Wykorzystanie metod ewolucyjnych sztucznej inteligencji w projektowaniu algorytmów kwantowych

Robert Nowotniak

Wydział Fizyki Technicznej, Informatyki i Matematyki Stosowanej
Politechnika Łódzka

3 czerwca 2008

Przedmowa

Przedmiotem niniejszej pracy są obliczenia kwantowe oraz systemy hybrydowe, łączące metody ewolucyjne sztucznej inteligencji oraz informatykę kwantową.

Programowanie przyszłych komputerów kwantowych będzie wymagającym zadaniem z powodu unikalnych, nieintuicyjnych efektów mechaniki kwantowej. W różnych jego aspektach mogą być pomocne metody ewolucyjne, programowanie i algorytmy genetyczne. W niniejszej pracy zostanie to zaprezentowane na przykładzie protokołu teleportacji kwantowej, kodowania supergęstego, algorytmu Grovera oraz generowania stanów splątanych.

Poznanie zasad działania przyszłych komputerów kwantowych oraz zrozumienie podstaw informatyki kwantowej były dla autora niniejszej pracy pasjonującym wyzwaniem podczas wielu ostatnich miesięcy. Przezwyciężenie wielu napotkanych trudności nie byłoby możliwe bez życzliwej pomocy wielu osób, którym niniejszym serdecznie dziękuję!

Robert Nowotniak, Łódź, 2008

Spis treści

Wprowadzenie

Informatyka kwantowa zajmuje się wykorzystaniem możliwości obliczeniowych układów, podlegających prawom mechaniki kwantowej. Od czasu powstania pierwszych koncepcji komputera kwantowego ([Fey82], [Deu85]) stało się jasne, że takie maszyny byłyby w stanie rozwiązywać niektóre problemy algorytmiczne w sposób znacznie bardziej *efektywny*, dzięki wykorzystaniu praw mechaniki kwantowej.

Zgodnie z prawem Moore'a średnio co dwa lata podwaja się liczba pojedynczych elementów elektronicznych, które mogą być umieszczone w układach scalonych, powodując tym samym wykładniczy przyrost dostępnych mocy obliczeniowych. Postępująca minimalizacja elektronicznych układów logicznych powoduje jednak, że wielkość pojedynczych elementów zbliży się w pewnej chwili do skali atomowej, a wówczas projektowanie takich układów będzie wymagało uwzględnienia nie tylko praw fizyki klasycznej, ale także praw mechaniki kwantowej.

Model obliczeń kwantowych jest w silny sposób związany z fizycznym aspektem realizacji samego procesu obliczeniowego. Jednym z dowodów świadczących o falowej naturze światła jest eksperyment fizyczny, w którym światło przechodzi przez dwie szczeliny, tworząc obraz w postaci prążków interferencyjnych. Foton porusza się *jednocześnie* dwiema ścieżkami, a ostateczny efekt stanowi interferencję funkcji falowych związanych z tym fotonem, dla każdej z dróg. Analogiczna własność jest wykorzystywana w obliczeniach kwantowych (ang. *quantum computing*). Komputer kwantowy mógłby wykonywać obliczenia w nietypowy, współbieżny sposób. Ogromna liczba obliczeń, może nawet nieskończona, może być wykonywana równolegle. Wyniki tych obliczeń ostatecznie oddziałują na siebie, podobnie jak w przypadku interferencji funkcji falowych fotonu, poruszającego się jednocześnie dwiema drogami. Na końcowy wynik ma zatem wpływ, w pewnym stopniu, każdy z pojedynczych, prowadzonych współbieżnie, rezultatów obliczeń.

Podobnie jak w przypadku klasycznej teorii obliczeń, obliczenia kwantowe mogą być wykonywane i analizowane z wykorzystaniem różnych modeli formalnych. Badając teoretyczne możliwości obliczeniowe i algorytmiczne

współczesnych maszyn, a także urządzeń które powstaną kiedykolwiek, posługujemy się modelami takimi jak rachunek lambda Churcha, model bramek logicznych lub — najbardziej powszechny — model maszyny Turinga. Przyjmuje się — zgodnie z *hipotezą Churcha-Turinga* — że jeśli dane zadanie jest obliczalne w sposób algorytmiczny (istnieje dla niego „efektywna procedura"[Fey96]), to jest ono także obliczalne w modelu maszyny Turinga. Dodatkowo rozpatrywanie zadań obliczeniowych z punktu widzenia różnych modeli formalnych wiąże się w każdym przypadku z niewiększym niż wielomianowym wzrostem złożoności. Oznacza to, że wszystkie możliwe problemy obliczeniowe należą do jednoznacznie ustalonych klas złożoności, niezależnie od modelu obliczeń, w którym są rozpatrywane.

Wraz z pojawieniem się koncepcji komputera kwantowego, pojawiły się jednak nowe, potencjalne, zupełnie nieoczekiwane możliwości maszyn obliczeniowych, które stały się przyczyną dużego zainteresowania informatyką kwantową w świecie naukowym. Model obliczeń kwantowych może być pierwszym poważnym kontrargumentem, pozwalającym zakwestionować hipotezę Churcha-Turinga. Powstanie komputerów kwantowych nie oznaczałoby po prostu dalszego zwiększenia osiągalnych mocy obliczeniowych o pewien współczynnik – choćby bardzo duży – jak ma to miejsce współcześnie, lecz oznaczałoby powstanie modelu obliczeniowego, który miałby większe możliwości niż jakakolwiek maszyna Turinga, niż uniwersalna maszyna Turinga lub niż jakikolwiek model równoważny wielomianowo dowolnemu spośród modeli klasycznych. Komputer kwantowy posiadałby wykładniczą przewagę nad wszystkimi klasycznymi modelami obliczeniowymi, jak deterministyczne lub probabilistyczne maszyny Turinga. Taka maszyna obliczeniowa byłaby w stanie rozwiązywać *niektóre* problemy obliczeniowe spoza klasy P w czasie wielomianowym. Przykładem takiego algorytmu jest algorytm Shora [Sho94], pozwalający na faktoryzację liczb w czasie wielomianowym. Jak dotąd nie udało się ustalić, czy istnieje algorytm klasyczny, rozwiązujący to zadanie ze złożonością mniejszą niż wykładnicza.

Możliwość faktoryzacji liczb w czasie wielomianowym przez komputer kwantowy stanowiłaby poważne zagrożenie dla niektórych używanych obecnie kryptosystemów. Używany współcześnie powszechnie kryptosystem RSA opiera się na założeniu, że rozkład dużych liczb na czynniki pierwsze jest bardzo kosztownym obliczeniowo, czasochłonnym zadaniem. Możliwości informatyki kwantowej byłyby rewolucyjne nie tylko z punktu widzenia kryptoanalizy, ale dostarczałyby także nowych, unikalnych metod szyfrowania. Dzięki efektom mechaniki kwantowej nie byłby możliwy podsłuch kanału komunikacji bez ingerencji w jego stan (np. kwantowa dystrybucja klucza [BB84]).

Czy w kwantowym modelu obliczeniowym byłby obliczalny w czasie wielomianowym jakikolwiek problem z klasy problemów NP-zupełnych? Jak dotąd nie udało się znaleźć odpowiedzi na to pytanie. Gdyby było to możliwe, komputery kwantowe posiadałyby potęgę rozwiązywania dowolnego problemu z klasy NP w czasie wielomianowym. Model obliczeń kwantowych wykorzystywałby bezpośrednio najdokładniejszy znany opis fizycznej rzeczywistości, wszechświata w którym żyjemy.

Oprócz problemów algorytmicznych rozpatruje się także możliwości wykorzystania obliczeń kwantowych z punktu widzenia teorii informacji (protokół teleportacji kwantowej [BBC+93], kodowanie supergęste [BW92]) oraz z punktu widzenia teorii gier [Sum05, Sum06, Kli04].

Obliczenia kwantowe mogą być opisywane za pomocą różnych modeli formalnych, podobnie jak ma to miejsce w przypadku klasycznej teorii obliczeń. Podstawowym z nich jest model *kwantowej maszyny Turinga* – dodającej odpowiednie, kwantowe „rozmycie" do klasycznej maszyny Turinga. Na każdym odcinku taśmy kwantowej maszyny Turinga może się znajdować superpozycja wszystkich możliwych symboli, które byłyby przetwarzane w sposób współbieżny, zgodny z ewolucją unitarną układu kwantowego. Podobnie głowica takiej maszyny może się znajdować równocześnie w wielu położeniach. Dowolny algorytm kwantowy może być wykonany za pomocą tej abstrakcyjnej maszyny. W niniejszej pracy został wykorzystany równoważny, uniwersalny model kwantowych bramek logicznych tj. model obwodów kwantowych (ang. *quantum circuits*) – pozwalający na dużo prostszy opis obliczeń i algorytmów.

Istnieje obecnie bardzo dużo potencjalnych „ścieżek", dających nadzieję na powstanie w przyszłości *skalowalnego* komputera kwantowego. Wymagane jest w tym celu wykorzystanie miniaturowych układów, mogących m.in. przechowywać „delikatny" stan koherencji kwantowej. Do takich zjawisk fizycznych należą m.in. magnetyczny rezonans jądrowy, polaryzacja światła, kropki kwantowe, pułapki jonowe, stany energetyczne elektronów na powłokach elektronowych oraz wiele innych. W roku 2001 zespół z ośrodka badawczo-rozwojowego IBM Almaden Research Center przeprowadził pomyślny eksperyment z prymitywnym, siedmiokubitowym komputerem kwantowym, opartym na magnetycznym rezonansie jądrowym. Za jego pomocą udało się wykonać pierwszą, fizyczną implementację algorytmu Shora. Największą obecnie przeszkodą w zbudowaniu skalowalnego komputera kwantowego jest trudność z utrzymaniem koherencji wraz ze zwiększaniem rozmiaru rejestrów kwantowych.

Kolejnym problemem związanym z komputerem kwantowym będzie trudność w jego programowaniu — model obliczeń kwantowych jest dużo bardziej złożony niż w przypadku obliczeń klasycznych, głównie z powodu nieintuicyj-

nych efektów mechaniki kwantowej. W ciągu kilkunastu ostatnich lat rozwoju informatyki kwantowej znaleziono jedynie około kilkunastu algorytmów kwantowych. Niektóre z nich zostaną przedstawione w niniejszej pracy. Trudność projektowania algorytmów kwantowych wynika z niewielkiej analogii pomiędzy algorytmami kwantowymi i klasycznymi — co zostanie dokładnie omówione na przykładzie działania algorytmu Grovera [Gro96] — oraz własności, którym one podlegają. Do takich unikalnych własności należą: kwantowy paralelizm, superpozycja stanów, kwantowe splątanie oraz interferencja amplitud prawdopodobieństwa, będących liczbami zespolonymi.

Projektowanie obliczeń kwantowych jest zadaniem bardzo trudnym z powodu tych nieintuicyjnych efektów. Algorytmy kwantowe w całkowicie odmienny sposób przetwarzają przestrzeń rozważań niż algorytmy klasyczne. Są to algorytmy probabilistyczne, wykorzystują takie własności jak kwantowa superpozycja i paralelizm.

Jednym z głównych zadań sztucznej inteligencji jest wspomaganie człowieka w rozwiązywaniu trudnych problemów lub wyręczanie człowieka w wykonywaniu pracochłonnych zadań — szczególnie takich, w których wymagane są umiejętności łatwo dostępne jedynie człowiekowi. W ciągu ostatnich lat pojawiły się liczne prace nad systemami hybrydowymi, w których próbuje się łączyć różnorodne metody sztucznej inteligencji (m.in. sieci neuronowe, sieci bayesowskie, systemy rozmyte, metody heurystyczne) z możliwościami oferowanymi przez informatykę kwantową. Możliwe tego typu podejścia zostały przedstawione m.in. w [AT04, Sga07].

W niniejszej pracy zostaną przedstawione możliwości automatycznego projektowania elementów algorytmów kwantowych za pomocą metod ewolucyjnych, algorytmów i programowania genetycznego. Zostanie wykazane, że są to skuteczne narzędzia w rozwiązywaniu tego typu zadań. Metody ewolucyjne mogą być wykorzystane na różnych etapach projektowania obliczeń kwantowych. Mogą służyć do projektowania pojedynczych bramek kwantowych oraz całych obwodów kwantowych. Za ich pomocą udało się już zaprojektować lepsze — mniej złożone — obwody niż równoważne obwody zaprojektowane przez człowieka [YI00, Rub00].

Oprócz systemów tego typu rozważano także podejście odwrotne, tzn. algorytmy genetyczne, które bezpośrednio czerpałyby z możliwości obliczeń kwantowych [RSFAF01, HPLK01, HK00]. W pamięci komputera kwantowego mógłby być reprezentowany chromosom, w którym na każdym locusie znajduje się superpozycja możliwych wartości genu. Przetwarzanie takiego chromosomu, zgodnie z ewolucją unitarną, oznaczałoby przetwarzanie przez komputer kwantowy *jednocześnie* całej przestrzeni rozważań.

Ogólna klasyfikacja i analiza możliwych systemów hybrydowych, łączących metody ewolucyjne i obliczenia kwantowe, zostały przedstawione w pracy [GPT04].

Prawdopodobny dynamiczny rozwój informatyki kwantowej w przyszłości może mieć także ewentualny wpływ na lepsze zrozumienie wielu nierozwiązanych dotąd problemów sztucznej inteligencji. Czy komputery kiedykolwiek będą w stanie symulować pracę systemów tak złożonych jak ludzki umysł? Według początkowej koncepcji zwolenników tzw. silnej sztucznej inteligencji, własności takie jak świadomość, myślenie abstrakcyjne w naturalny sposób *wyłoniłyby* się w chwili odpowiedniego wzrostu wyrafinowania algorytmów wykonywanych przez maszyny obliczeniowe. Dotychczasowy rozwój badań nad sztuczną inteligencją nie wskazuje na to, by mogło okazać się to prawdą. Ludzki umysł, działając w sposób niealgorytmiczny, posiada zdolności niedostępne maszynom obliczeniowym, równoważnym wielomianowo maszynie Turinga, i których zdolności automatycznego wnioskowania ograniczone są przez twierdzenie Gödla. Według niektórych badaczy (Roger Penrose, Stuart Hameroff) efekty kwantowe mogą być nieodzowne do pełnego zrozumienia procesów, zachodzących w mózgach istot, obdarzonych inteligencją. Kluczowa rola *pomiaru* w mechanice kwantowej, powodująca redukcję funkcji falowej, wpływającego na rzeczywistość i kreująca ją, może w nietypowy sposób wiązać się z cechą świadomości[Pen89, Ham06].

Wszystkie przedstawione powyżej rozważania prowadzą do wniosku, że systemy hybrydowe, łączące metody sztucznej inteligencji oraz możliwości oferowane przez obliczenia kwantowe, mogą stać się w przyszłości bardzo obiecującym kierunkiem badań.

Cel i zakres pracy

Celem niniejszej pracy było stworzenie środowiska symulacji obliczeń kwantowych, zaimplementowanie podstawowych algorytmów kwantowych oraz zaprezentowanie możliwości wykorzystania metod ewolucyjnych w projektowaniu elementów takich algorytmów.

W pracy został zaproponowany model obiektowy dla obliczeń kwantowych oraz przygotowane zostało środowisko symulacji obliczeń, oparte na zaproponowanym modelu. Została w tym celu stworzona biblioteka *qclib*, wykorzystująca technikę programowania zorientowanego obiektowo.

Rozdział 1 niniejszej pracy przedstawia teoretyczne podstawy informatyki kwantowej. Zostały w nim opisane podstawowe dla informatyki kwantowej obiekty matematyczne oraz postulaty mechaniki kwantowej, której własności determinują możliwości kwantowego modelu obliczeniowego.

W rozdziale 2 zostały przedstawione podstawowe jednostki informacji w informatyce kwantowej oraz model obliczeniowy kwantowych bramek logicznych. Rozdział przedstawia zasady symulacji działania obwodów kwantowych.

W rozdziale 3 został przedstawiony ogólny model obiektowy dla obliczeń kwantowych, za pomocą którego można symulować pracę komputera kwantowego w dowolnym obiektowym języku programowania. Zaproponowany model został zaimplementowany w języku Python, a rozdział 3 opisuje interfejs programistyczny stworzonej biblioteki *qclib*, która będzie wykorzystywana w późniejszych rozdziałach.

Rozdział 4 opisuje podstawowe efekty informatyki kwantowej, wykorzystywane w algorytmach. Znajduje się w nim omówienie stanów splątanych, paradoksu Einsteina-Podolskiego-Rosena oraz własności kwantowego paralelizmu.

W rozdziale 5 opisane zostały cztery podstawowe algorytmy kwantowe: generowanie stanów splątanych, protokół teleportacji kwantowej, kodowanie supergęste oraz algorytm Grovera.

Rozdział 6 przedstawia możliwości wykorzystania metod ewolucyjnych sztucznej inteligencji w różnych aspektach projektowania elementów algo-

rytmów kwantowych. Wykonane eksperymenty numeryczne ilustrują kilka możliwych podejść. W rozdziale zostały przedstawione podstawowe metody ewolucyjne – algorytmy genetyczne, programowania genetyczne – w kontekście projektowania bramek oraz całych obwodów kwantowych.

W pracy został wykorzystany język programowania *Python* oraz biblioteka obliczeń numerycznych *Numerical Python*. Zaimplementowano moduł *qclib.py*, który pozwala na symulację obliczeń kwantowych w tym języku.

Biblioteka *Numerical Python* pozwala na wykonywanie obliczeń numerycznych, wszystkich które są niezbędne do symulacji pracy komputera kwantowego. Udostępnia wydajne operacje na macierzach, obliczanie wyznaczników, śladów oraz operacje z zakresu algebry liniowej. Zaproponowany model obiektowy może być jednak wykorzystany do opracowania analogicznej biblioteki obliczeń kwantowych w dowolnym języku programowania, pozwalającym na programowanie w sposób zorientowany obiektowo oraz zapewniającym odpowiedni zbiór operacji na macierzach liczb zespolonych.

Dla stworzonego modułu opracowano zestaw testów jednostkowych, dzięki którym była możliwa ciągła weryfikacja poprawności działania modułu oraz jego jednoczesna optymalizacja i refaktoryzacja kodu źródłowego. Testy jednostkowe zostały opracowane przy pomocy biblioteki *unittest*.

Schematy obwodów kwantowych były generowane za pomocą narzędzia qasm2circ, używanego przez autorów książek z zakresu informatyki kwantowej [NC00].

W całej niniejszej pracy będzie przyjęta konwencja, wg której najbardziej znaczące kubity (kubity najstarsze) będą rysowane od góry schematów, a najmniej znaczące — od dołu. Numerowanie elementów (np. kubitów w rejestrach kwantowych) wszędzie będzie się zaczynało od zera, o ile nie zaznaczono inaczej.

Rozdział 1

Podstawy teoretyczne

„If quantum mechanics hasn't profoundly shocked you, you haven't understood it yet"

Niels Bohr

Model obliczeń kwantowych i wizja zbudowania potężnych, skalowanych komputerów kwantowych w przyszłości opierają się na wykorzystaniu własności miniaturowych układów, rządzących się prawami mechaniki kwantowej. Szczegółowe wprowadzenie w teorię mechaniki kwantowej wykracza poza zakres niniejszej pracy — a także poza możliwości jej autora — jednak precyzyjna wiedza o fizycznych aspektach realizacji procesu obliczeniowego nie jest niezbędna do zrozumienia zasad przeprowadzania takiego rodzaju obliczeń. Dokładne wprowadzenie do mechaniki kwantowej znajduje się np. w [Lib87].

Do zapisu obliczeń kwantowych wykorzystywany jest formalizm przestrzeni Hilberta, którego podstawy będą przedstawione w niniejszym rozdziale.

1.1 Podstawy matematyczne

Definicja 1.1. Przestrzeń Hilberta $\mathcal{H}$ to rzeczywista lub zespolona przestrzeń wektorowa z wprowadzoną operacją iloczynu skalarnego $\mathcal{H} \times \mathcal{H} \to \mathbb{C}$ (lub $\mathbb{R}$), zupełna ze względu na normę wyznaczoną przez ten iloczyn.

Iloczyn skalarny w powyższej definicji jest też nazywany **iloczynem wewnętrznym** (ang. *inner product*), będącym uogólnieniem standardowego iloczynu skalarnego (ang. *dot product*) w przestrzeniach euklidesowych.

Wszystkie n-wymiarowe przestrzenie Hilberta są izomorficzne, więc każdą taką przestrzeń można oznaczać po prostu jako $\mathcal{H}_n$. W mechanice kwantowej wykorzystuje się zazwyczaj nieskończenie wymiarową przestrzeń Hilberta

$\mathcal{H}_\infty$, jednak w informatyce kwantowej zakłada się skończony rozmiar pamięci komputera kwantowego, używa się więc przestrzeni o skończonej liczbie wymiarów.

Definicja 1.2. Dowolną funkcję $\langle \cdot, \cdot \rangle : \mathcal{H} \times \mathcal{H} \to \mathbb{C}$ nazywamy **iloczynem wewnętrznym**, jeśli spełnia ona warunki:

1. $\langle \cdot, \cdot \rangle$ jest liniowa ze względu na pierwszy z argumentów
2. $\langle u, v \rangle = \langle v, u \rangle^*$ (sprzężona symetria)
3. $\langle u, v \rangle \geqslant 0$
4. $\langle v, v \rangle = 0 \iff v = 0$

W informatyce kwantowej iloczyn wewnętrzny wektorów u i v definiuje się jako sumę:

$$\langle u, v \rangle = \langle u | v \rangle = \sum_i u_i^* v_i \tag{1.1}$$

gdzie $u_i^* \in \mathbb{C}$ oznacza sprzężenie zespolone i-tej współrzędnej wektora u.

Iloczyn wewnętrzny pozwala w łatwy sposób zdefiniować **normę wektora**:

$$||u|| = \sqrt{\langle u, u \rangle} \tag{1.2}$$

Pojęcie *zupełności* w definicji 1.1 oznacza, że każdy zbieżny ciąg wektorów, należących do przestrzeni, posiada granicę, która także należy do tej przestrzeni.

Definicja 1.3. Przestrzeń wektorowa $\mathcal{H}$ jest **zupełna**, jeśli dla każdego zbieżnego ciągu wektorów u_i, tzn. takiego że

$$\lim_{m,n \to \infty} ||u_m - u_n|| = 0 \tag{1.3}$$

istnieje wektor, należący do tej przestrzeni, $u \in \mathcal{H}$, taki że

$$\lim_{m \to \infty} ||u_m - u|| = 0 \tag{1.4}$$

Sprzężenie hermitowskie macierzy $A_{m \times n}$ to macierz transponowana sprzężeń zespolonych elementów macierzy A. Tzn. dla macierzy:

$$A_{m \times n} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

jej sprzężeniem hermitowskim jest:

$$A^{\dagger}_{m\times n} = \left(\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \right)^{\dagger} = \begin{bmatrix} a^*_{11} & a^*_{21} & \cdots & a^*_{m1} \\ a^*_{12} & a^*_{22} & \cdots & a^*_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ a^*_{1n} & a^*_{2n} & \cdots & a^*_{mn} \end{bmatrix}$$

gdzie a^* oznacza sprzężenie zespolone liczby $a \in \mathbb{C}$.

Przykład 1.4. *Dla macierzy kwadratowej:*

$$A = \begin{bmatrix} 2i & 1+3i \\ -1 & 2 \end{bmatrix}$$

jej sprzężeniem hermitowskim jest:

$$A^{\dagger} = \begin{bmatrix} -2i & -1 \\ 1-3i & 2 \end{bmatrix}$$

Definicja 1.5. Macierz unitarna to macierz kwadratowa o elementach rzeczywistych lub zespolonych, która pomnożona przez sprzężenie hermitowskie niej samej, daje macierz jednostkową:

$$U\ U^{\dagger} = U^{\dagger}U\ = I_n \tag{1.5}$$

Zarówno wiersze jak i kolumny macierzy unitarnej tworzą ortonormalne zbiory wektorów.

Własność 1.6. *Macierz unitarna posiada macierz odwrotną $U^{\dagger}$ równą sprzężeniu hermitowskiemu jej samej, czyli:*

$$U^{\dagger} = U^{-1}.$$

Przykład 1.7. *Macierz $H_2 = \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \end{bmatrix}$ jest macierzą unitarną, ponieważ wykonując mnożenie macierzy $H_2 H_2^{\dagger}$, otrzymuje się macierz jednostkową:*

$$\begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \end{bmatrix} \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \end{bmatrix}^{\dagger} = \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \end{bmatrix} \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \end{bmatrix} = \begin{bmatrix} \frac{1}{2}+\frac{1}{2} & \frac{1}{2}-\frac{1}{2} \\ \frac{1}{2}-\frac{1}{2} & \frac{1}{2}+\frac{1}{2} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{1.6}$$

Macierz z przykładu (1.7) będzie się pojawiała w dalszej części pracy wielokrotnie, ponieważ opierać się na niej będzie kwantowa bramka Hadamarda.

Definicja 1.8. Macierz hermitowska (ang. *Hermitian matrix*) to macierz kwadratowa $A_{n\times n}$, równa swojemu sprzężeniu hermitowskiemu:

$$A = A^{\dagger} \tag{1.7}$$

Macierz hermitowska nazywana jest też **macierzą samosprzężoną** (ang. *self-adjoint matrix*).

Przykład 1.9. *Macierz*

$$A = \begin{bmatrix} 1 & 3+i \\ 3-i & 5 \end{bmatrix}$$

jest macierzą hermitowską, ponieważ

$$A^{\dagger} = \begin{bmatrix} 1 & 3+i \\ 3-i & 5 \end{bmatrix}^{\dagger} = \begin{bmatrix} 1 & 3+i \\ 3-i & 5 \end{bmatrix} = A$$

Macierze *unitarne* ($A\ A^{\dagger} = I$) i *hermitowskie* ($A = A^{\dagger}$) posiadają różne własności, będą wykorzystywane przy różnych operacjach wykonywanych na układach kwantowych i ważne jest ich rozróżnienie.

Definicja 1.10. Iloczyn Kroneckera. Dla dowolnych macierzy

$$A_{r\times s} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1s} \\ a_{21} & a_{22} & \cdots & a_{2s} \\ \vdots & \vdots & \ddots & \vdots \\ a_{r1} & a_{r2} & \cdots & a_{rs} \end{bmatrix} \qquad B_{t\times u} = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1u} \\ b_{21} & b_{22} & \cdots & b_{2u} \\ \vdots & \vdots & \ddots & \vdots \\ b_{t1} & b_{t2} & \cdots & b_{tu} \end{bmatrix}$$

ich iloczyn Kroneckera definiuje się jako macierz blokową

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

Przykład 1.11. *Niech będą dane macierze A oraz B:*

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \qquad B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

Wówczas iloczyn Kroneckera tych macierzy wynosi:

$$A \otimes B = \begin{bmatrix} 1\cdot 5 & 1\cdot 6 & 2\cdot 5 & 2\cdot 6 \\ 1\cdot 7 & 1\cdot 8 & 2\cdot 7 & 2\cdot 8 \\ 3\cdot 5 & 3\cdot 6 & 4\cdot 5 & 4\cdot 6 \\ 3\cdot 7 & 3\cdot 8 & 4\cdot 7 & 4\cdot 8 \end{bmatrix} = \begin{bmatrix} 5 & 6 & 10 & 12 \\ 7 & 8 & 14 & 16 \\ 15 & 18 & 20 & 24 \\ 21 & 24 & 28 & 32 \end{bmatrix}$$

Iloczyn Kroneckera macierzy jest szczególnym przypadkiem **iloczynu tensorowego**, który oznacza ogólny operator biliniowy, określany dla wektorów, macierzy lub przestrzeni wektorowych. Iloczyn Kroneckera jest zatem operacją łączną oraz liniową ze względu na obydwa argumenty.

Operację iloczynu tensorowego można posłużyć się do rozpięcia przestrzeni stanów układu składającego się z kilku podukładów. Jeśli $\mathcal{H}_n$ i $\mathcal{H}_m$ są przestrzeniami Hilberta o wymiarach n i m, to $\mathcal{H}_n \otimes \mathcal{H}_m$ jest przestrzenią Hilberta o wymiarze nm.

W informatyce kwantowej stosuje się powszechnie **notację Diraca**, która będzie używana także w tej pracy. Notacja pozwala na wygodny, krótki zapis wektorów kolumnowych i wierszowych, które są bardzo często wykorzystywane w obliczeniach kwantowych.

Wektory kolumnowe oznacza się jako tzw. wektory *ket* np. $|\phi\rangle$ (czyt. ket ϕ), gdzie ϕ to dowolny symbol, pozwalający odróżniać wektory ket od siebie. Wektory wierszowe oznacza się natomiast jako tzw. wektory *bra* np. $\langle\phi|$ (czyt. bra ϕ).

Pomiędzy wektorami ket i bra zachodzi zależność

$$|\phi\rangle^\dagger = \langle\phi| \tag{1.8}$$

Oznacza to, że sprzężenie hermitowskie wektora $|\phi\rangle$ oznaczamy jako $\langle\phi|$. Zgodnie z tą konwencją iloczyn wektorów $\langle u|$ i $|v\rangle$ to domyślny iloczyn skalarny, wykorzystywany w informatyce kwantowej (wzór (1.1))

$$\langle u||v\rangle = \langle u|v\rangle = \sum_i u_i^* v_i$$

Zamiast nazwy wektora w nawiasach $\langle\cdot|$ lub $|\cdot\rangle$ mogą się także pojawiać literalne wartości liczbowe, np. $|0\rangle$ lub $|1\rangle$.

Zapis $|n\rangle$, gdzie $n \in \mathbb{N}$ (n jest zazwyczaj zapisane w systemie binarnym), oznacza wektor kolumnowy posiadający wartość 1 na n-tej pozycji (licząc od 0) oraz 0 na wszystkich pozostałych miejscach.

Przykład 1.12. *Zapisy* $|0\rangle$, $|1\rangle$, $|101\rangle$ *i* $\langle 11|$ *oznaczają odpowiednio wektory:*

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$|101\rangle = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \qquad \langle 11| = |11\rangle^\dagger = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}$$

Wymiar przestrzeni, do której należy wektor (czyli liczba jego elementów), wynika zazwyczaj z kontekstu lub długości ciągu binarnego w nawiasach $\langle\cdot|$ lub $|\cdot\rangle$.

Możemy też używać skróconego zapisu dziesiętnego np. $|101\rangle = |5\rangle$. Zauważmy także, że wektory $\{|0\rangle, |1\rangle\}$ tworzą *bazę kanoniczną* dwuwymiarowej przestrzeni $\mathcal{H}_2$, wektory $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$ bazę kanoniczną przestrzeni $\mathcal{H}_4$ itd.

Własność 1.13. *Iloczyn tensorowy stanów bazowych $|\psi\rangle$ i $|\phi\rangle$ bazy kanonicznej, zapisanych w postaci binarnej, jest stanem bazowym w postaci konkatenacji $|\psi\phi\rangle$.*

Przykład 1.14. $|1\rangle \otimes |101\rangle = |1\rangle|101\rangle = |1101\rangle$, *ponieważ (z definicja iloczynu tensorowego (1.10)):*

$$|1\rangle \otimes |101\rangle = |1\rangle|101\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0\cdot 0 \\ 0\cdot 0 \\ 0\cdot 0 \\ 0\cdot 0 \\ 0\cdot 0 \\ 0\cdot 1 \\ 0\cdot 0 \\ 0\cdot 0 \\ 1\cdot 0 \\ 1\cdot 0 \\ 1\cdot 0 \\ 1\cdot 0 \\ 1\cdot 0 \\ 1\cdot 1 \\ 1\cdot 0 \\ 1\cdot 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = |1101\rangle$$

Definicja 1.15. Śladem macierzy kwadratowej $A_{n\times n}$ (ang. *trace of a matrix*) nazywa się sumę elementów na głównej przekątnej macierzy.

$$tr(A) = \sum_{i=1}^{n} a_{ii} = a_{11} + a_{22} + \cdots + a_{nn} \tag{1.9}$$

Własność 1.16.

$$tr(|u\rangle\langle v|) = \langle u\ |v\rangle \tag{1.10}$$

1.2 Postulaty mechaniki kwantowej

Możliwości kwantowego modelu obliczeniowego są determinowane przez przez kwantowomechaniczny opis fizycznej rzeczywistości. W tej sekcji zostały przedstawione postulaty mechaniki kwantowej, zgodnie z którą działałby komputer kwantowy.

W ogólnym przypadku ewolucję czasową układu kwantowego opisuje równanie Schrödingera z czasem, będące równaniem różniczkowym zwyczajnym, jednakże w informatyce kwantowej zmiany stanów opisuje się w postaci dyskretnych przekształceń, odpowiadających kolejnym etapom obliczeń kwantowych.

Postulat 1. Przestrzenią stanów izolowanego układu fizycznego jest przestrzeń Hilberta $\mathcal{H}$, a stan układu opisywany jest przez jednostkowy wektor $|\psi\rangle \in \mathcal{H}$ z przestrzeni $\mathcal{H}$, nazywany *wektorem stanu*.

Postulat 2 Przestrzenią stanów złożonego układu fizycznego jest iloczyn tensorowy przestrzeni stanów układów składowych. W przypadku n układów, znajdujących się w stanach ϕ_1, ϕ_2, ..., ϕ_n, stan układu jako całości opisywany jest przez iloczyn tensorowy $|\phi_1\rangle \otimes |\phi_2\rangle \otimes \cdots \otimes |\phi_n\rangle$.

Postulat 3 Ewolucja czasowa stanu $|\psi\rangle$ izolowanego układu fizycznego opisywana jest przez równanie Schrödingera z czasem:

$$i\hbar \frac{\delta}{\delta t}|\psi\rangle = \hat{H}|\psi\rangle \tag{1.11}$$

gdzie $\hat{H}$ jest operatorem samosprzężonym, nazywanym *hamiltonianem* układu, w przestrzeni $\mathcal{H}$.

