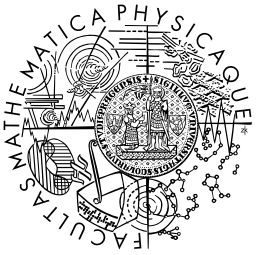


MATEMATICKO-FYZIKÁLNÍ FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Vilém Pech

4D Rubikova kostka: reprezentace a řešení metodami umělé inteligence

Katedra teoretikcké informatiky a matematické logiky

Vedoucí diplomové práce: Mgr. Vomlelová Marta, Ph.D.

Studijní program: Informatika

Studijní obor: Informatika se specializací Umělá inteligence

Praha 2023

Prohlašuji, že jsem tuto diplomovou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného titulu.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ....

Podpis autora

V prvé řadě velmi děkuji vedoucí práce paní Mgr. Martě Vomlelové, Ph.D. za veškerou ochotu, podporu, cenné rady i motivaci během psaní. Rovněž děkuji celé rodině, bez níž bych práci nikdy nedokončil.

Název práce: 4D Rubikova kostka: reprezentace a řešení metodami umělé inteligence

Autor: Vilém Pech

Katedra: Katedra teoretikcké informatiky a matematické logiky

Vedoucí diplomové práce: Mgr. Vomlelová Marta, Ph.D., Katedra teoretické informatiky a matematické logiky

Abstrakt: Hlavním cílem této práce je zobecnit řešič Rubikovy kostky Michaela Herdyho pomocí evolučních strategií pro čtyřrozměrnou Rubikovu kostku. Dále práce studuje jeho vlastnosti, snaží se ho vylepšit a porovnává výsledky s poznatky El-Souraniho a Borschbacha. Vysvětluje také zvolený způsob zobrazení čtyřrozměrného objektu do 3D a lehce naznačuje intuici pro snazší pochopení a představu. Součástí práce je také grafické prostředí, ve kterém lze vše názorně demonstrovat.

Klíčová slova: umělá inteligence, 4D Rubikova kostka, evoluční strategie

Title: Artificial Intelligence to Represent and Solve 4D Rubik's Cube

Author: Vilém Pech

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: Mgr. Vomlelová Marta, Ph.D., Department of Theoretical Computer Science and Mathematical Logic

Abstract: The main goal of this thesis is to generalize Micheal Herdy's algorithm for solving Rubik's cube using evolutionary strategies for a 4D Rubik's cube. The thesis then studies its characteristics, attempts to improve it, and compares the results with the findings of El-Sourani and Borschbach. The thesis explains the chosen projection of a four-dimensional object into 3D, and slightly suggests the intuition for a better understanding and idea. This work also includes a graphic environment in which everything can be demonstrated visually.

Keywords: artificial intelligence, 4D Rubik's cube, evolution strategies

Obsah

Úvod

Světoznámý hlavolam Rubikova kostka byl vynalezen maďarským profesorem Ernőem Rubikem v roce 1974. Původní název zněl Kouzelná kostka (v maďarštině „Bűvös kocka“), ten je často používaný dodnes například v Německu. Nedlouho po uvedení kostky na trh byla však kostka přejmenována po svém vynálezci a pod tímto názvem ji známe dodnes.

Počátky vícerozměrné Rubikovy kostky pochází už z roku 1988, kdy Melinda Green a Don Hatch vyvinuli první doloženou čtyřrozměrnou rubikovu kostku (samozřejmě jako počítačový program) a dali dohromady vzniknout relativně malé skupině nadšenců [1]. O prvenství se později v roce 2010 přihlásil ruský génius Andrey Astrelin, který údajně naprogramoval a složil Rubikův teserakt těsně před Melindou a Donem. Neměl však pro svoje tvrzení žádný důkaz kvůli ztrátě zdrojového kódu. Od té doby výrazně pomohl k vytvoření nejrůznějších hlavolamů. Rovněž drží další prvenství, mezi něž patří například složení sedmirozměrné Rubikovy kostky.

Řešiče Rubikovy kostky byly již několikrát předmětem výzkumu, ovšem vícerozměrným hlavolamům se doposud povětšinou věnovala jen hrstka nadšenců. Hlavolam Rubikovy kostky představuje prostor řádově 10^{19} různých pozic, oproti tomu jeho čtyřrozměrná verze jich má řádově až 10^{120} [1], což je pro představu mnohonásobně více než odhadovaný počet atomů ve vesmíru. Z toho důvodu může být obtížné se spoléhat na algoritmy hledající nejkratší řešení, jako je například algoritmus A*.

Proto je cílem této práce přinést do problematiky vícerozměrných hlavolamů další úhel pohledu přes evoluční strategie. Pokusíme se vytvořit řešič pro čtyřrozměrnou Rubikovu kostku na základě algoritmu M. Herdyho pro trojrozměrnou Rubikovu kostku vylepšeném El-Souranim a Borschbachem [2] a vše demonstrovat pomocí grafické aplikace. Poté porovnáme jeho chování s trojrozměrnou předlohou.

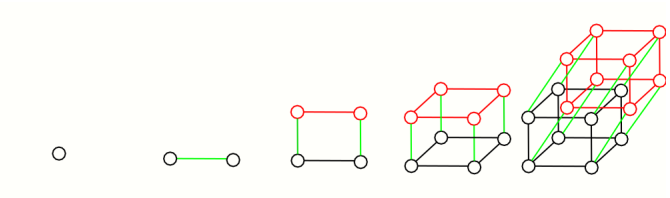
1. Zobecnění Rubikovy kostky

1.1 N-dimenzionální krychle

Pro pochopení Rubikova teseraktu je třeba si nejprve ujasnit, co znamená vícerozměrná krychle. Pro naše účely si vystačíme s krychlemi jednotkové délky hrany. Obecnou n-dimenzionální krychli budeme značit K_n a definujeme ji indukcí podle n.

