

Diplomová práce



České  
vysoké  
učení technické  
v Praze

**F3**

Fakulta elektrotechnická  
Katedra počítačů

## Identifikace vlastních klíčů v systému generálního a hlavních klíčů

Marek Kryška

Vedoucí: Radomír Černoch, MSc.  
Obor: Otevřená informatika  
Studijní program: Umělá inteligence  
Listopad 2017



## Poděkování

Rád bych poděkoval mému vedoucímu Radomíru Černochovi MSc., za příležitost pracovat na tomto projektu, za trpělivou ochotu při vymýšlení, testování a sepsávání tohoto textu a za jeho neskonale optimistický a nadšený přístup, který mi vždy velmi v řešení práce pomohlo.

## Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

.....  
Marek Kryška

V Praze, 12. listopadu 2017

## Abstrakt

•ežením systému klíe a zámku rozumíme fyzickou výrobu klíe a blokování v zámku. Návrh ežení musí odpovídat podmínkám, jen<sup>o</sup> jsou dané zákazníkem a íkají nám, jaké klíe otvírají které zámky v systému. Nalezení takového ežení je a<sup>o</sup> na urřitou třídou diagonálních lock-chart, jak ukazuje relevantní literatura [14], dokázaný NP -úplný problém. Tato práce se zabývá p<sup>o</sup>evodem obecného zadání z této třídou na onu urřitou třídou, jen<sup>o</sup> jsme schopni ežit polynomiální řasovou složitostí.

Práce nejprve formalizuje problém a p<sup>o</sup>edstavuje řtená<sup>o</sup>i základní vřled do notace lock-chart. Poté se sna<sup>o</sup>í najít vhodný p<sup>o</sup>evod úlohy hledání vlastního lock-chartu do třídou problému P, kde zkouří aplikovat p<sup>o</sup>evody do bipartitních graf. anebo p<sup>o</sup>evody pomocí teorie množin. Ukazuje, e problém lze p<sup>o</sup>evést na třídou P jen pro urřité typy zadání lock-chartu.

V praktické řásti pak p<sup>o</sup>edstavuje nřkolik algoritm. s p<sup>o</sup>ídanými parametry, kterými práce zkouří r-zné p<sup>o</sup>ístupy k nalezení maximálního ežení. Uvádí algoritmy konečné a optimální jako je t<sup>o</sup>eba SAT nebo r-zné stromové prohledávací struktury a algoritmy heuristické jako je metoda biklastrování a hladová kritéria. Práce ukazuje, e kombinací heuristických a optimálních algoritm. dosahujeme velmi dobrých výsledk. v hledání vlastního lock-chartu.

Klířová slova: lock-chart, klíe, zámek, diagonála

Vedoucí: Radomír řernoch, MSc.

## Abstract

Lock-chart solving (also known as general and master key solving) is a process for designing mechanical keys and locks so that every key can open and be blocked in a certain user-defined system of locks.

Solving this problem is proven as NP-hard problem except a class of diagonal lock-charts. Known literature [14] shows that diagonal lock-chart can be solved in polynomial time. This study goal is to extract diagonal lock-chart from general assignment.

Study formalizes problem and introduces basic insight into lock-chart notations.

Then it will try to find suitable conversion of eigen lock-chart to a problem from P class. There are several ways of doing it from conversion to bipartite graph to conversion by set theory, but none of them are applicable to general type of lock-chart.

In the practical chapters, the study introduces several algorithms with additional parameters, which examine different approaches of finding maximal binary diagonal. It introduces greedy and optimal algorithms such as SAT solvers or two tree-search based procedures and also shows heuristic algorithms such as biclustering method and three greedy criterias. This thesis shows, that combination of heuristic and optimal algorithms can achieve very good results in finding eigen lock-chart.

Keywords: lock-chart, key, lock, diagonal

Title translation: Finding individual keys in a master-key system

## Obsah

1 Úvod	1	4.2 Hypotéza o hladovém algoritmu	44
1.1 Pin-tumbler technologie	2	4.3 Hypotéza o biklastrování	45
1.1.1 Lock-chart	2	4.3.1 Biklastrování proti hladovému kritériu	46
1.2 Zámky v literatuře	3	4.4 Hypotézy o volbě parametru	47
1.3 Cíl práce	4	4.5 Vyhodnocení	48
2 Formalizace	5	5 Závěr	55
2.1 Důležitost matematické důkazy	5	5.1 Vylepšení do budoucna	55
2.2 Úvod do algoritmizace	6	5.1.1 Volba kritéria	55
2.3 Teorie grafů	7	5.1.2 Existence optimálního polynomiálního algoritmu	55
2.4 Lock-chart	8	5.1.3 Přizpůsobení algoritmu k jednotlivým třídám příkladů	56
2.5 Množina vlastních klíčů	10	5.1.4 Nalezení diagonály	56
2.5.1 Formální důkaz problému	11	A Literatura	57
2.5.2 Převod problému do třídy P	13	B Obrázky	59
2.5.3 Převod problému do třídy NP	15	C Obsah CD	63
3 Algoritmy	19		
3.1 Kritérium	19		
3.2 Ověření správnosti řešení	19		
3.3 Algoritmy s optimálním řešením	20		
3.3.1 SAT	21		
3.3.2 Strom podle kritéria	25		
3.3.3 Binární strom podle kritéria	28		
3.4 S heuristickým řešením	31		
3.4.1 Biklastrování	31		
3.4.2 Hladové kritérium	35		
4 Experimenty	41		
4.1 Hypotézy o SATu	42		

## Obrázky

1.1 Pro I klíče a °ezání klíče. . . . .	2	3.12 Kritérium podmnožiny s protipříkladem optimálnosti . . . .	39
1.2 Vlevo strukturovaný a vpravo nestrukturovaný lock-chart . . . . .	3	4.3 SAT konvergence k °ežení. . . . .	50
2.1 Obecný lock-chart. . . . .	9	4.6 Závislost na parametru D pro veřejná data. . . . .	51
2.2 Diagonální a vlastní lock-chart. .	10	4.9 Závislost na parametru D pro neveřejná data . . . . .	52
2.3 Lock-chart a jeho maximální vlastní řástečný lock-chart. . . . .	12	B.1 Příklad G09 konvergence k °ežení. 59	
2.4 Permutovaný lock-chart z Obrázku 2.3 a jeho maximální vlastní řástečný lock-chart. . . . .	13	B.2 Výsledky pro veřejná data. . . .	60
2.5 Lock-chart a jeho p°íslužné maximální párování (to se rovná vlastnímu lock-chartu). . . . .	14	B.3 Výsledky pro neveřejná data. . .	61
2.6 Lock-chart a jeho p°íslužné maximální párování (to se nerovná vlastnímu lock-chartu). . . . .	14		
2.7 řástečné uspořádání (s vyznačeným antichain) a p°evedení do lock-chartu . . . . .	16		
3.1 Odhad počtu klauzulí v závislosti na velikosti °ežení . . . . .	21		
3.2 Lock-chart a jeden z jeho možných stromů podle kritéria . . . . .	26		
3.3 Obecný zapsaný binární strom pro tři uzly . . . . .	29		
3.4 První dvě nalezená °ežení v binárním stromu . . . . .	30		
3.5 •ezy p°i perfektní volbě parametru r. . . . .	33		
3.6 Matice ohodnocení M p°ed seřazením. . . . .	35		
3.10 Tři iterace metody biklastrování	36		
3.11 Perfektní párování pro vlastní lock-chart. . . . .	37		

## Tabulky

4.1 Průměrné hodnoty SAT pro veřejnou sadu dat. ....	42
4.2 Průměrné hodnoty SAT pro neveřejnou sadu dat. ....	43
4.3 Počet optimálních řešení pro jednotlivé SAT instance. ....	43
4.4 SAT dol. v porovnání s ostatními algoritmy. ....	44
4.5 Porovnání prvního řešení a jeho času pro veřejná data. ....	45
4.6 Porovnání prvního řešení a jeho času pro neveřejná data. ....	46
4.7 Porovnání kritérií času BC (vlevo podle klíče a vpravo podle násobení). ....	46
4.8 Procentuální rozdíl v parametru D pro obě sady. ....	48
4.9 Průměrný první a poslední řešení pro veřejnou sadu příklad. ....	49
4.10 Rozdíl v parametru R pro neveřejnou sadu dat. ....	49
4.11 Průměrný první a poslední řešení pro neveřejnou sadu příklad. ....	53





# Kapitola 1

## Úvod

Jen málo vynálezů provází lidskou historií tak dlouho jako mechanické zámky a klíče. Jejich počátky můžeme vystopovat do období Mezopotámie a pak další zdokonalení provedené starověkými Egypťany. Během průmyslové revoluce se i zámky a klíče dočkaly svého vylepšení, především zdokonalením od Linuse Yale Sr. a později Jr. Díky ceně mají dnes mechanické zámky největší zastoupení na trhu a vyrábí se ve velkých počtech. Stačí se jen zamyslet, kolik zámků jsme museli otevřít při cestě do práce anebo kolik klíčů nám cinká v kapse u kalhot.

Zámky a klíče zaujímají v naší kultuře neochvějně místo. Od běžného každodenního předmětu, bez kterého si neumíme představit život - zapomenutí klíče doma je již ustáleným klíčem pro začátek špatného dne - nebo pro romantizovanou verzi v rámci poekadel, či nepolapitelného zloděje se zperhákem, jenž otevře cokoli. Není divu, že otvírání zámků bez originálního klíče, tedy lockpicking, dokonce existuje ve formě sportu. Už samotný francouzský král Ludvík XVI. byl vášnivým podporovatelem a účastníkem takových soutěží. Dnes se kromě kriminálních seriálů a filmů setkáme s lockpickingem třeba v počítačových hrách, kde bez této schopnosti nejste schopni projít určenými lokacemi.

Mechanické zámky a klíče se vyznačují ještě jednou zvláštností a to nejen dobou, jenž slouží lidstvu. Ačkoliv moderní svět zná nástup elektronických zámků a jiných alternativních metod zabezpečení, mechanické zámky a klíče jako by tento fakt vůbec neohrožoval, naopak na trhu stále zaujímají dominantní místo. Můžeme tvrdit, že je to snad konzervativní povahou lidí a víry v ověřený způsob bezpečnosti nebo faktem, že fyzicky vyrobený klíč je bytelný a jen těžko jej něco poničí a například po vystavení extrémnějším teplotám či pošasí (mráz, déšť, letní vedra) bude pravděpodobně stejně funkční. Reálný důvod bude praktičtější, výroba klíčů a zámků na mechanické bázi je stále nejlevnější, způsobem, ačkoliv do budoucna by se dalo očekávat, že cesta povede smížením elektronických a mechanických zámků k zvýšení kvality zabezpečení.

Obrázek 1.1: Profil klíče a ozání klíče.

## 1.1 Pin-tumbler technologie

Klasická technologie zámku typu pin-tumbler spoívá ve dvou aspektech klíče. Za prvé je to jeho profil. Ten udává, zda klíč úspěšně zajede do cylindru (dále o něm budeme hovořit pouze jako o zámku). Na Obrázku 1.1 na levé straně vidíme příklad takového zámku a klíče, jenž zapadá. Zámek v uvedeném příkladě je po stranách vytvarován do pozic 1, 4, 6, 7, 9, 10, 11, 13 a přijme každý klíč, jenž je vytvarován stejnou kombinací nebo její podmnožinou. Tedy například klíč z obrázku s tvarováním 1, 4, 9, 10, 11, 13 přijímá.

Druhý aspekt je ozání klíče. Na Obrázku 1.1 vpravo vidíme, že pokud je klíč zasunutý do zámku, tak jeho ozání zaplní tzv. piny. Jestliže se ozání klíče shoduje s délkou pinu, jsme schopni klíčem otožit v zámku a tím ho otevřít. Pin, jak je vidět i na obrázku, může být na sobě naskládáno více a tím vytváří oddělující hranici. Pokud spodní piny zaklapnou do ozání klíče a oddělující hranice je v rovině, jsme schopni klíčem otožit a zámek otevřít. Jestliže by alespoň na jedné pozici měl klíč špatné ozání, nevytvoří se oddělující hranice a zámek nebude mŕno otevřít.

### 1.1.1 Lock-chart

Bezpečnostní systém obsahuje množinu klíčů a množinu zámků. Každý klíč z množiny pak otevírá nějakou danou podmnožinu zámků. Jedním ze způsobů, jak zapsat tuto skutečnost je lock-chart. Lock-chart je v podstatě binární maticí, kde řádek odpovídá zámku, sloupec klíči a jejich pr-nik skutečnosti, zda klíč otevírá daný zámek. Jestliže má klíč otevírá pouze jeden zámek, hovoříme o něm jako o vlastním. Pokud klíč otevírá více zámků, jedná se o klíč hlavní a jestliže otevírá všechny zámků říkáme, že se jedná o klíč generální. Jestliže máme lock-chart, který pokud převedeme na strom, kde každý uzel je klíč a platí pro něj, že jestliže klíč otevře zámek, tak jej otevře i všechny

Obrázek 1.2: Vlevo strukturovaný a vpravo nestrukturovaný lock-chart

zámky v jeho implikovaném podstromu, pak mluvíme o takovém lock-chartu jako o strukturovaném. Všechny ostatní lock-charty jsou nestrukturované

## 1.2 Zámky v literatu<sup>o</sup>e

Ažkoliv jsme zmínili, že mechanické zámky jsou v naší společnosti samozřejmostí a matematický popis problému lze přesně provést, není mnoho prací které by se touto problematikou zabývalo. Společnosti, jenž vyrábějí zámky a klíče, reálně využívají kombinaci softwaru, se kterým lze počítat jen určité třídy problému a může provádět velké množství preprocesovacího výkonu, a ručního dopřítávání s kontrolou šlovice. Výsledky se může občas spíše blížit přibližným odhadem než optimálnímu řešení.

Existuje studie, která řeší problém na CSP. Ulrich Junker [13] řeší CSP do podoby hierarchie stromu, kde každý uzel je menší část problému. Jelikož se struktura skoro podobá stromu, lze problém ve velikosti stromu řešit polynomiálně a ve velikosti v něm podproblém exponenciálně.

Díležitou prací je pro nás diplomová práce Anny Lawer [1], která problém řeší na NP-úplnou úlohu. Dále provádí převod mezi rozřazeními lock-chartu a převodem na SAT. Lawer se pak jeví zaměřuje na větší bezpečnost pro zákazníka, jmenovitě tedy minimalizaci počtu klíčů, které otevrou zámek, tedy minimalizaci případu, kdy dva různé zákazníci otevrou zámek sobě navzájem.

Radomír fernoč [14] ve své disertační práci pak ukazuje, že některé třídy lock-chartů jsou polynomiálně řešitelné, zatímco k ostatním třídám musíme využít různých optimalizačních algoritmů jako jsou modifikace SATu nebo algoritmy na bázi CSP, nebo stále zůstávají ve třídě problému NP-úplných (pokud se nenajde důkaz  $P = NP$ ). fernoč využívá tzv. rotating constant method [18], mezi společnostmi hojně využívané strategie k fyzickému návrhu klíče, jenž určuje určité pozice v řezání klíče podle klíčů hlavních. Tato metoda pak umožňuje vytvořit diagonální lock-chart ve složitosti třídy P, díky převodu na problém hledání největší antichain v řádkovém uspořádání.

## ■ 1.3 Cíl práce

Hledáme tedy způsob, jak třídy NP -úplných problémů řešit efektivněji. Víme, že jsme schopni diagonální lock-charty vyřešit v polynomiálním čase [4]. Na základě myšlenky rozděl a panuj se pokusíme najít v obecném zadání tento polynomiální podproblém, po jehož vyřešení se obecné zadání zredukuje.

Úkolem se pak stává nalezení nejvčetnějšího diagonálního (v textu ukážeme, že postačí pouze vlastní) lock-chartu v obecném zadání.

Tato práce si dává za cíl několik úkolů:

1. Formalizovat tento problém v rámci existující literatury.
2. Převod úlohy na její existující problémy.
3. Najít konečný algoritmus zaručující optimální řešení a zároveň takový algoritmus, jenž vrátí řešení v polynomiálním čase.

## Kapitola 2

### Formalizace

V této kapitole si na začátku zavedeme několik definic a pojmů, ke kterým se budeme dále v textu odkazovat. Jedná se o všeobecně známé informace z oboru počítačových věd, je tedy zavedeno jen to, co se bezprostředně týká výsledků a zkoumání cílů práce. Vynecháváme zavádění základních axiomů (případně paradoxů tím vznikajících), stejně tak neřešíme všechny přípustné podoby algoritmů nebo grafů, neboť nejsou pro tuto práci relevantní. Pro čtenáře pohybující se v této oblasti by nemělo jít o nové pojmy, protože je raději pro pořádek uvádíme. Poté navážeme definicemi lock-chartů, představením problému této práce a návrhy jeho řešení, které detailně ukážeme v následující kapitole.

#### 2.1 Definice matematické definice.

V této sekci si zavedeme několik definic z oblasti diskrétní matematiky [10].

**Definice 2.1. (Množina).** Mějme konečnou množinu symbolů  $d_1, \dots, d_n$ , poté skupinu takovýchto symbolů  $f \in \{d_1, \dots, d_n\}^*$  říkáme množinou. Nechť existuje množina  $A$ . Množina  $B$ , která vznikla smazáním některých prvků z množiny  $A$  se nazývá její podmnožinou (značeno  $B \subseteq A$ ). Množiny bývají obecně značeny velkým písmenem (např.  $A, B, C$ ) zatímco její prvky ( $a \in A$ ) písmeny malými. Počet prvků množiny nazýváme její velikostí (jestliže mluvíme o prázdné množině  $\emptyset$ , pak její velikost je rovna nule). Na množinách lze provádět operace sjednocení ( $A \cup B$ ), průniku ( $A \cap B$ ) a rozdílu ( $A \setminus B$ ). Jestliže platí  $|A| > |B|$ , pak je  $B$  vlastní podmnožinou  $A$  a značíme tento vztah jako  $B \subset A$ .

