VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

EVOLUČNÍ NÁVRH KVANTOVÉHO OPERÁTORU

EVOLUTIONARY DESIGN OF QUANTUM OPERATOR

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **PAVEL KRAUS**
AUTHOR

VEDOUCÍ PRÁCE **Ing. MICHAL BIDLO, Ph.D.**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačových systémů (UPSY) Akademický rok 2021/2022

Zadání bakalářské práce

Student: **Kraus Pavel**
Program: Informační technologie
Název: **Evoluční návrh kvantového operátoru**
Evolutionary Design of Quantum Operator
Kategorie: Umělá inteligence

Zadání:

1. Nastudujte základy evolučních algoritmů a principů kvantového počítání.
2. Zvolte vhodný operátor z oblasti kvantových výpočtů a realizujte jeho řešení ve zvoleném simulátoru kvantového počítače.
3. Pro kvantový operátor navrhněte vhodnou reprezentaci a implementujte vybraný evoluční algoritmus pro jeho automatický návrh.
4. Proveďte sadu experimentů využívajících různá nastavení parametrů evoluce.
5. Sestavte srovnávací studii demonstrující schopnosti navrženého algoritmu na zvolené úloze a s ohledem na známou (existující) realizaci zvoleného operátoru.
6. Zhodnoťte dosažené výsledky a diskutujte možnosti pokračování projektu.

Literatura:

- Dle pokynů vedoucího projektu.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Bidlo Michal, Ing., Ph.D.**
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 29. října 2021

Abstrakt

Cílem této práce bylo využít pro návrh kvantových operátorů v podobě unitárních matic v přímé reprezentaci různé evoluční algoritmy. Byly zvoleny algoritmy evoluční strategie, diferenciální evoluce, optimalizace hejnem částic a optimalizace umělým včelstvem. Třetí a čtvrtý zmíněný algoritmus byl pro návrh kvantových operátorů použit v této práci poprvé. Na experimentech bylo ukázáno, že použití přímé reprezentace dosahuje výsledků přijatelné kvality.

Abstract

The goal of this thesis is to utilize various evolutionary algorithms for quantum operator design in the form of unitary matrices in direct representation. Evolution strategy, differential evolution, Particle Swarm Optimization and artificial bee colony algorithms were chosen. In this thesis, the third and fourth algorithms were used for the first time in relation to quantum operator design. The experiments have shown that the utilization of direct representation gives results of acceptable quality.

Klíčová slova

kvantové počítání, kvantový operátor, evoluční algoritmus, evoluční strategie, diferenciální evoluce, diferenciální evoluce, optimalizace hejnem částic, optimalizace umělým včelstvem

Keywords

quantum computing, quantum operator, evolutionary algorithm, evolution strategy, differential evolution, particle swarm optimization, artifcial bee colony

Citace

KRAUS, Pavel. *Evoluční návrh kvantového operátoru*. Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Michal Bidlo, Ph.D.

Evoluční návrh kvantového operátoru

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Bidla Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Pavel Kraus
18. května 2022

Poděkování

Velké poděkování patříme mému vedoucímu Ing. Michalu Bidlovi Ph.D. za cenné rady, doporučení kterými směry se vydat a zároveň upozorňování na možné slepé uličky. Tato práce byla podpořena Ministerstvem školství, mládeže a tělovýchovy České republiky prostřednictvím e-INFRA CZ (ID:90140).

Obsah

Kapitola 1

Úvod

Principy kvantové mechaniky vnášejí do oblasti provádění výpočtů nový vítr. Nové způsoby se nám nabízí již od počátku tohoto oboru. Principy superpozice a kvantového provázání mohou tvořit základ velmi výkonného výpočetního aparátu. [13]

Jeden z prvních vědců, který se využití kvantových jevů pro výpočty zabýval, byl Paul Beniof. Ten ve své práci z roku 1979 popsal teoretický základ kvantové počítání a navrhl, že by takový počítat mohl být zkonstruován. [1, 13] Za hlavního průkopníka je ovšem považován Richard Feynman, který v roce 1981 popsal, že klasický výpočetní systém nemůže adekvátně reprezentovat jevy kvantové mechaniky a navrhl, že by bylo výhodné tyto jevy v počítačích využít. [13, 10]

Jako příklad demonstrace velkého významu kvantového počítání nám může sloužit algoritmus navržený vědcem Peterem Shorem v roce 1994 určený pro kvantový počítač, který provádí faktorizaci velkých čísel velice rychle. Faktorizace je na klasických počítačích obecně těžko vyřešitelný problém s exponeciální složitostí. [10] Na této skutečnosti zakládá princip asymetrické kryptografie implementovaný v algoritmu RSA. [19] Pokud bychom byli schopni faktorizovat velká čísla v rozumném čase, přestává dávat smysl proužívat algoritmy založené na tomto principu pro šifrování dat.

Z dosavadního popisu kvantových jevů se může zdát, že máme k dispozici aparát, který současné klasické počítače strčí do kapsy, kvantové počítače mají ovšem spoustu problému, se kterými se potýkáme.

Současné konstrukce kvantových počítačů jsou ovšem pouze prototypy, které jsou předmětem výzkumu a vypadají spíše jako ENIAC (jeden z historicky prvních počítačů) než jako notebook nebo tablet. Často zabírají prostory celé laboratoře a pro jejich provoz je potřeba spoustu dalších nástrojů a zařízení, které se starají například o odstnínění od elektromagnetického šumu, mechanických vibrací, tepla (kvantové čipy často operují v teplotách na úrovni mili Kelvinů) a dalších vnějších zdrojů, které způsobují zhoršení výkonu. [2]

Cílem této práce je navrhnout systém, který bude pomocí evolučních algoritmů hledat kvantové operátory. Práce navazuje na předchozí publikace Ing. Petra Žufana a Ing. Michala Bidla, Ph.D., které zkoumaly návrh kvantových operátorů s použitím různých komplexnějších reprezentací. [24, 22] Tato práce se bude snažit zjistit, zda použití přímé reprezentace kandidátních řešení povede k návrhu kvantových operátorů přijatelné kvality. V navrženém systému jsou implementovány čtyři algoritmy: Evoluční strategie, Diferenciální evoluce, Optimalizace hejnem částic a Optimalizace umělým včelstvem. První dva již byly použity ve zmíněných publikacích, u nich se budu zabývat nastavením parametrů, kterými se publikace nezabývaly. Poslední dva algoritmy jsou v této práci použity pro návrh kvantových

operátorů poprvé. Algoritmy budou s použitím různých nastavení otestovány na návrhu dvou kvantových operátorů: MAX3 a CNOT.

Druhá kapitola se věnuje základním principům kvantového počítání a kvantových algoritmů. Ve třetí kapitole jsou dopodrobna popsány principy použitných evolučních algoritmů. Čtvrtá kapitola detailně popisuje použitou reprezentaci, nastavení algoritmů a způsob vyhodnocení kandidátních řešení. V páte kapitole jsou popsány jednotlivé experimenty a zhodnoceny jejich výsledky. Poslední kapitola pak shrnuje čeho bylo v práci dosaženo a nabízí možnosti, kterými se vydat při dalším bádání.

Kapitola 2

Kvantové počítání

2.1 Základní principy

2.1.1 Qubit

Qubit je kvantový bit. Qubit je podobný klasickému bitu v tom, že může nabývat stavů 0 a 1. Od klasického qubitu se ale liší tím, že může nabývat spojitých hodnot představující superpozici stavů. [12]

Klasický bit můžeme fyzicky reprezentovat například elektrickým napětím. Pro reprezentaci qubitu musíme požít jiné fyzikální jevy. Jedním takovým jevem může být spin elektronu. [17] Dále budeme ovšem přistupovat ke qubitu jako k abstraktní matematické entitě.

Pro popis kvantových stavů se zpravidla používá **Bra-ket** notace (též Dirackova). [17] Pro potřeby této práce není potřeba znát celý matematický aparát, který vedl k zavedení této notace. Stačí nám vědět, že jde o způsob, jak označit vektor stavu qubitu. Typické je použití "|>" znaků. Například $|a\rangle = (\dots)$ je ket popisující stav qubitu.

První dva stavy qubitu (společné s bitem) zobrazíme s využitím bra-ket notace následovně pomocí vektorů [12]:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{2.1}$$

Tyto dva vektory jsou známé jako **standardní výpočetní báze**. Tvoří ortonormální bázi tohoto vektorového prostoru. Analogicky ke klasickému bitu, vektor $|0\rangle$ vyjadřuje logickou 0 a vektor $|1\rangle$ vyjadřuje logickou 1. [17]

Qubity ale mohou nabývat nekonečně mnoho jiných stavů. Ostatní stavy můžeme popsat jako lineární kombinaci standardních výpočetních bází, téže známá jako **superpozice**:[17]

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{2.2}$$

kde α a β jsou komplexní čísla.

2.1.2 Superpozice a měření

Pro shrnutí superpozice a měření v této podsekci bylo vycházeno z [17].

Bit můžeme prozkoumat a jednoznačně určit, zda je ve stavu 0 nebo 1. Klasické počítače toto dělají neustále, například při čtení dat z paměti. U qubitu nemůžeme jednoznačně určit

v jakém stavu se nachází. Kvantové mechanika nám říká, že můžeme o qubitu získat pouze velice omezené informace.

Pokud změříme qubit, dostaneme vždy buď hodnotu 0 nebo 1. Nikdy dopředu nevíme, jakou hodnotu změříme. Supermozice nám ale popisuje, s jakou pravděpodobností naměříme hodnotu 0 nebo 1. Z definice superpozice 2.2 $|\alpha|^2$ značí, jaká je pravděpodobnost, že po změření qubitu dostaneme hodnotu 0. Analogicky $|\beta|^2$ pro hodnotu 1. Součet pravděpodobností musí být 1, musí tedy platit:

$$|\alpha|^2 + |\beta|^2 = 1 \tag{2.3}$$

Geometricky to můžeme interpretovat tak, že stav qubitu musí být normalizován na délku 1. Obecně tedy můžeme říct, že **stav qubitu je jednotkový vektor ve dvourozměrném komplexním vektorovém prostoru**.

Například, qubit může být ve stavu:

$$|\psi\rangle = -\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle \tag{2.4}$$

což můžeme napsat také jako:

$$|\psi\rangle = \begin{pmatrix} \frac{-1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} \tag{2.5}$$

Pokud budeme měřít stav tohoto qubitu, s pravděpodobností 50% ($|-\frac{1}{\sqrt{2}}|^2 = 0.5$) naměříme hodnotu 0. Analogicky hodnotu 1 s pravděpodobností 50%.

2.1.3 Systémy s více qubity

V předchozích kapitolách jsme se věnovali reprezentaci jednoho qubitu. Kvantové systémy se však často skládají z mnoha qubitů. Složitější stavy jsou popsány vektorem o velikosti 2^n, kde n udává počet qubitů, ze kterých se systém skládá. Pro kombinaci několika jednotlivých qubitů do složitějšího systému se používá **tenzorový součin** ($\otimes$). [23]

Mějme vektory $|a\rangle$ a $|b\rangle$, každý reprezentující stav jednoho qubitu. Tenzorový součin těchto dvou vektorů bude potom vypdat takto [6]:

$$|a\rangle \otimes |b\rangle = \begin{pmatrix} \alpha_0 \\ \alpha_1 \end{pmatrix} \otimes \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix} = \begin{pmatrix} \alpha_0 \cdot \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix} \\ \alpha_1 \cdot \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix} \end{pmatrix} = \begin{pmatrix} \alpha_0\beta_0 \\ \alpha_0\beta_1 \\ \alpha_1\beta_0 \\ \alpha_1\beta_1 \end{pmatrix} \tag{2.6}$$

Stav systému kombinující qubity $|a\rangle$ a $|b\rangle$ můžeme zkráceně zapsat jako $|ab\rangle$.

Tenzorový součin není komutativní operace: [23]

$$|0\rangle \otimes |1\rangle = |0,1\rangle = |01\rangle \neq |10\rangle = |1,0\rangle = |1\rangle \otimes |0\rangle \tag{2.7}$$

První ket říká, že první qubit je ve stavu 0 a druhé ve stavu 1. Druhý ket říká, že první qubit je ve stavu 1 druhý ve stavu 0.

Kombinací dvou qubitů dostáváme vektorový prostor, který má 4 standardní výpočetní báze: [12]

$$|00\rangle = \begin{pmatrix}1\\0\\0\\0\end{pmatrix} \quad |01\rangle = \begin{pmatrix}0\\1\\0\\0\end{pmatrix} \quad |10\rangle = \begin{pmatrix}0\\0\\1\\0\end{pmatrix} \quad |11\rangle = \begin{pmatrix}0\\0\\0\\1\end{pmatrix} \tag{2.8}$$

Obdobným způsobem můžeme vyjádřit např. stav systému, který pracuje s kvantovým bytem (tedy systém, který obsahuje 8 qubitů): [23]

$$|\psi\rangle = |q_0\rangle \otimes |q_1\rangle \otimes |q_2\rangle \otimes |q_3\rangle \otimes |q_4\rangle \otimes |q_5\rangle \otimes |q_6\rangle \otimes |q_7\rangle \tag{2.9}$$

Výsledný stav bude vektor, který je podmnožinou vektorového prostoru: [23]

$$\mathbb{C}^2 \otimes \mathbb{C}^2 \otimes \mathbb{C}^2 \otimes \mathbb{C}^2 \otimes \mathbb{C}^2 \otimes \mathbb{C}^2 \otimes \mathbb{C}^2 \otimes \mathbb{C}^2 \tag{2.10}$$

Tento vektorový prostor lze zapsat jako: $(\mathbb{C}^2)^{\otimes 8}$. Jedná se o komplexní vektorový prostor o rozměru $2^8 = 256$. Jelikož je jen jeden vektorový prostor tohoto rozměru, je tento prostor izomorfní k $\mathbb{C}^{256}$. [23]

Výsledný vektor potom bude vypadat takto:

$$|\psi\rangle = \begin{pmatrix}c_0\\c_1\\\vdots\\c_{254}\\c_{255}\end{pmatrix} \tag{2.11}$$

kde $|c_0|^2$ udává pravděpodobnost, že po změření stavu dostaneme výsledek 00000000, $|c_1|^2$ pravděpodobnost, že dostaneme výsledek 00000001, $|c_{255}|$ pravděpodobnost, že dostaneme výsledek 11111111 apod. [23]

Jelikož jednotlivá čísla udávají pravděpodobnost, musí i zde platit: [23]

$$\sum_{n=0}^{255} |c_n|^2 = 1 \tag{2.12}$$

Ve světě klasických počítačů nám stačí pro zápis stavu 1 bytu pouze 8 hodnot - 8 bitů. V kvantovém světě je potřeba pro zápis stavu 8 qubitů celkem 256 komplexních čísel. Tento exponenciální růst byl jedním z důvodů, proč vědci začali přemýšlet nad využitím kvantových jevů pro výpočty. Pokud bychom chtěli emulovat kvantový počítač na klasickém počítači s regisrem 64 qubitů, museli bychom být schopni ukládat $2^{64} = 18\,446\,744\,073\,709\,551\,616$ komplexních čísel. Toho v současné chvíli nejsme schopni. [23]

Na závěr této sekce si ukážeme, jak lze zapsat stav více qubitů pomocí standardních výpočetních bází a ket notace. Mějme stav dvou qubitů popsaný vektorem:

$$\frac{1}{\sqrt{3}}\begin{pmatrix}1\\0\\-1\\1\end{pmatrix} = \begin{pmatrix}\frac{1}{\sqrt{3}}\\0\\\frac{-1}{\sqrt{3}}\\\frac{1}{\sqrt{3}}\end{pmatrix} \tag{2.13}$$

můžeme jej zapsat jako:

$$\frac{1}{\sqrt{3}}|00\rangle - \frac{1}{\sqrt{3}}|10\rangle + \frac{1}{\sqrt{3}}|11\rangle = \frac{|00\rangle - |10\rangle + |11\rangle}{\sqrt{3}} \tag{2.14}$$

2.2 Reprezentace kvantových operací

V předchozí kapitole jsme si popsali, jak můžeme reprezentovat stavy jednotlivých qubitů. Kvantové systémy ovšemn jsou statické a jen jejich jejich popis nám pro kvantové výpočty nestačí. Stavy systému se v průběhu času mění. [23]

2.2.1 Obecný kvantový operátor a jeho aplikace na stav

Změnu stavu kvantového systému v čase popisujeme kvantovým operátorem. Operátor reprezentujeme pomocí **unitární matice**. Pro n rozměrný vektor stavu bude mít matice velikost $n \times n$. [23]

Pro následující popis operací s unitární maticí bylo vycházeno z [6].