Definice 1 (N-rozměrná krychle)**.** *N-rozměrná krychle je definovaná induktivně podle N.*

- *K_0 je bod s jedinou stěnou (rovněž K_0).*
- *K_{n+1} vznikne z K_n tak, že zkopírujeme všechny stěny a posuneme je ve směru nové dimenze o jednotkovou délku. Dále každou starou stěnu K_h spojíme s její kopií novou stěnou K_{h+1}.*



Obrázek 1.1: Tvorba K_n

Je jednoduché si všimnout, že každá h-rozměrná stěna krychle K_n je opět nějaká krychle K_h, kde $h \leq n$. Nyní se podíváme, kolik má n-rozměrná krychle m-rozměrných stěn. Jejich počet budeme značit $S_{m,n}$.

Věta 1. *Počet m-rozměrných stěn n-rozměrné krychle je dán předpisem*

$$S_{m,n} = 2^{n-m}\binom{n}{m}, \text{ kde } \binom{n}{m} = \begin{cases} \frac{n!}{m!\,(n-m)!} & \text{pro } m \leq n, \\ 0 & \text{jinak.} \end{cases}$$

Důkaz. Z definice 1 máme induktivní předpis pro $S_{m,n}$.

1. $\forall m > 0 : S_{m,0} = 0$
2. $S_{0,n} = 2^n$
3. $\forall m > 0, n > 0 : S_{m+1,n+1} = 2 \cdot S_{m+1,n} + S_{m,n}$

První bod je přímo z definice K_0, která má pouze jedinou stěnu, samu sebe. Druhý hovoří o 0-rozměrných stěnách (vrcholech) krychle. Na začátku má K_0 pouze jediný. Při indukčním kroku se vrcholy zduplikují a žádné nové nevzniknou. Podobně třetí vychází z indukce. Nechť máme K_n a z ní vytváříme K_{n+1}. Nejprve se zkopírují všechny staré $m+1$-rozměrné stěny, kterých bylo $S_{m+1,n}$. Pak každou starou stěnu K_m spojujeme s její kopií pomocí nové stěny K_{m+1}, tedy za každou stěnu K_m v původní krychli K_n vznikne nová stěna K_{m+1}.

Jelikož je každá hodnota $S_{m,n}$ tímto předpisem jednoznačně určena, jsou tyto tři rovnosti postačující podmínkou pro hledaný vzorec.

1. Bod má jedinou stěnu
   Nechť $m > 0$,
$$S_{m,0} = 2^{0-m}\binom{m}{0} = 2^{-m} \cdot 0 = 0$$

2. Počet vrcholů krychle
$$S_{0,n} = 2^{n-0}\binom{0}{n} = 2^n \cdot 1 = 2^n$$

3. Indukční krok
   Nechť $m > 0, n > 0$,
$$\begin{aligned}
S_{m+1,n+1} &= 2^{(n+1)-(m+1)}\binom{n+1}{m+1} = \\
&= 2^{n-m}\binom{n+1}{m+1} \cdot \frac{n+1}{n+1} = \\
&= 2^{n-m}\binom{n+1}{m+1} \cdot \left(\frac{n-m}{n+1} + \frac{m+1}{n+1}\right) = \\
&= 2^{n-m}\left[\frac{n-m}{n+1}\binom{n+1}{m+1} + \frac{m+1}{n+1}\binom{n+1}{m+1}\right] = \\
&= 2^{n-m}\left[\binom{n}{m+1} + \binom{n}{m}\right] = \\
&= 2^{n-m}\binom{n}{m+1} + 2^{n-m}\binom{n}{m} = \\
&= 2^{(n+1)-(m+1)}\binom{n}{m+1} + 2^{n-m}\binom{n}{m} = \\
&= 2 \cdot 2^{n-(m+1)}\binom{n}{m+1} + 2^{n-m}\binom{n}{m} = \\
&= 2 \cdot S_{m+1,n} + S_{m,n}
\end{aligned}$$

Uvedený výraz splňuje postačující podmínku pro počety stěn, tedy věta je dokázána.

□

Z explicitního vyjádření $S_{m,n}$ je vidět, že počet m-rozměrných stěn se s rostoucí dimenzí kostky exponenciálně zvyšuje. Růst můžeme pozorovat v tabulce několika prvních hodnot $S_{m,n}$ 1.1.

1.2 Rubikova kostka

Základem Rubikovy kostky je krychle rovnoměrně rozdělená na 9 stejných menších krychliček zvané dílky. Stěny dílků, jež tvoří povrch celé kostky mají danou barvu (samolepku). Na počátku ve složeném stavu má každá strana Rubikovy kostky stejnou barvu. Lze rozlišovat tři typy dílků podle toho, kolik mají samolepek: mechanizmus, střed, bok, roh. Střed je zajímavý tím, že jeho poloha

	m				
	0	1	2	3	4
$n=0$	1	0	0	0	0
$n=1$	2	1	0	0	0
$n=2$	4	4	1	0	0
$n=3$	8	12	6	1	0
$n=4$	16	32	24	8	1

Tabulka 1.1: Hodnoty $S_{m,n}$

vůči ostatním středům je neměnná. To znamená, že správná barva každé strany je v libovolném stavu kostky daná barvou středového dílku.

S hlavolamem lze provádět tahy tak, že se všech šest dílků na jedné straně otočí o 90° po nebo proti směru hodinových ručiček. Dohromady tak Rubikova kostka nabízí 12 možných tahů. Cílem je pomocí těchto tahů dostat hlavolam zpátky do složeného stavu.

1.3 Rubikův teserakt

Podle normální Rubikovy kostky nadefinujeme obecnou n-dimenzionální Rubikovu kostku. Základem je n-dimenzionální krychle K_n.Ta je v každé dimenzi rozdělená rovnoměrně na tři části, čímž je rozdělena na 3^n n-dimenzionálních dílků.