**Definice 2.2. (Relace).** Mějme množinu  $X_1, \dots, X_n$  a kartézsky součin  $X_1 \times \dots \times X_n$  pak libovolnou podmnožinu tohoto kartézského součinu nazveme relací. Jestliže se  $n$  rovná  $n = 1$ , relaci nazýváme unární, jestliže  $n = 2$ , relaci nazýváme binární. Nechť je  $R$  relace na  $X \times X$ . Pro všechny  $x, x^0, x^{00} \in X$ : jestliže platí, že  $(x; x) \in R$ , pak říkáme, že je reflexivní. Jestliže  $(x; x) \in R$  neplatí pro žádné  $x$  pak nazýváme relaci ireflexivní. Pokud platí

zároveň  $(x; x^0) \in R$  a  $(x^0; x) \in R$ , což implikuje  $x = x^0$ , pak říkáme, že je relace antisymetrická. Jestliže máme  $(x; x^0) \in R$  a zároveň  $(x^0; x^{00}) \in R$ , což implikuje, že platí  $(x; x^{00}) \in R$  pak o relaci  $R$  říkáme, že je tranzitivní. Mějme relaci, která je reflexivní, antisymetrická a tranzitivní, pak o ní můžeme říci, že se jedná o řádkové uspořádání

Definice 2.3. (Funkce). Nechť jsou  $X$  a  $Y$  množiny a mějme  $f : X \rightarrow Y$ . Pak o  $f$  jako o relaci na  $X \times Y$  můžeme hovořit jako o funkci, pokud  $(x; y) \in f$ . Funkce je prostá jestliže pro libovolná  $x_1, x_2 \in X$ , které se nerovnájí  $x_1 \neq x_2$ , platí, že  $f(x_1) \neq f(x_2)$ . Funkce je surjektivní, jestliže pro každý prvek cílové množiny  $y \in Y$  existuje nějaký vzor  $x \in X$ . Jestliže je funkce zároveň prostá a surjektivní, říkáme, že se jedná o bijekci. Pokud máme dvě zobrazení  $f : X \rightarrow Y$  a  $f^{-1} : Y \rightarrow X$ , pak říkáme, že  $f^{-1}$  je inverzní zobrazení k funkci  $f$ , jestliže platí pro všechny dvojice  $(x; y)$ , kde  $x \in X$  a  $y \in Y$ , že  $f^{-1}(f(x)) = x$  a  $f(f^{-1}(y)) = y$ .

## 2.2 Úvod do algoritmizace

Jelikož výstupem této práce je mimo jiné i řada r-zných algoritmů, tedy přístup jak daný problém vyřešit, zavádíme řadu pár základních definic z oblasti algoritmizace. Ta nám dá malý pohled o tom, jaké jsou přístupy k řešení daných výpočetních problémů a r-zná úskalí, které mohou při jistých metodách řešení nastat [11].

Definice 2.4. (Konečný algoritmus). Pod pojmem algoritmus rozumíme postup k řešení nějakého obecného problému. Jak konečný algoritmus nazveme takový algoritmus, který pro libovolně velké množství vstupních údajů v konečném počtu kroků skončí.

Když mluvíme v této práci o nějakém algoritmu  $a$  (jelikož se zaměříme na speciální podmnožinu všech možných typů algoritmů), tak má význam jako nějaká procedura, která vrátí celé číslo. Nemá smysl očekávat jiný výstup, než celé kladné číslo v případě existujícího řešení nebo nula v případě, že algoritmus nenalezl řešení.

(Optimální a řádkové řešení). Nechť existuje algoritmus  $a \in A$ , kde  $A$  je množina r-zných algoritmů a funkce  $f$  nám říká návratovou hodnotu algoritmu k danému problému. Měkněme o výsledku funkce  $f(a)$ , že je řešením pokud  $f(a) > 0$  a hovoříme o něm jako o optimálním, pokud pro všechny ostatní algoritmy řešící stejný problém, tedy všechny  $a^0 \in A$  a  $a \neq a^0$ , platí, že  $f(a) > f(a^0)$ . Jestliže platí, že  $f(a) > 0$  pak hovoříme o tom, že algoritmus  $a$  dává řádkové řešení. V případě  $f(a) = 0$  říkáme, že algoritmus nevrací řádkové řešení.

(Heuristický algoritmus). Mějme algoritmus, měkněme o něm, že je to heuristický algoritmus jestliže není zaručeno, že nalezne optimální řešení, ale pouze vhodné přiblížení k řešení. Důvody k upřednostnění takového algoritmu

před jiným optimálním (nalezneme vždy správné řešení) může být časová, či paměťová složitost.

(Asymptotická složitost). Při řešení problémů výpočetní techniky nás zajímá, jak porovnat efektivitu a rychlost různých algoritmů. Ke klasifikaci algoritmů se obvykle používá tzv. asymptotická složitost, což je rozdělení algoritmů do tříd složitostí, u kterých platí, že od určité velikosti dat, je algoritmus dané třídy vždy pomalejší než algoritmus třídy předchozí, bez ohledu na to, jestli je některý z požadovaných násobných výkonových (c je zde libovolná konstanta). Řekněme, že v nekonečnu slouží k rozlišení jednotlivých tříd. Řekněme, že pokud se n blíží k nekonečnu, tak neexistuje reálná konstanta taková, aby byl algoritmus z vyšší třídy rychlejší než ten z třídy předchozí.

1    $\log(n)$     $n$     $n \log(n)$     $n^k$     $k^n$     $n!$     $n^n$

Standardně se asymptotické složitosti zapisuje pomocí Landauovi notace (např.  $O(n^2)$  pro kvadratickou funkci).

Definice 2.5. (Třída složitosti P a NP). Řekněme, že rozhodovací úloha  $U_1$  (tj. taková, jenž má binární výstup "ANO", "NE") leží ve třídě P, jestliže existuje deterministický Turingův stroj, který rozhodne jazyk  $L_{U_1}$  a pracuje v polynomiálním čase. Řekněme, že rozhodovací úloha  $U_2$  leží ve třídě NP, jestliže existuje nedeterministický Turingův stroj, který rozhodne jazyk  $L_{U_2}$  a pracuje v polynomiálním čase.

Alternativně lze NP problémy definovat tak, že je to množina problémů, u kterých lze pro daný výsledek v polynomiálním čase ověřit jeho správnost (ale obecně nikoliv nalézt řešení v polynomiálním čase). Dodejme ještě poznámku, že platí  $P \subseteq NP$  a otázka platnosti rovnosti  $P = NP$  je stále otevřená a nevyřešená.

## 2.3 Teorie grafů

V této sekci zavedeme několik formalismů týkajících se teorie grafů, neboť tyto nám slouží jako mocný nástroj nejen při vizualizaci logických chartů, ale i při jejich řešení za předpokladu převodu úlohy na podobnou v oblasti teorie grafů. [8]

Definice 2.6. (Orientovaný graf). Orientovaný graf  $G$  je trojice  $(V; E; \rightarrow)$ , kde  $V$  je neprázdná konečná množina vrcholů (též zvaných uzlů),  $E$  je konečná množina jmen hran (též zvaných orientovaných hran) a  $\rightarrow$  je přiřazení, které každé hraně  $e \in E$  přiřazuje uspořádanou dvojici vrcholů a nazývá se vztah incidence.

Uveďme pro doplnění ještě další pojmy spojené s orientovanými grafy. Jestliže platí  $(e) = (u; v)$  pro  $u, v \in V$ , potom o vrcholu  $u$  říkáme, že je počáteční vrchol hrany  $e$  a vrchol  $v$  je koncový vrchol hrany  $e$ ; značíme  $PV(e) = u$  a  $KV(e) = v$ . O vrcholech  $u, v$  říkáme, že jsou krajní vrcholy hrany  $e$ , též že jsou incidentní s hranou  $e$ .

De nice 2.7. (Neorientovaný graf). Neorientovaný graf je trojice  $G = (V; E)$ , kde  $V$  je neprázdná konečná množina vrcholů (též zvaných uzelů),  $E$  je konečná množina jmen hran a je pořazení, které každé hraně  $e \in E$  přiřazuje množinu  $f_u; v$  (kde  $u, v \in V$  jsou vrcholy) a nazývá se vztah incidence.

V dalších notacích pro zjednodušení vynecháme vztah incidence při zápisu grafu. Pro účely této práce a dalších formalismů se nebude jeho de nice nijak měnit.

De nice 2.8. (Matice sousednosti). Mějme konečnou množinu vrcholů grafu  $G = (V; E)$ , kterých je  $n$  a čtvercovou matici  $A = n \times n$ , jejíž hodnota na místě  $a_{vu}$  je celé číslo odpovídající počtu hran vedoucích z vrcholem  $u$  do vrcholu  $v$ , pak takové matici říkáme matice sousednosti. Prvky na diagonále tak obvykle odpovídají počtu hran vedoucích z vrcholu  $v$  zpět do vrcholu  $v$ .

De nice 2.9. (Stupeň vrcholu/uzlu). Mějme daný orientovaný graf  $G = (V; E)$ . Vstupní stupeň vrcholu  $v$ , značíme  $\text{je}d(v)$ , je roven počtu hran, pro které je  $v$  koncovým vrcholem (které do vrcholu  $v$  vstupují), tj.  $\text{je}d(v) = |\{e \in E; KV(e) = v\}|$ . Výstupní stupeň vrcholu  $v$ , značíme  $\text{je}d^+(v)$ , je roven počtu hran, pro které je  $v$  počátečním vrcholem (které z vrcholu  $v$  vystupují), tj.  $\text{je}d^+(v) = |\{e \in E; PV(e) = v\}|$ . Stupeň vrcholu  $v$ , značíme  $\text{je}d(v)$ , je roven počtu hran, které jsou incidentní s vrcholem  $v$ , tj.  $\text{je}d(v) = \text{je}d(v) + \text{je}d^+(v)$ . Je dán neorientovaný graf  $G = (V; E)$ . Stupeň vrcholu  $v$ , značíme  $\text{je}d(v)$ , je roven počtu hran, které jsou incidentní s vrcholem  $v$ .

De nice 2.10. (Bipartitní graf). Nechť existuje graf  $G = (V; E)$  takový, že  $V$  lze rozdělit na dvě množiny  $V_1$  a  $V_2$  tak, že platí  $V = V_1 \cup V_2$  a  $V_1 \cap V_2 = \emptyset$ . Pak o grafu  $G$  říkáme, že je bipartitní.

De nice 2.11. (Souvislý graf). Nechť  $G = (V; E)$  je graf a  $v, u \in V$  jsou jeho dva libovolné vrcholy. Jestliže existuje cesta mezi každými dvěma vrcholy  $v$  a  $u$ , pak říkáme, že takový graf  $G$  je souvislý.

De nice 2.12. (Kružnice a cyklus). Mějme orientovaný či neorientovaný graf  $G = (V; E)$ . Jestliže existuje v grafu  $G$  cesta (tj. sled vrcholů a hran, které se v cestě neopakují) a je uzavřená (začíná ve stejném vrcholem, kde i končí) pak o této cestě říkáme, že je cyklus, pokud je graf orientovaný, anebokružnice pokud se jedná o graf neorientovaný.

De nice 2.13. (Strom). Mějme orientovaný či neorientovaný graf  $G = (V; E)$ . Řekněme o něm, že se jedná o strom pokud je souvislý a neobsahuje kružnici.

## 2.4 Lock-chart

Tato sekce se zabývá de nicí lock-chartu. Ten vzniká z potřeby mít formu v jaké zapsat vazby a podmínky mezi klíči a zámky. Lock-chart nám tedy slouží jako zadání, které může nabývat mnoho podob. Uvedeme několik typu lock-chartů včetně několika, jenž se přímo netýkají této práce. Důvod existence tolika druhů lock-chartů je dán nejednotností postupů výrobců při zadávání a zpracování klíčových systémů.



Obrázek 2.1: Obecný lock-chart.

Definice 2.14. (Obecný lock-chart). Obecným lock-chartem rozumíme trojici  $(K; L; E)$ .  $K$  je množina klíčů (keys) a  $L$  je množina zámků (locks), které jsou navzájem disjunktní  $L \cap K = \emptyset$ .  $E$  je pro nás binární relace na  $L \times K$  a nazýváme ji množinou hran (edges).

Existuje přímá spojitost mezi lock-chartem a grafem  $G$ . Každý lock-chart  $(K; L; E)$  je z definice také neorientovaným bipartitním grafem  $G(K; L; E)$  s nezávislými množinami  $K$  a  $L$ . Z toho lze zjednodušit definici  $E$  jako  $E(k) = \{l \in L \mid (k, l) \in E\}$ ,  $E(l) = \{k \in K \mid (k, l) \in E\}$ . Tedy platí, že pokud  $(k, l) \in E$  pak můžeme říct, že "klíč  $k$  otvírá zámek  $l$ ". V opačném případě říkáme, že "zámek blokuje klíč  $k$ ".

V textu budeme obecný lock-chart díky jeho rozšířenosti a univerzálnosti nazývat zjednodušeně pouze jako lock-chart.

Příklad 2.15. Obrázek 2.1 na straně 9 znázorňuje obecný lock-chart. Sloupce reprezentují klíče a řádky reprezentují zámky. Jestliže je políčko matice zabarvené černě značí to, že daný klíč ve sloupci otvírá příslušný zámek v řádku. Formálně bychom údaje z matice zapsali jako:

$$\begin{aligned} K &= \{g; m_1; m_2; m_3; k_1; k_2; k_3; k_4; k_5; k_6\} \\ L &= \{l_1; l_2; l_3; l_4; l_5\} \\ E &= \{(g; l_1); \dots; (m_1; l_1); (m_1; l_2); \dots; (k_1; l_1); (k_2; l_2); \dots\} \end{aligned}$$

Tedy z definice platí následující tvrzení:  $E(g) = L$ ,  $E(m_1) = \{l_1; l_2\}$  a  $E(k_i) = E(l_i)$  pro  $1 \leq i \leq 6$ .

Definice 2.16. (Generální, hlavní a vlastní klíč). Nechť je  $K$  množina klíčů,  $L$  množina zámků,  $E$  množina hran a  $k \in K$ . Jestliže  $|E(k)| = 1$  pak  $k$  je nazýván vlastním klíčem. Jestliže  $|E(k)| > 1$  pak o  $k$  říkáme, že je hlavním klíčem. A pokud platí, že  $|E(k)| = |L|$  pak o  $k$  mluvíme jako o klíči generálním.

Lock-chart na Obrázku 2.1 na straně 9 obsahuje jeden generální klíč, tři hlavní klíče  $m_1, m_2, m_3$  a šest vlastních klíčů od  $k_1$  do  $k_6$ .

Pravděpodobně nejjednodušším případem lock-chartu s generálním klíčem je diagonální lock chart. Každý takový lock-chart je pak definován pouze

Obrázek 2.2: Diagonální a vlastní lock-chart.

jediným parametrem a to počtem jeho vlastních klíčů.

Definice 2.17. (Diagonální lock-chart). Jestliže lock-chart obsahuje pouze jediný generální klíč a žádný další hlavní klíč, nazýváme pak takový lock-chart diagonálním.

Levý lock-chart na Obrázku 2.2 na straně 10 je diagonální. Má pět klíčů  $K = \{g; k_1; k_2; k_3; k_4\}$  a čtyři zámky  $L = \{l_1; l_2; l_3; l_4\}$ . Každý vlastní klíč  $k_1, k_2, k_3, k_4$  otevírá právě jeden zámek  $E(k_i) = \{l_i\}$  a generální klíč otevírá všechny zámky  $E(g) = L$ .

Pokud odstraníme z tohoto typu lock-chartu generální klíč dostaneme ještě jednodušší lock-chart. Ten se sice v praxi příliš nepoužívá, ale pro tuto práci je stejnější.

Definice 2.18. (Vlastní lock-chart). Jestliže lock-chart neobsahuje žádný hlavní klíč pak o něm hovoříme jako vlastním lock-chartu.

Příklad vlastního lock-chartu najdeme na pravé straně v Obrázku 2.2 na straně 10.

Pro úplnost ještě zavádíme definici pro vztah dvou lock-chartů ve smyslu podmnožiny. Vlastní lock-chart z Obrázku 2.2 na straně 10 je zároveň řádkovým lock-chartem lock-chartu diagonálního na stejném obrázku vlevo.

Definice 2.19. (Řádkový lock-chart). Nechť  $T = \{K, L, E\}$  je lock-chart, pak  $T = \{K, L, E\}$  je řádkovým lock-chartem k lock-chartu  $T$  pokud platí, že  $K \subseteq K, L \subseteq L$  a  $E \subseteq E$ . Tedy píšeme, že  $T \subseteq T$ .

## 2.5 Množina vlastních klíčů.

Cílem této práce je extrahování řádkového lock-chartu z obecného zadání bez přidání podmínek, tak aby splňoval podobu maximálního lock-chartu vlastního. Tedy pokud si představíme zadaný lock-chart jako binární matici, naším úkolem je najít největší diagonálu zámek-otvíraných klíčů.