Pro unitární matici U musí platit:

$$UU^\dagger = U^\dagger U = I \tag{2.15}$$

kde $U^\dagger$ je hermitovsky sdružená matice U a I je jednotková matice.

Aplikaci operátoru na stav provedeme jednoduše tak, že vektor stavu vynásobíme maticí:

$$U\left|\psi\right\rangle = \left|\psi'\right\rangle \tag{2.16}$$

kde $\left|\psi\right\rangle$ je stav systému v čase t a $\left|\psi'\right\rangle$ je stav systému v času $t+1$.

Díky vlastnostem unitární matice je operátor reverzibilní - jednoduše se můžeme vrátit k původnímu stavu před aplikací operátoru. Většinu operací provedených v kvantovém světe (kromě například měření) můžeme vrátit zpět. Jednoduše tímto způsobem:

$$\left|\psi\right\rangle = U^\dagger\left|\psi'\right\rangle \tag{2.17}$$

Aplikaci oprátoru na stav vícekrát za sebou můžeme provést tímto způsobem:

$$U^n\left|\psi\right\rangle = \left|\psi''\right\rangle \tag{2.18}$$

kde $\left|\psi\right\rangle$ je stav systému v čase t a $\left|\psi''\right\rangle$ je stav systému v čase $t+n$.

Obdobně jako můžeme qubity slučovat do složitějších systémů (viz 2.1.3), můžeme slučovat i operátory. Pokud U operuje nad prostorem $\mathbb{C}^n$ a U' operuje nad prostorem $\mathbb{C}^m$, potom $U \otimes U'$ bude operovat nad $\mathbb{C}^n \otimes \mathbb{C}^m$ následujícím způsobem:

$$(U \otimes U')(\left|\psi\right\rangle \otimes \left|\psi'\right\rangle) = U\left|\psi\right\rangle \otimes U'\left|\psi'\right\rangle \tag{2.19}$$

Příklad aplikace operátoru na stav:
mějme unitární matici U (která odpovídá Hadamardovu hradlu):

$$U = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{2.20}$$

kterou aplikuje na $\left|0\right\rangle$:

$$U\left|0\right\rangle = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}\begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} \tag{2.21}$$

2.2.2 Klasická hradla

Pro popis klasických hradel bylo v této subsekci vycházeno z [6].

Klasická logická hradla jsou určena pro manipulaci s bity. Tak jako můžeme pomocí vektoru stavu qubitu reprezentovat klasické bity, můžeme pomocí matice reprezentovat klasické logické operace. Pokud má hradlo na vstupu n bitů a na výstupu m bitů, bude mít matice reprezentující logickou operaci velikost $2^m \times 2^n$.

Nejjednodušším příkladem je hradno NOT. Na vstupu hradla je jeden bit a na výstupu je taky jeden bit, hradlo tedy bude reprezentované maticí 2×2:

$$NOT = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{2.22}$$

Aplikací NOT na $|0\rangle$ dostaneme $|1\rangle$ a naopak aplikací na $|1\rangle$ dostaneme $|0\rangle$.

Další známá hradla jsou reprezentována následujícími maticemi:

$$AND = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad NAND = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \tag{2.23}$$

$$OR = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \end{pmatrix} \quad NOR = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix} \tag{2.24}$$

Jelikož tato hradla mají 2 bity na vstupu a jedn na výstupu, jsou reprezentovány maticemi 2×4. Tato hradla nejsou reversibilní. (Jelikož mají hradla na výstupu jeden bit, nelze vždy z výstupu určit, jaké hodoty byly na dvou vstupech.)

Aplikace může vypadat následovně:

$$\begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{2.25}$$

můžeme také zapsat jako:

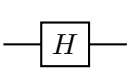
$$AND\,|11\rangle = |1\rangle \tag{2.26}$$

Klasická hradla můžeme aplikovat pouze na standardní výpočetní báze.

2.2.3 Kvantová hradla

Kvantové hradlo je jednoduše operátor, který operuje s qubity. Takový operátor je reprezentován unitární maticí. [6] V této sekci si uvedeme několik příkladů kvantových hradel:

Hadamardovo hradlo [1] Tohle hradlo je jedno z nejvýznamějších v kvantovém počítání. Dovoluje nám vzít standardní výpočetní bázi a převést ji do stavu superpozice dvou stavů. Diagram vypadá následovně: [12]



[1] Tohle hradlo sice navrhl matematik John Sylvester, bylo ale pojmenováné po matematikovy jménem Jacques Hadamard [12]

a pomocí matice:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{2.27}$$

Příklad aplikace Hadamardova hradla na $|0\rangle$:

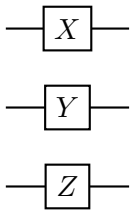
$$H|0\rangle = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1+0 \\ 1+0 \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \tag{2.28}$$

Obdobně aplikací na $|1\rangle$ dostaneme:

$$H|1\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}} \tag{2.29}$$

Hadamardovo hradlo tedy vezme standardní výpočetní báze a udělá její projekci do stavu superpozice $(|0\rangle + |1\rangle)/\sqrt{2}$ nebo $(|0\rangle - |1\rangle)/\sqrt{2}$.

Pauli X, Y, Z Tato hradla patří mezi nejužitečnější v kvantových obvodech. [17] Slouží pro rotaci stavu okolo os x, y a z. (Osy v reprezantaci bitu pomocí Blochovy sféry). Zobrazit je v diagramu můžeme takto: [12]

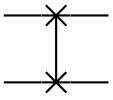


Pomocí matic:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{2.30}$$

Všimněme si, že X je stejné jako NOT.

SWAP Funkce tohoto operátoru je jednoduchá - prohazuje stavy dvou qubitů mezi sebou. Značení v diagramu: [17]

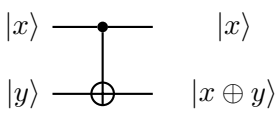


a pomocí matice:

$$SWAP = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.31}$$

Swap je také možné realizovat pomocí aplikace hradla CNOT na stav 3x za sebou. [17]

Controlled-NOT Též známé jako CNOT nebo CX. [6]



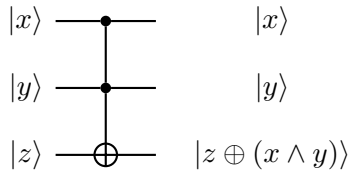
Toto hradlo má 2 qubity na vstupu a 2 na výstupu. Horní vstup ($|x\rangle$) je kontrolní bit. Ovládá, co bude na výstupu. Pokud $|x\rangle = |0\rangle$, potom na spodním výstupu bude to stejné co na vstupu ($|y\rangle$). Pokud $|x\rangle = |1\rangle$, potom na výstupu bude opak. Pokud napíšeme horní qubit jako první a spodní jako druhý, CNOT udělá z $|x, y\rangle$ vektor $|x, x \oplus y\rangle$, kde $\oplus$ značí binárně exklusivní operaci OR. [6]

Matice reprezentující toto hradlo vypadá následovně:

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{2.32}$$

Toto hradlo může být revertováno samo sebou. Jak bylo zmíněno, po aplikaci tohoto hradla se stav $|x, y\rangle$ změní na stav $|x, x \oplus y\rangle$. Po opětovné aplikace hradla na tento stav dostaneme $|x, x \oplus (x \oplus y)\rangle$. Jelikož je $\oplus$ asociativní, můžeme výsledek upravit na $|x, (x \oplus x) \oplus y\rangle$. Protože $x \oplus x$ je vždy rovno 0, můžeme výsledný stav dále upravit na $|x, y\rangle$, čímž získáváme původní stav. [6]

Toffoliho hradlo Též známe jako CCNOT nebo CCX [6]



Toto hradlo je podobné CNOT, má ale dva kontrolní bity místo jednoho. Pokud jsou oba dva horní bity ($|x\rangle$ a $|y\rangle$) ve stavu $|1\rangle$. Můžeme také říct, že CCNOT vezme stav $|x, y, z\rangle$ a transformuje jej do stavu $|y, y, z \oplus (x \wedge y)\rangle$ [6]

Matice reprezentující toto hradlo vypadá následovně:

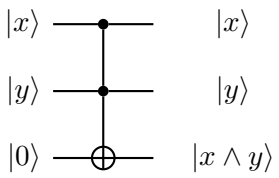
$$CCNOT = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \tag{2.33}$$

Klasické hradlo NOT nemá žádný kontrolní bit, CNOT má jeden kontrolní bit a CCNOT 2. Pokud bychom chtěli obdobně použít více kontrolních bitů, můžeme toho jednoduše

dosáhnout pomocí více CCNOT hradel. Například pro 3 kontrolní bity bychom potřebovali 3 hradla CCNOT. [6]

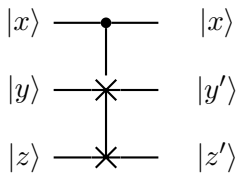
Dalším důvodem, proč je CCNOT zajímavé, je jeho univerzálnost. Jinými slovy, s pomocí hradel CCNOT jsme schopni vytvořit jakékoliv logické hradlo. Z tohov vyplývá, že za pomoci pouze tohoto hradla bychom byli schopni zkonstruovat reverzibilní počítač. Podle teorie by takový počítač nespotřebovával žádnou energii a ani by neprodukoval žádné teplo. [6]

Jako ukázka univerzálnosti nám poslouží příklad, jak pomocí tohoto hradla vytvořit hradlo AND: [6]



Jediná úprava oproti obecnému CCNOT je ta, že spodní vstupní bit má vždy hodnotu $|0\rangle$.

Fredkinovo hradlo



Toto hradlo funguje podobně jako $SWAP$, navíc ale disponuje kontrolním bitem, který řídí, zda bude probíhat přehození stavu nebo se na výstupu nebude nic měnit. Pokud je $|x\rangle$ ve stavu $|0\rangle$, potom $|y\rangle = |y\rangle$ a $|z\rangle = |z'\rangle$. Pokud je $|x\rangle$ ve stavu $|1\rangle$, potom $|y'\rangle = |z\rangle$ a $|z\rangle = |y\rangle$. [6]

Matice tohoto hradla:

$$CSWAP = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.34}$$

Fredkinovo hradlo je také jako Toffoliho univerzální. [6]

2.3 Kvantové algoritmy

Algoritmy bývají často navrženy dlouho před tím, než jsme schopni zkonstruovat stroje, na kterých bychom je mohli spustit. Není tomu jinak ani u kvantových algoritmů, pro které se stále nemáme stroje, které by je byli schopné vykonávat. [6]

V předchozích sekcích jsme si definovali, jak můžeme popisovat stavy qubitů a jak s nimi manipulovat pomocí unitárních matic. Nyní můžeme s využitím těchto znalostí začít tvořit kvanové algoritmy.

Kvantové algoritmy následují tento základní rámec: [6]

1. Systém začíná s qubity, které jsou ve jednom z klasických stavů (tedy $|0\rangle$ nebo $|1\rangle$)
2. Qubity v systému jsou uvedeny do superpozice několika stavů (například pomocí Hadamardova hradla)
3. Následuje aplikace operátorů qubitů, které mění jeho stav, pořád v rámci superpozice.
4. Nakonec proběhne měření qubitů.

Jistě bychom našli různé modifikace tohoto rámce, nicméně pro jednoduchost nám toto zobecnění postačí.

V následujích sekcích budou představeny dva algoritmy, na kterých si ukážeme, jakým způsobem může být kvantové počítání efektivnější než to klasické.

2.3.1 Deutschův algoritmus

Myšlenky tohoto algoritmu byly původně publikovány v [7]. Pro shrnutí algoritmu v této podsekci bylo vycházeno z [6, 17].

Jedná se o velice jednoduchý algoritmus, který řeší spíše vykonstruovaný problém. Důvodem, proč tento algoritmus exsituje, je na jednoduchém příkladu ukázat, že kvantové výpočty mohou být efektivnější než klasické.

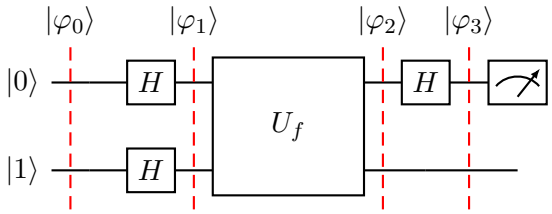
Mějme funkci $f(x)$, jejíž definiční obor i obor hodnot je množina $\{0,1\}$ ($f : \{0,1\} \to \{0,1\}$). Takové funkce existují čtyři:

1. $f(0) = 0, f(1) = 0$
2. $f(0) = 0, f(1) = 1$
3. $f(0) = 1, f(1) = 0$
4. $f(0) = 1, f(1) = 1$

O funkci můžeme říct, že je vyvážená, pokud $f(0) \neq f(1)$ (2. a 3. varianta) nebo konstantní, pokud $f(0) = f(1)$ (1. a 4. varianta). Deutschův algoritmus řeší následující problém: Máme zadanou funkci $f : \{0,1\} \to \{0,1\}$ jako black box. Může funkci vyhodnotit, ale nemůžeme se "podívat dovnitř"a zjistit, jak je definovaná (resp. o jakou ze čtyř výše uvedených variant se jedná). Naším úkolem je zjistit, jestli je funkce vyvážená nebo konstantní.

S použitím klasického počítače bychom nejdřív vyhodnotili funkci pro oba možné vstupy, $f(0)$ a $f(1)$ a následně výstupy porovnali. V takovém případě bychom funkci f museli vyhodnotit dvakrát. Superpočítač může být v superpozici dvou základních stavů ve stejném čase. Využijeme superpozice stavů k vyhodnocení obou výstupů najednou.

Diagram algoritmu vypadá následovně:



Algoritmus můžeme též zapsat v podobě operací s maticemi a vektory následovně:

$$(H \otimes I)U_f(H \otimes H)\,|0,1\rangle \tag{2.35}$$

Aplikace hradla U_f na stav značí vyhodnocení funkce f. Hradlo má podobu následující unitární matice:

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.36}$$

$|\phi_i\rangle$ nad diagramem budeme používat pro popis stavu v každém časovém okamžiku výpočtu:

$|\phi_0\rangle$ Začátek algoritmu, qubity jsou zatím v jednom z klasických stavů, tedy $|0,1\rangle$.

$|\phi_1\rangle$ Aplikace Hadamardova hradla na počáteční stavy.

$$|\phi_1\rangle = \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right)\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) = \frac{+\,|0,0\rangle - |0,1\rangle + |1,0\rangle - |1,1\rangle}{2} = \begin{pmatrix} +\frac{1}{2} \\ -\frac{1}{2} \\ +\frac{1}{2} \\ -\frac{1}{2} \end{pmatrix} \tag{2.37}$$

$|\phi_2\rangle$ Aplikace hradla U_f:

$$|\phi_2\rangle = \left(\frac{(-1)^{f(0)}\,|0\rangle + (-1)^{f(1)}\,|1\rangle}{\sqrt{2}}\right)\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) \tag{2.38}$$

podívejme se zblízka na část, kde se odehrává vyhodnocení funkce f:

$$(-1)^{f(0)}\,|0\rangle + (-1)^{f(1)}\,|1\rangle \tag{2.39}$$

Pokud je f konstatnní, bude tahle část vypadat následovně: (záleží, jestli je konstatně 1 nebo 0)

$$+1(|0\rangle + |1\rangle) \text{ nebo } -1(|0\rangle + |1\rangle) \tag{2.40}$$

A pokud je f vyvážená:

$$+1(|0\rangle - |1\rangle) \text{ nebo } -1(|0\rangle - |1\rangle) \tag{2.41}$$

Po úpravě tedy dostaneme, že:

$$|\phi_2\rangle_1 = (\pm 1)\left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right)\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) \quad \text{pokud je } f \text{ konstantí} \tag{2.42}$$

$$|\phi_2\rangle_2 = (\pm 1)\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right)\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) \quad \text{pokud je } f \text{ vyvážená} \tag{2.43}$$

$|\phi_3\rangle$ Opětovná aplikace Hadamardova hradla:

$$|\phi_2\rangle_1 = (\pm 1)\,|0\rangle \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) \quad \text{pokud je } f \text{ konstantí} \tag{2.44}$$

$$|\phi_2\rangle_2 = (\pm 1)\,|1\rangle \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) \quad \text{pokud je } f \text{ vyvážená} \tag{2.45}$$

Teď už nám zbývá jen změřit horní bajt. Pokud je ve stavu $|0\rangle$, tak s jistotou víme, že funkce f je konstantní a pokud je ve stavu $|1\rangle$, je funkce vyvážená. Kýženého výsledku jsme dosáhli voláním funkce f pouze jednou oproti algoritmu pro klasický počítač, kde jsme museli funkci f volat dvakrát, abychom zadaný problém vyřešili.

