A \otimes B = \begin{pmatrix} a_{11}B & \dots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \dots & a_{mn}B \end{pmatrix}. \quad (6)$$

A kvantumállapotok

A kvantumállapotokat 2 dimenziós komplex vektorokkal írhatjuk fel, és görög betűkkel jelölhetjük: pl.: $|\Psi\rangle$. A kvantumállapotokat jelölő vektorokat mindig 1-re normálják.

A **számítási bázis** két bázisvektora a $|0\rangle$ és az $|1\rangle$ vektorok. Ez a két vektor ortonormált, vagyis 1-re normáltak és ortogonálisak, vagyis *belső szorzatuk* 0:

$$|0\rangle \equiv \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad |1\rangle \equiv \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \quad (7)$$

$$\langle 0|1\rangle = \langle 1|0\rangle = 0, \quad (8)$$

$$\langle 0|0\rangle = \langle 1|1\rangle = 1. \quad (9)$$

A *szuperpozíció* egy kvantummechanikai jelenség, ez teszi lehetővé, hogy a kvantumbitek végtelen sok állapotban lehessenek. Ugyanis azok nem csak a $|0\rangle$ és $|1\rangle$ bázisállapotokban lehetnek, hanem ezek bármely szuperpozíciójában:

$$|\Psi\rangle = \alpha |0\rangle + \beta |1\rangle, \quad \alpha, \beta \in \mathbb{C}, \quad |\alpha|^2 + |\beta|^2 = 1. \quad (10)$$

Ezt felírhatjuk vektoros jelöléssel is a számítási bázisban:

$$|\Psi\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}. \quad (11)$$

Bizonyos állapotokat olyan gyakran használnak, hogy külön jelük van (a számítási bázis $|0\rangle / |1\rangle$ mellett). A $|+\rangle / |-\rangle$ bázist Hadamard bázisnak hívják:

$$|+\rangle = \frac{1}{\sqrt{2}} (|0\rangle + |1\rangle), \quad |-\rangle = \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle). \quad (12)$$

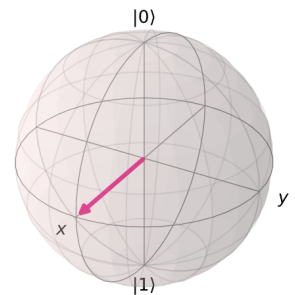
Bloch-gömb

Az egybites kvantumállapotokat a Bloch gömbön ábrázolhatjuk. Ez egy egységsugarú gömb aminek az „északi pólusán” van a $|0\rangle$ állapot, a "déli póluson" az $|1\rangle$. Egy általános kvantumállapotot ($|\Psi\rangle = \alpha |0\rangle + \beta |1\rangle$), ahol: $\alpha, \beta \in \mathbb{C}$, $\sqrt{|\alpha|^2 + |\beta|^2} = 1$) felírhatunk az alábbi alakban is:

$$|\Psi\rangle = e^{i\chi} \left(\cos \frac{\Theta}{2} |0\rangle + e^{i\phi} \sin \frac{\Theta}{2} |1\rangle \right), \quad (13)$$

ahol $\phi \in [0; 2\pi[$ és $\Theta \in [0; \pi]$. Itt a $\chi \in [0, 2\pi]$ globális fázis értéke nem befolyásol semmilyen mérést, ezért elhagyható.

Ezek megadják az ábrázolandó kvantumállapot koordinátáit a Bloch gömbön. A Bloch gömb két átellenes oldalán lévő vektor mindig ortonormált.



1. ábra. A Bloch gömb. Itt a $|+\rangle$ állapot látszik: $\Theta = \pi/2$, $\phi = 0$

A sűrűségmátrix

Gyakran szükség van olyan kvantumállapottal való számolásra, amiről csak statisztikus információnk van, azaz az állapot p_j valószínűséggel $|\Psi_j\rangle$, ahol j valamilyen (diszkrét vagy folytonos) index. Ilyen esetekben hasznos a sűrűségmátrix,

$$\hat{\rho} = \sum_j p_j |\Psi_j\rangle \langle \Psi_j|, \quad (14)$$

ami egy ún. kevert állapotot jellemez. A sűrűségmátrixhoz egy pont tartozik a Bloch-gömb belsejében, ami a $|\Psi_j\rangle$ állapotokhoz tartozó helyvektorok konvex kombinációja, p_j súlyokkal.

A kvantumbitek mérése

Ugyan a szuperpozíció végtelen sok kvantumállapotot tesz lehetővé, hogy hozzáférjünk a tárolt információhoz meg kell mérnünk a kvantumbitét. Viszont a fizika törvényei szerint, a bit megmérésevel az "beugrik" a mért értéknek megfelelő állapotba, és így tipikusan elveszítünk a mérés előtti állapotról sok információt. Továbbá egy kvantumbitből csak egy (klasszikus)bitnyi információt lehet kinyerni mérés segítségével. Ezért a mérést sokszor kell elvégezni, és így statisztikailag lehet következtetni az eltárolt információra.

A méréshez mindig szükség van egy ortonormált bázisra, amelyben a mérést el lehet végezni, ez gyakran (bár nem kizárólagosan) a számítási bázis. A mérés után a kvantumállapot beugrik az egyik mérési bázisba, vagyis egy mérés után több plusz információt nem lehet kinyerni a kvantumbitből. Összesen az az egy (klasszikus)bitnyi információ van meg, hogy melyik mérési bázisba ugrott az állapot.

Egy általános szuperpozíció $|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle$ például a számítási bázisban történő mérés során várható értékben $|\alpha|^2$ valószínűséggel a $|0\rangle$, $|\beta|^2$ valószínűséggel az $|1\rangle$ állapotba fog beugrani.

Az összefonódás

Több kvantumbites rendszerekben a kvantumbitek vektorai a tenzorszorzat segítségével egy többdimenziós vektortérnek lesznek az elemei. A kvantumos összefonódás az, ha a kvantumállapotot nem lehet felírni az egyes biteket jelölő vektorok tenzorszorzataként. Például egy 2 kvantumbites rendszerben a Bell-állapotok összefonódottak:

$$|\Phi^\pm\rangle = \frac{1}{\sqrt{2}}(|00\rangle \pm |11\rangle), \quad |\Psi^\pm\rangle = \frac{1}{\sqrt{2}}(|01\rangle \pm |10\rangle). \quad (15)$$

Ezek egyikét sem lehet felírni 2 kvantumbit tenzorszorzataként.

1.2. A kvantumáramkörök

A klasszikus számítógépek matematikai leírására több modellt is használhatunk, például a Turing-gép vagy a lambda kalkulus. Továbbá az áramkörmodell, ebben a modellben a számításokat (matematikai)logikai kapuk segítségével végezzük el. Minden kapu bemenete egy vagy több bit (amelynek az értéke 0 vagy 1), a kimenete egy bit. Egybites kapu például a *nem* (NOT) kapu. Kétbites az *és* (AND), a *nem és* (NAND), a *vagy* (OR), a *nem vagy* (NOR), és a *kizáró vagy* (XOR). Mindegyik egy bizonyos szabály alapján adja a kimenetet a bemenet(ek) alapján, ezt általában igazságtáblával adjuk meg.

A kvantumszámítógépek programozásában jelenleg elsősorban az áramkörmodellt használjuk, a kvantumkapuk azonban eltérnek a klasszikus kapuktól. Ugyanis egy kvantumbit állapota nem csak 0 vagy 1 lehet, mint egy klasszikus bitnek, hanem végtelen állapota lehet, hasonlóan az analóg számítógépekhez. Az összefonódásnak azonban nincs klasszikus megfelelője. Továbbá a kvantumrendszerek időfejlődése (a mérés előttig) mindig visszafordítható, vagyis ha a kapuk inverzét fordított sorrendben végrehajtjuk, úgy az eredeti állapotot kapjuk vissza, vagyis reverzibilis kapukat kell használni (a klasszikus kapuknak van reverzibilis megfelelője, de azok eltérnek a leggyakrabban használt klasszikus kapuktól).

1.3. A kvantumkapuk

A kvantumkapukat mátrixokkal ábrázolhatjuk, és a kapu hatását az állapoton mátrixszorzat segítségével írhatjuk le. Például az egyik legegyszerűbb kvantumkapu az X kapu (1. táblázat),

aminek a hatása:

$$X|0\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} = |1\rangle, \tag{16}$$

$$X|1\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} = |0\rangle. \tag{17}$$

Ugyanezt a számolást elvégezhetjük a Dirac-jelölés segítségével is:

$$X|0\rangle = (|0\rangle\langle 1| + |1\rangle\langle 0|)|0\rangle = |0\rangle\langle 1|0\rangle + |1\rangle\langle 0|0\rangle = |1\rangle, \tag{18}$$

$$X|1\rangle = (|0\rangle\langle 1| + |1\rangle\langle 0|)|1\rangle = |0\rangle\langle 1|1\rangle + |1\rangle\langle 0|1\rangle = |0\rangle. \tag{19}$$

Ha ismerjük egy kapu hatását egy bázisban, ami két ortonormált bázisállapotból áll, akkor - a linearitás miatt - megadható a kapu hatása egy szuperpozícióban lévő általános állapoton:

$$X(\alpha|0\rangle + \beta|1\rangle) = \beta|0\rangle + \alpha|1\rangle. \tag{20}$$

Néhány több kvantumbites kapuk egybites kapuk tenzorszorzata segítségével állítható elő, a CNOT kapu (21) azonban nem állítható elő így.