Ponieważ obliczenia kwantowe opisuje się za pomocą serii dyskretnych przekształceń, zamiast równania różniczkowego (1.11), dynamikę układu kwantowego można opisać prościej w następujący sposób:

Jeśli izolowany układ fizyczny w chwili czasu t_1 znajdował się w stanie $|\psi\rangle$, to w chwili czasu $t_2 \geqslant t_1$ stan układu $|\psi'\rangle$ może być opisany za pomocą operatora unitarnego $U\ = U(t_2 - t_1)$ jako:

$$|\psi'\rangle = U\ |\psi\rangle \tag{1.12}$$

Postulat 4. Pomiar *rzutowy*[1] (ang. *projective measurement*) stanu określony jest za pomocą pewnego operatora hermitowskiego M. Ponieważ macierze hermitowskie są macierzami normalnymi ($MM^\dagger = M^\dagger M$), operator M posiada rozkład spektralny

$$M = \sum_m m P_m \tag{1.13}$$

gdzie P_m jest operatorem rzutowym (ang. *projector*) na przestrzeń własną M, a wartości własne m odpowiadają możliwym do otrzymania wynikom pomiarów.

Prawdopodobieństwo otrzymania rezultatu m pomiaru stanu układu określone jest wyrażeniem:

$$p(m) = \langle \phi | P_m | \phi \rangle \tag{1.14}$$

Wraz z uzyskaniem rezultatu pomiaru m stan układu zostaje ustalony na

$$|\phi'\rangle = \frac{P_m |\phi\rangle}{\sqrt{p(m)}} \tag{1.15}$$

Stan bezpośrednio po pomiarze staje się więc jednym z wektorów własnych użytego operatora pomiarowego M.

[1] możliwe jest także bardziej ogólne sformułowanie operacji pomiaru (ang. *general measurement*), ale jest ono rzadko wykorzystywane w informatyce kwantowej

Rozdział 2

Model kwantowych bramek logicznych

> *„The theory of computation has traditionally been studied almost entirely in the abstract, as a topic in pure mathematics. This is to miss the point of it. Computers are physical objects, and computations are physical processes. What computers can or cannot compute is determined by the laws of physics alone, and not by pure mathematics."*
>
> *David Deutsch*

W niniejszym rozdziale zostanie przestawiony model obliczeniowy kwantowych bramek logicznych — model uniwersalny tzn. równoważny modelowi kwantowej maszyny Turinga, a zatem pozwalający na wykonywanie dowolnych algorytmów kwantowych i na symulację pracy komputera kwantowego.

Zostaną tu przedstawione podstawowe obiekty i jednostki informacji kwantowej oraz metody symulacji pracy obwodów kwantowych. Model obliczeniowy zostanie przedstawiony za pomocą formalizmu przestrzeni Hilberta, obiektów matematycznych wprowadzonych w sekcji 1.1 oraz własności mechaniki kwantowej, wynikających z postulatów z sekcji 1.2.

2.1 Kubity i rejestry kwantowe

Podstawową „klasyczną" jednostką logiczną w informatyce jest *bit*, mogący przyjmować jedną z spośród dwóch wartości logicznych: 0 albo 1. W informatyce kwantowej podstawową jednostką informacji są kubity (ang. *qubits*) i rejestry kwantowe (ang. *quantum registers*). Kubit (podobnie jak bit) może przyjmować wartości 0 albo 1, oznaczane w notacji Diraca odpowiednio $|0\rangle$

i $|1\rangle$ (co oznacza, że są to wielkości wektorowe a nie skalarne). Kubit może przyjmować także wartość będącą dowolną superpozycją tych dwóch stanów bazowych.

Definicja 2.1. Kubit (ang. *qubit*) to wektor znormalizowany w dwuwymiarowej, zespolonej przestrzeni Hilberta.

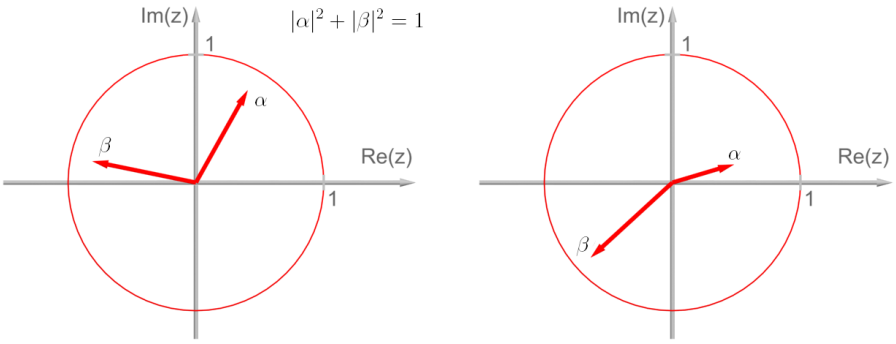
$$|\phi\rangle = \alpha|0\rangle + \beta|1\rangle$$
$$\alpha, \beta \in \mathbb{C} \qquad |\alpha|^2 + |\beta|^2 = 1$$

Liczby α i β nazywamy **amplitudami prawdopodobieństwa**.

Wektory $|0\rangle$ i $|1\rangle$ tworzą **bazę standardową** (lub **obliczeniową**) dwuwymiarowej, zespolonej przestrzeni stanów kubitu. Korzystając z notacji Diraca, stan kubitu można także zapisać jako wektor kolumnowy

$$|\phi\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$$

Jedna z możliwych geometrycznych reprezentacji kubitu przedstawiona jest na rysunku 2.1(a). Ponieważ amplitudy prawdopodobieństwa α i β reprezentowane są przez pojedyncze wektory na płaszczyźnie zespolonej, stan jednego kubitu reprezentuje para wektorów α i β — przy założeniu, że $|\alpha|^2 + |\beta|^2 = 1$.



(a) Stan kubitu (b) Inny stan kubitu

Rysunek 2.1: Kubit może być reprezentowany jako para wektorów na płaszczyźnie zespolonej

W wyniku pomiaru wartości kubitu[1], kubit przyjmuje wartość $|0\rangle$ (*ket 0*) z prawdopodobieństwem $|\alpha|^2$ lub wartość $|1\rangle$ (*ket 1*) z prawdopodobieństwem $|\beta|^2$.

[1] dotyczy to pomiaru wykonywanego w bazie standardowej

Własność 2.2. *Odczyt wartości kubitu nieodwracalnie niszczy jego stan. Stan kubitu zostaje ustalony na jeden z wektorów bazy pomiarowej.*

Układ wielu kubitów tworzy **rejestr kwantowy**, który — zgodnie z jednym z przedstawionych postulatów mechaniki kwantowej — może być rozpatrywany jako układ izolowany złożony z wielu układów składowych (poszczególne kubity należące do rejestru). Przestrzenią stanów takiego rejestru jest przestrzeń, będąca iloczynem tensorowym przestrzeni stanów poszczególnych kubitów. Stan n-kubitowego rejestru kwantowego jest zatem opisywany jako wektor znormalizowany w 2^n-wymiarowej, zespolonej przestrzeni Hilberta. Pozwala jednocześnie zapisać superpozycję 2^n różnych wartości.

W przypadku rejestru kwantowego, składającego się jedynie z dwóch kubitów $|q_0\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle$ i $|q_1\rangle = \alpha_2|0\rangle + \alpha_3|1\rangle$, stan rejestru jest opisywany jako iloczyn tensorowy (gdzie $\alpha_0, \alpha_1, \alpha_2, \alpha_3 \in \mathbb{C}$):

$$\begin{aligned} |q_0\rangle \otimes |q_1\rangle &= (\alpha_0|0\rangle + \alpha_1|1\rangle) \otimes (\alpha_2|0\rangle + \alpha_3|1\rangle) \\ &= \alpha_0\alpha_2|0\rangle \otimes |0\rangle + \alpha_0\alpha_3|0\rangle \otimes |1\rangle + \alpha_1\alpha_2|1\rangle \otimes |0\rangle + \alpha_1\alpha_3|1\rangle \otimes |1\rangle \\ &= \alpha_0\alpha_2|00\rangle + \alpha_0\alpha_3|01\rangle + \alpha_1\alpha_2|10\rangle + \alpha_1\alpha_3|11\rangle \end{aligned}$$

Pojedynczy kubit $|\psi\rangle \in \mathcal{H}_2$ reprezentowany jest jako para liczb zespolonych, układ dwukubitowy reprezentowany jest wektorem z przestrzeni $\mathcal{H}_4$. Stan trzykubitowego rejestru jest opisany przez wektor znormalizowany z przestrzeni $\mathcal{H}_8$ itd.

Pewien stan $|\phi\rangle$ trzykubitowego rejestru kwantowego może być zapisany w bazie standardowej $\{|000\rangle, |001\rangle, ..., |111\rangle\}$ jako:

$$|\phi\rangle = \alpha_0|000\rangle + \alpha_1|001\rangle + \ldots \alpha_7|111\rangle$$

Powyższy zapis w notacji Diraca jest równoważny wektorowi kolumnowemu:

$$|\phi\rangle = \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_7 \end{bmatrix}$$

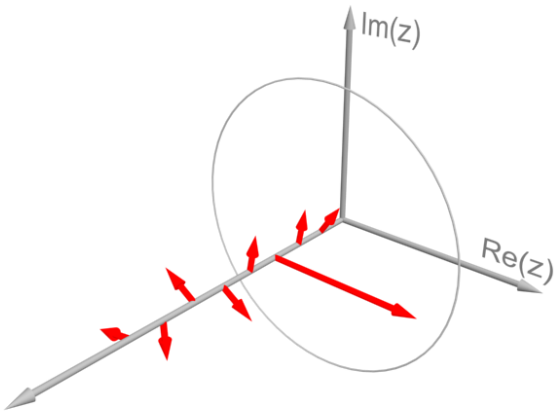
gdzie $|\alpha_0|^2 + |\alpha_1|^2 + ... + |\alpha_7|^2 = 1$.

W wyniku pomiaru stanu takiego trzykubitowego rejestru $|\phi\rangle$ jego stan zostaje ustalony na wektor bazowy $|000\rangle$ z prawdopodobieństwem $|\alpha_0|^2$, $|001\rangle$ z prawdopodobieństwem $|\alpha_1|^2$ itd. Wynika to bezpośrednio z postulatu czwartego mechaniki kwantowej, przedstawionego w sekcji 1.2 na stronie 17.

Własność 2.3. *Wraz z liniowym wzrostem liczby kubitów w rejestrze kwantowym rośnie w tempie wykładniczym wymiar przestrzeni stanów takiego rejestru.*

Jedną z możliwych geometrycznych reprezentacji stanu rejestru kwantowego przedstawiają rysunki 2.2 i 2.3. Wzdłuż trzeciej osi układu mogą być „odkładane" wektory reprezentujące kolejne amplitudy prawdopodobieństw[2].

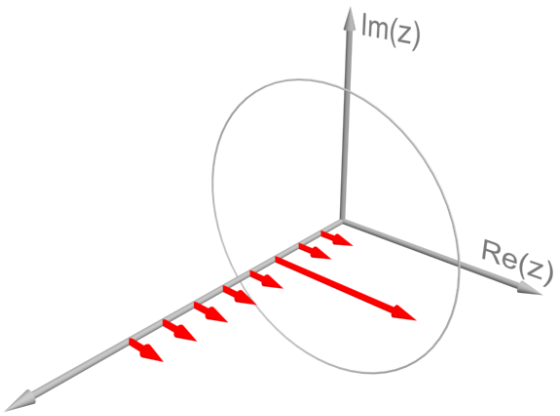
W trakcie wykonywania operacji kwantowych (algorytmy kwantowe) stany kubitów oraz rejestrów kwantowych ulegają zmianie, wektory zmieniają swoje położenie na płaszczyźnie zespolonej, zgodnie z ewolucją unitarną.



Rysunek 2.2: Reprezentacja trzykubitowego rejestru kwantowego

Z własności 2.3 wynika ogromna przewaga, jaką posiadać będą komputery kwantowe nad klasycznymi urządzeniami obliczeniowymi. Liniowe zwiększanie rozmiaru rejestru będzie oznaczało wykładniczy wzrost wymiaru dostępnej przestrzeni. n-kubitowy rejestr kwantowy może zapisywać jednocześnie superpozycję 2^n możliwych wartości. Z drugiej strony z własności 2.3 wynika jedna z trudności projektowania układów do obliczeń kwantowych — choćby dla trzykubitowego rejestru kwantowego jego przestrzenią stanów jest ośmiowymiarowa, zespolona przestrzeń Hilberta.

[2] W niektóre środowiskach symulacji obliczeń kwantowych podejmowane są inne próby wizualizacji rejestrów kwantowych przez interfejs użytkownika — np. użycie różnych kolorów lub reprezentacja na sferze Blocha.



Rysunek 2.3: Stan trzykubitowego rejestru, bez składowych urojonych

2.2 Podstawowe bramki kwantowe

W dalszej części pracy będzie przyjęta następująca definicja bramki kwantowej:

Definicja 2.4. n-kubitowa **bramka kwantowa** (ang. *quantum gate*) to dowolne odwzorowanie 2^n-wymiarowej przestrzeni Hilberta nad ciałem liczb zespolonych w taką samą 2^n-wymiarową przestrzeń, opisywane za pomocą macierzy unitarnej o elementach zespolonych i wymiarze $2^n \times 2^n$.

Podczas ewolucji czasowej układu kwantowego musi być zachowany warunek normalizacyjny wektora stanu, czyli jednostkowa suma prawdopodobieństw w statystycznej interpretacji mechaniki kwantowej. Jednocześnie obliczenia wykonywane przez bramkę kwantową muszą być obliczeniami odwracalnymi — co wynika z termodynamiki obliczeń[3] oraz wymaganej odwracalności względem czasu ewolucji układu kwantowego [Fey96]. Dowodzi się ([Hir04]), że jedynie odwzorowania unitarne spełniają powyższe wymogi. Z definicji 2.4 wynika także, że bramki kwantowe zawsze muszą posiadać tyle samo wejść ile wyjść.

W niniejszym podrozdziale zostaną przedstawione elementarne bramki kwantowe jedno-, dwu- oraz wielokubitowe. Za pomocą bramek kwantowych wykonuje się operacje na kubitach i rejestrach kwantowych. Rezultat użycia bramki kwantowej na rejestrze kwantowym określa własność (2.5):

Własność 2.5. *Stan rejestru kwantowego $|\phi\rangle$, po przetworzeniu przez bramkę kwantową opisaną macierzą unitarną U, jest wyrażony w postaci iloczynu macierzy U oraz wektora kolumnowego odpowiadającego rejestrowi $|\phi\rangle$:*

$$U \cdot |\phi\rangle$$

Aby mnożenie we własności 2.5 było wykonalne, bramka i rejestr kwantowy muszą być odpowiednich rozmiarów. Przykładowo na trzykubitowy rejestr kwantowy, opisywany przez wektor znormalizowany, w ośmiowymiarowej, zespolonej przestrzeni Hilberta, można działać tylko bramkami trzykubitowymi, opisywanymi przez macierze unitarne rozmiaru 8×8.

Podstawowe bramki jednokubitowe to bramka negacji, obrotu amplitudy prawdopodobieństwa, bramka Hadamarda, bramka o macierzy jednostkowej.

[3] odwzorowanie realizowane przez bramkę kwantową na wektorach bazy standardowej musi być bijekcją. Ponieważ entropia całego układu nie może maleć, to odwzorowanie nieróżnowartościowe musiałoby się wiązać np. z wydzieleniem ciepła, a to skutkowałoby interakcją z otoczeniem i dekoherencją układu

Bramka negacji

Bramka negacji zamienia współczynniki stanów bazowych $|0\rangle$ i $|1\rangle$ i jest określona macierzą:

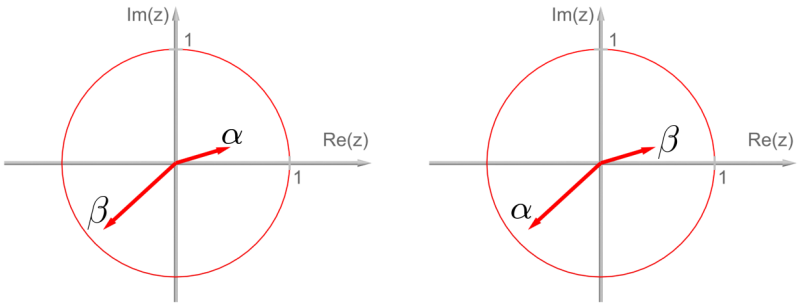
$$Not = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Dla stanów bazowych $|0\rangle$ i $|1\rangle$ bramka Not działa tak jak klasyczna operacja negacji:

$$Not|0\rangle = |1\rangle$$
$$Not|1\rangle = |0\rangle$$

Natomiast dla dowolnej superpozycji tych stanów bazowych bramka powoduje zamianę związanych z nimi amplitud prawdopodobieństw:

$$Not\left(\frac{1}{3}|0\rangle + \frac{2\sqrt{2}}{3}i|1\rangle\right) = \frac{2\sqrt{2}}{3}i|0\rangle + \frac{1}{3}|1\rangle$$



Rysunek 2.4: Bramka Not zamienia amplitudy związane ze stanami bazowymi $|0\rangle$ i $|1\rangle$

Przykład 2.6. *Działanie bramki Not na stan kubitu* $|\phi\rangle = \frac{1}{3}|0\rangle + \frac{2\sqrt{2}}{3}|1\rangle$*:*

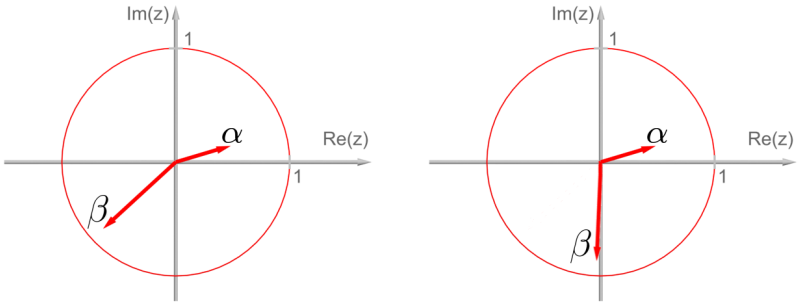
$$Not|\phi\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\left(\frac{1}{3}|0\rangle + \frac{2\sqrt{2}}{3}|1\rangle\right) = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} \frac{1}{3} \\ \frac{2\sqrt{2}}{3} \end{bmatrix} =$$
$$= \begin{bmatrix} \frac{2\sqrt{2}}{3} \\ \frac{1}{3} \end{bmatrix} = \frac{2\sqrt{2}}{3}|0\rangle + \frac{1}{3}|1\rangle \qquad (2.1)$$

Bramka obrotu amplitudy prawdopodobieństwa

Bramka powoduje obrót amplitudy prawdopodobieństwa związanej ze stanem bazowym $|1\rangle$ na płaszczyźnie zespolonej wokół punktu $(0, 0)$ i jest określona macierzą:

$$PhaseShift = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{bmatrix}$$

Bramka zdefiniowana w ten sposób powoduje obrót amplitudy o kąt $\frac{\pi}{4}$.



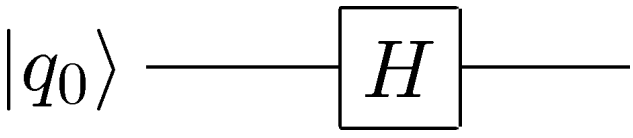
Rysunek 2.5: Bramka $PhaseShift$ powoduje obrót amplitudy związanej ze stanem $|1\rangle$

Bramka Hadamarda

Bramka Hadamarda określona jest za pomocą macierzy:

$$H = \frac{\sqrt{2}}{2} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Symbol, używany do oznaczenia bramki Hadamarda na schematach obwodów kwantowych, przedstawiony jest na rysunku 2.6.



Rysunek 2.6: Symbol bramki Hadamarda

Najczęściej wykorzystywaną własnością bramki Hadamarda jest to, że przekształca ona stan bazowy $|0\rangle$ w równomierną superpozycję stanów bazowych, z tego względu bramka ma podstawowe znaczenie dla obliczeń kwantowych.

$$H|0\rangle = \frac{\sqrt{2}}{2}(|0\rangle + |1\rangle)$$

$$H|1\rangle = \frac{\sqrt{2}}{2}(|0\rangle - |1\rangle)$$

ponieważ:

$$H|0\rangle = \frac{\sqrt{2}}{2}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \end{bmatrix} = \frac{\sqrt{2}}{2}|0\rangle + \frac{\sqrt{2}}{2}|1\rangle$$

$$H|1\rangle = \frac{\sqrt{2}}{2}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} \end{bmatrix} = \frac{\sqrt{2}}{2}|0\rangle - \frac{\sqrt{2}}{2}|1\rangle$$

Wektory

$$H|0\rangle = \frac{\sqrt{2}}{2}(|0\rangle + |1\rangle)$$

$$H|1\rangle = \frac{\sqrt{2}}{2}(|0\rangle - |1\rangle)$$

stanowią ortonormalną bazę w przestrzeni stanów jednego kubitu, nazywaną *bazą Hadamarda*.

Bramka sterowanej negacji (CNot)

Bramka wykonuje operację sterowanej negacji (xor) na jednym z kubitów (ang. *target qubit*), drugi z kubitów jest kubitem sterującym (ang. *control*).

Macierz tej bramki to:

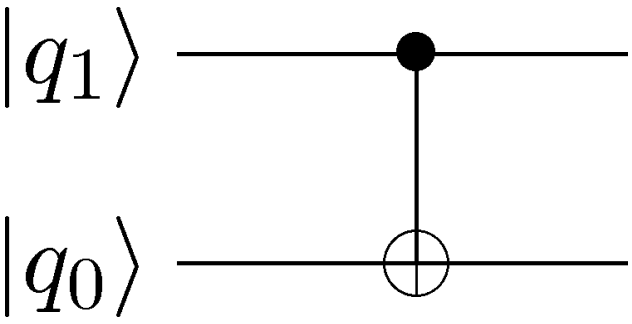
$$CNot = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Istotna uwaga: Bramka CNot w wersji opisanej przez powyższą macierz wykonuje operację sterowanej negacji na mniej znaczącym kubicie, natomiast bardziej znaczący (starszy) kubit jest kubitem sterującym.

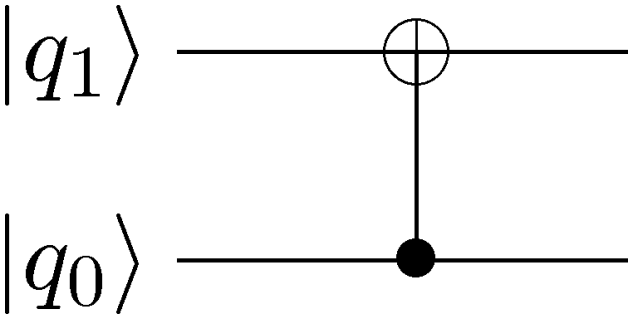
Bramka CNot działa zatem w następujący sposób na wektory bazy standardowej dwukubitowego rejestru:

$$CNot|00\rangle = |00\rangle \qquad CNot|01\rangle = |01\rangle$$

$$CNot|10\rangle = |11\rangle \qquad CNot|11\rangle = |10\rangle$$



(a) Bardziej znaczący kubit $|q_1\rangle$ jest kubitem sterującym, a operacja jest wykonywana na mniej znaczącym kubicie $|q_0\rangle$



(b) Mniej znaczący kubit $|q_0\rangle$ jest kubitem sterującym, a operacja jest wykonywana na bardziej znaczącym kubicie $|q_1\rangle$

Rysunek 2.7: Symbol bramki CNot

Symbol używany do oznaczania tej bramki na diagramach jest przedstawiony na rysunku 2.7.

Przykład 2.7. *Sprawdźmy działanie bramki CNot na stan dwukubitowego rejestru* $|10\rangle$*:*

$$CNot|10\rangle = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = |11\rangle$$

Analogiczna bramka $CNot_2$ (rys. 2.7(b)), która w odwrotny sposób korzysta z kubitów wejściowych — młodszy kubit jest kubitem sterującym — opisana jest przez macierz:

$$CNot_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

Przykład 2.8. *Sprawdźmy działanie bramki $CNot_2$ na stan dwukubitowego rejestru $|11\rangle$:*

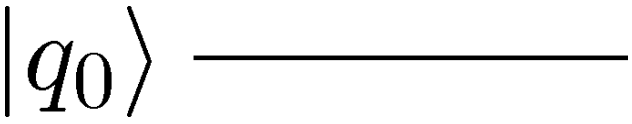
$$CNot_2|11\rangle = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = |01\rangle$$

Bramka o macierzy jednostkowej

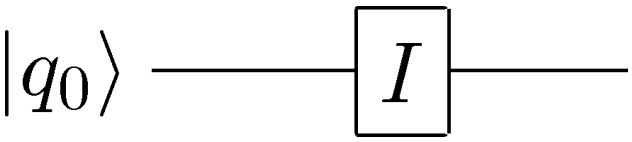
Bramka nie wykonująca żadnego działania na kubicie wejściowym opisana jest macierzą jednostkową:

$$I_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Bramka ta jest oznaczana jako brak jakiegokolwiek symbolu na linii kubitu (rys. 2.8(a)). W dalszej części pracy taka bramka będzie też niekiedy oznaczana w sposób jawny za pomocą symbolu 2.8(b).



(a) Brak zmiany stanu kubitu



(b) Jawne oznaczenie bramki o macierzy jednostkowej

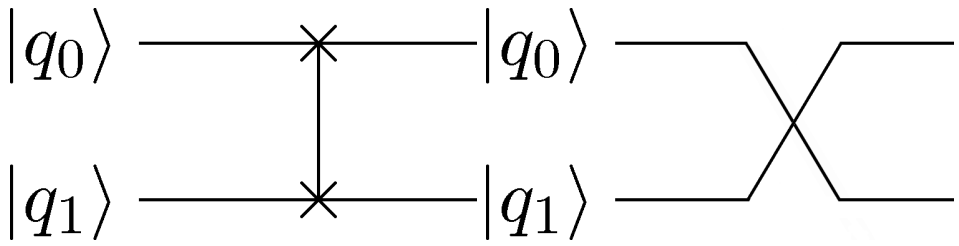
Rysunek 2.8: Bramka nie zmieniająca stanu kubitu

Bramka zamieniająca kolejność kubitów

Zamiana kolejności kubitów w rejestrze kwantowym może być wykonana przy użyciu bramki Swap:

$$Swap = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Na diagramach obwodów kwantowych bramka jest oznaczana jednym z symboli przedstawionych na rysunku 2.9.



(a) Oznaczenie bramki swap (b) Inne oznaczenie bramki swap

Rysunek 2.9: Dwa oznaczenia bramki swap

Bramka Swap jest przydatna, gdy istnieje potrzeba zastosowania innej bramki, która ma działać na niesąsiadujących ze sobą kubitach w rejestrze kwantowym lub na innej kolejności kubitów.

Na parę kubitów bramka Swap działa w następujący sposób:

$$Swap|00\rangle = |00\rangle$$
$$Swap|01\rangle = |10\rangle$$
$$Swap|10\rangle = |01\rangle$$
$$Swap|11\rangle = |11\rangle$$

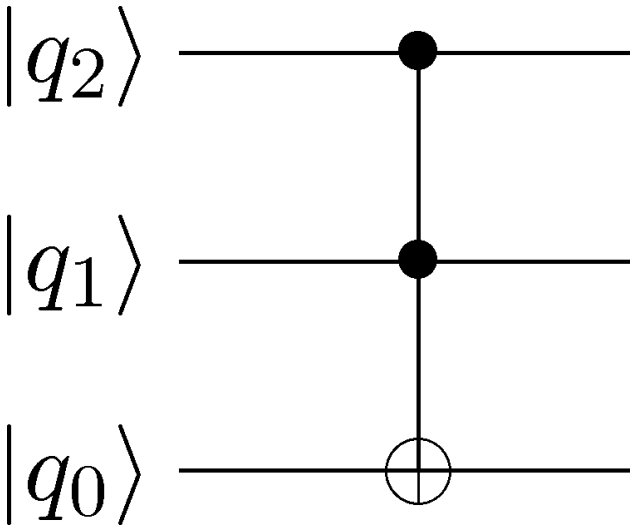
Przykład 2.9. *Zaprezentujmy rezultat działania bramki Swap na stan* $|01\rangle$ *dwukubitowego rejestru:*

$$Swap|01\rangle = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} |01\rangle = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = |10\rangle$$

Bramka Toffoliego

Bramka Toffoliego to bramka trzykubitowa, wykonuje ona operację podwójnej sterowanej negacji (Controlled-Controlled Not) — stan docelowego kubitu jest odwracany tylko w przypadku, gdy na obydwu wejściach kontrolnych podawana jest wartość prawdziwa.

Bramkę Toffoliego oznacza się na schematach obwodów kwantowych za pomocą symbolu przedstawionego na rysunku 2.10.



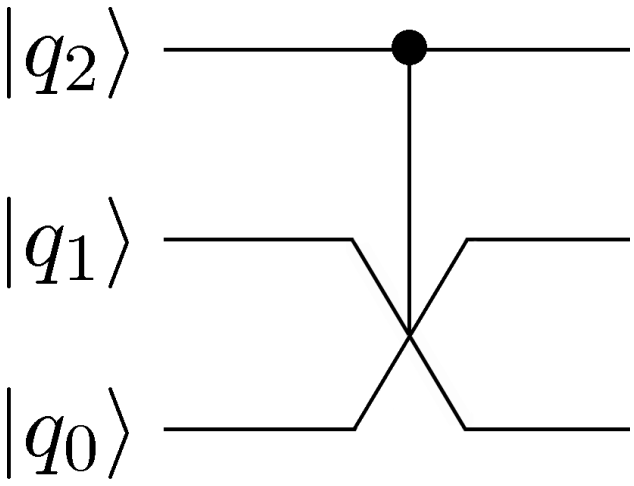
Rysunek 2.10: Symbol bramki Toffoliego

Bramka Fredkina

Bramka Fredkina to również bramka trzykubitowa — wykonuje ona operację sterowanej zamiany kolejności kubitów (ang. *controlled swap*). W niniejszej pracy bramka będzie oznaczana za pomocą symbolu przedstawionego na rysunku 2.11.

Dwa najmniej znaczące kubity na wejściu bramki są zamieniane, jeśli najbardziej znaczący kubit jest w stanie $|1\rangle$:

$$
\begin{array}{lcl}
|000\rangle & \rightarrow & |000\rangle \\
|001\rangle & \rightarrow & |001\rangle \\
|010\rangle & \rightarrow & |010\rangle \\
|011\rangle & \rightarrow & |011\rangle \\
|100\rangle & \rightarrow & |100\rangle \\
|101\rangle & \rightarrow & |110\rangle \\
|110\rangle & \rightarrow & |101\rangle \\
|111\rangle & \rightarrow & |111\rangle
\end{array}
$$



Rysunek 2.11: Symbol bramki Fredkina

Uwaga: należy przy tym pamiętać o przyjętym porządku (kolejności) kubitów. Kubity najbardziej znaczące (kubity od strony nawiasu „|" w notacji Diraca) rysowane są od góry schematu obwodu kwantowego — w tym przypadku $|q_2\rangle$.

Inne bramki kwantowe

Pozostałe bramki kwantowe, spotykane w obliczeniach kwantowych, oraz ich oznaczenia to m.in.:

- bramki Pauliego

  $$X = \sigma_x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \qquad Y = \sigma_y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \qquad Z = \sigma_z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

  Powyższa bramka σ_x odpowiada zatem po prostu bramce Not.

- bramka fazy (ang. *phase gate*) i bramka $\pi/8$ (ang. $\pi/8$-*gate*)

$$S = \sqrt{Z} = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix} \qquad T = \sqrt{S} = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{bmatrix}$$

  Uwaga: w literaturze dotyczącej informatyki kwantowej przyjęło się nazywać powyższą bramkę T bramką $\pi/8$, mimo że wykonuje ona obrót o kąt $\pi/4$. Wynika to wyłącznie ze względów historycznych oraz z możliwości wyrażenia bramki T jako:

$$T = e^{i\pi/8} \begin{bmatrix} e^{-i\pi/8} & 0 \\ 0 & e^{i\pi/8} \end{bmatrix} \tag{2.2}$$

- bramka $\sqrt{NOT}$

$$\sqrt{NOT} = \frac{1}{2} \begin{bmatrix} 1-i & 1+i \\ 1+i & 1-i \end{bmatrix}$$

  Złożenie ze sobą dwóch bramek $\sqrt{NOT}$ daje bramkę NOT. Podobnie każdy inny operator unitarny posiada pierwiastki dowolnych naturalnych stopni.

- bramki obrotów $R_x(\theta)$, $R_y(\theta)$:

$$R_x(\theta) = \begin{bmatrix} \cos(\frac{\theta}{2}) & -i\sin(\frac{\theta}{2}) \\ -i\sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) \end{bmatrix} \qquad R_y(\theta) = \begin{bmatrix} \cos(\frac{\theta}{2}) & -\sin(\frac{\theta}{2}) \\ -\sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) \end{bmatrix}$$

- bramka $J(\theta)$

$$J(\theta) = \begin{bmatrix} e^{-i\theta} & 0 & 0 & 0 \\ 0 & e^{i\theta} & 0 & 0 \\ 0 & 0 & e^{i\theta} & 0 \\ 0 & 0 & 0 & e^{-i\theta} \end{bmatrix}$$

Macierze wszystkich bramek kwantowych są oczywiście macierzami unitarnymi, tzn. spełniają warunek (1.5):

$$U\ U^\dagger = U^\dagger U\ = I_n$$

Własność 2.10 (Twierdzenie o nieklonowaniu)**.** *Nie istnieje operacja kwantowa kopiująca wiernie dowolne kubity, tzn. bramka kwantowa, opisana taką macierzą unitarną U, że dla dowolnego* $|\psi\rangle$*:*

$$U|\psi\rangle|0\rangle = |\psi\rangle|\psi\rangle$$

Bramki uniwersalne

W klasycznym rachunku zdań pewien zbiór funktorów zdaniotwórczych jest uniwersalny (ang. *universal*) lub funkcjonalnie zupełny (ang. *functional complete*), jeśli dowolny inny funktor może być wyrażony za pomocą funktorów należących do tego zbioru. Przykłady takich zbiorów funktorów dla rachunku zdań (logiki boolowskiej) to: $\{\neg, \vee\}$ (negacja i alternatywa), $\{\neg, \rightarrow\}$ (negacja i implikacja), $\{\bar{\wedge}\}$ (zaprzeczenie koniunkcji, tzw. kreska Sheffera).