Nejnaivnějším přístupem k nalezení takové diagonály by bylo přeskupení zadaného lock-chartu do podoby obecného lock-chartu z Obrázku 2.1 na straně 9. V takovém případě by stažilo pouze "ukrojit" pravou část matice. Problémy ale v této metodě nastávají dva a jak se dále v práci ukáže jsou to problémy klíčové (jak ve smyslu tématu práce tak v důležitosti). První z problémů je počet takových kombinací lock-chartu. Museli bychom vyzkoušet  $|K|^{|L|}$  množin, což by znamenalo exponenciální růst počtu kombinací v závislosti na vstupní matici.

De nice 2.20. (Permutace). Nechť je  $n, k \in \mathbb{W}$ . Faktoriál je funkce  $n! = n(n-1)\dots 2 \cdot 1$ . Počátek řady je de nován jako  $0! = 1$ . Jako binomický koeficient pak nazýváme  $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ , který říká počet k voleb z  $n$  objektů. Jinými slovy tuto funkci můžeme nazývat jako permutaci.

Druhý problém je praktičtější. Reálné zadání lock-chartu totiž nemusí být "hezky" zadané jako v doposud uvedených obrázcích, ale může nabývat až skoro náhodných tvarů. Toto je zapříčiněno lidským zásahem do tvorby lock-chartu. Ten se totiž nemusí zapisovat postupným systematickým řazením hlavních a vlastních klíčů. Zároveň v obecném zadání předpokládáme existenci hlavních a centrálních vložek.

De nice 2.21. (Hlavní zámek a centrální vložka). Nechť je  $K$  množina klíčů,  $L$  množina zámků,  $E$  množina hran a  $k \in K$ . Jestliže  $j \in E(l)_j = 1$  pak  $l$  je nazýván vlastním zámkem. Jestliže  $j \in E(l)_j > 1$  pak o  $k$  říkáme, že je hlavním zámkem. A pokud platí, že  $j \in E(l)_j = |K|$  pak o  $l$  mluvíme jako o centrální vložce.

Centrální vložka a hlavní zámky jsou vlastně takovou obdobou hlavních a generálních klíčů jen z opačné dimenze lock-chartu a to z množiny zámků.

### 2.5.1 Formální de nice problému

(Přeznění vlastního klíče a zámku). Z de nicí uvedených výše víme, že vlastní klíčem rozumíme takový klíč  $k \in K$ , pro který platí  $j \in E(k)_j = 1$ . Jestliže platí, že  $E(k) = l$  a zároveň  $j \in E(k)_j = 1$  pro nějaký zámek  $l \in L$  pak říkáme, že  $k$  je vlastním klíčem zámku  $l$ . V opažném případě tvrdíme, že  $k$  není vlastním klíčem žádného zámku z  $L$ . Obdobně tvrzení platí pokud máme  $E(l) = k$  a zároveň  $j \in E(l)_j = 1$ , pak říkáme, že  $l$  je vlastním zámkem klíče  $k$ . V opažném případě tvrdíme, že  $l$  není vlastním zámkem žádného klíče z  $K$ .

De nice 2.22. (Přeznění). Mějme lock-chart  $T = (K; L; E)$ . Nechť existuje množina dvojic klíčů a zámků  $\sigma = f(k_i; l_j); \dots; g$ . Mějme, že pro každou dvojici  $(i; j)$  v množině  $\sigma$  platí, že  $E(k_i) = l_j \wedge E(l_j) = k_i$ , což znamená, že všechny dvojice klíčů a zámků  $(k_i; l_j)$  v  $\sigma$  tvoří bijekci. Jestliže pro všechny  $(k_i; l_j) \in \sigma$  platí, že  $E(k_i) = E(l_j) = 1$  pak říkáme, že  $\sigma$  je přezněním problému hledání vlastních klíčů (vlastního lock-chartu).

De nice 2.23. (Optimální přeznění vlastních klíčů). Nechť máme množinu všech přeznění  $O = f \sigma_1; \sigma_2; \dots; \sigma_n$ , kde  $\sigma_i$  je jedno přeznění pro  $i \in \mathbb{N}$ . Počtu dvojic

Obrázek 2.3: Lock-chart a jeho maximální vlastní řádkový lock-chart.

přiznání vlastního klíče k zámku; j říkáme velikost řešení. Řekneme, že řešení je optimální pokud se rovná  $\alpha_{\max} = \max\{j_1; j_2; \dots; j_n\}$ .

Vidíme, že optimální řešení je takové, které najde bijekci vlastních klíčů k vlastním zámkům a je zároveň bijekcí maximální.

Definice 2.24. (řádkové a sloupcové řešení vlastních klíčů). Nechť máme dvě řešení  $\alpha$ ,  $\alpha_{\max}$ , kde  $\alpha_{\max}$  je optimálním řešením. Jestliže  $j_{\alpha} > 0$  a zároveň  $\alpha_{\max} > 0$  pak říkáme, že se jedná o řádkové řešení. Pokud  $j_{\alpha_{\max}} = 0$ , pak říkáme, že zadání nemá řádkové řešení.

Příklad, že by zadání nemělo řešení je pouze teoretický. Reálně by mohl nastat pouze v případě, že lock-chart nemá řádkové zámky otvírané klíči a takové zadání je z principu nesmyslné. Jakmile existuje alespoň jeden případ, kdy klíč otevírá zámek, tak se tento případ rovná řešení o velikosti  $j_{\alpha} = 1$ . Stejně tak nemá uvažovat v případě řešení o velikosti  $j_{\alpha} = 1$  o dalších řádkových řešeních, neboť řešení velikosti je nejmenší možné.

Poznamenejme ještě, že optimálních řešení může být více, pak o řádkovém řešení hovoříme v souvislosti s tím optimálním řešením ke kterému se vztahuje. Obecně neplatí, že každé řádkové řešení je podmnožinou každého optimálního. Příklad 2.25. Mějme dané zadání lock-chartu z Obrázku 2.3 na straně 12. Vlevo je zadání od výrobce a vpravo je vyznačená maximální množina vlastních klíčů. červená barva značí přiznání vlastního klíče jeho příslušnému zámku.

Maximálním řešením problému hledání maximální množiny vlastních klíčů je množina  $\alpha_{\max} = \{(k_1; l_5); (k_2; l_1); (k_4; l_2); (k_6; l_6)\}$ .

Velikost řešení je  $j_{\alpha_{\max}} = 4$ .

Pro ilustraci je ještě uveden Obrázek 2.4 na straně 13 stejného problému, jen s permutovanými řádky a sloupci matice pro lepší přehlednost řádkové a jednoduchým vizuálním ověřením správnosti řešení.

Jde si všimnout toho, že maximální, tedy optimální, řešení nemusí být

Obrázek 2.4: Permutovaný lock-chart z Obrázku 2.3 a jeho maximální vlastní řástečný lock-chart.

nutný pouze jen jedno v celém zadání.

Například množina dvojic  $f(k_1; l_5); (k_2; l_1); (k_4; l_2); (k_6; l_4)g$  je také optimálním řešením.

## 2.5.2 Převod problému do třídy P

Můžeme si položit otázku, zda jde nalezení maximálního vlastního lock-chartu převést na jiný výpočetní problém. Exponenciální  $r$ -st složitosti daný permutacemi zadaného lock-chartu není nijak lákavý k prozkoumání. Převodem na "jednodušší" úlohu bychom získali lepší představu o časové složitosti problému od které se dá dále optimalizovat. Jako první se vybíjí maximální párování [2] díky možnému převodu lock-chartu na bipartitní graf z de nice obecného lock-chartu. Zároveň víme, že hledání maximálního párování lze převést na úlohu toku v sítích a jsme takovou úlohu schopni řešit v polynomiálním čase.

De nice 2.26. (Maximální párování). Nechť  $G = (V; E)$  je neorientovaný graf. Množina  $M \subseteq E$  se nazývá párováním grafu  $G$ , pokud žádné dvě hrany v  $M$  nemají společný vrchol. Prázdná množina je pak párováním na každém grafu.

Maximální párování grafu je takové párování, které má nejvíce hran. Graf může mít několik různých maximálních párování.

Pro úplnost je třeba uvádíme de nice perfektního párování, neboť se jedná o speciální případ, kdy zadaný lock-chart je již vlastní.

De nice 2.27. (Perfektní párování). Nechť  $G = (V; E)$  je neorientovaný graf. Perfektní párování grafu je párování, které pokrývá všechny vrcholy grafu. Grafy s lichým počtem vrcholů nemohou mít perfektní párování. Každé perfektní párování je zároveň maximálním párováním.

Na Obrázku 2.5 na straně 14 vidíme, že takový převod je perspektivní. Vyznačené maximální párování je skutečně naší hledanou maximální dia-

Obrázek 2.5: Lock-chart a jeho příslušné maximální párování (to se rovná vlastnímu lock-chartu).

Obrázek 2.6: Lock-chart a jeho příslušné maximální párování (to se nerovná vlastnímu lock-chartu).

gonálou. Platí tedy, že každé maximální párování se rovná maximálnímu vlastnímu lock-chartu? Tedy, odpovídá každé maximální párování nějakému optimálnímu řešení úlohy? Odpověď na tuto otázku je bohužel záporná. Není tak těžké najít protipříklad viz Obrázek 2.6 na straně 14.

Zkusíme převod grafu, tedy nějakého uspořádání vrcholů, na podobu binární matice, tedy lock-chartu. Chceme najít v tomto uspořádání nějakou vlastnost, která by byla ekvivalentní s hledáním našeho vlastního řádkového lock-chartu. V matematických denticích jsme si v úvodu této kapitoly pověděli o řádkovém uspořádání. To je relací, která je zároveň reexivní, antisymetrická a tranzitivní. Pokud bychom předpokládali, že zadání splňuje vlastnosti řádkového uspořádání, jako vhodná volba ekvivalence s vlastním lock-chartem se jeví nalezení antichain podmnožiny.

Definice 2.28. (Antichain). Nechť  $S$  je množina vrcholů řádkového uspořádání. Měkneme o jeho podmnožině  $A \subseteq S$ , že je antichain pokud nejsme schopni řádkové dva vrcholy porovnat.

Jinými slovy platí to, že mezi každými dvěma vrcholy z podmnožiny  $A$  neexistuje řádková relace. [16]

Graf převedeme na matici následovně: Nejdříve vytvoříme matici sousednosti pro zadaný graf a poté tuto matici interpretujeme jako lock-chart, kde řádky a sloupce odpovídají stejným vrcholům ze zadaného grafu.

Jelikož je množina reaktivní, vzniká nám v lock-chartu diagonála. Z vlastnosti antisymetrie máme zaručeno, že po seřazení řádků a sloupců vznikne trojúhelníková matice, tedy máme jisté, že bude v nejhorším případě i čtvercová podmnožina řádků a sloupců ve tvaru trojúhelníkové matice a z dříve zmíněné množiny antichain víme, že každé dva uzly z její množiny jsou na sobě nezávislé, v tomto případě nejsou spojeny hranou. To znamená, že pokud políčko matice má nenulovou hodnotu a není prvkem diagonály, pak víme, že pro daný řádek a sloupec tohoto políčka, tedy dvou různých vrcholů, platí, že oba dva nemohou být přítomny v antichain množině. Maximální antichain podmnožina je skutečně ekvivalentem maximálního vlastního číselného lock-chartu.

Příklad 2.29. Na Obrázku 2.7 na straně 16 vidíme vpravo zadání, které splňuje podmínky číselného uspořádání. Vlevo na stejném obrázku je pak jeho převod do podoby grafu a červeně vyznačená maximální podmnožina antichain.

Maximálním řešením problému hledání maximální množiny vlastních klíčů je množina  $\alpha_{\max} = \{ (k_3; l_3); (k_5; l_5); (k_7; l_7) \}$ . Velikost řešení je  $|\alpha_{\max}| = 3$ .

Vzímáme si však jedné zvláštnosti na rozdíl od převodu na maximální párování uvedené výše. Graf je orientovaný a neobsahuje uzel pro každý klíč a každý zámek, ale pouze pro každou dvojici  $k_i$  a  $l_i$  pro  $i \in \mathbb{N}$ , kde se dané indexy  $i$  rovnají. Toto je zpravidla po číselném předpokladem, že zadání splňuje podmínky číselného uspořádání.

Je například lock-chart na Obrázku 2.3 na straně 12 číselné uspořádání? Díky permutaci na Obrázku 2.4 na straně 13 víme, že není (stačí si zkontrolovat, že daná matice nemůže být nikdy trojúhelníková). Ale zde přesně leží problém převodu. Za prvé se omezujeme jen na určitou třídu zadání lock-chartu (obecně nemusí být v podobě číselného uspořádání) a za druhé abychom ověřili, že zadání je skutečně číselným uspořádáním, museli bychom opět provést počet permutací, který roste exponenciálně s velikostí zadaného vstupu.

### ■ 2.5.3 Převod problému do třídy NP

Jelikož nejsme schopni převést úlohu na jinou ze třídy P, což by značně usnadnilo výpočetní složitost, provedeme alespoň převod na SAT, který nám pak poslouží jako dolní odhad při tvorbě nových algoritmů.

Dejme 2.30. (Konjunktivní normální forma). Mějme výrokovou proměnnou  $x$ , její určené literály jsou pak  $x$  a  $\neg x$ . Jako klauzuli označujeme disjunkci literálů (značíme  $x \vee y$  pro literály  $x$  a  $y$ ) a pokud máme konjunktivi klauzulí (značíme  $a \wedge b$  pro klauzule  $a$  a  $b$ ), pak o takové konjunktivi říkáme, že je to formule v CNF neboli v konjunktivní normální formě. Velikost CNF značíme

Obrázek 2.7: řádkové uspořádání (s vyznačeným antichain) a převedení do lock-chartu

jako  $\{C\}$  pro množinu všech klauzulí  $C$ .

SAT (Boolean satisfiability problem) je kombinatorický problém, který se snaží vyřešit formuli. To znamená, že pro nějakou formuli  $F$  ve tvaru CNF, jenž obsahuje konjunkci klauzulí, se snažíme najít ohodnocení tak, aby byly všechny klauzule splněné. Úloha leží ve třídě NP-úplných problémů (dokázána Cook-Levinovou větou [9], [15] v sedmdesátých letech minulého století). Pokud jsou všechny klauzule v zadání maximálně o velikosti  $n$  literálů, hovoříme o  $n$ -SAT [12].

Samotný převod problému provedeme následovně: Mějme zadaný lock-chart, tedy množinu klíčů  $K$ , množinu zámků  $L$  a množinu hran  $E$ . Jako množinu proměnných vezmeme všechny klíče, zámkové pozice v lock-chartu kde nějaký klíč otevírá zámek.

Tedy formálně:

$\text{Lit} = \{k_1; k_2; \dots; k_n; l_1; l_2; \dots; l_m; e_{1,1}; e_{1,2}; \dots; e_{1,m}; e_{2,1}; \dots; e_{n,m}\}$ , kde  $i, j$  jsou indexy klíčů a zámků a  $e_{i,j} \in E$  je hrana mezi daným klíčem a zámkem  $k_i \in K, l_j \in L$ .

Klauzule pak máme následujících tvarů:

1. Pokud vybereme klíč  $k \in K$  a zámek  $l \in L$  do řešení a platí, že  $E(k) = l$  a  $E(l) = k$ , pak říkáme, že dvojice  $(k; l)$  je součástí řešení.
2. Jestliže máme hranu  $e \in E$  v řešení, která je daná jako  $E(k) = l$ , kde je  $k \in K$ , pak součástí řešení je  $k$ .
3. Jestliže máme hranu  $e \in E$  v řešení, která je daná jako  $E(l) = k$ , kde je  $l \in L$ , pak součástí řešení je  $l$ .
4. Pokud vybereme klíč  $k \in K$  do řešení, pak musí být alespoň jedna hrana z něj vedoucí v řešení.



5. Pokud vybereme zámeček  $2 \in L$  do  $\circledast$ zení, pak musí být alespoň jedna hrana z  $n\check{r}j$  vedoucí v  $\circledast$ zení.
6. Jestliže máme hranu  $e \in E$  v  $\circledast$ zení a existuje jiná hrana  $e' \in E$ , která se shoduje v jednom z krajních uzlů s uzlem vedoucím z  $e$ , pak taková hrana  $e'$  není součástí  $\circledast$ zení.

Přepis klauzulí je pak následující pro  $1 \leq i, u \leq j \in K$  a  $1 \leq j, h \leq j \in L$ :

1.  $k_i \leq l_j - e_{i,j}$  pro  $8i, j$ .
2.  $e_{i,j} \leq k_i$  pro  $8i, j$ .
3.  $e_{i,j} \leq l_j$  pro  $8i, j$ .
4.  $k_i \leq e_{i,1} \leq \dots \leq e_{i,j} \leq l_j$  pro  $8i, j$ .
5.  $l_j \leq e_{1,j} \leq \dots \leq e_{K,j}$  pro  $8i, j$ .
6.  $e_{i,j} \leq e_{i,h}$  pro  $8i, j; h$ , kde  $j \in h$ .
7.  $e_{i,j} \leq e_{u,j}$  pro  $8i, j; u$ , kde  $i \in u$ .

Mějme na paměti, že ne všechny typy klauzulí se musí v lock-chartu vyskytovat. Pokud bychom měli třeba zadaný lock-chart vlastní, tak pak nemá poslední dvě pravidla smysl generovat. Množinu takto vygenerovaných klauzulí nazveme  $Cl_s$ .