Uvedený popis algoritmu můžeme zobecnit na Deutschův-Jozsův algoritmus. Funkci $f : \{0,1\} \to \{0,1\}$ můžeme rozšířit na větší množinu: $f : \{0,1\}^n \to \{0,1\}$. Funkci nazýváme vyváženou, pokud přesně polovinu vstupních hodnot vyhodnotí funkce f jako 0 a druhou polovinu jako 1. Pokud jsou všechny vstupní hodnoty vyhodnoceny jako 1 nebo jako 0, funkce je konstatní. Deutschův-Joszův algoritmus pro tohle zobecnění řeší, jestli je funkci konstantní nebo vyvážená.

Při použití klasických počítačů je pro vyřešení problému potřebný počet volání funkcí v nejhorším případě $2^{n-1} + 1$. Díky kvantovému počítání jsme schopni problém vyřešit pouze jedním voláním. Jde tedy o exponenciální zrychlení.

2.3.2 Kvantová teleportace

Pro shrnutí algoritmu v této podsekci bylo vycházeno z [15].

Nejedná se o teleportaci čehokoliv fyzického, ale o přenos stavu jednoho qubitu na druhý qubit způsobem, který je naprosto bezpečný. Díky tomu bychom byli schopni vytvořit nový bezpečný způsob komunikace.

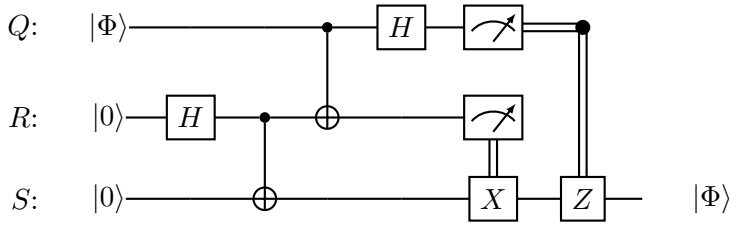
Princip komunikace si ukážeme na modelovém příkladu, kde Alice je odesílatel, který chce chce odeslat stav qubitu Q Bobovi.

1. Začneme nastavením tří qubitů.
   (a) Alice má qubit ve stavu $|\Phi\rangle$, jehož stav chce bezpečně poslat Bobovi.
   (b) K dosažení tohohle cílu, Alice začíná s dalšími dvěmi qubity - R a S. S bude poslán bobovi a R zůstane u Alice.
2. Alice připraví EPR pár s qubity R a S aplikací Hadamardova hradla na qubit R a potom CNOT mezi R a S (kontrolní qubit je R). V tento okamžik může Alice poslat qubit S Bobovi. (Pokud by byl qubit realizován např. fotonem, může být poslán přes optický kabel.)
3. Alice provede Bellovo měření na qubit Q a její část EPR pár - R. Toto se realizuje aplikací CNOT na tyto dva qubity, přičemž Q je kontrolní bit, následně aplikací Hadamardova hradla na Q a nakonec klasické měření obou qubity - tím dostaneme standardní výpočetní báze.
4. Po změření pošle Alice obě dvě hodnoty Bobovi klasickým komunikačním kanálem. Výsledkem měření můžou být 4 hodnoty: 00, 01, 10 nebo 11.

5. Podle toho, jaké hodnoty Bob dostane, provede operace nad svým qubitem S. Po provedení bude qubit S ve stejném stavu jako originální Q:

   - **00** - Žádná operace - qubit S je již ve stejném stavu jako Q
   - **01** - Aplikace hradla Z
   - **10** - Aplikace hradla X
   - **11** - Aplikace nejdřív hradla Z, potom X

Obvod kvantové teleportace vypadá následovně:



Kvantová teleportace má jistě využití v kvantové komunikaci, můžeme ji ovšem využít i v jiných částech kvantové počítání. Například při konstrukci kvantové modulární architektury může být využívána pro posílání qubitů mezi moduly kvantového počítače. [4]

2.4 Simulátor kvantových výpočtů

Se zájmem o kvantové počítání roste počet knihoven a nástrojů pro tohle prostředí s využitím všech nejpoužívanějších jazyků, jako jen např. Python, C/C++, Java a další. Několik předních výzkumných center volí jazyk Python jako nástroj pro tvorbu kvantových obvodů. Mezi důvody patří například jeho flexibilita nebo high-level přístup k programování, díky čemuž se programátor může soustředit hlavně na kvantové počítání a může vypustit formality programování jako takového. Díky tomu je možné se Python naučit relativně rychle. [14]

Většina dnešních kvantových počítačů není dostatečně velká, abychom na nich mohli provádět univerzální kvantové výpočty, jejich plné nasazení je tedy stále limitováno. Schopnost simulovat kvantové počítače malých rozměrů je nesmírně důležitá, protože umožňuje vědcům a inženýrům lépe pochopit výsledky, které by očekávali od kvantového stroje stejné velikosti. Simulace také umožňuje lepší pochopení kvantových počítačů jako takových. [11]

Existuje několik technik, jak simulovat univerzální kvantové počítač, všechny ale trpí "exponenciální explozí" klasické paměti. Abychom do paměti uložili celý stav systému s n qubity, musíme být schopni uložit 2^n komplexních čísel. Jako příklad si představme, že každé komplexní číslo vyžaduje jeden byte a systém, který chceme simulovat, má 30 qubitů. Pro takovou simulaci bychom potřebovali minimálně 2^{30} bytů, tedy gigabyte. Pro systém s 50 qubity je to už ale 2^{50}, tedy petabyte. Nároky na tak velkou pamět jsou za hranicemi dnešních nejlepších superpočítačů. [14]

Nejjednodušší metoda simulace kvantového počítače je vyjádření kvantového obvodu jako transformaci stavu $|\psi\rangle$ pomocí unitární matice U. Simulátor potom jednoduše provede maticové násobení pro získání nového stavu $|\psi'\rangle$. [14]

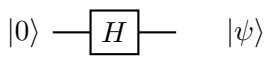
$$|\psi'\rangle = U\,|\psi\rangle \tag{2.46}$$

Všiměme si, že tato metoda vyžaduje ukládat v paměti celou unitární matici, která má velikost $2^n \times 2^n$. Ukládání stavu qubitu se nám tolik nedaří redukovat, nicméně existuje metoda, díky které nemusíme mít v paměti celou unitární matici. Můžeme mít uložené matice pro operaci s jedním nebo dvěmi qubity a ty aplikovat postupně. [14]

Existují i simulátory, které dokáží efektivně pracovat s velkým množstvím qubitů, jako například simulátor Cliffordova obvodu. Tyto obvody nicméně nejsou univerzální. [14]

Pro účely této bakalářské práce byl zvolen simulátor Quantum++, který nabízí simulaci univerzálního kvantového počítače. Na běžném stroji (Intel i5 8Gb RAM) je simulátor schopný v rozumném čase simulovat až 12 qubitů ve stavu superpozice. Implementace simulátoru je v jazyce C++, navíc existuje wrapper pro Python 3, který byl v této práci využíván. Na něm si ukážeme i příklady použití simulátoru: [11]

Uvedení qubitu do superpozice Základní nastavení a použití si ukážeme na simulaci tohohle jednoduchého kvantového obvodu:



Implementace simulace bude vypadat následovně:

```
from pyqpp import *

#Vytvoreni objektu kvantoveho obvodu. Parametr znaci, kolik ma obvod qubitu.
#Obvod s jednim qubitem, ktery je na zacatku ve stavu |0>
qc = QCircuit(1)

#aplikace Hadamardova hradla na qubit s indexem 0
qc.gate(gates.H, 0)

#spusteni simulace obvodu
engine = QEngine(qc)
engine.execute()

#vypis stavu systemu po provedni simulace obvodu
print(engine.get_psi())
```

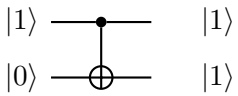
Výpis programu:

```
[0.70710678+0.j 0.70710678+0.j]
```

což odpovídá očekávanému výsledku

$$H\left|0\right\rangle = \frac{\left|0\right\rangle + \left|1\right\rangle}{\sqrt{2}} \tag{2.47}$$

Aplikace hradla CNOT Jako druhý příklad si uvedeme jednoduchou operaci na systému s dvěmi qubity. Pro jednoduchost bude uvedena jen definice obvodu. Import knihovny a funkce pro spuštění simulace a výpis stavu po provedení obvodu budou vynechány.



```
#vytvori obvod s jednim qubitem, pocatecni stav |00>
qc = QCircuit(2)

#nejdriv prvni qubit nastavime na hodnotu |1>
#s vyuzitim hradla NOT nebo tez zname jako Pauliho hradlo X
qc.gate(gates.X, 0)

#aplikace hradlca CNOT. Druhy parametr udava kontrolni bit
qc.gate(gates.CNOT, 0, 1)
```

Výpis programu:

```
[0.+0.j 0.+0.j 0.+0.j 1.+0.j]
```

což odpovídá očekávanému výsledku

$$|11\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \tag{2.48}$$

Kapitola 3

Přírodou inspirované optimalizační algoritmy

Evoluční algoritmy se inspirují konceptem přirozeného výběru, který je nám znám především díky vědci Charlesovi Darwinovi. Využívá skutečnosti, že znaky definucíjí jedince (organismus) jsou zakódovány v DNA a v průběhu času se mění díky křížení (vznik potomků, kteří zdědí vlastnosti svých rodiů) a mutacemi (náhodná změna v DNA). Díky přirozenému výběru mohou přežít pouze ti jedinci, kteří se lépe přizpůsobí prostředí (mají lepší vlastnosti), ve kterém existují. Jedinci s horšími vlastnostmi mají menší šanci na přežití a tedy i menší šanci na předání DNA do další generace. [3]

V rámci evolučních elgoritmů rozumíme populaci jako množinu jedinců, kde každý jedinec představuje kandidátní řešení problému. Kvalitu každého jedince vyhodnocuje fitness funkce. Ta každému jedinci přiřadí číslo, díky kterému můžeme porovnat, kteří jedinci jsou lepší. Typicky se snažíme najít jedince s co nejmenší hodnotou fintess funkce. Jedinec, kterému fitneess funkce přiřadí hodnotu 0, představuje ideální řešení. Na jedince z populace jsou v průběhu evoluce aplikovány genetické operátory (mutace), které mohou, ale také nemusí vygenerovat potomka, který má lepší fitness funkci. Tlak přirozeného výběru potom zapříčiní zlepšování populace - do dalších generací se budou dostávat obecně jedinci s lepší fitness funkcí. [3]

Obecně evoluční algoritmy můžeme popsat následovně:

```
1 Inicializace populace kandidátních řešení
2 while Není splněna ukončovací podmínka do
3 |   Výběr jednotlivců ze současné populace (rodičů)
4 |   Vygenerování nových jednotlivců (potomků) z vybraných rodičů
5 |   Nahrazení některých členů stávající populace nově vygenerovanými jedinci
6 end
```

Algoritmus 1: Obecný popis evolučních algoritmů [3]

Hlavními představiteli evolučních algoritmů jsou Genetický algoritmus, Evoluční strategie, Diferenciální evoluce, Genetické programování a Evoluční programování. [3]

Tato práce se zabývá návrhem kvantových operátorů použitím dvou algoritmů z hlavních představitelů: Evoluční strategie a Diferenciální evoluce a dvěma méně známými algoritmy: Optimamlizace hejnem částic a Optimalizace umělým včelstvem. V následujích podkapitolách budou blíže popsány principy těchto algoritmů.

3.1 Evoluční strategie

Popis algoritmu v této sekci vychází z [20].

Algoritmus evoluční strategie byl původě poblikován již v roce 1964, té době však nenašel velké uplatnění, protože jsme neměli dostupné počítače s velkým výpočetním výkonem.

3.1.1 Základní (1+1) varianta

Jedná se o nejprimitivnější variantu algorimu. V takové podobě byl původně navržen. (1+1) označení znamená, že populace má vždy pouze jednoho jedince a v každé generaci se generuje pouze jeden potomek. Jediný ladicí parametr je zde rozptyl normálního rozdělení σ^2.

Parametr σ^2 by měl být dostatečně velký, aby evoluce mohla za rozumný počet generací prohledat celý vyhledávací prostor a zároveň dostatečně malý, aby mohlo být efektivně nalezeno optimální řešení.

```
1 σ² = rozptyl mutace, σ² ∈ ℝ⁺
2 x₀ = náhodně vygenerovaný jedinec
3
4 while není splněna ukončovací podmínka do
5 |   Vygenerování náhodného vektoru r, r_i ~ N(0, σ²) pro i ∈ [1, n]
6 |   x₁ = x₀ + r
7 |   if f(x₁) je lepší než f(x₀) then
8 |   |   x₀ = x₁
9 end
```

Algoritmus 2: (1+1) Evoluční strategie

3.1.2 (μ, 1) varianta

Parametr μ zde uvádí, kolik rodičů je použito v každé generaci (tzn. velikost populace. Každý jedinec má také přiřazený vektor σ, který kontroluje velikost mutace. Mezi vybranými jedinci dojde ke křížení a následně je provedena mutace, čímž je vygenerováno nové kandidátní řešení. Do další generace potom postoupí μ nejlepších jedinců (resp. jeden nejhorší je zahozen). Nejlepší jedinci se v průběhu evoluce nezhoršují. Tomuto principu se říká elitismus.

Náhodný vektor r, který se stará o mutaci, je vygenerován tak, aby respektoval vektor v, který kontroluje velikost mutace jednotlivých prvků každého jedince.

Inicializace populace náhodných jedinců $\{x_k, \sigma_k\}$, kde $k \in [1, \mu]$

while *není splněna ukončovací podmínka* **do**
    Náhodný výběr dvou rodičů z populace $\{x_k, \sigma_k\}$
    Křížení rodičů pomocí zvolené metody, výsledkem je vektor $(x_{\mu+1}, \sigma_{\mu+1})$
    $\Sigma_{\mu+1} = diag(\sigma^2_{\mu+1,1}, \ldots, \sigma^2_{\mu+1,n})$
    Vygenerování náhodného vektoru r, $r_i \sim N(0, \Sigma_{\mu+1})$ pro $i \in [1, n]$
    $x_{\mu+1} = x_{\mu+1} + r$
    Přidání $x_{\mu+1}$ do populace jedinců
    Odstranění nejhoršího jedince z populace
end

Algoritmus 3: (μ, 1) Evoluční strategie

3.1.3 Metody křížení

Metod existuje více, níže si popíšeme 2 z nich.

Diskrétní křížení Každý prvek potomka je náhodně vybrát z jednoho z rodičů.

$$\text{parent } 1 = (\boldsymbol{y_{11}}, y_{12}, \boldsymbol{y_{13}}, \boldsymbol{y_{14}})(\sigma_{11}, \sigma_{12}, \boldsymbol{\sigma_{13}}, \boldsymbol{\sigma_{14}}) \tag{3.1}$$

$$\text{parent } 2 = (y_{21}, \boldsymbol{y_{22}}, y_{23}, y_{24})(\boldsymbol{\sigma_{21}}, \boldsymbol{\sigma_{22}}, \sigma_{23}, \sigma_{24}) \tag{3.2}$$

$$\text{potomek} = (y_{11}, y_{22}, y_{13}, y_{14})(\sigma_{21}, \sigma_{22}, \sigma_{13}, \sigma_{14}) \tag{3.3}$$

Intermediární křížení Každý prvek potomka je průměrem prvků obou rodičů.

$$\text{parent } 1 = (y_{11}, y_{12}, y_{13}, y_{14})(\sigma_{11}, \sigma_{12}, \sigma_{13}, \sigma_{14}) \tag{3.4}$$

$$\text{parent } 2 = (y_{21}, y_{22}, y_{23}, y_{24})(\sigma_{21}, \sigma_{22}, \sigma_{23}, \sigma_{24}) \tag{3.5}$$

$$\text{potomek} = (\tfrac{y_{11}+y_{21}}{2}, \tfrac{y_{12}+y_{22}}{2}, \tfrac{y_{13}+y_{23}}{2}, \tfrac{y_{14}+y_{24}}{2})(\tfrac{\sigma_{11}+\sigma_{21}}{2}, \tfrac{\sigma_{12}+\sigma_{22}}{2}, \tfrac{\sigma_{13}+\sigma_{23}}{2}, \tfrac{\sigma_{14}+\sigma_{24}}{2}) \tag{3.6}$$

3.1.4 (μ, λ) a (μ + λ) varianty

Dalším zobecněním je přidání λ parametru. Dosud jsme do každého generace generovali pouze jednoho potomka, teď budeme generovat λ potomků.