Každé $(n-1)$-rozměrné stěny dílků, které tvoří povrch celé kostky mají svoji barvu, tedy samolepku. Každá $S_{n-1,n}$ ze základní krychle tvoří jednu stranu. Samolepka je na dané straně, pokud stěna dílku, ke které je přiřazena, je součást této strany. Dílek je na dané straně, pokud je na téže straně jedna z jeho samolepek. Ve složeném stavu mají všechny samolepky na stejné straně shodnou barvu. Podobně jako u Rubikovy kostky budeme rozlišovat typy dílků podle počtu samolepek, dílek s t-samolepkami budeme nazývat *dílek typu t*.

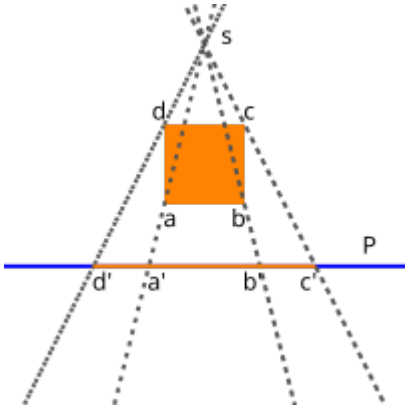
S každou stranou lze otáčet o 90° po nebo proti směru hodinových ručiček v rámci nadroviny společné strany, na níž leží. Hlavolam je složený, opět pokud jsou na každé straně stejné samolepky.

Rubikův teserakt je jen n-dimenzionální Rubikova kostka pro $n = 4$. Tvoří ho celkem 27 dílků, kde oproti třem dimenzím máme nový dílek typu 4. Typy dílků budeme v Rubikově teseraktu nazývat postupně od $t = 0$: *mechanizmus*, *centrum*, *střed*, *bok*, *roh*.

1.4 Zobrazení Rubikova teseraktu

Člověk je bytost žijící v trojrozměrném světě a je přirozeně obtížné představovat si čtyřrozměrné objekty přímo. Proto si Rubikův teserakt zobrazíme do 3D a budeme o něm nadále snáze uvažovat. Způsobů jak zobrazit objekt do prostoru nižší dimenze je mnoho, my ale použijeme centrální projekci. Budeme používat značení a znalosti ze skript prof. Mgr. Milana Hladíka, Ph.D. [3].

Definice 2 (Centrální projekce)**.** *Mějme vektorový prostor $\mathbb{R}^n$, v něm střed projekce $\boldsymbol{s}$ a nadrovinu $P = \{\boldsymbol{x} | \boldsymbol{x}^T \boldsymbol{p} = b\}$. Potom centrální projekce $C^{\boldsymbol{s}}_{\boldsymbol{p},b}$ je zobrazení*



Obrázek 1.2: Centrální projekce čtverce.

$\mathbb{R}^n \to P$ *takové, že bod* $\boldsymbol{a}$ *se zobrazí na bod* $\boldsymbol{a'}$*, kde* $\boldsymbol{a'}$ *je průsečík nadroviny* P *s přímkou procházející body* $\boldsymbol{s}$ *a* $\boldsymbol{a}$*.*

Tvrzení 2 (Vzorec centrální projekce)**.** *Nechť* $(\boldsymbol{s}-\boldsymbol{a})^T\boldsymbol{p} \neq 0$*, potom je centrální projekce* $C^{\boldsymbol{s}}_{\boldsymbol{p},b}(\boldsymbol{a})$ *daná předpisem*

$$C^{\boldsymbol{s}}_{\boldsymbol{p},b}(\boldsymbol{a}) = \lambda\boldsymbol{a} + (1-\lambda)\boldsymbol{s},$$

pro

$$\lambda = \frac{b - \boldsymbol{s}^T\boldsymbol{p}}{(\boldsymbol{a}-\boldsymbol{s})^T\boldsymbol{p}}.$$

Důkaz. Mějme tedy $C^{\boldsymbol{s}}_{\boldsymbol{p},b}(\boldsymbol{a})$ pro nějaké $\boldsymbol{s},\boldsymbol{p},\boldsymbol{a} \in \mathbb{R}^n$, $b \in \mathbb{R}$ a $(\boldsymbol{s}-\boldsymbol{a})^T\boldsymbol{p} \neq 0$. Přímku procházející body $\boldsymbol{s}$ a $\boldsymbol{a}$ si vyjádříme jako afinní kombinaci obou vektorů $L = \{\lambda\boldsymbol{a} + (1-\lambda)\boldsymbol{s} | \lambda \in \mathbb{R}\}$. Hledáme bod $\boldsymbol{a'}$, který leží na přímce i nadrovině $\boldsymbol{a'} \in P \cap L$.

Musí tedy platit

$$\lambda\boldsymbol{a} + (1-\lambda)\boldsymbol{s} = \boldsymbol{a'} \text{ a zároveň } \boldsymbol{a'}^T\boldsymbol{p} = b.$$

Dosadíme $\boldsymbol{a'}$ do druhé rovnice a vyjádříme λ

$$\begin{aligned}
\boldsymbol{a'}^T\boldsymbol{p} &= b \\
[\lambda\boldsymbol{a} + (1-\lambda)\boldsymbol{s}]^T\boldsymbol{p} &= b \\
\sum_{i=1}^{n}[\lambda\boldsymbol{a} + (1-\lambda)\boldsymbol{s}]_i\boldsymbol{p}_i &= b \\
\sum_{i=1}^{n}[\lambda(\boldsymbol{a}-\boldsymbol{s})_i\boldsymbol{p}_i + \boldsymbol{s}_i\boldsymbol{p}_i] &= b \\
\lambda\sum_{i=1}^{n}(\boldsymbol{a}-\boldsymbol{s})_i\boldsymbol{p}_i + \sum_{i=1}^{n}\boldsymbol{s}_i\boldsymbol{p}_i &= b \\
\lambda(\boldsymbol{a}-\boldsymbol{s})^T\boldsymbol{p} + \boldsymbol{s}^T\boldsymbol{p} &= b \\
\lambda &= \frac{b - \boldsymbol{s}^T\boldsymbol{p}}{(\boldsymbol{a}-\boldsymbol{s})^T\boldsymbol{p}},
\end{aligned}$$

což po dosazení do rovnice přímky přesně dává hledaný vzorec.