Příklad 2.31. Vezměme jako zadání lock-chartu Obrázek 2.5 na straně 14. Formule pro převod úlohy budou vypadat následovně:

- |  |  |   |
|--|--|---|
| <ol style="list-style-type: none"> <li>1. <math>k_1 \leq l_1 - e_{1,1}</math></li> <li><math>k_2 \leq l_2 - e_{2,2}</math></li> <li><math>k_1 \leq l_3 - e_{1,3}</math></li> <li><math>k_3 \leq l_3 - e_{3,3}</math></li> <li><math>k_3 \leq l_4 - e_{3,4}</math></li> </ol> | <ol style="list-style-type: none"> <li>3. <math>e_{1,1} \leq l_1</math></li> <li><math>e_{2,2} \leq l_2</math></li> <li><math>e_{1,3} \leq l_3</math></li> <li><math>e_{3,3} \leq l_3</math></li> <li><math>e_{3,4} \leq l_4</math></li> </ol> | <ol style="list-style-type: none"> <li>5. <math>l_1 \leq e_{1,1}</math></li> <li><math>l_2 \leq e_{2,2}</math></li> <li><math>l_3 \leq e_{1,3} - e_{3,3}</math></li> <li><math>l_4 \leq e_{3,4}</math></li> </ol> |
| <ol style="list-style-type: none"> <li>2. <math>e_{1,1} \leq k_1</math></li> <li><math>e_{2,2} \leq k_2</math></li> <li><math>e_{1,3} \leq k_1</math></li> <li><math>e_{3,3} \leq k_3</math></li> <li><math>e_{3,4} \leq k_3</math></li> </ol>                               | <ol style="list-style-type: none"> <li>4. <math>k_1 \leq e_{1,1} - e_{1,3}</math></li> <li><math>k_2 \leq e_{2,2}</math></li> <li><math>k_3 \leq e_{3,3} - e_{3,4}</math></li> </ol>   | <ol style="list-style-type: none"> <li>6. <math>e_{1,1} \leq e_{1,3}</math></li> <li><math>e_{3,3} \leq e_{3,4}</math></li> <li><math>e_{1,3} \leq e_{3,3}</math></li> </ol>                                      |

Můžeme si ověřit, že červeně vyznačené  $\circledast$ zení na straně 14 je skutečně i maximálním  $\circledast$ zením těchto klauzulí. Stejně tak protipříklad zvedený u Obrázku 2.6 na straně 14 již díky převodu na SAT nehrozí být pokládán jako  $\circledast$ zení.

## 2. Formalizace

Máme sice správně formálně přešpaný název problém do úlohy typu SAT, ale pokud se pozorně podíváme na vygenerovaná pravidla, řádně nám nezaručuje případ, kdy jsou všechny klauzule splnitelné. Šádný literál totiž nenabývá pravdivé hodnoty, tedy máme řešení, avšak velikosti nula (jak jsme ji výše devalovali, korektně toto nepovažujeme ani za řešení).

Proto je potřeba ještě zavést další sadu klauzulí, které nám zaručí, že nalezneme maximální řešení a řádné jiné (menší). Je nutné SAT jako takový iterovat přes permutace výběru klíče z celkových  $jKj$  pro ověření existence řešení velikosti  $h$ .

Příklad 2.32. Vezmeme ještě jednou lock-chart z Obrázku 2.5 na straně 14. Jelikož má zadání tři klíče, tedy  $jKj = 3$ , budeme SATem iterovat velikost řešení od 1 do 3. To znamená, že pro každou iteraci budeme přidávat klauzule o velikosti  $v_i = jKj - i$  pro  $i = 0; 1; 2$ .

V každé iteraci přidáme příslušné omezující klauzule do zadání úlohy SAT:

- |                     |                 |                 |
|---------------------|-----------------|-----------------|
| 1. (Iterace 1.)     | 2. (Iterace 2.) | 3. (Iterace 3.) |
| $k_1 \_ k_2 \_ k_3$ | $k_1 \_ k_2$    | $k_1$           |
|                     | $k_2 \_ k_3$    | $k_2$           |
|                     | $k_1 \_ k_3$    | $k_3$           |

Teoreticky vzato by šlo u v první iteraci interpretovat všechny klíče jako pravdivé, ale prakticky algoritmy pro řešení SATu (např. miniSAT) se snaží minimalizovat počet pravdivých literálů, tedy v první iteraci považuje za nutnou podmínku učinit pravdivým pouze jeden literál.

## Kapitola 3

### Algoritmy

V této kapitole si ukážeme algoritmy implementované k hledání maximálního vlastního lock-chartu ze zadání. Zavedíme několik druhů algoritmů, které se navzájem různě doplňují, volají a snaží se zlepšit řešení výsledku algoritmů jiných.

#### 3.1 Kritérium

Než se vrhneme na konkrétní implementace algoritmů zavedeme si kritérium jako komparátor mezi dvěma uzly grafu. Tedy pro naše konkrétní zadání lock-chartu maticí máme, že každý klíč  $k \in K$  a každý zámek  $l \in L$  je zároveň uzlem, případně spojen hranou pokud existuje pátá  $E$  takové, že platí  $(e) = (k; l)$ . Říkáme tedy, že máme množinu uzlů  $U = K \cup L$ . Pokud hovoříme o nějakém libovolném kritériu znamená to, že jsme schopni booleovskými (ohodnocením true nebo false) vyhodnotit následující nerovnost  $c : u \leq v$  pro každé dva uzly  $u, v \in U$ .

Definice 3.1. (Jednoznačně dané vyhodnocení kritéria). Řekneme, že vyhodnocení je jednoznačně dané pokud je deterministické.

#### 3.2 Ověření správnosti řešení

To, že je problém ve třídě NP znamená, že jsme schopni v polynomiálním čase ověřit správnost řešení. Jelikož budeme potřebovat proceduru ke kontrole, zda u daný iterativní či rekurzivní algoritmus máme ukončit a vrátit nalezené řešení, zavedeme polynomiální kontrolu výsledku.

Algoritmus 3.1. (Ověření řešení).

```

vstup : lock-chart T = ( K; L; E )
výstup: true pokud je řešení diagonálou, false pokud není
1 Funkce Ověření( T )
2   for k : K do
3     col = true;
4     for l : L do
5       if (k, l) ∈ E then
6         if col then
7           col = false;
8         else
9           return false
10        end
11      end
12    end
13    if col then
14      return false
15    end
16  end
17  for l : L do
18    row = true;
19    for k : K do
20      if (k, l) ∈ E then
21        if row then
22          row = false;
23        else
24          return false
25        end
26      end
27    end
28    if row then
29      return false
30    end
31  end
32  return true;

```

### 3.3 Algoritmy s optimálním řešením

Jak jsme si již výše uvedli, algoritmus který vrací optimální řešení je nejlepší možný, co se týče správnosti a velikosti řešení. Problém je v tom, že jsme schopni převést hledání maximálního vlastního částelného lock-chartu

Obrázek 3.1: Odhad počtu klauzulí v závislosti na velikosti řešení

nejlépe na úlohu SAT, která leží ve třídě složitosti NP-úplných úloh.

### 3.3.1 SAT

Zásadní nevýhodou jakékoliv implementace SATu je exponenciální nárůst klauzulí podle velikosti zkoumaného lock-chartu. Na Obrázku 3.1 na straně 21 vidíme odhad očekávaného nárůstu počtu klauzulí. Tento odhad jsme vyčíslili z odhadovaného počtu klauzulí pro  $i$ -tou iteraci přidávaných podmínek pro vynucení velikosti právě rovně  $i + 1$  (jestliže iterujeme od 0). Odhad se tak vlastně rovná binomické větě. Modrou barvou máme vyznačený odhad oblastí, na které bychom mohli výpočetně dosáhnout s dobrou výpočetní technikou, u červené sekce nemáme zaručeno, že algoritmus vůbec nesele na nedostatek paměti. Mějme na paměti, že toto uvažujeme pro obecné zadání, jestliže budeme řešit malé lock-charty (řádově do desítek klíčů a zámků), SAT se pravděpodobně doplní. Jestliže budeme řešit lock-chart větších velikostí (řádově stovky a více klíčů a zámků), modrá oblast na ukázané křivce se pravděpodobně zmenší směrem od středu.

Všimáme si jiného trendu. Jelikož odhad se blíží podobě binomickému rozdělení, "lehčí" sekce se nachází na obou koncích křivky. Z toho nám plynou dva případy:

1. řešení leží na krajích rozdělení.

To znamená, že pokud začneme iterovat algoritmus SATu od 1 výše, můžeme dosáhnout alespoň na řádkové řešení, které využijeme v dalších

algoritmech. Pokud bychom iterovali od shora dol., tedy od  $\min(|K|, |L|)$  níže, m.ºeme dosáhnout na optimální ºezení jako na první nalezené ºezení.

## 2. ºezení leºí ve stºedu rozdílení.

V takovém pºípad platí, ºe m.ºeme dostat ástené ºezení pºi hledání smrem nahoru od 1 jako v minulém pºípad, avak hledání odshora dol. nám nic nepºinese. K optimálnímu ºezení se pravdpodobn pomocí SATu nedostaneme.

Mjme instanci SATu a pºedpokládejme, ºe se s ní díky potu vygenerovaných klauzulí nedopoítáme optimálního ºezení. Má smysl o takovém algoritmu vbec uvaºovat? eknme, ºe existuje njaký jiný rychlý algoritmus, který nám pom.ºe "pºehoupnout" se pºes velikosti ºezení, kde je poet klauzulí kriticky velký. To udlá tak, ºe nám poskytne njaké ástené ºezení. Od tohoto výsledku se m.ºeme dále pokouet ºezení vylepit. Pak má smysl v kombinaci s takovým rychlým (nap. heuristickým) algoritmem SAT kombinovat. Proto následující algoritmy budou slouit spºe k optimalizaci a vylepení ºezení, které najdeme jinými rychlejšími metodami.

## ■ SAT bez modifikací

Algoritmus 3.2. (SAT bez modifikací).

```

vstup : lock-chart T = ( K; L; E )
výstup: Pºíºazení vlastních klí. a zámek.
1 Funkce SAT(T)
2   for i = 1 : min(|K|, |L|) do
3     Lit = K [ L [ E;
4     Cls = SAT-Klauzule (Lit );
5     ºezení = 0;
6     Cls [ Omezující-Klauzule (Lit; i );
7     s = Vyºe-SAT(Cls);
8     if s je modelem Clsthen
9       Ovºezení (s);
10      ºezení = Extrahujºezení (s);
11     else
12       return ºezení
13     end
14   end

```

Jedná se o istou implementaci pºevodu z úlohy hledání maximální ásteného vlastního lock-chartu na SAT úlohu z minulé kapitoly. Funkce SAT-Klauzule pºevádí jednotlivé klíe a zámky do vztah., které podmiují vlastní

ĚásteĚný lock-chart. Funkce Omezující-Klauzule přidává podmínku pro pravdivost právě i klíče a tedy i řešení o velikosti maximálně i. Ověření je procedura pro korektnost výsledku. Funkce Vyřeš-SAT vyhodnotí, které literály dostanou booleovskou proměnnou true a které false, tak aby všechny klauzule byly splněné (pokud to jde). Jestliže jde najít takové ohodnocení pro všechny klauzule říkáme, že je modelem množiny klauzulí CIs. Funkce Extrahuj-řešení slouží k převodu z výsledku SATu na naši dvojici klíče a zámek o.

### ■ SAT s redukcí

Nechť existuje lock-chart  $T = (K; L; E)$  s optimálním řešením  $o_{\max} > n$ . Mějme instanci SATu, jenž našla nějaké ĚásteĚné řešení velikosti v iteraci i, kde  $i < \min(|K|, |L|)$ . Pak staří pro iteraci  $i + 1$  v SATu uvažovat řešení pouze ĚásteĚný lock-chart<sup>0</sup>, který neobsahuje klíček  $2 \leq k \leq K$ , jenž nesplňují nerovnost  $|L| \geq |E(k)| + 1 - n$  a zámky  $1 \leq l \leq L$ , které nesplňují nerovnost  $|K| \geq |E(l)| + 1 - n$ .

Algoritmus 3.3. (SAT s redukcí).

```

vstup : lock-chart T = ( K; L; E )
výstup: Přiznání vlastních klíče a zámek.
1 Funkce SAT(T)
2   for i = 1 : min(|K|, |L|) do
3     Lit = K [ L [ E;
4     CIs = SAT-Klauzule (Lit);
5     řešení = 0;
6     CIs [ Omezující-Klauzule (Lit; i);
7     s = Vyřeš-SAT(CIs);
8     if s je modelem CIs then
9       Ověření (s);
10      řešení = Extrahuj-řešení (s);
11      (K; L; E) = Redukce-Lock-Chartu (K; L; E);
12    else
13      return řešení
14    end
15  end

```

D-kaz. Jestliže jsme našli ĚásteĚné řešení o velikosti, znamená to, že se jedná buď o optimální řešení  $o_{\max} = n$  anebo o řešení menší než optimální  $o_{\max} > o_n$ . To znamená, že řešení může obsahovat jedině zámky a klíče, které neotvírají a nejsou otevírány alespoň 1 příslušnými klíči a zámky. Mějme klíček  $k \in K$ , nerovnost  $|L| \geq |E(k)| + 1 - n$  nám říká, že klíček k neotvírá alespoň  $|L| \geq |E(k)|$  zámek, počet neotvíraných zámek v řešení je  $n - 1$ .

Kdyby existoval klíč  $k \in K$ , který neotvírá  $\{L_j \mid j \in E(k)\}$  zámek, u  $n$  z nerovnice vidíme, že je to méně, než je potřeba k řešení, kde je neotvíráno  $n - 1$ . Pro zámky platí obdobné tvrzení.  $\square$

Mějme stejný algoritmus pro SAT jako v algoritmu 3.2 na straně 22. Přidejme navíc funkci Redukce-Lock-Chartu, která na konci každé iterace, je-li nalezeno řešení, odebere příslušný počet řádků a sloupců lock-chartu, které nesplňují nerovnost danou předpisem redukce.

### SAT s počátečním řádkovým řešením

Mějme řádkové řešení o velikosti  $n$ , tedy i dané přirozené číslo klíčů a zámků, kde  $j \in E(k) \Rightarrow j \leq n$ . Potom můžeme SAT využít jako vylepšovací metodu od dané velikosti řádkového řešení. Tedy začneme iteraci algoritmu od daného. Níže uvedený pseudokód obsahuje redukci prostoru při nalezení řádkového řešení stejně jako v algoritmu 3.3 na straně 23.

Algoritmus 3.4. (SAT s počátečním řádkovým řešením).

```

vstup : lock-chart  $T = (K; L; E)$ , velikost řádkového řešení  $n$ 
výstup: Přirozené číslo klíčů a zámků
1 Funkce SAT( $T, n$ )
2   for  $i = n : \min(|K|, |L|)$  do
3     Lit =  $K \cup L \cup E$ ;
4     Cls = SAT-Klauzule (Lit);
5     řešení = 0;
6     Cls [ Omezující-Klauzule (Lit; i);
7     s =  $\forall$  řešení-SAT(Cls);
8     if s je modelem Cls then
9       Ověř řešení (s);
10      řešení = Extrahuj řešení (s);
11       $(K; L; E) =$  Redukce-Lock-Chartu ( $K; L; E$ );
12    else
13      return řešení
14    end
15  end

```

Poznámka: Všimněme si, že nevyužíváme konkrétní přirozené číslo daného řešení vstupem  $n$ , ale pouze jeho velikost. Kdybychom SAT modifikovali tak, aby začínal na konkrétním řádkovém řešení, tedy i jeho přirozené číslo klíčů a zámků, mohli bychom dojít k horšímu (menšímu) řešení.



### 3.3.2 Strom podle kritéria

Mějme množinu seřazených uzlů, které říkáme sekvence. Každý uzel je buďto klíčem nebo zámkem. Předpokládejme, že pokud budeme procházet sekvenci v pořadí od prvního prvku do posledního a v každém kroku daný uzel odstraníme z grafu, zbude nám pouze diagonála. Tedy máme  $\sigma = (u_1; u_2; \dots; u_n)$ , kde  $n = |K| + |L|$  a  $u_i \in U$  pro  $i \in \{1, 2, \dots, n\}$ , která značí pořadí klíče a zámku, které musíme z lock-chartu odstranit, aby nám zůstaly pouze klíče, zámky a hrany, představující vlastní klíče a zámky.

Jestliže máme sekvenci a po odstranění všech uzlů z ní zůstane nějaké řešení, pak tento vztah zapisujeme jako  $\sigma \in \mathcal{R}$ .

Nechť máme sekvenci  $\sigma_{\text{subopt}}$ , která nám říká, jak dostat řešení  $\sigma_{\text{subopt}}$ , kde  $|\sigma_{\text{subopt}}| \leq |\sigma_{\text{max}}|$ , pro  $\sigma_{\text{max}}$ , které je optimálním řešením úlohy.

Definice 3.2. (Blízká řešení). Nechť máme dvě různá řešení úlohy  $\sigma_1, \sigma_2$ , kde  $|\sigma_1| < |\sigma_2|$  a dvě sekvence  $s_1, s_2$ , kde  $s_1 \in \sigma_1$  a  $s_2 \in \sigma_2$ . Říkáme, že míra blízkosti je počet úprav v sekvenci, přidání nového prvku (uzlu ze sekvence) do množiny řešení nebo smazání posledního prvku (uzlu ze sekvence) z množiny řešení.

Předpokládejme, že řešení  $\sigma_{\text{subopt}}$  leží blízko  $\sigma_{\text{max}}$ , tedy jestliže si znázorníme sekvenci do podoby stromu, kde nejlevnější větve symbolizují sekvenci částelného řešení  $\sigma_{\text{subopt}}$ , pak tento předpoklad říká, že by se v takovém stromě mohlo nacházet lepší řešení vzdálené mírou blízkosti, tedy daný počtem backtracků (odebrání uzlu) a dalšího prohledávání (přidání uzlu) ve stromě.