Velikost populace je vždy μ, vegenerováním λ nových jedinců tedy musíme vždy λ nejhorších odstranit. Můžeme vybírat z množiny rodičů i potomků. Do dalších generací se tedy budou dostávat jak rodiče, tak potomci. To se označuje jako varianta (μ + λ)

Dalším způsobem je vybírat do další generace pouze z nově vygenerovaných jedinců. Jinak řečeno, rodiče se nedostávají do další generace. Žádný jedinec tedy neexistuje ve více než jedné generaci. Tenhle způsob se označuje (μ, λ). Je zřejmé, že musí plati $\lambda \geq \mu$, tedy vždy musíme vygenerovat více jedinců než je velikost populace.

(μ, λ) obecně funguje lépe, pokud je fitness funkce časově proměnlivá. V (μ + λ) může mít jedinec dobrou fitness, ale nevhodné σ. Takový jedinci budou zůstávat v populaci a tím znesnadňovat zlepšování.

3.1.5 Samoadaptace

Doteď jsme neměli žádný způsob, jak v průběhu evoluce adaptovat vektor rozptylu σ. Tak jako mutujeme vektory jedinců představující kandidátní řešení, můžeme mutovat vektor σ. Jakmile je vytvořen potomek křížením, zmutujeme vektor rozptylu následujícím způsobem:

$$\sigma_i = \sigma_i exp(\tau' p_0 + \tau p_i) \tag{3.7}$$

$$x_i = x_i + \sigma_i r_i \tag{3.8}$$

pro $i \in [1, n]$, kde p_0, p_1 a r_i jsou náhodná čísla z $N(0, 1)$. τ' a τ jsou ladící parametry. Doporučené hodnoty:

$$\tau = P_1 \left(\sqrt{2\sqrt{n}}\right)^{-1} \tag{3.9}$$

$$\tau' = P_2 \left(\sqrt{2n}\right)^{-1} \tag{3.10}$$

kde P_1 a P_2 jsou proporční konstanty. Typicky mají hodnotu 1.

3.1.6 Více rodičů

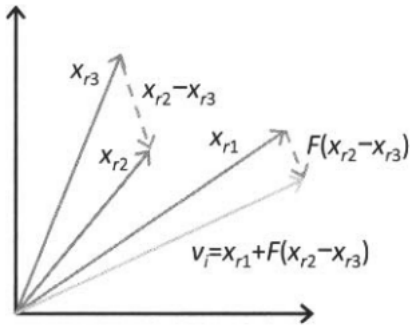
V popisu algoritmu jsme zatím vždy uváděli, že pro vygenerování potomka se použijí vždy pouze dva rodiče. Tento způsob můžeme rozšířit a použít obecně více rodičů a to jak u intermediárního, tak diskrétního křížení.

U diskrétního se prvky potomka jednoduše vybírají náhodně ze všech rodičů, kdy na každého rodiče připadá stejná pravděpodobnost. U intermediárního se vypočítá průměr všech rodičů.

3.2 Diferenciální evoluce

Popis algoritmu v této sekci vychází z [20].

Původní myšlenky tohohle algoritmu byli publikovány v roce 1997 v [21]. Je unikátní v tom, že sice patří mezi evoluční, nebyl ale inspirován biologií. Jedná se o populačně založený algoritmus navržený pro optimalizaci funkcí v n-rozměrném spojitém prostoru. Jedince v populaci reprezentují n-rozměrné vektory. Hlavní myšlenkou je vypočítat rozdíl dvou vektorů z populace a ten posléze přičíst k třetímu vektoru, čímž získáme nové kandidátní řešení. Tento princip můžeme vidět na obrázku níže.



Obrázek 3.1: Princip diferenciální evoluce ilustrovaná na dvourozměrném problému

x_{r1}, x_{r2} a x_{r3} jsou kandidátní řešení. Zmenšená verze rozdílů mezi x_{r2} a x_{r3} je přičtena k x_{r1}, čímž získáváme nový mutační vektor v_i.

Mutační vektor je dále zkombinován (je provedeno křížení) s x_i, kde $i \notin \{x_1, x_2, x_3\}$, čímž je vytvořen vektor nového kandidátního řešení u_i. Křížení je prováděno následovně:

$$u_{ij} = \begin{cases} v_{ij} & \text{pokud } (r_{cj} < c) \text{ nebo } (j = J_r) \\ x_{ij} & \text{v opačném případě} \end{cases} \tag{3.11}$$

pro $j \in [1, n]$ kde n je rozměr prohledávaného prostoru, c je konstantní koeficient křížení a J_r náhodné celé číslo z intervalu $[1, n]$. Koeficient křížení udává, jak moc bude pro kandidátní řešení použit mutační vektor. Základní algoritmus vypadá následovně:

```
F = škálovací koeficient, F ∈ [0.4, 0.9]
c = koeficient křížení, c ∈ [0.1, 1]
Inicializace populace kandidátních řešení {x_i}, i ∈ [1, N]

while není splněna ukončovací podmínka do
    foreach jedinec x_i, i ∈ [1, N] do
        r_1 = náhodné celé číslo ∈ [1, N],   r_1 ≠ i
        r_2 = náhodné celé číslo ∈ [1, N],   r_2 ∉ {i, r_1}
        r_3 = náhodné celé číslo ∈ [1, N],   r_3 ∉ {i, r_1, r_2}
        v_i = x_r1 + F(x_r2 − x_r3) // Mutační vektor
        J_r = náhodné celé číslo ∈ [1, n]]

        foreach rozměr j ∈ [1, n] do
            r_cj = U(0, 1)
            if (r_cj < c) nebo (j = J_r) then
                u_ij = v_ij
            else
                u_ij = x_ij
            end
        end
    end

    foreach index jedice populace i ∈ [1, N] do
        if f(u_i) < f(x_i) then
            x_i = u_i
    end
end
```

Tato podoba algoritmu diferenciální evoluce se nazývá klasická, nebo také DE/rand/1/bin. To znamená, že vektor x_{r1} je volen náhodně (rand), pracuje se jen s jedním rozdílem vektorů (1) a počet prvků mutačního vektoru, který přispívá ke kandidátnímu řešení, odpovídá binomickéhu rozdělení.

Existuje mnoho úprav a rozšíření diferenciální evoluce. V této práci se jimi ale zabývat nebudeme.

3.3 Optimalizace hejnem částic

Popis algoritmu v této sekci vychází z [20].

Dále jen PSO (z anglického Particle Swarm Optimization). Stáda zvířat jsou často schopná se vyhnout útoku predátorů lépe než jedinci. Například lev dokáže lépe rozeznat jedinou zebru díky kontrastu černobílých s okolním prostředí, ale jedinci ve stádu zeber zapadnout a není jednoduché je rozeznat odděleně. Zvířata ve stádech jsou také úspěšnější v hledání potravy než jedinci samotní. Také se ve stádech pohybují rychleji než sami. Jistě bychom v přírodě našli další příklady, kdy je pro jedince výhodnější fungovat ve skupinách.

Algoritmus PSO je inspirovaný pozorováním právě těchto skupin jedinců, kteří spolupracují, aby dosáhli většího úspěchu, než kdyby ke spolupráci nedocházelo. Nemusíme ale spoléhat jen na říši zvířat, PSO je založený na těchto základních principech, které můžeme vidět i na lidech:

Setrvačnost Máme tendenci držet se starých způsobů, které se v minulosti osvědčily.

Vliv společnosti Slyšíme o úspěchu těch nejlepších v naší společnosti a snažíme se napodobit jejich přístup.

Vliv okolí Nejvíc věcí se učíme od lidí, kteří jsou nám nejblíž. Naši přátelé nás ovlivňují více než společnost jako taková.

3.3.1 Základní algoritmus

Mějme množinu jedinců představující kandidátní řešení $\{x_i\}$ a pro každého jedince rychlost v_i, kterou se pohybuje po prohledávaném prostoru. Tento pohyb je základ PSO a hlavní aspekt, kterým se PSO liší od ostatních evolučních algoritmů.

Jedinec se pohybuje vyhledávacím prostorem s určitou setrvačností, proto má tendenci udržovat svoji rychlost, ta se ovšem může z různých důvodů měnit. Algoritmus si pamatuje nejlepší pozici v minulosti, ke které má tendenci se vracet. Druhým důvodem je nejlepší pozice jedince v nejbližším okolí, ke které má jedinec také tendenci směřovat. Tyto dva faktory náhodně ovlivňují rychlost každého jedince.

Základní algoritmus je popsaný níže:

```
Inicializace populace náhodných jedinců {x_i}, i ∈ [1, N]
Inicializace vektoru rychlosti pro každého jedince v_i, i ∈ [1, N]
Inicializace nejlepšího jedince za celou dobu evoluce b_i
σ = velikost okolí, σ < N
φ_{1,max}, φ_{2,max} = koeficienty maximálního vlivu
v_max = maximální rychlost

while není splněna ukončovací podmínka do
    foreach jedinec x_i, i ∈ [1, N] do
        H_i = {σ nejbližších sousedů x_i}
        h_i = jedinec s nejlepší fitness z okolí H_i
        // U značí rovnoměrné pravděpodobnostní rozdělení
        Vygenerování náhodného vektoru φ_1, φ_1(k) ~ U(0, φ_{1,max}) pro k ∈ [1, n]
        Vygenerování náhodného vektoru φ_2, φ_2(k) ~ U(0, φ_{2,max}) pro k ∈ [1, n]
        v_i = v_i + φ_1(b_i − x_i) + φ_2(h_i − x_i)

        if |v_i| > v_max then
            v_i = v_i v_max/|v_i|

        x_i = x_i + v_i
        b_i = argmin{f(x_i), f(b_i)}
    end
end
```

Algoritmus 4: Optimalizace hejnem částic

Algoritmus má v základní podobě několik parametrů, kterými můžeme evoluci ladit:

1. **velikost okolí** σ - $\sigma < N$. Malé okolí se lépe vyhýbá lokálním minimům, s použitím velkého okolí evoluce rychleji konverguje.

2. **maximální míra kongnitivního učení** ϕ_1 **a maximální míra sociálního učení** ϕ_2

3. **maximální rychlost** v_{max} - doporučuje se parametr nastavit na 10% až 20% rozsahu vyhledávacího prostoru. [8]

3.3.2 Možná jednoduchá přizpůsobení algoritmu

Algoritmus nabízí spoustu možností, jak si výpočet přizpůsobit. Incializaci vektorů rychlosti můžeme například provést náhodně nebo všechny nastavit na nulu.

Úprava rychlosti může být zjednodušena následovně:

$$v_i = v_i + \Phi_1(b_i - x_1) + \Phi_2(h_1 - x_1) \tag{3.12}$$

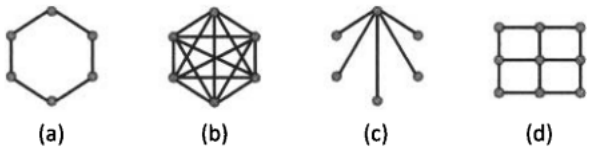
kde Φ_1 a Φ_2 nejsou vektory, ale skalární hodnoty (desetinná čísla). Tato modifikace se nazývá lineární PSO. [18] PSO s takovou úpravou má ovšem většinou horší výsledky než původní verze.

Provedení pohybu $x_i = x_i + v_i$ může vést k přesunutí jedince mimo hranice prohledávaného prostoru. Tomu můžeme zamezit například definicí hranic prostoru a ověřením, zda

se jedinec nedostal za hranice.

$$x_i = min(x_i, x_{max}) \tag{3.13}$$
$$x_i = max(x_i, x_{min}) \tag{3.14}$$

Základní algoritmus říká, že každého jedince ovlivňuje jeho nejbližší okolí. Uspořádání okolí se nazývá topologie hejna. Okolí se může měnit s každou generací (dynamické okolí) nebo zůstává stejné po celou dobu evoluce (statické okolí). Výzkumníci experimentovali s mnoha topologiemi. V ?? můžeme vidět příklady některých topologií. (a) představuje prstencovou topologii, (b) je celá topologie, (c) kolečková topologie a (d) čtvercová.



Obrázek 3.2: Ukázky topologií

3.3.3 Váhování setrvačnosti

Abychom se vyhnuli limitování rychlosti pomocí parametru v_{max}, můžeme udělat úpravy v rovnici, která modifikuje rychlost jedinců, díky kterým se nebude rychlost zvyšovat za přijatelné hranice. Jedním ze způsobů je použít váhování setrvačnosti.

Jak můžeme vidět v algoritmus 4, jedinci mají tendenci si zachovávat svoji rychlost v průběhu evoluce, nicméně jisté změny jsou možné vzhledem ke kognitivnímu a sociálnímu učení.

$$v_i(k) = v_i(k) + \phi_1(k)(b_i(k) - x_i(k)) + \phi_2(k)(h_i(k) - x_i(k)), \quad k \in [1, n] \tag{3.15}$$

kde n je dimeze řešeného problému. Bylo empiricky zjištěno, že snižování setrvačnosti v průběhu evoluce může vést k lepším výsledkům. Rovnice 3.15 může být upravena na:

$$v_i(k) = \mathbf{w}v_i(k) + \phi_1(k)(b_i(k) - x_i(k)) + \phi_2(k)(h_i(k) - x_i(k)) \tag{3.16}$$

kde w je váha setrvačnosti, která běžně začíná přibližně na 0.9 v první generaci a končí na cca 0.4 v poslední generaci. [9]

PSO může být dále modifikován např. přidáním koeficientu zúžení [5] nebo dalšími úpravami, které ovšem pro účely této práce nebyly implemenetovány.

3.4 Optimalizace umělým včelstvem

Popis algoritmu v této sekci vychází z [20].

Dále jen ABC (z anglického názvu Artificial Bee Colony). Jedná se o algoritmus inspirován chováním včel. Původně byl publikován v [16]. Je založen na hledání optimálního zroje potravy. Umístění zdroje potravy je analogické k lokaci v prohledávaném prostoru optimalizačního problému. Množství nektaru zdroje potravy je analogické k hodnotě fitness funkce. ABC simuluje tři různé typy včel:

Prvním typem jsou včelí sběrači, kteří cestují mezi úlem a zdrojem potravy. Každý sběrač se spjat s jedním zdrojem potravy, ze kterého nosí med do úlu. Kromě toho také provádí lokální prohledávání okolo svého zdroje potravy. Pokud najde lepší zdroj, starý opustí a začne nosit nektar z nového.

Druhý typ včel, pozorovatelé, nemají přiřazený žádný zdroj jídla, ale pozorují chování sběračů po příletu do úlu s nektarem. Vyhodnocují, kolik sběrači nosí nektaru (počet nektaru je reprezentovaný hodnotou fitness funkce) a na základě toho se (s pomocí pravděpodobnosti) rozhodují, kde se budou hledat nové zdroje jídla.

Třetí typ včel jsou průzkumníci. Ani tyhle včeli, stejně jako pozorovatelé, nejsou přirazeny k žádnému zdroji jídla. Průzkumníci pozorují jednotlivé sběrače, jak dobře se jim daří nacházet nové a lepší zdroje potravy (resp. jestli množství nektaru, které nosí do úlu, se zvětšuje). Pokud sběrači stagnují a po dlouhou dobu nosí do úlu pořád stejné množství nektaru, průzkumník určí náhodně nový zdroj v prohledávaném prostoru a přiřadí ho sběrači.

Jak můžeme vidět v popisu algoritmu níže, každý sběrač náhodně mění svoji pozici v prohledávaném prostoru. Pokud tahle změna vede k lepšímu výsledku (lepší hodnota fitness), sběrač se přesouvá na novou pozici. Pozorovatelé také náhodně mění pozici sběračů, kde sběrač je náhodně vybrán. I zde následuje rozhodnutí, zda modifikace vedla k lepšímu výsledku. Pokud ano, sběrač se přesouvá. Nakonec průzkumník nahradí zdroje, které po určitém počtu náhodných modifikací nevedli k žádnému zlepšení, náhodným novým zdrojem. Tímto způsobem je algoritmus schopný eliminovat lokální optima. Hodnota $T(x_i)$ udává, kolik náhodných úprav bez zlepšení proběhlo.

Algoritmus má 2 parametry:

1. P_f - velikost populace sběračů. Tímto parametrem je zároveň určena velikost populace pozorovatelů. Obvykle se volí hodnota $N/2$.