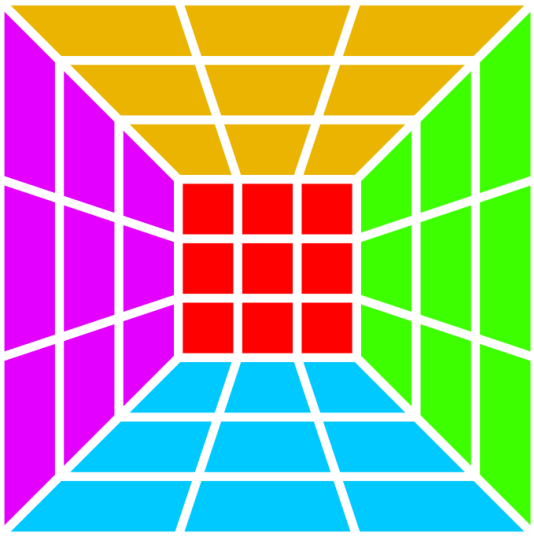
□

Poznámka. Samotná existence výrazu je podmíněna předpokladem $(\boldsymbol{s}-\boldsymbol{a})^T\boldsymbol{p} \neq 0$. Kromě toho má nutnost této nerovnosti i jednoduché geometrické vysvětlení.

Prvním případem je, když jeden ze dvou násobených vektorů je nulový vektor. Pro první to znamená, že $\boldsymbol{s} = \boldsymbol{a}$, což způsobí, že se máme pokoušet o přímku danou pouze jediným bodem. Podobně $\boldsymbol{p} = \boldsymbol{o}$ (pomocí $\boldsymbol{o}$ značíme nulový vektor) neurčuje nadrovinu, do níž projektujeme.

Může se ale také stát, že ani jeden nebude nulový, ale přesto je výsledný skalární součin roven nule. V takovém případě jsou vektory ortogonální a přímka L je s nadrovinou P rovnoběžná, tudíž spolu nemají žádný průsečík. Konkrétně toto platí pro všechna $\boldsymbol{a}$ v nadrovině rovnoběžné s P procházející $\boldsymbol{s}$.

Pro snazší pochopení zobrazíme tímto způsobem nejprve Rubikovu kostku do roviny. Při zobrazení Rubikovy kostky centrální projekcí by strana nejblíže ke středu projekce překrývala všechny ostatní. tak jako na obr. 1.4 strana $c'd'$ překrývá zbylé tři projektované strany, z toho důvodu tu jedinou vynecháme a dostaneme výsledný obraz 1.4, který si lze představit jako že se díváme dovnitř kostky skrze vynechanou stranu.



Obrázek 1.3: Rubikova kostka zobrazená centrální projekcí.

Není těžké si představit jednotlivé tahy kostky. Otočíme-li zadní stěnu, v projekci se otočí samolepky uprostřed a s nimi i sousední z bočních stran. Když otočíme levou, pravou, horní nebo dolní, může se zdát, že se najednou odnikud objevila nová barva. Pozorného čtenáře však toto nepřekvapí, neboť jde o samolepky ze strany vynechané během projekce.

Ještě je důležité si uvědomit, jak se projeví otáčení celé kostky o 90°. Pokud otočíme celou kostku proti směru hodinových ručiček ve směru nejmenší zobrazené strany (z pohledu projekce zadní), otočí se v témže směru celý objekt.Zajímavější je ale otočit kostku například vodorovně (podle horní strany).

Tutéž projekci použijeme pro zobrazení Rubikova teseraktu do 3D.

2. Evoluční algoritmy

Jedním z úspěšných heuristikcých prohledávání stavového prostoru jsou takzvané evoluční algoritmy. Jejich historii Jsou to algoritmy, které simulují nějakým způsobem vývoj podobně jako v přírodě podle Darwinovy teorie. První nápady v tomto odvětví prezentoval už na konci čtyřicátých let Alan Turing ve své práci Genetical or evolutionary search [4]. K dalšímu vývoji došlo až v šedesátých letech díky američanům Fogela, Owense a Walshe, jež dali základ Evolučnímu programování a později John Henry Holland stál za vznikem genetických algoritmů. Zároveň vznikaly i další odnože, mezi které patří například evoluční strategie nebo evoluční programování a všechny tyto směry, mající původ v přírodě, se vyvíjely nezávisle na sobě než se později celá tato oblast sloučila pod název evoluční algoritmy.

Evoluční algoritmy mají společnou kostru a různými jejími restrikcemi se pak dělí na čtyři hlavní směry [5]:

1. Genetické algoritmy
2. Genetické programování
3. Evoluční strategie
4. Evoluční programování

Algorithm 1 Obecný evoluční algoritmus

```
Inicializuj populaci náhodně
Ohodnoť populaci fitness funkcí
while není konec do
    Vyber rodiče
    Náhodně z rodičů vztvoř potomky
    Ohodnoť potomky fitness funkcí
    Vyber podle fitness novou populalci
end while
```

2.1 Evoluční strategie

Mezi nejjednodušší ze zmíněné čtveřice evolučních algoritmů patří evoluční strategie (ES). V šedesátých letech se inženýři potýkali s řadou multimodálních optimalizačních problémů. Řešení přinesly právě evoluční strategie [6]. Zálkadní myšlenka byla pouhá dvě pravidla:

1. Změň náhodně o kousek všechny parametry.
2. Pokud nové hodnoty objekt nezhoršují, nech je, jinak vrátit staré.

Takto jednoduchý na doméně nezávislý algoritmus, dnes známý jako (1 + 1)-ES, dokázal v té době překonat daleko složitější specializované algoritmy jako třeba gradientní sestup, který byl výpočetně náročný na tehdejší hardware, či

dokonce pro složitost problému nepoužitelný [6]. Evoluční strategie se tak dostaly do širšího povědomí.

