Evaluacija disjunktnih normalnih formi pomoću Groverovog algoritma

Mateo Martinjak

KVA-RAC Seminar

1. Groverov algoritam

Groverov algoritam je kvantni algoritam koji nalazi s velikom vjerojatnošću,jedinstvneno rješenje neke funkcije f , i to radi u $O(\sqrt{n})$ poziva funkcije f , gdje je n veličina domene funkcije.

Analogan problem u klasičnom slučaju nemože se riješiti u manje od $O(\mathrm{n})$ poziva, jer se može desiti da je tražena vrijednost bas zadnja vrijednost domene.

To je naime vjerojatnosni algoritam, tj. daje točno rješenje problema s vjerojatnošću manjom od 1. Nema gornje ograde na broj ponavljanja koje bi bilo potrebno da se nađe traženo rješenje.

Detaljnije,uzmemo li da je domena veličine n, i funkcija f takva da

$$f(x) = \begin{cases} 1, x = \omega \\ 0, x \neq \omega \end{cases} \text{ za neki } \omega.$$

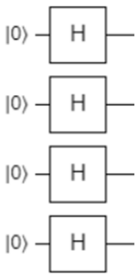
Taj f reprezentira proročicu U_ω u našem kvantnom sklopu:

$$U_\omega|x\rangle = \begin{cases} -|x\rangle, x = \omega \\ |x\rangle, x \neq \omega \end{cases}$$

Korak 1: Stavljanje ulaza u superpoziciju

Treba nam početno stanje: $|\Psi\rangle = \frac{1}{\sqrt{N}}\sum_{x=0}^{N-1}|x\rangle$

Kako bi ga dobili na ulaz $|00\ldots0\rangle$ primjenimo operator H:



Izlaz je tada: $(\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle))^{\otimes n}$ što je upravo $\frac{1}{\sqrt{N}}\sum_{x=0}^{N-1}|x\rangle$

Moraz ćemo malo drukčije taj vektora zapisati.

$|\alpha\rangle = \frac{1}{\sqrt{N-M}}\sum_x|x\rangle$ predstavlja vektor kao sumu onih koji nisu rješenja proročice, a

$|\beta\rangle = \frac{1}{\sqrt{M}}\sum_x|x\rangle$ sumu onih koji jesu rješenja.

Očito je početno stanje onda

$$|\Psi\rangle = \sqrt{\frac{N-M}{N}}|\alpha\rangle + \sqrt{\frac{M}{N}}|\beta\rangle \in span\{|\alpha\rangle, |\beta\rangle\}.$$

Grafički ćemo taj problem rješiti tako da ćemo uz određene operatore taj vektor što bliže preslikati blizu $|\beta\rangle$.

Korak 2: Implementacija proročice

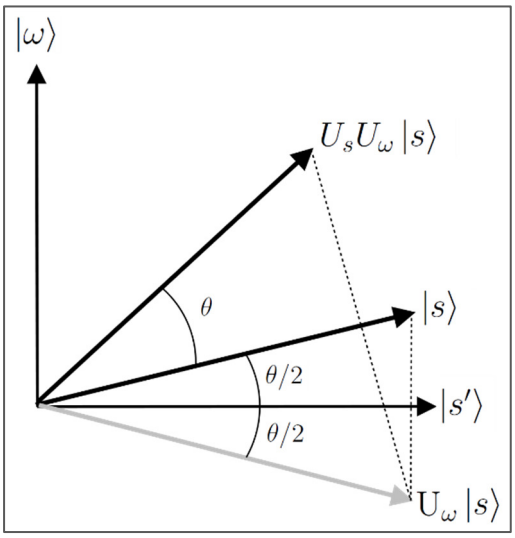
Proročica je zapravo klasični logički sklop koji dolazi kao ulaz u našem problemu. Jedino na željenom izlazu amplitudu okrećemo pomoću Z vrata u ovom slučaju, kako bi algoritam „znao“ koje ulaze mora amplitudno povećati.