2. L - limit stagnace. Obvykle se volí $L = Nn/2$.

Alogoritmus lze modifikovat různými způsoby. V této práci byl však použit algoritmus v základní podobě - tak, jak je popsaný níže.

```
1  N = velikost populace
2  L = limit stagnace (kladné celé číslo)
3  P_f = velikost populace sběračů (P_f < N)
4  P_o = velikost populace pozorovatelů, P_o = N − P_f
5  Inicializace náhodné populace sběračů {x_i}, i ∈ [1, P_f]
6  Inicializace počítadla pokusů sběračů T(x_i) = 0, i ∈ [1, P_f]
7
8  while není splněna ukončující podmínka do
9  |
10 |  Včelí sběrači:
11 |  foreach sběrač x_i, i ∈ [1, P_f] do
12 |  |  k = náhodné číslo z intervalu [1, N]. k ≠ i
13 |  |  s = náhodné číslo z intervalu [1, n]
14 |  |  r = U(−1, 1) ;   // U značí rovnoměrné pravděpodobnostní rozdělení
15 |  |  v_i(s) = x_i(s) + r(x_i(s) − x_k(s))
16 |  |  if f(v_i) je lepší než f(x_i) then
17 |  |  |  x_i = v_i  T(x_i) = 0
18 |  |  else
19 |  |  |  T(x_i) = T(x_i) + 1
20 |  |  end
21 |  end
22 |
23 |  Včelí pozorovatelé:
24 |  foreach pozorovatel v_i, i ∈ [1, P_o] do
25 |  |  Výběr sběrače x_j, kde Pr(x_j) ∝ fitness(x_j), pro j ∈ [1, P_f]
26 |  |  k = náhodné číslo z intervalu [1, P_f]. k ≠ j
27 |  |  s = náhodné číslo z intervalu [1, n]
28 |  |  r = U(−1, 1)
29 |  |  v_i(s) = x_j(s) + r(x_j(s) − x_k(s))
30 |  |  if f(v_i) je lepší než f(x_j) then
31 |  |  |  x_j = v_i  T(x_j) = 0
32 |  |  else
33 |  |  |  T(x_j) = T(x_i) + 1
34 |  |  end
35 |  end
36 |
37 |  Včelí průzkumníci:
38 |  foreach sběrač x_i, i ∈ [1, P_f] do
39 |  |  if T(x_i) > L then
40 |  |  |  x_i = náhodně vygenerovaný nový jedinec
41 |  |  |  T(x_i) = 0
42 |  end
43 end
```

Algoritmus 5: Optimalizace umělým včelstvem

Kapitola 4

Evoluční návrh kvantového operátoru

4.1 Reprezentace kvantového operátoru

Pro evoluční návrh kvantových operátorů bylo již dříve použito více složitějších reprezentací, například reprezentace Hutsell & Greenwood, McKinnon nebo pomocí QR dekompozice. Tyto metody využívají různé principy pro konstukci unitárních matic. [24]

Například QR dekompozice využívá toho, že matice $N \times N$ s komplexními čísly může být rozložena na $A = QR$, kde Q je unitární matice. Tímto způsobem je jedinec v evoluci reprezentován maticí A, ze které jsme schopni získat unitární matici Q pomocí QR dekompozice, která reprezentuje kvantový operátor, který chceme získat. [24]

Tato práce se zaměřuje na návrh kvantových operátorů bez použití složitějších matematických aparátů. Unitární matice jsou v evoluci reprezentovány přímo vektorem komplexních čísel:

$$\begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{pmatrix} \rightarrow (c_{11}, c_{12}, c_{13}, c_{21}, c_{22}, c_{23}, c_{31}, c_{32}, c_{33}) \tag{4.1}$$

Například Hadamardovo hradlo (viz 2.2.3):

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \tag{4.2}$$

bude ve zvolené reprezentaci vypadat následovně:

$$(0.70710678 + 0.j, 0.70710678 + 0.j, 0.70710678 + 0.j, -0.70710678 + 0.j) \tag{4.3}$$

V navržené reprezentaci jsou prvky vektoru komplexní čísla. Pro jejich reprezentaci byla implementována třída zapouzdřující reálnou a imaginární hodnotu a také operace nad komplexními čísly. Na veškeré operace nad prvky se tedy pohlíží jako na komplexní matematické operace.

4.2 Nastavení algoritmů

4.2.1 Evoluční strategie

Algoritmus byl implementován ve všech variantách, které jsou popsané v 3.1. Algoritmus má mnoho varianta a parametrů, kterými můžeme evoluci ladit. Tenhle algoritmus byl již použit pro konstrukci kvantových operátorů [24, 22], proto bylo při nastavování parametrů přihlíženo k dosaženým výsledkům v těchto publikacích.

Bylo zjištěno, že lepší výsledky dává plus strategie (výběr jedinců do další generace probíhá z rodičů i potomků). V publikaci [24] vycházelo velice dobře nastavení (15+100), tedy velikost populace je 15 a každou generaci se vytváří 100 potomků, s implementovanou samoadaptací. Rozptyl normálního rozložení, kterým se kontroluje velikost mutace, zde byl nastaven na 0.3.

V této práci bylo použito výše popsané nastavení jako výchozí. V experimentech budeme zkoumat, jaký vliv má na evoluci různé typy křížení - intermediární a diskrétní a jak evoluci ovlivní použití více rodičů pro křížení. Konkrétně budeme sledovat dva a šest rodičů.

4.2.2 Diferenciální evoluce

Implementace algoritmu byla provedena podle popisu v předchozí kapitole 3.2, tedy v základní variantě DE/rand/1/bin. Algoritmus v této podobě byl již použit pro návrh kvantových operátorů v [24]. V této publikaci se ukázalo, že si algoritmus vede velice dobře při nastavení velikosti populace na 20.

V této práci bude algoritmus nastaven na velikost populace 20. Škálovací faktor evoluce bude nastaven na hodnotu 0.7. Experimentálně bylo zjištěno, že kolem této hodnoty vede evoluce k nejlepším výsledkům. V experimentech budeme zkoumat, jaký má na výsledek vliv koeficient křížení. Ten budeme nastavovat na čtyři hodnoty: 0.1, 0.3, 0.6 a 0.9.

4.2.3 Optimalizace hejnem částic

Implementace algoritmu byla provedena podle popisu v předchozí kapitole 3.3 s využitím váhování setrvačnosti. Váhování setrvačnosti je definováno jako konstantní parametr.

Pro definici okolí byla použita celá topologie. Výběr nejbližšího okolí jedince je implementováno tak, že se pro daného jedince vypočítá vzdálenost ke všem ostatním jedincům v populaci. Jedinci vy populaci jsou posléze seřazeni vzestupně podle vzdálenosti od vybraného jedince a je vybráno prvních x jedinců, kde x je velikost okolí.

Rychlost jedinců na začátku evoluce je generována stejné jako hodnoty jedinců, tedy pomocí rovnoměrného rozdělení na intervalu $[-1, 1]$.

Literatura udává doporučené hodnoty váhy setrvačnosti, maximální míru kongnitivního učení a maximální míru sociálního učení a velikosti populace pro danou velikost problému [20]. Bohužel se experimentálně ukázalo, že pro problém konstrukce kvantových operátorů tyto doporučené hodnoty nevedou k dobrým výsledkům. Experimentálně byly proto tyto hodnoty vyladěny tak, aby evoluce konvergovala k rozumného řešení:

Parametr	Hodnota
v_{max}	0.4
w	-0.2
σ	30

V experimentech budeme sledovat, jakých výsledků bude algoritmus dosahovat s těmito hodnotami a různým poměrem mezi maximální mírou kognitivního ϕ_1 učení a maximální mírou sociálního učení ϕ_2. Budeme konkrétně sledovat hodnoty (0.9, 1.3), (0.3, 2), (1.3, 0.9), (2, 0.3), kde první hodnota v závorce je ϕ_1 a druhá ϕ_2.

4.2.4 Optimalizace umělým včelstvem

Implementace algoritmu byla provedena podle popisu v předchozí kapitole 3.4. Nastavení parametrů algorimu bylo provedeno podle doporučení, tedy:

1. P_f (velikost populace sběračů) bylo nastaveno podle $N/2$
2. L (limit stagnace) bylo nastaveno podle $Nn/2$

V popisu algoritmu 3.4 je uvedeno, že včelí pozorovatelé si vybírají sběrače na základě pravděpodobnosti, která je nepřímo úměrná hodnotě fitness funkce sběrače (čím menší fitness, tím větší pravděpodobnost, že bude vybrán). Pravděpodobnost každého sběrače je vypočítáná následujícím způsobem:

$$Pr(x_i) = \frac{fitness(x_i)}{\sum_{j=0}^{N} fitness(x_j)} \tag{4.4}$$

Pro každého pozorovatele potom vygenerováno náhodné číslo z normálního rozdělení $o_k = N(\mu, \sigma^2)$ s parametry:

$$\mu = \frac{1}{P_f} \tag{4.5}$$

$$\sigma^2 = \frac{0.15}{P_f} \tag{4.6}$$

kde P_f je velikost populace sběračů.

Z populace sběračů je posléze vybrán takový jedinec, pro kterého platí

$$o_k < Pr(x_i) \tag{4.7}$$

Experimentálně bylo ověřeno, že evoluce s tímto nastavením vede k dobrým výsledkům. V experimentech níže bude porovnán algorimus porovnán s použitím různé velikosti populace (myšleno celé populace, sběrači i pozorovatelé), přitom populace sběračů bude vždy stejné velká - 10. Budou porovnány tyto velikosti populace: 10, 20, 30, 50.

Jelikož tento algorimus funguje na podobných principech jako diferenciální evoluce [20] a evoluční návrh kvantových operátorů byl již dříve na tomto algoritmu ukázán [24], velikost populace byla inspirována nejlepším dosaženým výsledkem tímto algoritmem.

4.3 Vyhodnocení kandidátního řešení

Vyhodnocení kandidátních řešení bylo implementováno podle prací, které se již v minulosti zabývaly využití evolučních algoritmů pro návrh kvantového operátoru. [24, 22]

Jak bylo již dříve popsáno (viz 2.2.1), aplikace kvantového operátoru na stav můžeme popsat jako vynásobení vektoru (reprezentující stav) unitární maticí (reprezentující operátor):

$$|o\rangle = U\,|i\rangle \tag{4.8}$$

Cílem je najít takový kvantový operátor U, pro který tento vztah platí. Obecně mějme trénovací množinu $M = (|i_1\rangle, |o_1\rangle), \ldots (|i_j\rangle, |o_j\rangle)$, hledáme tedy takovou unitární matici, která splňuje 4.8 pro každou dvojici $x_k \in M$.

Pro vyhodnocení kvality kandidátního řešení je implementována následující fitness funkce:

$$fitness(U) = \frac{\sum_{v=0}^{|M|} \sum_{w=0}^{N} | \, |o_v\rangle \, (w) - U \, |i_v\rangle \, (w)|}{|M|N} \tag{4.9}$$

kde N je velikost stavového vektoru, v označuje index dvojice v trénovací množině M a w index prvku stavového vektoru. Pro každou dvojici $(|o_v\rangle, |i_v)\rangle$ z trénvací množiny se na $|i_v\rangle$ aplikuje kandidátní řešení v podobě matice U a provede se rozdíl prvků výsledného stavu a požadovaného stavu $|o_v\rangle$. Rozdíly se sečtou a nakonec se výsledek vydělí součinem velikosti trénovací množiny a velikosti stavového vektoru. Díky tomu se hodnota fitness nezvětšuje s větší trénovací množinou nebo s větším počtem qubitů, se kterými operátor pracuje.

Trénovací množina je vygenerována pomocí simulátoru kvantového počítače (viz 2.4). Pro samotnou aplikaci operátoru na vstupní stav v rámci vyhodnocování fitness funkce nebyl využit simulátor. Jde o jednoduchou matematickou operaci mezi mezi vektorem a maticí, která byla přímo implementováno.

Nalezené nejlepší řešení je posléze ověřeno pomocí simulátoru kvantového počítače.

Trénovací množina pro návrh hradla NOT (viz 2.2.2) může vypadat takto:

$$M = \left\{ [\begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}], [\begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}] \right\} \tag{4.10}$$

Kapitola 5

Experimentální výsledky

Evoluční algoritmy s různými nastaveními (viz 4.2) byly otestovány na návrhnu dvou hradel. Každý algoritmus byl ve všech nastaveních spuštěn celkem 48 krát. Jelikož se jedná o náročnější výpočetní úlohy, byl pro experimenty využit superpočítač Karolina.

U každého experimentu sledujeme nejlepší fitnes funkci kandidátního řešení získaného v průběhu evoluce ze všech 48 běhů. Dále sledujeme průměr nejlepších dosažených hodnot fitness ze 48 běhů.

Průběh evoluce sledujeme na grafech, kde každá křivka představuje jeden běh. Zde sledujeme nejlepší fitness v populaci v závislosti na generaci.

Dále sledujeme statistické zhodnocení dosažených výsledků ze všech 48 běhů na krabicových grafech.

Nejlepší dosažené hodnoty fitness a algoritmus s nastavením, který tohoto výsledku dosáhl, jsou v tabulce označeny tučně.

Nakonec je pro každou úlohu uvedena unitární matice, která dosáhla ze všech experimentů nejlepší hodnotu fitness a je ukázáno, jak tato matice funguje v simulátoru kvantového počítače na jednom z prvků trénovací množiny.

5.1 Značení experimentů

Experimenty se v tabulkách a grafech pro přehlednost označují zkratkami algoritmů a jejich nastavení. Formát každého označení můžeme zjednodušit na:
`<evoluční algoritmus>-<použité nastavení evolučního algoritmu>`

Evoluční algoritmy jsou označeny zkratkami, které byly již dříve v této práci použity, pro úplnost si je zde ovšem uvedeme.

- **ES** - Evoluční strategie
- **DE** - Diferenciální evoluce
- **PSO** - Optimalizace hejnem částic
- **ABC** - Optimalizace umělým včelstvem

Nastavení evolučních algoritmů má různé formáty, proto si je popíšeme každý zvlášť:

Evoluční strategie `<metoda křížení>-<počet rodičů využitých pro křížení>`
kde metoda křížení může být buď `D` - diskrétní nebo `I` - intermediární. například `ES-D-6`

znamená evoluční strategie s disktrétním křížením a šesti rodiči pro vygenerování nového potomka.

Diferenciální evoluce `C<hodnota koeficientu křížení>`
například `DE-C0.9` znamená diferenciální evoluci, kde koeficient křížení má hodnotu 0.9.

Optimalizace hejnem částic `(<maximální míra kognitivního učení>-<maximální míra sociálního učení>)`
například `PSO-(0.3-0.9)` znamená optimalizace hejnem částis s hodnotou maximální míry kognitivního učení 0.3 a hodnotou maximální míry sociálního učení 0.9.

Optimalizace umělým včelstvem `P<velikost celkové populace>` například `ABC-P30` znamená optimalizaci umělým včelstvem s celkovou velikostí populace 30.

5.2 CNOT

Prvním experimentem je pokus o návrh hradla CNOT (viz 2.2.3). Víme, že hradlo CNOT můžeme popsat následující unitární maticí:

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{5.1}$$

Tuhle matici se budeme pokoušet najít. Trénovací množina byla sestavena ze čtyř standardních výpočetních bází, vypadá následovně:

$$M = \left\{ [\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}], [\begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}], [\begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}], [\begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}] \right\} \tag{5.2}$$

Každý z experimentů byl ukončen po vyhodnocení fitness funkce 40 000× .