Egy kvantumbites kapuk

Jel	Mátrix	Qiskit ábra	Hatás a Bloch-gömbön
X	$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = 0\rangle\langle 1 + 1\rangle\langle 0 $	q -  -	π -vel forgatás az x -tengely körül
Y	$\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} = -i 0\rangle\langle 1 + i 1\rangle\langle 0 $	q -  -	π -vel forgatás az y -tengely körül
Z	$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} = 0\rangle\langle 0 - 1\rangle\langle 1 $	q -  -	π -vel forgatás a z -tengely körül

1. táblázat. A Pauli-kapuk

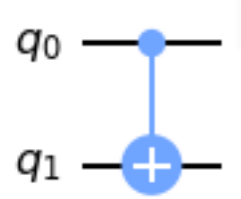
Jel	Mátrix	Qiskit ábra	Hatás a Bloch-gömbön
H [Hadamard]	$\frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$	q -  -	$ 0\rangle \leftrightarrow +\rangle, 1\rangle \leftrightarrow -\rangle$
$P(\phi)$	$\begin{pmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{pmatrix}$	q -  -	ϕ -vel forgatás az z -tengely körül
$R_x(\Theta)$	$e^{-i\Theta\hat{X}} = \begin{pmatrix} \cos\Theta & -i\sin\Theta \\ -i\sin\Theta & \cos\Theta \end{pmatrix}$	q -  -	Θ -val forgatás az x -tengely körül
I [Azonosság]	$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$	q -  -	Nem forgat a Bloch-gömbön
U [Általános]	$\begin{pmatrix} \cos\left(\frac{\Theta}{2}\right) & -e^{i\lambda}\sin\left(\frac{\Theta}{2}\right) \\ e^{i\phi}\sin\left(\frac{\Theta}{2}\right) & e^{i(\phi+\lambda)}\cos\left(\frac{\Theta}{2}\right) \end{pmatrix}$	q -  -	Általános forgatás a Bloch-gömbön

2. táblázat. Egyéb egybites kvantumkapuk

Két kvantumbites kapu a CNOT kapu

A kapu egy X kapuként viselkedik a cél biten, ha a kontroll bit az $|1\rangle$ állapotban van, és nem változtatja a cél bit állapotát, ha a kontroll bit a $|0\rangle$ állapotban van. Ezt ezzel a mátrix-szal lehet leírni:

$$\text{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad (21)$$



2. ábra. A CNOT kapu. Itt a q_0 a kontroll, a q_1 a cél qubit.

Ennek a kapunak kiemelt szerepe van, ugyanis ez tud több bitet összekapcsolni, vagyis az összefonódást létrehozni bitek között, ami sok kvantumalgoritmus előnyét adja, nem feltétlenül kell egy kvantumalgoritmushoz összefonódás, bár szinte mindig. Azok a kvantumalgoritmusok, ahol nincs összefonódás, hatékonyan szimulálhatók hagyományos számítógépen. Viszont érdekes módon statisztikus értelemben "kevés" összefonódás elég a kvantumszámítógépeknek, hogy hatékonyabbak legyenek a hagyományos számítógépeknél [9].

1.4. A hibamodellek

A kvantumszámítógépeken az eltárolt információra veszélyt jelent a dekoherencia, vagyis az eltárolt információ módosulása, esetleg eltűnése, amit a környezettel való (nem kívánt) kölcsönhatás okoz, ilyen lehet egy a kozmikus sugárzásból eredő részecske. A hibajavítás szimulációjához szükség van hibamodellekre, amik egyfajta hibának az egyszerűsített leírásai, aminek hatásait különböző paraméterek mellett lehet vizsgálni.

Koherens forgatások

A szimuláció minden körében egy kis Θ szögű forgatás történik a Bloch gömbön az x tengely körül (egy R_x kapuval) minden adatot tároló kvantumbiten [1]. Egy ilyen forgatás után az új állapot így alakul 1 kvantumbit esetén:

$$|\Psi\rangle = e^{i\Theta\hat{X}} |\Phi\rangle = \left(\cos \Theta + i \sin \Theta \hat{X} \right) |\Phi\rangle \quad (22)$$

Az állapot N kvantumbit esetén(, ahol $c = \cos \Theta$, $s = \sin \Theta$):

$$|\bar{\Psi}\rangle = e^{i\Theta\hat{X}_1} e^{i\Theta\hat{X}_2} \dots e^{i\Theta\hat{X}_N} |\bar{\Phi}\rangle = e^{i\Theta(\hat{X}_1 + \hat{X}_2 + \dots + \hat{X}_N)} |\bar{\Phi}\rangle = \quad (23)$$

$$|\bar{\Psi}\rangle = (c + is\hat{X}_1)(c + is\hat{X}_2) \dots (c + is\hat{X}_N) |\bar{\Phi}\rangle = \left[\prod_{k=1}^N (c + is\hat{X}_k) \right] |\bar{\Phi}\rangle, \quad (24)$$

ahol $|\bar{\Psi}\rangle$ az új kvantumállapot a forgatás után, $|\bar{\Phi}\rangle$ a kiindulási kvantumállapot, i az imaginárius egység ($i = \sqrt{-1}$), e az Euler-féle szám, Θ a forgatás szöge, $\hat{X}$ a Pauli X operátor (egy mátrix, a kalap a tetején, azt jelöli, hogy operátor).

Véletlen Pauli X hibák

A véletlen hibák esetén valamelyik adatbiten egy bizonyos p valószínűséggel egy Pauli X hiba hat, ezt a sűrűségoperátor segítségével lehet felírni matematikailag [1].

1 kvantumbit esetén:

$$\hat{\rho}^* = (1-p)\hat{\rho} + p\hat{X}\hat{\rho}\hat{X}, \quad (25)$$

vagyis $(1-p)$ valószínűséggel nem változik az állapot, p valószínűséggel egy X hiba hat a kvantumbiten.

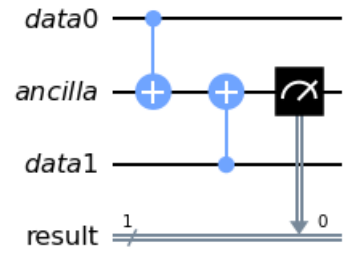
1.5. Kvantumos hibajavítás

A kvantumszámítógépeken, még ha nem is történik művelet, az eltárolt kvantuminformáció ki van téve a zajnak, hibáknak (amiket itt véletlen X hibákkal és koherens forgatásokkal modellezek. Más hibamodell is lehetséges például X, Y és Z hiba is hathat egyenlő valószínűséggel, ez a *depolarizáció*), ami ellen védekezni kell. Erre az egyik legegyszerűbb módszer az *ismétléses kód* [6] használata, ez azon alapul, hogy az információt N qubiten tároljuk egy helyett. Például a $|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle$ állapot 3 kvantumbiten bekódolva a:

$$|\bar{\Psi}\rangle = \alpha|000\rangle + \beta|111\rangle, \text{ ahol } \alpha, \beta \in \mathbb{C}, \sqrt{|\alpha|^2 + |\beta|^2} = 1 \quad (26)$$

alakban írható fel. Időnként vizsgáljuk, hogy történt-e hiba az egyik biten, ezt úgy állapítjuk meg, hogy mivel N páratlan, többségi szavazást tartunk az információról. Így ha kis valószínűséggel történik hiba, az javítható.

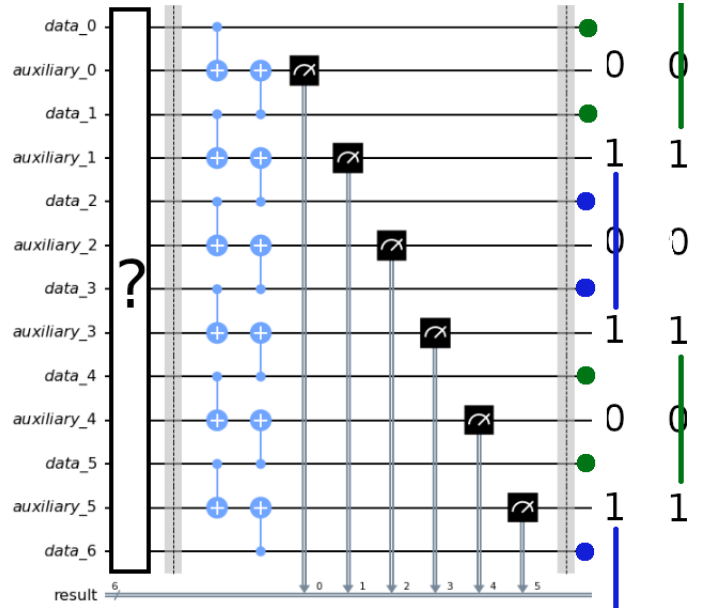
Viszont a kvantummechanika törvényei szerint nem érdemes számítás közben egyszerűen megmérni a kvantumbitek állapotát, mivel ez megváltoztatná őket (vagyis valószínűleg tönkretenné a számítást), szerencsére erre nincs is szükség. Ugyanis van egy módszer arra, hogy megvizsgáljuk két qubit paritását (egyenlőségét) anélkül, hogy bármi információt szereznénk a bitekben tárolt információról. Ez *ancilla* (segéd) qubitekkel és CNOT kapukkal oldható meg. Az áramkör ehhez a 3. ábrán látható. A kvantumbitek alapból a $|0\rangle$ vannak beállítva, ha nem történik hiba, a CNOT kapuk triviálisan hatnak az ancillán (nem változtatják). Ha mindkét data biten történik egy-egy X hiba (zaj), akkor mindkét CNOT kapu egy-egy X-szel fog hatni az ancillán, aminek a következménye az hogy annak az állapota végül a kiindulási $|0\rangle$ lesz. Amennyiben azonban csak a *data0* vagy csak a *data1* biteken történik egy X hiba, akkor csak az egyik CNOT fog X-szel hatni az ancillán, így az végül az $|1\rangle$ állapotba kerül. Az ancilla bitet kimérhetjük, mivel azzal nem szeretnénk a továbbiakban műveletet végezni. Tehát az ancillamérés alapján meg lehet határozni két qubit paritását: ha a mérés eredménye 0, a két kubit egyenlő, ha a mérés eredménye 1, akkor a két qubit eltérő információt tartalmaz. A hibák javításához azonban legalább 3 kvantumbitre van szükség (hogy tudjunk többségi szavazást tartani, hogy melyik állapota változhatott meg egy hiba miatt).



3. ábra. A paritásvizsgáló áramkör. Itt a *data0* és a *data1* kvantumbiteken vannak az adatok. Az *ancilla* qubit határozza meg a paritást. A *result* egy klasszikus bit, abba kerül a mérés eredménye.

Hibajavítás hibátlan kiolvasás esetén

Itt elég egy többségi szavazást tartani a mérési eredmények alapján. A 4. ábrán egy $N=7$ -es ismétléses kód kvantumáramkörét láthatjuk. Arra vagyunk kíváncsiak, hogy a ?-lel letakart területen (az adattároló (data) kvantumbiteken) történt-e X hiba, ehhez csak az ancillamérések eredményét használhatjuk fel (az ábra jobb oldalán a számok, a 0 azt jelenti, hogy a mellette lévő 2 kvantumbit paritása megegyezik, az 1 azt jelenti, hogy a mellette lévő két kvantumbit paritása különböző). Ebből következik a hibajavító algoritmus: az egyeseket "össze kell kötni" egymással, vagy a kód szélével. Kettő ilyen összekötés lehetséges: késsel és zölddel jelölve az ábrán. Az összekötés azt jelenti, hogy a belekötött kvantumbiteken egy X hiba történt (a vízszintes vonalak végén lévő körök). A kétféle összekötésből ezután azt választjuk ki, amelyik kevesebb hibát jósol, itt a kék összekötést. Ebből következik, hogy az algoritmus egy ilyen ancillamérés során a data 2, 3, és 6 kvantumbiteken jelez X hibákat. Ezt az módszert MWPM (Minimum-Weight Perfect Matching, Legkisebb Súlyú Teljes Párosítás) problémának hívják.