Detaljnije o tome u poglavlju 4.

Korak 3: Operator refleksije

Dalje nam treba operator refleksije $|\Psi\rangle\langle\Psi| - I$.

Koja je ideja?



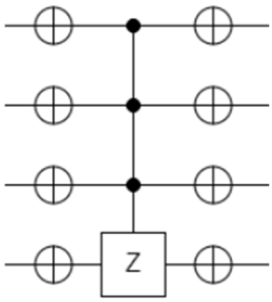
Želimo ulazni vektor nekako zrcaliti do rezultata, a u tome će nam pomoći proročića i operator reflekcije.

Proročica prvo zrcali amplitudu u negativnom smjeru,tada operator reflekcije oko početnog operatora zrcali to stanje natrag. Rezultatno stanje moža nakon jedne iteracije bude dovoljno blizu rezultatnog vektora, ali to ovisi o broju ulaznih qubita n.

Sa G ćemo nazvati kompoziviju ta 2 operatora.

Refleksiju oko vektora početnog vektora

možemo implementirati pomoću *X* i kontroliranih *Z* vrata.



Korak 4: Složenost

Ponovimo ,ulaz je $|\Psi\rangle = \sqrt{\frac{N-M}{N}}|\alpha\rangle + \sqrt{\frac{M}{N}}|\beta\rangle$.

Nakon malo računa dobitmo rješenje vektora nakon djelovanja operatora G:

$$G^k|\Psi\rangle = cos(\frac{2k+1}{2}\theta)|\alpha\rangle + sin(\frac{2k+1}{2}\theta)|\beta\rangle$$

Koliki moramo izabrati k?

Očito kada $|\Psi\rangle$ preslikamo dovoljno blizu vektora rješenja $|\beta\rangle$ a to će biti kada

$sin(\frac{2k+1}{2}\theta) \approx 1$.

Dobije se gruba procijena da za $k \in O(\sqrt{N})$.

2. Implementacija klasičnih sklopova u kvantnom algoritmu

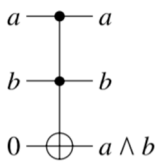
Logička vrata koja nam trebaju , AND i OR nisu invertibilna pa nemaju kvantne reprezentacije, ali ih možemo sastaviti od kvantnih sklopova korištenjem kvantnih vrata X i Toffolijevih vrata. NOT vrata već postoje u kvantnom obliku, to su zapravo X vrata.

2.1 OR vrata

OR vrata se mogu implementirati lagano pomoću Toffolijevih vrata.Moramo se samo pobrinuti da je na ulazu qubita koji nam služi za evaluaciju nula, inače bi dobili NAND vrata koja netrebamo.

Za što koristimo AND vrata?
U algoritmu (4.poglavlje) AND služi da bi za „podprobleme“ koji su nezavisni jedni od drugog, dobili odgovor u kojem slučaju svi ti podproblemi daju 1 na izlazu.
Na primjer, imamo li za ulaze a i b qubit.

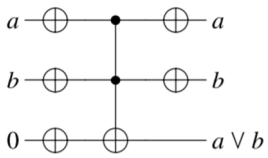
Prvi sklop će nešto izračunati i dati izlaz na qubit 3, drugi će isto nešto izračunati pomoću a i b i slati izlaz na 4. qubit. Mi ćemo na petom dobiti 1 samo kada oba sklopa daju 1 na svojim izlazima 3,4.



AND gate

2.2 AND vrata

Pošto nam na izlazu treba ,za 2 ulaza A i B izlaz A+B , možemo lako dobiti što tražimo koristeći Demorgan pravilo $A + B = \overline{\bar{A}\bar{B}}$. Vrata ćemo implementirati opet pomoću Toffolijevih vrata.



OR gate

Ta vrata će zapravo biti podproblemi iz (2.1). Kad oni daju neki rezultat na svojim izlazima, AND vrata će ih pomnožiti i dati 0 ili 1 za svaki a,b ulaz qubita.

