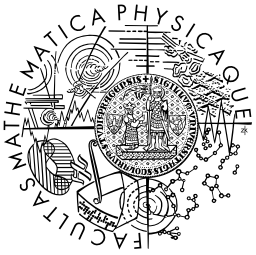

MATEMATICKO-FYZIKÁLNÍ FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Taisiia Starchenko

Generování latinských čtverců a ortogonalita

Katedra algebry

Vedoucí bakalářské práce:	prof. RNDr. Aleš Drápal, CSc., DSc.
Studijní program:	Matematika
Studijní obor:	Matematika pro informační technologie

Praha 2024

Prohlašuji, že jsem tuto bakalářskou 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

Na tomto místě bych chtěla vyjádřit své hluboké poděkování vedoucímu mé bakalářské práce, profesoru Aleši Drápalovi, za jeho pomoc, trpělivost a odborné vedení. Děkuji také za podporu a pomoc při překonávání obtížných situací, které se během práce vyskytly. Jeho cenné rady byly pro mě nepostradatelné a velmi si toho vážím. Rovněž bych ráda poděkovala za seminář, který pomohl nasměrovat vývoj této práce.

Rada bych ne v poslední řádě poděkovala své rodině za podporu. Bez jejich pomoci bych tuto práci nedokázala úspěšně dokončit.

Název práce: Generování latinských čtverců a ortogonalita

Autor: Taisiia Starchenko

Katedra: Katedra algebry

Vedoucí bakalářské práce: prof. RNDr. Aleš Drápal, CSc., DSc., katedra algebry

Abstrakt: Práce se zabývá otázkou transformace jednoho latinského čtverce na druhý pomocí lokálních změn a aplikuje tuto teorii pro návrh algoritmů hledání latinského čtverce ortogonálního danému. V první kapitole pracujeme s konceptem nevlastního latinského čtverce, rozšiřujícím dobře známý koncept latinského čtverce. Představujeme alternativní přístup k důkazu souvislosti grafu rozšířeného prostoru latinských čtverců. V druhé kapitole propojujeme algoritmus Jacobsona a Matthewse, který realizuje náhodné procházení tímto grafem, s metodou generování binární operace ortogonální vůči dvěma zadaným binárním operacím. Navrhujeme dva heuristické algoritmy pro hledání binární operace, která je ortogonální k danému latinskému čtverci a aproximuje latinský čtverec. Nakonec oba algoritmy porovnáváme ve výpočetních experimentech na testovací množině latinských čtverců řádů 7-10.

Klíčová slova: latinský čtverec, náhodný latinský čtverec, ortogonální operace, ortogonální latinský čtverec

Title: Generating latin squares and orthogonality

Author: Taisiia Starchenko

Department: Department of Algebra

Supervisor: prof. RNDr. Aleš Drápal, CSc., DSc., department of Algebra

Abstract: The thesis investigates how to transform one Latin square into another using local changes, and applies this approach to develop algorithms for finding Latin squares orthogonal to a given one. In the first chapter, we work with the concept of a improper Latin square, which extends the well-known concept of a Latin square. We introduce an alternative approach to proving the connectivity of the graph of the extended Latin square space. In the second chapter, we connect Jacobson and Matthews' algorithm, which implements random walk on the graph, with a method for generating a binary operation orthogonal to two given binary operations. We propose two heuristic algorithms for finding a binary operation orthogonal to a given Latin square and approximating a Latin square. Finally, we compare both algorithms in computational experiments on a test set of Latin squares of orders 7-10.

Keywords: latin square, random latin square, orthogonal operation, orthogonal latin square

Obsah

Úvod

Latinský čtverec řádu n je čtvercová matice typu $n \times n$ vyplněná n různými symboly tak, že se každý symbol objevuje v každém řádku a sloupci právě jednou. Zajímavou vlastností latinských čtverců je, že libovolná permutace jejich řádků, sloupců nebo symbolů vytváří opět latinský čtverec. Latinské čtverce vzniklé aplikací posloupnosti takových permutací se nazývají *izotopické*. Není těžko ověřit, že izotopie je relace ekvivalence, která rozděluje prostor všech latinských čtverců daného řádu na disjunktní třídy. Následující tabulka uvádí počet latinských čtverců řádu n (posloupnost A002860 v [1]) a počet tříd ekvivalence (posloupnost A040082 v [1]) tvořených relací ekvivalence na množině latinských čtverců daného řádu:

n	Počet latinských čtverců řádu n	Počet tříd ekvivalence
5	161280	2
6	812851200	22
7	61479419904000	564
8	108776032459082956800	1676267
9	$> 10^{27}$	115618721533
10	$> 10^{36}$	208904371354363006
11	$> 10^{47}$	$> 10^{25}$

Objevuje se otázka, zda existuje transformace latinského čtverce taková, že jenom pomocí posloupností těchto transformací jde převést libovolný latinský čtverec na libovolný jiný. Odpověď je kladná a najdeme ji v článku Jacobsona a Matthewse [2], musíme ale prostor latinských čtverců rozšířit o takzvané *nevlastní latinské čtverce*. Těmto transformacím budeme říkat *latinské záměny*.

Práce je rozdělená na dvě kapitoly. Cílem první kapitoly je dokázat existenci posloupnosti latinských záměn, která transformuje libovolný latinský čtverec na libovolný jiný. Tato část práce je založena na zpracování a reprodukci klíčových výsledků článku [2]. Avšak místo původního formalismu v práci používáme zcela odlišný a snad přehlednější přístup. V [2] autoři používají množinovou reprezentaci latinských čtverců a popisují vztah dvou řádků čtverce pomocí neorientovaných grafů, z čehož vycházejí v následujících důkazech. V našem přístupu latinské čtverce definujeme pomocí zobrazení do prvků volné abelovské grupy, které nazýváme abelovskou konfigurací (viz definice 1.1.12). Vztah mezi dvěma řádky popisujeme konfigurací zobecněného cyklu (viz definice 1.1.15).

V závěrečné sekci první kapitoly popíšeme algoritmus Jacobsona a Matthewse, tak jak je uveden v [2]. Příspěvek této práce tedy není v novém algoritmu, ale v odlišném důkazu skutečnosti, že tento algoritmus opravdu dovoluje cestou lokálních záměn přejít od jednoho latinského čtverce k jinému latinskému čtverci téhož řádu.

Druhá kapitola se zabývá ortogonálními latinskými čtverci. Využívají se v ní dosud nepublikovaný výsledek Falcona a Vojtěchovského [3], který dává do souvislosti latinské čtverce a binární operace ortogonální ke dvěma dalším binárním operacím na dané množině. Jde v zásadě o výpočetní experiment, jehož cílem bylo zjistit, zda takto nelze nacházet latinský čtverec ortogonální k danému la-

tinskému čtverci. Výpočetní výsledky naznačují, že tato metoda sama o sobě vede k úspěchu pouze u velmi malých řádů. Nicméně je možné, že ji půjde dále obohatit. Také metoda může mít smysl pokud cílem není nutně latinský čtverec, ale tabulka ortogonální operace, která se latinskému čtverci blíží.

V práci jsou popsány dva různé heuristické algoritmy prohledávání prostoru latinských čtverců použitých při hledání ortogonální operace. Ten úspěšnější z nich vychází ze známých metod umělé inteligence, které se souhrnně nazývají genetické algoritmy. Srovnání obou implementovaných heuristických algoritmů je provedeno na konci druhé kapitoly.

Algoritmy byly naprogramovány v jazyku Python a jsou uvedeny v příloze práce.

1. Generování latinských čtverců

1.1 Vlastní a nevlastní latinské čtverce

Latinské čtverce lze definovat vícero způsoby. Níže definujeme latinský čtverec jako speciální případ zobrazení z kartézského součinu dvou konečných množin $A \times B$ do prvků volné abelovské grupy $F(C)$, která je generována konečnou množinou C. Tedy $F(C)$ jsou všechny lineární kombinace prvků množiny C s koeficienty v $\mathbb{Z}$.

Definice 1.1.1. (Abelovská konfigurace). Nechť A, B, C jsou neprázdné konečné množiny. *Abelovskou konfigurací* nazveme parciální zobrazení $\mu\colon A \times B \to F(C)$ s definičním oborem $\mathrm{Dom}(\mu)$, pro něž platí:

1. $\forall c \in C\ \exists (a,b) \in \mathrm{Dom}(\mu)\colon \mu(a,b) = \sum_{c' \in C} \gamma_{c'} c'$, kde $\gamma_c \neq 0$,
2. $\forall a \in A\ \exists b \in B\colon (a,b) \in \mathrm{Dom}(\mu)$,
3. $\forall b \in B\ \exists a \in A\colon (a,b) \in \mathrm{Dom}(\mu)$.

To, že abelovská konfigurace $\mu\colon A \times B \to F(C)$ je zobrazení parciální znamená, že pro nějakou dvojici $(a,b) \in A \times B$ může být μ nedefinované. Budeme μ často interpretovat (resp. definovat) pomocí čtvercové tabulky o $|A| \times |B|$ políčkách s řádky indexovanými množinou A a sloupci indexovanými množinou B. Do políček této tabulky jsou vepsány prvky $F(C)$. Políčka mimo definiční obor necháme nevyplněná. Tabulku sestrojíme tak, že v pozici $(a,b) \in \mathrm{Dom}(\mu)$ promítneme symbol $\mu(a,b)$.

První podmínka kladená na abelovskou konfiguraci říká, že neexistuje vlastní podmnožina C, která by generovala $\mathrm{Im}(\mu)$. C budeme nazývat *množinou symbolů*, A *množinou řádků*, B *množinou sloupců*. Zbylé dvě podmínky říkají, že neexistují podmnožiny $A' \subseteq A$, $B' \subseteq B$ takové, že $A' \neq A$ nebo $B' \neq B$ a zúžení $\mu|_{A' \times B'}$ definuje stejné zobrazení.

Definice 1.1.2. (Vyváženost). Nechť $\mu\colon A \times B \to F(C)$ je abelovská konfigurace. Řekneme, že μ je *řádkově vyvážená*, pokud $\mathrm{Dom}(\mu) = A \times B$ a existují $c_1, c_2, \ldots, c_{|B|} \in C$, že pro každé $a \in A$ platí $\sum_{b \in B} \mu(a,b) = c_1 + c_2 + \ldots + c_{|B|}$. Řekneme, že μ je *sloupcově vyvážená*, pokud $\mathrm{Dom}(\mu) = A \times B$ a existují $c'_1, c'_2, \ldots, c'_{|A|} \in C$, že pro každé $b' \in B$ platí $\sum_{a' \in A} \mu(a',b') = c'_1 + c'_2 + \ldots + c'_{|A|}$.

Příklad 1.1.3. Ať $A = \{1,2,3\}$, $B = \{1,2,3,4\}$, $C = \{a,b,c,d\}$. Tabulkou předepíšeme abelovskou konfiguraci μ:

μ	1	2	3	4
1	$a+b$	c	$2d$	$-d$
2	a	b	d	c
3	d	b	c	a

Protože $\mathrm{Dom}(\mu) = A \times B$ a pro každé $r \in A$ součty $\sum_{l \in B} \mu(r,l)$ se rovnají $a + b + c + d$, μ je řádkově vyvážena konfigurace.

Definice 1.1.4. (Abelovský latinský čtverec). *Abelovský latinský čtverec* je řádkově a sloupcově vyvážená abelovská konfigurace $\mu\colon A\times B \to F(C)$, pro kterou platí $|A| = |B| = |C|$. Hodnotu $|A|$ budeme nazývat *řádem* abelovského latinského čtverce.

Příklad 1.1.5. Doplníme tabulku abelovské konfigurace z příkladu 1.1.3 o další řádek tak, aby výsledné zobrazení bylo abelovským latinským čtvercem, tedy aby bylo navíc sloupcově vyvážené. Ať $A = \{1,2,3,4\}$, $B = \{1,2,3,4\}$, $C = \{a,b,c,d\}$. Pak máme:

μ	1	2	3	4
1	$a+b$	c	$2d$	$-d$
2	a	b	d	c
3	d	b	c	a
4	$-a+c$	$a+d-b$	$-2d+b+a$	$b+2d$

Definice 1.1.6. (Vložení). Nechť $\mu_1\colon A_1\times B_1 \to F(C_1)$, $\mu_2\colon A_2\times B_2 \to F(C_2)$ jsou abelovské konfigurace, zobrazení $\alpha\colon A_1 \to A_2$, $\beta\colon B_1 \to B_2$, $\delta\colon C_1 \to C_2$ jsou injektivní. Trojice (α,β,δ) je *vložení* μ_1 do μ_2, pokud $\forall(a,b) \in \mathrm{Dom}(\mu_1)$: $\mu_1(a,b) = \sum\limits_{c\in C} \gamma_c c \implies \mu_2(\alpha(a),\beta(b)) = \sum\limits_{c\in C} \gamma_c \delta(c)$.

Máme-li abelovské konfigurace μ_1 a μ_2, řekneme, že μ_2 *obsahuje* μ_1, pokud existuje vložení μ_1 do μ_2. Když budou zobrazení (α,β,δ) patrná z kontextu, nebudeme je explicitně uvádět.