Definice 3.3. (Úplná sekvence). Nechť máme lock-chart  $T = (K; L; E)$  a kritérium, které generuje sekvenci. Pokud sloužíme sekvenci a uzly (klíči, zámky), které sekvence neobsahuje, vytvoříme tak novou sekvenci  $\sigma^0$ , jenž nazýváme úplnou pokud prvních  $|s|$  prvků v  $\sigma^0$  je totožných s prvním prvkem  $s$ . Jinými slovy, jedná se o řadu obsahující všechny uzly z lock-chartu  $T$ , seřazené podle daného kritéria.

Poznamenejme si, že úplná sekvence má účel především účel takový, abychom mohli v algoritmu uvažovat všechny uzly. Jestliže v dané blízké řešení mluvíme o odstranění posledního prvku, myslíme tím prvek jenž nemusí být nutně posledním uzlem úplné sekvence.

Postup k hledání optimálního řešení je následující: Procházíme strom do hloubky dokud nenajdeme první řešení (nemusí být nutně optimální, jelikož to závisí na volbě kritéria). Jakmile na něj narazíme, nastavíme si spodní hranici hloubky pod kterou už nemusíme hledat. Backtrackujeme o dva uzly výše a zkontrolujeme v sekvenci změnit daný uzel za klíč nebo zámek, který v sekvenci po něm následuje. Jestliže nacházíme lepší nebo stávající nejlepší nalezené řešení, popisujeme spodní hranici a hodnotu nejlepšího řešení, pokud se tak nestalo zkontrolujeme další uzel v pořadí až do konce dané sekvence. Jestliže jsme vyčerпали všechny uzly, backtrackujeme o uzel výše, jestliže jsme v konci

Obrázek 3.2: Lock-chart a jeden z jeho možných stromů podle kritéria

stromu vrátíme poslední nalezené řešení jako optimální.

V postupu jsme uvedli několik tvrzení, které dokážeme. Jako první to bude korektnost ořezávání stromu. Tedy fakt, že pepsáním spodní hranice prohledávání nikdy neztratíme optimální řešení.

Důkaz. Mějme libovolné řešení  $\sigma_{\text{subopt}}$  o velikosti  $|\sigma_{\text{subopt}}| = n$ , sekvenci přes níž jsme se k tomuto řešení dostali, tedy  $s_{\text{subopt}}$  o velikosti  $|s_{\text{subopt}}| = m$ . To znamená, že jsme odstranili z lock-chartu  $m$  klíčů a zámků (uzlů) a máme libovolný čtvercový lock-chart o velikosti  $|K^0_j| = |L^0_j|$ , kde  $|K^0_j| = |L^0_j| = |K_j| + |L_j| - |s_{\text{subopt}}|$  pro  $K^0 = K$  a  $L^0 = L$ . Pokud bychom odstranili lichý počet uzlů, lock-chart který nám zbyde už nebude čtvercový a při odstranění sudého počtu uzlů bude libovolný lock-chart menší než  $p$ -vodní, tedy nemůžeme dalším prohledáváním do hloubky najít lepší řešení.  $\square$

Dále ještě uvádíme tvrzení, že při nalezení nějakého řešení provádíme dva kroky backtracku místo tradičního jednoho. Stejně jako v minulém případě, ani u toto drobného ořezání prostoru nám nehrozí ztráta optimálního řešení.

D-kaz. Mějme řástečné řešení  $O_{\text{subopt}}$  o velikosti  $j_{O_{\text{subopt}}}$  a sekvenci  $p$  o velikosti  $n$ . Jestliže jsme se k tomuto řešení dostali, tedy  $(O_{\text{subopt}}, p)$  o velikosti  $j_{O_{\text{subopt}}}$  a sekvenci  $p$  o velikosti  $n$ . Jestliže jsme se k řástečnému řešení dostali pomocí odstranění uzlů, žádná jiná kombinace odstraněných uzlů nemůže zvětšit výsledný prořezaný řástečný vlastní lock-chart. Nejlepší řešení jsme schopni najít pomocí sekvence o velikosti  $m^2$  (pokud takové existuje).  $\square$

Algoritmus 3.5. (Strom podle kritéria).

```

vstup : lock-chart  $T = (K; L; E)$ , sekvence
výstup: Velikost vlastního lock-chartu

1 Funkce CT(T, s)
2    $v_{\max} = 0$ ;
3   used = fg;
4   CT-Rekurze(T, s, 0, |s|);
5   return  $v_{\max}$ ;

6 Funkce CT-Rekurze(T, s, h, lb)
7   if  $lb == h$  then
8     return false;
9   end
10  for node : s do
11    if node ∈ used then
12      continue;
13    end
14     $T^0 = T \setminus \text{node}$ ;
15    used = used \ node;
16    if  $O_{\text{subopt}} \in O_{\text{subopt}}(T^0)$  then
17      if  $v_{\max} < \text{Velikost}(O_{\text{subopt}}(T^0))$  then
18         $v_{\max} = \text{Velikost}(O_{\text{subopt}}(T^0))$ ;
19         $lb = h$ ;
20        used = used \ node;
21        return true;
22      end
23    else
24      solFound = CT-Rekurze( $T^0$ , s, h+1, lb);
25      if solFound then
26        used = used \ node;
27        return false;
28      end
29    end
30  end

```

Příklad 3.4. Mějme zadaný lock-chart z Obrázku 3.2 na straně 26 a kritérium, které nám vygenerovalo úplnou sekvenci  $s = (k_1; l_4; l_2; k_2; k_3; k_5; l_3)$ . Inicializujme si dolní hranici na  $lb = |s|$  a hodnotu nejlepšího nalezeného

°e²ení na  $v_{\max} = 0$ .

P°i prohledání nejlevšj²ího podstromu nará°íme v hloubce  $h = 4$  na první °e²ení o velikosti  $j_{\text{subopt}} = 2$ . M°eme tedy spodní hranici  $lb$  nastavit na  $lb = 4$  a hodnotu nejlepšího nalezeného °e²ení na  $v_{\max} = 2$ . Nalezené °e²ení je  $o_{\text{subopt}} = f(k_3; l_1; (k_5; l_3))g$ .

Backtrackujeme o dvš patra stromu na uzlu  $l_4$  a zkoušíme odstranit uzel v sekvenci  $pol_2$ , šim° se stává uzel  $k_2$ . Po jeho odstranšní nalšzáme °e²ení o velikosti 3, tedy pro  $lb = 3$ ,  $v_{\max} = 3$  a  $o_{\text{subopt}} = f(k_3; l_1; (k_4; l_2); (k_5; l_3))g$ .

Backtrackujeme o dvš patra stromu výše, ale po vyzkoušení všech ostatních prv° sekvence, namšsto uzlu  $l_4$  a posléze po dalším backtracku uzlu  $k_1$ , nenalšzáme lepší °e²ení.  $o_{\max} = f(k_3; l_1; (k_4; l_2); (k_5; l_3))g$  a velikost optimálního °e²ení je  $j_{\text{opt}} = 3$ ;

### 3.3.3 Binární strom podle kritéria

Algoritmus 3.6. (Binární strom podle kritéria).

```

vstup : lock-chart  $T = (K; L; E)$ , sekvence
výstup: Velikost vlastního lock-chartu

1 Funkce BCT( $T, s$ )
2    $v_{\max} = 0$ ;
3   BCTRecursion( $T, s.\text{první}(), 0, |s|$ );
4   return  $v_{\max}$ ;

5 Funkce BCTRecursion( $T, s, h, lb$ )
6   if  $lb == h$  then
7     return ;
8   end
9   if Ovš°-°e²ení( $T$ ) then
10    if Velikost-°e²ení( $T$ )  $v_{\max}$  then
11       $v_{\max} = \text{Velikost-°e²ení}(\mathbf{T})$ ;
12       $lb = h$ ;
13    end
14  end
15  node = s;
16   $T^0 = T \setminus \text{node}$ ;
17  BCTRecursion( $T^0, s.\text{next}(), h+1, lb$ );
18  BCTRecursion( $T, \text{node}.\text{dalš}(), h, lb$ );

```

U algoritmu stromu podle kritéria leší jeden optimalizašní problém. M°eme toti° projít stejnou cestou ve stromu dvakrát, co° vede ke zbytešnému výpošetnímu výkonu navíc. Stejnou cestou rozumíme dvš sekvence  $s_1$  a  $s_2$ ,

Obrázek 3.3: Obecně zapsaný binární strom pro  $t$ i uzly

kde  $s_1 = s_2$ , ale pořadí prvků v sekvencích se liší. Abychom se backtrackování stejnou cestou v jiném pořadí ve stromu vyhnuli, sestavíme strom binárně, tedy v každém uzlu stromu se bude rozhodovat, zda daný uzel patří do sekvence vygenerované kritériem nebo ne.

Na Obrázku 3.3 na straně 29 máme strom pro sekvenci řítající  $t$ i prvky, tedy pro lock-chart o velikosti  $2 \times 1$  (nebo po transpozici matice  $1 \times 2$ ). V každém uzlu se algoritmus rozhoduje, zda daný uzel bude ležet v sekvenci, jenž vede k optimálnímu řešení. Tedy TRUE včetně nám říká, že daný uzel nebude v optimální řešení  $\alpha_{\max}$  (TRUE jelikož tvrzení, že bude z p-vodního lock-chartu smazán je pravda), včetně FALSE pak říká opak, tedy uzel leží v  $\alpha_{\max}$ . Nejlevější cesta stromem k listu končí odstraněním všech uzlů grafu, nejpravější cesta naopak všechny uzly zanechává.

Jestliže pořadí uzlů pro jednotlivé hloubky generuje nějaké kritérium, říkáme, že se jedná o binární strom podle kritéria. Předpokládáme, že optimální řešení bude ležet buďto v nejlevější cestě anebo v cestě jí blízké.

Postup k hledání optimálního řešení je následující: Procházíme strom do

Obrázek 3.4: První dvě nalezená řešení v binárním stromu

hloubky dokud nenajdeme první řešení (nemusí být nutně optimální, to závisí na volbě kritéria). Jakmile na něj narazíme, nastavíme si spodní hranici hloubky pod kterou už nemusíme hledat a backtrackujeme o dva uzly výše. Až do tohoto momentu byl postup stejný jako u obecného stromu. Rozdíl je v tom, že jakmile vkročíme do uzlu přes hranu FALSE neupravujeme našemu rekurzivnímu volání hloubku o jedničku jako u větvi TRUE. Větve FALSE totiž nezvětšuje délku sekvence, tedy z hlediska velikosti řešení jej nemusí měnit. Jestliže nacházíme lepší řešení upravíme parametry hranic a nejlepšího dosaženého řešení, pokud nalezneme lepší stejné řešení po menším počtu TRUE hran, pak parametry také přeepisujeme. Končíme ve chvíli, kdy jsme celý strom prošli.

Příklad 3.5. Máme zadáný lock-chart z Obrázku 3.2 na straně 26 a kritérium, které nám vygenerovalo úplnou sekvenci  $s = f(k_1; l_4; l_2; k_2; k_3; k_5; l_3)g$ . Inicializujeme si dolní hranici na  $lb = jsj$  a hodnotu nejlepšího nalezeného řešení na  $v_{max} = 0$ .

Při prohledání nejlevějšího podstromu narazíme v hloubce  $h = 4$  při rozhodování u uzlu  $u = k_3$  na první řešení o velikosti  $|s_{subopt}| = 2$ . Můžeme tedy spodní hranici  $lb$  nastavit na  $lb = 4$  a hodnotu nejlepšího nalezeného řešení na  $v_{max} = 2$ . Nalezené řešení je  $s_{subopt} = f(k_3; l_1); (k_5; l_3)g$ .

Backtrackujeme o dvě patra stromu na uzlu  $l_2$ . Volíme, že  $l_2$  nebude v sekvenci hranou FALSE, poté jdeme levou hranou a odebíráme z potenciálního řešení uzel  $k_2$ . Voláme rekurzivně samo sebe a při rozhodování na uzlu  $u = k_3$  nalézáme řešení o velikosti 3, tedy v  $lb = 3$ ,  $v_{max} = 3$  a  $s_{subopt} = f(k_3; l_1); (k_4; l_2); (k_5; l_3)g$ .

Backtrackujeme o dvě patra stromu na uzlu  $l_2$ . Jelikož jsme vyzkoušeli obě větve, tak backtrackujeme na uzlu  $l_4$ . Pro zjednodušení obrázku nerozvíjíme dále strom do hloubky, protože víme, že již nalezené řešení je optimální z předchozích řešení tohoto příkladu. Algoritmus by dále procházel všechny

uzly  $a^0$  do hloubky tří větvi TRUE, ale nenašel by ji lepší řešení, tedy  $O_{\max} = f(k_3; l_1); (k_4; l_2); (k_5; l_3)$  a velikost optimálního řešení je  $O_{\max} = 3$ ;

Pro zjednodušení pseudokódu, neuvažujeme optimálnější verzi s dvojitým backtrackem při nalezení řešení, ale pouze s jedním. V kódu také pro pohlednost zapisujeme první prvek sekvence jako `s:první` a následující prvek v sekvenci jako `node:daří`.

## 3.4 S heuristickým řešením

Vidíme, že zásadní problém výše uvedených algoritmů je jejich výpočetní složitost, kde k rostoucímu počtu klíče a zámků roste exponenciálně i složitost problému. Proto navrhujeme heuristické procedury na ořezání prostoru grafu, tedy nalezení nějakého účelného lock-chartu pro následné dopožití algoritmů optimálními.

### 3.4.1 Biklastrování

Metoda biklastrování (v textu budeme náležitě uvedený a modifikovaný algoritmus dále už jen jako BC od anglického slova biclustering poprvé použité Borisem Mirkinem [3]), jenž bývá dnes nejčastěji používaná k vyřezávání dat v oblasti bioinformatiky [4] se ukázal jako perspektivní přístup k analýze rozsáhlých datových souborů. Myšlenka biklastrování stojí na rozeznávání podobnosti a jejich následné označení ve vstupních datech.

V praxi můžeme pak pomocí metody biklastrování identifikovat třeba skupiny genů, které jsou koherentní v rámci skupin podmínek, tyto skupiny se nazývají biclustery. Geny patřící ke stejnému biclusteru by měly mít blízké biologické funkce [5], [6].

Můžeme ale na paměti, že v obecné formě je biklastrování dokázáno jako NP-těžký problém, lze jej totiž převést na problém hledání maximálního párování přes hrany v úplném bipartitním grafu [7].

Myšlenka biklastrování se tedy nezdá být až tak špatná. Hledáme totiž v ideálním případě takový účelný lock-chart, který má podobu lock-chartu vlastního. Pokusíme se tedy vytvořit takový klastr, jenž v binární matici rozezná diagonálu.

Definice 3.6. (Metoda biklastrování). Nechť  $T = f(K; L; E)$  je lock-chart, pak  $BC(K; L; E) = f(K; L; E)$  je funkce, kde  $K \subseteq K, L \subseteq L, E \subseteq E$ . Nebo-li jedná se o proceduru, jenž ořeže zadaný lock-chart do podoby menšího účelného lock-chartu.

Postup k hledání heuristického řešení je následující: Zvolíme parametr  $r$ , jenž nám říká na kolik klastrů chceme lock-chart rozdělit. Parametr  $r$  odpovídá  $2^r$  klastřům. Můžeme zvolené kritérium  $c$ , které najde vhodné místo k řešení

lock-chartu, tedy místo kde lock-chart rozdělíme na dva částné lock-charty sobě disjunktní. Poté ověříme, zda dané částné lock-charty nejsou řešením. Pokud ano, vrátíme velikost řešení. Pokud ne, pokračujeme v dalším kroku. Ve chvíli kdy vyčerpáme nastavené množství kroků, vrátíme největší nalezené řešení z množiny vytvořených částných lock-chartů  $T_i$ , kde  $i \in \{1, \dots, 2^k\}$ .

Ani zde však není nic bez problému. Úspěch algoritmu závisí na dvou parametrech ( $r; c$ ) a nic nám nezaručuje, že nalezneme vůbec nějaké řešení.

(Volba kritéria kroku  $c$ .) V první řadě si uveďme, co od takového kritéria očekáváme. Cílem je najít takové místo, tedy takovou dvojici klíčů z  $K$  anebo dvojici zámků z  $L$  mezi kterými provedeme krok. Jelikož se snažíme o co nejlepší časovou výpočetní složitost, uvažujme, že se jedná o polynomiální proceduru, tedy chceme aby každému klíči a zámku v lock-chartu přiřadil reálné číslo. Jedná se tedy o funkci:  $K \times L \rightarrow \mathbb{R}$ . Jako vhodné místo k provedení kroku se zdá být taková dvojice klíčů (zámků), kde je rozdíl jejich kritérií maximální nebo naopak minimální.

Toto vede k následující úvaze: Pokud si seřadíme řádky a sloupce matice (lock-chartu), pomůže to nějak řešení biklastrování? Případně úvahu druhým směrem. Existuje nějaké seřazení, které nám usnadní řešení lock-chartu?

Dejme 3.7. (Úplné seřazení). Nechť máme lock-chart  $T = (K; L; E)$  a kritérium kroku  $c$ , které každému klíči a zámku přiřadí reálné číslo, tedy  $c: K \times L \rightarrow \mathbb{R}$ . Jestliže dokážeme jednoznačně seřadit lock-chart, pak říkáme, že se jedná o úplné seřazení.