3. Groverov algoritam na primjeru (Jupyter notebook – Chapter 1)

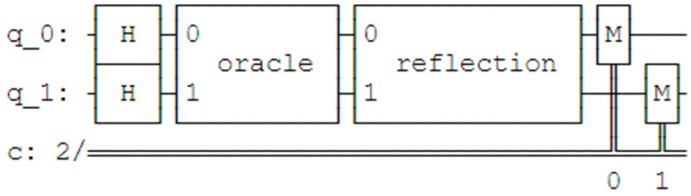
Prvi dio projekta je na jednostavnom primjeru implementirati Groverov algoritam. Detaljnija dokumentacija koda piše u Jupyter bilježnici,tu je zapisan opis problema.

Neka je zadana funkcija koja daje kao rezultat stanje obrnute amplitude ako je 11 na ulazu, a inače ništa.

$$f(x) = \begin{cases} -11, \mathrm{x} = 11 \\ \mathrm{x}\,, \text{inače} \end{cases}$$

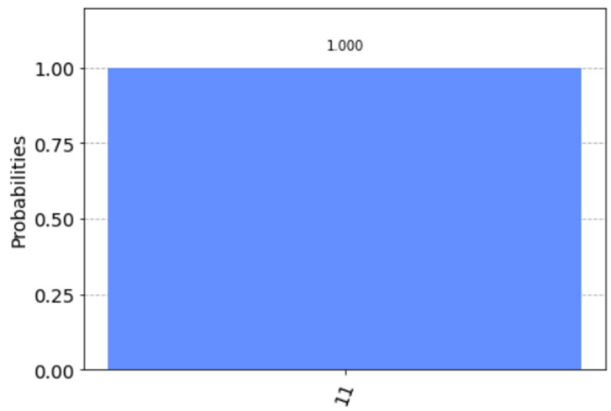
Kontrolirana Z vrata predstavljaju takvu proročicu u primjeru.Stanje 11 čemo označiti kao rezultat koji tražimo (tako smo definirali oracle).

Koji je cilj? S minimalno poziva f naći taj rezultatni vektor.



Dovoljan broj poziva proročice je 1, jer algoritam u samo jednom koraku okrene početni vektor za 90 stupnjeva.

```
plot_histogram(job.result().get_counts())
```



Rješenje je ispravno, algoritam daje 11 s vjerojatnošću 100 posto.

4. Algoritam na naprednijem primjeru – evaluacija logičkih tvrdnji (Jupyter notebook – Chapter 3)

Željeli bi za proizvoljnu logičku tvrdnju koja u sebi sadrži n nekih logičkih varijabli,evaluirati ju te naći za koje izbore varijabli tvrdnja rezultira istinom/laži.

Logičke varijable će zapravo biti qubiti u našem algoritmu, a proizvoljna tvrdnja će modelirati neku proročicu.

4.1 Ulaz

Za zadani ulaz koji se nalazi u konjuktivnoj normalnoj formi, tj oblika je:

$A_1 \wedge A_2 \wedge \ldots \wedge A_n$,

gdje je svaki A_i elementarna disjunkcija:

$A_i = B_1 \vee B_2 \vee \ldots \vee B_m$, $B_i =$ logička varijabla ili njena negacija.

Primjer takvog ulaza: $(A \vee B) \wedge (\neg A \vee \neg B) \wedge (A)$

Koristit ćemo kompaktiji zapis koji ćemo koristiti u algoritmu.

Napravit ćemo polje elementarnih disjunkcija P=[] i u njega staviti svaku disjunkciju po pravilu:

Ako varijabla B_i nastupa bez negacije, tada je u članu te elementarne disjunkcije i-ti član 0.

Ako varijabla B_i nastupa sa negacijom, tada je jednak 1.

Ako u toj elementarnoj disjunkciji B_i nije prisutna, tada pišemo -1.