Definice 1.1.7. (Vlastní latinský čtverec). Nechť $\mu\colon A\times B \to F(C)$ je abelovský latinský čtverec řádu n. Řekneme, že μ je *(vlastní) latinský čtverec* řádu n, pokud $\mathrm{Im}(\mu) = C$.

Příklad 1.1.8. Podmínka kladená na obraz latinského čtverce zajišťuje, že se v každém políčku tabulky daného zobrazení objeví právě jeden symbol. Řádková a sloupcová vyváženost pak zajišťuje, že se každý symbol v každém řádku a sloupci objeví právě jednou. Ať $A = \{1,2,3,4\}$, $B = \{1,2,3,4\}$, $C = \{a,b,c,d\}$. Definujeme tabulkou vlastní latinský čtverec μ řádu 4:

μ:

a	b	c	d
b	c	d	a
c	d	a	b
d	a	b	c

Definice 1.1.9. (Konstrukční konfigurace). Nechť $\mu\colon A\times B \to F(C)$ je abelovská konfigurace. Řekneme, že μ je *konstrukční konfigurace*, jestliže
$|(a,b) \in \mathrm{Dom}(\mu); \mu(a,b) \notin C| \leq 1$ a pokud $\mu(a',b') \notin C$, pak $\mu(a',b') = c_1+c_2-c_3$, kde $c_1,c_2,c_3 \in C$ jsou po dvou různá.

Definice 1.1.10. Nechť $\mu\colon A\times B \to F(C)$ je konstrukční konfigurace a $\mu(a',b') = c_1+c_2-c_3 \notin C$, $(a',b') \in \mathrm{Dom}(\mu)$. Pozici (a',b') v tabulce zobrazení μ budeme nazývat *nevlastní políčko*, symbol c_3 *nevlastní symbol*, řádek a' a sloupec b' *nevlastní řádek a sloupec*.

Příklad 1.1.11. Konstrukční konfiguraci budeme zobrazovat jako obdélníkovou matici, kde políčka, ve kterých není zobrazení definované, zůstanou nevyplněná. Obvykle budeme každý řádek vložené konstrukční konfigurace označovat symbolem z množiny řádků, se kterým je daný řádek ztotožněn. Sloupce budeme označovat pouze tehdy, když je to podstatné v daném kontextu. Následující konfigurace s nevlastním symbolem b je obsažená v abelovské konfiguraci μ z příkladu 1.1.5:

$$\begin{array}{c} \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} a & b & & \\ d & & c & a \\ & a+d-b & & \end{pmatrix} \end{array}.$$

Definice 1.1.12. (Konstrukční latinský čtverec). *Konstrukční latinský čtverec* je abelovský latinský čtverec, který je zároveň konstrukční konfigurací.

Z definice vlastního latinského čtverce plyne, že se jedná o konstrukční latinský čtverec, pro který platí, že $|(a,b) \in A \times B; \mu(a,b) \notin C| = 0$.

Definice 1.1.13. (Nevlastní latinský čtverec). Konstrukční latinský čtverec $\mu\colon A \times B \to F(C)$, řádu n, $n \geq 3$, pro který existuje $(a,b) \in A \times B$, že $\mu(a,b) \notin C$ nazveme *nevlastní latinský čtverec* řádu n.

Protože je nevlastní latinský čtverec sloupcově a řádkově vyvážený a všechna políčka, kromě nevlastního, jsou vyplněná právě jediným symbolem, tak se každý symbol vyskytne právě jednou, až na nevlastní symbol, který se v nevlastním řádku a sloupci vyskytne dvakrát.

Jinými slovy, nevlastní latinský čtverec se od vlastního liší tím, že dovoluje v jednom políčku výskyt záporného symbolu a dvou kladných. V nevlastním řádku se záporný symbol vyskytuje dvakrát jako kladný v ostatních políčcích. Každý jiný symbol se v nevlastním řádku vyskytuje právě jednou. Podobně to platí i v nevlastním sloupci.

Příklad 1.1.14. Definujeme vlastní latinský čtverec $\mu_1\colon A \times B \to F(C)$ a nevlastní latinský čtverec $\mu_2\colon A \times B \to F(C)$, kde $A = B = C = \{1,2,3\}$:

μ_1:

1	2	3
3	1	2
2	3	1

μ_2:

1	2	3
3+2-1	1	1
1	3	2

Definice 1.1.15. (Cyklická konfigurace). Nechť $\mu\colon A \times B \to F(C)$ je konstrukční konfigurace, $A = \{r,r'\}$. Řekneme, že μ je *cyklická konfigurace* nebo $r - r'$ *cyklus*, pokud:

1. μ je řádkově vyvážená.
2. neexistuje $B' \subsetneq B$, že $\mu|_{A \times B'}$ je řádkově vyvážená konstrukční konfigurace.

Druhá podmínka v definici cyklické konfigurace říká, že vynecháním libovolného množství sloupců z matice konstrukční konfigurace vznikne zobrazení, které už není řádkově vyvážené.

Lemma 1.1.16. *Nechť* $\mu\colon A\times B \to F(C)$ *je* $r-r'$ *cyklus. Pokud* $\mathrm{Dom}(\mu)$ *obsahuje nevlastní políčko, ať je v pozici* $(r,c) \in A\times B$. *Pak* μ *je jednoho z těchto tvarů:*

$$\begin{matrix} r \\ r' \end{matrix}\begin{pmatrix} a_0 & a_1 & \dots & a_{k-1} & a_k \\ a_1 & a_2 & \dots & a_k & a_0 \end{pmatrix} \tag{C0}$$

$$\begin{matrix} & & & & & & c & & & & & \\ r \\ r' \end{matrix}\begin{pmatrix} b_0 & b_1 & \dots & s & x+y-s & a_0 & \dots & a_f & s & c_0 & \dots & c_k \\ y & b_0 & \dots & b_m & a_0 & a_1 & \dots & s & c_0 & c_1 & \dots & x \end{pmatrix}, a_0 \neq s \tag{C1}$$

$$\begin{matrix} & & & & & c & & & & \\ r \\ r' \end{matrix}\begin{pmatrix} b_0 & b_1 & \dots & s & x+y-s & s & a_0 & \dots & a_k \\ y & b_0 & \dots & b_m & s & a_0 & a_1 & \dots & x \end{pmatrix} \tag{C2}$$

$$\begin{matrix} & & & & c & & & & \\ r \\ r' \end{matrix}\begin{pmatrix} x+y-s & a_0 & \dots & a_k & b_0 & b_1 & \dots & s \\ a_0 & a_1 & \dots & x & y & b_0 & \dots & b_m \end{pmatrix}, a_0 \neq s \tag{C3}$$

Důkaz. Předpokládejme, že $\mathrm{Dom}(\mu)$ neobsahuje nevlastní políčko. Zvolíme libovolně $a_0 \in C$. Pak existuje $(r,\gamma_0) \in \mathrm{Dom}(\mu)$, že $\mu(r,\gamma_0) = a_0$. Zkonstruujeme posloupnost sloupců $\gamma_0, \gamma_1, \dots, \gamma_q \in B$ tak, že pro každé $i \in \{1,2,\dots,q\}$ platí $\mu(r,\gamma_i) = \mu(r',\gamma_{i-1})$. Konstrukci ukončíme v okamžiku, kdy nalezneme sloupec γ_q takový, že $\mu(r',\gamma_q) = a_0$ a položíme $k = q$. Taková posloupnost existuje, neboť každý řádek tabulky zobrazení μ je vyplněn konečným množstvím symbolů. Zjevně $\sum\limits_{0\le i\le k} \mu(r',\gamma_i) = \sum\limits_{0\le i\le k} \mu(r,\gamma_i)$, tedy množina $M = \{\gamma_0,\gamma_1,\dots,\gamma_q\}$ poskytuje řádkově vyváženou konstrukční konfiguraci. Dokážeme teď, že žádná vlastní podmnožina M neposkytuje řádkově vyváženou konfiguraci. Sporem, nechť existuje $M' \subsetneq M$, $M' \neq \emptyset$, pro niž platí $\mu|_{A\times M'}$ je řádkově vyvážená. Ať $i' \in \{1,2,\dots,k-1\}$ je nejmenší index takový, že $\gamma_{i'} \in M'$ a $\gamma_{i'+1} \notin M'$. Pak ale $\sum\limits_{0\le i\le k} \mu(r',\gamma_i) - \sum\limits_{0\le i\le k} \mu(r,\gamma_i) = \mu(r',\gamma_{i'})$, tj. symbol $\mu(r',\gamma_{i'})$ se vyskytne v řádku r' ale nikdy v r, tedy $M' = M$. Pak μ je tvaru (C0).

Předpokládejme, že $\mathrm{Dom}(\mu)$ obsahuje nevlastní políčko. Obdobně zkonstruujeme posloupnost sloupců $c{=}\gamma_0,\gamma_1,\dots,\gamma_q \in B$ tak, že pro každé $i \in \{1,2,\dots,q\}$ platí $\mu(r,\gamma_i) = \mu(r',\gamma_{i-1})$. Pokud $\mu(r',\gamma_q) \in \{x,y\}$, ukončíme konstrukci a položíme $k = q$. Budeme předpokládat, že $\mu(r',\gamma_k) = x$. Z vyváženosti μ musí existovat sloupec $\delta_0 \in B$, že $\mu(r',\delta_0) = y$. Zkonstruujeme posloupnost $\delta_0,\delta_1,\dots,\delta_\ell \in B$ tak, že $\mu(r',\delta_j) = \mu(r,\delta_{j-1}) \neq s$, $j \in \{1,2,\dots,\ell\}$. Pokud $\mu(r,\delta_\ell) = s$, pak položíme $m = \ell$ a konstrukci ukončíme. Z konstrukce posloupností vyplývá, že $\sum\limits_{0\le i\le k} \mu(r',\gamma_i) - \sum\limits_{0\le i\le k} \mu(r,\gamma_i) = s - y$ a $\sum\limits_{0\le i\le m} \mu(r',\delta_i) - \sum\limits_{0\le i\le m} \mu(r,\delta_i) = y - s$, tedy $\sum\limits_{0\le i\le k} \mu(r',\gamma_i) + \sum\limits_{0\le i\le m} \mu(r',\delta_i) = \sum\limits_{0\le i\le k} \mu(r,\gamma_i) + \sum\limits_{0\le i\le m} \mu(r,\delta_i)$. Označme $M = \{\gamma_0,\gamma_1,\dots,\gamma_k\}$, $T = \{\delta_0,\delta_1,\dots,\delta_m\}$. Množina $M \cup T$ poskytuje řádkově vyváženou konstrukční konfiguraci. Chceme ověřit, že neexistuje $M' \subseteq M$, $T' \subseteq T$, že $M' \cup T'$ je nevlastní podmnožina $M \cup T$ a $\mu_{A\times(M'\cup T')}$ je řádkově vyvážená

konfigurace. Sporem, předpokládejme $\emptyset \neq M' \cup T' \subsetneq M \cup T$ poskytuje řádkově vyváženou konfiguraci a nechť $M' \subsetneq M$. Pokud $M' = \emptyset$, pak $T' \neq \emptyset$ a $\delta_0 \in T'$. Pak se ale symbol $y = \mu(r', \delta_0)$ vyskytne v řádku r' ale nikdy v r, tedy $M' \neq \emptyset$. Ať $i' \in \{1, 2, \ldots, k-1\}$ je nejmenší index takový, že $\gamma_{i'} \in M'$ a $\gamma_{i'+1} \notin M'$. Neexistuje i $\delta_z \in T'$, $z \in \{1, 2, \ldots, m\}$, že $\mu(r, \delta_z) = \mu(r', \gamma_{i'})$, protože pak bychom mohli položit $\gamma_{i'+1} = \delta_z$. Pak se ale symbol $\mu(r', \gamma_{i'})$ vyskytne v řádku r' ale nikdy v r, tedy $M' = M$. Obdobnými úvahami odvodíme $T' = T$, tedy $M' \cup T' = M \cup T$.

Pokud $s \notin \{\mu(r', \gamma_0), \mu(r', \gamma_1), \ldots, \mu(r', \gamma_k)\}$ (tj. s se v řádku r' nevyskytuje), pak cyklus má tvar (C3). Jinak, rozlišíme případy $k = 0$ a $k > 0$. Je-li $k = 0$, pak je cyklus μ ve tvaru (C2), jinak je ve tvaru (C1).

□

Důsledek 1.1.17. *Ať μ je konstrukční latinský čtverec řádu n, μ' je cyklická konfigurace obsažená v μ. Pak μ' je v jednom ze tvarů (C0), (C1), (C2) nebo (C3). Navíc je-li μ' ve tvaru (C1), pak $\min(m, k+f+1) \leq (n-1)/2$, je-li μ' ve tvaru (C2), pak $k+m+1 \leq n-3$, je-li μ' ve tvaru (C3), pak $\min(m+1, k+1) \leq (n-1)/2$.*

Důkaz. První část tvrzení plyne z lemmatu 1.1.16. Řád matice konstrukční konfigurace μ' je omezen řádem latinského čtverce μ, ve kterém je tato konfigurace obsažena. Nahlédneme, že ve tvaru (C1) platí nerovnost $m+k+f+1 \leq n-1$, z čehož plyne, že $\min(m, k+f+1) \leq (n-1)/2$. Pokud je μ' ve tvaru (C2), platí $m+k+2 \leq n-1$, tudíž $\min(m+1, k+1) \leq (n-1)/2$. Je-li μ' ve tvaru (C3), pak do $\mathrm{Dom}(\mu')$ nepatří aspoň dvě políčka z různých sloupců (zbylé výskyty s v řádku r čtverce μ), a platí $k+m+1 \leq n-3$.