Podobnie pojęcie uniwersalności dotyczy kwantowych bramek logicznych. Zbiór bramek kwantowych jest *uniwersalny*, jeśli za jego pomocą można przedstawić dowolną inną bramkę kwantową.

Do uniwersalnych zbiorów bramek kwantowych należą m.in.:

- Podstawowy zbiór kwantowych bramek uniwersalnych (ang. *standard universal set of gates*):

  $$\{\mathrm{H}, \mathrm{S}, \mathrm{CNot}, \pi/8\}$$

  (bramka Hadamarda, bramka fazy[4], sterowana negacja, i bramka $\pi/8$)

- Uniwersalny zbiór bramek, wykorzystywanych w fizycznych realizacjach obliczeń kwantowych za pomocą spektroskopii NMR (ang. *liquid state NMR set*) — na podstawie [VYC00]:

  $$\{\sigma_x, \sigma_y, R_z(\theta), J(\theta)\}$$

- bramka Hadamarda, bramka fazy, sterowana negacja i bramka Toffoliego

  $$\{\mathrm{H}, \mathrm{S}, \mathrm{CNot}, \mathrm{Toffoli}\}$$

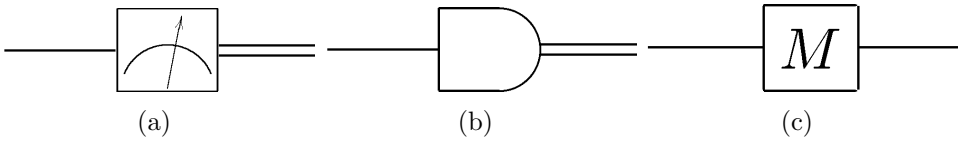
Uniwersalne zbiory bramek kwantowych wykorzystywane są w automatycznym projektowaniu operatorów unitarnych i obwodów kwantowych przez algorytmy i programowanie genetyczne, ponieważ za ich pomocą można wyrazić dowolne obliczenia kwantowe.

[4] bramka fazy może być wyrażona za pomocą bramki $\pi/8$ ($S^2 = T$), jednak pojawia się ona w tym zbiorze, ponieważ jest to wykorzystywane w tzw. obliczeniach *fault-tolerant quantum computing*, które wykraczają poza zakres niniejszej pracy

2.3 Pomiary stanu rejestrów kwantowych

Ważnymi operacjami wykonywanymi na rejestrach kwantowych są operacje pomiaru stanu. Są to jedyne operacje niezgodne z ewolucją unitarną układu kwantowego, więc wymagają one szczególnego uwzględnienia. Pomiar stanu rejestru kwantowego powoduje bezpowrotne utracenie przechowywanego w nim, „delikatnego" stanu koherencji kwantowej — czyli superpozycji możliwych wartości.

Operacje pomiaru stanu są oznaczane są za pomocą symboli przedstawionych na rysunku 2.12. Za pomocą podwójnej ciągłej linii oznacza się niekiedy bity klasyczne. Ponieważ rezultatem pomiaru stanu kubitu może być wyłącznie $|0\rangle$ lub $|1\rangle$, na rysunkach 2.12(a) i 2.12(b) schematy zakończone są podwójną linią.



Rysunek 2.12: Równoważne symbole używane do oznaczenia pomiaru stanu

Rezultatem pomiaru stanu kubitu $|r\rangle = \alpha|0\rangle + \beta|1\rangle$ jest stan $|0\rangle$ z prawdopodobieństwem $|\alpha|^2$ albo $|1\rangle$ z prawdopodobieństwem $|\beta|^2$. Stan kubitu przyjmuje jedną z tych wartości po wykonaniu tej operacji.

Pomiarom mogą także podlegać całe rejestry kwantowe. Po wykonaniu pomiaru stan rejestru jest ustalany na jedną z wartości należących do bazy pomiarowej — najczęściej do bazy standardowej $\{|0\rangle, |1\rangle, \ldots, |2^n - 1\rangle\}$ w przypadku n-kubitowego rejestru. Wybór wektora bazowego jest dokonywany zgodnie z rozkładem prawdopodobieństwa określonym przez długości rzutów wektora stanu rejestru na poszczególne wektory bazy (zgodnie z jednym z postulatów mechaniki kwantowej 1.2).

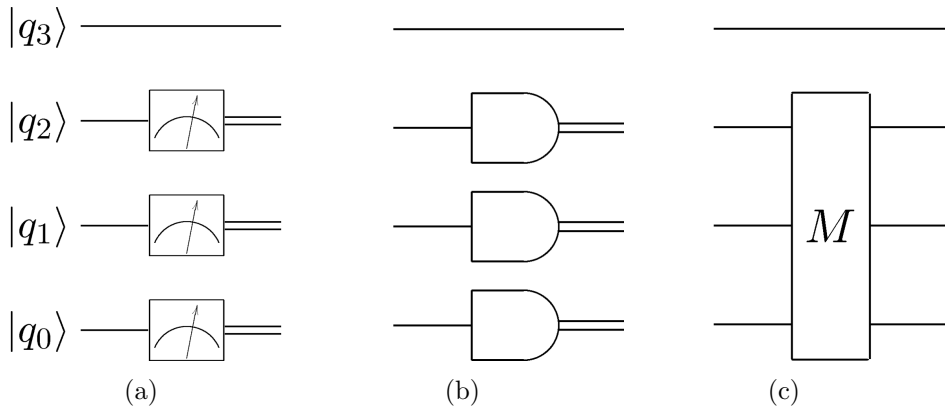
Dysponując pewnym trzykubitowym rejestrem w stanie $|q\rangle = \alpha_0|000\rangle + \alpha_1|001\rangle + \ldots \alpha_7|111\rangle$, oraz wykonując na nim operację pomiaru w bazie standardowej, otrzymuje się stan $|000\rangle$ z prawdopodobieństwem $|\alpha_0|^2$, $|001\rangle$ z prawdopodobieństwem $|\alpha_1|^2$ itd. Stan rejestru zostaje ustalony na jeden z tych wektorów bazowych.

Częściowy pomiar stanu rejestru

Szczególnego uwzględnienia wymaga pomiar stanu rejestru kwantowego, który dotyczy tylko pewnego *podzbioru* kubitów, należących do rejestru. Taka

operacja będzie wykorzystywana np. w obwodzie realizującym protokół teleportacji kwantowej (5.2). Częściowy pomiar może wiązać się z z *natychmiastowym* (nielokalne efekty mechaniki kwantowej) wpływem na pozostałe kubity należące do rejestru — pod warunkiem że stan trójki kubitów był przed operacją pomiaru w stanie splątanym (4.1). Wszystkie te szczególne własności zostaną opisane w dalszych rozdziałach.

Pomiar tego typu jest oznaczany na schematach za pomocą jednego z symboli pokazanych na rysunku 2.13. Przedstawione symbole oznaczają pomiar trzech mniej znaczących kubitów z czterokubitowego rejestru kwantowego.



Rysunek 2.13: Równoważne oznaczenia pomiaru kilku kubitów

Algorytm pomiaru części kubitów, należących do rejestru, zostanie zaprezentowana na przykładzie trzykubitowego rejestru kwantowego. Przykład ten będzie można uogólnić na większą liczbę kubitów.

Rezultatem pomiaru dowolnych m kubitów z n-kubitowego rejestru kwantowego może być jeden z 2^m wektorów bazy standardowej. Wektor ten jest wybierany zgodnie z rozkładem prawdopodobieństw, podobnie jak w poprzednich przypadkach.

Załóżmy, że pomiarowi podlegać będą dwa bardziej znaczące kubity q_2 i q_1 (licząc od zera od kubitów najmniej znaczących) trzykubitowego rejestru kwantowego $|\phi\rangle$ w stanie:

$$|q_2 q_1 q_0\rangle = |\phi\rangle = \alpha_0|\mathbf{00}0\rangle + \alpha_1|\mathbf{00}1\rangle + \alpha_2|\mathbf{01}0\rangle + \alpha_3|\mathbf{01}1\rangle + \alpha_4|\mathbf{10}0\rangle + \alpha_5|\mathbf{10}1\rangle + \alpha_6|\mathbf{11}0\rangle + \alpha_7|\mathbf{11}1\rangle \tag{2.3}$$

$$\alpha_0, \alpha_1, \ldots, \alpha_7 \in \mathbb{C} \quad \text{oraz} \quad |\alpha_0|^2 + |\alpha_1|^2 + \cdots + |\alpha_7|^2 = 1$$

W wyrażeniu (2.3) za pomocą pogrubienia zaznaczono kubity, podlegające pomiarowi. Rezultatem takiego pomiaru może być jeden spośród czte-

rech wektorów bazowych: $|00\rangle$, $|01\rangle$, $|10\rangle$, $|11\rangle$. Prawdopodobieństwo otrzymania stanu $|00\rangle$ jest określone przez te amplitudy związane ze stanami bazowymi rejestru $|\phi\rangle$, w których na pozycjach mierzonych kubitów występuje podłańcuch 00. W tym przypadku są to amplitudy α_0 i α_1. Prawdopodobieństwo otrzymania stanu $|00\rangle$ na dwóch mierzonych kubitach wynosi więc $|\alpha_0|^2 + |\alpha_1|^2$. Prawdopodobieństwo otrzymania rezultatu $|01\rangle$ określone jest przez amplitudy, związane ze stanami bazowymi $|010\rangle$ i $|011\rangle$, więc wynosi ono $|\alpha_2|^2 + |\alpha_3|^2$. Analogicznie, otrzymanie $|10\rangle$ możliwe jest z prawdopodobieństwem $|\alpha_4|^2 + |\alpha_5|^2$, a otrzymanie $|11\rangle$ z prawdopodobieństwem $|\alpha_6|^2 + |\alpha_7|^2$.

Wykonanie takiej operacji pomiaru dwóch kubitów może zaburzać stan *całego* rejestru. Jeśli w wyniku pomiaru otrzymano stan $|00\rangle$, to te amplitudy $|\phi\rangle$, które nie wpływały na otrzymanie $|00\rangle$, zostają ustalone na wartość 0. W przypadku rezultatu $|00\rangle$ zerowane są więc amplitudy $\alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6, \alpha_7$. Po otrzymaniu $|01\rangle$ zerowane są amplitudy $\alpha_0, \alpha_1, \alpha_4, \alpha_5, \alpha_6, \alpha_7$. Jeśli otrzymano $|10\rangle$, zerowane są amplitudy $\alpha_0, \alpha_1, \alpha_2, \alpha_3, \alpha_6, \alpha_7$, natomiast jeśli $|11\rangle$, to $\alpha_0, \alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5$

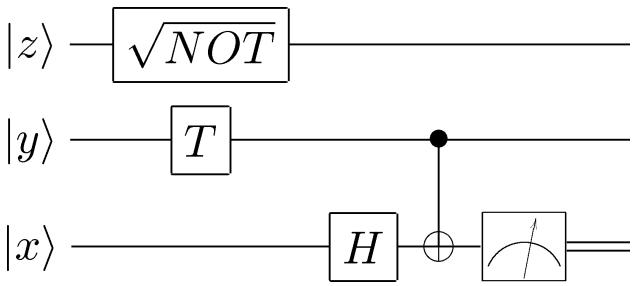
Po przeprowadzeniu takiej aktualizacji amplitud, związanych ze stanami bazowymi rejestru $|\phi\rangle$, stan całego rejestru $|\phi\rangle$ zostaje poddany obowiązkowej normalizacji.

Analogicznie operację pomiaru podzbioru kubitów, należących do rejestru kwantowego, można uogólnić na większą liczbę kubitów. W n-kubitowym rejestrze kwantowym może podlegać pomiarowi dowolny m-kubitowy podzbiór kubitów ($1 \leqslant m \leqslant n$), a kubity podlegające pomiarowi nie muszą ze sobą sąsiadować.

2.4 Obwody kwantowe

Symulowanie działania obwodu kwantowego — a zatem symulacja pracy komputera kwantowego — wykonywane jest przez prosty algorytm, który będzie przedstawiony w tym podrozdziale.

Przykładowe schematy obwodów kwantowych przedstawione są na rysunkach 2.14 i 2.16. Pojawiające się na schematach oznaczenia bramem kwantowych zostały wprowadzone w sekcji 2.2.



Rysunek 2.14: Przykładowy obwód kwantowy

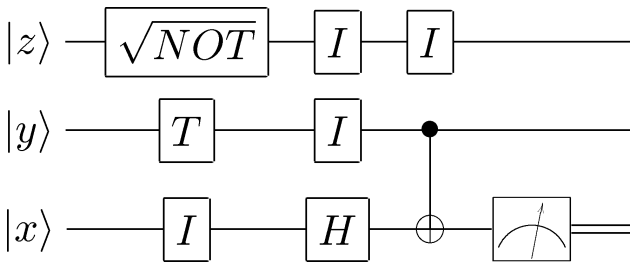
Przedstawiony na rysunku 2.14 obwód kwantowy składa się z pięciu bramek kwantowych, działających na trzykubitowym rejestrze. Obwód składa się z czterech etapów obliczeń (ang. *stages* lub *steps*) i kończy się operacją pomiaru jednego kubitu. Schematy obwodów kwantowych są zawsze analizowane od strony lewej do prawej. W pierwszym etapie bramki $\sqrt{NOT}$ i T działają odpowiednio na kubitach z i y. W drugim etapie bramka Hadamarda działa na kubicie x. W trzecim etapie pojawia się bramka sterowanej negacji, działająca na kubicie x, gdzie kubitem sterującym jest kubit y. W ostatnim etapie obliczeń pomiarowi podlega kubit x. Ta operacja pomiaru jest przykładem pomiaru podzbioru kubitów, należących do rejestru, i jest ona wykonywana zgodnie z algorytmem przedstawionym w sekcji 2.3.

Symulacja pracy obwodu kwantowego wykonywana jest wg następujących reguł:

1. W miejscach w obwodzie kwantowym, w których nie występują żadne bramki, zostają umieszczone bramki o macierzy jednostkowej I. Obwód równoważny obwodowi z rys. 2.14, z zaznaczonymi jawnie bramkami jednostkowymi, przedstawiony jest na rys. 2.15.

2. Połączeniu równoległemu bramek kwantowych odpowiada bramka opisywana przez iloczyn tensorowy poszczególnych bramek. Obowiązuje przy tym kolejność od najbardziej znaczącego[5] kubitu.

[5] kubity najbardziej znaczące są w całej niniejszej pracy rysowane od góry schematów



Rysunek 2.15: Zaznaczone jawnie bramki I o macierzy jednostkowej

Zatem, dla przykładu, pierwszemu etapowi obliczeń w obwodzie z rys. 2.15 odpowiada bramka o macierzy:

$$\sqrt{NOT} \otimes T \otimes I$$

drugiemu etapowi obliczeń w tym obwodzie odpowiada bramka o macierzy:

$$I \otimes I \otimes H$$

itd.

3. Połączeniu szeregowemu bramek odpowiada bramka opisywana przez zwykły iloczyn macierzy poszczególnych bramek (mnożenie macierzy). Trzem pierwszym etapom obliczeń w przykładowym obwodzie odpowiada zatem bramka opisywana wyrażeniem:

$$(\sqrt{NOT} \otimes T \otimes I) \cdot (I \otimes I \otimes H) \cdot (I \otimes CNot)$$

4. Stan otrzymywany na wyjściu etapu obliczeń lub na wyjściu całego obwodu kwantowego jest równy iloczynowi macierzy etapu lub obwodu i stanu wejściowego — jeżeli wewnątrz obwodu nie występowały operacje pomiaru stanu.

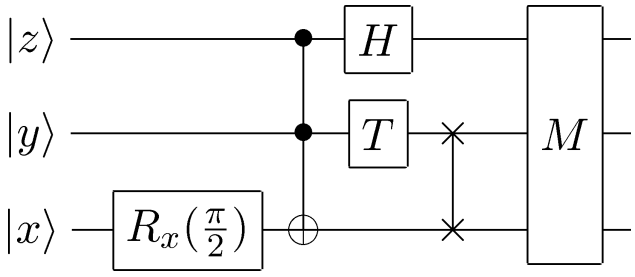
   Rezultatem otrzymywanym po trzech pierwszych etapach obliczeń obwodu 2.14, po podaniu na jego wejściu stanu $|000\rangle$, byłby zatem stan opisywany wyrażeniem:

$$\Big((\sqrt{NOT} \otimes T \otimes I) \cdot (I \otimes I \otimes H) \cdot (I \otimes CNot)\Big)\,|000\rangle$$

Inny przykładowy obwód kwantowy prezentuje rysunek 2.16.

Korzystając ponownie z przedstawionych zasad symulacji działania obwodów kwantowych, pierwsze cztery etapy obliczeń w tym obwodzie są opisywane macierzą unitarną 8×8, daną za pomocą wyrażenia:

$$\left(I \otimes I \otimes R_x\left(\frac{\pi}{2}\right)\right) \cdot \text{Toffoli} \cdot (H \otimes T \otimes I) \cdot (I \otimes Swap)$$



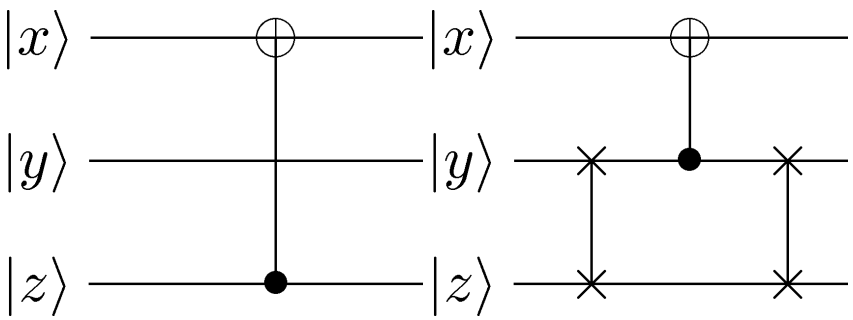
Rysunek 2.16: Przykładowy obwód kwantowy II

Obwód 2.16 zakończony jest operacją pomiaru stanu całego 3-kubitowego rejestru, przetwarzanego przez obwód. Rezultatem otrzymywanym na wyjściu jest zatem jeden z wektorów bazy standardowej $\{|000\rangle, |001\rangle, \ldots, |111\rangle\}$, zgodnie z rozkładem prawdopodobieństw, określonym przez amplitudy, modyfikowane przez obwód.

Przykład: Niesąsiadujące kubity i bramka Swap

Podczas projektowania obwodów kwantowych zachodzi niekiedy potrzeba zastosowania bramki, działającej na kubitach, które nie sąsiadują ze sobą w rejestrze kwantowym.

Najprostszy przykład fragmentu takiego układu, zawierającego bramkę CNot, przedstawia rysunek 2.17(a).



(a) Kubit sterujący i docelowy CNot nie sąsiadują ze sobą

(b) Równoważny obwód kwantowy

Rysunek 2.17: Równoważne obwody kwantowe

Aby znaleźć macierz, opisującą ten fragment układu, należy posłużyć się bramką Swap:

$$Swap = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Przy użyciu tej bramki obwód z rysunku 2.17(a) może być przekształcony do równoważnej postaci, jak na rysunku 2.17(b).

Wówczas — zgodnie z przedstawionymi regułami symulacji działania obwodów kwantowych — macierz takiego fragmentu układu dana jest za pomocą wyrażenia:

$$(I_2 \otimes Swap) \cdot (CNot2 \otimes I_2) \cdot (I_2 \otimes Swap)$$

Podstawiając macierze I_2, $Swap$ oraz $CNot2$ do powyższego wyrażenia oraz wykonując operacje na macierzach, otrzymujemy macierz całego obwodu przedstawionego na rysunku 2.17(a).

$$circuit = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Fragment obwodu kwantowego, opisany powyższą macierzą, działa wówczas w następujący, pożądany sposób:

$$circuit|000\rangle = |000\rangle$$
$$circuit|010\rangle = |010\rangle$$
$$circuit|101\rangle = |001\rangle$$
$$circuit|111\rangle = |011\rangle$$
$$\dots$$

Taka sama operacja może być wykorzystana w przypadku, gdy istnieje potrzeba zamiany kolejności wejść bramki kwantowej — np. zamiana kubitu sterującego i poddawanego operacji alternatywy wykluczającej w bramce CNot.

2.5 Szybki algorytm obliczeń kwantowych

Symulacja działania pracy obwodu kwantowego sprowadza się w najprostszym przypadku do pomnożenia macierzy całego obwodu przez wektor wejściowy, tak jak zostało to przedstawione w poprzednim podrozdziale. Macierz jest obliczana przez znalezienie iloczynów tensorowych wszystkich etapów obliczeń, a następnie pomnożenie macierzy kolejnych etapów. Łatwo jest zauważyć, że w takim podejściu złożoność obliczeniowa – czasowa i pamięciowa – algorytmu rośnie w tempie wykładniczym (wraz ze wzrostem liczby kubitów).

Aby rozwiązać ten problem, w [SGM05] zaproponowano alternatywny, wydajny algorytm symulacji pracy obwodów kwantowych, wykorzystujący własności iloczynu tensorowego (1.10).

Główna zasada działania tego algorytmu polega na tym, że obliczane są wyjścia kolejnych etapów obwodu kwantowego — w odróżnieniu od najprostszego podejścia nie jest obliczana macierz całego układu. Załóżmy, że pewien etap obliczeń w obwodzie kwantowym składa się z L bramek. Pierwszy element wektora wyjściowego danego etapu obliczeń to iloczyn skalarny iloczynu tensorowego pierwszych wierszy L macierzy bramek we fragmencie obwodu i wektora wejściowego etapu obliczeń. Drugi element wektora wyjściowego danego etapu obliczeń to iloczyn skalarny iloczynu tensorowego pierwszych wierszy macierzy $L-1$ pierwszych bramek i drugiego wiersza macierzy ostatniej bramki w danym etapie obliczeń itd. Takie podejście znacząco redukuje złożoność pamięciową. Dokładne pomiary oraz bardziej obszerne przykłady znajdują się w [SGM05].

Dodatkowo takie podejście upraszcza kwestię występowania operacji pomiaru stanu wewnątrz obwodu kwantowego. Wystąpienie operacji pomiaru uniemożliwiało w prostym podejściu zapisanie całego obwodu przy pomocy jednej macierzy unitarnej — ponieważ operacja pomiaru stanu układu kwantowego nie jest operacją zgodną z ewolucją unitarną. Algorytm zaproponowany w [SGM05] oblicza stany otrzymywane na kolejnych fragmentach obwodu. Jeśli w pewnym fragmencie występuje operacja pomiaru, to stan rejestru jest odpowiednio przekształcany i podawany na wejście kolejnego etapu obliczeń w obwodzie.

Rozdział 3

Model obiektowy dla obliczeń kwantowych

> *„In natural science, Nature has given us a world and we're just to discover its laws. In computers, we can stuff laws into it and create a world."*
>
> *Alan Kay*

W rozdziale zostanie przedstawiona propozycja modelu obiektowego, pozwalającego na wykonywanie obliczeń kwantowych — a więc na symulację pracy komputera kwantowego — w dowolnym obiektowym języku programowania.

Opracowany model obiektowy pozwala w łatwy sposób opisywać obwody kwantowe, przedstawione w poprzednim rozdziale, i symulować ich pracę.

Wymagania do obliczeń kwantowych

Do symulacji obliczeń kwantowych potrzebne jest przygotowanie pewnego zbioru obiektów oraz operacji na nich. Zostanie to zaprezentowane w tym podrozdziale.

Podstawowe elementy, wymagane do symulacji obliczeń kwantowych to:

1. wektory liczb zespolonych (macierze kolumnowe) — do reprezentowania kubitów (wektory dwuelementowe) i rejestrów kwantowych (2^n-elementowe)

2. macierze liczb zespolonych (bramki kwantowe) — do reprezentacji operatorów unitarnych, które powodują zmiany stanów rejestrów kwantowych.

3. odpowiednie struktury łączące powyższe elementy — do reprezentowania całych układów bramek kwantowych (obwody kwantowe)

Oprócz wektorów i macierzy liczb zespolonych niezbędne są także odpowiednie operacje wykonywane na tych obiektach.

Potrzebne działania na wektorach (kubity i rejestry kwantowe) to: sprzężenie hermitowskie (1.3), normalizacja, pomiar kwantowy (redukcja wektora stanu). W przypadku pomiaru stanu rejestru muszą być wyróżnione dwie wersje tej operacji: pomiar całego rejestru oraz pomiar części kubitów należących do rejestru, co zostało zdefiniowane w podrozdziale 2.3.

Niezbędne operacje na macierzach (reprezentujących bramki kwantowe) to: obliczanie iloczynu tensorowego (1.10), mnożenie macierzy, sprzężenie hermitowskie (1.3), mnożenie macierzy przez wektor (działanie na stan w rejestrze kwantowym). Dodatkowo przydatne mogą być operacje takie jak: sprawdzenie, czy macierz jest unitarna oraz inne podstawowe operacje na macierzach — obliczanie wyznacznika, śladu, odwrotności.

Reprezentacja całego obwodu kwantowego może być zapisana w postaci struktury drzewa, składającego się z powyższych obiektów. Wybór właściwej struktury jest istotny z punktu widzenia wydajności wykonywanych symulacji. Zgodnie z własnością 2.3, dodawanie kolejnych kubitów do rejestru kwantowego powoduje wykładniczy wzrost używanej pamięci oraz kosztu obliczeniowego. Jednym z rozwiązań tego problemu jest algorytm wydajnej symulacji obliczeń kwantowych, zaproponowany w [SGM05], który został opisany w podrozdziale 2.5.

Wszystkie powyższe niezbędne operacje matematyczne są zapewniane np. przez bibliotekę do obliczeń numerycznych *Numerical Python*.

3.1 Opis modelu

Wykorzystanie techniki programowania zorientowanego obiektowo pozwala programiście na posługiwanie się strukturami wysokiego poziomu, wykonywanie operacji na obiektach reprezentujących konkretne pojęcia z dziedziny problemu. Zaproponowany model wykorzystuje związki dziedziczenia i zawierania, abstrakcyjne typy danych, funkcje wirtualne oraz polimorfizm. Najczęściej używane operacje na obiektach — jak łączenie bramek kwantowych, przetwarzanie stanu w rejestrze kwantowym przez bramkę — zostały dodatkowo udostępnione w postaci przeciążonych operatorów.

Podstawowe cechy zaproponowanego modelu obiektowego to:

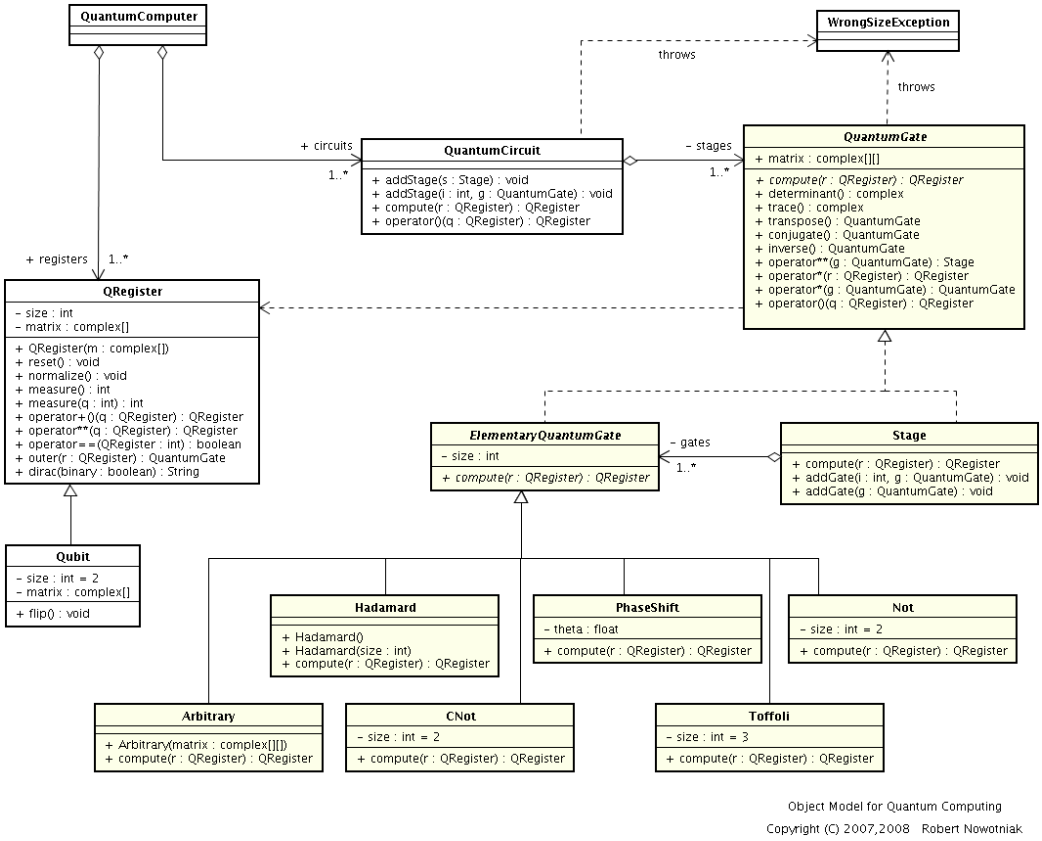
- Obwody kwantowe mogą być zapisywane za pomocą zwięzłej notacji, dzięki wykorzystaniu przeciążonych operatorów ** i *, opakowujących

operacje iloczynu tensorowego i składania odwzorowań, co odpowiada połączeniom równoległym i szeregowym bramek kwantowych.

- Struktura obiektowa jest wzorowana na koncepcji abstrakcyjnych drzew składniowych (ang. *abstract syntax trees*), które znakomicie nadają się do opisu wyrażeń matematycznych lub algorytmów i kodów programów. Przez analogię zaproponowany model pozwala na zapis struktury obwodu kwantowego, która opisuje obliczenia kwantowe.

- Struktura danych dobrze nadaje się do przetwarzania przez algorytmy genetyczne, programowanie genetyczne lub dowolne metody ewolucyjne, co zostanie wykorzystane i zaprezentowane w rozdziale 6.

Rysunek 3.1 przedstawia diagram klas w języku UML modelu obliczeń kwantowych. Opis najważniejszych klas modelu został zawarty w tabeli 3.1 na stronie 46.



Rysunek 3.1: Model obiektowy obliczeń kwantowych

Tabela 3.1: Podstawowe klasy modelu

QRegister	rejestr kwantowy. klasa opakowuje wektor liczb zespolonych, zapewnia spełnienie warunku normalizacyjnego
Qubit	kubit, wektor z dwuwymiarowej przestrzeni, szczególny przypadek QRegister
QuantumCircuit	klasa reprezentująca kompletny obwód kwantowy, posiada uporządkowaną kolekcję kolejnych etapów obliczeń (stages)
QuantumGate	klasa abstrakcyjna, ogólny węzeł abstrakcyjnego drzewa składniowego, definiuje podstawowe operacje na bramce kwantowej (podanie śladu, wyznacznika, ...), dziedziczą z niej konkretne bramki kwantowe.
Stage	pojedynczy etap obliczeń kwantowych, może składać się z wielu połączonych równolegle bramek podstawowych (uporządkowana kolekcja *gates*).
ElementaryQuantumGate	klasa abstrakcyjna, z której dziedziczą wszystkie konkretne bramki podstawowe
Hadamard	bramka Hadamarda
Not	bramka negacji
CNot	bramka sterowanej negacji (domyślnie bardziej znaczący kubit jest kubitem sterującym)
PhaseShift	bramka obrotu amplitudy prawdopodobieństwa
Toffoli	bramka Toffoliego (podwójna sterowana negacja)
Fredkin	bramka Fredkina — operacja sterowanej zamiany kubitów (ang. *controlled swap*)
Arbitrary	bramka o dowolnej macierzy unitarnej, podawanej jako argument konstruktora
WrongSizeException	wyjątek wyrzucany przez bibliotekę przy operacji na obiektach o niezgodnych rozmiarach (bramki, rejestry kwantowe)

Podstawowymi klasami modelu są klasy QRegister, QuantumCircuit, *QuantumGate*. Reprezentują one odpowiednio rejestry, obwody i bramki kwantowe. Klasa *QuantumGate* jest klasą abstrakcyjną, z której dziedziczą konkretne bramki kwantowe. Dziedziczy z niej także klasa Stage, reprezentująca jeden etap obliczeń. Abstrakcyjna metoda *compute()* z klasy *QuantumGate*, definiowana w klasach dziedziczących, odpowiada za obliczanie stanu otrzymywanego na wyjściu danej bramki tzn. wykonywanie mnożenia macierzy danej bramki przez wektor wejściowy.

Działanie przeciążonego operatora `**` na rejestrach lub bramkach kwantowych powoduje wykonanie operacji iloczynu tensorowego[1]. Natomiast operator `*` — na tych samych typach obiektów — powoduje obliczanie iloczynu skalarnego (wewnętrznego). Odpowiada to odpowiednio połączeniom równoległym i szeregowym bramek w obwodzie kwantowym. Dysponując obiektem, reprezentującym bramkę lub cały obwód kwantowy, można ponadto przetworzyć dowolny stan rejestru kwantowego, „wywołując" bramkę lub obwód, tak jak funkcję, oraz podając rejestr kwantowy jako argument — jest to możliwe dzięki przeciążeniu operatora `()` dla klas QuantumCircuit i *QuantumGate*.