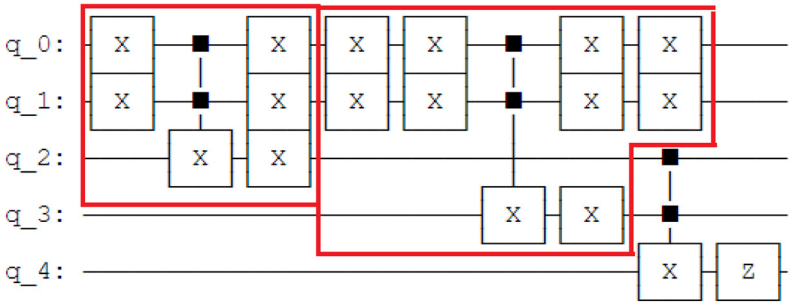
Tada P za gornji primjer izgleda: $[\,[0,0]\,,[1,1\,]\,,[0\,,-1]\,]$.

4.2 Implementacija proročice

Koliko nam qubita ukupno onda treba (osim n ulaznih)?

Za svaku elementarnu disjunkciju treba nam dodatni qubit za rezultat tog podizraza, te nam na kraju treba dodatni qubit da bi pomoću toffolijevih vrata mogli pomnožiti sve te podizraze od svake disjunkcije.

Uzmimo primjer: : $[\,[0,0]\,,[1,1\,]\,]$



Treći qubit je rezultat prve disjunktcije ,treći qubit druge , te na kraju množimo ta dva rezultata te okrećemo amplitudu pomoću Z vrata. Disjunkcije su tako implementirane da nakon računa ulazni qubiti budu vraćeni u početna stanja (paran broj X vrata).

4.3 Implementacija reflekcije

Pametna implementacija sklopa za refleksiju je dana već u Quiskit dokumentaciji:

```
def diffuser(nqubits):
    qc = QuantumCircuit(nqubits)
    # Apply transformation |s> -> |00..0> (H-gates)
    for qubit in range(nqubits):
        qc.h(qubit)
    # Apply transformation |00..0> -> |11..1> (X-gates)
    for qubit in range(nqubits):
        qc.x(qubit)
    # Do multi-controlled-Z gate
    qc.h(nqubits-1)
    qc.mct(list(range(nqubits-1)), nqubits-1)  # multi-controlled-toffoli
    qc.h(nqubits-1)
    # Apply transformation |11..1> -> |00..0>
    for qubit in range(nqubits):
        qc.x(qubit)
    # Apply transformation |00..0> -> |s>
    for qubit in range(nqubits):
        qc.h(qubit)
    # We will return the diffuser as a gate
    U_s = qc.to_gate()
    U_s.name = "U$_s$"
    return U_s
```

Sklop izrađuje QuantumCircuit objekt s ulazom od n qubita.

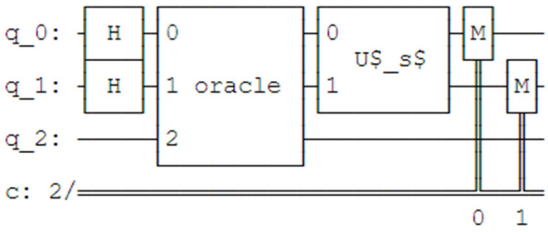
Na te qubite primjenjuje se H vrata, zatim X vrata na svaki qubit.

Zatim se primjenjuje kontrolirana Z vrata koja su sa svake strane ograđena s H vratima na zadnjem qubitu, te opet X i H na svim vratima.

Time je složen sklop za refleksiju za proizvoljno mnogo ulaza.

4.4 Rezultat

Završni krug se sastoji od H vrata na ulazu ,proročice ,te operatora refleksije.



Zanima nas za koje ulaze logički sud daje istinu odnostno laž,pa smo prokazali rješenje pomoću histograma. Točna rješenja koja proročica ponudi su ona čija je zarotirana i uvećana amplituda nakon djelovanja reflekcije, pa je evidentno iz grafa za koje ulaze je sklop točan.