4. ábra. Példa a hibajavításra. Hibátlan ancillamérések, $N=7$ A kék összekötés jelez kevesebb hibát, vagyis az algoritmus a data 2,3,6 kvantumbiteken jelez X hibát.

A koherens hibáknál szindrómamérések és a hibajavítás egy globális forgatást ad a kvantumállapothoz.

Ha több információt szeretnénk megtudni a kvantumrendszer állapotáról, akkor megvizsgálhatjuk, hogy az hogyan változik a szindrómamérések és a hibajavítás elvégzése után. Ehhez nézzünk egy példát egy $N=3$ qubites rendszerben.

Az eredeti állapot a (26) egyenlet alapján:

$$|\bar{\Phi}\rangle = \alpha |000\rangle + \beta |111\rangle, \text{ ahol } \alpha, \beta \in \mathbb{C}, \sqrt{|\alpha|^2 + |\beta|^2} = 1. \quad (27)$$

Az állapot a koherens hiba után, az (24) egyenlet alapján (, ahol $c = \cos \Theta$, $s = \sin \Theta$):

$$\begin{aligned} |\bar{\Psi}\rangle &= (c + is\hat{X}_1)(c + is\hat{X}_2)(c + is\hat{X}_3)|\bar{\Phi}\rangle = \\ &= c^3|\bar{\Phi}\rangle + isc^2(\hat{X}_1 + \hat{X}_2 + \hat{X}_3)|\bar{\Phi}\rangle - s^2c(\hat{X}_1\hat{X}_2 + \hat{X}_1\hat{X}_3 + \hat{X}_2\hat{X}_3)|\bar{\Phi}\rangle - is^3\bar{X}|\bar{\Phi}\rangle, \end{aligned} \quad (28)$$

ahol $\bar{X}$ a logikai $\hat{X}$ (ami az összes kvantumbiten hat): $\bar{X} = \hat{X}_1\hat{X}_2\hat{X}_3$. Ezután megmérjük a szindrómákat. A szindrómamérések eredménye egy lesz az alábbi négyből:

$\hat{Z}_1\hat{Z}_2$	$\hat{Z}_2\hat{Z}_3$	Mérés után az állapot	Javítás után az állapot	Ebből Θ_L
0	0	$ \Phi'\rangle = \frac{1}{C_1}(c^3 \Phi\rangle - is^3\bar{X} \Phi\rangle)$	$ \Phi'\rangle = \frac{1}{C_1}(c^3 \Phi\rangle - is^3\bar{X} \Phi\rangle) = e^{i\Theta_L\bar{X}}$	$\tan\Theta_L = \frac{-s^3}{c^3}$
0 1 1	1 0 1	$ \Phi'\rangle = \frac{1}{C_2}(isc^2\hat{X}_1 \Phi\rangle - s^2c\bar{X}_2\hat{X}_3 \Phi\rangle)$	$ \Phi'\rangle = \frac{1}{C_2}(isc^2 \Phi\rangle - s^2c\bar{X} \Phi\rangle) = \frac{ics}{C_2}(c \Phi\rangle + is\bar{X} \Phi\rangle) = ie^{i\Theta_L\bar{X}} \Phi\rangle$	$\Theta_L = \Theta$

3. táblázat. A globális forgatások (Θ_L) kiszámolása $N = 3$ -nál, különböző szindrómamérések esetén

A $\hat{Z}_1\hat{Z}_2$ jelöli azt a szindrómamérést, ami az 1. és a 2. kvantumbit paritását hasonlítja össze, $\hat{Z}_2\hat{Z}_3$ azt a mérést, ami a 2. és a 3. paritását vizsgálja. A $\{C_1; C_2\}$ konstansok, amik egységshosszúra normálják a kvantumállapot-vektorok értékét.

Hibajavítás hibás kiolvasás esetén

A kvantumszámítógépeken a méréseknek is van valamekkora hibája. Ez azt jelenti, hogy az állapot a $|0\rangle$ állapotban van és a (számítási bázisban) megmérve mégis 1-et ad, vagy $|1\rangle$ helyett 0-t. Ezt én úgy modellezem, hogy a szindrómamérések elvégzése után q valószínűséggel a mérés eredményét megváltoztatok 0-ról 1-re és fordítva.

Hibás ancillakiolvasásoknál nem elegendő egyetlen mérési kört csinálni, ugyanis a hibás kiolvasás azt eredményezni, hogy nem lehet megbízhatóan megmondani, hogy egy kvantumbiten valóban történt X hiba, vagy csak az ancilla kiolvasása okozott eltérést. Ezt úgy oldhatjuk meg, hogy nem csak egyszer vizsgáljuk a kvantumbitek paritását, hanem T körben egymás után. Ezután a korábban ismertetethez hasonló, csak két dimenziós, MWPM problémát megoldó *Blossom algoritmust* használom, ami pythonba a *PyMatching* könyvtárba van implementálva. Ez a könyvtár C++-ban van írva a jó teljesítményért [3], viszont az interfész miatt könnyen használható pythonnal.

1.6. A hibajavítás hatékonyságának a meghatározása

A szindróma mérések eredményét a PyMatching python könyvtár segítségével feldolgozom. A program megkeresi a MWPM-et, majd az alapján meghatározza, hogy melyik kvantumbiteken kell X hibákkal hatnia, hogy kijavítsa a hibákat.

A hűtlenség

A hibajavítás hatékonyságának a meghatározására bevezetjük a *hűtletnég (infidelity)* mennyiséget, ez a 0 és 1 közötti (valós) szám azt mutatja meg, hogy a kvantumállapot, amibe a rendszer került végül (a hibajavításokat elvégezve), mennyire tér el attól a kvantumállapottól ($|\Psi\rangle$), amelyből indult. A hűtlenség általánosan a

$$P_L = 1 - \langle\Psi|\hat{\rho}|\Psi\rangle \quad (29)$$

összefüggéssel határozható meg. A vizsgálatban nálam a kiindulási állapot mindig a logikai $|\bar{0}\rangle$ (pl. egy 5 kvantumbites rendszerben $|00000\rangle$) állapot volt, így a hűtlenség:

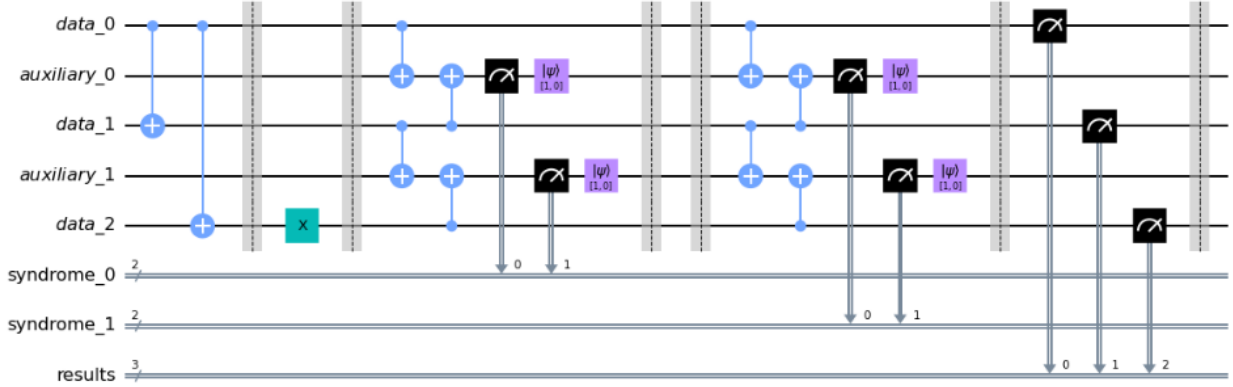
$$P_L = 1 - \langle\bar{0}|\hat{\rho}|\bar{0}\rangle, \quad (30)$$

ahol $\hat{\rho}$ pedig annak az állapotnak a sűrűségoperátora, amibe a rendszer kerül a szimuláció végén - ezt a Qiskit visszaadja.

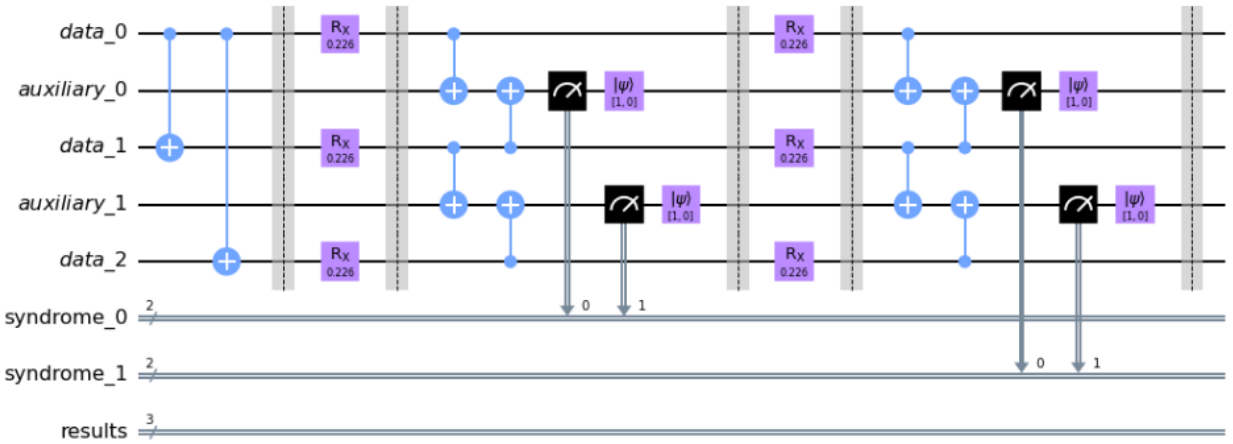
A szimulációt sokszor futtattam és a kiszámolt hűtlenség értékeknek az átlagát meghatároztam, ezek vizsgálatával lehet összehasonítani a hibamodelleket. A hűtlenség minél nagyobb, a hibák annál veszélyesebbek az eltárolt kvantuminformációra.