Wykaz przeciążonych operatorów binarnych dla odpowiednich typów danych przedstawia tabela 3.2. Obliczenie iloczynu tensorowego macierzy odpowiada połączeniu równoległemu bramek kwantowych lub obliczeniu stanu układu, złożonego z kilku rejestrów kwantowych.

Tabela 3.2: Przeciążone operatory binarne

operator	arg. 1	arg. 2	rezultat	działanie
`**`	QRegister	QRegister	QRegister	iloczyn tensorowy
`*`	QGate	QRegister	QRegister	iloczyn skalarny
`**`	QGate	QGate	QGate	iloczyn tensorowy
`*`	QGate	QGate	QGate	iloczyn skalarny

Zatem przetworzenie pewnego stanu *psi* rejestru kwantowego przez bramkę h może być zapisane jako $h * psi$ lub $h(psi)$. Podobnie złożenie kilku etapów (ang. *stages*) obliczeń w obwodzie kwantowym uzyskiwane jest poprzez `stage1 * stage2`. Obliczenie stanu złożonego rejestru kwantowego może być uzyskane przez `psi1 ** psi2`, natomiast bramka składająca się z równoległego połączenia bramki Hadamarda oraz CNot może być utworzona poprzez `h ** cnot`.

Więcej przykładów użycia wszystkich tych operacji zostanie pokazane na końcu niniejszego rozdziału.

[1] Iloczyn tensorowy jest związany nie tylko z mnożeniem elementów macierzy, ale także wymiarów macierzy — stąd propozycja użycia operatora `**` dla tej operacji.

Uwaga: Implementując zaproponowany model w dynamicznych językach programowania obiektowego, w których nie są ściśle wyróżnione klasy abstrakcyjne, nie ma konieczności tworzenia klasy Arbitrary, ponieważ można byłoby zainstancjonować klasę QuantumGate i zdefiniować jej macierz. Uwaga ta dotyczy języka Python, jednak np. w języku Java musiałaby istnieć odrębna klasa Arbitrary.

Zgodnie z konwencją i stylem preferowanym w wielu językach programowania, nazwy klas rozpoczynają się od wielkiej litery, a nazwy instancji tych klas – od małej.

3.2 Implementacja w języku Python

Przedstawiony model obiektowy dla obliczeń kwantowych został zaimplementowany w bibliotece *qclib* dla języka Python. Pełny kod źródłowy biblioteki znajduje się w załączniku na stronie 110.

W bibliotece zostały zaimplementowane wszystkie klasy, opisane w tabeli 3.1. Dodatkowe zdefiniowane funkcje i stałe pomocnicze, zostały opisane w tabelach 3.3 i 3.4.

Dla biblioteki *qclib* został przygotowany zestaw testów jednostkowych (przy użyciu biblioteki *unittest*), pozwalający na łatwy refaktoring i optymalizację kodu biblioteki.

Tabela 3.3: Pomocnicze funkcje biblioteki qclib

dec2bin(n)	Konwersja liczby n na zapis binarny
Ket(n, m)	tworzy m-kubitowy rejestr kwantowy (QRegister) w stanie bazowym $\|n\rangle$
epr()	Funkcja generująca dwukubitowy rejestr kwantowy (QRegister) w stanie splątanym (para EPR)

Tabela 3.4: Predefiniowane stałe pomocnicze

s2	stała pomocnicza $\frac{\sqrt{2}}{2}$, która bardzo często pojawia się w obliczeniach kwantowych.
ket0	równoważne Ket(0,1)
ket1	równoważne Ket(1,1)
h	obiekt klasy Hadamard, jednokubitowa bramka Hadamarda
cnot	obiekt klasy CNot
cnot2	obiekt CNot(0,1) – bramka CNot z odwróconą kolejnością kubitów
I	równoważne Identity(1), czyli jednokubitowa bramka identycznościowa

3.3 Przykłady użycia API biblioteki qclib

W niniejszym podrozdziale przedstawione są przykłady wykorzystania interfejsu API (ang. *application programming interface*) oraz przeciążonych operatorów napisanej biblioteki *qclib*.

- wypisanie stanu (nieznormalizowanego) $0.3|0\rangle$ w postaci wektorowej:

  ```
  print 0.3 * ket0
  ```

  rezultat:

  ```
  [[ 0.3]
   [ 0. ]]
  ```

- stan $0.7i\ |1\rangle$

  ```
  0.7j * ket1
  ```

- stan nieznormalizowany (superpozycja) $0.5|0\rangle + (0.1 + 0.7i)|1\rangle$:

  ```
  0.5 * ket0 + (0.1 + 0.7j) * ket1
  ```

- stan $0.5|0\rangle + (0.1 + 0.7i)|1\rangle$ po znormalizowaniu:

  ```
  (0.5 * ket0 + (0.1 + 0.7j) * ket1).normalize()
  ```

- wypisanie powyższego, znormalizowanego stanu $0.5|0\rangle + (0.1 + 0.7i)|1\rangle$ w postaci wektorowej:

  ```
  print (0.5 * ket0 + (0.1 + 0.7j) * ket1).normalize()
  ```

  rezultat:

  ```
  [[ 0.57735027+0.j        ]
   [ 0.11547005+0.80829038j]]
  ```

- przypisanie rejestrowi *qreg* stanu $|qreg\rangle = 0.5|0\rangle + (0.1 + 0.7i) * |1\rangle$

  ```
  qreg = 0.5 * ket0 + (0.1 + 0.7j) * ket1
  ```

- wypisanie stanu rejestru *qreg* w notacji Diraca:

  ```
  print qreg.dirac()
  ```

  rezultat:

  ```
  +0.5|0> +(0.1+0.7j)|1>
  ```

- stan bazowy $|01\rangle$ dwukubitowego rejestru:

```
ket0 ** ket1
```

- stan bazowy $|0100\rangle$ czterokubitowego rejestru:

```
ket0 ** ket1 ** ket0 ** ket0
```

- rejestr $qreg$, złączony (iloczyn tensorowy) z dwukubitowym stanem $|00\rangle$ (czyli $|qreg\rangle \otimes |0\rangle \otimes |0\rangle$):

```
qreg ** ket0 ** ket0
```

- wypisanie powyższego stanu w notacji Diraca:

```
print (qreg**ket0**ket0).dirac()
```

  rezultat:

```
+0.5|000> +(0.1+0.7j)|100>
```

- utworzenie obiektu o nazwie h, będącego instancją klasy Hadamard (bramka Hadamarda):

```
h = Hadamard()
```

- połączenie szeregowe bramki I (brak operacji) oraz dwóch bramek Hadamarda:

```
I * h * h
```

- przypisanie kubitowi q rezultatu przetworzenia stanu $|0\rangle$ przez bramkę Hadamarda:

```
q = h(ket0)
```

- pomiar stanu kubitu i wypisanie rezultatu w notacji Diraca:

```
print q.measure().dirac()
```

  rezultat:

```
|1>
```

- połączenie szeregowe bramki CNot (2-kubitowa) oraz połączenia równoległego bramki Hadamarda oraz bramki o macierzy jednostkowej:

```
CNot() * (h ** I)
```

- połączenie równoległe bramki o macierzy jednostkowej I, bramki CNot oraz bramki Hadamarda:

```
I ** CNot() ** h
```

- pomiar stanu całego rejestru *qreg*2:

  ```
  qreg2.measure()
  ```

- pomiar stanu drugiego i trzeciego (licząc od zera) kubitu, należącego do rejestru *qreg*2:

  ```
  result = qreg2.measure(2, 3)
  ```

- wypisanie rezultatu pomiaru dwóch kubitów w notacji Diraca:

  ```
  print result.dirac()
  ```

  rezultat:

  ```
  |10>
  ```

- utworzenie obiektu *circ* reprezentującego obwód kwantowy, składającego się z trzech nietrywialnych bramek kwantowych:

  ```
  circ = (I ** h ** I) * (I ** cnot) * (cnot2 ** I)
  ```

- wypisanie rezultatu działania obwodu kwantowego *circ* na stanie $|000\rangle$ w notacji Diraca:

  ```
  print circ(ket0 ** ket0 ** ket0).dirac()
  ```

  rezultat

  ```
  +0.707107|000> +0.707107|111>
  ```

Rozdział 4

Wykorzystywane efekty informatyki kwantowej

„spooky action at the distance"
Albert Einstein

W niniejszym rozdziale zostaną przedstawione podstawowe efekty mechaniki kwantowej, wykorzystywane w obliczeniach kwantowych. Będą one niezbędne do wykonywania operacji takich jak teleportacja kwantowa lub kodowanie supergęste.

4.1 Stany splątane

Stan dwukubitowego rejestru kwantowego w postaci $\frac{\sqrt{2}}{2}(|00\rangle + |11\rangle)$ nazywany jest parą EPR. Nazwa ta pochodzi od nazwisk Einsteina, Podolskiego i Rosena, którzy jakie pierwsi odkryli, że korelacje przestrzenne takiego stanu prowadzą do paradoksów w mechanice kwantowej [EPR35].

Definicja 4.1. Mówimy, że stan $|\phi\rangle$ dwóch kubitów jest **splątany** (ang. *entangled*), jeśli nie można go przedstawić w postaci iloczynu tensorowego pojedynczych kubitów, tzn:

$$|\phi\rangle \neq (a|0\rangle + b|1\rangle) \otimes (c|0\rangle + d|1\rangle)$$

gdzie

$$a, b, c, d \in \mathbb{C}$$
$$|a|^2 + |b|^2 = 1 \qquad |c|^2 + |d|^2 = 1$$

Przykład 4.2. *Stan* $|\Phi^+\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$ *jest stanem splątanym.*
Nie można go przedstawić jako iloczynu:

$$|\Phi^+\rangle \neq (a|0\rangle + b|1\rangle) \otimes (c|0\rangle + d|1\rangle)$$

Ponieważ wymnażając

$$\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle \neq (a|0\rangle + b|1\rangle) \otimes (c|0\rangle + d|1\rangle)$$

otrzymuje się:

$$\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle \neq ac|00\rangle + ad|01\rangle + bc|10\rangle + bd|11\rangle$$

Z powyższego wynika układ równań:

$$\begin{cases} ac = \frac{1}{\sqrt{2}} \\ bd = \frac{1}{\sqrt{2}} \\ ad = 0 \\ bc = 0 \end{cases} \tag{4.1}$$

Układ jest sprzeczny, co dowodzi, że $|\Phi^+\rangle$ *jest stanem splątanym.*

Stan $|\Phi^+\rangle$ należy do tzw. *bazy Bella*, którą tworzą następujące wektory:

$$\begin{aligned} |\Phi^+\rangle &= \tfrac{1}{\sqrt{2}}|00\rangle + \tfrac{1}{\sqrt{2}}|11\rangle \\ |\Phi^-\rangle &= \tfrac{1}{\sqrt{2}}|00\rangle - \tfrac{1}{\sqrt{2}}|11\rangle \\ |\Psi^+\rangle &= \tfrac{1}{\sqrt{2}}|01\rangle + \tfrac{1}{\sqrt{2}}|10\rangle \\ |\Psi^-\rangle &= \tfrac{1}{\sqrt{2}}|01\rangle - \tfrac{1}{\sqrt{2}}|10\rangle \end{aligned} \tag{4.2}$$

Powyższy zbiór wektorów tworzy bazę ortonormalną w przestrzeni stanów $\mathcal{H}_4$ dwukubitowego rejestru kwantowego. Każdy z tych wektorów reprezentuje stan splątany.

4.2 Paradoks Einsteina-Podolskiego-Rosena

W publikacji [EPR35] Einstein, Podolski oraz Rosen przedstawili „eksperyment myślowy", w którym rozważali pary cząstek znajdujących się w stanach splątanych. Przyjęty przez nich tok rozumowania prowadził do wniosku, że musi być prawdziwa jedna spośród dwóch trudnych wówczas do zaakceptowania hipotez: mechanika kwantowa jest teorią „niepełną" (ang. *incomplete*) lub nie obowiązuje zasada *lokalnego realizmu*.

Zgodnie z jednym z postulatów mechaniki kwantowej (1.2), pełna informacja o stanie układu kwantowego opisywana jest przez funkcję falową (wektor stanu z przestrzeni Hilberta stanów układu) — jednakże istnieją takie stany układów złożonych, które nie mogą być opisane za pomocą stanów należących do nich podukładów. Innymi słowy stan układu jako całości może być lepiej określony niż jego elementów składowych. Jest to jedna z nietypowych własności mechaniki kwantowej, mająca wiele zastosowań.

Taka własność fizycznej rzeczywistości, w której ujawnia się *nielokalna* natura mechaniki kwantowej, budziła początkowo wiele wątpliwości w początkowym etapie rozwoju tej gałęzi fizyki. Wykorzystanie stanów splątanych pozwala *w pewnym sensie* na natychmiastowe oddziaływanie na odległe obiekty w nietypowy sposób (*„spooky action at the distance"* – Albert Einstein).

Rozważane były pary cząstek — znajdujące się w stanie splątanym — które zostały rozsumięte na pewną odległość w przestrzeni. Zgodnie z postulatami mechaniki kwantowej, wykonanie operacji pomiaru stanu pierwszego obiektu powoduje *natychmiastową* zmianę stanu drugiego obiektu — ustalenie jego wartości, redukcję wektora stanu. Ta własność jest niezależna od odległości dzielących cząstki w kwantowym stanie splątanym. Zostało to także pomyślnie wykazane w różnych fizycznych eksperymentach.

$$|\Phi^+\rangle = \frac{\sqrt{2}}{2}\left(|00\rangle + |11\rangle\right)$$

Dysponowanie dwukubitowym rejestrem kwantowym w stanie $|\Phi^+\rangle$, a następnie wykonanie operacji pomiaru jedynie na pierwszym kubicie należącym do takiej splątanej pary, spowoduje otrzymanie rezultatu $|0\rangle$ lub $|1\rangle$ z prawdopodobieństwem $\left(\frac{\sqrt{2}}{2}\right)^2 = 50\%$. Jednakże po otrzymaniu na pierwszym kubicie wartości $|0\rangle$, jest już całkowicie pewne, że stan drugiego kubitu ma również wartość $|0\rangle$. Analogicznie, otrzymanie wartości $|1\rangle$ przy odczycie stanu pierwszego kubitu, *natychmiastowo* determinuje wartość drugiego kubitu, należącego do splątanej pary. Ta własność jest niezależna od fizycznej odległości dzielącej obydwa kubity.

Taką naturę stanów splątanych próbowano początkowo uzasadnić za pomocą teorii opierających się na koncepcji tzw. *zmiennych ukrytych*. Byłyby to pewne wewnętrzne własności układów kwantowych, które mogłyby determinować stan par takich jak $|\Phi^+\rangle$ jako całości od początku ich utworzenia. Istnienie zmiennych ukrytych świadczyłoby o „niezupełności" mechaniki kwantowej, która nie przewiduje istnienia takich zmiennych.

W roku 1964 John S. Bell udowodnił [Bel64], że żadne teorie, opierające się na istnieniu domniemanych zmiennych ukrytych, nie byłyby wystarczające do przewidywania wyników wszystkich pomiarów wykonywanych na stanach splątanych mechaniki kwantowej. Tym samym została potwierdzona własność *nielokalnych* efektów mechaniki kwantowej.

Dodatkowo należy zauważyć, że taki efekt wbrew pozorom nie jest sprzeczny ze szczególną teorią względności (wg. której prędkość światła jest maksymalną prędkością przesyłania informacji we wszechświecie). Pomimo że użycie stanów splątanych pozwala na natychmiastowe zaburzanie stanu odległych obiektów, nie może być to w żaden sposób wykorzystane do natychmiastowego przesyłania informacji. Pomiar stanu jednej cząstki, należącej do splątanej pary, powoduje natychmiastową zmianę stanu odległej cząstki, jednak nie jest możliwe dokładne odczytanie stanu otrzymywanego po drugiej stronie. Zgodnie z przedstawionymi (1.2) postulatami mechaniki kwantowej, rezultatem odczytania stanu takiej cząstki będzie jedna dwóch wartości, z prawdopodobieństwem określonym przez rzut wektora stanu na wektory użytej bazy pomiarowej. Podczas tej operacji faktyczny stan ulegnie zniszczeniu[1], nie będzie więc możliwe poznanie faktycznych współrzędnych tego wektora.

Nietypowe własności stanów splątanych, mimo ich ograniczeń, są wykorzystywane w wielu algorytmach kwantowych.

4.3 Kwantowy paralelizm

Wszystkie odwzorowania unitarne, realizowane przez bramki kwantowe, są odwzorowaniami liniowymi, więc wszystkie operacje wykonywane na rejestrze kwantowym są jednocześnie, w sposób współbieżny, wykonywane na każdym ze stanów bazowych przestrzeni stanów rejestru kwantowego:

$$U \sum_i \alpha_i |i\rangle = \sum_i \alpha_i U |i\rangle \tag{4.3}$$

Ponieważ wymiar przestrzeni stanów rośnie w tempie wykładniczym wraz ze wzrostem liczby kubitów w rejestrze (własność 2.3), niektóre problemy algorytmiczne mogą być rozwiązywane przez komputer kwantowy w czasie wielomianowym, podczas gdy dla algorytmów klasycznych te problemy posiadałyby wykładniczą złożoność obliczeniową.

Kwantowy paralelizm i potencjalna możliwość wykonywania jednoczesnej ogromnej liczby obliczeń przez komputer kwantowy mogą być dobrze

[1] stan zostanie ustalony na jeden z wektorów własnych operatora pomiarowego

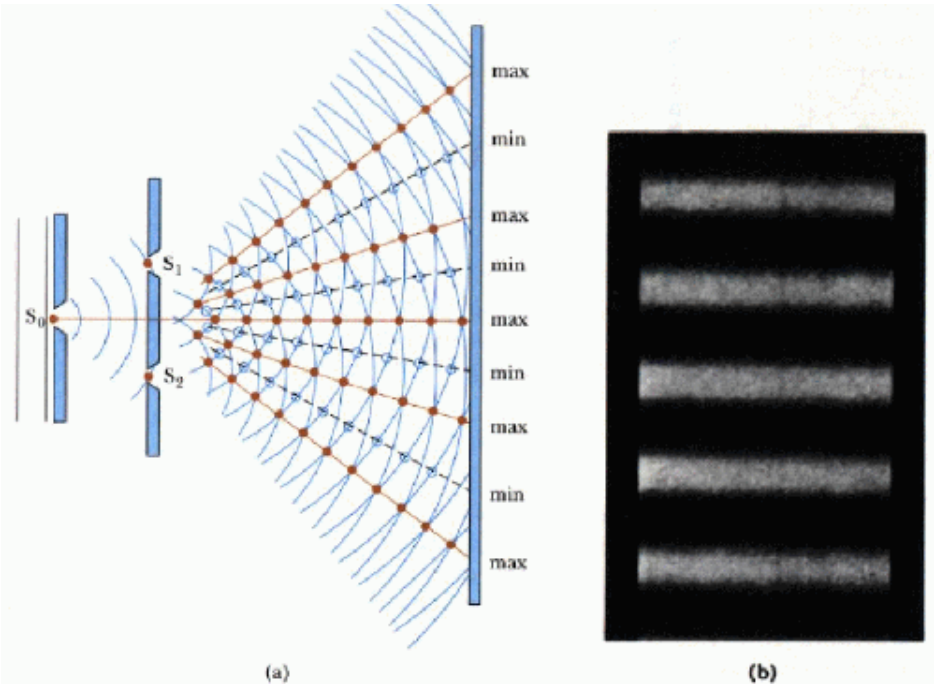
zilustrowane przez przedstawienie analogii do podstawowego eksperymentu fizycznego (rys. 4.1), w którym ujawnia się falowa natura światła.

W takim eksperymencie światło przechodzi przez dwie szczeliny (ang. *two-slit experiment*). Otrzymany w rezultacie obraz tworzy naprzemienne jaśniejsze i ciemniejsze obszary — tzw. prążki interferencyjne. Występowanie ciemniejszych obszarów świadczy o tym, że światło zachowuje się jak fala. Promienie biegnące różnymi drogami interferują ze sobą, w sposób konstruktywny lub destruktywny, wzmacniając lub zmniejszając prawdopodobieństwo jednego z możliwych końcowych rezultatów. Taka własność obserwowana jest także w przypadku pojedynczego fotonu, który poruszając się niejako jednocześnie dwiema drogami, znajduje się ostatecznie w stanie otrzymanym z superpozycji obydwu dróg.

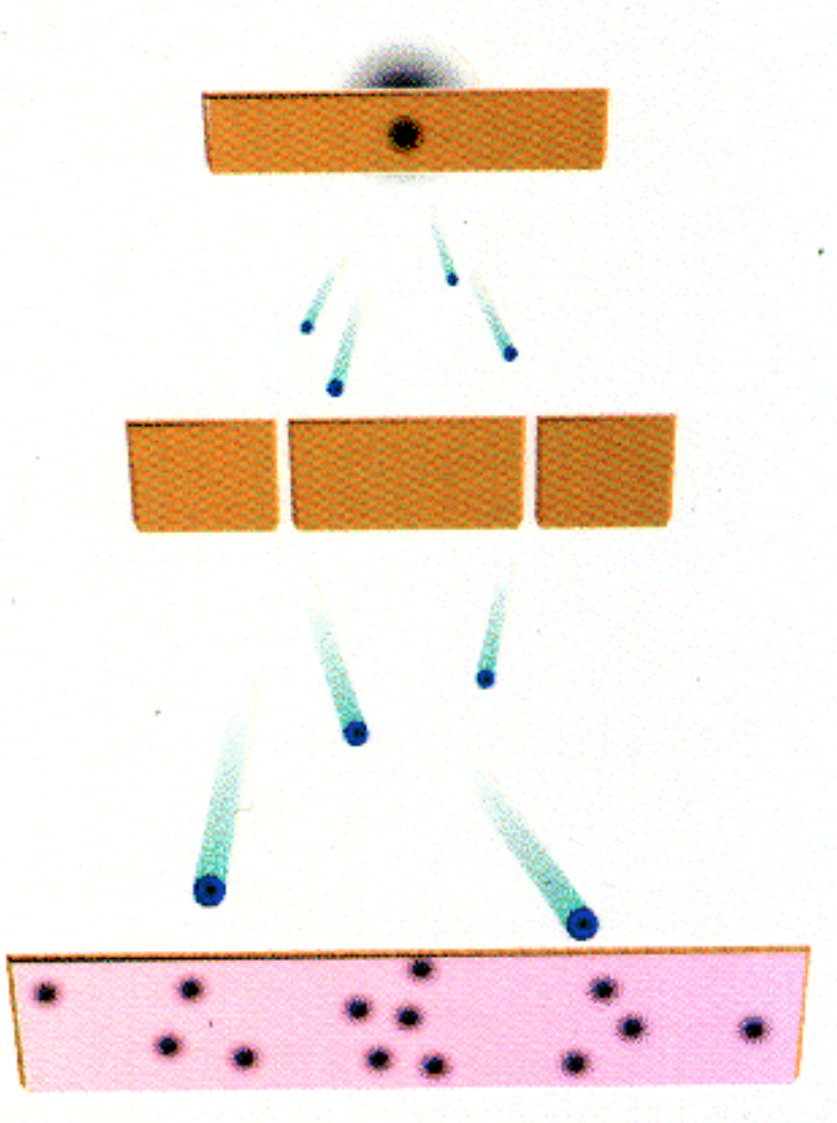
Przedstawiona nietypowa własność, wynikająca z mechaniki kwantowej, jest także wykorzystywana w koncepcji kwantowego modelu obliczeniowego. Komputer kwantowy byłby w stanie wykonywać ogromną liczbę obliczeń, być może nawet nieskończoną, w sposób całkowicie współbieżny, podobnie jak foton światła może poruszać się jednocześnie wieloma drogami.

Wykonywanie obliczeń w taki nietypowy sposób miałoby jednak sens jedynie wówczas, gdy celem nie byłoby otrzymanie wszystkich możliwych wyników, lecz gdy wystarczający byłby wynik, na który w pewnym stopniu miałby wpływ każdy z rezultatów składowych. Możliwość przeprowadzania obliczeń w taki właśnie sposób, w przypadku niektórych problemów znacznie bardziej *efektywny* od algorytmów klasycznych, daje potencjalną ogromną przewagę kwantowemu modelowi obliczeniowemu nad modelami klasycznymi.

Nietypowa własność kwantowego paralelizmu oraz wykorzystanie wzmacniania amplitudy prawdopodobieństwa (interferencja konstruktywna) są wykorzystywane w kwantowym algorytmie Grovera, który szczegółowo zostanie opisany w następnym rozdziale.



Rysunek 4.1: Eksperyment z dwiema szczelinami



Rysunek 4.2: Pojedyncze fotony w tym eksperymencie wydają się poruszać *jednocześnie* dwiema drogami

Rozdział 5

Algorytmy kwantowe

„Is the moon there when nobody looks at it?"
Albert Einstein

W rozdziale zostaną zaprezentowane cztery podstawowe algorytmy kwantowe, które zostały zaimplementowane przy użyciu biblioteki *qclib*, opisanej w rozdziale 3. Najprostszy przedstawiony algorytm będzie służył do generowania stanów splątanych, opisanych w sekcji 4.1. Zostanie także przedstawiony obwód realizujący protokół teleportacji kwantowej, kodowanie supergęste oraz algorytm Grovera. Dużo bardziej szczegółowe opisy tych algorytmów można znaleźć m.in. w [Hir04, GK03].

5.1 Generowanie stanów splątanych

Zgodnie z definicją 4.1 ze strony 52 łatwo można wykazać[1], że stan trzykubitowego rejestru kwantowego w postaci

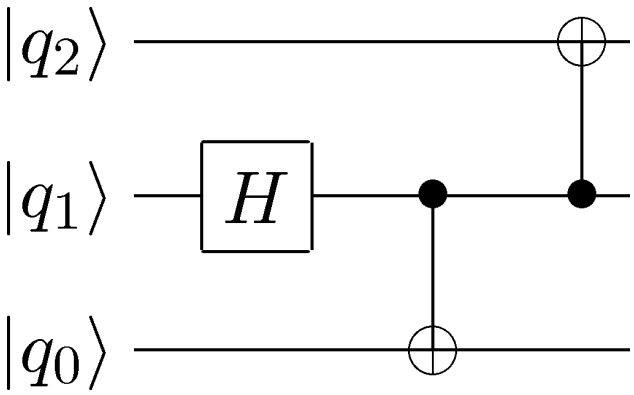
$$|\phi\rangle = \frac{\sqrt{2}}{2}|000\rangle + \frac{\sqrt{2}}{2}|111\rangle \tag{5.1}$$

jest stanem splątanym — nie można przedstawić go w postaci iloczynu tensorowego pojedynczych kubitów.

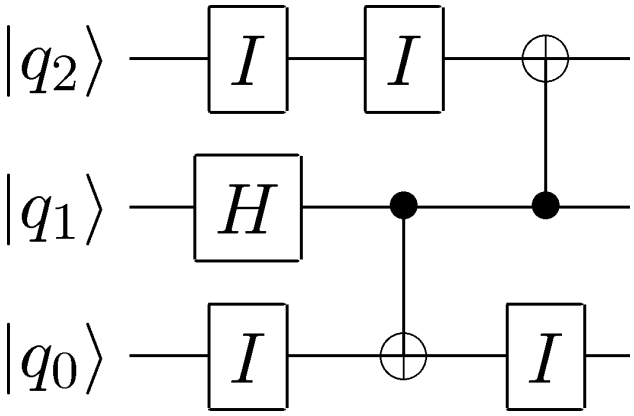
Aby uzyskać taki stan ze stanu bazowego $|000\rangle$, należy wykonać obliczenia przy pomocy obwodu kwantowego przedstawionego na schemacie 5.1.

Rysunek 5.2 przestawia ten sam schemat z zaznaczonymi w sposób jawny bramkami opisywanymi przez macierze jednostkowe.

[1] podobnie jak na stronie 52 można przeprowadzić dowód „nie wprost", tzn. wykazać, że przedstawienie stanu jako iloczynu tensorowego prowadzi do sprzecznego układu równań.



Rysunek 5.1: Obwód kwantowy, generujący stan splątany trzykubitowego rejestru



Rysunek 5.2: Schemat z zaznaczonymi bramkami jednostkowymi

Wykorzystując zaproponowany model obiektowy obliczeń kwantowych oraz opracowaną bibliotekę *qclib*, obwody te są reprezentowane przez wyrażenie:

```
1 circ = (I ** h ** I) * (I ** cnot) * (cnot2 ** I)
```

Natomiast obliczenie stanu otrzymanego na wyjściu obwodu dla stanu wejściowego $|000\rangle$ wykonywane jest poprzez:

```
1 input = ket0 ** ket0 ** ket0
2 output = circ(input)
```

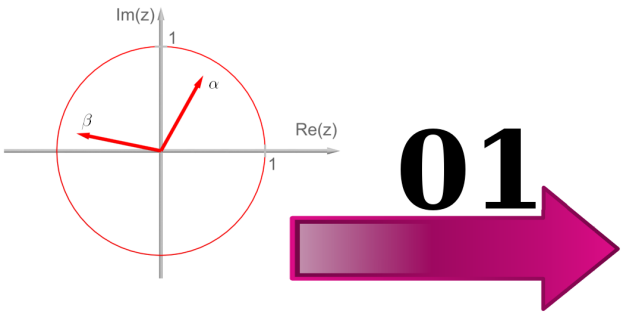
Po wyświetleniu końcowego stanu używanego rejestru kwantowego `output` w notacji Diraca (metoda dirac() klasy QRegister), otrzymamy następujący rezultat: `+0.707107|000> +0.707107`

Otrzymana wartość rejestru kwantowego to oczekiwany stan splątany 5.1.

Skąd bierze się wiedza o tym, że do wygenerowania trzykubitowego stanu splątanego należy posłużyć się właśnie takim obwodem jak przedstawiony na schemacie 5.1? W ogólnym przypadku, zaprojektowanie obwodu kwantowego, który realizuje pożądane odwzorowanie jest trudnym zadaniem, znajdują więc tutaj zastosowanie metody ewolucyjne sztucznej inteligencji, co będzie zaprezentowane w rozdziale 6.

5.2 Protokół teleportacji kwantowej

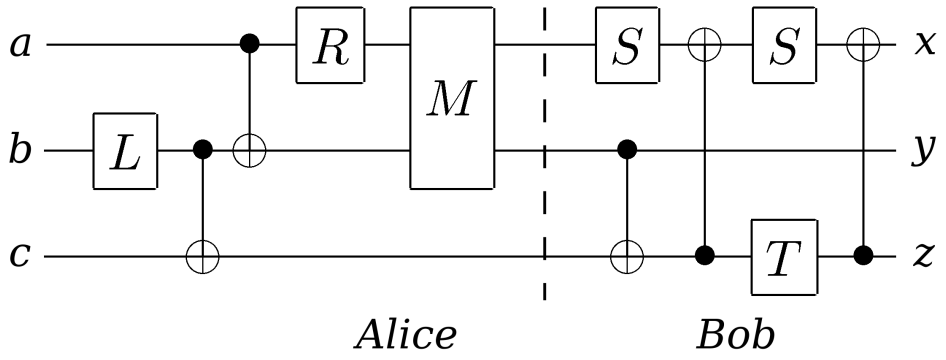
Teleportacja kwantowa pozawala na przesłanie stanu kubitu z użyciem dwóch klasycznych bitów oraz kwantowego kanału komunikacji — czyli splątanej pary EPR. Stan kubitu nie zostaje bezpośrednio odczytany podczas tej operacji, a więc nie ulega zniszczeniu superpozycja stanów, w której się on znajduje.



Rysunek 5.3: Teleportacja kwantowa pozwala przesłać *dokładny* stan kubitu (określony przez dwa wektory na płaszczyźnie zespolonej) jedynie za pomocą dwóch klasycznych bitów informacji

Teoretyczna propozycja protokołu teleportacji kwantowej została przedstawiona w roku 1993 [BBC+93], a w roku 1998 został zaproponowany przez G. Brassarda obwód [Bra96] w modelu kwantowych bramek logicznych, realizujący ten protokół. Przeprowadzano także pomyślne eksperymenty fizyczne, potwierdzające możliwość teleportacji stanów kwantowych pojedynczych cząstek [BPM+97, NKL98] oraz pojedynczych atomów [CSB+04, RHR+04].

Rysunek 5.4 przedstawia układ bramek kwantowych, realizujący protokół teleportacji kwantowej, zaproponowany w pracy [Bra96].