Mějme diagonální lock-chart z Obrázku 2.2 na straně 10. Pokud bychom jako funkci  $c$  vzali počet otvíraných zámků, případně pro řádky matice počet klíčů otvírajících příslušný zámek a předem dané indexování klíčů, pak dostáváme úplné seřazení. Problém je ale takový, že obecně nejsme úplné seřazení schopni nalézt, tedy obecně platí, že neexistuje jednoznačné kritérium k seřazení. Stačí si vyzkoušet třeba příklad z Obrázku 2.3 na straně 12.

Reálně se proto nakonec osvědčila nejlépe jedna z nejjednodušších funkcí kroku a to počet hran, které z vedou z klíče do daného zámku. Tedy pro každé políčko matice  $M$  (o rozměrech  $|K| \times |L|$ ) vypočteme hodnotu následující

$$\text{formulí } m_{ij} = \sum_{k \in K} E(k, j) \text{ a hodnotu sloupce pro klíč } k_i \text{ jako } c: k_i \rightarrow \sum_{j \in L} m_{ij}.$$

Všimáme si, že hodnota není závislá na volbě zámku.

V algoritmu můžeme provést ještě jednu optimalizaci k urychlení výpočtu. Ohodnocení řádků a sloupců lock-chartu provádíme pouze jednou při spuštění, tedy neuvažujeme možnost, že bychom mohli změnit ohodnocení klíčů a zámků při rozdělení na částné lock-charty. Toto zjednodušení by mohlo být implementováno k urychlení výpočtu.

(Volba parametru  $r$ .) Vidíme, že má smysl uvažovat parametr  $r$  z intervalu  $1 \leq r \leq \log_2(|K| \times |L|)$ , kde  $K$  je množina klíčů a  $L$  je množina zámků. Přiřazením



Obrázek 3.5: Časy při perfektní volbě parametru  $r$ 

dolní limit nám říká, že provedeme pouze jeden řez a horní limit naopak, že každé políčko matice, tedy každý prvek lock-chartu bude v jednom klastru.

Předpokládejme, že máme perfektní kritérium řezu, tedy takové, že dokážeme oddělit generální a hlavní klíče od klíčů vlastních pomocí jednoho řezu a stejnou proceduru i pro centrální vložku a vložky hlavní. Pak by optimální volba parametru  $r$  byla  $r = 2$ .

Příklad 3.8. Na Obrázku 3.5 na straně 33 máme znázorněný příklad perfektního kritéria řezu. Pomocí dvou řezů mezi klíči  $k_4, k_5$  a zámky  $l_6, l_7$  jsme schopni oddělit maximální vlastní lock-chart.

Příklad 3.9. Mějme zadaný lock-chart z Obrázku 3.2 na straně 26 a volbu parametru  $r = 3$ . Kriteriační funkci řezu zvolme následovně:  $m_{i,j} = j \in (k_i) \setminus j \in (l_j)$  a řez budeme provádět na místě, kde je rozdíl průměrného kritéria pro nějaké dva řádky nebo sloupce matice největší. Nejprve spočteme hodnoty matic  $m_{i,j}$  podle vzorce  $\text{prom}_{i,j}$ . Dostáváme pravou matici ohodnocení z Obrázku 3.6 na straně 35. Následně vypočteme průměrnou hodnotu řádku a sloupce a podle nich seřadíme řádky a sloupce matice.

(Iterace 1). Mějme ohodnocenou matici z Iterace 1 na Obrázku 3.10 na straně 36. Jelikož největší rozdíl  $d = 13 - 7:8 = 4:2$  je mezi řádky jedna a dva provedeme zde řez. Dále by algoritmus procházel paralelně obě dvě podmatice, my však budeme sledovat pouze tu dolní, jelikož vidíme, že horní

oddělená podmatice není nijak zajímavá k prozkoumání (může obsahovat vlastní řástečný lock-chart o velikosti maximálně 1).

(Iterace 2). •ez provádíme mezi prvním a druhým sloupcem díky rozdílu  $d = 8 - 5:3 = 2:7$ . Vímáme si, že jsme hodnoty jednotlivých polí matice a tedy i pr-měry příslušných řádků a sloupců porovnávali oproti iteraci 1. Pokračujeme to další iterace pouze s pravou částí matice, i když algoritmus zpracovává podmatice obě.

(Iterace 3). Provedeme •ez mezi prvním a druhým sloupcem pro rozdíl  $d = 3:3 - 1:6 = 1:7$  Při dalším zavolání rekurze bychom pouze ověřili, že pravá podmatice je vlastním řástečným lock-chartem.

Z rekurzí se nám vrátí několik •ezení, jelikož jsme zvolili parametr  $r = 3$ , máme tedy  $2^3 = 8$  potenciačních •ezení. Množina velikostí všech •ezení by vypadala následovně  $V = \{1, 3\}$ , přičemž •ezení o velikosti jedna se několikrát opakovalo.

Výsledkem metody je tedy  $o = f(k_3; l_1); (k_4; l_2); (k_5; l_3)g$ , shodou náhod i hledané optimální •ezení  $o_{max}$ .

Algoritmus 3.7. (Metoda biklastrování).

```

vstup : lock-chart T, parametr k
výstup : Velikost vlastního lock-chartu

1 Funkce BC(T, k)
2   if k ≤ 0 then
3     Rozděl T podle výsledku •ez( T ) na T1 a T2;
4     a = BC(T1, k-1);
5     b = BC(T2, k-1);
6     return max(a, b);
7   else
8     if Ověř•ezení( T ) then
9       if Velikost•ezení( T ) > vmax then
10        | return Velikost•ezení( T );
11        | end
12        else
13        | return 0;
14        | end
15    end

```

Poznámky k příkladu: Je zcela jasné, že v tomto příkladě nemá smysl dále •ezat "plné" řádky a sloupce, jen nám zbudou po rozdělení. Jen bychom plynuli výpočetními prostředky. Ilustrované tři iterace jsou tedy i zázorným optimalizované verze BC.

Vímáme si však, že volba kritéria započínala, že jsme vlastně osekávali

Obrázek 3.6: Matice ohodnocení  $M$  před seřazením.

o jeden klíč nebo zámek při každé iteraci. Obecně to nemusí pro dané kritérium platit, ale zkusme vytvořit takové pravidlo, které postupně otevírá prostor po jednom klíči nebo zámku.

### 3.4.2 Hladové kritérium

Mějme zadaný lock-chart ve formě bipartitního grafu. V minulé kapitole jsme si ukázali, že metodou maximálního párování nelze dosáhnout požadovaného výsledku. Je tedy potřeba nějakým způsobem omezit graf tak, aby řešení bylo jednoznačné a korektní.

Definice 3.10. (Jednoznačné řešení). Mějme bipartitní graf  $G = (V; E)$ , množinu klíčů  $K$  a množinu zámků  $L$ , kde platí  $|K| = |L| = |E|$  a zároveň  $d(l) = d(k)$  pro všechna  $k \in K$  a  $l \in L$ . Takovou množinu  $E$  nazveme perfektním párováním, jestliže pro všechny  $k$  platí, že  $|E(k)| = 1$  a pro všechna  $l$  platí  $|E(l)| = 1$ . Pak o této množině hovoříme jako o jednoznačném řešení.

Na Obrázku 3.11 na straně 37 máme příklad takového jednoznačného řešení. Vidíme, že se shoduje s vlastním lock-chartem.

Hledáme tedy nějaké pravidlo (kritérium) podle kterého bychom z původního zadaného lock-chartu dostali řádný vlastní lock-chart a jeho příslušný graf v podobě perfektního párování. Je potřeba nějakým způsobem z množiny uzlů  $U = \{k_1, k_2, \dots, k_n; l_1, l_2, \dots, l_n\}$  mazat prvky, tak abychom po konečném počtu kroků dostali řešení (ne nutně optimální). Jedná se tedy v podstatě o speciální případ metody biklastrování, kdy provádíme řez tak, aby výsledkem byl sloupec (řádek) a zbytek matice.

Definice 3.11. (Hladový výběr). Mějme množinu uzlů  $U$ , kde každý uzel  $u \in U$  má ohodnocení  $p(u) \in \mathbb{R}$ . Řekneme, že výběr prvku  $z \in U$  je hladový, jestliže vybíráme prvek  $\max(p(U))$ . [17]

Definujme si funkci  $p$  jako stupeň uzlu. Pravidlo hladového výběru pak vypadá nadřadně. Skutečně se chceme co nejdříve zbavit z lock-chartu převedením generálních a hlavních klíčů (obdobně centrálních a hlavních vložek).



Obrázek 3.10: Tři iterace metody biklastrování

Příklad 3.12. Mějme zadání z Obrázku 2.5 na straně 14. Pak funkce vrací následující vektor pro  $p = (k_1, k_2, k_3, l_1, l_2, l_3, l_4) = (2, 1, 2, 1, 1, 2, 1)$ .

Pokud pomocí tohoto hladového kritéria vygenerujeme sekvenci  $u^0 = (l_3, k_1, k_3, k_2, l_1, l_2, l_4)$ , tak  $u^0$  po jednom kroku, tedy odstranění uzlu  $l_3$  dostáváme optimální řešení  $|j_{\max}| = 3$ . Problém nastává ve chvíli, kdy po přidání uzlu se stejnou kardinalitou prohodíme existuje dalších 96 seřazení, kde se dostaneme pouze k lokálnímu řešení  $|j_{\text{opt}}| = 2$ . Potřebujeme tedy silnější kritérium, než pouhý stupeň uzlu.

(Perspektivní hrana). Mějme prvotní ohodnocení stupněm vrcholu. Chceme dále seřadit ty vrcholy, které se ve svém stupni shodují s alespoň jedním jiným uzlem, tedy vyhodnotit, který z nich je "perspektivnější" do budoucna a který ne. Víme, že jednoznačné řešení má podobu perfektního párování v grafu, což je to takový stav, kdy se stupně rovnají velikosti jedna. Předáváme, tedy navíc pravidlo, které řadí uzly podle toho, zda nějaká hrana z nějaké vedoucí není hranou jedinou uzlu druhého. Formálně každé hraně pro každý uzel přiřadíme celé číslo  $a_i$  :  $u = (a_1, \dots, a_n)$  pro  $u \in U$  a  $a_i \in \mathbb{N}$ ,

Obrázek 3.11: Perfektní párování pro vlastní lock-chart

tedy vektor přirozených čísel každému uzlu, jen odpovídá stupňům uzlu koncovým k příslušným hranám. Jestliže v daném vektoru není obsaženo číslo jedna hovoříme pak o takovém vrcholu jako o perspektivnějším ve smyslu v porovnání se všemi ostatními uzly se stejným stupněm.

Myšlenka je tedy následující: Jestliže z uzlu vede cesta hranou do koncového uzlu pouze se stupněm jedna, pak pokud tento uzel (pořádkovní) odstraníme, tuto cestu z potenciačního řešení navždy ztratíme. Pokoušíme se tedy ji ponechat ve výsledné podmatici co nejdéle.

Algoritmus 3.8. (Hladový bez modifikací).

```

vstup  : lock-chart  $T = (K; L; E)$ 
výstup : Velikost vlastního lock-chartu

1 Funkce Hladový( $T$ )
2   while Maximální-Párování( $T$ )  $\neq$   $|E|$  do
3        $n = \max(T)$ ;
4       if  $1 \neq \text{single}(n)$  then
5            $T = T - n \text{ f } \text{nodeg}$ ;
6       end
7   end
8   return Velikost-řešení( $T$ );

```

Příklad 3.13. Mějme zadání z Obrázku 2.5 na straně 14. Funkce `single` vrací následující vektory pro vrcholy `single : k1 ! f1; 2g`, `single : k2 ! f1g`, `single : l4 ! f2g`.

Pomocí hladového kritéria zvolíme uzel k odstranění z lock-chartu. Hladové kritérium vyhodnotí jako největší uzly  $t^i$  a to  $k_1$ ,  $k_3$  a  $l_3$  s velikostí 2. Aplikujeme na něj předané pravidlo perspektivní hrany, tedy `single : k1 ! f1; 2g`, `single : k3 ! f2; 1g` a `single : l3 ! f2; 2g`. Z příslušných výsledků funkcí vidíme, že jako jasný vítěz a tedy kandidát k odstranění je uzel  $l_3$ , jelikož jako jediný neobsahuje perspektivní hranu. Po odstranění uzlu dostáváme optimální řešení  $|j_{\max}| = 3$ .

Ve výše uvedeném příkladě jsme viděli, že lze touto metodou získat opti-

mální řešení, toto však neplatí obecně. Perspektivní hranou pouze zvýšíme pravděpodobnost nalezení lepšího řešení.

### ■ Hladový s redukcí požátekního prostoru

Mějme následující tvrzení: Jestliže je uzel obsažen v nějakém maximálním párování pak je bude spíše ležet v konečném řešení. Jestliže tedy máme deterministický algoritmus pro hledání maximálního párování - vrátí po opakovaném volání na stejném vstupu jedno a totéž párování, pak pokud toto párování zavoláme na lock-chart  $T = (K; L; E)$  k dalšímu hledání řešení budeme uvažovat pouze ty uzly příslušného bipartitního grafu, které byly obsaženy v daném maximální párování.

Algoritmus 3.9. (Hladový s redukcí požátekního prostoru).

```

vstup : lock-chart  $T = (K; L; E)$ 
výstup : Velikost vlastního lock-chartu
1 Funkce Hladový( $T$ )
2    $T = T \setminus \text{Maximální-Párování}(T)$ ;
3   while  $\text{Maximální-Párování}(T) \neq |E|$  do
4      $n = \max(T)$ ;
5     if  $1 \neq \text{single}(n)$  then
6        $T = T \setminus \text{node}(n)$ ;
7     end
8   end
9   return  $\text{Velikost-Řešení}(T)$ ;

```

### ■ Hladový s odstraněním duplikací

V průběhu iterací hladového algoritmu může nastat situace, kdy vzniknou dva totožné řádky nebo sloupce v matici dané zadaným lock-chartem. Tato modifikace řešení nejen problém vzniklých duplikací, ale i odstranění prázdných uzlů, které vznikají při nutnosti smazání perspektivního uzlu. Obě operace zlepšují velikost řešení za cenu výrazného zpomalení průběhu algoritmu.

Obrázek 3.12: Kritérium podmnožiny s protipříkladem optimálnosti

Algoritmus 3.10. (Hladový s odstraněním duplikací).

```

vstup : lock-chart  $T = (K; L; E)$ 
výstup: Velikost vlastního lock-chartu
1 Funkce Hladový( $T$ )
2    $T = T \setminus \text{Maximální-Párování}(T)$ ;
3   while Maximální-Párování( $T$ )  $\neq \emptyset$  do
4      $n = \max(T)$ ;
5     if  $n \neq \text{single}(n)$  then
6        $T = T \setminus n$ ;
7        $T = \text{Odstraní-Duplikace}(T)$ ;
8     end
9   end
10  return Velikost-řešení( $T$ );

```

Výběr hladového kritéria je daný dosažením poměrně dobrých výsledků proti výpočetnímu výkonu. Obecně lze ale algoritmy využívající nějaké kritérium dosadit kritérium vlastní podle zadaného typu lock-chartu (tyto úvahy dále rozvineme v další kapitole). Pro doplnění ještě uvedeme dva poznatky s experimenty s volbou kritéria:

(Kritérium podmnožiny). Mějme řádek (sloupec) daný je binární vektor, kde 1 symbolizuje, že klíč otevírá (zámek je otvírán) a 0 opak. Mějme, že jsou dva binární vektory,  $a$  a  $b$  v relaci podmnožina jestliže pro  $c = a \wedge b$  platí, že  $\min(c) = 0$ . Tedy neobsahuje žádnou 1, což znamená, že vektor s méně vstupy obsahuje jen tytéž otvírané položky jako vektor s větším počtem otvíraných. Libovolný neprázdný vektor je podmnožinou generálního klíče. Takové kritérium bohužel není obecně optimální a oproti hladovému je

### 3. Algoritmy

k výpočtu složitější, ale pro určité třídy problémů by mohlo poskytnout řešení optimální.

(Pravidelné odstranění řádků a sloupců). Toto přidání kritérium vypadá na první pohled rozumně, jelikož se snažíme dosáhnout čtvercové matice jako výsledku. Má to však smysl uvažovat pouze v případech, kdy máme zajištěno, že lock-chart neobsahuje duplicity, což dává zároveň požadavek na výpočetní výkon.



## Kapitola 4

### Experimenty

V této kapitoly si ukážeme výsledky různých implementovaných algoritmů a ověříme některé hypotézy, které porovnávají jejich rychlost, co se týče času a velikosti řešení.

Pracujeme s dvěma sadami dat:

(Veřejná sada dat). Tato sada dat byla vyjmuta z diplomové práce Anny Lawer [1]. Dokonce zanecháváme pro přehlednost řetězec i stejnou notaci pojmenování příkladů. Sada se dělí na tři typy příkladů. Máme tu ty které jsou označeny jako lehké, ty jsou zhruba do velikosti 20 klíčů. Středně těžké jsou do 40 a zároveň obsahují více hlavních klíčů a jako těžké jsou považovány příklady zhruba do 100 klíčů. Sada obsahuje 20 příkladů.