2. A szimulációs program

A mérési hibák áramköri szintű modellezésére a qiskit python szoftverfejlesztő készletet használtam. Az elkészített áramkörökkel Monte-Carlo szimulációt végeztem, amely azon alapul, hogy véletlenszerű eseményeket generálunk sokszor, majd ezek statisztikai elemzésével vonunk le következtetéseket a problémáról. A 5. ábrán láthatunk egy áramkört amely random X, a 6. ábrán pedig egy olyat ami koherens hibákat szimulál.



5. ábra. Az áramkör, amely random X hibákat szimulál, $N = 3$, $T = 2$, $p = 0.05$



6. ábra. Az áramkör, amely koherens hibákat szimulál, $N = 3$, $T = 2$, $\Theta = 0.2255$

A két áramkör felépítése hasonló. 5 kvantumbit van mindkét áramkörben (a felső 5 vízszintes vonal), ezért 5, mert $N = 3$, vagyis kell 3 qubit az adatoknak és még $N - 1$ az ancilláknak, vagyis általánosan: N bites kódhoz $2N - 1$ kvantumbit kell. Az ábrán a többi kettőzött vonal klasszikus biteket jelöl, ezek tárolják az ancillamérések eredményét. Ezekből mivel $N - 1$ ancilla van és T -szer ismétljük a mérést, általánosan $(N - 1)T$ klasszikus bit kell.

Az első szakaszban a kvantumállapotot bekódoljuk N kvantumbitre. Fontos, hogy ismeretlen kvantumállapotot nem lehet lemásolni (klónozni), viszont nekünk nem is az kell, hanem hogy több bit összefonódásában legyen eltárolva az információ, ennek a létrehozásához van szükség a CNOT kapukra.

A második szakaszban van lehetőség a hibák megjelenésére: a random X hibákat úgy modellezem, hogy p valószínűséggel betesz a program ide egy X kaput; a koherens hibák modellezése úgy történik, hogy mindenhova berak egy kis Θ szögű forgatást - egy $R_x(\Theta)$ kaput. A Θ értékét így határozhatjuk meg:

$$\sin^2 \Theta = p \Rightarrow \Theta = \arcsin(\sqrt{p}), \quad (31)$$

mivel ha egy $|0\rangle$ állapotban lévő kvantumbiten egy Θ forgatás hat, akkor $\sin^2 \Theta = p$ valószínűséggel fogunk 1-et mérni.

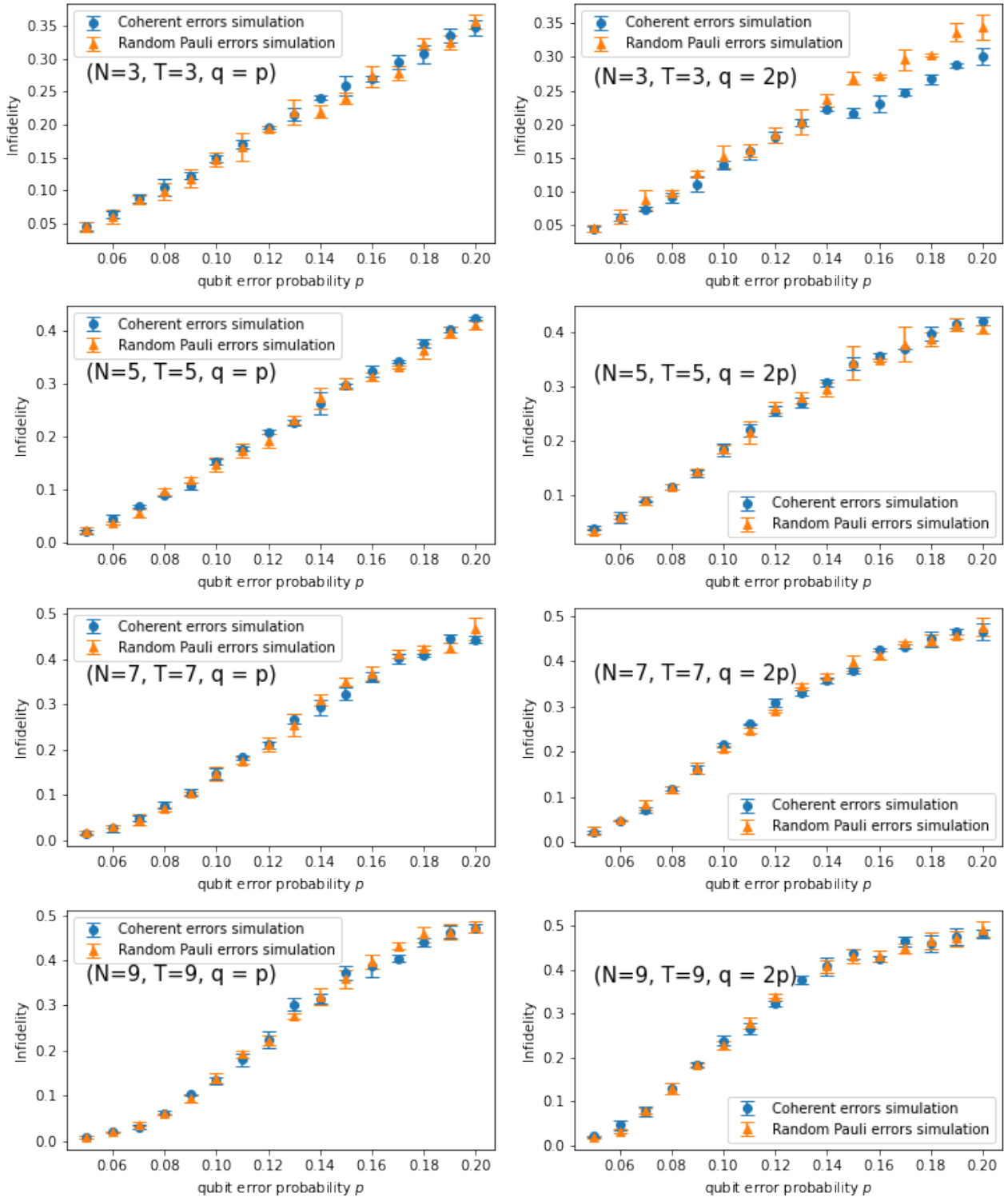
Ezután következik a paritások meghatározása a 1.5. fejezetben leírtak szerint. Egy fontos különbség, ami nem szerepelt abban a fejezetben, itt azonban meg kell tennünk, hogy visszainicializáljuk az ancillákat a $|0\rangle$ állapotba, mivel azok a mérés során a $|0\rangle$ vagy az $|1\rangle$ állapotba is kerülhetnek, viszont a következő méréshez mindenképpen a $|0\rangle$ állapotban kell lenniük, ezért visszaállítom őket arra az állapotra. Ezután ez a szakasz ismétlődik T alkalommal.

A random X hibák szimulációjánál még a végén az összes kvantumbitét kimérem (feltételezve, hogy vége a számolásnak, már nem kell a bennük tárolt kvantuminformáció), ez egy plusz ellenőrzésre használható.

3. A szimuláció eredményei

Ezután ismertetem a szimuláció futtatásnak legfőbb eredményeit. Ezeket legtöbb esetben hibasáv vagy hisztogram típusú diagramon ábrázolom.

3.1. A két hibamodell ugyanakkora veszélyt jelent az eltárolt információra



7. ábra. A hűtlenség a kvantumbitek p hibájának függvényében különböző paraméterek mellett

Az ábrán több futtatás eredménye látható. Az összes futtatásban a kód méretével, megegyezik a körök száma: $N = T$. A bal oldali oszlopban olyan futtatások eredményei láthatók, ahol a q ancillamérési hibák valószínűsége megegyezik a p kvantumbitek hibájának a valószínűségével. A jobboldali oszlopban olyan futtatások, ahol a mérési hibák valószínűsége kétszerese a kvantumbitek hibájának. Az egyes sorokban egyes kódméreteket ábrázoltam $N = \{3; 5; 7; 9\}$. Az összes grafikonon a p hibák függvényében ábrázolom a hűtlenség értékeit. Teljesen azonos paraméterekkel minden ponthoz legalább háromszor futtattam le a szimulációt, és az értékek átlagát jelenítem meg adatpontként, a σ szórásukat meg a hibaszívon. Kék körökkel vannak jelölve a koherens hibák hűtlenségei, narancssárga háromszögekkel meg a véletlen X hibák hűtlenség értékei.

A legnyilvánvalóbb információ ami megállapítható a grafikonról, hogy a koherens és a véletlen hibák szinte mindig hibahatáron belül megegyeznek egymással (kivéve a jobb felső ábrát), ebből az következik, hogy **mind a két hibamodell ugyanakkora veszélyt jelez a kvantumbitekre** azonos paraméterek mellett. Ezt az eredményt megerősíti a [4] cikk egyik eredménye is, miszerint hibátlan kiolvasások esetén a koherens és az inkohere hibák is ugyanakkora hűtlenséget adnak.

Megfigyelhető még továbbá az is, hogy ahogy nő a kód mérete, úgy nő az egyes hűtlenségek értéke azonos p hibaérték mellett. A nagyobb méretű kódoknál ez a növekedés gyorsabban történik, ugyanakkora p esetén egy nagyobb kód, nagyobb hűtlenséget ad.