Rysunek 5.4: Obwód kwantowy realizujący protokół teleportacji

Oprócz widocznych bramek sterowanej negacji wykorzystywane są w tym obwodzie także bramki kwantowe, opisane następującymi macierzami:

$$L = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} \qquad R = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}$$

$$S = \begin{pmatrix} i & 0 \\ 0 & 1 \end{pmatrix} \qquad T = \begin{pmatrix} -1 & 0 \\ 0 & -i \end{pmatrix}$$

Aby przeprowadzić symulację teleportacji kwantowej przy pomocy zaimplementowanej biblioteki *qclib*, należy zacząć od zdefiniowania bramek kwantowych, pojawiających się w obwodzie Brassarda:

```
1  L = Arbitrary(s2 * array([
2      [ 1, -1],
3      [ 1,  1],
4      ]))
5  R = Arbitrary(s2 * array([
6      [ 1,  1],
7      [-1,  1],
8      ]))
9  S = Arbitrary([
10     [ 1j,  0],
11     [ 0,   1],
12     ])
13 T = Arbitrary([
14     [-1,   0],
15     [ 0, -1j],
16     ])
```

Potrzebny jest także pewien dowolny stan kubitu, który będzie podlegał teleportacji, przyjmijmy, że będzie to następujący stan *psi*:

```
psi = Qubit([
    [      2.0/7 * (cos(pi/2/9) + 1.0j*sin(pi/2/9)) ],
    [ sqrt(45)/7 * (cos(pi/3*2) + 1.0j*sin(pi/3*2)) ],
    ])
```

Przy użyciu przeciążonych operatorów biblioteki *qclib* można następnie zapisać część nadającą oraz odbierającą obwodu za pomocą następujących wyrażeń (proszę porównać ze schematem na rysunku 5.4!):

```
alice = (I ** L ** I) * (I ** cnot) * (cnot ** I) * (R
   ** I ** I)

bob = (S ** cnot) * (I ** Swap()) * (cnot2 ** I) * \
  (I ** Swap()) * (S ** I ** T) * (I ** Swap()) * (cnot2
      ** I) * (I ** Swap())
```

Na wejściu a obwodu podawany jest stan przesyłanego kubitu *psi*. Pozostałe dwa kubity mają na początku stan $|00\rangle$. Teleportacja kwantowa rozpoczyna się od wygenerowania splątanej pary EPR (dwie pierwsze bramki, działające na kubitach b i c) i rozesłania jej do obydwu stron. Strona wysyłająca wykonuje następnie operacje na posiadanym kubicie a oraz na kubicie b, należącym do splątanej pary.

Zasymulowanie pracy całego obwodu może być wykonane w następujący sposób:

```
input = psi ** ket0 ** ket0
qreg = alice(input)
cbits = qreg.measure(1, 2)
output = bob(qreg)
```

Przerywana pionowa linia na rysunku 5.4 oznacza etap obliczeń, w którym wykonywana jest operacja pomiaru stanu dwóch kubitów (a, b). Wynik tego pomiaru (para bitów) jest przesyłany do strony odbierającej (Bob), gdzie jest on podawany („*reinject*") na a i b (kubit c należy nadal do wygenerowanej na początku splątanej pary). Po wykonaniu obliczeń, opisywanych przez obwód po prawej stronie przerywanej linii, na wyjściu z strona odbierająca otrzymuje początkowo przesyłany stan *psi*.

Należy zauważyć, że za pomocą teleportacji kwantowej przesłanie jedynie dwóch klasycznych bitów informacji powoduje przesłanie pełnego opisu stanu $\alpha|0\rangle + \beta|1\rangle$.

Potencjalna możliwość teleportacji układów większych niż pojedyncze cząstki prowadzi do bardzo interesujących rozważań i wniosków. Zgodnie z *zasadą nieoznaczoności Heisenberga* nigdy nie będzie możliwe dokładne

zapisanie wiernej kopii stanu materii np. w ludzkim mózgu. Jest to kontrargumentem dla koncepcji *determinizmu* we wszechświecie oraz argumentem na istnienie *wolnej woli*. Może być to pewną przesłanką, świadczącą o tym, że zjawiska kwantowe są nieodzowne do zrozumienia działania mózgów istot inteligentnych i posiadanych przez nich własności takich jak świadomość. Za taką koncepcją opowiada się część świata naukowego [Pen89]. Teleportacja kwantowa pozwala na przesyłanie wiernego stanu układu kwantowego bez jego rejestrowania — poprzez przesłanie informacji klasycznej oraz wykorzystanie kanału kwantowego. Pierwotny obiekt, który podlega teleportacji, ulega przy tym bezpowrotnemu zniszczeniu i zostaje wiernie odtworzony w miejscu docelowym.

5.3 Kodowanie supergęste

Kodowanie supergęste pozwala na przesłanie dwóch klasycznych bitów (jedna z czterech wartości: 00, 01, 10, 11) w pojedynczej jednostce informacji kwantowej, czyli za pomocą pojedynczego kubitu. Jest to zatem działanie *komplementarne* w stosunku do protokołu teleportacji kwantowej.

Jaką unikalną własność posiada kodowanie supergęste? Rezultatem operacji pomiaru stanu dowolnego dwupoziomowego układu kwantowego (kubitu) może być jedna spośród dwóch wartości (np. pozioma lub pionowa polaryzacja fotonu). Pomimo tego, przesyłając tylko jeden kubit, można zakodować a następnie odczytać jedną spośród czterech wartości. Podobnie jak w przypadku teleportacji kwantowej, wykorzystywane są nielokalne efekty mechaniki kwantowej oraz kwantowy kanał komunikacji w postaci splątanej pary EPR:

$$|\Phi^+\rangle = \frac{\sqrt{2}}{2}\left(|00\rangle + |11\rangle\right)$$

Przyjmijmy, że b_1 i b_2 to odpowiednio pierwszy i drugi klasyczny bit informacji, która ma zostać przesłana. Kodowanie supergęste wykonywane jest według następującego algorytmu:

1. wygenerowanie splątanej pary EPR w postaci $|\Phi^+\rangle$ i „rozesłanie jej" (po jednym kubicie) do komunikujących się stron
2. Jeśli bit $b_1 = 1$, operacja obrotu amplitudy o π na kubicie znajdującym się po stronie wysyłającej
3. Jeśli bit $b_2 = 1$, operacja negacji (Not) na kubicie znajdującym się po stronie wysyłającej

4. przesłanie kubitu do miejsca docelowego — w tym momencie obydwa kubity znajdują się u strony odbierającej

5. działanie bramką opisaną przez poniższą macierz B na całą parę kubitów

$$B = \begin{bmatrix} \frac{\sqrt{2}}{2} & 0 & 0 & \frac{\sqrt{2}}{2} \\ 0 & \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ \frac{\sqrt{2}}{2} & 0 & 0 & -\frac{\sqrt{2}}{2} \\ 0 & -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \end{bmatrix} \tag{5.2}$$

Za pomocą biblioteki *qclib* powyższy protokół kodowania supergęstego może zostać zapisany w następujący krótki sposób:

```
1  # Generate entangled EPR pair
2  qregiter = epr()
3
4  b1 = 1
5  b2 = 1
6
7  # Perform coding operations on Alice qubit
8  if b1:
9      qregiter = (PhaseShift(pi) ** I)(qregiter)
10
11 if b2:
12     qregiter = (Not() ** I)(qregiter)
13
14 B = Arbitrary([
15     [s2,   0,  0,  s2],
16     [ 0,  s2, s2,   0],
17     [s2,   0,  0, -s2],
18     [ 0, -s2, s2 ,  0],
19     ])
20
21 print B(qregiter).dirac()
```

W powyższym kodzie najpierw tworzony jest dwukubitowy rejestr kwantowy `qregister`, którego stan odpowiada splątanej parze EPR. Zmienne `b1` i `b2` przechowują bity informacji, które mają podlegać kodowaniu. Następnie wykonywane są warunkowe operacje na pierwszym kubicie należącym do splątanej pary (linie 8-12). W linii 14 zdefiniowana zostaje macierz B odpowiadająca wprowadzonej wcześniej macierzy (5.2). `s2` to predefiniowana w bibliotece stała równa $\frac{\sqrt{2}}{2}$. Ostatecznie (linia 21) para kubitów przetwa-

rzana jest przez bramkę opisywaną przez macierz B oraz wypisywany jest końcowy stan rejestru kwantowego w notacji Diraca.

W rezultacie wykonania powyższego przykładu (wartości `b1=1` i `b2=1`) będzie wypisanie przez program łańcucha: `|11>`

Ten stan bazowy odpowiada parze bitów (`b1`, `b2`), które podlegały w tym przykładzie kodowaniu.

5.4 Algorytm Grovera

Algorytm Grovera jest jednym z pierwszych, bardzo praktycznych zastosowań możliwości obliczeń kwantowych, które mogłyby być realizowane przez komputer kwantowy. Takie możliwości pozwoliłyby zoptymalizować przeszukiwanie dużych zbiorów danych i mogłyby zrewolucjonizować dziedzinę projektowania baz danych [Gro96].

Algorytm Grovera służy do rozwiązywania problemu o następujących założeniach. Przeszukiwany jest pewien N-elementowy zbiór, w którym dokładnie jeden obiekt posiada poszukiwaną cechę. Elementy w tym zbiorze nie są w żaden sposób uporządkowane.

Najlepszy klasyczny („trywialny") algorytm, znajdujący poszukiwany element, może mieć przy takich założeniach złożoność obliczeniową $O(N)$, ponieważ w najgorszym przypadku musi „sprawdzić" $N-1$ elementów, a w średnich przypadku $\frac{N}{2}$ elementów. Rozmiarem zadania jest ilość elementów w zbiorze. Taki algorytm musi wykonać w najgorszym przypadku sprawdzenie każdego elementu.

Zaproponowany w roku 1996 algorytm Grovera realizuje to zadanie ze złożonością $O(\sqrt{N})$. Algorytm należy to klasy złożoności BQP (ang. *bounded-error, quantum, polynomial*[2]). Wykorzystywany jest rejestr kwantowy, w którym przetwarzana jest jednoczesna *superpozycja* wszystkich rozwiązań, a algorytm wykonuje iteracyjne „wzmacnianie amplitudy prawdopodobieństwa" (ang. *amplitude amplification*) poszukiwanego rozwiązania.

Wymiar przestrzeni stanów używanego rejestru kwantowego określony jest przez rozmiar N przeszukiwanego zbioru[3]. Każdemu elementowi zbioru odpowiada jeden wektor bazy przestrzeni stanów rejestru: $|0\rangle$, $|1\rangle$, ... $|N-1\rangle$. Początkowy stan rejestru ustawiany jest na „równomierną superpozycję" wszystkich możliwych rozwiązań. Taki stan początkowy będzie dalej oznaczany jako $|\phi_0\rangle$:

[2] ograniczone prawdopodobieństwo błędu w czasie wielomianowym

[3] zakłada się, że jeśli rozmiar zbioru nie jest potęgą liczby 2, to zbiór jest uzupełniany dodatkowymi elementami, nieposiadającymi poszukiwanej cechy

$$|\phi_0\rangle = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} |k\rangle \tag{5.3}$$

W kolejnych iteracjach algorytmu wykonywane są cyklicznie dwie „procedury kwantowe", reprezentowane dalej przez operatory $\mathcal{A}$ i $\mathcal{B}$.

Operator $\mathcal{A}$ pełni rolę „procedury kwantowej", sprawdzającej — jednocześnie — wszystkie elementy i zmieniającej znak amplitudy poszukiwanego elementu. Operator $\mathcal{B}$ powoduje *obrót*[4] amplitud wokół średniej arytmetycznej wszystkich amplitud. Niech $|\omega_0\rangle$ oznacza odtąd wektor bazowy związany z poszukiwanym elementem zbioru.

Operatory $\mathcal{A}$ i $\mathcal{B}$ dane są za pomocą wyrażeń:

$$\mathcal{A} = \mathbb{I} - 2|\omega_0\rangle\langle\omega_0| \tag{5.4}$$

$$\mathcal{B} = 2|\phi_0\rangle\langle\phi_0| - \mathbb{I} \tag{5.5}$$

Początkowy stan rejestru kwantowego ustalany jest na wartość będącą „równomierną superpozycją" wszystkich możliwych rozwiązań, czyli na superpozycję $|\phi_0\rangle$ wszystkich stanów z równymi amplitudami.

Algorytm polega na cyklicznym wykonywaniu — odpowiednią liczbę razy — przekształcenia:

$$|\phi_{n+1}\rangle = \mathcal{B}\mathcal{A}|\phi_n\rangle \tag{5.6}$$

Po wykonaniu odpowiedniej liczby iteracji, uzyskiwane jest maksymalne prawdopodobieństwo otrzymania w wyniku pomiaru stanu rejestru stanu $|\omega_0\rangle$, związanego z poszukiwanym elementem.

Geometryczna interpretacja tego algorytmu, pokazujące działanie operatorów $\mathcal{A}$ I $\mathcal{B}$, została przedstawiona na rysunku 5.5 na stronie 72. Rysunek 5.5(a) pokazuje początkową równomierną superpozycję, kolejne rysunki przedstawiają operacje zmieniające znak amplitudy, związanej z poszukiwanym elementem, oraz obracające wartości wszystkich amplitud wokół wartości średniej. Po zmaksymalizowaniu prawdopodobieństwa otrzymania stanu, związanego z poszukiwanym elementem, wykonywana jest ostatecznie operacja pomiaru (rys. 5.5(j)).

Optymalna liczba iteracji $\bar{k}$ określona jest za pomocą wyrażenia:

$$\bar{k} = \left\lfloor \frac{\pi}{4} \left(\arcsin\left(\sqrt{\frac{1}{N}} \right) \right)^{-1} \right\rfloor \tag{5.7}$$

[4] obrót liczby x wokół r jest tu rozumiany jako funkcja $f(x, r) = r + (r - x)$

Prawdopodobieństwo znalezienia poszukiwanego elementu przy przeszukiwaniu N-elementowego zbioru jest wówczas większe od $1-\frac{1}{N}$. (Dowód tego znajduje się m.in. w [GK03]).

Działanie operatora $\mathcal{A}$ na dowolny stan *bazowy* $|\omega\rangle$ może zostać przedstawione w następujący sposób:

$$\begin{aligned}
\mathcal{A}|\omega\rangle &= (\mathbb{I} - 2|\omega_0\rangle\langle\omega_0|) \cdot |\omega\rangle \\
&= |\omega\rangle - 2|\omega_0\rangle \cdot \begin{cases} 1 & \text{gdy } \omega = \omega_0 \\ 0 & \text{gdy } \omega \neq \omega_0 \end{cases} \\
&= \begin{cases} -|\omega\rangle & \text{gdy } \omega = \omega_0 \\ |\omega\rangle & \text{gdy } \omega \neq \omega_0 \end{cases}
\end{aligned} \tag{5.8}$$

Jak widać z wyrażenia (5.8), operator $\mathcal{A}$ zmienia tylko amplitudę prawdopodobieństwa stanu $|\omega_0\rangle$, związanego z poszukiwanym elementem zbioru. Znak amplitudy tego stanu zamieniany jest na przeciwny.

Działanie operatora $\mathcal{B}$ na *dowolny* stan $|\phi\rangle$ może być natomiast przedstawione w następujący sposób:

$$\begin{aligned}
\mathcal{B}|\phi\rangle &= \mathcal{B}\left(\sum_{\omega=0}^{N-1} \alpha_\omega |\omega\rangle\right) = \sum_{\omega=0}^{N-1} \alpha_\omega \mathcal{B}|\omega\rangle = \sum_{\omega=0}^{N-1} \alpha_\omega \left(\frac{2}{\sqrt{N}}|\phi_0\rangle - |\omega\rangle\right) \\
&= \frac{2}{\sqrt{N}} \sum_{\omega=0}^{N-1} \alpha_\omega |\phi_0\rangle - \sum_{\omega=0}^{N-1} \alpha_\omega |\omega\rangle \\
&= \frac{2}{\sqrt{N}} \sum_{\omega=0}^{N-1} \alpha_\omega \left(\frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} |k\rangle\right) - \sum_{\omega=0}^{N-1} \alpha_\omega |\omega\rangle \\
&= 2 \cdot \underbrace{\frac{1}{N} \sum_{\omega=0}^{N-1} \alpha_\omega}_{\alpha} \sum_{k=0}^{N-1} |k\rangle - \sum_{\omega=0}^{N-1} \alpha_\omega |\omega\rangle \\
&= \sum_{\omega=0}^{N-1} (2\alpha - \alpha_\omega)|\omega\rangle = \sum_{\omega=0}^{N-1} \left[\alpha + (\alpha - \alpha_\omega)\right] |\omega\rangle
\end{aligned} \tag{5.9}$$

gdzie α_ω to amplituda prawdopodobieństwa związana ze stanem bazowym $|\omega\rangle$, a α to średnia arytmetyczna wszystkich amplitud.

Patrząc na początek i koniec wyrażenia (5.9), można stwierdzić, że operator $\mathcal{B}$ przetwarza dowolny stan $|\phi\rangle$, wykonując obrót każdej z amplitud (współrzędnych w bazie standardowej) wokół średniej wartości amplitud.

Za pomocą biblioteki *qclib* powyższy algorytm może zostać zaimplementowany w następujący sposób (omówienie kodu znajduje się pod listingiem):

```
1  N = 2 ** 3
2
3  # index of the searched element
4  n = N - 3
5
6  steps = int(floor(math.pi * 1.0/(math.asin(sqrt(1.0 / N)
       )) / 4))
7
8  print 'Total numer of steps: ' + str(steps)
9  print
10
11 # identity gate
12 I = Identity(3)
13
14 w0 = Ket(n,3)
15
16 # initial state of quantum register
17 phi0 = QRegister([ones(N) / sqrt(N)])
18
19 # computation gates
20 A = I - 2 * w0.outer(w0)
21 B = 2 * phi0.outer(phi0) - I
22
23 phi = phi0
24 step = 0
25
26 while step <= steps:
27     print 'Step number: ' + str(step)
28     print '----------------'
29     print 'Probability of search success: ',
30     print '%0.4f' % abs(float(phi.matrix[n]))**2
31     print
32     if step < steps:
33         phi = B(A(phi))
34     step += 1
35
36 print 'Final quantum register |phi> state: '
37 print phi
38 print
39
40 print 'Probability of search success: ',
41 print '%0.4f' % abs(float(phi.matrix[n]))**2
```

```
42 print
43
44 print 'Correct element: ', n
45 print 'State after measurement: ', phi.measure().dirac(
       binary = False)
```

Na powyższym listingu najpierw definiowany jest rozmiar przeszukiwanego zbioru $N = 2^3 = 8$ oraz indeks n elementu, który posiada poszukiwaną cechę (ten element zbioru będzie wyszukiwany). Następnie obliczana jest prawidłowa liczba iteracji *steps* (ze wzoru (5.7)), dla zadanego rozmiaru zbioru N. Rejestr kwantowy *phi*0, zdefiniowany w linii 17 to początkowa, równomierna superpozycja wszystkich możliwych rozwiązań. W liniach 20 i 21 są natomiast zdefiniowane operatory A i B, zgodnie z wyrażeniami (5.4) i (5.5). W linii 26 znajduje się główna pętla, wykonująca odpowiednią liczbę iteracji algorytmu Grovera. W jej każdym przebiegu wyświetlane jest aktualne prawdopodobieństwo znalezienia szukanego elementu zbioru (linia 30) oraz wykonywane są dwie „operacje kwantowe" $\mathcal{A}$ i $\mathcal{B}$ na aktualnym stanie rejestru kwantowego (linia 33). Po opuszczeniu pętli wyświetlany jest bieżący stan rejestru oraz podawane jest prawdopodobieństwo znalezienia poszukiwanego elementu. W linii 45 wykonywany jest ostatecznie pomiar stanu rejestru *phi* (metoda measure()) oraz wyświetlany jest wynik w notacji Diraca.

Wykonanie powyższego programu (dla parametrów $N = 8$ i $n = 5$) spowoduje otrzymanie następującego rezultatu:

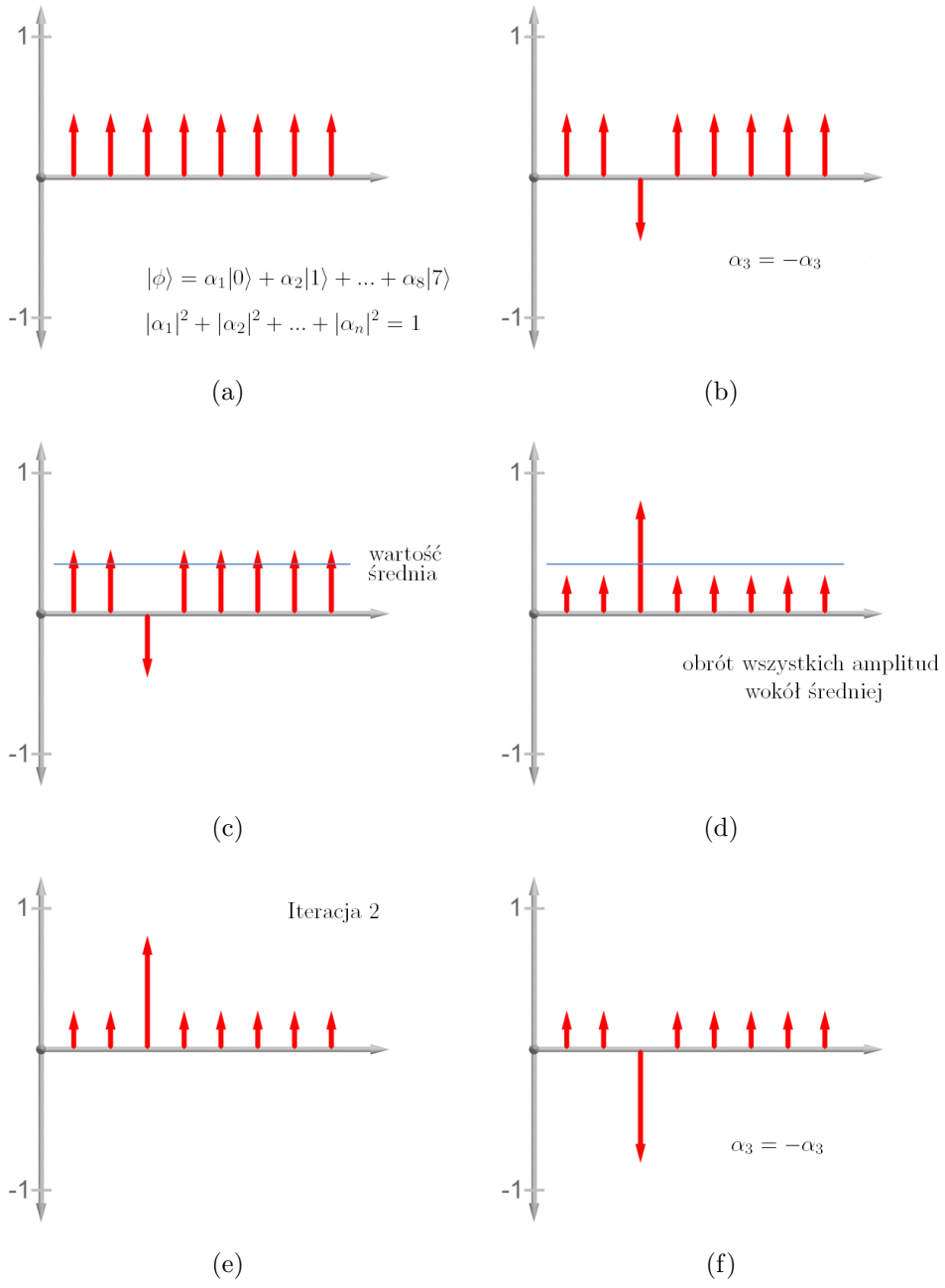
```
Step number: 0
----------------
Probability of search success:  0.1250

Step number: 1
----------------
Probability of search success:  0.7812

Step number: 2
----------------
Probability of search success:  0.9453

Final quantum register |phi> state:
[[-0.08838835]
 [-0.08838835]
 [-0.08838835]
 [-0.08838835]
 [-0.08838835]
```

```
 [ 0.97227182]
 [-0.08838835]
 [-0.08838835]]

Probability of search success:  0.9453

Correct element:  5
State after measurement:  |5>
```

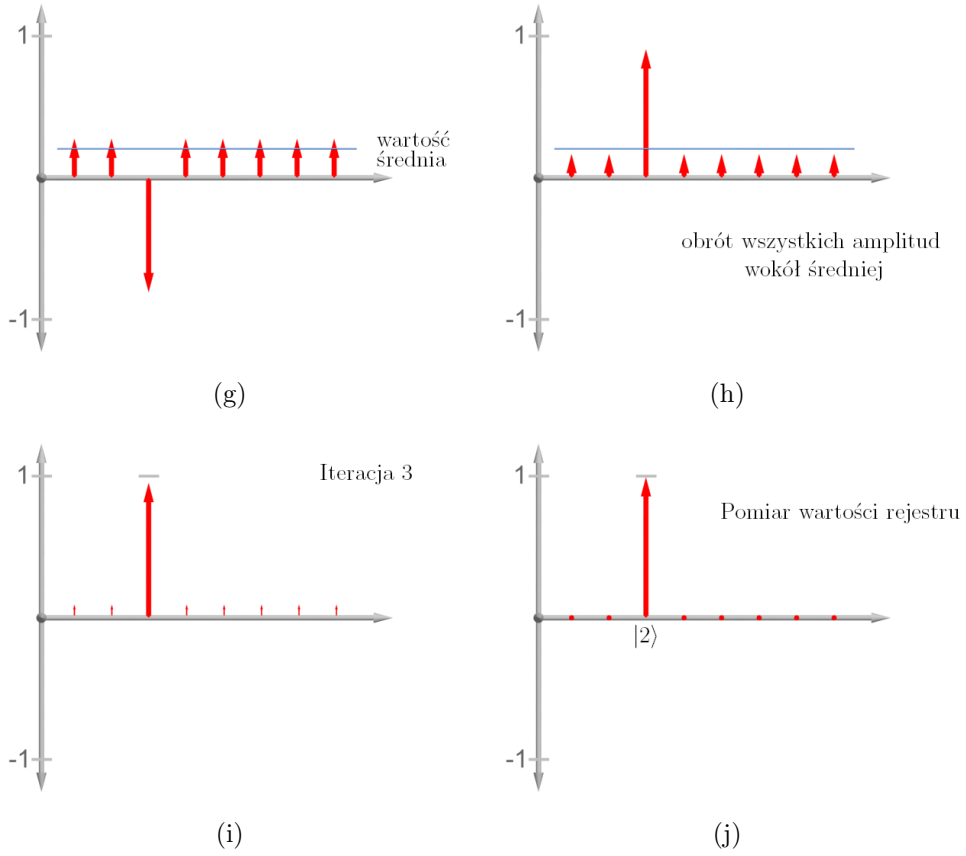
Jak można zauważyć, kwantowy algorytm Grovera działa w zupełnie inny sposób niż klasyczne algorytmy. Klasyczny algorytm przeszukiwania wykorzystywałby kolejne iteracje do eksploracji (sprawdzania) kolejnych „obszarów" przestrzeni rozważań. Algorytm kwantowy przetwarza natomiast całą przestrzeń (superpozycję rozwiązań, znajdującą się w rejestrze kwantowym) *jednocześnie* w pojedynczym kroku, a kolejne iteracje służą do zwiększania prawdopodobieństwa otrzymania właściwego rozwiązania.

Rysunek 5.6 na stronie 74 przedstawia wykres prawdopodobieństwa sukcesu (znalezienia właściwego elementu) w kolejnych iteracjach algorytmu. Z wykresu można odczytać, że przy przeszukiwaniu ośmioelementowego zbioru potrzebne są dwie iteracje algorytmu, aby osiągnąć pierwsze optymalne prawdopodobieństwo odnalezienia szukanego elementu.

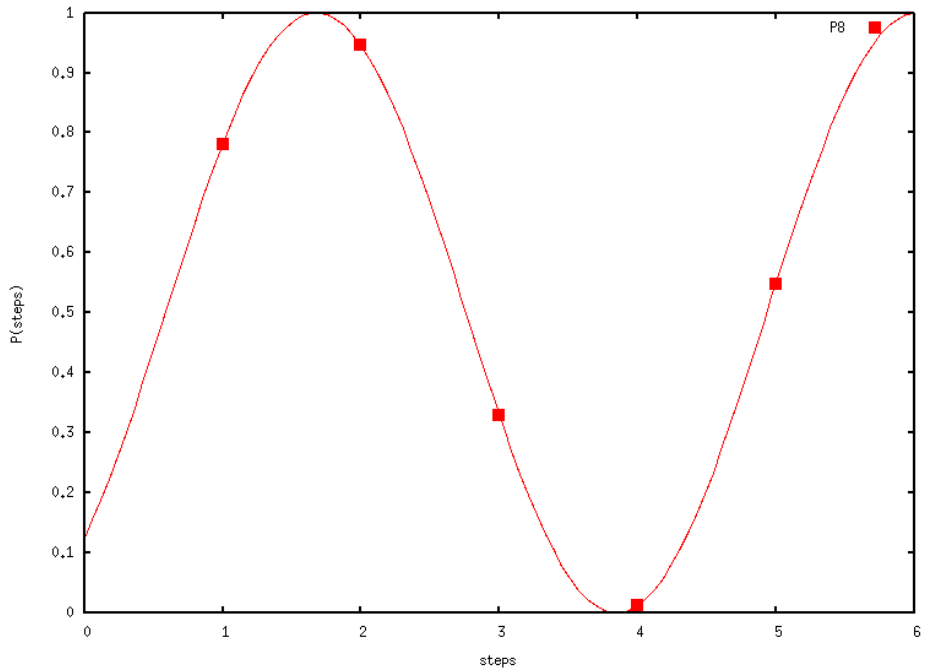
Rysunek 5.7 na stronie 74 przedstawia natomiast wykresy prawdopodobieństw przy przeszukiwaniu ośmio-, szesnasto- i trzydziestodwuelementowego zbioru. Z tych wykresów można odczytać, że przeszukiwanie czterokrotnie większego zbioru wymaga jedynie dwukrotnie większej liczby operacji — algorytm posiada złożoność $O(\sqrt{N})$. Ścisły dowód matematyczny takiej złożoności można znaleźć w [GK03].



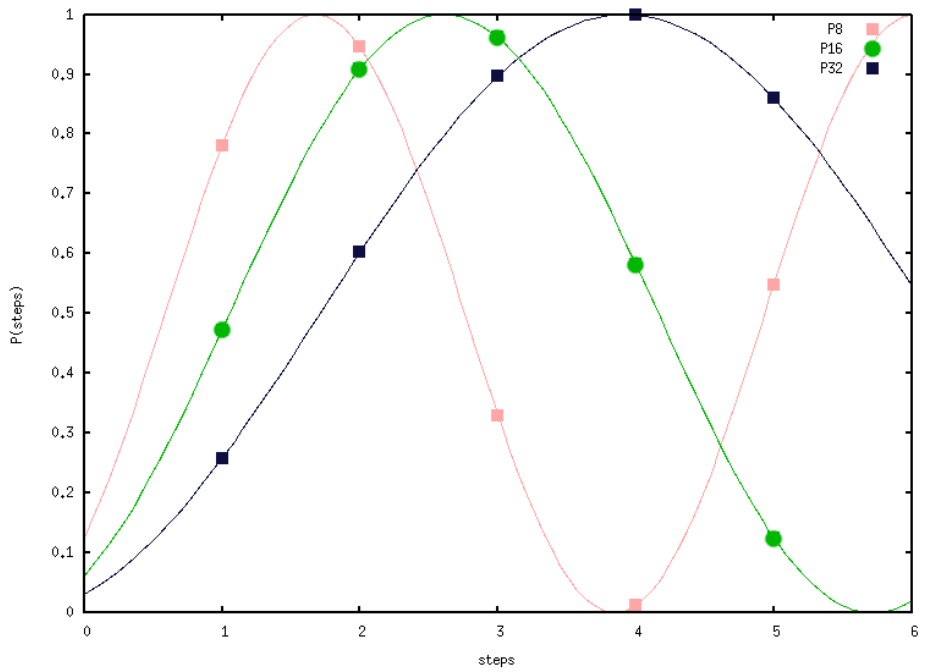
Rysunek 5.5: Kolejne etapy algorytmu Grovera



Rysunek 5.5: Kolejne etapy algorytmu Grovera



Rysunek 5.6: Wykres prawdopodobieństwa sukcesu w kolejnych iteracjach



Rysunek 5.7: Wykresy prawdopodobieństw dla przeszukiwania 8-, 16- i 32-elementowego zbioru

Podsumowanie rozdziału

W rozdziale zostały opisane cztery podstawowe algorytmy kwantowe: generowanie stanów splątanych, teleportacja kwantowa, kodowanie supergęste oraz algorytm Grovera. W każdej z tych sekcji mogły pojawić się wątpliwości, w jaki sposób można byłoby ustalić, że właśnie tak mają wyglądać różne aspekty obliczeń, by ostatecznie realizowane były pożądane operacje. W jaki sposób można byłoby ponownie zaprojektować te algorytmy?

Znalezienie obwodu kwantowego, który realizuje określoną operację jest zadaniem trudnym. W przypadku generowania stanów splątanych i protokołu teleportacji kwantowej zostały przedstawione gotowe schematy takich obwodów. Znalezienie odpowiedniej macierzy unitarnej, takiej jak macierz 5.2 w kodowaniu supergęstym, również jest wymagającym zadaniem. Podobnie jest w przypadku znajdowania operatorów 5.4 i 5.5, używanych w algorytmie Grovera.

We wszystkich powyższych aspektach projektowania obliczeń kwantowych można posłużyć się metodami ewolucyjnymi sztucznej inteligencji, co zostanie przedstawione w kolejnym rozdziale.

Rozdział 6

Metody ewolucyjne i obliczenia kwantowe

> *„The theories of quantum computation suggest that every physical object, even the universe, is in some sense a quantum computer. If this is the case, then according to Turing's work which says that all computers are functionally equivalent, computers should be able to model every physical process. Ultimately this suggests that computers will be capable of simulating conscious rational thought. These theories provoke a minefield of philosophical debate, but maybe the quantum computer will be the key to achieving true artificial intelligence."*
>
> *Simon Bone*

Projektowanie algorytmów kwantowych i programowanie przyszłych komputerów kwantowych są wymagającymi zadaniami z powodu kilku istotnych trudności. Nieintuicyjne efekty mechaniki kwantowej (w szczególności kwantowy paralelizm i interferencja amplitud) powodują, że wiele aspektów procesu obliczeniowego jest trudnych do „uchwycenia" przez ludzki umysł. Jak dotąd odkryto jedynie około kilkunastu istotnych algorytmów kwantowych. Najważniejsze — i jednocześnie najprostsze — z nich zostały zaprezentowane w poprzednich częściach pracy.