□

Pozorování 1.1.18. Všimneme si, že pokud latinský čtverec obsahuje $r - r'$ cyklus ve tvaru (C3), pak existuje jiný $r - r'$ cyklus obsahující kladný výskyt nevlastního symbolu. Ten je tvaru (C0), neboť neobsahuje nevlastní políčko. Tabulka sjednocení těchto dvou cyklů vypadá následujícím způsobem:

$$\begin{array}{c} r \\ r' \end{array} \begin{pmatrix} b_m & b_{m-1} & \ldots & b_1 & s & x+y-s & a_1 & \ldots & a_k & s & c_1 & \ldots & c_l \\ s & b_m & \ldots & b_2 & b_1 & a_1 & a_2 & \ldots & x & c_1 & c_2 & \ldots & y \end{pmatrix}$$

1.2 Latinské záměny a výměna řádků cyklu

V této sekci zavedeme pojem latinské záměny, který umožní transformaci vlastního latinského čtverce do nevlastního a obráceně. Následně dokážeme existenci posloupnosti latinských záměn, které prohodí obsah řádků cyklické konfigurace vlastního latinského čtverce.

Definice 1.2.1. (Latinská záměna). Nechť μ je latinský čtverec. Ať je μ vlastní a obsahuje konfiguraci $\begin{array}{c} \\ r \\ r' \end{array} \begin{array}{c} \begin{array}{cc} c & c' \end{array} \\ \begin{pmatrix} s^* & s \\ s & a \end{pmatrix} \end{array}$. *Latinskou $s - s^*$ záměnou* nazveme tah, který

danou konfiguraci změní na $\begin{matrix} & c & c' \\ r & s & s^* \\ r' & s^* & a+s-s^* \end{matrix}$. Ať je μ nevlastní a obsahuje konfiguraci $\begin{matrix} & c & c' \\ r & x+s^*-s & s \\ r' & s & a \end{matrix}$, kde s je nevlastní symbol, *latinskou* $s-s^*$ *záměnou* nazveme tah, který danou konfiguraci změní na $\begin{matrix} & c & c' \\ r & x & s^* \\ r' & s^* & a+s-s^* \end{matrix}$. Pokud $a = s^*$, tímto tahem vznikne vlastní latinský čtverec.

Všimněme si, že latinskou $s - s^*$ záměnou se dané zobrazení pozmění tak, že se $s - s^*$ přičte k prvkům hlavní diagonály zvolené konfiguraci a odečte se od prvků vedlejší. Zbylé políčka tabulky zobrazení zůstanou nepozměněné.

Chceme-li na vlastní latinský čtverec řádu n aplikovat nějakou latinskou záměnu, máme n^2 možnosti jak zvolit dvojici $(r, c) \in \mathrm{Dom}(\mu)$ a $n - 1$ možností pro volbu symbolu $s \in C$ takového, že $s \neq s^*$, $s^* = \mu(r,c)$. Touto volbou dostaneme jednoznačně určené $(r, c'), (r', c), (r', c') \in \mathrm{Dom}(\mu)$, pro něž platí $\mu(r,c') = \mu(r',c) = s$ a $\mu(r',c') = a$. Existuje tedy $n^2(n-1)$ možných latinských záměn pro vlastní latinský čtverec řádu n.

V případě nevlastního latinského čtverce musí byt jedno políčko přípustné konfigurace nevlastní. Za $s^* \in C$ je třeba zvolit jeden ze dvou kladných symbolů nevlastního políčka. V nevlastním řádku r volíme jedno ze dvou políček obsahujících kladný výskyt nevlastního symbolu a stejně postupujeme i pro nevlastní sloupec c. Počet latinských záměn pro nevlastní latinský čtverec tedy nezávisí na řádu čtverce a je vždy roven 8.

Příklad 1.2.2. Uvažujme konstrukční konfigurace μ_1 a μ_2, definované v příkladu 1.1.14. Aplikováním latinské $2-1$ záměny na konfiguraci $\begin{matrix} & 2 & \\ 3 & 2 & 1 \end{matrix}$ obsazenou v μ_1 dostaneme čtverec μ_2. Příslušný tah budeme značit takto $\begin{matrix} & 2 & \\ 3 & 2 & 1 \end{matrix} \overset{2-1}{\to} \begin{matrix} & 2 & \\ 3 & 3+2-1 & 1 \end{matrix}$.

Následující lemma umožní pro každý nevlastní latinský čtverec nalézt posloupnost latinských záměn, která ho transformuje na vlastní.

Lemma 1.2.3. *Ať nevlastní latinský čtverec obsahuje konfiguraci* $\begin{matrix} & s & x_1 & \dots & x_{k-1} & z+x_k-s \\ r & x_1 & x_2 & \dots & x_k & s \end{matrix}$, *kde* $k \geq 1$, s *je nevlastní symbol. Pak latinskými záměnami pouze uvnitř této konfigurace lze danou konfiguraci pozměnit na* $\begin{matrix} & x_1 & x_2 & \dots & x_k & z \\ r & s & x_1 & \dots & x_{k-1} & x_k \end{matrix}$. *Počet tahů je roven* k.

Důkaz. Dokážeme indukcí podle k. Pro $k = 1$ použijeme následující tah $\begin{matrix} r \\ r' \end{matrix} \begin{pmatrix} s & z + x_1 - s \\ x_1 & s \end{pmatrix} \overset{x_1 - s}{\to} \begin{matrix} r \\ r' \end{matrix} \begin{pmatrix} x_1 & z \\ s & x_1 \end{pmatrix}$. Je-li $k \geq 2$, použijeme tah

$\begin{matrix} r \\ r' \end{matrix} \begin{pmatrix} s & z + x_k - s \\ x_1 & s \end{pmatrix} \overset{x_k - s}{\to} \begin{matrix} r \\ r' \end{matrix} \begin{pmatrix} x_k & z \\ x_1 + s - x_k & x_k \end{pmatrix}$, položíme $s' = x_k$, $z' = s$ a použijeme indukční předpoklad na konfiguraci o jeden sloupec kratší $\begin{matrix} r' \\ r \end{matrix} \begin{pmatrix} s' & x_{k-1} & \dots & x_2 & z' + x_1 - s' \\ x_{k-1} & x_{k-2} & \dots & x_1 & s' \end{pmatrix}$, což vyžaduje $k-1$ tahů.

□

Důsledek 1.2.4. *Ať vlastní latinský čtverec obsahuje cyklickou konfiguraci tvaru (C0):* $\begin{matrix} r \\ r' \end{matrix} \begin{pmatrix} x_1 & x_2 & \dots & x_k & x_{k+1} \\ x_2 & x_3 & \dots & x_{k+1} & x_1 \end{pmatrix}$, $k \geq 1$. *Pak latinskými záměnami pouze uvnitř toho $r - r'$ cyklu lze získat cyklus* $\begin{matrix} r \\ r' \end{matrix} \begin{pmatrix} x_2 & x_3 & \dots & x_{k+1} & x_1 \\ x_1 & x_2 & \dots & x_k & x_{k+1} \end{pmatrix}$. *Počet tahů je roven k.*

Důkaz. Dokážeme indukcí podle k. Pro $k = 1$ jde o následující záměnu $\begin{matrix} r \\ r' \end{matrix} \begin{pmatrix} x_1 & x_2 \\ x_2 & x_1 \end{pmatrix} \overset{x_2 - x_1}{\to} \begin{matrix} r \\ r' \end{matrix} \begin{pmatrix} x_2 & x_1 \\ x_1 & x_2 \end{pmatrix}$. Je-li $k \geq 2$, použijeme záměnu

$\begin{matrix} r \\ r' \end{matrix} \begin{pmatrix} x_1 & x_2 \\ x_2 & x_3 \end{pmatrix} \overset{x_2 - x_1}{\to} \begin{matrix} r \\ r' \end{matrix} \begin{pmatrix} x_2 & x_1 \\ x_1 & x_3 + x_2 - x_1 \end{pmatrix}$. Položíme $z = x_2$, $s = x_1$ a použijeme lemma 1.2.3 na cyklus $\begin{matrix} r' \\ r \end{matrix} \begin{pmatrix} x_1 & x_{k+1} & \dots & x_4 & x_3 + x_2 - x_1 \\ x_{k+1} & x_k & \dots & x_3 & x_1 \end{pmatrix}$, což vyžaduje $k-1$ tahů.

□

Důsledek 1.2.5. *Ať μ je nevlastní latinský čtverec řádu n. Pak existuje posloupnost nejvýše $(n-1)/2$ latinských záměn, která transformuje μ na čtverec vlastní.*

Důkaz. Předpokládejme, že $(r,c) \in \mathrm{Dom}(\mu)$ je nevlastní políčko. Existuje řádek $r' \in A$, $r' \neq r$, že $\mu(r',c) = s$. Podle lemmatu 1.1.16, μ obsahuje $r - r'$ cyklus ve tvaru (C2): $\begin{matrix} & & & & & & c & & & \\ r \\ r' \end{matrix} \begin{pmatrix} b_0 & b_1 & \dots & s & x + y - s & s & a_0 & \dots & a_k \\ y & b_0 & \dots & b_m & s & a_0 & a_1 & \dots & x \end{pmatrix}$, kde s je nevlastní symbol. Bez újmy na obecnosti lze předpokládat, že $k \leq m$. Pak dle důsledku 1.1.17 $k \leq (n-1)/2$. Aplikujeme lemma 1.2.3 na konfiguraci $\begin{matrix} & c & & & & \\ r \\ r' \end{matrix} \begin{pmatrix} x + y - s & s & a_0 & \dots & a_k \\ s & a_0 & a_1 & \dots & x \end{pmatrix}$, což vyžaduje nejvýše $(n-1)/2$ tahů.

□

1.3 Prohození obsahu dvou políček ve stejném řádku jako posloupnost latinských záměn

Cílem této sekce je dokázat lemma 1.3.2, podle něhož existuje posloupnost latinských záměn, která prohodí obsah dvou políček ve stejném řádku latinského čtverce.

Lemma 1.3.1. *Ať μ je nevlastní latinský čtverec řádu n, který obsahuje konfiguraci tvaru* $\begin{matrix} r \\ r' \\ u \\ v \end{matrix}\begin{pmatrix} & & s^* & a_1 & \dots & a_m \\ & & a_1 & a_2 & \dots & s^* \\ t+s-s^* & & & & & \\ & s^* & & & & \end{pmatrix}$, *kde s^* je nevlastní symbol. Pak posloupností tahů pouze uvnitř řádků u, v, r, r' lze ji transformovat na* $\begin{matrix} r \\ r' \\ u \\ v \end{matrix}\begin{pmatrix} & & a_1 & a_2 & \dots & s^* \\ & & s^* & a_1 & \dots & a_m \\ t+s-s^* & & & & & \\ & s^* & & & & \end{pmatrix}$ *a to tak, že ve všech políčcích tabulky μ mimo konfiguraci zůstane původní stav. Počet tahů je nejvýše $n+m-1$.*

Důkaz. Podle lemmatu 1.1.16 μ obsahuje konfiguraci $\begin{matrix} u \\ v \end{matrix}\begin{pmatrix} s^* & x_1 & \dots & x_{k-1} & x_k+s-s^* \\ x_1 & x_2 & \dots & x_k & s^* \end{pmatrix}$, kde $x_k = t$ a dle důsledku 1.1.17 platí nerovnost $k \leq (n-1)/2$. Položíme $z = s$ a použijeme lemma 1.2.3. Výsledný latinský čtverec je vlastní, což nám umožní následně aplikovat důsledek 1.2.4 na $\begin{matrix} r \\ r' \end{matrix}\begin{pmatrix} a_1 & a_2 & \dots & s^* \\ s^* & a_1 & \dots & a_m \end{pmatrix}$, tj. prohodit obsah vyznačených částí řádků r a r'. Následně inverzním sledem vrátíme změny provedené v řádcích u a v.

Počet záměn se skládá z aplikace lemmatu 1.2.3 pro $k \leq (n-1)/2$, vracení těchto změn a použití lemmatu 1.2.4 pro $k = m$: $2((n-1)/2)+m = n+m-1$

□

Lemma 1.3.2. *Ať $\mu\colon A\times B \to F(C)$ je latinský čtverec řádu n obsahující buď konfiguraci tvaru* $\begin{matrix} & c & c' \\ t & s & s' \\ r & x+y-s & \\ r' & & s \end{matrix}$ *nebo* $\begin{matrix} & c & c' \\ t & s & s' \\ r & x+y-s & s \end{pmatrix}$, *kde s je nevlastní symbol, pokud μ je nevlastní. Pokud je μ vlastní (tj. $y = s$ nebo $x = s$), uvažujeme pouze druhou možnost. Pak existuje posloupnost nejvýše $2n$ latinských záměn, jejíž aplikací se μ transformuje na konstrukční latinský čtverec takový, že $\mu(t,c) = s'$ a $\mu(t,c') = s$. Ke změnám mimo políčka (t,c), (t,c') dojde pouze v řádcích r a r'. Je-li výsledný čtverec nevlastní, pak (r,c) je nevlastní políčko.*

Důkaz. Rozdělíme důkaz na dva případy.