(Komerční sada dat). Tato sada nemá žádné příhodné rozdělení jako sada veřejná. Snad jen poznamenejme, že se zde vyskytují příklady od velikosti řádu jednotek až po stovky. Největší příklad obsahuje zhruba 2000 klíčů a zámek. V tabulkách porovnávající algoritmy nebudeme uvažovat tento komerční příklad G11 aťkoliv byl testován. Důvodem je jeho rozsáhlost, protože jen dva z algoritmů dobře (hladová kritéria bez duplikací) a kdybychom ho započítali do průměrných hodnot výrazně by zlepšil tyto dva algoritmy co se týče velikosti řešení a neúměrně je zhoršil v rámci časového výsledku. Sada obsahuje 11 příkladů včetně vyloučeného příkladu G11. Největším dopočítaným příkladem pro všechny algoritmy se tak stává komerční příklad o velikosti zhruba 500 klíčů a zámek.

Dále budeme uvádět jednotlivé značení algoritmů v tabulkách a grafech:

H odpovídá hladovému kritériu, BC metodou biklastrování a SAT standardně implementaci SATu. Předaný parametr R nebo negace R odpovídá pořádkové redukci pohledovacího prostoru podle metody maximálního párování nebo nevyužití redukce a parametr D nebo negace D odpovídá odstranění duplikací v prostoru pro každou iteraci daného algoritmu nebo jeho nepoužití. Mějme na paměti, že SAT redukovaný odpovídá redukci prostoru klauzulí a ne parametru R. U každé metody biklastrování přidáváme

## 4. Experimenty

typ	$j_{o1j}$	$j_{o nj}$	$t_1$	$t_n$	ex [%]
SAT redukovaný	1.0000	9.1279	0.0017	4.5458	100
SAT redukovaný H R : D	16.9151	16.9151	0.0052	0.0052	100
SAT redukovaný H R D	17.2112	17.2112	0.0289	0.0289	100
SAT redukovaný H : R : D	16.9151	16.9151	0.0050	0.0050	100
SAT redukovaný H : R D	17.2112	17.2112	0.0160	0.0160	100
SAT nahoru	1.0000	9.7621	0.0016	6.7261	100
SAT dol	15.2085	15.2085	0.4741	0.4741	85

Tabulka 4.1: Průměrné hodnoty SAT pro vešnou sadu dat.

Číslo, které odpovídá parametru  $r$ , také například BC R D 15 by řeklo, že  $r = 15$ . Zároveň u této metody uvažujeme jako kritérium řezu pouze podle klíče, tedy místo kde řez provádíme bylo spočteno pouze na základě informací dané klíčem.

Dále se budeme v tabulkách odkazovat na počet klíčových řísečů  $j_{o1j}$  nám řeká velikost prvního nalezeného řezení daným algoritmem  $j_{o nj}$  řezení posledního.  $t_1$  je pak odpovídající čas nalezení prvního řezení a obdoba je čas nalezení náhodného řezení. Hodnota  $ex$  nám udává pro kolik procent například vrátil algoritmus aspoň nějaký výsledek. Speciálně u vešné sady je třeba přidáváme ke zkoumání kolonkuopt, který řeká kolik procent například došlo k optimálnímu řezení. Díky rozsáhlosti a komplexitě nevěšných dat není optimální řezení u všech těchto například známo. Hodnotu opt uvádíme pouze v Tabulce 4.3 na straně 43.

Uveřme je třeba, že zde uvedené metody biklastrování v sobě volají i výpočet hladového kritéria v případě, že chceme u jednotlivých klastrů vyhodnotit, zda se v nich nenachází vlastní lock-chart. Tato modifikace byla přidána za účelem zvýšení šance nalezení řezení.

Všechny testy probíhaly na výpočetním serveru fVUT FEL Hyperion, kde bylo k dispozici až 16 vláken a 124 GB paměti RAM. Každému algoritmu byl přidán timeout 2 hodiny, ale málokterý z prbých musel být tímto timeoutem oznámen, protože buď došel dříve anebo výpočet selhal na nedostatku paměti.

### 4.1 Hypotézy o SATu

Mějme hypotézu takovou: SAT algoritmus vrací optimální řezení.

Z uvedených tabulek vidíme, že tvrzení hypotézy není dost silné, aby platilo, neboť u vešné sady dekrementační SAT v 15% případů nevrací řádné řezení. U komerční sady dat je situace ještě horší, řádná varianta SATu zde nedobíhá sto procentně a když se podíváme u vešné sady na

	$j_{0j}$	$j_{0n}$	$t_1$	$t_n$	ex [%]
SAT redukovaný	1.0000	5.2360	0.0395	29.8451	100
SAT redukovaný H R : D	23.9639	24.8413	0.1437	0.3306	72.73
SAT redukovaný H R D	33.0140	33.8888	2.4923	3.8956	100
SAT redukovaný H : R : D	21.7335	24.0994	0.1888	0.4242	72.73
SAT redukovaný H : R D	23.8255	24.6979	1.3200	2.4596	72.73
SAT nahoru	1.0000	5.2360	0.0387	31.0238	100
SAT dol.	12.8836	12.8836	1.8745	1.8745	36.36

Tabulka 4.2: Průměrné hodnoty SAT pro nevětnou sadu dat.

typ	opt [%]
SAT redukovaný	25
SAT redukovaný H R : D	80
SAT redukovaný H R D	100
SAT redukovaný H : R : D	80
SAT redukovaný H : R D	100
SAT nahoru	50
SAT dol.	85

Tabulka 4.3: Počet optimálních řešení pro jednotlivé SAT instance.

počet nalezených optimálních řešení, tak vidíme, že pouze SAT redukovaný s odstraněním duplicit nachází vždy optimální řešení.

Reformulujeme hypotézu takto: Jestliže SAT algoritmus doběhne, pak vrací optimální řešení.

Tedy jestliže uvažujeme pouze takové příklady, kdy nám stažila k dopožitání paměť a SAT měl veškeré výpočetní prostředky aby doběhl, pak na všech takových příkladech SAT vrací optimální řešení. Na Obrázku 4.3 na straně 50 máme dva takové příklady u příkladů z větné sady M108 a M109. SAT pak vrací optimální řešení na všechny příklady, kde doběhne. Hypotézu z dat potvrzujeme, i když jsme stejného výsledku mohli docílit teoretickým důkazem správnosti hypotézy.

Důkaz. Algoritmus postupně iteruje přes všechny možné velikosti řešení, tedy od 1 do  $jK$  pokud jdeme směrem nahoru a ve směru dolů je první nalezené řešení největší možné. Máme tedy zaručeno, že SAT vyzkouší existenci všech velikostí řešení v jednom směru a maximální možnou ve směru druhém.

Předkládáme ještě jednu hypotézu: SAT dolů vrací nejlepší řešení, ale je nejméně spolehlivý

Z Tabulek 4.1 a 4.2 vidíme, že SAT dolů má nejnižší pravděpodobnost, že vůbec dosáhne na nějaké řešení v porovnání s ostatními typy SAT, tedy je skutečně nejméně spolehlivý. Avšak ve chvíli, kdy už řešení vrací,

typ	SAT dol.
H R : D	1.0343
H R D	1.0195
H : R : D	1.0586
H : R D	1.0145
BC R : D 2	1.2351
BC R D 2	1.1601
BC : R : D 2	1.1794
BC : R D 2	1.1392
BC R : D K/2	1.0806
BC R D K/2	1.0333
BC : R : D K/2	1.0946
BC : R D K/2	1.0282
Strom podle kritéria H	1.0104
Binární strom podle H	1.0104
SAT redukovaný	1.7792
SAT redukovaný H R : D	1.0195
SAT redukovaný H R D	1.0050
SAT redukovaný H : R : D	1.0157
SAT redukovaný H : R D	1.0000
SAT nahoru	1.6636
SAT dol.	1.0000

Tabulka 4.4: SAT dol. v porovnání s ostatními algoritmy.

tak vrací nejlepší, jak dokazuje Tabulka 4.4 na straně 44, která ukazuje korelaci optimálnosti dvou řešení vrácené dvěma různými algoritmy. Jestliže je tento index větší, než 1 tak je daný porovnávaný algoritmus lepší, jestliže je výsledkem korelace 1 tak jsou si algoritmy rovny a číslo menší než 1 říká, že je algoritmus horší, než ten se kterým je porovnáván. Tento index je pořízen jako srovnání všech případů, kdy oba algoritmy vrací nějaké řešení. SAT dol. s výjimkou SAT redukovaný H : R D nad všemi ostatními algoritmy ostře dominuje. SAT dol. vrací nejlepší možné řešení, pokud u nějaké vrací.

## 4.2 Hypotéza o hladovém algoritmu

Hladové kritérium je jeden z přístupů, kde pomocí mnoha iterací nezlepujeme nějaké šťastné řešení, ale pouze vracíme jedno jediné nalezené touto procedurou. Proto uvažujeme následující hypotézu: Hladové kritérium nachází nejvyšší první řešení jen mezi všemi uvedenými algoritmy. Navíc k této hypotéze prozkoumáme jestli je odpovídající řešení ze všech možných řešení minimální.

Budeme mezi sebou porovnávat pouze některé implementace jednotlivých

typ	$j_{0,j}$	$t_1$ [s]	ex [%]
H R : D	16.9151	0.0110	100
H R D	17.2112	0.0259	100
H : R : D	16.9151	0.0099	100
H : R D	17.2112	0.0221	100
BC R : D 2	8.8425	0.0064	100
BC R D 2	4.1549	0.0043	100
BC : R : D 2	8.8425	0.0059	100
BC : R D 2	4.1549	0.0040	100
BC R : D K/2	10.1949	0.0076	100
BC R D K/2	5.6676	0.0090	100
BC : R : D K/2	10.1949	0.0072	100
BC : R D K/2	5.6676	0.0071	100
SAT redukovaný	1.0000	0.0017	100
SAT nahoru	1.0000	0.0016	100
SAT dol-	15.2085	0.4741	85

Tabulka 4.5: Porovnání prvního řešení a jeho času pro veřejná data

algoritmu, neboť jestliže by nějaký modifikovaný algoritmus využíval řádkové řešení hladového kritéria, tak víme, že se první řešení bude s hladovým kritériem shodovat a čas nalezení prvního řešení nebude nikdy menší. Výjimku tvoří metoda biklastrování, kde sice hladový algoritmus využíváme ke kontrole výsledku, ale díky řezání prostoru se jeho velikost mění. Jinými slovy, hladové kritérium, zde není nějaký menší podproblém a můžeme ho uvažovat ke srovnání.

V Tabulkách 4.5 a 4.6 máme vyznačenou hodnotu průměrné velikosti prvního řešení  $j_{0,j}$  a času jeho nalezení  $t_1$ . Vidíme si, že nejmenších časů dosahují skoro všechny metody biklastrování a SAT redukovaný se SATem nahoru, ale co se týče do velikosti řešení hladové kritérium zcela jasně dominuje v obou sadách. Hypotézu potvrzujeme.

## 4.3 Hypotéza o biklastrování

Hypotéza: Volba kritéria času podle klíče dominuje nad volbou pro sloučená kritéria klíče a zámku

V minulé kapitole jsme na příklady ukazovali kritérium času, které počítalo ohodnocení klíče jako průměr hodnot sloupce matice  $M$ , kde každé políčko  $m_{i,j}$  bylo dáno násobkem počtu klíče a zámku pro daný řádek  $i$  a sloupec  $j$ . Tato hypotéza říká, že dosáhneme vždy lepšího výsledku v metodě biklastrování pokud namísto  $m_{i,j} = j \cdot k_j \cdot l_{i,j}$  hodnotu políčka matice vypočteme pouze jako  $m_{i,j} = j \cdot k_j$ . V Tabulce 4.7 vidíme, že je taková hypotéza správná pouze pro malou volbu parametru času  $r$ . Tento úkaz je zapříčiněn tím, že krité-

typ	$j_{0j}$	$t_1$ [s]	ex [%]
H R : D	31.9171	0.4075	100
H R D	33.0140	2.1703	100
H : R : D	25.4610	0.4232	100
H : R D	28.4160	2.6374	100
BC R : D 2	9.1143	0.1806	100
BC R D 2	2.6261	0.0644	100
BC : R : D 2	8.1399	0.1893	100
BC : R D 2	2.0215	0.0523	100
BC R : D K/2	11.5810	0.2349	100
BC R D K/2	4.7785	0.1557	100
BC : R : D K/2	11.4508	0.2576	100
BC : R D K/2	5.0251	0.1074	100
SAT redukovaný	1.0000	0.0395	100
SAT nahoru	1.0000	0.0387	100
SAT dol-	12.8836	1.8745	36.36

Tabulka 4.6: Porovnání prvního  $\rho$ ezení a jeho času pro nevěrná data

typ	$j_{0j}$	$j_{0n}$	$j_{0j}$	$j_{0n}$
BC R : D 2	9.6150	22.0744	1.2413	12.5370
BC R D 2	3.6823	23.1888	1.1669	12.5370
BC : R : D 2	9.2245	23.0745	1.2413	12.7735
BC : R D 2	3.3455	23.7668	1.1669	12.7735
BC R : D K/2	11.5424	25.0603	6.7747	25.4648
BC R D K/2	5.6408	26.3080	2.1696	26.4845
BC : R : D K/2	11.4946	26.0071	5.9081	25.6730
BC : R D K/2	5.7458	27.3708	2.1696	26.9275

Tabulka 4.7: Porovnání kritérií  $\rho$ ezu BC (vlevo podle klíče a vpravo podle násobení).

rium násobku se odvažuje více  $\rho$ ezat směrem ke středu lock-chartu, zatímco kritérium dané pouze klíčem má tendenci  $\rho$ ezávat lock-chart po sloupcích a tím se blížit hladovému kritériu.

Fakt, že jsou si hodnoty  $\rho$ ezení blízké pro volbu  $r = j, k, j=2$  je dán opět příkladem G09. Pokud bychom jej z měření vypustili průměrná hodnota  $j_{0n}$  spadne na  $j_{0n} = 21$ . Hypotéza tedy platí jen pro malou volbu parametru  $\rho$ ezur.

#### 4.3.1 Biklastrování proti hladovému kritériu

Tuto podkapitolu uvádíme jen jako poznámku k naměřeným datům. Jelikož se jedná pouze o jediný příklad a k testování algoritmu byla použita omezená velká sada dat, nejsme schopni vyvodit z výsledku příkladu G09 žádné pravidlo

nebo typ třídy příkladu se kterým do budoucna počítat.

Z uvedených tabulek si všímáme následujícího pozorování. Metoda biklastrování vrací pro stejnou sadu parametrů  $R$ ,  $D$  vždy stejné nebo horší řešení, než hladové kritérium s jedinou výjimkou a to u komerčního příkladu G09.

Jelikož nemůžeme v této práci komerční příklady zveřejnit, dovolujeme si přednést alespoň teoretickou úvahu, proč - jak je vidět i v příloženém Obrázku B.1 na straně 59 - v tomto příkladě dosahuje metoda biklastrování lepších výsledků (co se týče do velikosti řešení) než hladové kritérium.

Metoda biklastrování si před provedením úzku lock-chart seřadí (resp. jeho řádky a sloupce tedy klíče a zámky). Hladové kritérium tak neřadí, to si pouze napořídí vhodnou sekvenci kandidátů k odstranění. Předpokládejme, že existují dva klíče nebo zámky, které mají stejnou hodnotu pro hladové kritérium, tedy jsou navzájem z pohledu kritéria ekvivalentní. Jelikož hladové kritérium funguje deterministicky, tak v takovém případě vždy upřednostní jeden z uzlů (např. pomocí indexu daného klíče nebo zámku). Rozdíl je tedy následující: biklastrování má šanci upřednostnit díky opakovanému řazení a řazení lock-chart jiný z dvojice ekvivalentních uzlů.

Pro příklad G09 vrací metoda biklastrování o řád vyšší řešení, než metoda hladová. V tomto příkladě všechny hladová kritéria vrací velikost řešení pouze  $j_{oh} = 11$ .

## 4.4 Hypotézy o volbě parametru

Hypotéza: Algoritmus s parametrem  $D$  pro odstranění duplikace nám zaručí lepší řešení za cenu horšího času, než jeho protějšek s parametrem  $D$  zakázaným

že velikost řešení bude větší pro algoritmy se zapnutým parametrem  $D$ , než s vypnutým:  $D$  se dala už odhadnout z Tabulek 4.5 a 4.6. Předkládáme Obrázek 4.6 pro veřejná data a Obrázek 4.9 pro komerční data se srovnáním jak velikosti řešení, tak i rychlosti nalezení takového řešení. S výjimkou dvou metod biklastrování se hypotéza potvrzuje.

V Tabulce 4.8 na straně 48 máme vynesené hodnoty  $j_{oh}$  pro velikost řešení z veřejné sady a velikost řešení  $j_{oh}$  ze sady neveřejné. Tabulka nám říká o kolik procent jsme schopni najít lepší řešení při použití parametru  $D$  a za jaké procentuální zvýšení doby nalezení takového řešení. Jestliže se v tabulce nachází záporná hodnota, říká to, že nalzáme menší řešení v případě, že se nachází ve sloupci s velikostí řešení anebo v rychlejším čase v případě, že se hodnota nachází ve sloupci pro čas výpočtu. Taková situace nastává pouze u dvou algoritmů a to u: BC R 2 a BC : R 2. Tedy u metod biklastrování při parametru úzku  $r = 2$ .