Od ostatních evolučních algoritmů se odlišuje čtyřmi restrikcemi [5]:

1. Výběr pro reprodukci je nevychýlený.
2. Výběr do další generace je deterministický.
3. Měnící operátory jsou parametrizované.
4. Jedinci se skládají z kandidátského řešení a parametrů pro operátory.

2.1.1 Značení

Postupným zobecňováním EA se rovněž rozšiřovalo i značení. Základní tvar $(\mu\dagger\lambda)$ používal pouze dva parametry μ a λ a dále rozlišoval dvě varianty výběru jedinců do nové generace.

- μ: počet jedinců v populaci
- λ: počet nově narozených jedinců během jedné generace
- $+$: výběr do nové generace se provádí na nových jedincích spolu se současnou populací (výběr z $\lambda + \mu$ jedinců)
- ,: výběr do nové generace se provádí pouze na nových jedincích

Prvním rozšířením byl parametr ρ. Je závislý na operátoru rekombinace a značí počet rodičů, kteří se účastní jedné rekombinace.

Méně často se přidává další parametr κ jako zobecnění $\dagger$ notace a značí maximální stáří jedince. Každý jedinec se narodí se stářím 0 a v každé generaci se jeho stáří zvýší o jedna. Při výběru kandidátů do generace nové se vybírá pouze mezi jedinci současné populace, kteří nedosáhli maximální stáří κ, a novorozenci. Algoritmus se pak označuje $(\mu/\rho, \kappa, \lambda)$.

Algorithm 2 $(\mu/\rho,\kappa,\lambda)$-ES

```
Inicializuj populaci náhodně
while není konec do
    Zvyš stáří celé populace o 1
    for i ∈ {1, 2, ..., λ} do
        Vyber ρ rodičů k reprodukci uniformně náhodně
        Zkřiž rodiče
        Zmutuj nového jedince
    end for
    Zahoď všechny jedince staré κ
    Deterministicky vyber μ nových jedinců
end while
```

2.1.2 Operátory

Jelikož není v evolučních algoritmech pevně předepsaná reprezentace a jsou to sípš black-box principy, jejichž implementace závisí na problému, neexistuje ani žádná striktní kuchařka na volbu operátorů. K velmi oblíbenému poli výzkumu ES pařtila spojitá optimalizace, kde je obecně cílem najít vektor hodnot, jež minimalizují/maximalizují danou reálnou funkci. Hansen a kol. uvádí některé principy, jež se pro tento druh problému osvědčily [7].

Fitness

Fitness funkce slouží k ohodnocení jedinců, jak moc jsou silní, neboli jak dobré představujou řešení. Podoba silně závisí na daném problému, pro spojitou optimalizaci to typicky bývá hodnota funkce, kterou se snažíme maximalizovat.

Rekombinace

Operátor rekombinace vytváří nové jedince ze současné populace. Aby byla splněna první restrikce EA, vybere se typicky uniformě náhodně ρ rodičů a z nich se vytvoří nový potomek s cílem zužitkovat informaci z rodičů.

- Diskrétní (někdy též zvaná dominantní), značeno ρ_D, prochází vektor potomka a pro každou složku vybere z jeho rodičů uniformě náhodně jednoho, jehož hodnotu na dané pozici vybere.
- Středová (ρ_I) vytvoří aritmetický průměr rodičů, jejich těžiště.
- Vážená (ρ_W) je podobná středové s tím rozdílem, že počítá vážený průměr podle fitness funkce.

První zmíněná je velmi blízká křížení v genetických algoritmech a skutečně nikdo nebrání použití například n-bodové křížení, nicméně se tento postup v ES běžně nepoužívá a častěji se volí po vzoru selekce deterministický způsob. Jedním takovým je například vážená rekombinace s $\rho = \mu$ a $\kappa = 1$. Tím, že každý potomek je stejný, zůstává odpovědnost za dostatečně rozmanitou populaci na mutaci.

V počátcích se velmi často rekombinace nepoužívala. Někteří lidé proto nabyli dojmu, že se v evolučních strategiích klade větší důraz na mutaci než na rekombinaci, či dokonce že ES rekombinaci nepoužívá [5]. Naopak už Rechenberg ve své disertační práci uvedl, že rekombinace může výrazně zrychlit evoluci co do počtu generací [6].

Mutace

Obecně v evolučních algoritmech je cílem mutace změnit jedince jen trochu, aby nebyl výsledný efekt ve fitness funkci příliš drastický. Pokud bude mutace příliš slabá, hrozí degenerace populace a snadné uvíznutí v lokálním extrému, naopak přílišná změna jedince brání v konvergenci. V EA bývá většinou nejtěžší vyhodnotit fitness funkci, proto bychom chtěli něco jako optimální krok (learning rate), který bude na začátku pro špatnou populaci vyšší, aby se neztrácel čas zbytečně malými krůčky daleko od optima, ale zároveň aby jedinci blízko optima konvergovali a neoscilovali okolo něho. To vede k důležité myšlence ES, že operátor

mutace bude parametrizovaný a parametry budou adaptovány spolu s jedinci (3. a 4. bod restrikce ES).

Máme-li spojitou reálnou funkci, pak lze přičíst náhodnou veličinu z vícerozměrného normálního rozdělení $\mathcal{N}(\mathbf{0}, \boldsymbol{C})$ pro nějakou kovarianční matici $\boldsymbol{C}$ jakožto parametr mutace.

3. Řešič Rubikova teseraktu

3.1 Herdyho algoritmus

Pro normální trojrozměrnou Rubikovu kostku publikoval M. Herdy (1, 50) evoluční strategii a El-Sourani s Borschbachem algoritmus lehce vylepšili dalšími mutacemi a použitím větší populace [2]. Jeho hlavní myšlenka je používat speciální předpočítané mutace, které budou dobře fungovat s jednoduchou fitness funkcí. Algoritmus tak na jednu stranu nepotřebuje složité předpočítávání velikých tabulek pro počítání funkce fitness, ale na stranu druhou je pracnější správně nadefinovat mutace a výsledné řešení měřeno na počty tahů bývá i pro kostku jeden tah od složeného tvaru velmi dlouhé.