3.2. A hibák és a kód méretének növelésével, a hibajavítás után, egyre többféle forgatási szög jelenik meg koherens hibák esetén

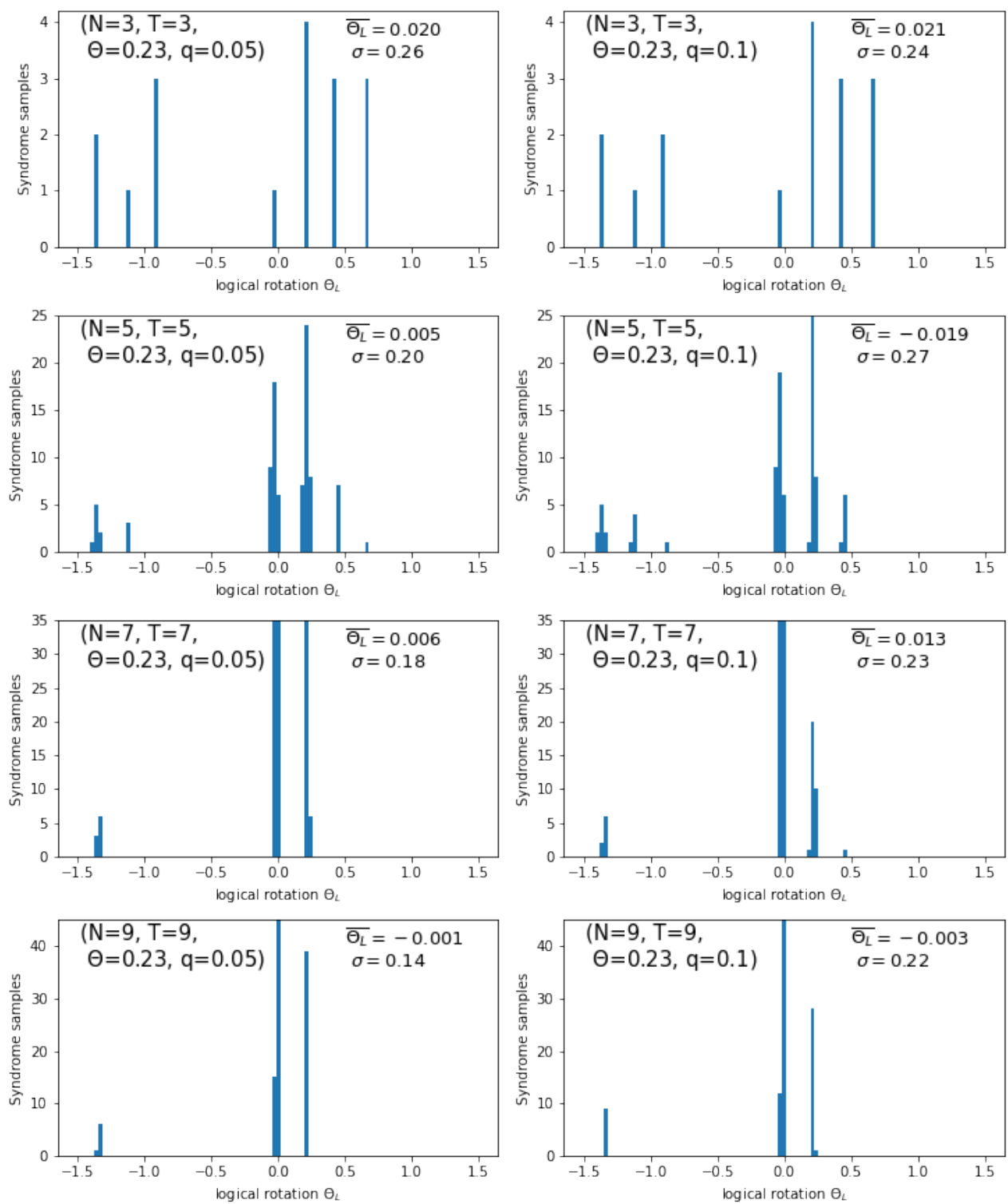
Θ , a forgatás ami hat minden körben az összes kvantumbiten a (31) egyenlet alapján függ a kvantumbitek hibájától.

A hisztogramokon megfigyelhetjük a hibajavítás után bekövetkező globális forgatásokat. Az egyes ábrák egy adott Θ forgatás mellett készültek, a baloldali oszlopban lévő grafikonoknál a kvantumbitek p hibája megegyezik a mérések q hibájával, a jobboldali oszlopban fele annak. A hisztogramok az egyes hibajavítás utáni logikai forgatások Θ_L értékeinek a gyakoriságát mutatják radiánban mérve.

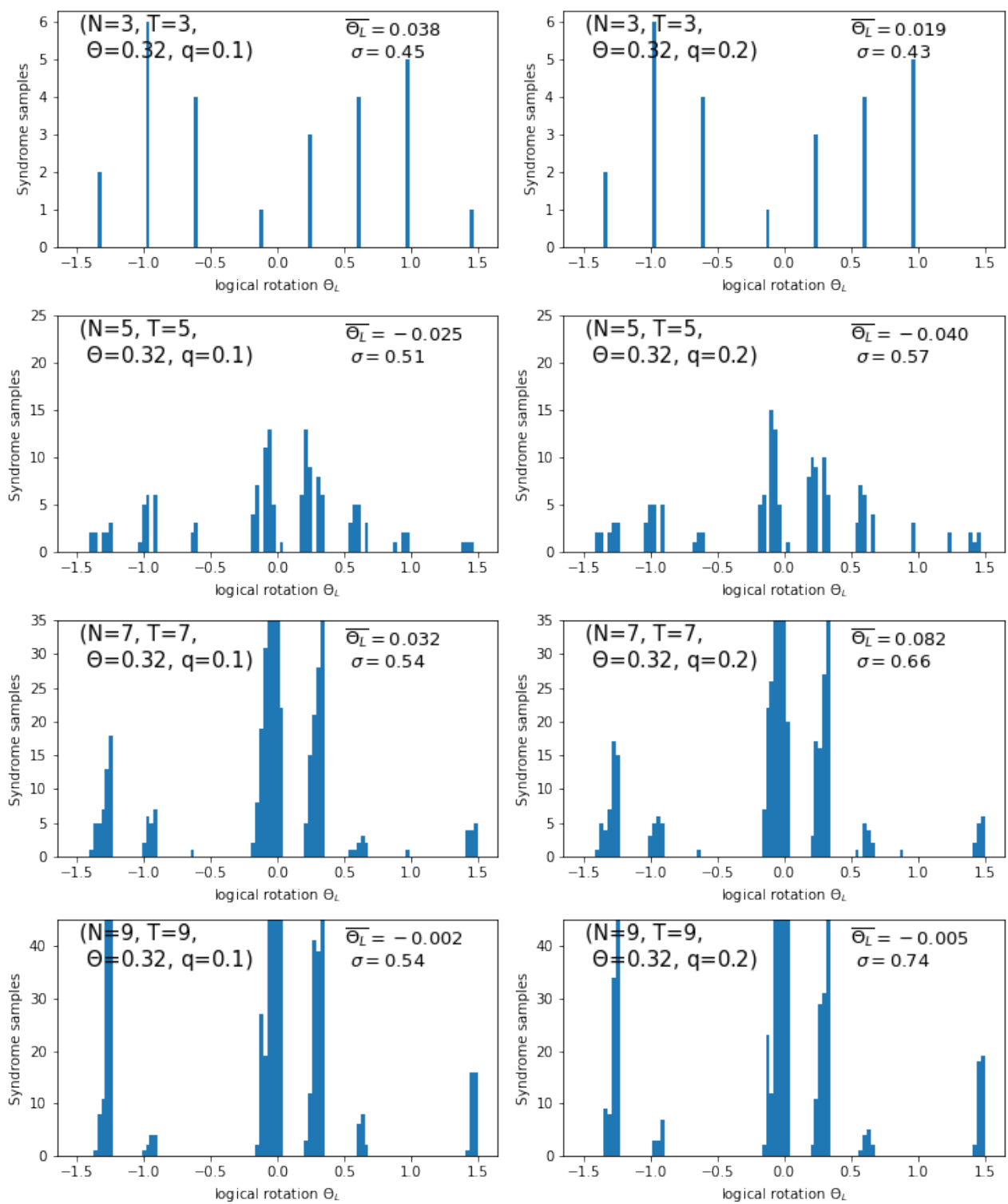
Bizonyos grafikonok nem a teljes eloszlást mutatják, mivel minden hisztogramon van egy kiemelkedő csúcs a 0-nál, mivel a 0 forgatás a leggyakoribb (szerencsére, mivel ez azt jelenti, hogy a hibajavítás sikeres volt, vagyis nem változott az eltárolt információ), viszont így elfedheti, hogy milyen eloszlást mutatnak a forgatások, ha azok nem 0-k. Ezért nincs a teljes eloszlás ábrázolva, csak a számunkra érdekes 0-tól különböző csúcsok.

Az összes grafikonról látható, hogy kis kódméreteken (N=3) a hibák nem feltétlenül követik a várt Gauss eloszlást, mivel szinte ugyanolyan gyakori az összes megjelenő forgatásérték. A kód méretének növelésével a várt eloszlás egyre jobban megfigyelhető: egyértelműen kiemelkedik 0-nál a leggyakoribb érték (még ha az le is van vágva a grafikonról) és az átlag ($\overline{\Theta_L}$) közel van 0-hoz minden hisztogramon.