Od pewnego czasu w wielu ośrodkach naukowych trwają pracę nad systemami hybrydowymi, wykorzystującymi metody sztucznej inteligencji oraz możliwości informatyki kwantowej [GPT04]. W szczególności podejmowane są próby wykorzystania metod ewolucyjnych sztucznej inteligencji jako narzędzia służącego do projektowania różnych aspektów obliczeń kwantowych [FTG03, YI00, SBBS99b]. Osiągnięto w ten sposób wiele obiecujących rezultatów. Systemy hybrydowe tego typu zostaną przedstawione w niniejszym rozdziale.

W publikacji [GPT04] zostały wyodrębnione następujące aspekty informatyki kwantowej, w których użycie metod ewolucyjnych sztucznej inteligencji może pełnić ważną rolę:

1. Znajdowanie bramek kwantowych (operatorów unitarnych) za pomocą metod ewolucyjnych

2. Projektowanie całych obwodów kwantowych — a zatem całych algorytmów kwantowych – za pomocą metod ewolucyjnych

3. Rozwój metod ewolucyjnych sztucznej inteligencji, które czerpałyby z możliwości oferowanych przez informatykę kwantową. Przykładem mógłby być chromosom, w którym na każdym locusie znajdowałaby się jednoczesna superpozycja wszystkich możliwych wartości genu. Komputer kwantowy mógłby przeszukiwać w sposób współbieżny całą przestrzeń rozwiązań, przetwarzając taki chromosom zgodnie z ewolucją unitarną układu kwantowego.

W dalszej części będą rozważane systemy hybrydowe pierwszego i drugiego rodzaju z powyższej listy.

W niniejszym rozdziale zostaną najpierw przedstawione ogólne informacje o podstawowych metodach ewolucyjnych, a następnie zostaną zaprezentowane możliwości wykorzystania tych metod w kontekście obliczeń kwantowych. Niektóre podejścia zostaną zilustrowane eksperymentami numerycznymi. W celu wykonania eksperymentów zaimplementowano te algorytmy w języku Python, oraz posłużono się biblioteką obliczeń kwantowych *qclib*, zaprezentowaną w poprzedniej części pracy.

Obliczenia ewolucyjne

Sztuczna inteligencja, metody ewolucyjne mogą wspomagać człowieka w rozwiązywaniu złożonych problemów – np. projektowaniu złożonych konstrukcji, podlegających nieintuicyjnym zasadom działania (*„Computer programs that "evolve" in ways that resemble natural selection can solve complex problems even their creators do not fully understand"* — John H. Holland). Zagadnienia z zakresu algorytmiki kwantowej należą do grupy problemów tego typu.

Niezależnie od rodzaju zadania, rozwiązywanego przez algorytm genetyczny, oraz od wybranego wariantu algorytmu, niezbędne jest ustalenie następujących elementów:

1. Sposób **reprezentacji rozwiązań**, które będą automatycznie projektowane i optymalizowane. Wybrana struktura musi być łatwa w przetwarzaniu przez algorytmy genetyczne.

2. Metoda generowania **losowej populacji** rozwiązań

3. Działanie **operatorów genetycznych**, odpowiednie dla przyjętej reprezentacji rozwiązań.

4. Możliwość zamiany reprezentacji genotypu osobniczego na konkretne rozwiązanie z przestrzeni rozważań (**odwzorowanie genotyp → fenotyp**) — czyli przechodzenie z przestrzeni genotypów (ciąg binarny, graf) do przestrzeni fenotypów (obwód kwantowy, macierz).

5. Sposób **oceny jakości** rozwiązania tzn. funkcja przystosowania, która każdemu możliwemu osobnikowi (np. projektowi obwodu kwantowego) przypisuje liczbę rzeczywistą określającą jakość rozwiązania. Taka ocena określa w jakim stopniu osobnik spełnia wymagane przez nas kryterium lub kryteria.

Właściwy wybór reprezentacji rozwiązań jest ważny z punktu widzenia tzw. *linkage problem* — dotyczy to wyboru takiej reprezentacji rozwiązań, by była możliwość „zbudowania" optymalnego rozwiązania z elementów składowych (bloków budujących) rozwiązań ponadprzeciętnie dobrych, dobrze rokujących. Oprócz reprezentacji rozwiązania ważne jest również „właściwe" działania operatorów: rekombinacji i mutacji. Reprezentacja osobników powinna zapewniać, że krosowanie genotypów dobrych osobników powoduje propagowanie się wysoko ocenianych rozwiązań w przyszłych pokoleniach. Podobnie z operatorem mutacji: w środowisku naturalnym występuje ona niezwykle rzadko, a ponadto niewielkie zmiany w kodzie genetycznym powodują niewielkie zmiany w przestrzeni fenotypów (np. zmiany dotyczące tylko jednej cechy). Działanie takiego operatora powinno mieć charakter stabilny. Bez tych dwóch elementów algorytm wykonywałby po prostu losowe przeszukiwanie przestrzeni rozważań, bez wykorzystywania mechanizmów selekcji naturalnej i adaptacji[1].

6.1 Algorytmy genetyczne

Algorytm genetyczny to probabilistyczny algorytm przeszukiwania przestrzeni rozwiązań, wykorzystujący zjawiska obserwowane w ewolucji biologicznej, selekcję naturalną oraz adaptację. Klasyczna wersja algorytmu genetycznego (ang. *simple genetic algorithm*) powstała w latach 1970, a za autora tej metody uznaje się Johna Hollanda [Hol75].

[1] Jedną z możliwości rozwiązania problemu *linkage problem* są tzw. *nieporządne algorytmy genetyczne* (ang. *messy genetic algorithms*)

Działanie algorytmu genetycznego oparte jest na zasadzie doboru naturalnego oraz możliwości przekazywania genów przez osobniki najlepiej dopasowane do warunków środowiska.

Algorytm genetyczny polega na iteracyjnym przetwarzaniu *populacji* rozwiązań, za pomocą operatorów genetycznych: selekcji, mutacji oraz rekombinacji. Powszechnie używane metody selekcji w algorytmach genetycznych to selekcja turniejowa, rankingowa oraz selekcja na zasadzie koła ruletki (wszystkie te metody są metodami probabilistycznymi).

Klasyczny algorytm genetyczny (ang. *simple genetic algorithm*) wykorzystuje kodowanie binarne oraz przebiega wg następujących etapów:

1. Generacja **losowej populacji** początkowej
2. Obliczenie wartości **funkcji przystosowania** chromosomów w populacji
3. **Selekcja** chromosomów przeznaczonych do reprodukcji
4. Tworzenie operatorami **mutacji** i **krzyżowania** nowej populacji
5. Jeśli nie zachodzi **warunek stopu** (w populacji nie istnieje satysfakcjonujące rozwiązanie), następuje powrót do etapu 2)

W klasycznych algorytmach genetycznych przyjmuje się następujące założenia:

- Genotypy osobnicze stanowią chromosomy w postaci binarnych ciągów.
- Stała liczność populacji.
- Stała długość chromosomu.
- Mutacja i krzyżowanie jednopunktowe.

Długość chromosomu jest uzależniona od oczekiwanej dokładności, tzn. ilości miejsc znaczących dla każdej zmiennej.

Jeśli zmienna $p \in \mathbb{N}^+$ (precyzja) określa liczbę oczekiwanych cyfr znaczących, a dziedziną argumentu jest $\langle x_{min}; x_{max} \rangle \subset \mathbb{R}$, to długością ciągu binarnego, który reprezentuje wartość tej zmiennej w chromosomie, powinna być najmniejsza liczba całkowita m, spełniająca nierówność:

$$(x_{max} - x_{min}) \cdot 10^p \leqslant 2^m - 1 \tag{6.1}$$

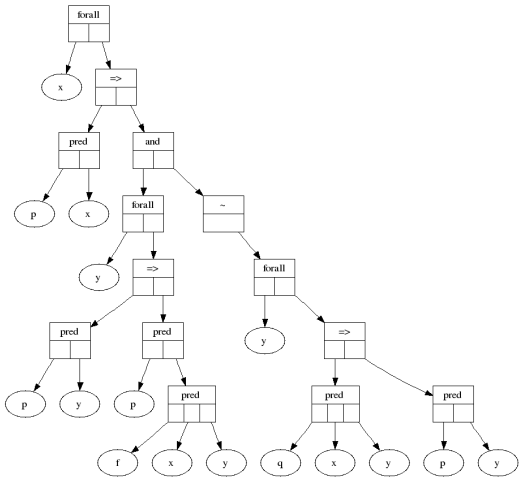
Z powyższej nierówności można wyznaczyć właściwą dla pożądanej dokładności długość ciągu binarnego w chromosomie:

$$m = \lceil \log_2 ((x_{max} - x_{min}) \cdot 10^p + 1) \rceil \tag{6.2}$$

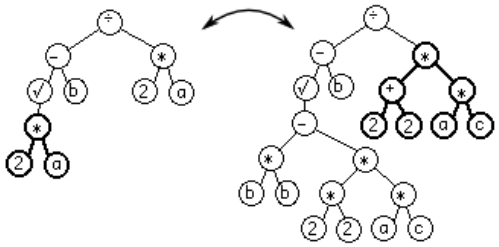
6.2 Programowanie genetyczne

Programowanie genetyczne to rodzaj obliczeń ewolucyjnych, w których przetwarzane populacje osobników reprezentowane są w postaci drzew. Taka reprezentacja pozwala na automatyczne generowanie rozwiązań problemów, na podstawie ich wysokopoziomowych opisów.

Drzewa, jako struktury danych, pozwalają w bardzo dobry sposób reprezentować niektóre rodzaje danych — np. wyrażenia matematyczne, formuły rachunku predykatów lub kody programów. Do takich zastosowań często używana jest struktura abstrakcyjnego drzewa składniowego (ang. *Abstract Syntax Tree*). Przykładowe drzewo, przedstawiające problem z innej dziedziny (zdanie w rachunku predykatów), które jest dobrą strukturą dla algorytmu programowania genetycznego, przedstawione jest na rysunku 6.1.



Rysunek 6.1: Za pomocą drzew można łatwo reprezentować problemy z niektórych dziedzin oraz optymalizować je za pomocą programowania genetycznego



Rysunek 6.2: Krosowaniu w programowaniu genetycznym odpowiada wymiana losowych gałęzi w drzewach

Algorytm programowania genetycznego przebiega w nieco inny sposób niż w przypadku algorytmów genetycznych. W algorytmach genetycznych operatory genetyczne mogły być stosowane już po etapie selekcji, na całej grupie osobników, które przeszły do reprodukcji.

Natomiast w programowaniu genetycznym reprodukcja (przenoszenie osobników do kolejnych generacji) jest niezależnym operatorem. Wybór tego operatora powoduje bezpośrednie przeniesienie osobnika do kolejnej generacji, bez poddawania go innym przekształceniom. Wykonywanie rekombinacji na populacji otrzymanej w wyniku reprodukcji powodowałoby częstą utratę dobrego materiału genetycznego, ponieważ w przypadku programowania genetycznego niewielkie zmiany struktur genotypów (drzew) wpływają często w bardzo dużej mierze na wartość przystosowania.

W przypadku programowania genetycznego nie jest zdefiniowane pojęcie schematu, nie obowiązuje zatem twierdzenie Hollanda ani hipoteza bloków budujących. Pomimo tego, programowanie genetyczne i reprezentacja rozwiązań w postaci drzew dobrze sprawdzają się w automatycznym rozwiązywaniu problemów z różnych dziedzin.

6.3 Znajdowanie operatorów unitarnych

Podstawowym elementem obwodów kwantowych, opisujących algorytmy kwantowe, są kwantowe bramki logiczne, definiowane za pomocą macierzy unitarnych. Każda bramka opisuje operację, wykonywaną na rejestrze kwantowym lub na części należących do niego kubitów. Macierze bramek kwantowych muszą oczywiście spełniać zależność:

$$U\ U^{\dagger} = U^{\dagger}U\ = I$$

Przypuśćmy, że istnieje pewien nieznany operator $F : \mathcal{H}_n \rightarrow \mathcal{H}_n$, oraz jest dany zbiór par (ciąg uczący) $S = \{(|x_1\rangle, |y_1\rangle), (|x_2\rangle, |y_2\rangle), \ldots, (|x_m\rangle, |y_m\rangle)\}$, co do którego wiadomo, że $F|x_i\rangle = |y_i\rangle$ dla każdej pary z S.

Celem jest znalezienie takiego operatora unitarnego G, który dobrze *aproksymuje* F dla zadanego zbioru par punktów. Należy zatem znaleźć taką macierz unitarną U, że $U\ \cdot |x_i\rangle \approx |y_i\rangle$ dla każdej pary z S.

Dla uproszczenia notacji, przyjmijmy dalej, że X i Y będą oznaczały macierze rozmiaru $n \times m$. Kolumny macierzy X i Y tworzą odpowiednio wektory kolejnych par ze zbioru S, tzn. $X = [x_1|x_2|\ldots|x_m]$, $Y = [y_1|y_2|\ldots|y_m]$. Zadanie polega zatem na znalezieniu takiej macierzy unitarnej U, która najlepiej spełnia warunek:

$$U\ X \approx Y \tag{6.3}$$

Deterministyczny algorytm rozwiązywania takiego problemu, znajdowania macierzy unitarnych został zaproponowany w [Ven00]. Zaprezentowany tam, iteracyjny algorytm, wykorzystujący metodę najszybszego spadku, posiada jednak kilka wad (nie gwarantuje zbieżności ani otrzymania macierzy unitarnej). Użycie algorytmów genetycznych jest jedną z metod rozwiązania tego problemu [FTG03], co zostanie zaprezentowane w dalszej części.

Projektowanie bramek kwantowych można zatem rozważać jako problem minimalizacji pewnej funkcji błędu $\epsilon(U)$. Przyjmijmy, że:

$$\epsilon(U) = \frac{1}{2}||UX - Y||_F^2 \tag{6.4}$$

gdzie $||\cdot||_F^2$ oznacza podniesioną do kwadratu normę Frobeniusa macierzy:

$$||A||_F^2 = \sum_i \sum_j |a_{ij}|^2 \tag{6.5}$$

Zaletą użycia normy Frobeniusa jest to, że może być ona łatwo obliczana przy pomocy śladu (1.15) macierzy:

$$||A||_F^2 = tr(A\ A^{\dagger}) \tag{6.6}$$

Dekompozycja macierzy unitarnych

Z punktu widzenia znajdowania macierzy unitarnych za pomocą algorytmów genetycznych ważna jest możliwość dekompozycji macierzy bramek kwantowych. Dowolna jednokubitowa bramka kwantowa $U_{2\times2} \in \mathbb{C}^{2\times2}$ może być wyrażona przy użyciu czterech parametrów rzeczywistych $\phi, \theta, \psi, \alpha \in \langle -\pi; \pi \rangle \subset \mathbb{R}$ jako iloczyn macierzy:

$$U_{2\times2} = \begin{bmatrix} e^{-i\phi} & 0 \\ 0 & e^{i\phi} \end{bmatrix} \begin{bmatrix} cos(\theta) & sin(-\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} \begin{bmatrix} e^{-i\psi} & 0 \\ 0 & e^{i\psi} \end{bmatrix} \begin{bmatrix} e^{i\alpha} & 0 \\ 0 & e^{i\alpha} \end{bmatrix} \quad (6.7)$$

Macierze unitarne większych rozmiarów można natomiast dekomponować według następujących reguł (dowód i przykłady dekompozycji znajdują się w [NC00]).

1. Dowolna macierz unitarna $U_{n\times n}$ może być zdekomponowana na iloczyn $\binom{n}{2} = \frac{(n-1)n}{2}$ macierzy unitarnych, takich że w każdej co najwyżej dwa wiersze są nietrywialne tzn. inne niż w macierzy I (ang. *two-level unitary matrix*).

2. Macierze takie jak w poprzednim punkcie (*two-level unitary matrix*) mogą być zdekomponowane na iloczyn macierzy bramek CNOT i bramek działających na pojedynczym kubicie ($U_{2\times2}$).

3. Dowolna jednokubitowa bramka $U_{2\times2}$ może być przybliżona z dowolną dokładnością za pomocą uniwersalnego zestawu bramek np.: $\{\mathrm{H}, \mathrm{S}, \mathrm{CNot}, \pi/8\}$. Inne uniwersalne zbiory bramek kwantowych zostały przedstawione w podrozdziale 2.2 na stronie 34.

Połączenie trzech powyższe punktów, prowadzi do wniosku, że dowolną bramkę kwantową można przedstawić dokładnie za pomocą pewnej liczby bramek CNot oraz bramek działających na pojedynczym kubicie. Każdą bramkę kwantową można także przybliżyć z dowolną dokładnością za pomocą uniwersalnego zbioru bramek kwantowych.

Generowanie losowych macierzy unitarnych

Początkowym etapem algorytmu genetycznego, którego zadaniem będzie znajdowanie optymalnej bramki kwantowej, jest wygenerowanie losowej populacji macierzy unitarnych określonych rozmiarów.

Procedura, generująca losowe macierze unitarne, powinna zwracać z takim samym prawdopodobieństwem macierze z różnych obszarów rozważanej

przestrzeni — ponieważ takie losowe macierze stanowić będą początkową pulę materiału genetycznego i zależy od nich skuteczność działania algorytmów genetycznych.

W niniejszej pracy do losowego generowania takich macierzy zostanie wykorzystana jedna z metod zaproponowana w publikacji [Mez06].

Zaczerpnięta z [Mez06] procedura generowania losowej macierzy unitarnej wykorzystuje rozkład QR macierzy, wg następującego algorytmu (funkcja wykorzystuje bibliotekę *Numerical Python*):

```
1  def random_unitary_matrix(n, real = False):
2      if not real:
3          z = (random.randn(n,n) + 1j * random.randn(n,n))
               / sqrt(2.0)
4      else:
5          z = (random.randn(n,n)) / sqrt(2.0)
6      q,r = linalg.qr(z)
7      d = diagonal(r)
8      ph = d / absolute(d)
9      q = multiply(q, ph, q)
10     return matrix(q)
```

W stosunku do procedury przedstawionej w [Mez06], w powyższej funkcji została dodana możliwość generowania macierzy o elementach wyłącznie rzeczywistych. Dzieje się tak w przypadku wywołania `random_unitary_matrix` z argumentem *real*, przyjmującym wartość logiczną *true*. Argument ten domyślnie posiada wartość $false$.

Przykładowe, zupełnie losowe macierze rozmiaru 3×3, o właściwym rozkładzie, otrzymane w ten sposób to:

```
[[-0.08907709+0.13082j  0.77156723+0.17318j  0.07529971-0.58649j]
 [-0.32757076-0.21801j -0.14797343-0.57527j  0.64299612-0.28086j]
 [-0.62753491+0.65292j  0.12950222-0.07129j  0.04044600+0.39546j]]

[[-0.04164491  0.80883509  0.58655903]
 [-0.54004129 -0.51214486  0.66787951]
 [ 0.84060758 -0.28895231  0.45813258]]
```

Eksperyment 1: SGA, macierz 2×2

W pierwszym, najprostszym eksperymencie zostanie zaprezentowane znalezienie macierzy unitarnej 2×2 za pomocą prostego algorytmu genetycznego. Zostanie w tym celu wykorzystany rozkład macierzy $U_{2\times2}$, pozwalający wyrazić ją za pomocą czterech parametrów rzeczywistych. Taka metoda ma ograniczony zakres zastosowań, jest specyficzna dla macierzy unitarnych rozmiaru 2×2, jednak wyniki tego eksperymentu posłużą do lepszej oceny późniejszych, bardziej wyszukanych podejść. Macierze unitarne rozmiaru 2×2 pełnią także szczególną rolę, ponieważ można posłużyć się nimi do dekompozycji macierzy unitarnych dowolnych rozmiarów.

W tym podejściu zastosowano reprezentację binarną oraz następujący sposób zakodowania rozwiązania w genotypie:

$$\underbrace{1100\cdots00111111}_{\phi}\underbrace{1111\cdots11011010}_{\theta}\underbrace{1010\cdots01011011}_{\psi}\underbrace{1011\cdots00101101}_{\alpha}$$

gdzie ϕ, θ, ψ i α to liczby rzeczywiste z przedziału $\langle -\pi; \pi \rangle$. Przy pomocy tych wartości wykonywane jest odwzorowanie na przestrzeń fenotypów (macierze unitarne 2×2):

$$U_{2\times2} = \begin{bmatrix} e^{-i\phi} & 0 \\ 0 & e^{i\phi} \end{bmatrix} \begin{bmatrix} cos(\theta) & sin(-\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} \begin{bmatrix} e^{-i\psi} & 0 \\ 0 & e^{i\psi} \end{bmatrix} \begin{bmatrix} e^{i\alpha} & 0 \\ 0 & e^{i\alpha} \end{bmatrix}$$

Właściwa długość chromosomu znajdowana jest za pomocą wzoru (6.2). Gdy pożądaną dokładnością parametrów ϕ, θ, ψ i α jest 9 cyfr znaczących, to chromosom stanowią ciągi o długości $4 * 33 = 132$ bitów.

Celem algorytmu było znalezienie operatora dla następującego zbioru par $\{(|x_i\rangle, |y_i\rangle)\}$ (przykład zaczerpnięty z [Ven00, TV06]):

$$S = \left\{ \begin{array}{c} \left(\frac{1}{\sqrt{5}} \begin{bmatrix} 2 \\ 1 \end{bmatrix}, \frac{1}{\sqrt{10}} \begin{bmatrix} 3 \\ 1 \end{bmatrix} \right), \\ \left(\frac{1}{\sqrt{20}} \begin{bmatrix} -2 \\ 4 \end{bmatrix}, \frac{1}{\sqrt{40}} \begin{bmatrix} 2 \\ -6 \end{bmatrix} \right) \end{array} \right\}$$

Prawidłowym rozwiązaniem dla takich danych wejściowych była macierz $U_{2\times2}$:

```
[[ 0.70710678  0.70710678]
 [ 0.70710678 -0.70710678]]
```

w tym przypadku jest więc to po prostu bramka Hadamarda.

Dobrane parametry zaimplementowanego algorytmu genetycznego, z którym zostały przeprowadzone eksperymenty, przedstawia tabela 6.1.

parametr	wartość	opis
Pc	0.9	prawdopodobieństwo rekombinacji
Pm	0.01	prawdopodobieństwo mutacji
N	40	liczność populacji
precision	9	liczba cyfr znaczących
chromlen	132	długość chromosomu
elitism	5	liczba osobników elitarnych[a]
selekcja	turniej	metoda selekcji
Ngen	40	liczba iteracji algorytmu

Tabela 6.1: Parametry prostego algorytmu genetycznego

[a] taka liczba najlepszych osobników przechodzi do reprodukcji, niezależnie od uwarunkowań selekcji

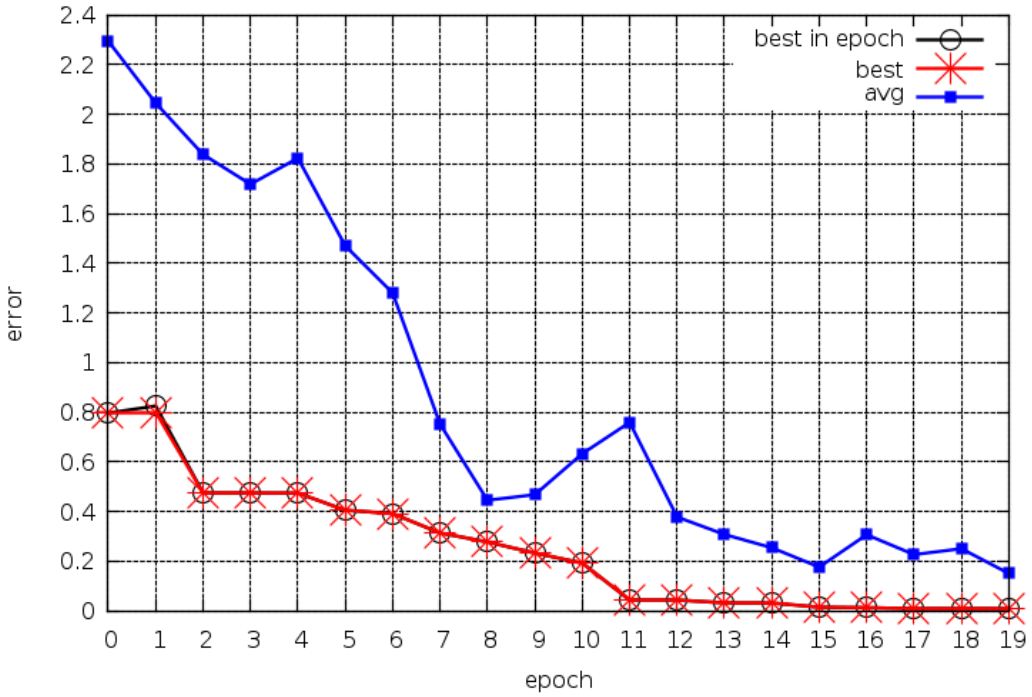
Jak można zauważyć, dobrane parametry przyjmują typowe wartości dla prostego algorytmu genetycznego. Warunkiem stopu było osiągnięcie docelowej liczby iteracji algorytmu (epok). Zastosowano mutację oraz krzyżowanie jednopunktowe.

Na rysunku 6.3 na stronie 87 przedstawiony jest wykres funkcji przystosowania w kolejnych epokach algorytmu genetycznego. W przeprowadzonym eksperymencie było potrzebne 19 epok, aby algorytm znalazł właściwe rozwiązanie (optimum globalne) z bardzo dużą dokładnością. Na rysunku przedstawione jest przystosowanie najlepszego osobnika w ogóle (*best*), najlepszego osobnika w danej epoce (*best in epoch*) oraz średnie przystosowanie osobnika w danej epoce (*avg*). Wykresy *best in epoch* i *best* prawie się pokrywają, wynika to z zastosowania elitarności na etapie selekcji — kilka najlepszych osobników zawsze przechodzi do etapu reprodukcji, więc najlepsze osobniki w danej opoce są ewentualnie nieznacznie gorsze od najlepszych osobników w poprzednich epokach.

Ostateczne rozwiązanie znalezione przez prosty algorytm genetyczny:

```
[[ 0.70710670 +3.39296048e-07j  0.70710687 +1.39649322e-07j]
 [ 0.70710687 +5.66355579e-08j -0.70710670 +1.43011215e-07j]]
```

Błąd znalezionego rozwiązania: `9.337e-14`



Rysunek 6.3: Wykresy błędów średnich, minimalnych i minimalnych w danej epoce dla eksperymentu z prostym algorytmem genetycznym

6.4 Zmodyfikowany algorytm genetyczny

Podstawowym mankamentem poprzedniego podejścia jest ograniczenie do znajdowania jedynie macierzy rozmiaru 2×2, czyli jednokubitowych bramek kwantowych.

W publikacji [FTG03] został zaproponowany zmodyfikowany algorytm genetyczny, dostosowany do znajdowania macierzy unitarnych dowolnych rozmiarów. Algorytm posiada — jak się okaże — niektóre elementy charakterystyczne dla strategii ewolucyjnej. Może być zastosowany także do znajdowania ogólnych operatorów liniowych (nie tylko unitarnych).

Algorytm wykorzystuje kodowanie rzeczywiste[2] (charakterystyczne dla strategii ewolucyjnych). Genotyp osobniczy reprezentowany jest bezpośrednio w postaci macierzy $n \times n \in \mathbb{R}^{n \times n}$. Allele stanowią wartości rzeczywiste, ograniczone do przedziału $\langle -1; 1 \rangle$. Początkową populację stanowią losowe macierze danego rozmiaru, wygenerowane za pomocą algorytmu przedstawionego w jednym z poprzednich rozdziałów. Funkcja oceny, która ma być optymalizowana przez algorytm, określona jest, tak jak poprzednio, za pomocą normy Frobeniusa macierzy, zgodnie ze wzorem 6.4 na stronie 82.

Przykładowe, zalecane parametry tego algorytmu genetycznego przedstawia tabela 6.2.

parametr	wartość	opis
Pc	0.85	prawdopodobieństwo rekombinacji
Pm	**0.95**	prawdopodobieństwo mutacji
Ps	**0.3**	napór selekcyjny
N	200	liczność populacji
elitism	30	liczba osobników elitarnych
Nm	2	liczba genów poddawanych mutacji
perturb	0.05	maksymalny zakres mutacji genu
Ngen	100	maksymalna liczba pokoleń

Tabela 6.2: Parametry zmodyfikowanego algorytmu genetycznego

Algorytm wykorzystuje na etapie selekcji elementy strategii elitarnej. W każdym pokoleniu pewna liczba najlepszych osobników zawsze przechodzi bezpośrednio do następnej generacji. Liczba tych osobników określona jest przez zmienną *elitism*. Pozostała część populacji tworzona jest za pomocą operatorów rekombinacji i mutacji, działających na losowych osobnikach

[2] Algorytm oraz przykładowe eksperymenty numeryczne zostaną zaprezentowane dla unitarnych macierzy rzeczywistych, jednak zasadę działania można uogólnić także na ciało liczb zespolonych

spośród najlepszych w danej epoce. Zmienna *Ps* określa, spośród jakiej części najlepszych osobników, będą losowane osobniki, przechodzące do reprodukcji i poddawane działaniu operatorów genetycznych.

Rekombinacja w tym zmodyfikowanym algorytmie polega na krosowaniu równomiernym — potomek tworzony jest z elementów dwóch macierzy, uczestniczących w rekombinacji, w ten sposób, że kolejne elementy pobierane są z losowego rodzica.

Operator mutacji powoduje natomiast zaburzanie wartości genu o losową wartość z przedziału $\langle -perturb; perturb \rangle$. W macierzy poddawanej mutacji Nm elementów macierzy poddawanych jest takiej losowej modyfikacji.

Algorytm posiada zatem elementy charakterystyczne dla strategii ewolucyjnej: reprezentację rzeczywistą, krosowanie równomierne, a bardzo ważną rolę odgrywa mutacja (duże prawdopodobieństwo mutacji).

Eksperyment 2: macierz 4×4

Znajdowana ma być macierz unitarna rozmiaru 4×4, czyli dwukubitowa bramka kwantowa, która ma wykonywać następujące odwzorowania (przykład zaczerpnięty bezpośrednio z [FTG03]):

$$S = \left\{ \begin{array}{l} \left(\frac{1}{\sqrt{14}} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 0 \end{bmatrix}, \frac{1}{\sqrt{14}} \begin{bmatrix} 2 \\ 3 \\ 1 \\ 0 \end{bmatrix} \right), \left(\frac{1}{\sqrt{26}} \begin{bmatrix} 0 \\ 5 \\ 0 \\ 1 \end{bmatrix}, \frac{1}{\sqrt{26}} \begin{bmatrix} 5 \\ 0 \\ 0 \\ 1 \end{bmatrix} \right), \\ \left(\frac{1}{\sqrt{85}} \begin{bmatrix} 9 \\ 2 \\ 0 \\ 0 \end{bmatrix}, \frac{1}{\sqrt{85}} \begin{bmatrix} 2 \\ 0 \\ 9 \\ 0 \end{bmatrix} \right), \left(\frac{1}{\sqrt{56}} \begin{bmatrix} 2 \\ 4 \\ 6 \\ 0 \end{bmatrix}, \frac{1}{\sqrt{56}} \begin{bmatrix} 4 \\ 6 \\ 2 \\ 0 \end{bmatrix} \right) \end{array} \right\},$$

Właściwe rozwiązanie stanowi w tym przypadku macierz:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Zastosowane parametry algorytmu (nieznacznie zmienione w stosunku do wartości zalecanych w [FTG03]) przedstawione są w tabeli 6.2.

Rozwiązanie znalezione przez algorytm genetyczny:

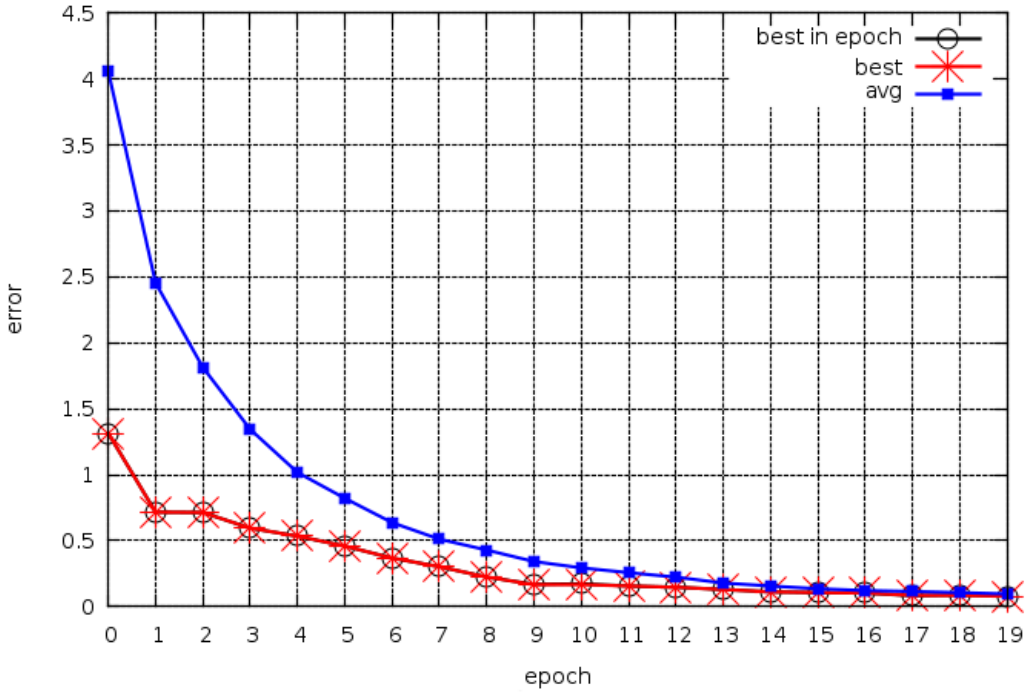
```
[[  0.01146880   0.937319989   0.040377419   0.309432930 ]
 [ -0.01242673   0.087410308   0.965340177  -0.307910461 ]
 [  1.00000000   0.001670577  -0.000561650   0.002388655 ]
 [ -0.08441850   0.364231730  -0.216343760  -0.81890440  ]]
```

Błąd znalezionego rozwiązania: $3.22548479429e-05$

Wykresy błędów, analogiczne jak w poprzednim eksperymencie, przedstawione są na rysunku 6.4. Na wykresach widoczna jest wykładnicza zbieżność średniego błędu populacji do zera.