3.1.1 Operátory Herdyho algoritmu

Fitness

Fitness funkce počítá vážený součet tří q hodnot.

$$fitness(x) = q_1(x) + 4 \cdot q_2(x) + 6 \cdot q_3(x)$$

- q_1: počet samolepek s odlišnou barvou od středu, u kterého se nachází
- q_2: počet špatně umístěných boků
- q_3: počet špatně umístěných rohů

Váhy jsou záměrně nastavené tak, že všichni tři sčítanci nabývají hodnot 0 až 48, aby během evoluce nebyl kladen vyšší důraz na jiný typ dílků. Kostka je složená právě tehdy, když je její hodnota fitness rovna nule.

Mutace

První, co by člověka mohlo napadnout je použít jako mutaci jeden z legálních tahů na daný stav kostky. Tento přístup však se zvolenou fitness funkcí nefunguje moc dobře, protože samotný tah změní téměř třetinu kostky, což se silně projeví ve fitness jedince a povede k rychlému uvíznutí v lokálním minimu.

Z toho důvodu přišel Herdy se souborem předdefinovaných sekvencí tahů, které mají ve výsledku malý vliv na stav kostky a tím pádem i malý vliv na hodnotu fitness [2]. Mutace ve výsledku náhodně natočí celou kostku, vybere náhodně jednu sekvenci tahů a aplikuje ji na jedince. Konkrétně popužil tyto sekvence a jejich inverze:

- prohození třech rohů
- prohození třech boků
- prohození dvou boků a dvou rohů
- otočení dvou boků
- otočení dvou rohů

Selekce a rekombinace

Pro rekombinaci algoritmus vybere uniformě náhodně jednoho jedince z populace a zduplikuje ho. Borschbach a El-Sourani zkoušeli různé sofistikované metody výběru rodiče, ale ukázalo se, že oproti uniformnímu výběru nepřináší žádnou výhodu [2]. Do další generace se pak dostane μ nejlepších jedinců podle jejich fitness.

3.2 ES pro Rubikův teserakt

Nyní Herdyho algoritmus upravíme pro Rubikův teserakt. El-Sourani a Borschbach používali ve své implementaci pro reprezentaci jedince pole 3×3 pro každou ze šesti stran krychle s hodnotami určující barvu jednotlivých samolepek [2]. Pro teserakt by to znamenalo mít pro každou z osmi stran pole $3 \times 3 \times 3$ s hodnotami samolepek. V takovéto reprezentaci se pak musí popsat každý tah zvlášť, což vede se zvýšením dimenze kostky k nepřehlednému repetitivnímu kódu.

Místo toho jsme však zvolili reprezentaci bližší lineární algebře. Jedinec (Rubikův teserakt) je pole $3 \times 3 \times 3$ dílků a dílek je pole $3 \times 3 \times 3$ barev. Reprezentace tak odpovídá Rubikově teseraktu ležící středem v počátku kartézských souřadnic. Budeme-li pole číslovat od -1, znamenají indexy do pole vektory složených z nul a ± 1 v prostoru $\mathbb{R}^4$. Samotný dílek pak představuje podobnou ideu. Mějme ho v počátku souřadnic, pak každá samolepka odpovídá jednomu ($\pm$) jednotkovému vektoru, který na ni směřuje.

Takováto reprezentace umožňuje popisovat tahy pomocí vektorů a matic, naopak nevýhodou je zbytečné plýtvání pamětí, protože každý dílek disponuje polem 27 prvků, ale využívá z nich nanejvýš 4.

3.2.1 Úprava operátorů

Fitness

Oproti Rubikově kostce máme nově dílek typu 4. Fitness funkci proto obohatíme o novou hodnotu q_4, která bude po vzoru q_2 a q_3 počítat počet špatně umístěných d4. Bylo by možné též najít takové koeficienty hodnot q, aby všechny přispívaly do výsledné hodnoty fitness stejným dílem, avšak jak později uvidíme, koeficienty hodnot ztratí význam, a proto je vůbec nepoužijeme.

Mutace

Budeme mít pro každý typ dílku základní mutaci na prohození trojice, tj. prohození třech středů, prohození třech boků a prohození třech rohů a stejně tak otočení dvojice dílků jednoho typu.

Při otáčení dílků však nastávají komplikace. Nevystačíme si pouze s otáčením dvojic daných dílků, protože můžou nastat stavy kostky, které v Rubikově kostce nastat nemohly. V rubikově teseraktu je totiž možné pomocí legálních tahů otočit jediný bok či roh. Kvůli tomu musíme přidat další mutace na otočení těchto dílků. Použité sekvence tahů jsou převzaté od Mathologera [8].

Použití pouze zmíněných mutací však nestačí. Může se totiž například stát, že je třeba prohodit tři rohy, ale rohy takové, které jsou daleko od sebe a ne na po-

zicích, které prohazuje sekvence na prohazování třech rohů. Potom bychom uvízli v lokálním minimu, ze kterého se nedokážeme dostat. El-Sourani a Borschbach tento problém řešili hlavně větší rozmanitou populací. Bohužel v Rubikově teseraktu je takových pozic mnohem více, a tedy by bylo třeba o to větší populaci, což se v experimentech neukázalo jako efektivní řešení, protože udržovat populaci dostatečně velikou je výpočetně příliš náročné. Proto jsme zvolili cestu přidání mutací.

Pro každou základní mutaci jsme přidali další tak, aby existovaly mutace (natočení a sekvence tahů) pro každou n-tici dílků. Tím pádem pokud není fitness rovna nule, tak existuje mutace, která ji sníží. Jelikož jsme tak odstranili lokální minima, ve kterých bychom mohli uvíznout, nepotřebujeme ani obrovskou populaci a vystačíme si s jednoduchou (1+1)-ES.