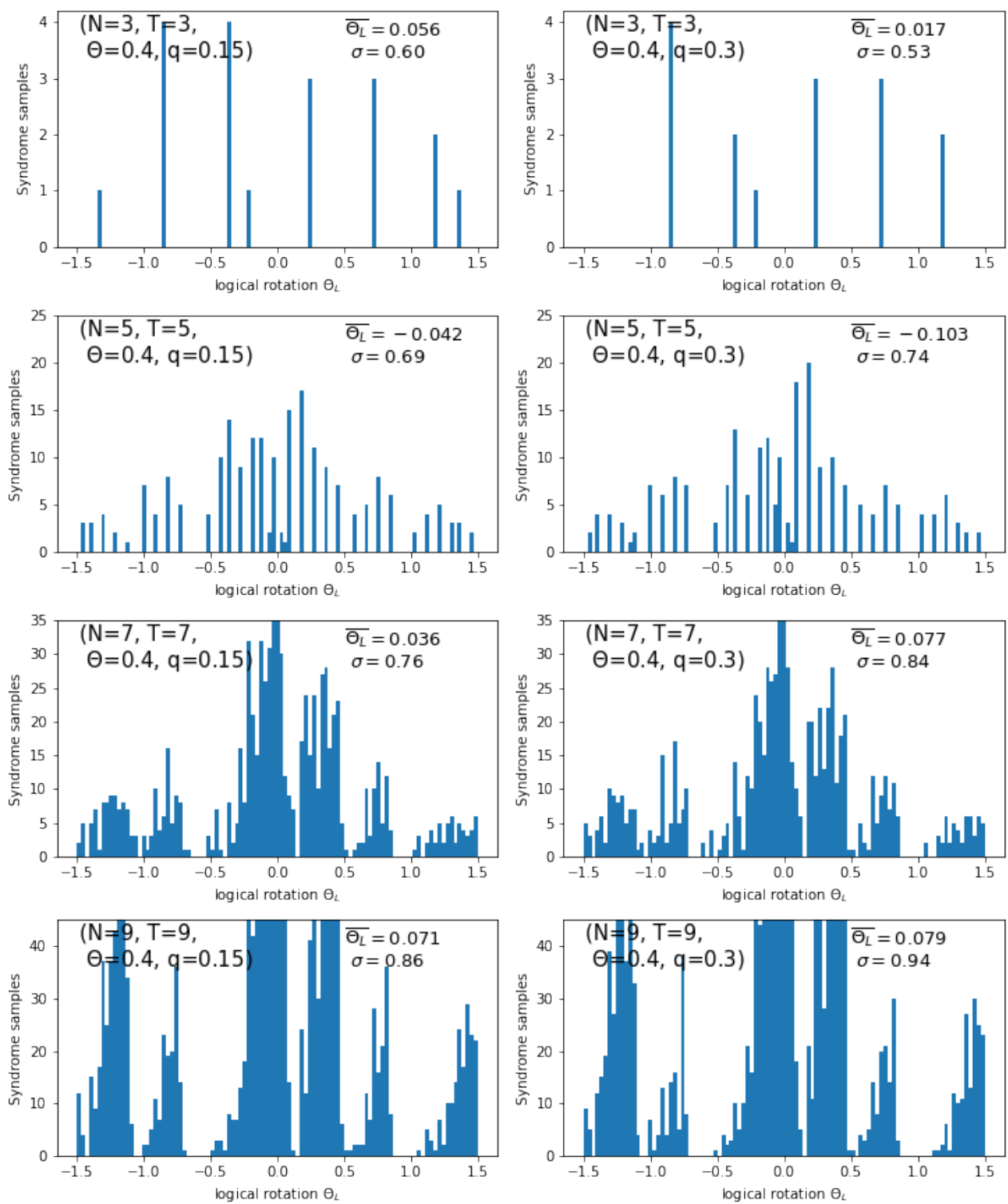
A kvantumbitek hibájának a növelésével egyre több globális forgatási szög jön létre. Amik $N = 5$ vagy nagyobb kódok esetén egyre jobban közelítenek egy haranggörbe alakjához. Ez a grafikonra írt σ szórásból is látszik, ahogy nő az ismétlések száma és a bithibák értéke úgy nő az adatok szórása is.



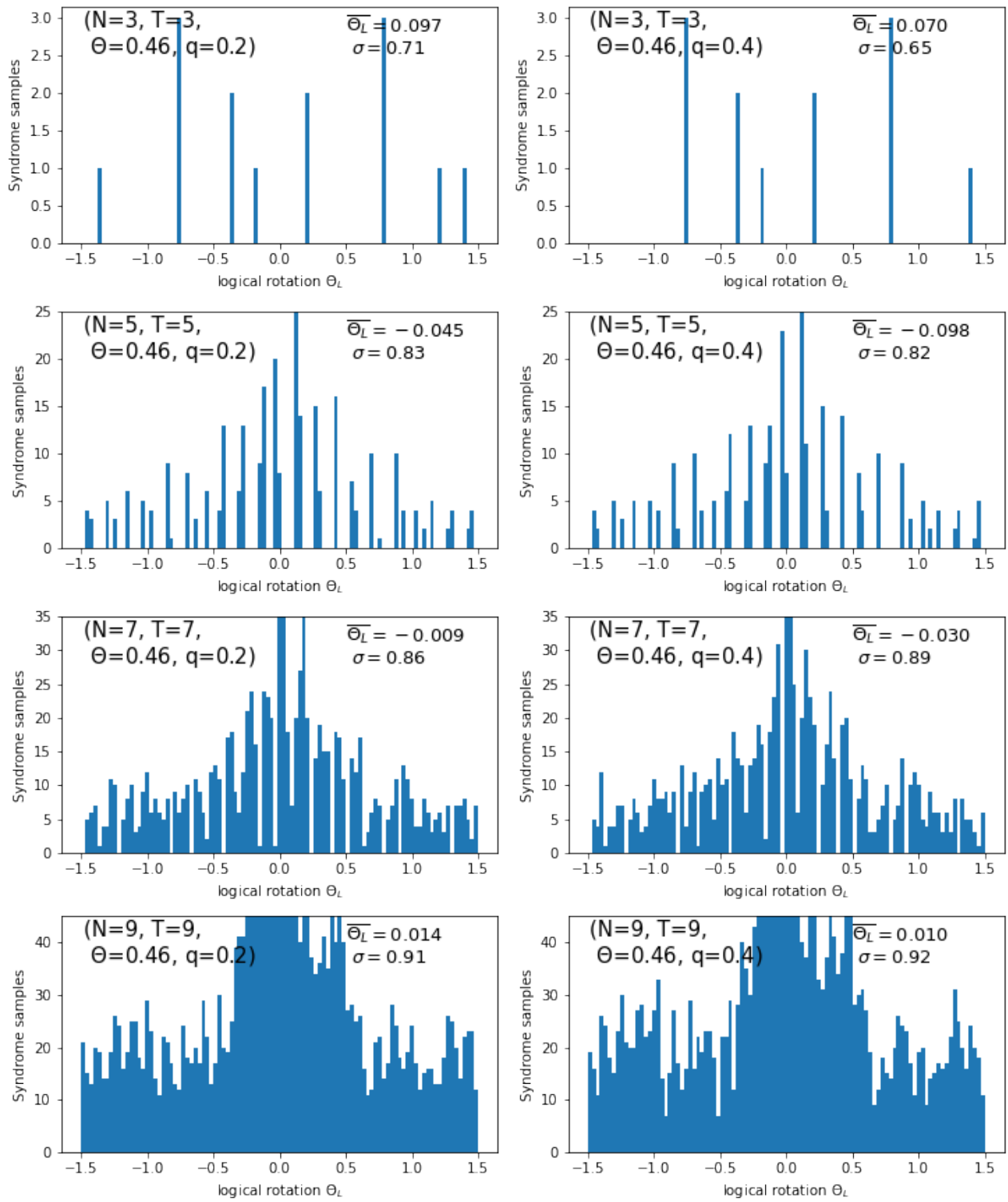
8. ábra. A Θ_L logikai forgatások mértéke és eloszlása különböző paraméterek mellett,
 $\Theta = 0.23$



9. ábra. A Θ_L logikai forgatások mértéke és eloszlása különböző paraméterek mellett,
 $\Theta = 0.32$



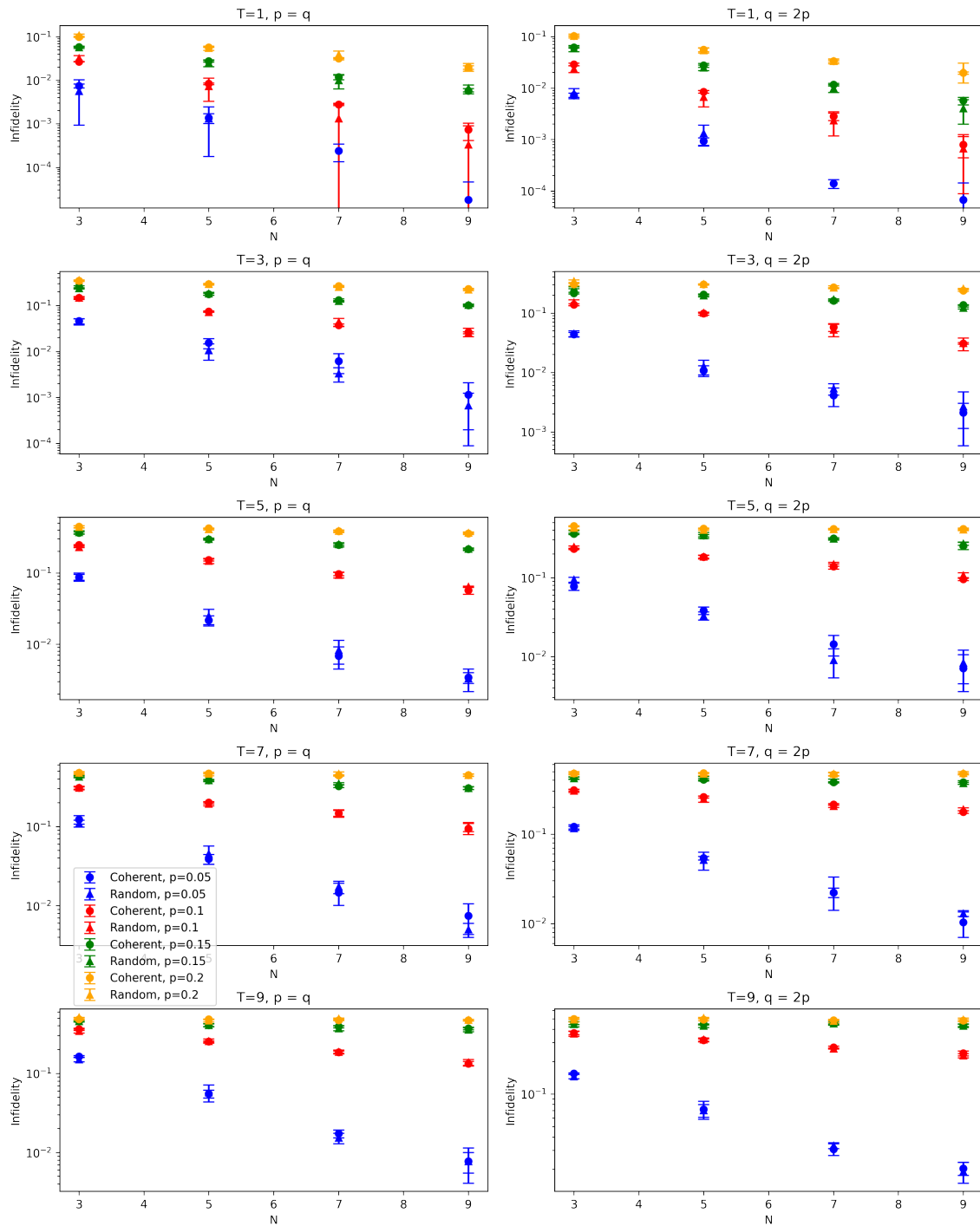
10. ábra. A Θ_L logikai forgatások mértéke és eloszlása különböző paraméterek mellett, $\Theta = 0.4$



11. ábra. A Θ_L logikai forgatások mértéke és eloszlása különböző paraméterek mellett,
 $\Theta = 0.46$

3.3. A kód méretének növelésével csökken a hűtlenség, bár néha nem szignifikánsan

A következőkben azt vizsgálhatjuk, hogy különféle kódméretek hogyan hatnak a hibajavítás hatékonyságára, vagyis a hűtlenségre.