Případ 1: μ obsahuje konfiguraci $\begin{matrix} & c & c' \\ t & s & s' \\ r & x+y-s & s \end{matrix}$. Ať nejprve

$\{s, s'\} \cap \{x, y\} \neq \emptyset$. Můžeme předpokládat, že $y = s'$ a provést tah $\begin{matrix} t \\ r \end{matrix}\begin{pmatrix} s & s' \\ x+s'-s & s \end{pmatrix} \overset{s' - s}{\to} \begin{matrix} t \\ r \end{matrix}\begin{pmatrix} s' & s \\ x & s' \end{pmatrix}$. Je-li $\{s, s'\} \cap \{x, y\} = \emptyset$, dvěma tahy za sebou dostaneme $\begin{matrix} t \\ r \end{matrix}\begin{pmatrix} s & s' \\ x+y-s & s \end{pmatrix} \overset{x - s}{\to} \begin{matrix} t \\ r \end{matrix}\begin{pmatrix} x & s'-x+s \\ y & x \end{pmatrix} \overset{s' - x}{\to}$

$\begin{matrix} t \\ r \end{matrix}\begin{pmatrix} s' & s \\ x+y-s' & s' \end{pmatrix}$.

Případ 2: μ obsahuje konfiguraci $\begin{matrix} & c & c' \\ t & s & s' \\ r & x+y-s & \\ r' & & s \end{matrix}$. Tento případ chceme redukovat na první, tj. ukázat, že existuje posloupnost tahů, která umístí symbol s na pozici (r, c') a ke změnám dojde pouze v řádcích r a r'. Protože μ je nutně nevlastní latinský čtverec, existuje řádek $u \in A$, $u \neq t$, že $\mu(u, c) = s$. Obsah políčka (r', c) je odlišný od s, pak dle lemmatu 1.1.16 je $r - r'$ cyklus obsahující nevlastní políčko (označíme ho μ') ve tvaru buď (C1) nebo (C3). Zvolíme $s^* \in C$, $s^* \in \{x, y\}$ symbol takový, že existuje posloupnost sloupců $\gamma_0, \gamma_1, \ldots, \gamma_q \in B$, že pro $i \in \{1, 2, \ldots, q\}$ platí $\mu'(r, \gamma_0) = s$, $\mu'(r, \gamma_i) = \mu'(r', \gamma_{i-1})$, $\mu'(r', \gamma_q) = s^*$ a položíme $c^* = \gamma_0$. Dále budeme pracovat s konfigurací, která obsahuje všechny $r - r'$ cykly, ve kterých figurují sloupce c',c,c^*. V případě tvaru C1 je touto konfigurací výchozí cyklus. Pokud je tvaru C3, je nutné přidat další cyklus tak, jak uvedeno v pozorování 1.1.18. Popsanou konfiguraci ještě rozšíříme o políčka (u,c) a (u,c^*) a označíme ν. Použijeme $s - s^*$ záměnu na $\begin{matrix} & c & c^* \\ r & x+s^*-s & s \\ u & s & f \end{matrix}$ a označme ji U-záměnou. Pokud μ' byla ve tvaru (C1), U transformuje ν následovně:

$$\begin{matrix} & & & & & & c & & & c' & c^* & & & \\ r & b_m & b_{m-1} & \ldots & b_1 & s & x+s^*-s & a_1 & \ldots & a_k & s & c_1 & \ldots & c_l \\ r' & x & b_m & \ldots & b_2 & b_1 & a_1 & a_2 & \ldots & s & c_1 & c_2 & \ldots & s^* \\ u & & & & & & s & & & & f & & & \end{matrix}$$

$$\downarrow U$$

$$\begin{matrix} & & & & & c & & & c' & c^* & & & \\ r & b_m & b_{m-1} & \ldots & b_1 & s & x & a_1 & \ldots & a_k & s^* & c_1 & \ldots & c_l \\ r' & x & b_m & \ldots & b_2 & b_1 & a_1 & a_2 & \ldots & s & c_1 & c_2 & \ldots & s^* \\ u & & & & & & s^* & & & & f+s-s^* & & & \end{matrix}$$

Byl-li cyklus $\operatorname{Dom}\mu'$ ve tvaru (C3), dostaneme:

$$\begin{array}{c|ccccccccccccc} & c' &&&&& c &&&& c^* &&& \\ r & b_m & b_{m-1} & \dots & b_1 & s & x+s^*-s & a_1 & \dots & a_k & s & c_1 & \dots & c_l \\ r' & s & b_m & \dots & b_2 & b_1 & a_1 & a_2 & \dots & x & c_1 & c_2 & \dots & s^* \\ u & &&&&& s &&&& f &&& \end{array}$$

$$\downarrow_U$$

$$\begin{array}{c|ccccccccccccc} & c' &&&&& c &&&& c^* &&& \\ r & b_m & b_{m-1} & \dots & b_1 & s & x & a_1 & \dots & a_k & s^* & c_1 & \dots & c_l \\ r' & s & b_m & \dots & b_2 & b_1 & a_1 & a_2 & \dots & x & c_1 & c_2 & \dots & s^* \\ u & &&&&& s^* &&&& f+s-s^* &&& \end{array}.$$

Nyní vysvětlíme, proč je možné umístit symbol s do políčka (r, c') takovým způsobem, že prohodíme obsah řádků $r - r'$ cyklu obsahujícího symbol s. Pokud $f = s^*$ (tj. po použití U-záměny čtverec je vlastní), aplikujeme důsledek 1.2.4 na tento cyklus. Pokud $k \neq s^*$, najdeme řádek $v \in A$, $v \neq u$, že $\mu(v,c^*) = s^*$ a na konfiguraci

$$\begin{array}{c|ccccccccccc} & c^* &&&&&& c &&&& c' \\ r & & s & b_1 & \dots & b_{m-1} & b_m & x & a_1 & \dots & a_k \\ r' & & b_1 & b_2 & \dots & b_m & x & a_1 & a_2 & \dots & s \\ u & f+s-s^* &&&&&&&&& \\ v & s^* &&&&&&&&& \end{array} \text{ nebo}$$

$$\begin{array}{c|ccccc} & c^* &&&& c' \\ r & & s & b_1 & \dots & b_m \\ r' & & b_1 & b_2 & \dots & s \\ u & f+s-s^* &&&& \\ v & s^* &&&& \end{array}$$

aplikujeme lemma 1.3.1 . Počet sloupců konfigurace vyznačeného $r - r'$ cyklu je nejvýše $n-4$, tedy použití lemmatu 1.3.1 vyžaduje nejvýše $n+n-3-1 = 2n-4$ záměny.

Zásadní je, že bez ohledu na to, zda byl čtverec po aplikaci záměny U vlastní nebo ne, sloupec c^* a řádek u zůstaly nepozměněné následnými úpravami. To znamená, že se obsah políček (u,c), (r,c^*) a (u,c^*) nezměnil, tedy můžeme aplikovat $s^* - s$ záměnu na $\begin{array}{c|cc} & c & c^* \\ r & a_1 & s^* \\ u & s^* & f+s-s^* \end{array}$ nebo $\begin{array}{c|cc} & c & c^* \\ r & x & s^* \\ u & s^* & f+s-s^* \end{array}$ a takto vrátit změny provedené v řádku u U-záměnou. Výsledný čtverec pak obsahuje buď konfiguraci $\begin{array}{c|cc} & c & c' \\ t & s & s' \\ r & a_1+s^*-s & s \end{array}$ nebo $\begin{array}{c|cc} & c & c' \\ t & s & s' \\ r & x+s^*-s & s \end{array}$.

Celkový počet záměn se skládá ze dvou záměn provedených v případě 1, U-záměny, její inverze a použití lemmatu 1.3.1: 2+1+1+(2n−4) = 2n.

□

1.4 Souvislost grafu rozšířeného prostoru latinských čtverců

Uvažujeme graf G, jehož vrcholy jsou vlastní i nevlastní latinské čtverce daného řádu, a kde hrany vyjadřují vazbu danou nějakou latinskou záměnou. V dané sekci dokážeme, že G je souvislý a $\mathrm{diam}(G) < 2(n-1)^3+(n-1)/2$.

Příklad 1.4.1. Uvažujme čtverce $\mu_1, \mu_2 \colon A\times B \to F(C)$ řadu 5, kde $A = B = C = \{1,2,3,4,5\}$.

μ_1:

1	2	3	4	5
2	5	1	3	4
3	4	5	1	2
4	1	2	5	3
5	3	4	2	1

μ_2:

1	2	3	4	5
5	1	2	3	4
4	5	1	2	3
3	4	5	1	2
2	3	4	5	1

Permutaci π_k definujeme jako permutaci množiny C, která posílá $\mu_1(k,j)$ na $\mu_2(k,j)$ pro každé $j \in B$. Pokud $B = \{1,2,\ldots,n\}$, budeme π_k zapisovat jako matici $\pi_k = \begin{pmatrix} \mu_1(k,1) & \mu_1(k,2) & \ldots & \mu_1(k,n) \\ \mu_2(k,1) & \mu_2(k,2) & \ldots & \mu_2(k,n) \end{pmatrix}$. Pak první řádky definují permutaci $\pi_1 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix}$, druhé řádky definují $\pi_2 = \begin{pmatrix} 2 & 5 & 1 & 3 & 4 \\ 5 & 1 & 2 & 3 & 4 \end{pmatrix}$ atd. Všimněme si, že první řádky tvoří triviální permutaci, protože se μ_1 a μ_2 v těchto řádcích shodují. Pak úlohu transformace k-tého řádku μ_1 na k-tý řádek μ_2 můžeme ekvivalentně formulovat jako úlohu transformace permutace tvořené k-tými řádky μ_1 a μ_2 na triviální. Čtverec μ_2 nazveme *cílový* a v matici π_k budeme zobrazovat jeho k-tý řádek jako druhý, tj. pod k-tým řádkem čtverce μ_1, který nazveme *výchozí*.

V lemmatu 1.4.4 dokážeme, že pro libovolné dva vlastní latinské čtverce $\mu_1, \mu_2 \colon A\times B \to F(C)$ a netriviální permutaci π_k, kde $k \in A$, existuje posloupnost latinských záměn, která transformuje μ_1 tak, že se permutace tvořená k-tými řádky trivializuje. Kromě toho v lemmatu dokážeme, že řádky μ_1 mimo následně definovanou množinu $\mathcal{R}_k$ zůstanou nepozměněné.

Definice 1.4.2. Nechť $\mu_1,\mu_2 \colon A\times B \to F(C)$ jsou vlastní latinské čtverce, μ_1 je výchozí, μ_2 je cílový čtverec. Uvažujme netriviální permutaci π_k, kde $k \in A$. Pak definujeme $\mathcal{R}_k \subset A$ jako množinu všech řádků $i \in A$ takových, že existují symboly $s_1, s_2 \in C, s_1 \neq s_2$ a sloupec $c \in B$ splňující: $\mu_2(k,c) = s_2$ a $\mu_1(k,c) = s_1$, $\mu_1(i,c) = s_2$.

Množina $\mathcal{R}_k$ je nejmenší v tom smyslu, že obsahuje pouze ty řádky výchozího čtverce, které musí být během úpravy permutace π_k na triviální pozměněné pro zachování vlastností latinského čtverce.

Příklad 1.4.3. Uvažujme latinské čtverce z příkladu 1.4.1. Máme $\pi_2 = \begin{pmatrix} 2 & 5 & 1 & 3 & 4 \\ 5 & 1 & 2 & 3 & 4 \end{pmatrix}$ a $\mathcal{R}_2 = \{4,5\}$, neboť políčka $(5,1)$, $(4,2)$ a $(4,3)$ obsahují cílové symboly pro políčka $(2,1)$, $(2,2)$ a $(2,3)$ a tedy musí být pozměněná.

Existuje více možnosti jak transformovat π_2 na triviální. Pokusíme se to udělat tak, aby nedošlo ke změnám mimo řádky z množiny $\mathcal{R}_2$. Tahem $\begin{smallmatrix} & 1 & 2 \\ 2 & 2 & 5 \\ 5 & 5 & 3 \end{smallmatrix} \overset{5-2}{\rightarrow}$ $\begin{smallmatrix} & 1 & 2 \\ 2 & 5 & 2 \\ 5 & 2 & 3+5-2 \end{smallmatrix}$ umístíme symbol 5 ve správné políčko. Po této úpravě μ_1 obsahuje konfiguraci $\begin{smallmatrix} & 2 & 3 \\ 2 & 2 & 1 \\ 4 & & 2 \\ 5 & 3+5-2 & \end{smallmatrix}$. Aplikujeme na ni lemma 1.3.2 a umístíme 2 a 1 ve správná políčka. Navíc z lemmatu víme, že se kromě toho pozmění jenom řádky 4 a 5.