3.2.2 Vylepšení algoritmu

Všechny operátory mutace pracují vždy jen s jedním typem dílků a také je buď prohazují, nebo otáčejí. Z toho důvodu je například zbytečné zkoušet mutace na prohazování rohů ve chvíli, kdy všechny rohy jedince jsou již správně umístěné. Stejně tak je zbytečné po otočení boku přepočítávat počet špatně umístěných dílků.

Proto je řešič rozdělen na šest částí: umisťování středů, umisťování boků, umisťování rohů, otáčení středů, otáčení boků a otáčení rohů. V každé z nich se používjí pouze patřičné mutace a relevantní hodnoty q. Pro otáčení dílků je q_1 rozděleno podle typu dílků na q_{1_t}, kde t značí typ dílků, jejichž samolepky se kontrolují. Díky tomu je výběr zlepšujících mutací i výpočet fitness efektivnější.

Jedinou výjimkou je případ, kdy je potřeba prohodit například dva středy a dva boky. V takovém případě nepomůže prohazování trojice středů a algoritmus by se zacyklil. Pokud nastane tato situace, restartuju algoritmus a pohnu jednou stranou o jeden tah. Toto řešení se ukázalo jako lepší, než přidávání další spoustu mutací, neboť řešič nejvíce zpomaluje právě čekání na zlepšující mutaci. To však lze provádět pouze v první ze šesti fází, kdy nevadí, že oním tahem rozhodíme dílky jiného typu.

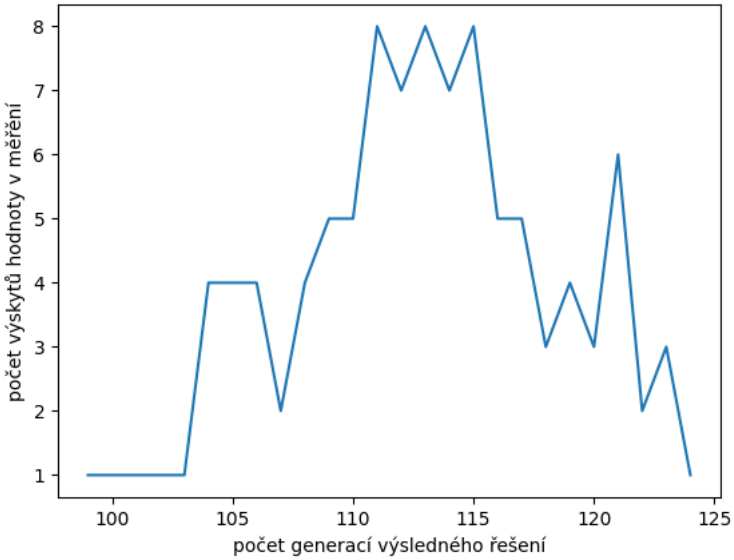
3.2.3 Porovnání obou algoritmů

Vytvořený algoritmus jsme nechali složit 100 náhodně zamíchaných kostek. Měření probíhalo na procesoru Intel(R) Core(TM) i5-9300H, 8 GiB RAM, operačním systému Manjaro Linux x86_64.

	Rubikův teserakt	Rubikova kostka
Průměrný čas (s)	33,38	5,04
Průměrný počet generací	113,15	18,20
Průměrný počet tahů	6122,16	234,60

Tabulka 3.1: Porovnání naměřených dat s výsledky El-Souraniho a Borschbacha.

Výsledky jsou zachyceny v tabulce 3.2.3 spolu s výsledky měřené El-souranim a Borschbachem [2] a v grafu 3.2.3 je zobrazena distribuce počtu potřebných generací na složení. Vidíme, že počet generací se pohyboval mezi 100 až 125



Obrázek 3.1: Distribuce počtu generací pro složení Rubikova teseraktu.

s průměrem 113,15, což je zhruba šestkrát více než u Rubikovy kostky. Stejný nárůst byl pozorován také pro celkový čas skládání.

Naproti tomu celkový počet tahů dosahuje až 26násobného nárůstu. To je zapříčiněno hlavně vyšší délkou sekvencí tahů používaných pro mutace. Zatímco mutace Rubikovy kostky nejsou delší než 14 tahů, sekvence pro otočení jednoho boku v Rubikově teseraktu jich má například 228.

4. Prostředí Rubikova teseraktu

Součástí této práce je interaktivní aplikace pro Rubikův teserakt, které se spolu s programátorskou dokumentací nachází v elektronické příloze.

4.1 Instalace a spuštění

Minimální požadavky:

- OS: Linux
- RAM: 100 MiB
- Hardware: monitor, myš, klávesnice
- OpenGL
- Python 3.10

Před spuštěním je třeba nainstalovat potřebné knihovny pomocí instalačního nástroje pip.

```
python -m pip install -r requirements.txt
```

Program se spustí pomocí hlavního souboru „maingui.py“, poté se otevře okono s prostředím.

```
python maingui.py
```

4.2 Ovládání

V prostředí se uživatel pohybuje pomocí myši. Držením levého tlačítka a pohybem myši posouvá v prostoru kamerou nahoru, dolů, doprava a doleva. Podobně držením pravého tlačítka a pohybem myši nahoru a dolů posunuje kameru dozadu, resp. dopředu.

Kliknutím levého tlačítka myši na samolepku středového dílku se otočí strana u této samolepky proti směru hodinových ručiček z pohledu od samolepky směrem k centru otáčené strany. Pokud se drží zmáčknutá klávesa ctrl, otáčí se ne jenom jedna strana, ale celý teserakt v témže směru. Zmáčknutá klávesa shift pak otáčí směr veškerého otáčení teseraktu.

Stisknutím klávesy R se spustí řešič na aktuální stav teseraktu. Jakmile doběhne, je uživatel dotázán na název souboru, kam se uloží nalezené řešení. Pro jeho provedení a vizualizaci nejprve načtěte soubor klávesou A a zadáním názvu. Pak ho lze spustit klávesou S. Rychlost animace lze v průběhu měnit posuvníkem. Rychlost se pohybuje od jednoho tahu za sekundu až po jeden tah za vykreslený snímek. Během animace lze s hlavolamem otáčet, je ale veliká šance, že se tím rozbije výsledný stav hlavolamu.