12. ábra. A hűtlenség különféle kódméretek és ismétlési körök mellett

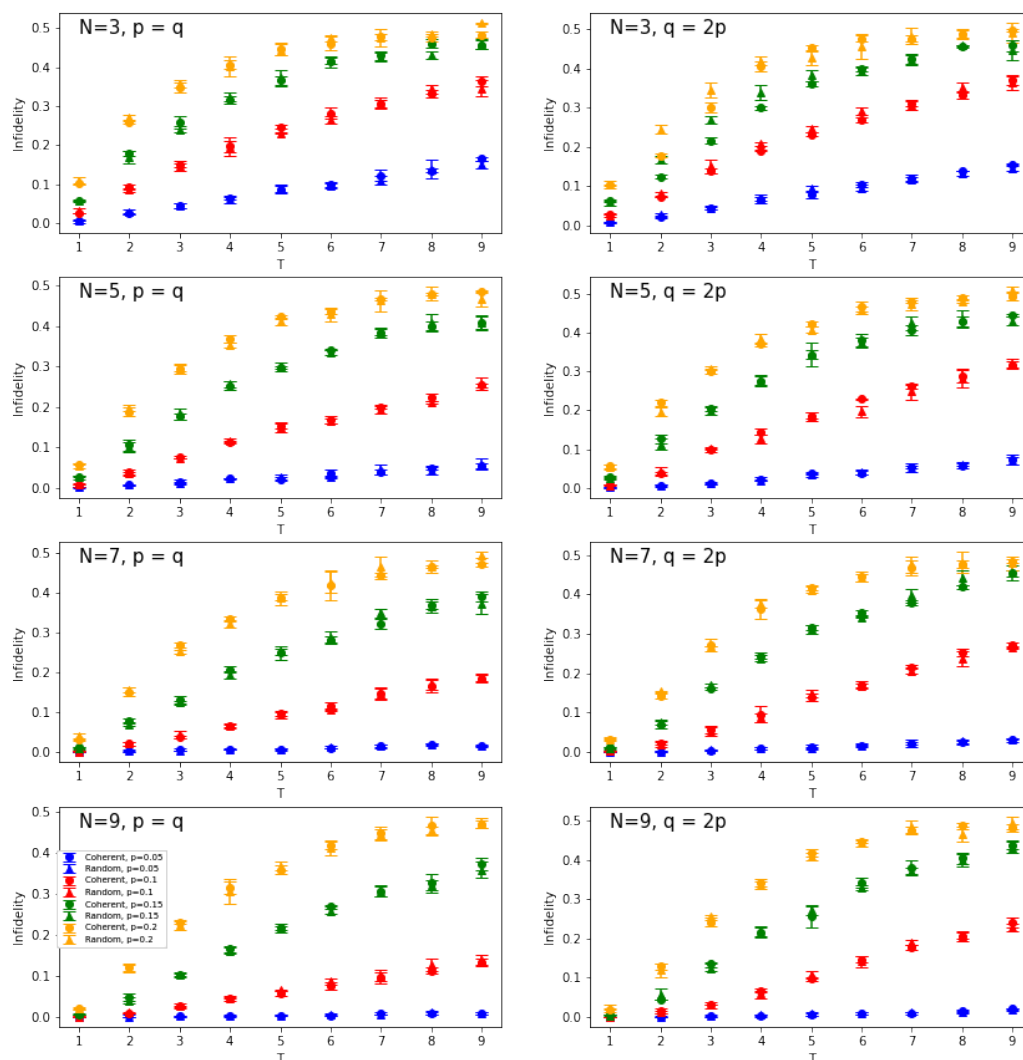
A 12. ábrán egy hibasáv diagramon látható, hogy hogyan változik a hűtlenség értéke különböző számú adat-qubitek esetén. A bal oldali oszlopban olyan szimulációk diagramjai vannak, ahol a p (kvantumbit) és a q (kiolvasási) hibák egyenlő nagyságúak, a jobb oldali oszlopban, pedig a $q=2p$. Az egymás alatti sorok egyre növekvő számú mérési kört mutatnak $T = \{1; 3; 5; 7; 9\}$. A különböző színű adatsorok különféle p értékeket mutatnak, külön koherens és külön véletlen Pauli X hibánként. A hibasávok nagysága az ugyanolyan paraméterek mellett történt szimulációs pontok szórását mutatja.

A grafikonról megállapítható, hogy - azonos paraméterek mellett - a koherens és a véletlen X hibák adatai hibán belül mindig megegyeznek, vagyis eszerint nincs különbség a két hibamodell között a hűtlenség tekintetében, és így a hibajavítás hatékonyságában. Ez abból látható, hogy a rombusszal és a körrel jelölt adatsorok szinte semennyire nem válnak el egymástól.

A grafikonról látható, hogy az adatbitek számának növelésével a hűtlenség exponenciálisan csökken (az exponenciális csökkenés logaritmikus skálán ábrázolva egyenest ad, és az ábrázolt pontok jó közelítéssel egy-egy egyenesre illeszkednek). Kis ismétlésszám esetén ($T = 1$) a kód jelentős hibajavítást eredményez $q = 0.2$ -nél egy, kisebb hibák esetén majdnem három nagyságrenddel is csökken a hűtlenség az adatbitek számának növelésével. $T = 3$ esetén a kis hibákat szintén nagy hatékonysággal tudja javítani a kód ($p = 0.05$ -nél ismét majdnem három nagyságrenddel, a nagyobb hibák esetén azonban a hibajavítás hatékonysága már nem csökken egy nagyságrenddel). Az ennél nagyobb ismétlésszámok esetén csak kis hibák mellett jár hatásnövekedéssel a bitek számának növelése. Nagy ismétlésszámnál, nagy bithibák esetén az adat-kvantumbitek számának növelése nem jár jelentős teljesítménynövekedéssel - a pontok szinte vízszintes egyenesekre esnek.

3.4. A körök számának növelése növeli a hűtlenséget

Az előző alfejezetben leírtakból sejthető, hogy az ismétlések számának a növelése nem javítja a hibajavítás hatékonyságát, hanem éppen rontja. Vizsgáljuk meg részletesebben is ezt a kérdést.

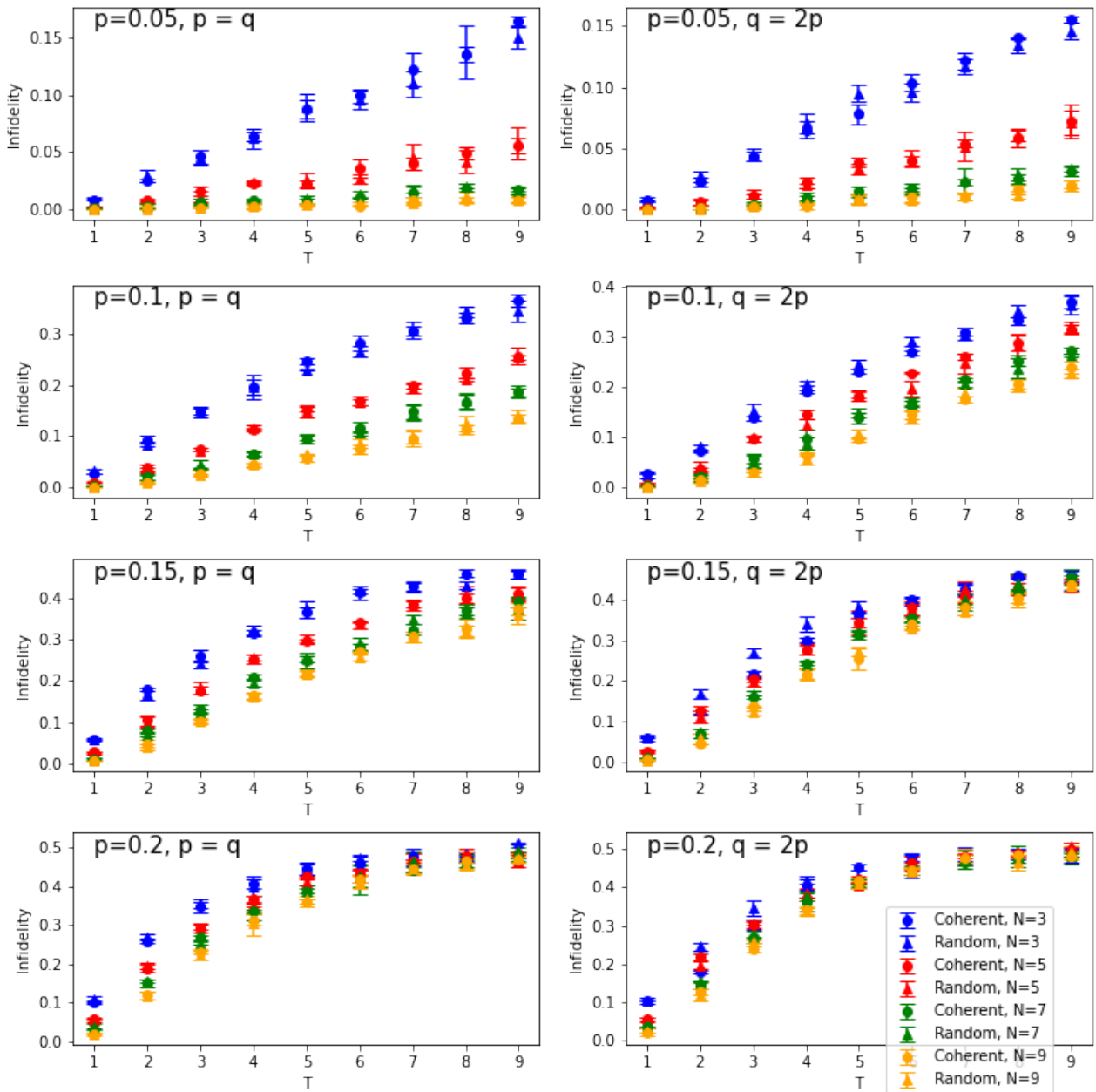


13. ábra. A hűtlenség az ismétlések számának növelése mellett, kód méretenként csoportosítva

A 13. és 14. ábrán fel van tüntetve a hűtlenség változása az ismétlések számának növekedése függvényében, egy-egy hibasáv típusú grafikonon. A hibasávok méretét az azonos paraméterek mellett futtatott több szimulációs eredmény szórása adja. A bal oldali oszlopban olyan szimulációk diagramjai vannak, ahol a p (kvantumbit) és a q (kiolvasási) hibák egyenlő nagyságúak, a jobb oldali oszlopban, pedig a $q = 2p$.