5.2.1 Výsledky

Nejlepší výsledek se podařilo získat pomocí experimentu `DE-C0.3`. Unitární matice vypadá následovně:

$$U = \begin{pmatrix} 1 & 0 & -0 & 0 \\ -0 & 1 & -0 & -0 \\ -0 & -0 & -0 & 1-0i \\ 0-0i & -0-0i & 1-0i & -0-0i \end{pmatrix} \tag{5.3}$$

Po aplikaci získané matice na $|00\rangle$ (první prvek z trénovací množiny) dostaneme přesně očekávaný výsledek:

$$|\psi\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{5.4}$$

Tabulka 5.1: Výsledky návrhu hradla CNOT

Algoritmus	Nastavení algoritmu	Nejlepší fitness	Průměrná fitness
ES	D-2	1.12424e-05	0.00561542
	D-6	4.93169e-07	0.00095630
	I-2	0.000442112	0.01173110
	I-6	0.00079083	0.00912911
DE	C0.1	1.12064e-14	2.85301958e-14
	C0.3	**5.59138e-16**	**2.14408222e-15**
	C0.6	5.79808e-16	1.29016833e-13
	C0.9	1.30346e-05	0.00574697
PSO	(0.9-1.3)	0.00289556	0.00523018
	(1.3-0.9)	0.00283067	0.00520601
	(0.3-2)	0.00565104	0.00884034
	(2-0.3)	0.0073226	0.01421913
ABC	P10	6.48525e-13	0.00065602
	P20	1.14432e-11	0.00022435
	P30	7.3576e-13	0.00021201
	P50	2.28632e-11	0.00024555

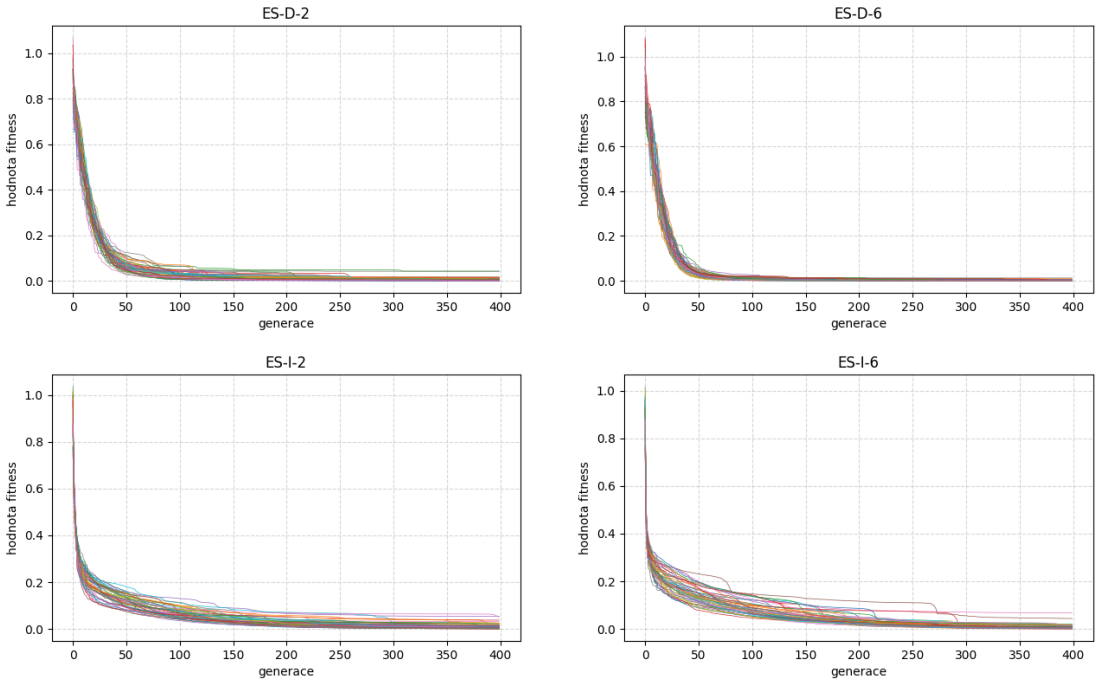
Evoluční strategie si v této úloze vedla daleko lépe ve variantě s diskrétním křížením. U diskrétního křížení se nejlépe osvědčila varianta s použitím 6 rodičů pro křížení.

Diferenciální evoluce si v této úloze vedla nejlépe. Při nastavení koeficientu křížení na hodnotu blízkou 1 se nejlepší dosažené výsledky oproti jiným nastavením algoritmu diferenciální evoluce výrazně zhoršují, i tak ale dosahuje dobrých výsledků.

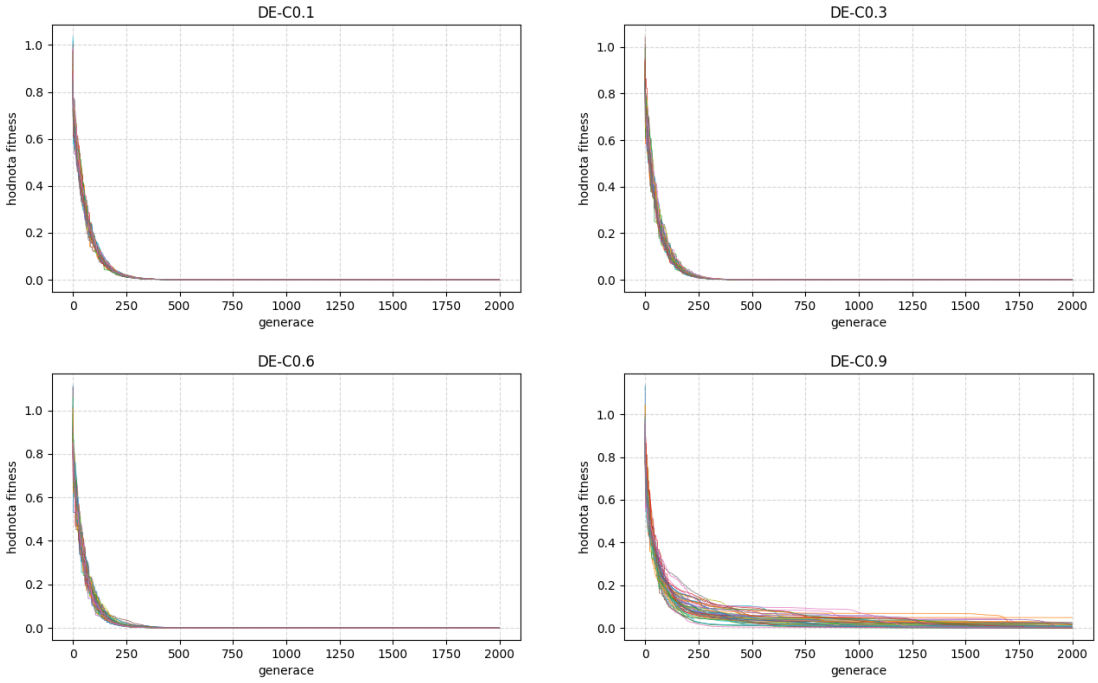
Optimalizace hejnem částic si v této úloze vedla nejhůře. O něco lepší nastavení se ukázala ta, kde mezi maximální mírou kognitivního učení a maximální mírou sociálního učení je malý rozdíl. Celkově ale mezi jednotlivými výsledky nebyly velké rozdíly.

Poslední algoritmus, optimalizace umělým včelstvem, si vedl velice dobře. Ve všech variantách byl schopný dosáhnout nejlepších výsledků s fitness hodnotou velmi blízkou nule. Celkovou úspěšností se blíží k výsledkům diferenciální evoluce. Nejlepších výsledků dosahoval při velikosti populace 10 a 20, velký rozdíl mezi nastavením ale není.

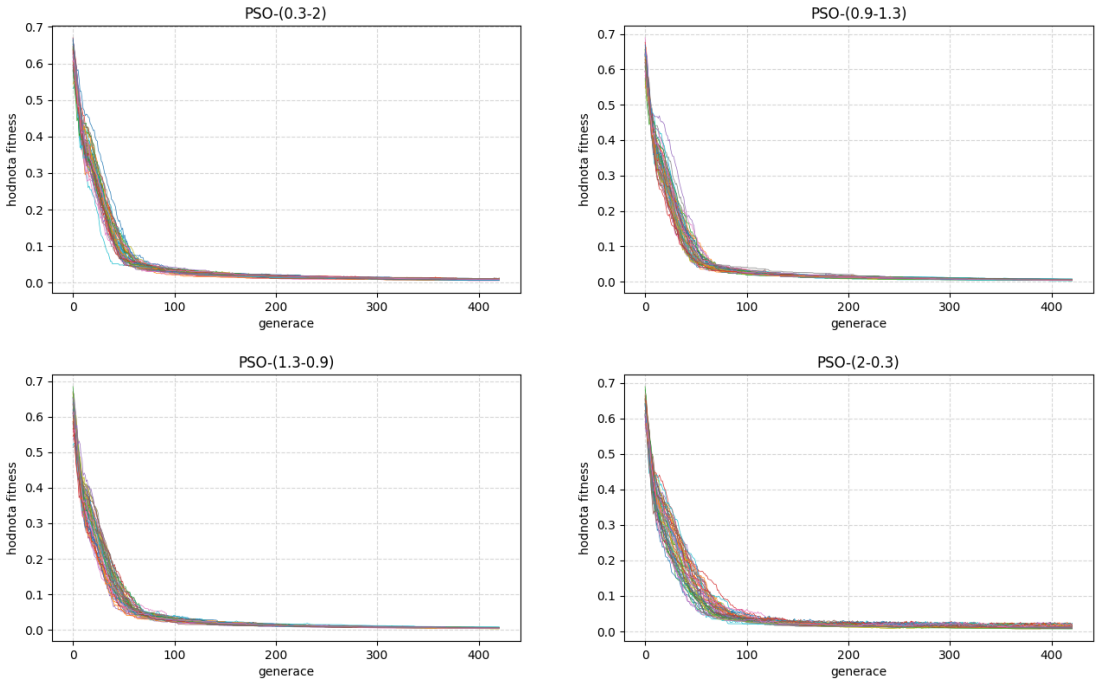
Grafy konvergenčních křivek jednotlivých experimentů lze vidět na obrázcích 5.1, 5.2, 5.3 a 5.4. Statistické zhodnocení běhů v podobě krabicových grafů lze vidět v 5.5. V každém grafu porovnánám pouze běhy jednoho konkrétního algoritmu, protože výsledky algoritmů se hodně liší. I v rámci některých algoritmů jsou výsledky velice odlišené, proto na grafech 5.6 je větší detail některých běhů algoritmů ABC a DE.



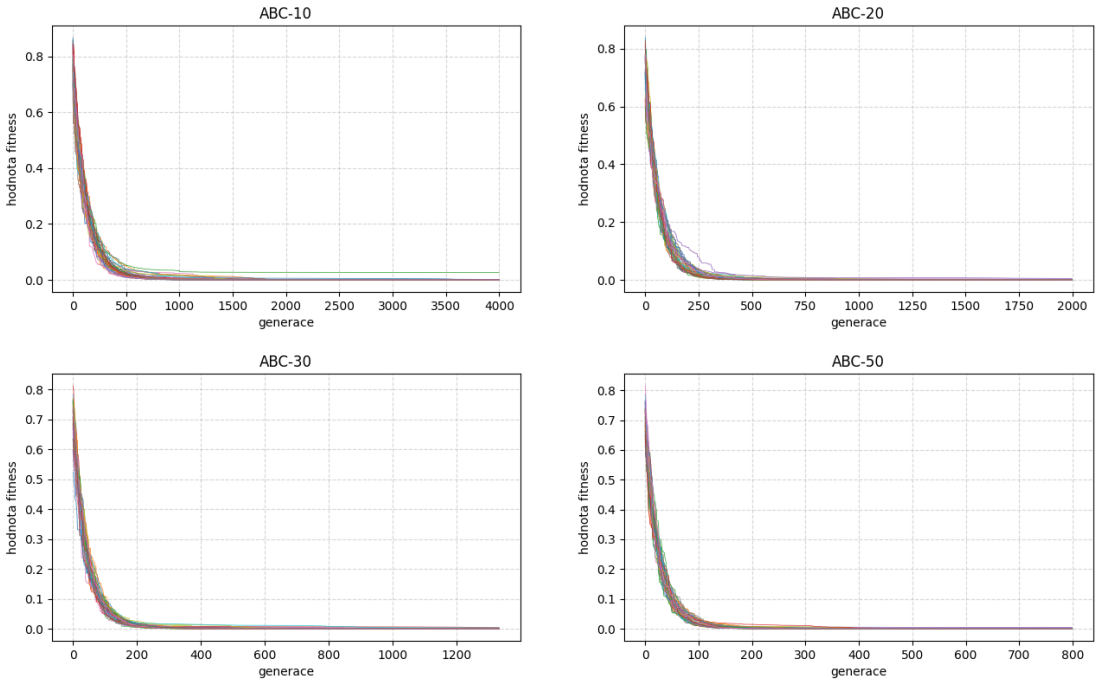
Obrázek 5.1: Průběh evoluce alogritmu ES při návrhu hradla CNOT



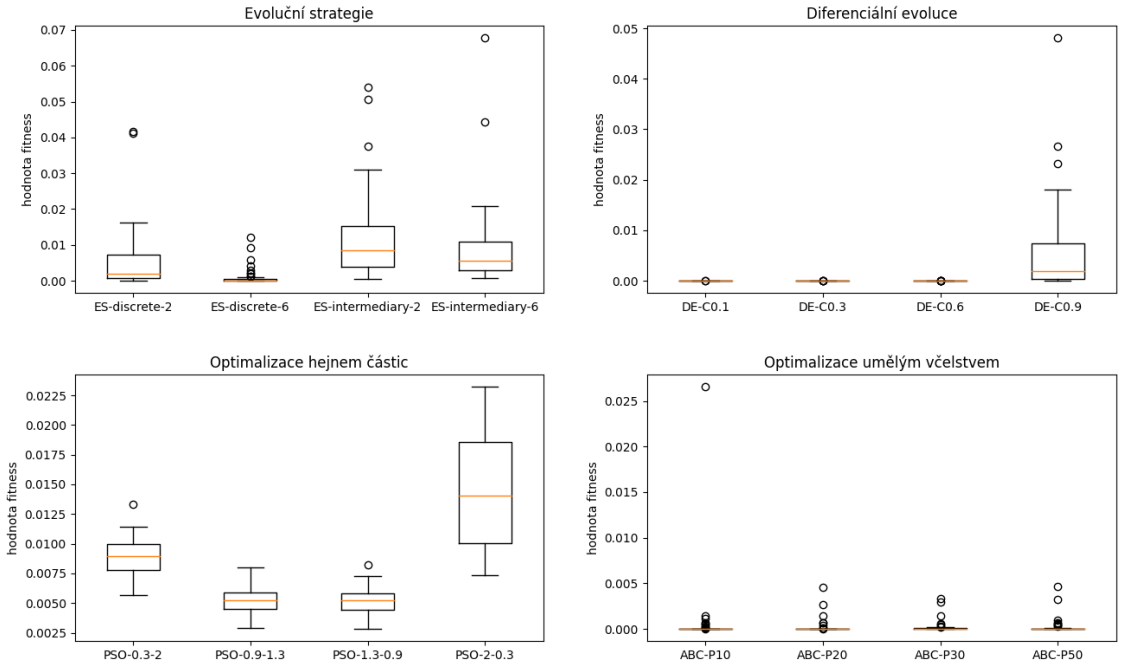
Obrázek 5.2: Průběh evoluce algoritmu DE při návrhu hradla CNOT



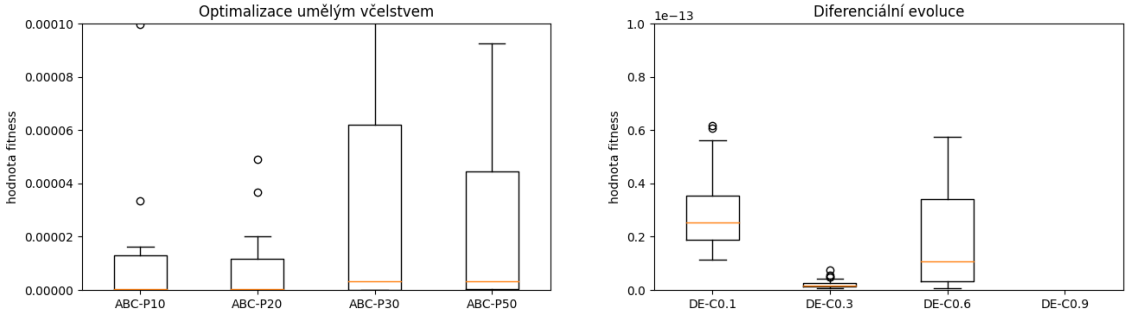
Obrázek 5.3: Průběh evoluce algoritmu PSO při návrhu hradla CNOT



Obrázek 5.4: Průběh evoluce algoritmu ABC při návrhu hradla CNOT



Obrázek 5.5: Statistické zhodnocení běhů návrhu hradla CNOT



Obrázek 5.6: Detailní statistické zhodnocení běhů algoritmů ABC a DE při návrhu hradla CNOT

5.3 Max3

V této úloze se budem snažit najít takové hradlo, které detekuje amplitudu vstupního stavu. Amplitudou se rozumí takový prvek vektoru, jehož absolutní hodnota je největší. Je tedy nejpravděpodobnější, že se qubit při měření zhroutí do tohohle stavu.

Například pro vstupní vektor popisující superpozici jednoho qubitu

$$\begin{pmatrix} 0.7 \\ 0.3 \end{pmatrix} \tag{5.5}$$

je očekávaný výstup

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \tag{5.6}$$

Budeme pracovat se stavem celkem 3 qubitů. Trénovací množina tentokrát obsahuje pouze jeden prvek. Vstupním vektor je superpozice

$$|i\rangle = \begin{pmatrix} 0.02570142522942053 - 0.050713214103136 24i \\ 0.10520139821724234 - 0.46655229712574786i \\ -0.38681941848462975 - 0.08005698740613776i \\ -0.1405785862248986 - 0.23330925418262133i \\ -0.24991868534155023 - 0.00950341931189315i \\ 0.2915219939642506 - 0.13895525502455086i \\ -0.41826900705091574 + 0.2636579871195689i \\ -0.231494698211255 + 0.26999574796325343i \end{pmatrix} \tag{5.7}$$

a výstupní vektor

$$|o\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \tag{5.8}$$

Jelikož tentokrát pracujeme se třemi qubity, velikost hledané matice se nám zvětší. V první úloze měla hledaná matice 16 prvků, v této jich má 64. Z tohoto důvodu bude podmínkou pro ukončení experimentů dosažení 120 000 vyhodnocení funkce fitness (3× více než v předchozí úloze).