W tym przypadku algorytmowi genetycznemu udało się znaleźć w ciągu 20 epok macierz, której elementy są zgodne z optymalną macierzą A z dokładnością do ok. trzech cyfr znaczących. Wynik ten mógłby być znacznie poprawiony przez zwiększenie liczby iteracji algorytmu.

Dodatkowo autorzy publikacji [FTG03] zalecają stopniowe zmniejszanie zakresu mutacji $\langle -perturb; perturb \rangle$ w kolejnych epokach w celu szybszego otrzymywania rozwiązań o dużej dokładności (to zalecenie nie było uwzględnione w niniejszym eksperymencie).



Rysunek 6.4: Wykresy błędów średnich, minimalnych i minimalnych w danej epoce

Eksperyment 3: macierz 4×4

W eksperymencie będzie optymalizowana macierz unitarna 4×4, czyli poszukiwana będzie pewna dwukubitowa bramka kwantowa.

Przypuśćmy, że pewien algorytm kwantowy wymaga bramki, która dla kubitu q zwraca $2q$ i $2q+1$ z równymi prawdopodobieństwami. Taki operator unitarny wykonywałby zatem odwzorowanie:

$$|00\rangle \rightarrow \frac{\sqrt{2}}{2}(|00\rangle + |01\rangle)$$

$$|01\rangle \rightarrow \frac{\sqrt{2}}{2}(|10\rangle + |11\rangle)$$

Czy istnieje operator unitarny, działający w ten sposób, oraz — jeśli istnieje — to jak on wygląda?

Dane wejściowe stanowi w tym przypadku zbiór par:

$$S = \left\{ \left(|00\rangle, \frac{\sqrt{2}}{2} (|00\rangle + |01\rangle) \right), \left(|01\rangle, \frac{\sqrt{2}}{2} (|10\rangle + |11\rangle) \right) \right\}$$

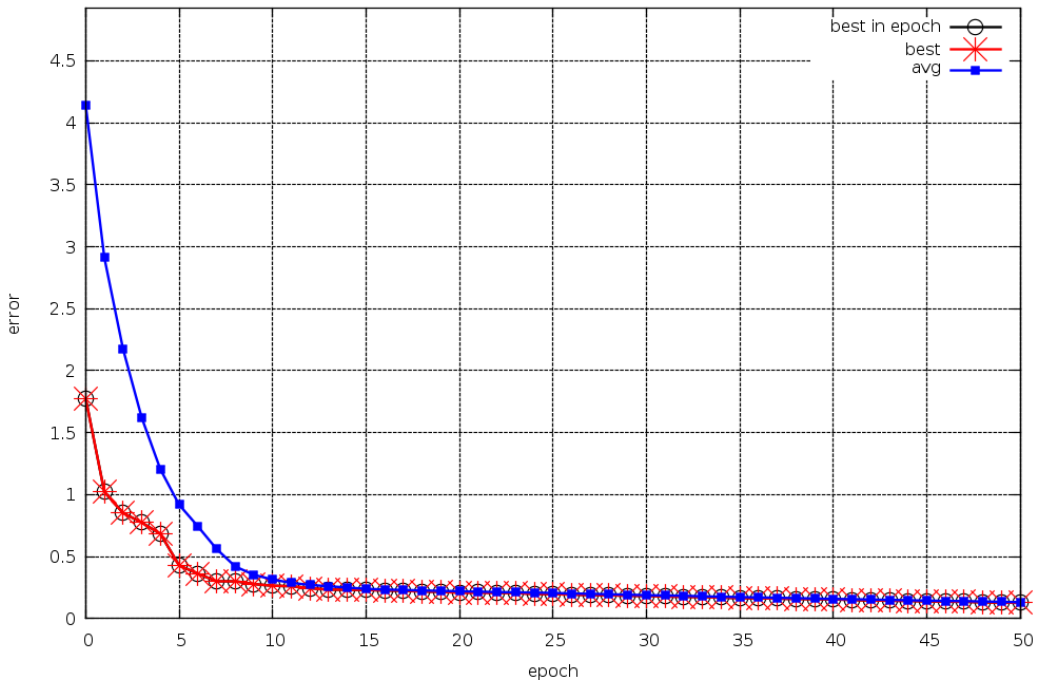
W niniejszym eksperymencie zastosowano takie same parametry jak poprzednio — zgodne z tabelą 6.2 na stronie 88.

Rozwiązanie, które zostało znalezione przez zmodyfikowany algorytm genetyczny, stanowi macierz:

```
[[ 0.7058177  -0.00139617  0.49949132  0.49891891]
 [ 0.70844042 -0.00809113 -0.49876789 -0.50100109]
 [-0.00093399  0.70620229  0.49109668 -0.49712111]
 [ 0.00424357  0.70642918 -0.49954462  0.49926453]]
```

Błąd znalezionego rozwiązania: $4.94789062079e-06$

W eksperymencie zostały zastosowane takie same parametry jak w poprzednim przypadku. Wykresy dla tego eksperymentu ze zmodyfikowanym algorytmem genetycznym przedstawia rysunek 6.5.

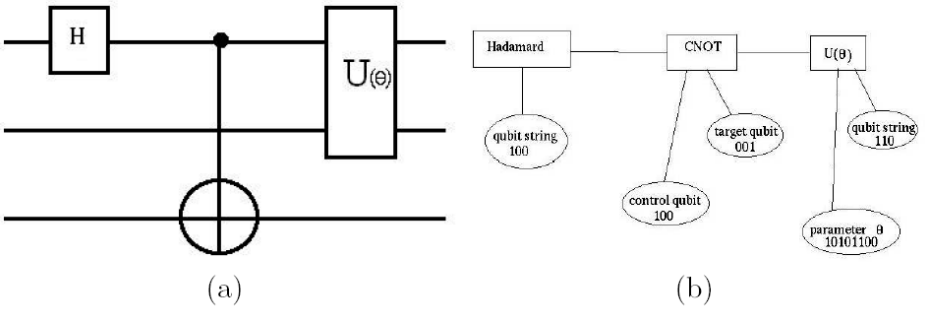


Rysunek 6.5: Wykresy błędów średnich, minimalnych i minimalnych w danej epoce

6.5 Projektowanie obwodów kwantowych

Projektowanie obwodów kwantowych, które będą wykonywały pożądane działania, jest wymagającym zadaniem. Wynika to z nieintuicyjnych własności mechaniki kwantowej oraz niewielkiej analogii między algorytmami klasycznymi a kwantowymi.

Różne możliwe podejścia w tworzeniu systemów hybrydowych, w których wykorzystuje się metody ewolucyjne w projektowaniu obwodów kwantowych, przedstawiono m.in. w [YI00, WG04, Rub00, SBBS99b]. W niniejszym rozdziale zostaną zaprezentowane (oraz poparte eksperymentami) niektóre z nich.



Rysunek 6.6: Przykładowy obwód kwantowy (a) oraz jego reprezentacja w postaci struktury – drzewa (b)

Zaproponowana w publikacji [Rub00] reprezentacja przykładowego obwodu kwantowego w postaci listy węzłów przedstawiona jest na rysunku 6.6. Każda z bramek w obwodzie kwantowych reprezentowana jest za pomocą węzła. Bramki mogą posiadać dodatkowo parametry: numer kubitu, na którym działa bramka, kubit kontrolny lub parametr określający np. kąt obrotu w przestrzeni stanów kubitu. Wymienione parametry bramki reprezentowane są w postaci ciągu binarnego.

Operatory genetyczne, przetwarzające tak reprezentowane obwody kwantowe, odpowiadają wymianie węzłów pomiędzy grafami (w przypadku rekombinacji) lub wstawianie losowo wygenerowanych węzłów (w przypadku mutacji).

Rekombinacja dla takiej struktury danych może przebiegać według jednego (losowo wybranego) ze sposobów: na poziomie bramek, łańcuchów binarnych, określających numer kubitu lub kubit sterujący lub parametr bramki. Krosowanie na poziomie bramek polega na wybraniu losowego punktu krosowania w każdym z dwóch obwodów podlegających rekombinacji. Następnie

dokonywana jest wymiana fragmentów obwodów przed i za punktami krosowania – na zasadzie takiej jak w klasycznym krosowaniu jednopunktowym.

Jeśli została wylosowana rekombinacja na poziomie łańcuchów binarnych, oznaczających kubity, na których działają bramki, to losowane są dwa węzły w obwodach, podlegających rekombinacji. Odpowiednie ciągi binarne są następnie poddawane „zwykłemu" krosowaniu jednopunktowemu. Krosowanie tego rodzaju może dotyczyć także wyłącznie ciągów binarnych, oznaczających numer kubitów kontrolnych lub ewentualnych parametrów bramek odpowiadających węzłom.

Mutacja w tym algorytmie polega na zastąpieniu losowego węzła nowym węzłem — rodzaj bramki oraz numery kubitów są dobierane losowo.

Przykładowe parametry dla opisanego algorytmu programowania genetycznego, służącego do automatycznego projektowania obwodów kwantowych, przedstawia tabela 6.3.

parametr	wartość	opis
Pc	0.75	prawdopodobieństwo rekombinacji
Pm	0.05	prawdopodobieństwo mutacji
N	100	liczność populacji
nstages	5	maksymalna długość obwodu
elitism	5	liczba osobników elitarnych
selekcja	ruletka	metoda selekcji
Ngen	100	liczba iteracji algorytmu
Nm	1	liczba mutowanych węzłów

Tabela 6.3: Parametry algorytmu programowania genetycznego dla automatycznego projektowania obwodów kwantowych

Eksperymenty, które zostaną przedstawione w dalszej części, będą korzystały jedynie z bramek pozbawionych parametrów[3], ponieważ zgodnie z regułami dekompozycji macierzy unitarnych, przedstawionymi na stronie 83, każda bramka kwantowa może być przybliżona z dowolną dokładnością za pomocą uniwersalnego, skończonego podzbioru bramek np. { H, S, CNot, $\pi/8$ }.

Podobnie jak w przypadku optymalizacji bramek kwantowych, automatyczne projektowanie obwodu kwantowego jest traktowane jako zadanie minimalizacji pewnej funkcji błędu, która może być określona jako $\epsilon(U) = \sum_i ||o_i\rangle - |y_i\rangle|$, gdzie $|o_i\rangle$ i $|y_i\rangle$ oznaczają odpowiednio pożądane i otrzymywane stany na wyjściu obwodu.

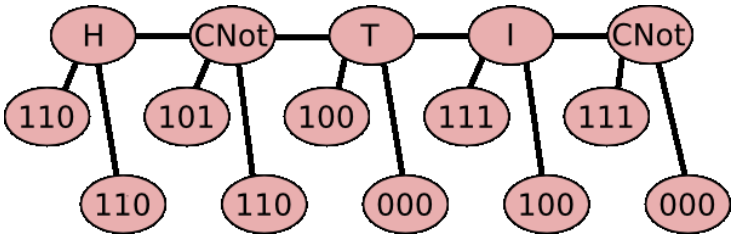
[3]bramki, które posiadają parametry, to bramki takie jak np. $R_x(\theta)$ ze strony 32

Eksperyment 4: trzykubitowe stany splątane

Celem tego eksperymentu będzie zaprojektowanie obwodu kwantowego, który generuje stan splątany $\frac{\sqrt{2}}{2}(|000\rangle + |111\rangle)$ ze stanu bazowego $|000\rangle$ trzykubitowego rejestru kwantowego. Algorytm generowania takiego stanu przedstawiony był w rozdziale 5.1 na stronie 59, a schemat odpowiadającego mu obwodu kwantowego przedstawiał rysunek 5.1. Spróbujmy za pomocą algorytmu programowania genetycznego znaleźć taki obwód — lub inny, równoważny obwód, realizujący to zadanie.

W przeprowadzonym eksperymencie zastosowano algorytm przedstawiony w poprzednim podrozdziale oraz parametry przedstawione w tabeli 6.3. Przestrzeń poszukiwań została ograniczona do maksymalnie pięcioetapowych obwodów, składających się z bramek { H, CNot, $\pi/8$ }. Optymalne rozwiązanie zostało znalezione po 21 epokach.

Rozwiązanie, które zostało znalezione przez algorytm programowania genetycznego, stanowi genotyp:

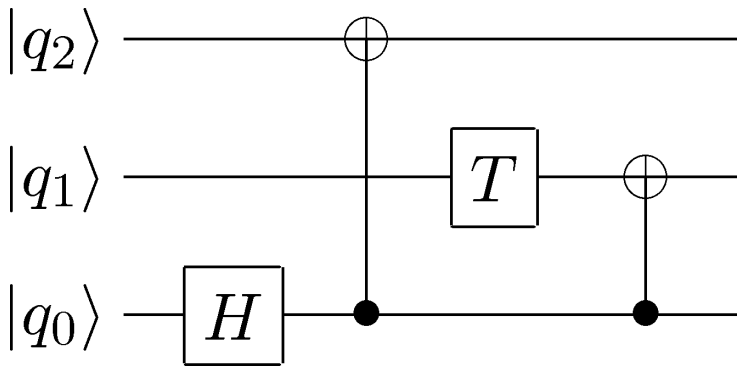


Rysunek 6.7: Reprezentacja w postaci drzewa znalezionego obwodu kwantowego

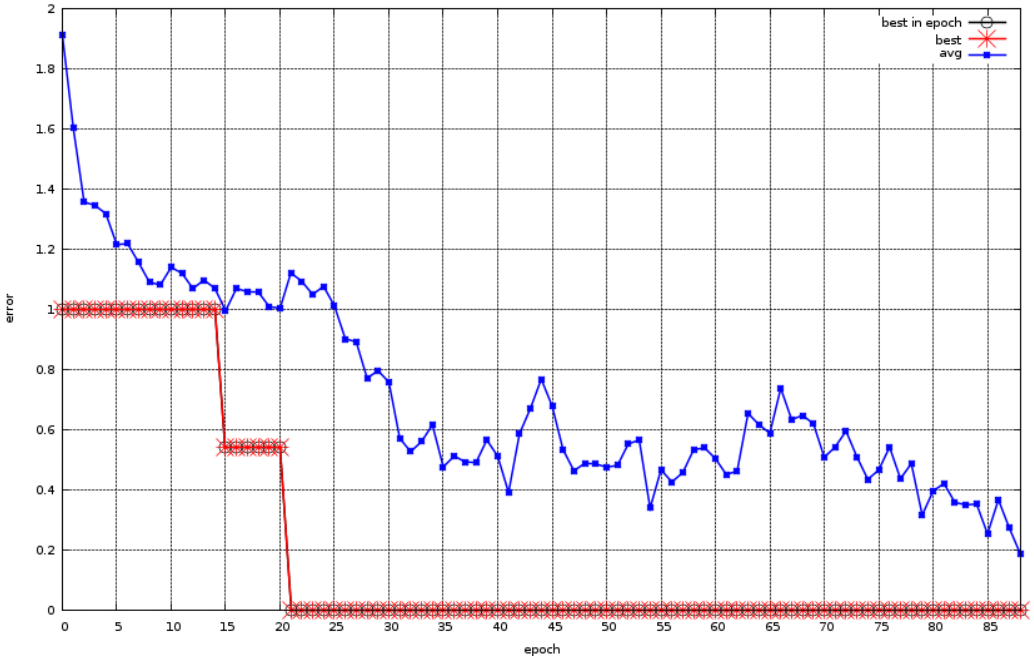
Na powyższym pierwszy ciąg binarny, powiązany z węzłem bramki, oznacza numer kubitu, na którym działa bramka. Drugi ciąg binarny (prawy podwęzeł) oznacza numer kubitu sterującego (dotyczy to tylko bramek $CNot$). Dodatkowo założono, że wartość ciągów binarnych jest dzielona modulo przez liczbę kubitów w obwodzie. Obwód odpowiadający temu genotypowi przedstawia zatem rysunek 6.8.

Wykresy błędów w kolejnych epokach algorytmu w tym eksperymencie przedstawia rysunek 6.9.

Ponieważ obwód kwantowy miał być automatycznie zaprojektowany za pomocą skończonego, uniwersalnego podzbioru bramek kwantowych { H, CNot, $\pi/8$ }, to zadanie to miało charakter problemu *kombinatorycznego*. Po znalezieniu przez algorytm właściwej kombinacji tj. poprawnego układu bramek kwantowych błąd minimalny spadł do zera w 21 epoce. Z wykresu można także odczytać, że w dalszych epokach średni błąd w danej epoce miał w dalszym ciągu tendencję malejącą.



Rysunek 6.8: Alternatywny obwód kwantowy, znaleziony za pomocą programowania genetycznego, generujący stan splątany $\frac{\sqrt{2}}{2}(|000\rangle + |111\rangle)$ ze stanu bazowego $|000\rangle$ trzykubitowego rejestru kwantowego



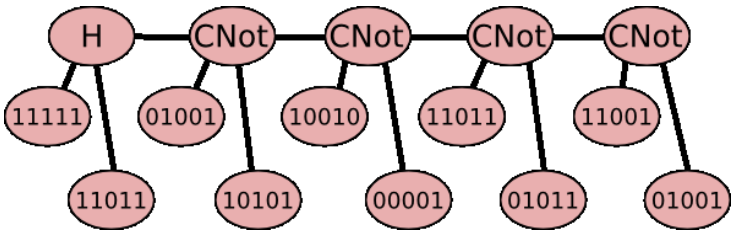
Rysunek 6.9: Wykresy błędów dla eksperymentu z programowaniem genetycznym

Eksperyment 5: 5-kubitowe stany splątane

W poprzednim eksperymencie był przedstawiony przykład zaprojektowania obwodu kwantowego, który był już zaprezentowany w poprzednich rozdziałach. W niniejszym eksperymencie zostanie znaleziony obwód kwantowy, który nie był w pracy dotąd przedstawiony.

Poszukiwany będzie obwód kwantowy, generujący stan splątany $\frac{\sqrt{2}}{2}(|00000\rangle + |11111\rangle)$ ze stanu bazowego $|00000\rangle$ pięciokubitowego rejestru kwantowego. W tym eksperymencie uległy drobnej zmianie parametry w stosunku do wartości z tabeli 6.3. Została zwiększona liczność populacji ($poplen = 500$), liczba osobników elitarnych ($elitism = 80$). Zostało także zmniejszone prawdopodobieństwo mutacji ($Pm = 0.002$).

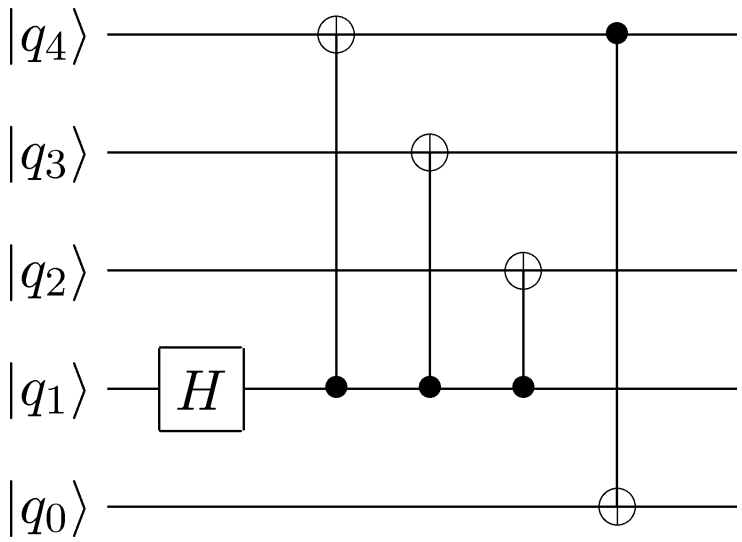
Rozwiązanie, które zostało automatycznie znalezione za pomocą algorytmu, stanowi genotyp:



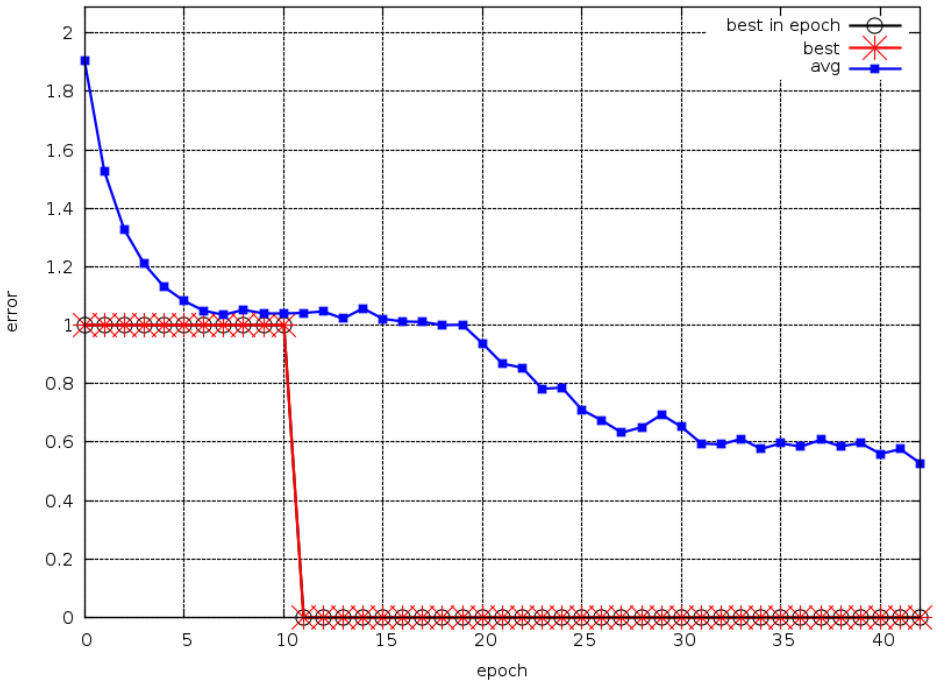
Rysunek 6.10: Reprezentacja w postaci drzewa znalezionego obwodu kwantowego

Schemat obwodu kwantowego, odpowiadający temu genotypowi przedstawia rysunek 6.11.

W przypadku tego eksperymentu najlepsze rezultaty otrzymywano dopiero po dobraniu odpowiednich parametrów dotyczących etapu selekcji. W przypadku bardzo dużych populacji oraz niewielkiego zróżnicowania przystosowań selekcja turniejowa lub za pomocą metody ruletki może nie dawać zbyt dobrych rezultatów — np. prawdopodobieństwo selekcji rozwiązania bliskiego optymalnego może wynieść 2%, a prawdopodobieństwo selekcji innych rozwiązań 1% (zbyt mały napór selekcyjny). Znacznie lepsze rezultaty może w takim przypadku przynieść zastosowanie selekcji rankingowej z odpowiednio dobraną funkcją rozkładu prawdopodobieństwa selekcji.



Rysunek 6.11: Obwód kwantowy, znaleziony za pomocą programowania genetycznego, generujący stan splątany $\frac{\sqrt{2}}{2}(|00000\rangle + |11111\rangle)$ ze stanu bazowego $|00000\rangle$ pięciokubitowego rejestru kwantowego



Rysunek 6.12: Wykresy błędów dla eksperymentu z programowaniem genetycznym

Minimalizacja obwodów kwantowych

Dla klasycznego rachunku zdań istnieją deterministyczne metody pozwalające na minimalizację formuł boolowskich: tablice Veitch'a, diagramy Carnougha, algorytm Quine'a-McKluskey'a. Podobnie zagadnienie *minimalizacji* rozpatruje się w przypadku obliczeń kwantowych. Celem takiej optymalizacji jest znalezienie najprostszego obwodu, realizującego pożądane obliczenia.

Skuteczne rozwiązanie takiego problemu za pomocą algorytmu genetycznego zostało przedstawione m.in. w [YI00]. Autorom tej publikacji udało się zaprojektować prostszy obwód kwantowy, realizujący protokół teleportacji kwantowej (omówionej w podrozdziale 5.2 na stronie 61) niż równoważny obwód, zaprojektowany wcześniej przez człowieka. Teleportacja kwantowa w modelu kwantowych bramek logicznych, przedstawiona po raz pierwszy w [Bra96], wykorzystywała obwód, składający się z jedenastu bramek kwantowych. W publikacji [YI00] za pomocą algorytmu genetycznego zaprojektowano równoważny obwód składający się z ośmiu bramek.

Optymalizacja obwodów kwantowych może być wykonywana pod względem różnych kryteriów. Rozważany może być różny zestaw bramek elementarnych, z których algorytm genetyczny buduje obwód kwantowy. Celem algorytmu może być zminimalizowanie liczby użytych bramek lub etapów obliczeniowych (długość obwodu). Wszystkie te kryteria są istotne dla potencjalnej fizycznej realizacji takich układów, ponieważ wraz ze wzrostem złożoności i wielkości obwodu realizującego obliczenia kwantowe, rośnie prawdopodobieństwo pojawienia się błędów i trudność w uniknięciu przedwczesnej dekoherencji.

W publikacji [YI00], gdzie została zaprezentowana skuteczna metoda minimalizacji obwodów kwantowych, zostało wykorzystane nieco inne podejście niż przedstawione w poprzednich podrozdziałach. Celem algorytmu było znalezienie obwodu kwantowego, równoważnego obwodowi 5.4 ze strony 62, realizującego protokół teleportacji kwantowej. Taki obwód kwantowy wykonuje operacje na trzech kubitach. W zastosowanym podejściu przyjęto następujące założenia:

- Reprezentacja całkowitoliczbowa rozwiązań — obwody kwantowe reprezentowane są za pomocą ciągów liczbowych
- Silne heurystyczne założenia aprioryczne — w zastosowanej metodzie ewolucyjnej została zawarta wstępna wiedza o poszukiwanym obwodzie kwantowym

- Chromosomy dzielone były na kodony (ang. *codons*), w tym przypadku były to podłańcuchy o długości trzech cyfr

$$\overbrace{112|231|001|331}^{\text{EPR-pair}}\,|\,\overbrace{132|012|221|302}^{\text{Alice's part}}\,|\,\overbrace{001|100|002|201}^{\text{Bob's part}}$$

- Przejście z przestrzeni genotypów do przestrzeni fenotypów wykonywane było za pomocą specjalnych tabel (str. 101).

Do przyjętych apriorycznych założeń należały:

- Teleportacja kwantowa musi się rozpocząć od wygenerowania splątanej pary EPR, która będzie stanowiła kwantowy kanał transmisji.
- Etap generacji pary EPR może się składać z bramek działających jedynie na dwóch kubitach.
- Strona wysyłająca i odbierająca mogą wykonywać operacje jedynie na posiadanych przez nie, odpowiednio, parach kubitów.
- Dozwolona jest tylko jednokrotna operacja pomiaru stanu, wykonywana u strony wysyłającej.

W publikacji [YI00] zastosowano krosowanie dwupunktowe, z prawdopodobieństwem krosowania $Pc = 0.7$. Liczność populacji wynosiła $N = 5000$. Przyjęto, że przestrzenią poszukiwań będą obwody, składające się jedynie z bramek { CNot, L, R }. Bramki L i R opisywane są przez macierze, które były przedstawione w sekcji 5.2.

Tabele 6.4, 6.4 i 6.4 służą do odwzorowywania kodonów na elementy obwodów kwantowych. Pozwalają one odczytać, jakim bramkom kwantowym odpowiadają dane łańcuchy liczbowe. Indeksy dolne przy nazwach bramek w tabelach oznaczają numery kubitów, na których działają bramki.

Za pomocą tego podejścia autorom [YI00] udało się znaleźć obwód przedstawiony na rysunku 6.14 na stronie 103. Obwód ten jest opisywany za pomocą genotypu:

$$112|231|001|331|132|012|121|302|220|020|001$$

	0	1	2	3	
0	$CNOT_{01}$ $CNOT_{01}$ $CNOT_{01}$	$CNOT_{10}$ $CNOT_{10}$ $CNOT_{10}$			0 1 2 3
1	L_0 L_0 L_0	L_1 L_1 L_1			0 1 2 3
2	R_0 R_0 R_0	R_1 R_1 R_1			0 1 2 3
3					*

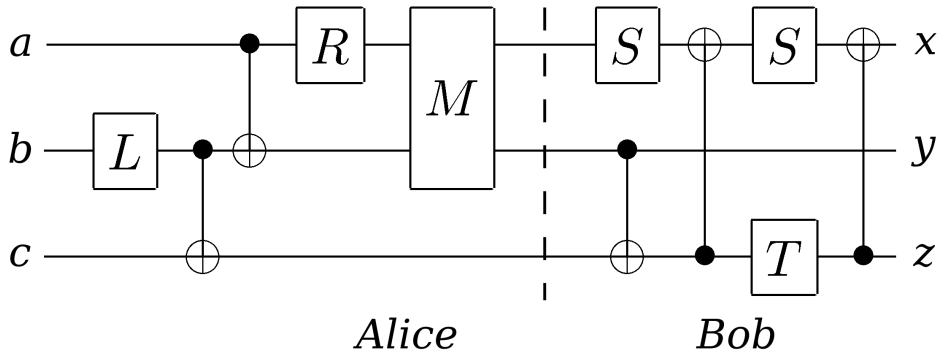
Tabela 6.4: Tabela przejścia do przestrzeni fenotypów dla fragmentu obwodu generującego parę EPR

	0	1	2	3	
0		$CNOT_{12}$ $CNOT_{12}$ $CNOT_{12}$	$CNOT_{21}$ $CNOT_{21}$ $CNOT_{21}$		0 1 2 3
1		L_1 L_1 L_1	L_2 L_2 L_2		0 1 2 3
2		R_1 R_1 R_1	R_2 R_2 R_2		0 1 2 3
3	pomiar				*

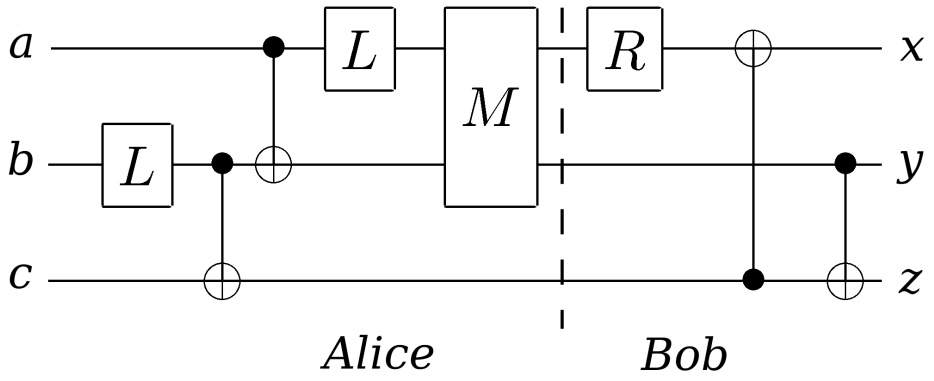
Tabela 6.5: Tabela przejścia do przestrzeni fenotypów dla części wysyłającej obwodu

	0	1	2	3	
0		$CNOT_{10}$	$CNOT_{20}$		0
	$CNOT_{01}$		$CNOT_{21}$		1
	$CNOT_{02}$	$CNOT_{12}$			2
					3
1	L_0	L_1	L_2		0
	L_0	L_1	L_2		1
	L_0	L_1	L_2		2
					3
2	R_0	R_1	R_2		0
	R_0	R_1	R_2		1
	R_0	R_1	R_2		2
					3
3					*

Tabela 6.6: Tabela przejścia do przestrzeni fenotypów dla części odbierającej obwodu



Rysunek 6.13: Oryginalny obwód teleportacji kwantowej z [Bra96]



Rysunek 6.14: Zoptymalizowany obwód kwantowy, przedstawiony w [YI00], znaleziony za pomocą algorytmu genetycznego

Zmodyfikowane operatory genetyczne

W publikacji [WG04] zaproponowano zmodyfikowane operatory genetyczne dla projektowania obwodów kwantowych za pomocą programowania genetycznego.

- **Mutacja**. Modyfikacji podlega losowo wybrany węzeł drzewa. Jeśli węzłem jest bramka, posiadająca parametry, to podlegają one losowej modyfikacji. Dla bramek nie posiadających parametrów, zmieniane są wartości, określające kubity, na których działa bramka.
- **Zamiana**. Z obwodu wybierane są losowo dwie bramki, które są następnie zamieniane miejscami w obwodzie.
- **Krosowanie**. Operator działa na dwóch obwodach kwantowych. W obydwu z nich wybierane są losowe punkty krosowania. Nowe obwody kwantowe są zbudowane z odpowiednio pierwszej i drugiej części wylosowanych do krosowania obwodów. Należy zauważyć, że w wyniku działania tego operatora ulegają zmianom także długości obwodów.
- **Transpozycja**. Podobnie jak krosowanie, operator działa na dwóch obwodach podlegających reprodukcji. Z pierwszego obwodu wybierany jest losowo „podobwód" (poprzez losowe wybranie dwóch punktów w tym obwodzie) — tak znaleziona część jest wstawiana w losowe miejsce drugiego obwodu.
- **Wstawienie**. Operator działa na pojedynczym obwodzie. W losowym punkcie obwodu jest wstawiany wygenerowany zupełnie losowo podobwód. Działanie tego operatora powoduje zatem zwiększenie długości obwodu.
- **Usunięcie**. Operator jest odwrotnością wstawienia. Losowo wybrana część obwodu kwantowego jest usuwana. W rezultacie otrzymywany jest obwód o mniejszej długości.