Lemma 1.4.4. *Ať μ_1,$\mu_2 \colon A \times B \to F(C)$ jsou vlastní latinské čtverce řádu n, μ_1 je výchozí, μ_2 je cílový čtverec. Uvažujme netriviální permutaci $\pi_k = \begin{pmatrix} a_1 & a_2 & \dots & a_{n-1} & a_n \\ b_1 & b_2 & \dots & b_{n-1} & b_n \end{pmatrix}$, kde $k \in A$, $a_i = b_j$ pro nějaké $i,j \in \{1,2,\dots,n\}$. Pak existuje posloupnost latinských záměn, jejichž aplikací na μ_1 transformujeme danou permutaci na triviální $\begin{pmatrix} b_1 & b_2 & \dots & b_{n-1} & b_n \\ b_1 & b_2 & \dots & b_{n-1} & b_n \end{pmatrix}$, aniž by došlo ke změnám mimo řádky z množiny $\mathcal{R}_k$. Horní hranice počtu záměn je $1+2n(n-2)+(n-1)/2$.*

Důkaz. Ať $W = \{i \in B; \mu_1(k,i) \neq \mu_2(k,i)\}$. Pro stanovení maximálního možného počtu záměn budeme předpokládat, že $W = B$. Existují $c, c' \in W$, že μ_1 obsahuje konfiguraci $\begin{smallmatrix} & c & c' \\ k & b_1 & a_1 \\ r & x & b_1 \end{smallmatrix}$, kde $r \in \mathcal{R}_k$, protože v pozici (r,c') se nachází cílový symbol pro políčko (k,c'). Položíme $s = b_1$, $s' = a_1$, $t = k$ a aplikujeme na danou konfiguraci lemma 1.3.2. Tímto krokem se symbol $s = b_1$ umístí ve správné políčko a nyní $W = B \setminus \{c'\}$. Zvolíme $m \in W$, že $s' = a_1 = b_m$ a řádek $r' \in \mathcal{R}_k$, že s' je cílový symbol v políčko (k,c'') pro nějaké $c'' \in W$. Pak μ_1 obsahuje konfiguraci $\begin{smallmatrix} & c & c'' \\ k & b_m & a_l \\ r & x+b_1-b_m & \\ r' & & b_m \end{smallmatrix}$ nebo $\begin{smallmatrix} & c & c'' \\ k & b_m & a_l \\ r' & x' & b_m \end{smallmatrix}$, kde $l = c''$. Položíme teď $c' = c''$, $s = s' = b_m$, $s' = a_l$, $t = k$ a aplikujeme lemma 1.3.2 na uvedenou konfiguraci. Tím se $s = b_m$ umístí ve správné políčko a nyní $W = A \setminus \{c',c''\}$. Opakujeme tento krok, dokud se každý symbol neumístí ve správné políčko (tj. $W = \emptyset$). Je-li výsledný čtverec vlastní, pak máme hotovo. Předpokládejme, že není, pak μ_1 obsahuje konfiguraci $\begin{smallmatrix} & c \\ k & b_f \\ r'' & z_p + y_t - b_f \end{smallmatrix}$, kde $r'' \in \mathcal{R}_k$, $z_p + y_t - b_f \notin C$. Existuje řádek $u \in A$, $u \neq k$, že μ_1 dle lemmatu 1.1.16 obsahuje cyklus tvaru $(C2)$:

$$\begin{array}{c} \\ r \\ u \end{array} \begin{array}{c} \begin{array}{cccccccc} & & & & c & & & \\ b_f & z_1 & \dots & z_{p-1} & z_p + y_t - b_f & y_{t-1} & \dots & y_1 & b_f \end{array} \\ \begin{pmatrix} z_1 & z_2 & \dots & z_p & b_f & y_t & \dots & y_2 & y_1 \end{pmatrix} \end{array}.$$

Bez újmy na obecnosti lze předpokládat, že $p \leq t$. Pak dle důsledku 1.1.17 $p \leq (n-1)/2$. Aplikujeme na $\begin{array}{c} \\ r \\ u \end{array} \begin{array}{c} \begin{array}{cccccc} & & & & c & \\ b_f & z_1 & \dots & z_{p-1} & z_p + y_t - b_f \end{array} \\ \begin{pmatrix} z_1 & z_2 & \dots & z_p & b_f \end{pmatrix} \end{array}$ lemma 1.2.3.

Výsledný čtverec je vlastní. Pokud řádek u byl touto posloupností záměn pozměněn poprvé, pak původní čtverec obsahoval v políčku (u, c) symbol b_f cílový do (k, c), tedy $u \in \mathcal{R}_k$.

Spočteme počet záměn. Aplikovali jsme lemma 1.3.2 (případ 1), pak nejvíc $(n-2)$–krát lemma 1.3.2 (případ 2) a případně lemma 1.2.3, tedy dohromady: $1+2n(n-2)+(n-1)/2$.

□

Věta 1.4.5. *Nechť μ_1,μ_2: $A\times B \to F(C)$ jsou konstrukční latinské čtverce řádu n. Pak existuje posloupnost latinských záměn, která transformuje μ_1 na μ_2. Horní hranice velikosti nejkratší takové posloupnosti je $2(n-1)^3+(n-1)/2$.*

Důkaz. Pro maximalizaci počtu záměn předpokládejme, že μ_1 a μ_2 jsou nevlastní. Aplikujeme důsledek 1.2.5 na oba čtverce. Nechť teď μ_1 je výchozí a μ_2 cílový čtverec, přičemž oba dva jsou vlastní. Předpokládejme, že π_k je netriviální pro každé $k \in A$ a předpokládejme $A = \{1, 2, \dots, n\}$. Aplikujeme lemma 1.4.4 na π_1. Tím se první řádek μ_1 transformuje na první řádek μ_2. Předpokládejme, že aplikovali jsme lemma 1.4.4 na prvních m řádků. Pak množina $\mathcal{R}_{m+1}$ neobsahuje žádný z už upravených řádků, což plyne z definice množiny $\mathcal{R}_{m+1}$. Použijeme-li lemma 1.4.4 na π_{m+1}, nezasáhneme už upravené řádky. Tímto způsobem transformujeme prvních $n-2$ řádků μ_1 na odpovídající cílové řádky μ_2. Poslední dva řádky potřebují se v nejhorším případě prohodit, což pomocí důsledku 1.2.4 uděláme $n-1$ záměnami. Nakonec aplikujeme na μ_1 tah, inverzní provedenému na začátku na μ_2 lemmatem 1.2.5.

Spočteme teď celkový počet latinských záměn. Ten se skládá z aplikování lemmatu 1.2.5 na oba čtverce a na jeden čtverec odpovídající inverzní tah, aplikování na $n-2$ řádků lemmatu 1.4.4 a aplikování jednou důsledku 1.2.4: $3 \cdot (n-1)/2+(n-2)(1+2n(n-2) + (n-1)/2)+(n-1) < 2(n-1)^3+(n-1)/2$.

□

1.5 Algoritmus Jacobsona a Matthewse

Předpokládejme, že latinský čtverec μ' vznikl z μ aplikováním posloupnosti latinských záměn. Díky větě 1.4.5 máme jistotu, že pro libovolné μ a μ' taková posloupnost existuje. Jinými slovy, vždy existuje cesta mezi dvěma libovolnými vrcholy grafu G definovaného v předchozí sekci.

Pokud začneme v libovolném vrcholu grafu G a budeme jím náhodně procházet, pak se po určitém počtu kroků konečný bod vzniklého sledu může nacházet

v libovolném vrcholu grafu G. V této sekci popíšeme algoritmus Jacobsona a Matthewse, který realizuje náhodné procházení grafem G. Sled, který vzniká při procházení G, budeme zaznamenávat jako posloupnost latinských čtverců. Pokud máme sled $[\mu_0, \mu_1, \ldots, \mu_m]$, pak pro každé $i \in \{1,2,\ldots,m\}$ jsou μ_i a μ_{i-1} spojené v G právě jednou hranou. V řeči latinských záměn, μ_i je vygenerován z μ_{i-1} jedinou latinskou záměnou vybranou náhodně. Takže μ_m je vygenerován z μ_0 pomocí m latinských záměn.

Vzhledem k tomu, že hlavním cílem této práce je generovat vlastní latinský čtverec z jiného vlastního čtverce, předpokládáme, že krajní body takové posloupnosti jsou vlastní latinské čtverce. Podmínkou zastavení algoritmu bude počet navštívených vlastních latinských čtverců. Začneme-li v nějakém vlastním latinským čtverci μ_0, pak algoritmus zastavíme, když se dostaneme do vlastního latinského čtverce μ_m takového, že sled $[\mu_0, \mu_1, ..., \mu_m]$ obsahuje právě k vlastních čtverců.

Algoritmus 1.5.1 Algoritmus Jacobsona a Matthewse

Vstup: Vlastní latinský čtverec μ_0, $k \geq 2$

Výstup: Vlastní latinský čtverec μ_m takový, že vzniklý sled $[\mu_0, \mu_1, ..., \mu_m]$ obsahuje právě k vlastních latinských čtverců.

$i \leftarrow 0$
$hit \leftarrow 0$ ▷ Počítadlo zásahů vlastních čtverců
while $hit < k$ **do**
    if μ_i je vlastní **then**
        $hit \leftarrow hit + 1$
        - náhodně zvolíme jednu z $n^2(n-1)$ možných latinských záměn
        - aplikujeme na μ_i záměnu zvolenou v předchozím kroku

        $\mu_{i+1} \leftarrow$ výsledný čtverec
    else
        - náhodně zvolíme jednu z 8 možných latinských záměn
        - aplikujeme na μ_i záměnu zvolenou v předchozím kroku

        $\mu_{i+1} \leftarrow$ výsledný čtverec
    end if
    $i \leftarrow i + 1$
end while
return μ_i

V řeči procházení grafem G, volíme náhodně ne latinskou záměnu ale výchozí hranu. Z vrcholu reprezentujícího μ_0 v G vychází $n^2(n-1)$ hran. Náhodně vybereme jednu a označíme μ_1 vrchol, který je spojen s μ_0 zvolenou hranou. Poté ověříme podmínku zastavení a opakujeme postup pro μ_1 s tím rozdílem, že je-li μ_1 nevlastní, pak od něj vede 8 hran.

V příloze k bakalářské práci naleznete dokument Latin_square.py, který obsahuje implementaci třídy `LatinSquare` v jazyce Python. Tato třída umožňuje ukládat jak vlastní, tak nevlastní latinské čtverce a obsahuje také řadu pomocných metod pro jednoduchou práci s nimi. Například metoda `make_move()` aplikuje na konstrukční latinský čtverec jednu z možných latinských záměn, kterou vybere náhodně. Pro dosažení náhodné volby využíváme knihovnu `random`. Jednou z metod je také `jm_algorithm(`k`)` s vstupním parametrem k, jež realizuje

logiku algoritmu Jacobsona a Matthewse. Tuto třídu budeme následně využívat pro implementaci algoritmů, které popíšeme v další kapitole.

2. Ortogonalita latinských čtverců

2.1 Generování binární operace ortogonální vůči dvěma zadaným

V této sekci dokážeme výsledky, které poskytují jednoduchý nastroj pro generování binární operace ortogonální vůči dané. Ačkoliv se v celé kapitole zaměřujeme na binární operace (budeme je značit zvláštním symbolem, například $\circ$), v této sekci budou všechna tvrzení dokázána v obecnějším tvaru pro m-ární operace.

Definice 2.1.1. (m-ární operace). Nechť $\mathcal{X}$ je konečná množina, $m \in \mathbb{N}$. *m-ární operací* na množině $\mathcal{X}$ nazveme zobrazení f:

$$f : \mathcal{X}^m \to \mathcal{X}$$

Definice 2.1.2. (Složení). Nechť $f, g_1, g_2, \ldots, g_m$ jsou m-ární operace na konečné množině $\mathcal{X}$, $\hat{a} \in \mathcal{X}^m$. Definujeme jejich *složení* $\odot_f(g_1, g_2, \ldots, g_m)$ jako m-ární operaci na $\mathcal{X}$ definovanou předpisem $\odot_f(g_1, g_2, \ldots, g_m)(\hat{a}) = f(g_1(\hat{a}), g_2(\hat{a}), \ldots, g_m(\hat{a}))$.

Příklad 2.1.3. Nechť binární operace $\circ$, $\star$ a $\cdot$ na množině $\{1, 2, 3, 4\}$ jsou definované jejích tabulkami výsledků (Cayleyho tabulkami):

$\circ$	1	2	3	4
1	1	2	3	4
2	2	1	4	3
3	3	4	1	2
4	4	3	2	1

$\cdot$	1	2	3	4
1	1	2	3	4
2	1	2	3	4
3	1	2	3	4
4	1	2	3	4

$\star$	1	2	3	4
1	1	2	3	4
2	2	3	4	1
3	3	4	1	2
4	4	3	1	2

Jejich složení $\odot_\star(\circ, \cdot)$ budeme zkrácené značit $\odot_\star$. Obdobnou zkratku použijeme dále v textu všude, kde bude jasné z kontextu, které operace by měly být uvedeny v závorkách. Pak Cayleyho tabulka operace $\odot_\star$ vypadá takto:

$\odot_\star$	1	2	3	4
1	1	3	1	2
2	2	2	1	2
3	3	3	3	1
4	4	4	4	4

Definice 2.1.4. (m-ární kvazigrupa). Nechť $\mathcal{X}$ je konečná množina, f je m-ární operace. Řekneme, že dvojice $(\mathcal{X}, f)$ je m-ární kvazigrupa, pokud pro každé $i \in \{1, 2, \ldots, m\}$, $(a_1, a_2, \ldots, a_{i-1}, a_{i+1}, \ldots, a_m) \in \mathcal{X}^{m-1}$, $b \in \mathcal{X}$ existuje právě jedno $a_i \in \mathcal{X}$, že $f(a_1, a_2, \ldots, a_{i-1}, a_i, a_{i+1}, \ldots, a_m) = b$.