5.3.1 Výsledky

I v této úloze si nejlépe vedla diferenciální evoluce. Nejlepší výsledek se podařilo získat pomocí nastavení `DE-C0.9`:

$$U = \begin{pmatrix} 0.315213 + 0.394437i & 0.847145 + 0.771695i & 1.185845 - 0.807382i & 0.395462 - 0.446970i \\ -0.663483 - 0.030603i & -0.060518 + 0.100803i & -0.790230 - 0.317202i & 1.112954 - 0.130012i \\ 0.045186 - 0.278917i & 0.091881 - 0.636365i & -0.149604 + 0.240602i & 0.339172 - 0.845545i \\ 0.818698 - 1.020025i & -0.554396 + 0.073301i & -0.278453 + 0.327571i & 1.134536 - 0.391589i \\ -1.078711 - 1.303117i & -0.481105 - 0.049793i & 0.907532 - 0.338547i & 0.566453 - 0.011908i \\ 0.598795 - 0.812902i & 0.757050 + 0.153425i & 0.855689 + 0.026721i & -0.007674 + 0.431053i \\ -1.677946 - 1.179904i & 0.511456 + 0.178230i & -0.573825 + 0.429993i & -0.155266 - 0.179667i \\ -0.290977 - 0.285377i & -0.825303 + 0.501536i & 0.788098 + 1.008834i & 0.312963 - 1.357508i \end{pmatrix} \tag{5.9}$$

$$\begin{pmatrix} -0.446582 - 0.428680i & 1.031731 - 0.602958i & 0.136749 - 0.413634i & 0.590792 + 0.124667i \\ 0.372860 - 0.649420i & 0.162207 - 0.404564i & -0.315289 - 0.079612i & 0.325664 + 0.313860i \\ 0.115669 - 0.200742i & -0.031122 + 0.331314i & 0.421187 + 0.210865i & -1.295548 - 1.458324i \\ 0.191399 + 0.882553i & 0.277144 - 0.841617i & 0.018678 - 0.337501i & 0.711084 - 1.225090i \\ -1.121206 - 0.525802i & -0.572361 - 0.078864i & -0.343881 + 0.469768i & -1.105059 - 0.944557i \\ -0.314843 - 0.366168i & 0.099887 - 0.227771i & 0.334296 - 0.329424i & 0.459804 - 0.698382i \\ 0.670581 - 0.651447i & 0.415347 - 0.151526i & -0.652321 - 1.008344i & -0.655966 - 0.418158i \\ -1.096143 - 0.105612i & -0.451344 - 0.567132i & -0.381072 - 0.127387i & -0.366455 - 0.441540i \end{pmatrix} \tag{5.10}$$

Můžeme říct, že se jedná o optimální řešení, protože fitness hodnota tohoto řešení je rovna 0 s přesností, kterou nabízí implementace datového typu double v c++. Po aplikaci matice na vstupní stav z trénovací množiny získáme následující vektor:

$$output = \begin{pmatrix} -4.53370191 \times 10^{-8} & -5.37824666 \times 10^{-7}i \\ 2.52200785 \times 10^{-7} & -3.27944263 \times 10^{-7}i \\ 3.90920540 \times 10^{-7} & -3.42433668 \times 10^{-7}i \\ -8.26158533 \times 10^{-8} & +1.53016845 \times 10^{-7}i \\ 2.97290029 \times 10^{-7} & +1.98834203 \times 10^{-7}i \\ 2.27477752 \times 10^{-7} & -1.96679380e \times 10^{-7}i \\ 9.99999768 \times 10^{-1} & +1.66394255e \times 10^{-7}i \\ 1.47583020 \times 10^{-7} & +6.32832004 \times 10^{-8}i \end{pmatrix} \tag{5.11}$$

který je velice blízko požadovanému výstupu popsaném v 5.8.

Tabulka 5.2: Výsledky návrhu hradla Max3

Algoritmus	Nastavení algoritmus	Nejlepší fitness	Průměrná fitness
ES	D-2	3.67522e-05	0.02059090
	D-6	0.251812	0.39214402
	I-2	3.32008e-07	0.00030114
	I-6	0.00206451	0.01543733
DE	C0.1	0.055149	0.07770562
	C0.3	0.102368	0.14270371
	C0.6	4.23774e-06	0.00177163
	C0.9	**0.0**	**8.12806758e-18**
PSO	(0.9-1.3)	2.48685e-05	7.55785917e-05
	(1.3-0.9)	3.19701e-05	6.672051875e-05
	(0.3-2)	0.000188512	0.00035234
	(2-0.3)	0.00258133	0.000207105
ABC	P10	2.42372e-05	5.41256523e-05
	P20	3.7392e-05	6.57254020e-05
	P30	2.52095e-05	6.25101729e-05
	P50	2.92587e-05	6.71039145e-05

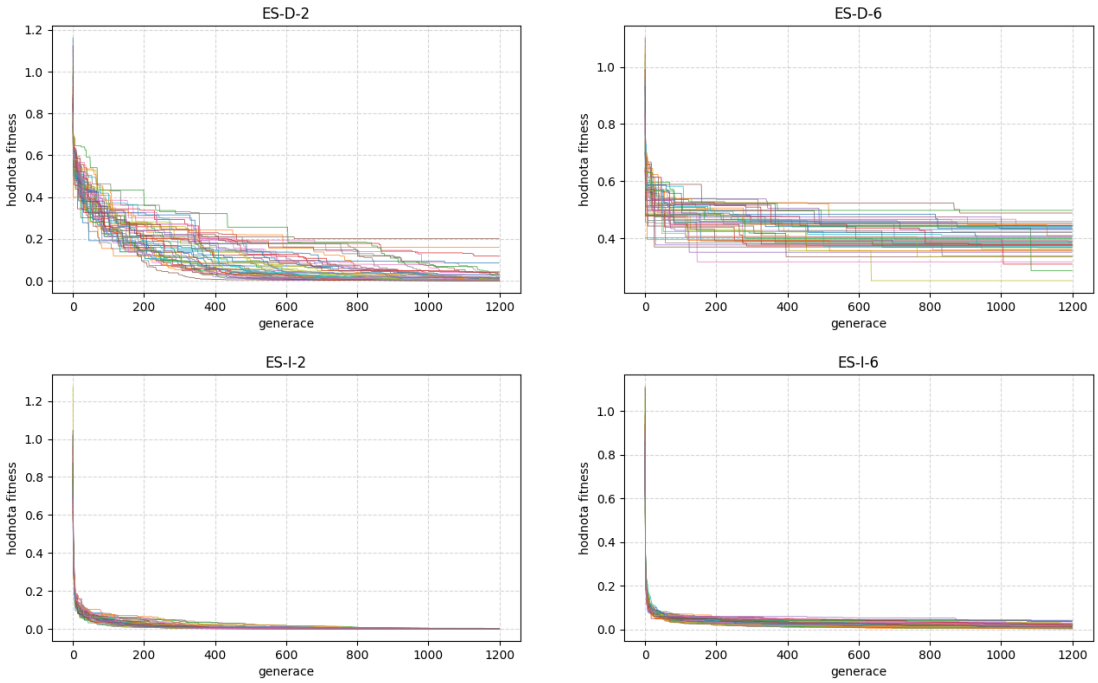
V této úloze si evoluční strategie oproti 5.8 vedla lépe s použitím intermediárního křížení a využitím pouze 2 rodičů pro generování potomků. S použitím nastavení `ES-D-60` docházelo v průběhu evoluce jen k velmi malému zlepšení. V předchozí úloze tohle nastavení naopak vedlo k nejlepším výsledkům.

Diferenciální evoluce si opět vedla nejlépe. Ve variantě `DE-C0.9` se dokonce podařilo najít optimalání řešení úlohy. S hodnotou koeficientu křížení blížící se nule ovšem úspěšnost algorimu rychle klesá. Správné nastavení tohoto parametru je pro tuto úlohu důležité.

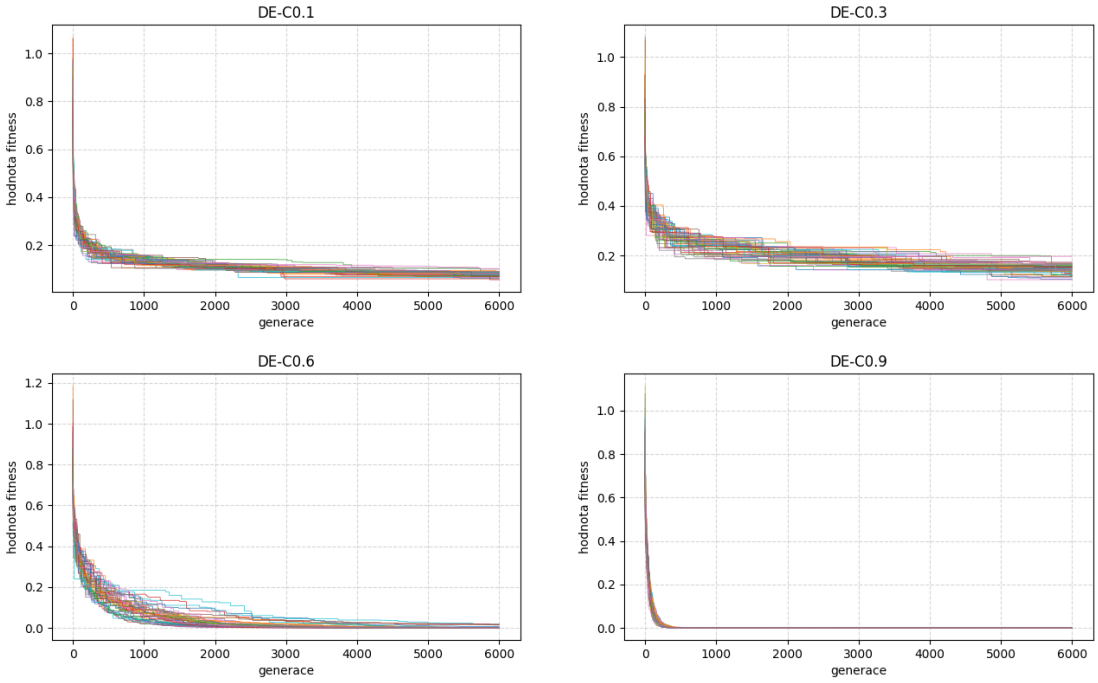
I u této úlohy si algoritmus optimalizace hejnem částic vedl lépe s nastavením, kde rozdíl mezi maximální mírou kognitivního učení a maximální mírou sociálního učení byl menší.

Algoritmus optimalizace umělým včelstvem opět ve všech nastaveních dosáhl dobrého výsledku. Tentokrát se ovšem tolik nepřiblížil diferenciální evoluci jako v první úloze. Konvergenční křivky se velice brzy přiblížili nule, zde se nicméně zlepšování výrazně zpomalilo.

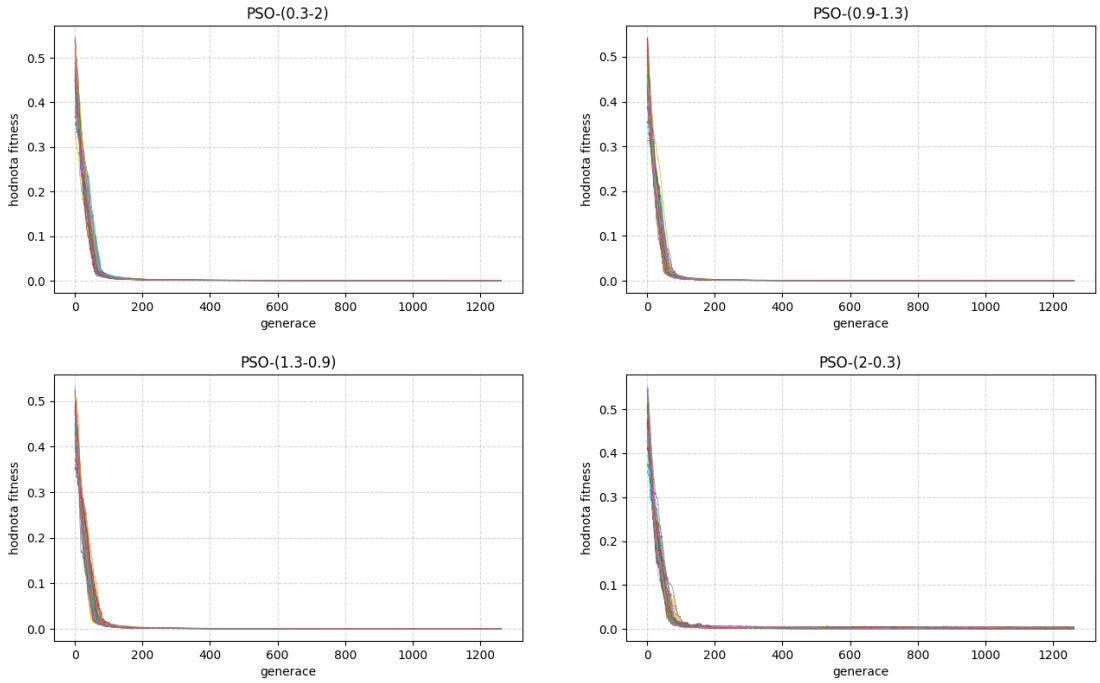
Grafy konvergenčních křivek jednotlivých experimentů lze vidět na obrázcích 5.7, 5.8, 5.9 a 5.10. Statistické zhodnocení běhů v podobě krabicových grafů lze vidět v 5.11. I v této úloze jsou v rámci některých algoritmů výsledky velice odlišené, proto na grafech 5.12 je větší detail některých běhů algoritmů ES, DE a PSO.