Řešič i zamíchání používají generátor náhodných čísel. Jeho seed se dá nastavit pomocí bílého vstupního okénka vepsáním celého čísla a potvrzením enterem.

Všechna tlačítka s jejich funkcemi jsou zachyceny v tabulce klávesových zkratek 4.2.

Klávesa	Funkce
levé tl. m.	tahy hlavolamu a posouvání kamery prostorem
pravé tl. m.	přibližování/oddalování kamery pohybem dolů/nahoru
prostřední tl. m.	otáčení hlavolamu
CTRL (držení)	změní tahy na otáčení celého hlavolamu
SHIFT (držení)	otočí směr veškerého otáčení
ENTER	potvrzení nastavení seedu náhodného generátoru
R	spuštění řešiče na aktuální stav teseraktu
A	načtení souboru s nalezeným řešením
S	spuštění animace načteného řešení
U	uloží stav teseraktu do souboru
K	načte stav teseraktu ze souboru
M	náhodně teserakt zamíchá
N	vytvoří nový teserakt ve složeném stavu
D	provede jeden další tah z načteného řešení

Tabulka 4.1: Klávesové zkratky a jejich funkce.

Závěr

Předmětem zkoumání byla vícerozměrná Rubikova kostka a úprava Herdyho řešiče pro Rubikův teserakt. V této práci se povedlo zobecnit krychli do více dimenzí, prozkoumali jsme vlastnosti jejich stěn. Díky tomu se nám podařilo zobecnit hlavolam Rubikova kostka. Pro uvažování o čtyřrozměrném hlavolamu jsme vysvětlili obecné zobrazení bodů do nadroviny pomocí centrální projekce, pro nějž jsem odvodily vzorec.

Následovala kapitola o evolučních algoritmech se zaměřením na evoluční strategie, kde jsme čtenáře seznámili s hlavními pojmy, značením a koncepty.

Ve třetí kapitole 3 jsme důkladně vysvětlili Herdyho evoluční strategii pro skládání Rubikovy kostky. Tu jsme upravili pro Rubikův teserakt a vylepšili na základě pozorovaném chování. Výsledný algoritmus jsme pak popsali, naimplementovali a porovnali jeho chování ve 3D a 4D. Řešič funguje stabilně skládá Rubikův teserakt v relativně rozumném čase.

Hlavní výhodou Herdyho algoritmu byly jednoduché operátory mutace a fitness, za které potom trpěla délka výsledného řešení. Toto se ukázalo ve 4D jako ještě větší problém. Zaprvé s novými dílky musely přijít nové mutace a to jak na prohazování, tak na otáčení. Mutace vylepšeného Herdyho algoritmu nezaručovaly v každé pozici existenci zlepšující mutaci, což ale díky velikosti Rubikovy kostky šlo dohnat větší populací a širším výběrem jedinců, takže uvíznutí pár jedinců v lokálním optimu nebyl problém. Totéž pro Rubikův teserakt však již nebylo možné, protože populace by musela být natolik veliká, že by řešič běžel neúnosně dlouho. Proto také vytvořený řešič používá místo obrovské populace mnoho předpočítaných mutací. Tím se však ale ztrácí jedna z jeho původních výhod, totiž jednoduché operátory bez gigantických předpočítaných tabulek.

Pokud bychom chtěli tentýž algoritmus upravit i pro vyšší dimenze, popsaný problém by jenom exponenciálně rostl. Navíc mutace jsou specifické pro daný hlavolam a ve vyšší dimenzi je třeba příliš mnoho úprav.

Hlavní cíle byly zobecnění algoritmu do 4D, jeho implementace a vizualizace. Přestože se při zkoumání nakonec algoritmus neukázal jako nejvhodnější, podařilo se všechny stanovené části této práce splnit.

4.3 Prostor pro další práci

Do budoucna je zde spousta míst pro vylepšování.

Po teoretické stránce by mohlo být zajímavé zkusit vyřešit problém s úpravami mutací při přechodu do vyšší dimenze. Například zkusit popsat obecné mutace pro n-rozměrnou Rubikovu kostku. Další nezodpovězenou otázkou zůstává, kolik je minimální počet potřebných tahů na složení libovolné validní pozice obecné n-rozměrné Rubikovy kostky.

Co se praktické části týče, algoritmus by se dal ještě více zefektivnit ať už rychlejší implementací nebo optimalizací operátorů mutace. Pro ty by pravděpodobně šlo najít sekvence tahů, které povedou ke stejnému výsledku za menší počet tahů.

Aplikace by mohla být dále rozšiřována o další funkce jako jsou třeba ukládání vlastních sekvencí nebo měnění parametrů projekce Rubikova teseraktu.

Seznam použité literatury

[1] Melinda Green. Superliminal, 2023.

[2] Nail El-Sourani and Markus Borschbach. Design and comparison of two evolutionary approaches for solving the rubik's cube. In Robert Schaefer, Carlos Cotta, Joanna Kołodziej, and Günter Rudolph, editors, *Parallel Problem Solving from Nature, PPSN XI*, pages 442–451, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[3] Milan Hladík. *Lineární algebra (nejen) Pro Informatiky*. Matfyzpress, 2019.

[4] A. E. Eiben and J. E. Smith. *Representation, Mutation, and Recombination*, pages 49–78. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.

[5] Günter Rudolph. *Evolutionary Strategies*, pages 673–698. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[6] Hans-Georg Beyer and Hans-Paul Schwefel. Evolution strategies - a comprehensive introduction. *Natural Computing*, 1:3–52, 03 2002.

[7] Nikolaus Hansen, Dirk V. Arnold, and Anne Auger. Evolution strategies. In *Handbook of Computational Intelligence*, 2015.

[8] Mathologer. Cracking the 4D Rubik's Cube with simple 3D tricks, 6 2016.

Seznam obrázků

Seznam tabulek

Seznam použitých zkratek

ES Evoluční strategie