Z definic binární kvazigrupy a latinského čtverce jednoduše plyne následující tvrzení.

Tvrzení 2.1.5. *Nechť $\mathcal{X}$ je konečná množina a f je binární operace na $\mathcal{X}$. Dvojice $(\mathcal{X}, f)$ je kvazigrupa pravě tehdy když je f latinský čtverec.*

Jinými slovy, latinský čtverec $\mu\colon A\times B \to F(C)$ nazveme kvazigrupou, pokud $A = B = C$. Takové latinské čtverce budeme vnímat jako binární operace. Operace f binární kvazigrupy $(f, \mathcal{X})$ je vždy latinský čtverec.

Definice 2.1.6. (Ortogonalita m-árních operací). Nechť $g_1, g_2, \ldots, g_m$ jsou m-ární operace na stejné konečné množině $\mathcal{X}$, $\hat{a} \in \mathcal{X}^m$. Definujeme zobrazení φ předpisem:

$$\begin{aligned}\varphi : \mathcal{X}^m &\to \mathcal{X}^m\\ \hat{a} &\mapsto (g_1(\hat{a}), g_2(\hat{a}), \ldots, g_m(\hat{a}))\end{aligned}$$

Řekneme, že g_1, g_2, …, g_m jsou *navzájem ortogonální*, pokud φ je bijekce.

Nechť $\circ$ a $\star$ jsou binární operace na $\mathcal{X}$, $|\mathcal{X}| = n$. Uvažujme tabulku o $n\times n$ políčcích, která v pozici (x, y) je vyplněna dvojicí $(x\circ y, x\star y)$. Z předchozí definice plyne, že $\circ$ a $\star$ jsou navzájem ortogonální, právě když prvky této tabulky jsou po dvou různé. Budeme říkat, že $(\circ, \star)$ tvoří *ortogonální dvojici*. Operace $\cdot$ z příkladu 2.1.3 tvoří ortogonální dvojici s každým latinským čtvercem.

Následující výsledky byly formulovány a dokázány v článku [3] v obecnějším tvaru, který připouští i nekonečné množiny.

Lemma 2.1.7. *Nechť $g_1, g_2, \ldots, g_m$ jsou navzájem ortogonální m-ární operace na konečné množině $\mathcal{X}$, f je m-ární operace na $\mathcal{X}$. Pak zobrazení $f \mapsto \odot_f(g_1, g_2, \ldots, g_m)$ je bijekce mezi m-árními operacemi na $\mathcal{X}$.*

Důkaz. Protože $\mathcal{X}$ je konečná množina, stačí dokázat, že $f \mapsto \odot_f(g_1, g_2, \ldots, g_m)$ je surjekce. Chceme ověřit, že libovolnou m-ární operaci h jde vyjádřit jako $\odot_{f_h}$ pro nějakou m-ární operaci $f_h(\hat{b})$, $\hat{b} = (b_1, b_2, \ldots, b_m) \in \mathcal{X}^m$. Z ortogonality $g_1, g_2, \ldots, g_m$ víme, že existuje právě jedno $\hat{a} = (a_1, a_2, \ldots, a_m) \in \mathcal{X}^m$, že $b_i = g_i(a_i)$, $i \in \{1, 2, \ldots, m\}$. Můžeme pak položit $f_h(g_1(\hat{a}), g_2(\hat{a}), \ldots, g_m(\hat{a})) = h(\hat{a})$ pro každé $\hat{a} \in \mathcal{X}^m$. Pak $h = \odot_{f_h}(g_1, g_2, \ldots, g_m)$.

□

Lemma 2.1.8. *Nechť $g_1, g_2, \ldots, g_m$ jsou navzájem ortogonální m-ární operace na konečné množině $\mathcal{X}$, f je m-ární kvazigrupa na $\mathcal{X}$. Pak $\odot_{f_h}(g_1, g_2, \ldots, g_m), g_2,$ $\ldots, g_m$ jsou rovněž navzájem ortogonální.*

Důkaz. Protože $\mathcal{X}$ je konečná množina, stačí ověřit, že pokud platí $(\odot_f(g_1, g_2, \ldots,$ $g_m)(\hat{a}), g_2(\hat{a}), \ldots, g_m(\hat{a})) = (\odot_f(g_1, g_2, \ldots, g_m)(\hat{b}), g_2(\hat{b}), \ldots, g_m(\hat{b}))$, pak $\hat{a} = \hat{b}$. Položíme $c_2 = g_2(\hat{a}) = g_2(\hat{b})$, $c_3 = g_3(\hat{a}) = g_3(\hat{b})$, …, $c_n = g_n(\hat{a}) = g_n(\hat{b})$. Pak z definice kvazigrupy rovnost $f(g_1(\hat{a}), c_2, c_3, \ldots, c_n) = f(g_1(\hat{b}), c_2, c_3, \ldots, c_n)$ implikuje $g_1(\hat{a}) = g_1(\hat{b})$. Tedy $g_i(\hat{a}) = g_i(\hat{b})$, $i \in \{1,2,\ldots,m\}$ a z ortogonality $g_1, g_2, \ldots, g_m$ plyne, že $\hat{a} = \hat{b}$.

□

Definice 2.1.9. (Ortogonální výměna). Nechť $g_1, g_2, \ldots, g_m$ jsou navzájem ortogonální m-ární operace. m-ární operaci f takovou, že $f, g_2, \ldots, g_m$ jsou rovněž navzájem ortogonální nazveme *ortogonální výměnou* pro $g_1, g_2, \ldots, g_m$.

Věta 2.1.10. *Nechť $g_1, g_2, \ldots, g_m$ jsou navzájem ortogonální m-ární operace na konečné množině $\mathcal{X}$. Je-li f m-ární kvazigrupa na $\mathcal{X}$, pak $\odot_{f_h}(g_1, g_2, \ldots, g_m)$ je ortogonální výměna pro $g_1, g_2, \ldots, g_m$. Navíc zobrazení $f \mapsto \odot_f(g_1, g_2, \ldots, g_m)$ je bijekce mezi všemi m-árními kvazigrupami a všemi m-árními operacemi, které jsou ortogonální výměny pro $g_1, g_2, \ldots, g_m$.*

Důkaz. První část tvrzení plyne z lemmatu 2.1.8. Protože $\mathcal{X}$ je konečná, stačí ověřit, že každá výměna h se dá získat jako $\odot_{f_h}(g_1, g_2, \ldots, g_m)$, kde f_h je kvazigrupa. Konstrukce f_h byla popsaná v důkazu lemmatu 2.1.7. Dokážeme, že takto zkonstruované zobrazení pro nějakou m-ární operaci h, která je výměnou, je zároveň kvazigrupou, tj. pro každé $i \in \{1, 2, \ldots, m\}$ pokud $f_h(a_1, a_2, \ldots, a_{i-1}, a_i, a_{i+1}, \ldots, a_m) = b = f_h(a_1, a_2, \ldots, a_{i-1}, a'_i, a_{i+1}, \ldots, a_m)$, $(a_1, a_2, \ldots, a_{i-1}, a_i, a'_i, a_{i+1}, \ldots, a_m) \in \mathcal{X}^{m+1}$, pak $a_i = a'_i$. Z ortogonality $g_1, g_2, \ldots, g_m$ plyne existence právě jednoho $\hat{c}$, že $a_i = g_i(\hat{c})$, $i \in \{1,2,\ldots,m\}$, $\hat{c} \in \mathcal{X}$, pak $b = h(\hat{c})$. Protože $g_1, g_2, \ldots, g_{i-1}, h, g_{i+1}, \ldots, g_m$ jsou navzájem ortogonální, proto $g_1(\hat{c}), g_2(\hat{c}), \ldots, g_m(\hat{c}), h(\hat{c})$ určují $\hat{c}$ jednoznačně a tedy $a_i = a'_i = g_i(\hat{c})$.

□

Věta 2.1.10 dovoluje pro nějaký latinský čtverec $\circ$ na množině $\mathcal{X}$ generovat sadu ortogonálních binárních operací. Stačí zvolit libovolnou operaci $\cdot$ na $\mathcal{X}$ ortogonální vůči $\circ$ a prohledávat prostor latinských čtverců řádu n pomoci algoritmu Jacobsona a Matthewse. Každý nalezený latinský čtverec označíme jako $\star$ a spočítáme $\odot_\star(\circ, \cdot)$. Pak $(\odot_\star, \circ)$ tvoří ortogonální dvojici.

2.2 Základní algoritmus hledání latinského čtverce ortogonálního danému

K danému latinskému čtverci obecně nemusí existovat latinský čtverec jemu ortogonální. V roce 1900 francouzský matematik Gaston Tarry (1843–1913) dokázal v [4] hypotézu Leonarda Eulera (1707–1783), že neexistuje dvojice navzájem ortogonálních latinských čtverců řádu 6. V této kapitole navrhneme metodu jak hledat ortogonální operaci, která latinský čtverec v jistém smyslu aproximuje a v ideálním případě ortogonální čtverec přímo nalezne. Pro tento účel zavedeme funkci, která bude vyhodnocovat, do jaké míry binární operace (resp. její Cayleyho tabulka) splňuje podmínky kladené na latinský čtverec. Výsledkem této funkce je skóre v rozmezí od 0 do 1, přičemž skóre 1 dostane latinský čtverec. Operace, jejíž tabulka je v každém řádku (resp. sloupci) vyplněna jediným symbolem dostane skóre 0. Definice skóre pro danou operaci f vyplývá z algoritmu 2.2.1, kde C_f značíme její Cayleyho tabulku. Níže je pak vzorec pro skóre uveden explicitně.

Příklad 2.2.1. Spočteme skóre operace $\circ$ z příkladu 2.1.3.
$n = 4$
$i = 1, r_i = 4, c_i = 4, s = 8 - 4 - 4 = 0$

$i = 2, r_i = 4, c_i = 4, s = 8 - 4 - 4 = 0$
$i = 3, r_i = 4, c_i = 4, s = 8 - 4 - 4 = 0$
$i = 4, r_i = 4, c_i = 4, s = 8 - 4 - 4 = 0$
$\texttt{score}(\circ) = 1 - 0/12 = 1$, tedy $\circ$ je latinský čtverec.

Algoritmus 2.2.1 Skóre binární operaci

procedure SCORE(f), kde f je binární operace, C_f je typu $n \times n$
    $s \leftarrow 0$
    for i in $\{1, 2, \ldots, n\}$ **do**
        $r_i \leftarrow$ počet po dvou různých symbolů v i-tém řádku C_f
        $c_i \leftarrow$ počet po dvou různých symbolů v i-tém sloupci C_f
        $s \leftarrow s + 2n - r_i - c_i$
    end for
    skóre $\leftarrow 1 - s/n(n-1)$
    return *skóre*
end procedure

Jinými slovy, označíme-li s skóre binární operace f na $\{1, 2, \ldots, n\}$, pak $s = 1-(\sum_{i=1}^{n} 2n-r_i-c_i)/n(n-1)$, kde r_i (resp. c_i) je počet po dvou různých symbolů v i-tém řádku (resp. sloupci) Cayleyho tabulky f.

Definice 2.2.2. Nechť $f_1, f_2, \ldots, f_m$ jsou binární operace na konečné množině, $N = [f_1, f_2, \ldots, f_m]$. Řekneme, že posloupnost N *se blíží k latinskému čtverci*, pokud pro každé $i \in \{1, 2, \ldots, m-1\}$ platí $\texttt{score}(f_i) \leq \texttt{score}(f_{i+1})$.

Navrhneme nyní algoritmus, který na vstupu dostane latinský čtverec $\circ$ a dva parametry n_{iter} a n_s. V i-tém, $i \in \{1,2,\ldots,n_{iter}\}$, iteračním kroku algoritmus nalezne operaci f_i, která je k dané operaci $\circ$ ortogonální a to tak, aby se posloupnost $[f_1, f_2, \ldots, f_{n_{iter}}]$ blížila k latinskému čtverci. Výstupem algoritmu bude buď operace nalezená v poslední iteraci (která nemusí být latinským čtvercem), nebo pokud byl latinský čtverec nalezen v i-té iteraci, $i < n_{iter}$, nalezený latinský čtverec. Jinými slovy, algoritmus řeší úlohu hledání operace ortogonální k $\circ$, jež maximalizuje skóre.

Připomeňme si, jak byl definován graf G v předchozí kapitole. Vrcholy grafu G jsou vlastní i nevlastní latinské čtverce daného řádu a hrany vyjadřují vazbu danou nějakou latinskou záměnou. Uvažujme graf G', který z G vznikne tak, že vynecháme všechny vrcholy reprezentující nevlastní latinské čtverce a dva vrcholy v G' spojíme hranou, pokud v G mezi nimi existovala cesta, která procházela pouze přes nevlastní latinské čtverce.