Niektóre z powyższych, zmodyfikowanych operatorów mogą powodować zmianę długości obwodu (liczbę węzłów). Stosowne jest zatem nałożenie ograniczenia na maksymalną i minimalną długość obwodu, podobnie jak ogranicza się maksymalną głębokość drzewa w „zwykłym" programowaniu genetycznym.

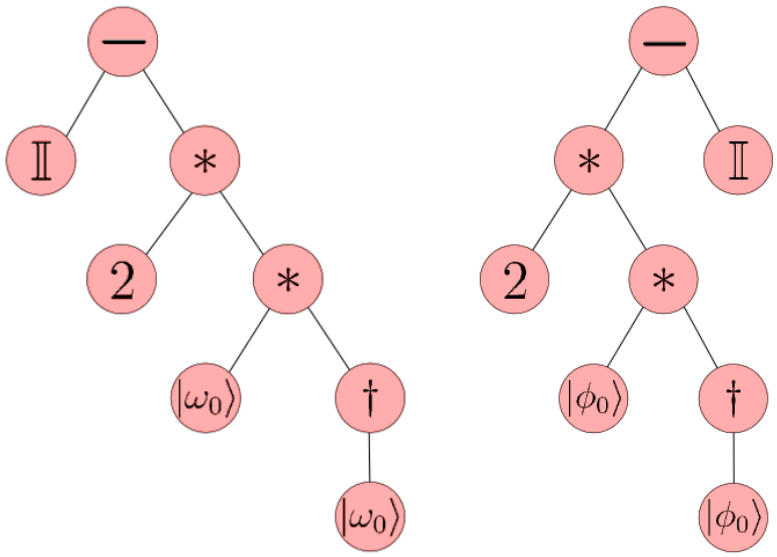
Reprezentacja symboliczna operatorów

Algorytm Grovera, przedstawiony w podrozdziale 5.4, wykorzystuje iteracyjne wykonywanie dwóch operacji kwantowych $\mathcal{A}$ i $\mathcal{B}$, danych za pomocą wyrażeń:

$$\mathcal{A} = \mathbb{I} - 2|\omega_0\rangle\langle\omega_0|$$

$$\mathcal{B} = 2|\phi_0\rangle\langle\phi_0| - \mathbb{I}$$

Zagadnienie projektowania obliczeń kwantowych, poprzez posługiwanie się takimi wyrażeniami symbolicznymi, za pomocą programowania genetycznego wydaje się być obiecującym kierunkiem do dalszych badań. Przykładowa reprezentacja operatorów $\mathcal{A}$ i $\mathcal{B}$ za pomocą drzew przedstawiona jest na rysunku 6.15.



(a) $\mathcal{A} = \mathbb{I} - 2|\omega_0\rangle\langle\omega_0|$ (b) $\mathcal{B} = 2|\phi_0\rangle\langle\phi_0| - \mathbb{I}$

Rysunek 6.15: Przykładowa reprezentacja operatorów $\mathcal{A}$ i $\mathcal{B}$ algorytmu Grovera w postaci drzew

6.6 Wnioski

W rozdziale zostało wykonane pięć eksperymentów z różnymi algorytmami genetycznymi (z różnymi parametrami) oraz zostały przedstawione możliwe inne podejścia, godne dalszej uwagi.

W pierwszym eksperymencie został zastosowany prosty algorytm genetyczny do znalezienia macierzy unitarnej 2×2, realizującej pożądane odwzorowanie dla zadanych par wektorów wejściowych i wyjściowych. Metoda była oparta na specyficznym sposobie dokompozycji macierzy unitarnych 2×2, więc posiadała ona ograniczony zakres zastosowań.

W drugim z eksperymentów został przetestowany zmodyfikowany algorytm genetyczny (mający elementy strategii ewolucyjnej), zaproponowany w publikacji [FTG03], przystosowany do znajdowania macierzy unitarnych dowolnych rozmiarów, co jest zaletą w stosunku do poprzedniego podejścia. O skuteczności tego algorytmu świadczy szybka, wykładnicza zbieżność średniego błędu populacji do zera, którą można było odczytać z wykresów 6.4 i 6.5.

Trzeci z eksperymentów prezentował metodę znalezienia bramki kwantowej, wykonującej prostą operację arytmetyczną na kubitach. Zastosowano w tym celu algorytm taki jak w poprzednim eksperymencie, drobnej zmianie uległy jedynie parametry algorytmu.

Przedstawiono również możliwości automatycznego projektowania obwodów kwantowych. Zostało to zilustrowane w eksperymencie, w którym obwód, generujący trzykubitowy stan splątany, został automatycznie zaprojektowany przy pomocy programowania genetycznego. Rezultatem eksperymentu było znalezienie alternatywnego obwodu (6.8), generującego trzykubitowy stan splątany (nie był to jednak obwód optymalny pod względem liczby użytych bramek kwantowych).

W piątym eksperymencie numerycznym został automatycznie zaprojektowany prosty obwód kwantowy (6.11), nieprezentowany wcześniej w pracy, generujący stan splątany pięciokubitowego rejestru kwantowego ze stanu bazowego $|00000\rangle$. Znaleziony obwód kwantowy jest także optymalny pod względem liczby użytych bramek i etapów obliczeń.

W rozdziale zostały także zaprezentowane możliwe nowe, obiecujące kierunki dalszych badań, wykorzystujące metody ewolucyjne w informatyce kwantowej. Symulacja dużych obwodów kwantowych wiąże się z wykładniczym wzrostem złożoności pamięciowej i obliczeniowej. Efektywną metodą przeszukiwania tak dużych przestrzeni rozwiązań mogłoby się okazać zastosowanie migracyjnego modelu HFC.

Podsumowanie

Celem niniejszej pracy były opracowanie środowiska symulacji obliczeń kwantowych oraz zbadanie możliwości wykorzystania metod ewolucyjnych sztucznej inteligencji w różnych aspektach projektowania obliczeń kwantowych.

W pracy został wykorzystany model obliczeniowy obwodów kwantowych oraz została opracowana obiektowa biblioteka dla języka Python. Przy użyciu stworzonej biblioteki zostały zaimplementowane podstawowe algorytmy kwantowe — generowanie stanów splątanych, algorytm Grovera, kodowanie supergęste oraz protokół teleportacji kwantowej.

Zostały zaprezentowane możliwości wykorzystania wybranych metod ewolucyjnych w projektowaniu operatorów unitarnych oraz całych obwodów kwantowych. Obliczenia ewolucyjne okazują się skutecznym narzędziem w różnych aspektach projektowania algorytmów kwantowych.

W dynamicznym rozwoju komputerów kwantowych przeszkadzają obecnie trudności związane z utrzymywaniem „delikatnego" stanu koherencji kwantowej i wiążące się z tym błędy oraz problemy z samym projektowaniem algorytmów kwantowych. Niepożądana dekoherencja układu kwantowego jest bardzo trudna do uniknięcia wraz ze zwiększaniem liczby kubitów w rejestrze kwantowym.

W wielu ośrodkach naukowych na świecie, zajmujących się badaniami nad informatyką kwantową, prowadzone są obecnie prace związane w szczególności z kwantową korekcją błędów oraz projektowaniem obliczeń kwantowych w taki sposób, by do pewnego stopnia były one niewrażliwe na pojawiające się błędy (ang. *fault tolerant quantum computing*). Prawdopodobnie także w takim aspekcie projektowania komputerów kwantowych metody ewolucyjne sztucznej inteligencji byłyby bardzo pomocnym narzędziem.

Wiele zagadnień, pominiętych w niniejszej pracy, mogłoby być interesującym i obiecującym kierunkiem dalszych badań. Należy do nich m.in. wykorzystanie obliczeń kwantowych z punktu widzenia kryptografii i kryptoanalizy oraz metody sztucznej inteligencji, czerpiące z możliwości informatyki kwantowej (kwantowe sieci neuronowe lub algorytmy genetyczne typu QIGA (ang. *quantum inspired genetic algorithm*)).

Streszczenia i słowa kluczowe

słowa kluczowe: informatyka kwantowa, sztuczna inteligencja, obliczenia ewolucyjne, algorytmy genetyczne, programowanie genetyczne

Streszczenie

Przedmiotem niniejszej pracy magisterskiej są obliczenia kwantowe oraz systemy hybrydowe, łączące metody ewolucyjne sztucznej inteligencji i informatykę kwantową. W pracy został przedstawiony model obliczeń kwantowych, wykorzystujący kwantowe bramki logiczne. Zaproponowano model obiektowy dla obliczeń kwantowych, na podstawie którego została stworzona biblioteka qclib, pozwalająca na symulację pracy komputera kwantowego. Za jej pomocą zaimplementowano podstawowe algorytmy kwantowe: algorytm Grovera, protokół teleportacji kwantowej, kodowanie supergęste i generowanie stanów splątanych. Zostały przedstawione podstawowe trudności, związane z projektowaniem obliczeń kwantowych, oraz możliwości wykorzystania metod ewolucyjnych sztucznej inteligencji w kontekście projektowania obliczeń tego rodzaju. Zaprezentowano wykorzystanie prostego oraz zmodyfikowanego algorytmu genetycznego przy projektowaniu operatorów unitarnych, a także wykorzystanie programowania genetycznego przy projektowaniu całych obwodów kwantowych. Niektóre możliwe podejścia zostały zilustrowane eksperymentami numerycznymi.

Abstract and keywords

keywords: quantum computing, artificial intelligence, automated design, evolutionary systems, genetic algorithms, genetic programming

Abstract

This master thesis is focused on quantum computing and its possible applications of genetic algorithms. Quantum computing is considered in the terms of quantum gates computational model. Proposal of object model for quantum computing is presented. Practical part of this thesis consists of the quantum computing library (qclib) allowing to simulate a quantum computer. The basic quantum algorithms are presented and implemented using qclib: Grover's fast search algorithm, quantum teleportation, quantum superdense coding and production of entangled quantum states. Generation of such quantum algorithms is a difficult task for human. They are unintuitive and computationally intensive. For that reason hybrid systems combining quantum computing and genetic algorithms are considered. In this thesis simple and modified genetic algorithms are used and compared to evolve quantum gates. Complete quantum circuits are created using genetic programming. Some of possible approaches are illustrated with numerical experiments.

Załącznik — kody źródłowe

Biblioteka qclib

```
1  #!/usr/bin/python
2  #
3  # Quantum Computing Python Library
4  # Copyright (C) 2008    Robert Nowotniak <robert@nowotniak.com>
5  #
6  # $Id: qclib.py 49 2008-06-16 09:12:08Z rob $
7  #
8  # qclib is free software; you can redistribute it and/or
9  # modify it under the terms of the GNU General Public License as published by
10 # the Free Software Foundation; either version 2 of the License, or
11 # (at your option) any later version.
12 #
13 # qclib is distributed in the hope that it will be useful,
14 # but WITHOUT ANY WARRANTY; without even the implied warranty of
15 # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
16 # GNU General Public License for more details.
17 #
18 # You should have received a copy of the GNU General Public License
19 # along with qclib; if not, write to the Free Software Foundation
20
21 from numpy import *
22 from random import random
23 import copy
24
25
26 # for floating point operations, comparisons etc.
27 epsilon = 10e-6
28
29
30 class QRegister:
31     '''Quantum register class'''
32
33     def __init__(self, m = None):
34         if m == None:
35             return
36         if isinstance(m, ndarray) or type(m) == type([]):
37             m = matrix(m)
38         if isinstance(m, matrix) and m.shape[0] == 1:
39             m = transpose(m)
40         if not isinstance(m, matrix) or m.shape[1] != 1:
41             raise WrongSizeException
42         self.matrix = m
43         self.size = int(math.log(m.size, 2))
44
```

```
45      def __rmul__(self, arg1):
46          # arg1 * self
47          if type(arg1) not in [int, float, complex]:
48              raise Exception()
49          result = copy.deepcopy(self)
50          result.matrix = arg1 * self.matrix
51          return result
52
53      def __add__(self, arg2):
54          # self + arg2
55          result = copy.deepcopy(self)
56          result.matrix = self.matrix + arg2.matrix
57          return result
58
59      def __sub__(self, arg2):
60          # self - arg2
61          result = copy.deepcopy(self)
62          result.matrix = self.matrix - arg2.matrix
63          return result
64
65      def __pow__(self, arg2):
66          # self ** arg2
67          result = QRegister()
68          result.matrix = kron(self.matrix, arg2.matrix)
69          result.size = int(math.log(result.matrix.size, 2))
70          return result
71
72      def __cmp__(self, other):
73          m1 = self.matrix
74          if isinstance(other, (matrix, ndarray)):
75              m2 = other
76          elif isinstance(other, QRegister):
77              m2 = other.matrix
78          else:
79              return -1
80          try:
81              if sum(abs(m1 - m2)) < epsilon:
82                  return 0
83              else:
84                  return 1
85          except Exception:
86              raise WrongSizeException, \
87                      'Comparison of different size quantum registers'
88
89      def __str__(self):
90          return str(self.matrix)
91
92      def reset(self, n = 0):
93          for i in xrange(self.matrix.size):
94              self.matrix[i] = 0
95          self.matrix[n] = 1
96
97      def normalize(self):
98          l = sqrt(sum([abs(x)**2 for x in self.matrix]))
99          self.matrix = self.matrix / l
100         return self
101
102     def measure(self, *qubits):
103         if len(qubits) == 0:
104             # measure all qubits in register
105             qubits = range(int(math.log(self.matrix.size, 2)))
106         qubits = list(qubits)
```

```
107             qubits.sort()
108             p = {} # results probabilities
109             # number of possible measurement results
110             nres = 2 ** len(qubits)
111             # enumerate all posible results
112             for i in xrange(nres):
113                 p[dec2bin(i, int(math.log(nres, 2)))[::-1]] = 0.0
114             for i in xrange(self.matrix.size):
115                 # reversed binary representation of base vector
116                 revbin = dec2bin(i, int(math.log(self.matrix.size, 2)))[::-1]
117                 # reversed binary representation of selected qubits
118                 revsel = ''.join([revbin[q] for q in qubits])
119                 p[revsel] += float(abs(self.matrix[i]) ** 2)
120             keys = p.keys()
121             # accumulated probabilities
122             last = p[keys[0]]
123             for k in keys[1:]:
124                 p[k] += last
125                 last = p[k]
126             p[keys[-1]] = 1.0
127             # get the measurement result according to probabilities
128             r = random()
129             for k in keys:
130                 if r <= p[k]:
131                     result = k
132                     break
133             # selective reset of amplitudes
134             for i in xrange(self.matrix.size):
135                 revbin = dec2bin(i, int(math.log(self.matrix.size, 2)))[::-1]
136                 revsel = ''.join([revbin[q] for q in qubits])
137                 if revsel != result:
138                     self.matrix[i] = 0.0
139             # normalize final state
140             self.normalize()
141             return Ket(int(result[::-1], 2), len(qubits))
142 
143         def dirac(self, reduce = True, binary = True):
144             """Return state in Dirac (bra-ket) notation"""
145             elems = []
146             if len(filter(lambda x: float(abs(x)) > 1 - epsilon, self.matrix)) == 1:
147                 single = True
148             else:
149                 single = False
150             for i in xrange(self.matrix.size):
151                 val = complex(real(self.matrix[i]), imag(self.matrix[i]))
152                 if reduce and abs(val) < epsilon:
153                     continue
154                 if abs(val) < epsilon:
155                     elem = '+0'
156                 elif imag(val) == 0:
157                     elem = '%+g' % abs(val)
158                 elif real(val) != 0:
159                     elem = '+%s' % str(val)
160                 else:
161                     # only imaginary part
162                     elem = '%+gj' % (imag(val))
163                 if single and reduce:
164                     elem = ''
165                 if binary:
166                     elem += ('|%0'+str(math.log(self.matrix.size, 2))+'d>') % int(dec2bin(i))
167                 else:
168                     elem += '|%s>' % i
```

```
169             elems.append(elem)
170         return ' '.join(elems)
171 
172     def outer(self, qreg):
173         '''Compute an outer product with another register'''
174         if self.matrix.size != transpose(qreg.matrix).size:
175             raise WrongSizeException, \
176                     'Outer product of different size registers'
177         result = Arbitrary(dot(self.matrix, transpose(qreg.matrix)))
178         return result
179 
180 
181 
182 class Qubit(QRegister):
183     '''Qubit class'''
184 
185     def __init__(self, val):
186         if not isinstance(val, int):
187             return QRegister.__init__(self, val)
188         self.size = 1
189         if val == 0:
190             self.matrix = transpose(matrix([[1, 0]]))
191         elif val == 1:
192             self.matrix = transpose(matrix([[0, 1]]))
193         else:
194             raise WrongSizeException
195 
196     def flip(self):
197         tmp = self.matrix[0]
198         self.matrix[0] = self.matrix[1]
199         self.matrix[1] = tmp
200 
201 
202 class QCircuit:
203     '''Quantum circuit class'''
204 
205     def __init__(self, *stages):
206         self.stages = stages
207 
208     def __call__(self, qreg):
209         # Efficient algorithm could be implemented here instead. Reference:
210         # Wissam A. Samad, Roy Ghandour, and Mohamad.
211         # Memory efficient quantum circuit simulator based on linked list architecture
212         result = copy.deepcopy(qreg)
213         for s in self.stages:
214             result = s(result)
215         return result
216 
217 
218 class QGate:
219     '''Quantum gate class'''
220 
221     def __pow__(self, arg2):
222         # parallel gates
223         if not isinstance(arg2, QGate):
224             raise Exception(repr(arg2))
225         result = Stage(self, arg2)
226         return result
227 
228     def __str__(self):
229         return str(self.matrix)
230 
```

```
231     def __mul__(self, arg2):
232         # self * arg2
233         if isinstance(arg2, QRegister):
234             # gate * reg
235             result = QRegister()
236             try:
237                 result.matrix = dot(self.matrix, arg2.matrix)
238             except:
239                 raise WrongSizeException, 'Wrong size of input register for this gate'
240             return result
241         if self.matrix.shape != arg2.matrix.shape:
242             raise Exception()
243         # gate * gate
244         result = QGate()
245         # order changed!
246         result.matrix = dot(arg2.matrix, self.matrix)
247         return result
248 
249     def __rmul__(self, arg1):
250         # arg1 * self
251         if type(arg1) not in [int, float, complex]:
252             raise Exception, 'Numerical coefficient expected'
253         result = copy.deepcopy(self)
254         result.matrix = arg1 * self.matrix
255         return result
256 
257     def __add__(self, arg2):
258         # self + arg2
259         result = copy.deepcopy(self)
260         result.matrix = self.matrix + arg2.matrix
261         return result
262 
263     def __sub__(self, arg2):
264         # self - arg2
265         result = copy.deepcopy(self)
266         result.matrix = self.matrix - arg2.matrix
267         return result
268 
269     def __call__(self, qreg):
270         return self.compute(qreg)
271 
272     def compute(self, qreg):
273         if not isinstance(qreg, QRegister):
274             raise Exception()
275         return self * qreg
276 
277     def trace(self):
278         return self.matrix.trace()
279 
280     def determinant(self):
281         return linalg.det(self.matrix)
282 
283     def transpose(self):
284         self.matrix = transpose(self.matrix)
285         return self
286 
287     def inverse(self):
288         self.matrix = linalg.inv(self.matrix)
289         return self
290 
291 
292 class Stage(QGate):
```

```
293     '''Quantum computing stage -- a layer in circuit'''
294
295     def __init__(self, *gates):
296         self.gates = gates
297         m = self.gates[0].matrix
298         for g in self.gates[1:]:
299             m = kron(m, g.matrix)
300         self.matrix = m
301         self.size = sum([g.size for g in gates])
302
303 #
304 # Elementary quantum gates
305 #
306
307 class ElementaryQuantumGate(QGate):
308     pass
309
310
311 class Identity(ElementaryQuantumGate):
312     def __init__(self, size = 1):
313         self.matrix = eye(2 ** size)
314         self.size = size
315
316
317 class Hadamard(ElementaryQuantumGate):
318     def __init__(self, size = 1):
319         h = s2 * matrix([
320             [1, 1],
321             [1, -1]])
322         m = h
323         for i in xrange(size - 1):
324             m = kron(m, h)
325         self.matrix = m
326         self.size = size
327
328
329 class CNot(ElementaryQuantumGate):
330     '''Controlled not gate'''
331
332     def __init__(self, control = 1, target = 0):
333         if control == target:
334             # (it would impose non-unitary matrix)
335             raise Exception('Control and target qubits cannot be equal')
336         elif control == 1 and target == 0:
337             self.matrix = matrix([
338                 [1, 0, 0, 0],
339                 [0, 1, 0, 0],
340                 [0, 0, 0, 1],
341                 [0, 0, 1, 0]])
342             self.size = 2
343         elif control == 0 and target == 1:
344             self.matrix = matrix([
345                 [1, 0, 0, 0],
346                 [0, 0, 0, 1],
347                 [0, 0, 1, 0],
348                 [0, 1, 0, 0]])
349             self.size = 2
350         else:
351             size = max(control, target) + 1
352             if size == 1:
353                 size = 2
354             dim = 2 ** size
```

```
355             self.matrix = eye(dim)
356             # find correct permutation of identity matrix columns
357             for b in xrange(dim):
358                 bstr = dec2bin(b, size)
359                 if bstr[-(control+1)] == '1':
360                     bstr = list(bstr)
361                     if bstr[-(target+1)] == '0':
362                         bstr[-(target+1)] = '1'
363                     else:
364                         bstr[-(target+1)] = '0'
365                     bstr = ''.join(bstr)
366                     self.matrix[:,b] = eye(dim)[:,int(bstr, 2)]
367             self.size = size
368
369 class Not(ElementaryQuantumGate):
370     '''Not gate'''
371
372     def __init__(self):
373         self.matrix = matrix([
374             [0, 1],
375             [1, 0]])
376         self.size = 1
377
378 class PhaseShift(ElementaryQuantumGate):
379     def __init__(self, angle = pi):
380         self.angle = angle
381         self.matrix = matrix([
382             [1, 0],
383             [0, exp(angle * 1j)]])
384         self.size = 1
385
386
387 class Toffoli(ElementaryQuantumGate):
388     '''Toffoli gate -- Controlled Controlled Not gate'''
389     def __init__(self):
390         self.matrix = matrix([
391             [1, 0, 0, 0, 0, 0, 0, 0],
392             [0, 1, 0, 0, 0, 0, 0, 0],
393             [0, 0, 1, 0, 0, 0, 0, 0],
394             [0, 0, 0, 1, 0, 0, 0, 0],
395             [0, 0, 0, 0, 1, 0, 0, 0],
396             [0, 0, 0, 0, 0, 1, 0, 0],
397             [0, 0, 0, 0, 0, 0, 0, 1],
398             [0, 0, 0, 0, 0, 0, 1, 0]])
399         self.size = 3
400
401
402 class Fredkin(ElementaryQuantumGate):
403     '''Fredkin gate -- Controlled Swap gate'''
404     def __init__(self):
405         self.matrix = matrix([
406             [1, 0, 0, 0, 0, 0, 0, 0],
407             [0, 1, 0, 0, 0, 0, 0, 0],
408             [0, 0, 1, 0, 0, 0, 0, 0],
409             [0, 0, 0, 1, 0, 0, 0, 0],
410             [0, 0, 0, 0, 1, 0, 0, 0],
411             [0, 0, 0, 0, 0, 0, 1, 0],
412             [0, 0, 0, 0, 0, 1, 0, 0],
413             [0, 0, 0, 0, 0, 0, 0, 1]])
414         self.size = 3
415
416
```

```
417 class Swap(ElementaryQuantumGate):
418     '''Qubits order swap gate'''
419     def __init__(self):
420         self.matrix = matrix([
421             [1, 0, 0, 0],
422             [0, 0, 1, 0],
423             [0, 1, 0, 0],
424             [0, 0, 0, 1]])
425         self.size = 2
426 
427 
428 class Arbitrary(ElementaryQuantumGate):
429     '''Quantum gate with arbitrary unitary matrix'''
430     def __init__(self, m):
431         m = matrix(m)
432         if (m.H * m == eye(m.shape[0])).any() == False:
433             pass
434             # raise Exception, 'Not unitary matrix for quantum gate'
435         self.matrix = m
436         self.size = int(math.log(m.shape[0], 2))
437 
438 
439 class WrongSizeException(Exception):
440     def __str__(self):
441         return 'Wrong size of quantum computing object'
442 
443 
444 def dec2bin(dec, length = None):
445     """convert decimal value to binary string"""
446     result = ''
447     if dec < 0:
448         raise ValueError, "Must be a positive integer"
449     if dec == 0:
450         result = '0'
451         if length != None:
452             result = result.rjust(length, '0')
453         return result
454     while dec > 0:
455         result = str(dec % 2) + result
456         dec = dec >> 1
457     if length != None:
458         result = result.rjust(length, '0')
459     return result
460 
461 
462 def Ket(n, size = None):
463     if (n == 0 or n == 1) and size == None:
464         return Qubit(n)
465     ket = QRegister()
466     if size == None:
467         size = int(floor(math.log(n, 2)) + 1)
468     ket.matrix = transpose(matrix([zeros(2 ** size)]))
469     ket.matrix[n] = 1
470     return ket
471 
472 
473 def epr(qreg = Ket(0) ** Ket(0)):
474     """Generate an EPR-pair for |00> input state"""
475     circ = (Hadamard() ** I) * CNot()
476     return circ(qreg)
477 
478 
```

```
ket0 = Ket(0)
ket1 = Ket(1)
s2 = sqrt(2) / 2


h = Hadamard()
I = Identity()
cnot = CNot()
cnot2 = CNot(0, 1)
T = Arbitrary(matrix([
    [1, 0],
    [0, exp(1.0j*pi/4)]]))
```

Bibliografia

[AT04] B. Aoun and M. Tarifi. Quantum artificial intelligence, 2004.

[BB84] Charles Bennett and Gilles Brassard. Quantum cryptography: Public key distribution and coin tossing. In *Proceedings of the IEEE Int. Conf. On Computer Systems and Signal Processing (ICCSSP)*, page 175. Bangalore, India, 1984.

[BBC+93] C Bennett, G Brassard, C Crepeau, R Jozsa, A Peres, and W Wootters. Teleporting an unknown quantum state via dual classical and EPR channels. *Phys Rev Lett*, pages 1895–1899, 1993.

[BBC+95] A. Barenco, C. H. Bennett, R. Cleve, D. P. Divincenzo, N. Margolus, P. Shor, T. Sleator, J. Smolin, and H. Weinfurter. Elementary gates for quantum computation, Mar 1995.

[Bel64] J. S. Bell. On the einstein-podolsky-rosen paradox. *Physics*, 1:195–200, 1964.

[BPM+97] Dik Bouwmeester, J Pan, K Mattle, M Daniell, M Eibl H Weinfurter, and Anton Zeilinger. Experimental quantum teleportation. *Nature*, 390:575–579, 1997.

[Bra96] Gilles Brassard. Teleportation as a quantum computation, May 1996.

[Bra03] Robbert L. Brak. Logics for quantum circuits. Master's thesis, Utrecht University, 2003.

[BW92] Charles H. Bennett and Stephen J. Wiesner. Communication via one- and two-particle operators on einstein-podolsky-rosen states. *Phys. Rev. Lett.*, 69(20):2881–2884, Nov 1992.

[CSB+04] M.B.J. Chiaverini, T. Schaetz, J. Britton, W. Itano, J. Jost, E. Knill, C. Langer, D. Leibfried, R. Ozeri, and D. Wineland.

Deterministic quantum teleportation of atomic qubits. *Nature*, 429, 2004.

[Deu85] David Deutsch. Quantum theory, the Church-Turing principle and the universal quantum computer. *Proceedings of the Royal Society of London Ser. A*, A400:97–117, 1985.

[EPR35] A. Einstein, B. Podolsky, and N. Rosen. Can quantum-mechanical description of physical reality be considered complete? *Phys. Rev.*, 47(10):777–780, May 1935.

[Fey82] Richard Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6&7):467–488, 1982.

[Fey96] Richard P. Feynman. *Wykłady o obliczeniach*. Prószyński i S-ka, Warszawa, 1996.

[Fit04] Damien Fitzgerald. Quantum qudit simulation. Master's thesis, National University of Ireland, Galway, 2004. M.Sc. in Software Design and Development, Supervisor: Dr. Michael McGettrick.

[FTG03] J. Faber, R.N. Thess, and G. Giraldi. Learning linear operators by genetic algorithms, 2003.

[GK03] Krzysztof Giaro and Marcin Kamiński. *Wprowadzenie do algorytmów kwantowych*. Exit, Warszawa, 2003.

[GPT04] Gilson A. Giraldi, Renato Portugal, and Ricardo N. Thess. Genetic algorithms and quantum computation, Mar 2004.

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *STOC '96: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, New York, NY, USA, 1996. ACM Press.

[Ham06] S. Hameroff. Consciousness, neurobiology and quantum mechanics: The case for a connection. *The Emerging Physics of Consciousness*, 2006.

[HGS+05] Jianjun Hu, Erik Goodman, Kisung Seo, Zhun Fan, and Rondal Rosenberg. The hierarchical fair competition (hfc) framework for sustainable evolutionary algorithms. *Evol. Comput.*, 13(2):241–277, 2005.

[HGSP02] Jianjun Hu, Erik D. Goodman, Kisung Seo, and Min Pei. Adaptive hierarchical fair competition (ahfc) model for parallel evolutionary algorithms. In *GECCO '02: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 772–779, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[Hir04] Mika Hirvensalo. *Algorytmy kwantowe*. WSiP, Warszawa, 2004.

[HK00] Kuk-Hyun Han and Jong-Hwan Kim. Genetic quantum algorithm and its application to combinatorial optimization problem. In *Proc. of the 2000 Congress on Evolutionary Computation*, pages 1354–1360, Piscataway, NJ, 2000. IEEE Service Center.

[Hol75] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1975.

[HPLK01] Kuk-Hyun Han, Kui-Hong Park, Chi-Ho Lee, and Jong-Hwan Kim. Parallel quantum-inspired genetic algorithm for combinatorial optimization problem. In *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, pages 1422–1429, COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea, 27-30 2001. IEEE Press.

[Kli04] Gleb V. Klimovitch. How quantum entanglement helps to coordinate non-communicating players, 2004.

[Lib87] Richard L. Liboff. *Wstęp do mechaniki kwantowej*. PWN, Warszawa, 1987.

[Mez06] Francesco Mezzadri. How to generate random matrices from the classical compact groups, Sep 2006.

[NC00] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, October 2000.

[NKL98] Nielsen, Knill, and Laflamme. Complete quantum teleportation using nuclear magnetic resonance. *NATURE: Nature*, 396, 1998.

[Pen89] Roger. Penrose. *The emperor's new mind*. Oxford, 1989.

[RHR+04] M. Riebe, H. Haeffner, CF Roos, W. Haensel, J. Benhelm, GPT Lancaster, TW Koerber, C. Becher, F. Schmidt-Kaler, and DFV James. Deterministic quantum teleportation with atoms. *Nature*, 429(6993):734–737, 2004.

[RSFAF01] Bart Rylander, Terry Soule, James Foster, and Jim Alves-Foss. Quantum evolutionary programming. In Lee Spector, Erik D. Goodman, Annie Wu, W. B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 1005–1011, San Francisco, California, USA, 7-11 July 2001. Morgan Kaufmann.

[Rub00] Ben I. P. Rubinstein. Evolving quantum circuits using genetic programming. In John R. Koza, editor, *Genetic Algorithms and Genetic Programming at Stanford 2000*, pages 325–334. Stanford Bookstore, Stanford, California, 94305-3079 USA, 2000.

[SBBS99a] Lee Spector, Howard Barnum, Herbert J. Bernstein, and Nikhil Swami. Finding a better-than-classical quantum AND/OR algorithm using genetic programming. In Peter J. Angeline, Zbyszek Michalewicz, Marc Schoenauer, Xin Yao, and Ali Zalzala, editors, *Proceedings of the Congress on Evolutionary Computation*, volume 3, pages 2239–2246, Mayflower Hotel, Washington D.C., USA, 6-9 1999. IEEE Press.

[SBBS99b] Lee Spector, Howard Barnum, Herbert J. Bernstein, and Nikhil Swamy. Quantum computing applications of genetic programming. pages 135–160, 1999.

[Sga07] Kyriakos N. Sgarbas. The road to quantum artificial intelligence, 2007.

[SGM05] Wissam A. Samad, Roy Ghandour, and Mohamad. Memory efficient quantum circuit simulator based on linked list architecture, Nov 2005.

[Sho94] Peter W. Shor. Algorithms for quantum computation: Discrete log and factoring. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 124–134. Institute of Electrical and Electronic Engineers Computer Society Press, 1994.

[Sum05] Johann Summhammer. Quantum cooperation of insects, 2005.

[Sum06] Johann Summhammer. Quantum cooperation of two insects, 2006.

[TV06] Neil Toronto and Dan Ventura. Learning quantum operators from quantum state pairs. In *IEEE World Congress on Computational Intelligence*, pages 2607–2612, July 2006.

[Ven00] Dan Ventura. Learning quantum operators. In *Proceedings of the Joint Conference on Information Sciences*, pages 750–752, March 2000.

[VYC00] Lieven M.K. Vandersypen, Costantino S. Yannoni, and Isaac L. Chuang. Liquid state nmr quantum computing, 2000.

[WG04] C. Williams and G. Gray. Automated design of quantum circuits. 2004.

[Yao93] A. Yao. Quantum circuit complexity. In *Proceedings of the 34th Annual Symposium on Foundations of Computer Science*, pages 352–361, Los Alamitos, CA, 1993. Institute of Electrical and Electronic Engineers Computer Society Press.

[YI00] Taro Yabuki and Hitoshi Iba. Genetic algorithms for quantum circuit design - evolving a simpler teleportation circuit. In Darrell Whitley, editor, *Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference*, pages 425–430, Las Vegas, Nevada, USA, 8 2000.

Spis rysunków

Spis tabel