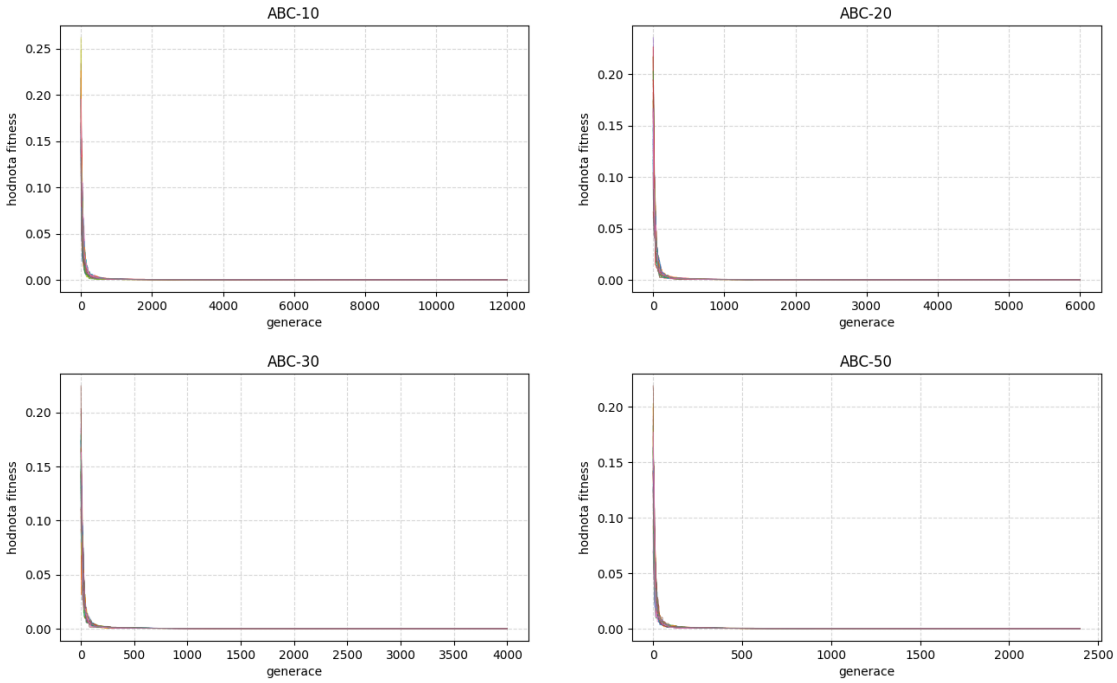
Obrázek 5.7: Průběh evoluce alogritmu ES při návrhu hradla MAX3



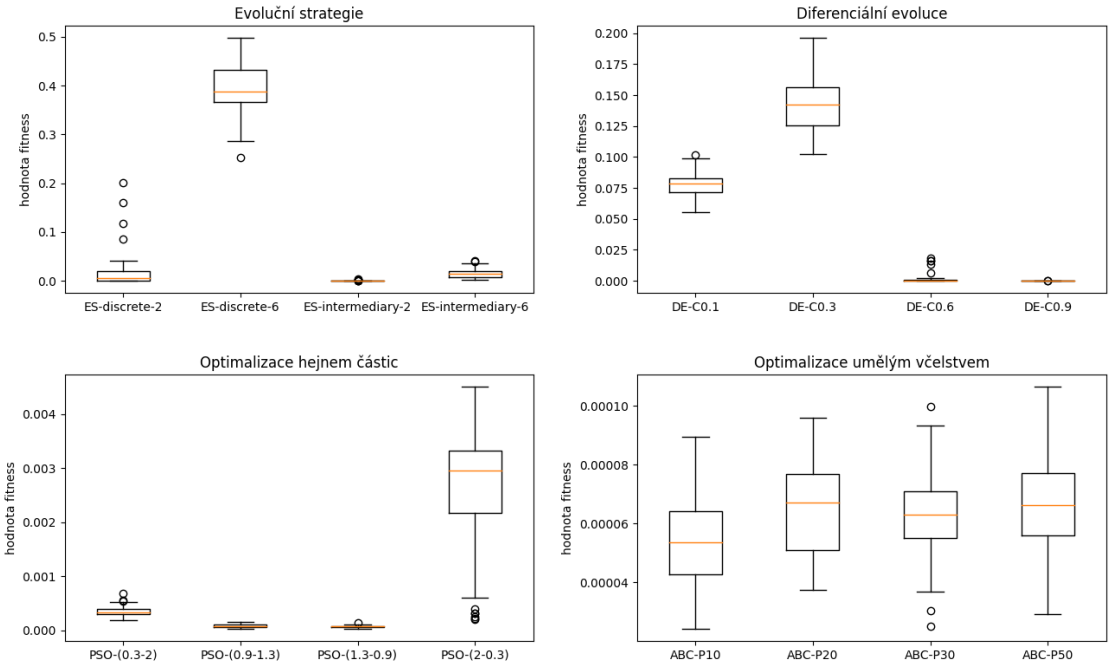
Obrázek 5.8: Průběh evoluce algoritmu DE při návrhu hradla MAX3



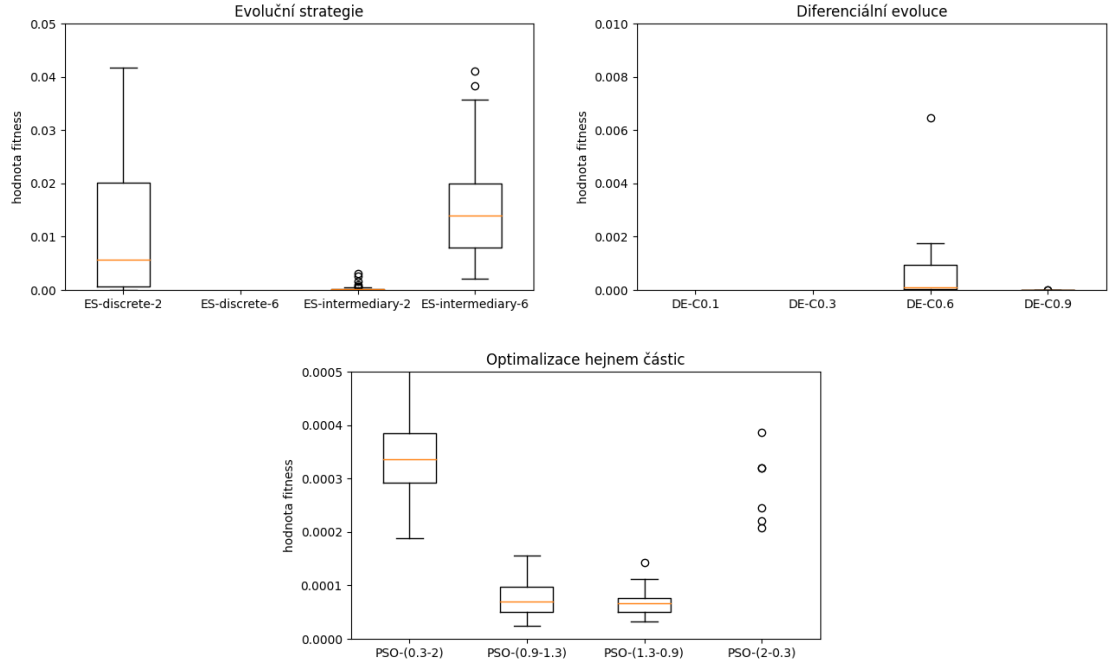
Obrázek 5.9: Průběh evoluce algoritmu PSO při návrhu hradla Max3



Obrázek 5.10: Průběh evoluce ABC při návrhu hradla Max3



Obrázek 5.11: Statistické zhodnocení běhů návrhu hradla MAX3



Obrázek 5.12: Detailní statistické zhodnocení běhů algoritmů ES, DE a PSO při návrhu hradla MAX3

5.4 Shrnutí výsledků experimentů

V obou úlohách se podařilo najít řešení s hodnotou fitness funkce blízké nule. V druhé úloze se dokonce podařilo najít řešení, jehož fitness je rovna nule.

Běhy algoritmů ES, DE a ABC trvaly rozumné množství času, oproti tomu PSO potřeboval k nalezení řešení několikrát větší množství času než ostatní algoritmy, řádově nižší desítky minut. Domnívám se, že tohle je způsobeno hlavně velkou režií při hledání, kteří jedinci spadají do okolí jiného jedince.

Z důvodu velké časové náročnosti a né příliš dobrých výsledků v první úloze shledávám algoritmus PSO jako nejméně vhodný. Navíc při počátečních pokusech na algoritmu trvalo dlouhou dobu najít takové nastavení, které by alespoň konvergovalo k lepšímu řešení. Při použití doporučených parametrů z literatury se fitness funkce naopak zhoršovala.

Naopak nejlepších výsledků dosahovaly algoritmy DE a ABC. Výhodou těchto algoritmů je navíc malé množství ladicích parametrů. Jasným favoritem je ovšem DE.

Kapitola 6

Závěr

V rámci této práce byl implementován systém pro návrh kvantových operátorů s využitím čtyř různých evolučních algoritmů.

Na návrhu dvou různých kvantových operátorů s využitím evolučních algoritmů bylo ukázáno, že použití přímé reprezentace kandidátních řešení může vést ke konstrukci kvantových operátorů přijatelné kvality. Zde je třeba zmínit, že se při návrhnu operátoru MAX3 povedlo dosáhnout řešení s hodnotou fitness funkce rovnou nule.

Sice bylo ukázáno, že pomocí príme reprezentace je možné navrhovat kvantové operátory, součástí práce ale není porovnání, zda je tato reprezentace horší než ostatní komplexnější reprezentace, které byly dřív pro tento účel použity. Abychom provedli průkazné srovnání, museli bychom implementovat všechny tyto reprezentace v jednom systému a provést experimenty se stejným nastavením. Takové porovnání by mohlo být pokračováním této práce.

V experimentech vycházel nejlépe algoritmus diferenciální evoluce, přestože byl v této práci implementován pouze v základní podobě. Dalším zkoumáním by mohlo být použití tohoto algoritmu v různých pokročilejších modifikací na návrh operátorů pro více než tři qubity.

Algoritmus optimalizace umělým včelstvem také nazbízí další možnosti modifkace, které by mohli vést ke zlepšení výsledků. Bylo by zajímavé tyto možnosti prozkoumat.

Při návrhnu implementace systému bylo myšleno na snadnou rozšiřitelnost. Provádět experimenty na různých operátorech je snadné, stačí přiložit soubor s trénovací množinou. Rozšíření systému o nové algoritmy lze provádět bez větších problémů.

Literatura

[1] Benioff, P. The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines. *Journal of Statistical Physics*. Květen 1980, sv. 22, s. 563–591.

[2] Bishnoi, B. *Quantum Computation*. arXiv, 2020. DOI: 10.48550/ARXIV.2006.02799. Dostupné z: https://arxiv.org/abs/2006.02799.

[3] Brabazon, A., O'Neill, M. a McGarraghy, S. *Natural Computing Algorithms*. 1st. Springer Publishing Company, Incorporated, 2015. ISBN 3662436302.

[4] Chou, K. S., Blumoff, J. Z., Wang, C. S., Reinhold, P. C., Axline, C. J. et al. Deterministic teleportation of a quantum gate between two logical qubits. *Nature*. Springer Science and Business Media LLC. sep 2018, sv. 561, č. 7723, s. 368–373. DOI: 10.1038/s41586-018-0470-y. Dostupné z: https://doi.org/10.1038%2Fs41586-018-0470-y.

[5] Clerc, M. The swarm and the queen: towards a deterministic and adaptive particle swarm optimization. In: *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*. 1999, sv. 3, s. 1951–1957 Vol. 3. DOI: 10.1109/CEC.1999.785513.

[6] Coutinho, S. C. Quantum Computing for Computer Scientists Noson S. Yanofsky and Mirco A. Mannucci, Cambridge University Press, 2008. *SIGACT News*. New York, NY, USA: Association for Computing Machinery. jan 2010, sv. 40, č. 4, s. 14–17. DOI: 10.1145/1711475.1711479. ISSN 0163-5700. Dostupné z: https://doi.org/10.1145/1711475.1711479.

[7] Deutsch, D. Quantum theory, the Church–Turing principle and the universal quantum computer. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*. 1985, sv. 400, s. 117 – 97.

[8] Eberhart a Shi, Y. Particle swarm optimization: developments, applications and resources. In: *Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No.01TH8546)*. 2001, sv. 1, s. 81–86 vol. 1. DOI: 10.1109/CEC.2001.934374.

[9] Eberhart, R. a Shi, Y. Comparing inertia weights and constriction factors in particle swarm optimization. In: *Proceedings of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No.00TH8512)*. 2000, sv. 1, s. 84–88 vol.1. DOI: 10.1109/CEC.2000.870279.

[10] Feynman, R. P. Simulating physics with computers. *International journal of theoretical physics*. World Scientific. 1982, sv. 21, 6/7, s. 467–488.

[11] Gheorghiu, V. Quantum++: A modern C++ quantum computing library. *PLOS ONE*. Public Library of Science. Prosinec 2018, sv. 13, č. 12, s. 1–27. Dostupné z: https://doi.org/10.1371/journal.pone.0208073.

[12] Hidary, J. D. Qubits, Operators and Measurement. In: *Quantum Computing: An Applied Approach*. Cham: Springer International Publishing, 2019, s. 17–36. ISBN 978-3-030-23922-0. Dostupné z: https://doi.org/10.1007/978-3-030-23922-0_3.

[13] Hidary, J. D. A Brief History of Quantum Computing. In: *Quantum Computing: An Applied Approach*. Cham: Springer International Publishing, 2021, s. 15–21. ISBN 978-3-030-83274-2.

[14] Hidary, J. D. Development Libraries for Quantum Computer Programming. In: *Quantum Computing: An Applied Approach*. Cham: Springer International Publishing, 2021, s. 67–86.

[15] Hidary, J. D. Teleportation, Superdense Coding and Bell's Inequality. In: *Quantum Computing: An Applied Approach*. Cham: Springer International Publishing, 2021, s. 87–99. ISBN 978-3-030-83274-2.

[16] Karaboga, D. a Basturk, B. A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (ABC) algorithm. *Journal of Global Optimization*. Listopad 2007, sv. 39, s. 459–471. DOI: 10.1007/s10898-007-9149-x.

[17] Nielsen, M. A. a Chuang, I. L. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. 10th. USA: Cambridge University Press, 2011. ISBN 1107002176.

[18] Paquet, U. a Engelbrecht, A. A new particle swarm optimiser for linearly constrained optimisation. In:. Leden 2003, sv. 1, s. 227–233. DOI: 10.1109/CEC.2003.1299579.

[19] Rivest, R. L., Shamir, A. a Adleman, L. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Commun. ACM*. New York, NY, USA: Association for Computing Machinery. feb 1978, sv. 21, č. 2, s. 120–126. DOI: 10.1145/359340.359342. ISSN 0001-0782. Dostupné z: https://doi.org/10.1145/359340.359342.

[20] Simon, D. *Evolutionary Optimization Algorithms*. Wiley, 2013. ISBN 9781118659502. Dostupné z: https://books.google.dz/books?id=gwUwIEPqk30C.

[21] Storn, R. a Price, K. Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *J. of Global Optimization*. USA: Kluwer Academic Publishers. dec 1997, sv. 11, č. 4, s. 341–359. DOI: 10.1023/A:1008202821328. ISSN 0925-5001. Dostupné z: https://doi.org/10.1023/A:1008202821328.

[22] Žufan, P. *Využití evolučních algoritmů v kvantovém počítání*. Brno, CZ, 2020. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.fit.vut.cz/study/thesis/22770/.

[23] Yanofsky, N. An Introduction to Quantum Computing. Září 2007.

[24] Žufan, P. a Bidlo, M. Advances in Evolutionary Optimization of Quantum Operators. *MENDEL*. Dec. 2021, sv. 27, č. 2, s. 12–22. DOI: 10.13164/mendel.2021.2.012. Dostupné z: https://mendel-journal.org/index.php/mendel/article/view/152.

Příloha A

Obsah přiloženého paměťového média

- **build** - složka se spustitelnými programy vytvořenými v této práci
- **trainingSet** - složka s trénovacími množinami pro návrh hradel
- **result** - složka, kam se zapisují výstupy běhu programu
- **scripts** - složka se scripty v jazyce Python (zpracování dat, simulátor kvantových výpočtů)
- **src** - zdrojové soubory systému v jazyce C++
- **tex** - zdrojové soubory pro vysázení technické zprávy
- **run.local.sh** - script, který obsahuje demostrační spuštění každého algoritmu na návrh hradla CNOT
- **Makefile** - Makefile obsahující příkazy pro překlad apod.
- **bp.pdf** - PDF soubor s technickou zprávou bakalářské práce

Příloha B

Manuál k systému

Hlavní část práce byla implementování v jazyce C++ a spouštěna v systému Linux.

B.1 Požadavky pro překlad

- `gcc` ve verzi alespoň 10.3
- `GNU make` ve verzi alespoň 4.3

B.2 Překlad

```
$ make build
```
Přeložené spustitelné soubory budou dostupné ve složce **build**.

B.3 Spuštění ukázkového běhu

```
$ ./run.local.sh
```

B.4 Spuštění běhů jednotlivých algoritmů

Pro každý algoritmus se po překladu aplikace vytvoří jeden spustitelný soubor ve složce **build**. Nastavení běhu evoluce se provádí přes parametry programu. Všechny algoritmy mají tyto společné parametry:

- `-env` - prostředí, ve kterém se běh spouští. Hodnota je buď `sc` - superpočítač nebo `local` - lokální počítač.
- `-operator` - udává, jaký soubor s trénovací množinou ze složky trainingData
- `-steps` - počet generací běhu
- `-popSize` - velikost celkové populace jedinců
- `-vectorSize` - velikost vektoru kandidátních řešení
- `-valueMin, -valueMax` - udávají interval, ze kterého se budou generovat náhodné hodnoty kandidátních jedinců na začátku evoluce

každý algoritmu má také svoje specifické parametry:

B.4.1 Parametry algoritmu ES

- `-variance` - rozptyl normálního rozložení, kterým se kontroluje velikost mutace
- `-generateChildren` - počet vygenerovaných potomků v každé generaci
- `-selectionStrategy` - způsob výběru jedinců (hodnoty plus nebo comma)
- `-crossingStrategy` - způsob křížení (hodnoty intermediary nebo discrete)
- `-parents` - počet rodičů použitých pro generování potomků
- `-adaptation` - použití samoadaptace (hodnoty yes nebo no)

B.4.2 Parametry algoritmu DE

- `-multiplier` - škálovací faktor
- `-crossoverRate` - koeficient křížení

B.4.3 Parametry algoritmu PSO

- `-maxVelocity` - maximální velikost vektoru rychlost
- `-influenceMax1` - maximální míra kognitivního učení
- `-influenceMax2` - maximální míra sociálního učení
- `-neighborhoodSize` - velikost okolí
- `-inertiaWeight` - váha setrvačnosti

B.4.4 Parametry algoritmu ABC

- `-stagnationLimit` - limit stagnace
- `-foragerPopulation` - velikost populace sběračů