Nechť vstupní operace $\circ$ na konečné množině $\mathcal{X}$ je latinský čtverec. Zvolíme nějakou operaci $\cdot$ na $\mathcal{X}$, která s $\circ$ tvoří ortogonální pár. Možnou volbou je třeba operace $f(x, y) = y$ na $\mathcal{X}$ (analogie operace $\cdot$ z příkladu 2.1.3), která je ortogonální ke každému latinskému čtverci. Poté zvolíme libovolný latinský čtverec řádu n a označíme ho $\star$. Podle tvrzení 2.1.5 a věty 2.1.10 touto volbou zajistíme, že operace $\odot_\star(\circ, \cdot)$ je ortogonální vůči $\circ$. Spočítáme $\texttt{score}(\odot_\star)$ a přejdeme k iterační části algoritmu.

Popíšeme jednu iteraci algoritmu. Cílem je nahradit operaci $\star$ operací $\star'$ takovou, že $\texttt{score}(\odot_{\star'}) \geq \texttt{score}(\odot_\star)$. Prostor operací $\star'$ (tj. prostor všech latinských čtverců) budeme prohledávat pomocí algoritmu Jacobsona a Matthewse.

Vzhledem k velikosti prohledávaného prostoru (viz tabulku v úvodu) požijeme heuristiku. Pustíme algoritmus 1.5.1 se vstupem $\star$ a parametrem $k = 2$. Výstupem bude náhodně zvolený vlastní latinský čtverec, který je s $\star$ v G' spojen jednou hranou. Takový čtverec nazveme náhodným sousedem operace $\star$. Takovým způsobem nalezneme n_s náhodných sousedů $\star$. Označíme je $[s_1, s_2, \ldots, s_{n_s}]$ a spočítáme $\sigma_i = \texttt{score}(\odot_{s_i})$ pro každé $i \in \{1, 2, \ldots, n_s\}$ (kde operace $\odot_{s_i}$ je definována jako $\odot_{s_i}(\circ, \cdot)$). Položíme $\sigma_{max} = \max(\sigma_i)$ a označíme $\star'$ operaci se skóre σ_{max}. Položíme $\star = \star'$, $f_i = \odot_\star$, pokud $\texttt{score}(\odot_{\star'}) \geq \texttt{score}(\odot_\star)$, jinak necháme $\star$ nepozměněnou z předchozí iterace a $f_i = \odot_\star$. Takto vzniklá posloupnost operací $[f_1, f_2, \ldots, f_{n_{iter}}]$ se zjevně blíží k latinskému čtverci.

Funkce $\texttt{argmax}([a_1, a_2, \ldots, a_k])$ v následujícím pseudokódu vrátí index největšího elementu vstupní posloupnosti.

Algoritmus 2.2.2 Algoritmus hledání ortogonální operaci

Vstup: $\circ$ - latinský čtverec, n_{iter}, n_s
Výstup: $\odot_\star$ - binární operace ortogonální vůči $\circ$, jež maximalizuje skóre

```
1: · ← libovolná binární operace ortogonální vůči ∘
2: ⋆ ← libovolný latinský čtverec
3: skóre ← score(⊙_⋆)
4: for i in {1, 2, ..., n_iter} do
5:     sousede ← [s_1, s_2, ..., s_{n_s}]
6:     index ← argmax([score(⊙_{s_1}), score(⊙_{s_2}), ..., score(⊙_{s_{n_s}})])
7:     ⋆' ← sousede[index]
8:     if score(⊙_{⋆'}) = 1 then
9:         ⋆ ← ⋆'
10:        return ⊙_⋆
11:    end if
12:    if score(⊙_{⋆'}) ≥ skóre then
13:        ⋆ ← ⋆'
14:        skóre ← score(⊙_⋆)
15:    end if
16: end for
17: return ⊙_⋆
```

Parametry n_{iter}, n_s můžeme nastavovat v závislosti na tom, jak velké skóre chceme dosáhnout. Zvětšení hodnot těchto parametrů zvyšuje šanci nalezení operace s větším skóre, ale může výrazně prodloužit dobu běhu algoritmu. Nejvíc časově náročným krokem je hledaní n_s náhodných sousedů (řádek 5 pseudokódu). Pokud víme, že ke vstupnímu čtverci ortogonální latinský čtverec existuje, pak má výhodu nastavit počet iterací n_{iter} na velkou hodnotu. V okamžik, když se latinský čtverec nalezne, prohledávání se zastaví.

Během testování algoritmu se opakovaně vyskytly situace, kdy se od určité iterace nezvyšovalo skóre. V takovém případě se ukázalo efektivním zavést počítadlo zaznamenávající v kolika iteracích už se skóre nemění. V okamžiku, kdy počítadlo dosáhne nějaké hodnoty (kterou lze zvolit pro každý čtverec zvlášť), aplikujeme na poslední nalezenou operaci posloupnost náhodných latinských záměn a výsledný latinský čtverec použijeme jako vstup další iteraci. Dále v textu

ale budeme vždy mít na myslí verzi algoritmu bez počítadla, pokud nebude explicitně uvedeno jinak.

Příklad 2.2.3. Logiku popsánou v algoritmu 2.2.2 realizuje funkce `find_ort`($\circ$, n_{iter}, n_s) v programu OrtMate_algorithm.py, který naleznete v příloze. Níže uvedený kód ukazuje výstup algoritmu pro cirkulant řádu 7 uložený v proměnnou `L_1` a parametry $n_{iter} = 300$, $n_s = 50$. Nalezená operace je latinským čtvercem. Doba hledání je 0.688 vteřin. K nalezení latinského čtverce stačilo 6 iterací.

```
> L_1
[[1 2 3 4 5 6 7]
[7 1 2 3 4 5 6]
[6 7 1 2 3 4 5]
[5 6 7 1 2 3 4]
[4 5 6 7 1 2 3]
[3 4 5 6 7 1 2]
[2 3 4 5 6 7 1]]
> find_ort(L_1, 300, 50)
[[7 3 5 4 2 1 6]
[5 6 1 2 3 7 4]
[3 2 4 7 6 5 1]
[6 5 7 1 4 3 2]
[1 4 2 3 5 6 7]
[4 7 3 6 1 2 4]
[2 1 6 5 7 4 3]]
```

Latinský čtverec `L_2` řádu 8 potřeboval 438 iterací a 51.328 vteřin pro nalezení latinského čtverce jemu ortogonálního s parametry $n_{iter} = 1000$, $n_s = 100$.

```
> L_2
[[2 1 8 4 6 3 7 5]
[3 8 2 6 4 5 1 7]
[8 6 4 5 1 7 3 2]
[7 4 3 1 5 2 8 6]
[1 7 5 3 2 6 4 8]
[6 3 7 2 8 1 5 4]
[4 5 6 7 3 8 2 1]
[5 2 1 8 7 4 6 3]]
> find_ort(L_2, 1000, 100)
[[2 8 1 7 5 4 3 6]
[6 3 4 1 2 8 5 7]
[7 2 8 3 4 6 1 5]
[8 5 3 6 7 1 2 4]
[1 4 2 5 3 7 6 8]
[3 7 5 8 6 2 4 1]
[4 1 6 2 8 5 7 3]
[5 6 7 4 1 3 8 2]]
```

2.3 Genetický algoritmus hledání latinského čtverce ortogonálního danému

Během testování algoritmu 2.2.2 se ukázalo, že se doba jeho běhu může výrazně lišit pro čtverce stejného řádu nebo dokonce pro ten samý latinský čtverec v různých spuštěních. Tato variabilita je způsobena velikostí prohledávaného prostoru (tj. řádem vstupního čtverce) a tím kolik v G' existuje čtverců ortogonálních danému. V této sekci popíšeme genetický algoritmus 2.3.2 vzniklý na základě algoritmu 2.2.2, který v průměru zkracuje dobu hledání. Na vstupu algoritmus dostane tři parametry navíc: m_{iter}, velikost populace l_p a počet náhodně vygenerovaných latinských čtverců v populaci n_r. Výstup se neliší od algoritmu 2.2.2 a tedy je to operace ortogonální k vstupní operaci, jež maximalizuje skóre. Výhodou tohoto algoritmu je, že je snadno paralelizovatelný. Porovnání obou algoritmů provedeme v přísti sekci.

Genetický algoritmus rovněž prohledává prostor všech vlastních latinských čtverců, ale na rozdíl od algoritmu 2.2.2 v každém iteračním kroku zpracovává celou sadu čtverců, které budeme říkat *populace*. Základní jednotkou algoritmu je populace l_p latinských čtverců, z nichž některé se mohou opakovat. Čtverce v první populaci jsou vygenerované náhodně. Je-li j-tá, $j \in \{1, 2, \ldots, m_{iter}\}$, populace tvořená čtverci $[p_1, p_2, \ldots, p_l]$, kde $l = l_p$, pak použitím základního algoritmu odvodíme $[p'_1, p'_2, \ldots, p'_l]$. Tedy p_i a p'_i, $i \in \{1, 2, \ldots, l\}$ jsou v G' spojené cestou maximalizující skóre. Posloupnost $[p'_1, p'_2, \ldots, p'_l]$ setřídíme podle skóre $s_i = \odot_{p'_i}(\circ, \cdot)$. Pro jednoduchost předpokládejme, že platí $s_1 \geq s_2 \geq \cdots \geq s_l$. Nyní použijeme parametr n_r a to tak, že se do nové populace vloží n_r náhodně vygenerovaných latinských čtverců. Zbylé čtverce se v počtu $l_p - n_r$ volí z množiny $\{p'_1, p'_2, \ldots, p'_l\}$ a to tak, že použijeme vážený náhodný výběr. Váha příslušná čtverci p'_i je s_i.

Volbou skóre čtverců jako váhy pravděpodobnosti jejich výběru při vytváření populace a přidáním náhodně vygenerovaných čtverců zajistíme dvě věci. Zaprvé, do každé nové populace se mohou dostat jedinci (latinské čtverce) s menším skóre a zároveň se přidají i čtverce náhodně vygenerované. Pokud se budou vybírat jen ty nejlepší jedince, tak se po určité době celá populace bude skládat z potomků stejného latinského čtverce. Taková situace hrozí tím, že se prohledávání zacyklí v lokálním optimu. Zadruhé, protože jedinci s největším skóre jsou vybíráni z větší pravděpodobností, budou mít v průměru i více potomků, tedy od nich se cesta větví s větší pravděpodobností.

Postup vytváření nové populace je popsán pseudokódem algoritmu 2.3.1.

Algoritmus 2.3.1 Zpracování populace

1: **procedure** `CREATE_POPULATION`($[p_1, p_2, \ldots, p_{l_p}]$, n_r)
2: $populace \leftarrow [\,]$
3: $s \leftarrow [p_1, p_2, \ldots, p_{l_p}]$
4: $w \leftarrow [s_1, s_2, \ldots, s_{l_p}]$
5: **for** i in $\{1, 2, \ldots, n_r\}$ **do**
6: • přidej do *populace* náhodně vygenerovaný latinský čtverec
7: **end for**
8: **for** i in $\{1, 2, \ldots, l_p - n_r\}$ **do**
9: • zvol čtverec náhodně ze seznamu s na základě příslušných vah w
10: • přidej zvolený čtverec do *populace*
11: **end for**
12: **return** *populace*
13: **end procedure**

Nyní shrneme celý postup do pseudokódu. Mírně modifikujeme základní algoritmus 2.2.2, který použijeme pro odvození čtverce p'_i z p_i. Označíme `find_ort_star`($\circ$, $\star$, n_{iter}, n_s) funkci, která realizuje logiku algoritmu 2.2.2, s tím rozdílem, že operace $\star$ na začátku (řádek 2 pseudokódu) není volena libovolně, ale je předána jako vstupní parametr a místo $\odot_\star$ v řádcích 10, 17 vrátí se $\star$.

Algoritmus 2.3.2 Algoritmus hledání ortogonální operaci

Vstup: $\circ$ - latinský čtverec, n_{iter}, m_{iter}, n_s, l_p, $w = (n_1, n_2, \ldots, n_{l_p}, m)$
Výstup: $\odot_\star$ - binární operace ortogonální vůči $\circ$, jež maximalizuje skóre

1: $populace \leftarrow [p_1, p_2, \ldots, p_{l_p}]$ ▷ l_p náhodně vygenerovaných lat.čtverců
2: **for** i in $\{1, 2, \ldots, m_{iter}\}$ **do**
3: $p \leftarrow [p'_1, p'_2, \ldots, p'_{l_p}]$, kde $p'_i =$ `find_ort_star`$(\star, \circ, populace[i], n_{iter}, n_s)$
4: $\star \leftarrow p[index]$
5: **if** `score`$(\odot_\star) = 1$ **then**
6: **return** $\odot_\star$
7: **end if**
8: $populace \leftarrow$ `create_population`(p, w)
9: **end for**
10: **return** $\odot_\star$

V podstatě máme dva vnořené cykly: jeden s m_{iter} iteracemi a druhý (krytý v kroku 3 pseudokódu) s n_{iter} iteracemi. Parametr n_{iter} lze interpretovat jako počet iterací základního algoritmu, po kterém chceme posoudit úspěšnost výsledných operací z hlediska skóre. Pokud byla nalezena operace se skóre 1, algoritmus se zastaví a vrátí ji. Parametr m_{iter} se spíše nastavuje s ohledem na časová omezení.