A 13. ábrán az egyes sorokban a növekvő kód méretek mellett van feltüntetve a hibajavító eljárás hatékonysága, a hűtlenség. Az egyes adatsorok az egyes bithibákat jelentik. Ismét a koherens és a véletlen X hibák hűtlensége hibán belül mindig egybeesik. Továbbá ahogy nő a kód mérete úgy nő a hűtlenség értéke is, és nagy bithiba ($p=0.2$) esetén viszonylag hamar minden paraméter mellett ($T=5$) kör után elérte legalább a 0.4-es hűtlenség értéket - a 0.5-ös hűtlenség azt jelenti, hogy az eredetileg eltárolt $|0\rangle$ állapotból semmi információ nem marad meg, a mérés után $1/2$ -ed eséllyel mérnénk $|0\rangle$ -t és $1/2$ -ed eséllyel $|1\rangle$ -et, ami már egy nagyon rossz eredmény egy hibajavító eljárástól. $p=0.15$ -ös hibánál is gyorsan romlik a hibajavítás hatékonysága, $p=1$ -nél is viszonylag hamar nő a hiba (az eltérés a kék és a piros adatsor között különösen a jobb alsó grafikonon figyelhető meg). $p=0.05$ -nél egy kicsit más a helyzet, a grafikonról leolvasható, hogy ha nagyobb kód méretet választunk, úgy kisebb méretben nő a

hiba az ismétlések számának növelésével. Vagyis a bithibák nagymértékben meghatározzák, hogy a hibajavítás során az ismétlések számának növelése, hamar elrontja a hűséget, vagy ez nem történik meg $T = 9$ kör alatt.



14. ábra. A hűtlenség az ismétlések számának növelése mellett, kvantumbit hibaként csoportosítva

A 14. ábrán az egyes sorokban az egyes bithibák szerint vannak csoportosítva az adatok, itt az egyes adatsorok a kód méretét jelölik. Ezen az ábrán az előzőnél talán jobban meg lehet figyelni a hűtlenség nagyságának változását. Ha nagy a kvantumbitek hibája ($p = \{0.15; 0.2\}$), akkor a hibajavítás során lényegében a kód méretétől függetlenül, a körök számának növelése hamar ($T=5$)-től elrontja a kvantumbitek hűségét. $p=0.15$ esetén a hűtlenség lassan növekszik a körök számának növekedésével. Kis bithibák esetén ($p=0.05$) egyrészt lassan nő a hűtlenség a körök számának növekedésével, másrészt számít a kód mérete: az $N=3$ méretű kód kétszer vagy háromszor rosszabbul teljesít, mint a nagyobb kódok. Ebből az következik, hogy a kód méretének növelése, csak egy bizonyos bithibák alatt érdemes, ugyanis túl nagy hibák esetén a kód méretétől függetlenül hamar elromlik az információ, a körök növelése esetén. **Vagyis a körök számának növelése nem javít a hibajavítás hatékonyságán.** Ez könnyen belátható a koherens forgatásoknál: egymás után sok kis forgatás eredménye ugyanaz, mint egy nagyobb forgatásnak. A véletlen hibák esetén a kód akkor nem tudja kijavítani a hibákat, ha egy körben az adatbitek több mint felén hat egy X hiba, a körök számának növelésével ennek az esélye egyre nagyobb.

4. A szimuláció futtatása

4.1. A program

A szimulációs programot python programozási nyelven írtam. A kvantumszimulációk elkészítéséhez a *Qiskit*¹, a MWPM probléma megoldásául szolgáló Blossom algoritmushoz a *PyMatching* [3] könyvtárat használtam. A matematikai számításokat *Numpy* és a *Pandas* segítségével végeztem, az ábrákat *Matplotlib*bel készítettem. A kódot egy *Jupyter Notebook*ban² írtam és futtatást először az otthoni asztali számítógépemen végeztem. Azonban viszonylag hamar kiderült, hogy az otthoni futtatás nagyon lassú ezért keresetem egy jobb megoldást.

A Google Colaboratory³ (röviden: *Colab*) egy ingyenesen használható felület, amely segítségével a Google nagy teljesítményű szerverein lehet futtatni programokat. Széles körben használható adatelemzéshez, mesterséges intelligencia tanításhoz, és szimulációk futtatásához. Ehhez az ingyenesség mellett az is hozzájárul, hogy rendkívül egyszerűen használható (és csak Google fiók kell hozzá), könnyen meg lehet osztani a projekteket, és lehet programokat futtatni GPU-n is, ami hatalmas sebességnövekedést eredményez a CPU-n történő futtatással szemben.

A program egy Monte-Carlo szimuláció, ami véletlen eseményeken alapul, hogy numerikus eredményeket adjon egy problémára. Ezért hogyha pontosabb eredményt szeretnénk elérni, több futtatást kell végezni, ami természetesen erőforrás-igényesebb, nálam az erőforrás a Google Colab miatt nagy részben megvolt, viszont a futtatás így is viszonylag sok időt vett igénybe. A *Qiskit* könyvtár lehetővé teszi, hogy a kvantumáramköröket GPU segítségével futtassuk, ez az én programomban körülbelül tízszeres sebességnövekedést eredményezett a CPU-hoz képest (bár csak nagyon kevés olyan szimuláció volt, amit CPU-n és GPU-n is lefuttattam, hogy vizsgáljam a sebességnövekedést). Arra hogy nem csak CPU-n lehet futtatni, azonban csak később jöttem rá, így a generált adatok egy része CPU-s a másik része GPU-s futtatás együttes eredménye.

4.2. A futtatási idők

A 4. táblázatban látható, hogy összesen mennyi futásidőre volt szükség az egyes grafikonok elkészítéséhez. Mivel sajnos csak a futtatások elkezdése után jöttem rá, hogy lehet GPU-n is futtatni a szimulációt ez a táblázat kevésbé informatív, mivel nem különbözteti meg a CPU-s és a körülbelül tízszer gyorsabb GPU-s futtatásokat, az viszont kiderül belőle, hogy nekem mennyi időbe telt az egyes grafikonokhoz legenerálni az adatokat. Az összes futtatás ideje együttesen 127 666.12s $\approx$ 35 óra volt.

Ábra	Teljes futási idő (s)	Ábrázolt pontok száma
7.	94 142.66	890
8.	5 285.53	2312
9.	5 608.96	5232
10.	5 584.58	8284
11.	4 774.30	8392
12.	49 719.14	1080
13.	59 929.88	1848
14.	59 929.88	1848

4. táblázat. Az egyes diagramok elkészítéséhez szükséges futtatási idő és az ábrázolt pontok száma

Az ábrázolt pontok száma mást jelent a hibasáv és a hisztogram típusú grafikonok esetén. Minden hibasáv típusú diagramon az adatpontok legalább 3 - ugyanolyan paraméterek mellett futtatok - szimuláció átlagai, a hibasávok pedig ezek szórásai. A Monte-Carlo metódus miatt sokszor kell futtatni, hogy pontos eredményt kapjunk, ezért én minden pont elkészítéséhez 1000-szer futtattam minden szimulációt és abból számolta ki a program a hűtlenségek átlagát. Vagyis mivel minden ponthoz 1000 szimuláció futott és legalább 3 pont (átlaga és szórása) alkot egy-egy ábrázolt pontot a hibasáv diagramokon, így legalább 3000 futtatás kellett - azonos paraméterek mellett - egy ábrázolt pont elkészítéséhez. A táblázatban látható, hogy

¹<https://qiskit.org/> [Online] Hozzáférés dátuma: 2022. 10. 29.

²<https://jupyter.org/> [Online] Hozzáférés dátuma: 2022. 10. 29.

³<https://colab.research.google.com/> [Online] Hozzáférés dátuma: 2022. 10. 29.

az egyes diagramok elkészítéséhez hány pontot kellett szimulálni (a táblázatban az összes pont száma fel van tüntetve, amelyből az átlagokat és a szórásokat ábrázoltam). A hisztogram típusú diagramokon az ábrázolt pontok száma, azt jelenti, hogy hány Θ_L érték jelent meg az egyes hisztogramokon.

5. Konklúzió

A TDK dolgozatomban bemutattam a kutatásomat, melyben összehasonlítottam a koherens és inkohereus hibák hatásait egy olyan kvantumszámítógép rendszerben, amelyben ismétléses kódot használunk az információ védelmére. Ehhez készítettem egy numerikus szimulációs programot és ennek a különböző paraméterek mellett futtatott adatait elemeztem. A szimulációs programot a *Google Colabon* futtattam.

A két hibamodell (koherens és inkohereus hibák) bármilyen vizsgált paraméter mellett ugyanakkora veszélyt jelentenek az eltárolt kvantuminformációra, ha ismétléses kóddal javítjuk a hibákat.

A körök számának növelése általában rontja a hibajavítás hatékonyságát, mivel több ideje (és így esélye) van az ismétléses kóddal nem kijavítható hibák megjelenésének. Ez nagyobb kódméretek esetén is így van. A nagyobb kód általában hatékonyabb a hibajavításra, mint a kisebb, de a körök számának növelése általában jóval jelentősebb hibát okoz, amit a nagyobb kódok sem tudnak kijavítani.

Hivatkozások

- [1] Sergey Bravyi, Matthias Englbrecht, Robert König, and Nolan Peard. Correcting coherent errors with surface codes. *npj Quantum Information*, 4(1):1–6, 2018.
- [2] Craig Gidney and Martin Ekerå. How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits. *Quantum*, 5:433, 2021.
- [3] Oscar Higgott. Pymatching: A python package for decoding quantum codes with minimum-weight perfect matching. *ACM Transactions on Quantum Computing*, 3(3):1–16, 2022.
- [4] Eric Huang, Andrew C Doherty, and Steven Flammia. Performance of quantum error correction with coherent errors. *Physical Review A*, 99(2):022313, 2019.
- [5] Rajeev et al. Suppressing quantum errors by scaling a surface code logical qubit. *arXiv preprint arXiv:2207.06431*, 2022.
- [6] Eleanor G Rieffel and Wolfgang H Polak. *Quantum computing: A gentle introduction*. MIT Press, 2011.
- [7] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.
- [8] Peter W Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*, pages 124–134. Ieee, 1994.
- [9] Maarten Van den Nest. Universal quantum computation with little entanglement. *Physical Review Letters*, 110(6):060504, 2013.