Příklad 2.3.1. Logiku algoritmu 2.3.2 realizuje funkce `find_ort_gen`($\circ$, n_{iter}, m_{iter}, n_s, l_p, n_r) v programu OrtMateGen_algorithm.py, který naleznete v příloze. Níže uvedený kód ukazuje výstup algoritmu s parametry $n_{iter} = 300, m_{iter} = 300, n_s = 50, l_p = 10, n_r = 2$ pro cirkulant `L_3` řádu 11. Nalezená operace je latinským čtvercem. Doba hledání je 26.8 minut.

```
> L_3
[[1 2 3 4 5 6 7 8 9 10 11]
```

```
[2 3 4 5 6 7 8 9 10 11 1]
[3 4 5 6 7 8 9 10 11 1 2]
[4 5 6 7 8 9 10 11 1 2 3]
[5 6 7 8 9 10 11 1 2 3 4]
[6 7 8 9 10 11 1 2 3 4 5]
[7 8 9 10 11 1 2 3 4 5 6]
[8 9 10 11 1 2 3 4 5 6 7]
[9 10 11 1 2 3 4 5 6 7 8]
[10 11 1 2 3 4 5 6 7 8 9]
[11 1 2 3 4 5 6 7 8 9 10]]
> find_ort_gen(L_3, 300, 300, 50, 10, 2)
[[3 1 2 9 10 6 4 8 11 5 7]
[2 9 6 11 5 3 7 1 8 10 4]
[5 10 4 1 8 11 2 3 6 7 9]
[7 3 8 10 1 5 6 9 2 4 11]
[1 7 9 6 4 2 11 10 3 8 5]
[4 1 3 7 9 1 5 6 10 11 8]
[11 5 10 4 3 9 8 7 1 6 2]
[9 8 7 2 11 10 1 4 5 3 6]
[6 11 5 8 7 4 3 2 9 1 10]
[10 4 1 5 6 8 9 11 7 2 3]
[8 6 11 3 2 7 10 5 4 9 1]]
```

Příklad 2.3.2. Čtverec `L_4` v následující ukázce kódu byl převzat z katalogu výsledků článku [5], konkrétně ze sady čtverců řádu 8, ke kterým neexistuje ortogonální latinský čtverec. Kód ukazuje výstup algoritmu s parametry $n_{iter} = 10, m_{iter} = 10, n_s = 50, l_p = 10, n_r = 2$. Algoritmus doběhl v čase 15.21 sekundy a nalezená operace má skóre 0.929.

```
> L_4
[[1 2 3 4 5 6 7 8]
[2 1 4 3 6 5 8 7]
[3 4 2 1 8 7 6 5]
[4 3 1 2 7 8 5 6]
[5 6 7 8 4 3 2 1]
[6 5 8 7 3 4 1 2]
[8 7 6 5 1 2 3 4]
[7 8 5 6 2 1 4 3]]
> find_ort_gen(L_4, 10, 10, 50, 10, 2)
[[8 2 4 5 7 6 3 1]
[4 1 7 2 8 3 6 5]
[5 6 3 8 4 7 1 2]
[1 3 6 7 5 2 4 8]
[3 7 1 6 2 5 8 4]
[6 8 5 3 1 4 2 7]
[7 4 2 1 3 8 5 6]
[2 5 8 4 6 1 7 3]]
```

2.4 Porovnání algoritmů v experimentech

V této sekci provedeme srovnání algoritmu 2.2.2 s algoritmem 2.3.2 z hlediska času běhu algoritmů. Proces odvození $[p'_1, p'_2, \ldots, p'_{l_p}]$ z $[p_1, p_2, \ldots, p_{l_p}]$ (tj. krok 3 v pseudokódu) v genetickém algoritmu budeme počítat na deseti jádrech. Realizujeme to pomocí knihovny `multiprocessing`. Testovací množina čtverců řádu 7 byla převzata z katalogu výsledků článku [5] a obsahuje 9 latinských čtverců řádu 7. Dalších 7 náhodně vygenerovaných latinských čtverců řádu 8 a 10 latinských čtverců řádů 9 a 10 byly rovněž poskytnuty jedním z autorů [5] Ianem Wanlessem. O každém latinském čtverci z testovací množiny víme, že existuje latinský čtverec jemu ortogonální.

Vstupní parametry budou nastaveny individuálně pro čtverce každého řádu a to tak, že pro čtverce řádu 7 algoritmy nutně vrátí latinský čtverec. Z důvodu omezených výpočetních možností budou pro čtverce řádů 8, 9 a 10 vstupní parametry nastaveny tak, že algoritmy naleznou ortogonální operace se skóre aspoň 0.94, 0.94 a 0.93 a v okamžik, když se taková operace nalezne, algoritmy budou zastavené. Kromě toho, pro čtverce řádů 8, 9 a 10 přidáme v základním algoritmu počitadlo změn skóre v iterací a to tak, že pokud se skóre nemění 2000 iterací, aplikujeme na poslední nalezenou operaci $\star$ algoritmus Jacobsona a Matthewse s parametrem $k = 300$.

Popíšeme podrobně testovací metodu, jejíž výsledky jsou uvedeny v tabulkách 2.4.1 a 2.4.2. Následně popsaný postup se aplikuje zvlášť na sadu čtverců každého řadu $n \in \{7, 8, 9, 10\}$. Nechť máme sadu $[L_1, L_2, \ldots, L_k]$ latinských čtverců řádu n. Pro každý čtverec provedeme w spuštění algoritmů. Definujeme posloupnosti $T_j^z = [t_1^z, t_2^z, \ldots, t_r^z]$ a $T_j^g = [t_1^g, t_2^g, \ldots, t_r^g]$, kde t_i^z (resp. t_i^g) je doba běhu algoritmu 2.2.2 (resp. 2.3.2) pro čtverec L_j v i-tém spuštění, $i \in \{1, 2, \ldots, r\}$, $j \in \{1, 2, \ldots, k\}$. Průměrnou dobu běhu jednoho spuštění algoritmu 2.2.2 pro čtverec L_j označíme jako P_j^z, kde $P_j^z = (\sum_{i=1}^r t_i^z)/r$ a $t_i^z \in T_j^z$. Podobně pro algoritmus 2.3.2 a čtverec L_j položíme P_j^g, kde $P_j^g = (\sum_{i=1}^r t_i^g)/r$ a $t_i^g \in T_j^g$. Nakonec vypočítáme a zaneseme do tabulek 2.4.2 a 2.4.1 průměrnou dobu běhu všech spuštění algoritmů pro všechny čtverce daného řádu. Tyto hodnoty označíme P^z a P^g, kde $P^z = (\sum_{j=1}^k P_j^z)/k$ a $P^g = (\sum_{j=1}^k P_j^g)/k$. P^z a P^g představují aritmetický průměr hodnot P_j^z a P_j^g pro každé j. V tabulkách výsledků uvedeme také minimální a maximální čas běhu algoritmu ve všech spuštěních algoritmů v sloupcích t_{min} a t_{max}. Hodnoty v posledních třech sloupcích v tabulkách 2.4.1 a 2.4.2 jsou uvedeny v sekundách a zaokrouhleny na celé číslo.

n	r	skóre	Parametry algoritmu	t_{min}	t_{max}	P^z
7	10	1	$n_{iter} = 100$, $n_s = 100$	2	3	3
8	5	> 0.94	$n_{iter} = 30000$, $n_s = 250$	11	4465	1489
9	5	> 0.94	$n_{iter} = 30000$, $n_s = 250$	21	5464	2263
10	5	> 0.93	$n_{iter} = 20000$, $n_s = 250$	237	4362	1239

Tabulka 2.4.1: Čas běhu algoritmu 2.2.2

Pro všechny experimenty s genetickým algoritmem 2.3.2 položíme velikost populace $l_p = 10$ a počet náhodně přidaných čtverců v populaci $n_r = 2$. Velikost populace byla zvolena stejná jako počet dostupných jader. Hodnota parametru n_r byla vybraná v souladu s výsledky výpočetních experimentů pro populaci o

velikosti 10. V následující tabulce uvedeme jenom parametry, které se mění pro různé řády.

n	r	skóre	Parametry algoritmu	t_{min}	t_{max}	P^g
7	10	1	$n_{iter} = 100$, $m_{iter} = 50$, $n_s = 100$	3	5	4
8	5	> 0.94	$n_{iter} = 100$, $m_{iter} = 100$, $n_s = 250$	35	1215	297
9	5	> 0.94	$n_{iter} = 100$, $m_{iter} = 300$, $n_s = 250$	106	2269	410
10	5	> 0.93	$n_{iter} = 200$, $m_{iter} = 300$, $n_s = 300$	155	653	367

Tabulka 2.4.2: Čas běhu algoritmu 2.3.2

O zrychlení času běhu algoritmu můžeme posoudit z hodnoty P^z/P^g. Uvedeme ji v závislosti na řádu čtverce v tabulce 2.4.3.

n	7	8	9	10
P^z/P^g	0.75	5.013	5.52	3.376

Tabulka 2.4.3: Hodnota P^z/P^g v závislosti na řádu n čtverců

Z tabulky 2.4.3 pozorujeme, že základní algoritmus pro čtverce řádu 7 se ukázal jako v průměru rychlejší. Pro řády $8-9$ čtverců z testovací množiny genetický algoritmus doběhl v průměru rychleji. Například, hodnotu 5.013 pro $n = 8$ v tabulce 2.4.3 můžeme interpretovat tak, že v průměru algoritmus 2.3.2 nalezne ortogonální pár se skóre větším jako 0.94 téměř pětkrát rychleji než algoritmus 2.2.2 pro stejné čtverce.

Pro všechny řády je vidět, že minimální čas běhu algoritmů t_{min} je menší pro algoritmus základní. To je způsobeno tím, že v genetickém algoritmu než zkontrolujeme, že nějaký čtverec v populaci dosáhl kýžené skóre, musí v dané iteraci doběhnout základní algoritmus (řádek 3 v pseudokódu 2.3.2) pro každý čtverec v populaci. Základní algoritmus vrátí nalezenou operaci ihned.

Závěr

Ortogonální latinské čtverce jsou široce využívané v různých oblastech matematiky. Například v experimentálním designu slouží dle [6] k rovnoměrné distribuci faktorů při testování hypotéz. Výsledky této práce by tím pádem mohly najít praktické využití.

V první kapitole práce byl navržen nový pohled na dobře známý koncept latinského čtverce. V řeči zavedených pojmů byla dokázána souvislost grafu rozšířeného prostoru latinských čtverců daného řádu. Pro přehlednost rozsáhlejších důkazů byla formulována a dokázána řada pomocných lemmat. Výklad byl také doplněn o vlastní lemma 1.1.16.

Druhá kapitola se věnovala důkazů teoretických základů a návrhu metody hledání latinského čtverce ortogonálního vůči danému. Navržené algoritmy byly implementovány a podrobeny testování. Jak ukázaly experimenty, genetický algoritmus v průměru zkracuje dobu hledání ve srovnání se základním algoritmem pro čtverce řádu většího než 7.

Největší řád čtverce, pro který podařilo najít ortogonální operaci, která je latinským čtvercem, byl 11 (příklad 2.3.1), a to pomocí algoritmu genetického. O daném čtverci však víme, například z příkladu 16.2.8 v [7], že existuje alespoň 10 latinských čtverců, se kterými tvoří ortogonální dvojici. Pro obecné (náhodně vygenerované) čtverce řádu striktně většího jak 8 z testovací množiny, už se operaci se skóre 1 v rozumném čase (méně než 3 hodiny) najít nepodařilo. Možným důvodem je omezený výpočetní výkon dostupného osobního počítače. Nicméně je možné, že existují aplikace, pro které ortogonální operace, která není latinským čtvercem, ale dostatečně dobře ho ve smyslu skóre aproximuje, mohla být stále užitečná.

Seznam použité literatury

[1] OEIS Foundation Inc. The On-Line Encyclopedia of Integer Sequences, 2024. Dostupné z: `http://oeis.org`.

[2] Mark T. Jacobson and Peter Matthews. Generating uniformly distributed random latin squares. *Journal of Combinatorial Designs*, 4(6):405–437, 1996.

[3] Raúl Falcón, Lorenzo Mella, and Petr Vojtěchovský. The Hadamard multiary quasigroup product. 12 2023. Preprint poskytnutý autory.

[4] Gaston Tarry. Le problème des 36 officiers. *Compte Rendu de l'Association Française pour l'Avancement des Sciences*, 1:122–123, 1900.

[5] Judith Egan and Ian Wanless. Enumeration of MOLS of small order. *Mathematics of Computation*, 85, 06 2014.

[6] Penn State's Department of Statistics. Crossover designs. Dostupné z: `https://online.stat.psu.edu/stat503/lesson/4/4.6`.

[7] J. Morris. *Combinatorics: An Upper-Level Introductory Course In Enumeration, Graph Theory, And Design Theory.* Online access: Center for Open Education Open Textbook Library. University of Lethbridge, Curriculum Redevelopment Centre, 2017.