#### 4. Experimenty

typ	$j_{ov}$ [%]	$t_v$ [%]	$j_{onv}$ [%]	$t_{nv}$ [%]
H R	0.8676	38.4854	1.6894	68.3817
H : R	0.8676	36.5406	5.4847	72.3479
BC R 2	1.7327	-0.3059	3.5641	-4.1101
BC : R 2	1.7327	0.3345	0.8805	-1.7965
BC R K/2	2.1444	33.4965	2.7256	69.0191
BC : R K/2	2.1444	27.2162	3.0688	69.0226
SAT redukovaný H R	0.8676	68.5278	15.4053	84.3559
SAT redukovaný H : R	0.8676	50.6691	1.2265	70.5783

Tabulka 4.8: Procentuální rozdíl v parametru D pro obě sady

Z výsledku je zřejmé, že odstranění duplicit příliš nevyplácí. Zlepšení ve většině případů přesahuje hodnotu 3% za zhruba desetinásobné zhoršení času nalezení tohoto řešení.

Hypotéza: Velikost řešení ovlivní nastavení parametru R maximálně o 20% u malých příkladů.

V Tabulce 4.9 na straně 49 vidíme, že jednotlivé dvojice algoritmů ližící se pouze v parametru R mají stejné výsledky. Tabulka obsahuje průměrné výsledky pro veřejnou sadu dat. Pro tyto příklady se tedy řešení liží o 0%.

Vezměme Tabulku 4.11 na straně 53 pro neveřejná data. Z ní vypočítáme průměrnou rozdíl dvojic řešení podle parametru R. Výsledkem je pak Tabulka 4.10 na straně 49, kde vidíme, že žádný z rozdílů nepřesahuje 20%.

Hypotézu potvrzujeme.

## 4.5 Vyhodnocení

V této části se budeme odkazovat na graf pro veřejná data z Obrázku B.2 na straně 60 a na graf pro neveřejná data z Obrázku B.3 na straně 61. Je z něj zřejmé, že pro malé příklady se chovají různé algoritmy biklastrování podobně, kdežto u větších sad, tj. ze sady dat neveřejných, volba parametru  $\alpha$  může za dražší cenu času přinést lepší řešení. V obou sadách se velmi podobně chová SAT redukovaný a SAT nahoru, což není žádné překvapení, neboť se oba algoritmy liží pouze v oné redukci, jinak iterují po stejné pořadí velikostí. Na malých datech zároveň tolik nehraje role hladového kritéria a dalších SAT s výjimkou SATu dol., který má výrazně horší čas, než výše zmíněné algoritmy. U větších typů příkladů z neveřejné sady se u roztažení nůžky v sekci hladových kritérií. Odstranění duplicity zlepšuje řešení, ale trvá ho nalézt déle a požadovaná redukce prostoru může zlepšit řešení až o 20%.



typ	$j_{0j}$	$j_{0n}$
H R : D	16.9151	16.9151
H R D	17.2112	17.2112
H : R : D	16.9151	16.9151
H : R D	17.2112	17.2112
BC R : D 2	8.8425	15.0220
BC R D 2	4.1549	15.5518
BC : R : D 2	8.8425	15.0220
BC : R D 2	4.1549	15.5518
BC R : D K/2	10.1949	16.3389
BC R D K/2	5.6676	17.0550
BC : R : D K/2	10.1949	16.3389
BC : R D K/2	5.6676	17.0550
Strom podle kritéria H	17.2112	17.2112
Binární strom podle H	17.2112	17.2112
SAT redukovaný	1.0000	9.1279
SAT redukovaný H R : D	16.9151	16.9151
SAT redukovaný H R D	17.2112	17.2112
SAT redukovaný H : R : D	16.9151	16.9151
SAT redukovaný H : R D	17.2112	17.2112
SAT nahoru	1.0000	9.7621
SAT dol-	15.2085	15.2085

Tabulka 4.9: Průměrné první a poslední řešení pro vešnou sadu příklad.

typ	rozdíl [%]
H : D	11.25
H D	7.49
BC : D 2	5.65
BC D 2	13.01
BC : D K/2	0.57
BC D K/2	2.51
SAT redukovaný H : D	4.88
SAT redukovaný H D	16.17

Tabulka 4.10: Rozdíl v parametru R pro vešnou sadu dat.

## 4. Experimenty

---

Průběh hledání řešení SAT pro příklad M108.

Průběh hledání řešení SAT pro příklad M109.

Obrázek 4.3: SAT konvergence k řešení.

Změna velikosti °ezení na veřejných datech.

Změna rychlosti času nalezení °ezení na veřejných datech.

Obrázek 4.6: Závislost na parametru D pro veřejná data

Změna velikosti °ezení na neve°ejných datech.

Změna rychlosti času nalezení °ezení na neve°ejných datech.

Obrázek 4.9: Závislost na parametru D pro neve°ejná data

typ	$ o_1 $	$ o_n $
H R $\neg$ D	31.9171	31.9171
H R D	33.0140	33.0140
H $\neg$ R $\neg$ D	25.4610	25.4610
H $\neg$ R D	28.4160	28.4160
BC R $\neg$ D 2	9.1143	33.5461
BC R D 2	2.6261	36.0257
BC $\neg$ R $\neg$ D 2	8.1399	37.8554
BC $\neg$ R D 2	2.0215	38.5279
BC R $\neg$ D K/2	11.5810	40.6963
BC R D K/2	4.7785	42.9769
BC $\neg$ R $\neg$ D K/2	11.4508	45.0280
BC $\neg$ R D K/2	5.0251	47.8791
Strom podle kritéria H	21.3019	27.4256
Binární strom podle H	21.3019	22.1495
SAT redukovaný	1.0000	5.2360
SAT redukovaný H R $\neg$ D	23.9639	24.8413
SAT redukovaný H R D	33.0140	33.8888
SAT redukovaný H $\neg$ R $\neg$ D	21.7335	24.0994
SAT redukovaný H $\neg$ R D	23.8255	24.6979
SAT nahoru	1.0000	5.2360
SAT dol	12.8836	12.8836

**Tabulka 4.11:** Pr m rn první a poslední řešení pro nevejnou sadu p íklad .



## Kapitola 5

### Závěr

Řešení lock-chartu je teoreticky i prakticky komplexní problém. Představili jsme několik druhů přístupů k nalezení maximální vlastního díle něho lock-chartu. Lokalizace takového lock-chartu může výrazně usnadnit výpočet lock-chartu obecného a tím i výpočet kapacity uzavírání reálných klíčů v výrobě. Ukázali jsme hlavní výhody a nevýhody jednotlivých přístupů. Máme velmi rychle pro uzavírání prostoru pomocí biklastrování, rychlé nalezení slušného díle něho řešení pomocí hladového kritéria, jenž se dá dále kombinovat s algoritmy, které prohledávají stromy reálných řešení v okolí nalezeného díle něho výsledku. V neposlední řadě jsme ukázali provedení na SAT, který umí v určitých modifikacích vrátit dobré výsledky, ale velmi nestabilní.

### 5.1 Vylepšení do budoucna

Představujeme pár úvah a cest, které by v budoucnu v řešení problému hledání vlastních lock-chartů stály k prozkoumání:

#### 5.1.1 Volba kritéria

Velkou otázkou zůstává jestli existuje lepší a rychlejší způsob, jak ohodnotit klíče a zámky. Podle takovýchto ohodnocení jsme například schopni volit místa uzavírání v metodách biklastrování. Cílem je toto ohodnocení spojit, což možná nejefektivnější a ideální řešení nemít potřebu opakovat v průběhu iterací.

#### 5.1.2 Existence optimálního polynomiálního algoritmu

V této práci ukazujeme polynomiální algoritmy, které však obecně nemusí vrátit optimální řešení. Existují optimální polynomiální procedury pro určité

tyto lock-chartu. Pokud bychom měli zadání bez vložek, lze uvažovat o metodě podmnožin. V situaci, kdy je zadán lock-chart s nými uspořádáním, jsme pomocí antichain také schopni problém vyřešit. Existence optimální polynomiální procedury pro obecný lock-chart je stále otevřený problém.

### ■ 5.1.3 Přiřazení algoritmů k jednotlivým třídám příkladů

Problémem zůstávají testovací sady. Jak jsme již dříve uvedli existuje minimum prací k tématu klíčových systémů a pouze jedna jediná, která představuje ve své sadě příklad. Komerčních příkladů je ještě méně a se zhruba 30 příklady lze těžko dlat statisticky signifikantní rozdílení na typy těchto příkladů. Je tedy možné, že určitý typ algoritmu velmi dobře řeší určitý typ příkladů, ale abychom tuto otázku zodpovědli, museli bychom mít přístup k většinu těchto zadání.

### ■ 5.1.4 Nalezení diagonály

Jelikož naše zadání neuvažuje předané podmínky z rozšíření lock-chartu, jedná se prakticky o nalezení nejdelší jednotkové diagonály v binární matici. Tato práce pak má šanci poskytnout potenciál pro zcela jinou výzkumnou oblast, kde je rovněž hledání takové binární diagonály žádoucí.





## Příloha A

### Literatura

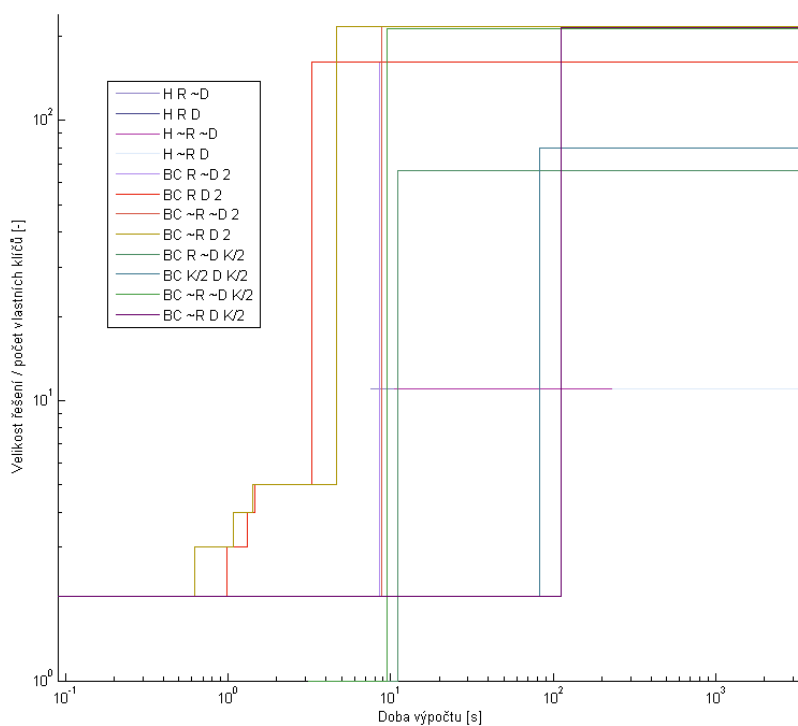
- [1] Anna Lawer. *Calculation of Lock Systems*. MA thesis, Royal Institute of Technology, 2004.
- [2] Alan Gibbons. *Algorithmic Graph Theory*. Cambridge University Press, Chapter 5, 1985.
- [3] Mirkin, Boris. *Mathematical Classification and Clustering*. Publisher Kluwer Academic Publishers, ISBN 0-7923-4159-7, 1996.
- [4] Cheng KO, Law NF, Siu WC, Liew AW, *Identification of coherent patterns in gene expression data using an efficient biclustering algorithm and parallel coordinate visualization*. BMC Bioinformatics, 2008 Apr 23.
- [5] Haifa Ben Saber, Mourad Elloumi *A novel biclustering algorithm of binary microarray data: BiBinCons and BiBinAlter*. PubMed Central<sub>R</sub>, Published online 2015 Nov 30. <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4666179/>.
- [6] Chen H-C, Zou W, Tien Y-J, Chen JJ *Identification of Bicluster Regions in a Binary Matrix and Its Applications*. PLoS ONE 8(8): e71680, 2013 <https://doi.org/10.1371/journal.pone.0071680>
- [7] Peeters R *The maximum edge biclique problem is NP-complete*. Discrete Applied Mathematics. 131: 651–654. doi:10.1016/S0166-218X(03)00333-0, 2003.
- [8] Jiří Demel *Grafy a jejich aplikace* nakladatelství Academia, Praha 2002, ISBN 80-200-0990-6
- [9] Cook, Stephen. *The complexity of theorem proving procedures* Proceedings of the Third Annual ACM Symposium on Theory of Computing. pp. 151–158, 1971.
- [10] Matoušek J., Nešetřil J. (2007): *Kapitoly z diskrétní matematiky*, Karolinum, Praha

- [11] Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001). Introduction to Algorithms (Second ed.). MIT Press and McGraw–Hill. pp. 651–664. ISBN 0-262-03293-7.
- [12] Eén N., Sörensson N. (2004) An Extensible SAT-solver. In: Giunchiglia E., Tacchella A. (eds) Theory and Applications of Satisfiability Testing. SAT 2003. Lecture Notes in Computer Science, vol 2919. Springer, Berlin, Heidelberg
- [13] Junker, U. (1998, October). Constraint-based Problem Decomposition for a Key Configuration Problem. In International Conference on Principles and Practice of Constraint Programming (pp. 265-279). Springer Berlin Heidelberg.
- [14] Bernoch, R., Kuželka, O., Železný, F. (2016). Polynomial and Extensible Solutions in Lock-Chart Solving. Applied Artificial Intelligence, 30(10), 923-941.
- [15] Levin, Leonid (1973). Universal search problems. Problems of Information Transmission (in Russian). 9 (3): 115–116. Translated into English by Trakhtenbrot, B. A. (1984). "A survey of Russian approaches to perebor (brute-force searches) algorithms". Annals of the History of Computing. 6 (4): 384–400.
- [16] Agnew, R. P. "Minimax Functions, Configuration Functions, and Partitions." J. Indian Math. Soc. 24, 1-21, 1961.
- [17] Hazewinkel, Michiel, ed. (2001) [1994], "Greedy algorithm", Encyclopedia of Mathematics, Springer Science+Business Media B.V. / Kluwer Academic Publishers, ISBN 978-1-55608-010-4
- [18] Don O'Shall. The Definitive Guide to RCM – Rotating Constant Method of Master Keying. Locksmithing Education, 2015. isbn: 9781937067137. url: [https://books.google.cz/books?id=5Hz\\_rQEACAAJ](https://books.google.cz/books?id=5Hz_rQEACAAJ).

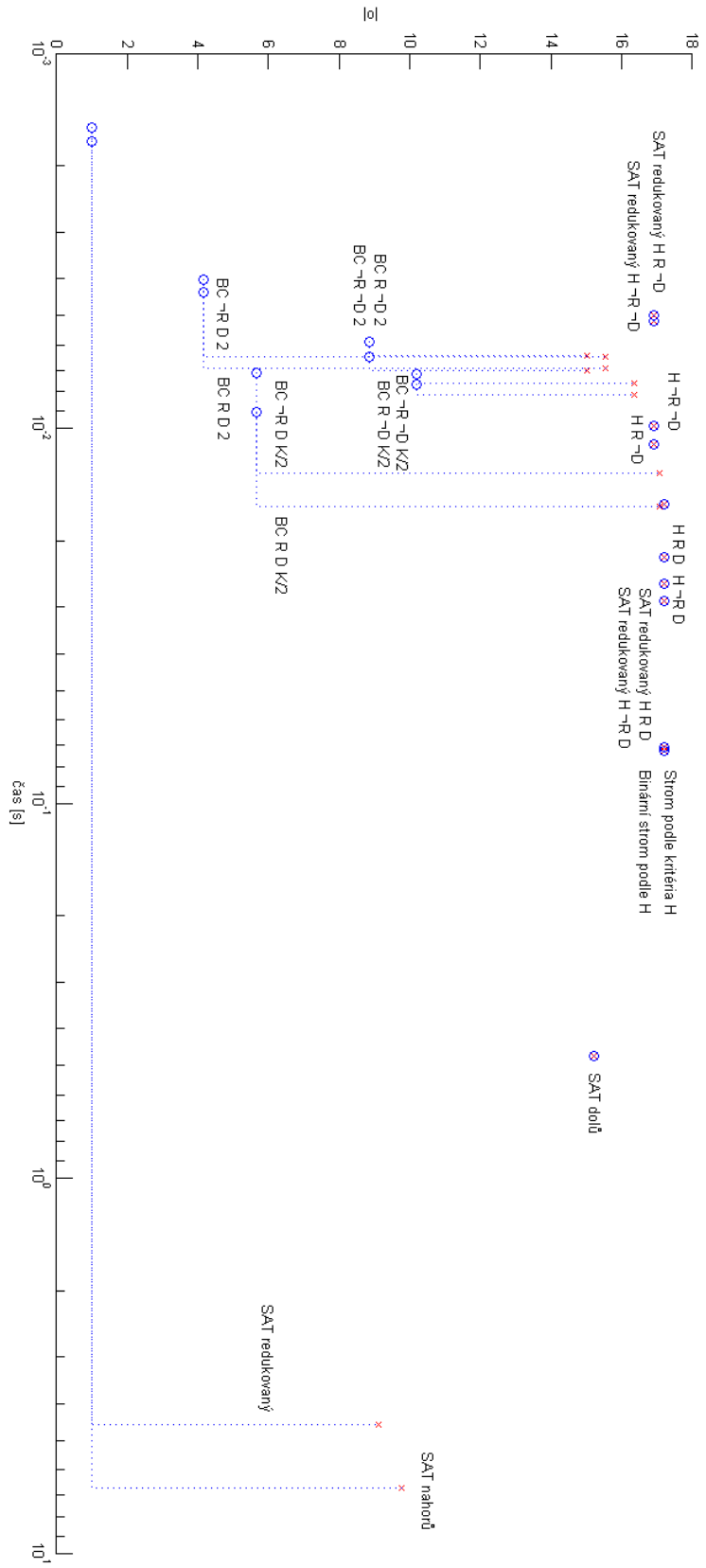
## Příloha B

### Obrázky

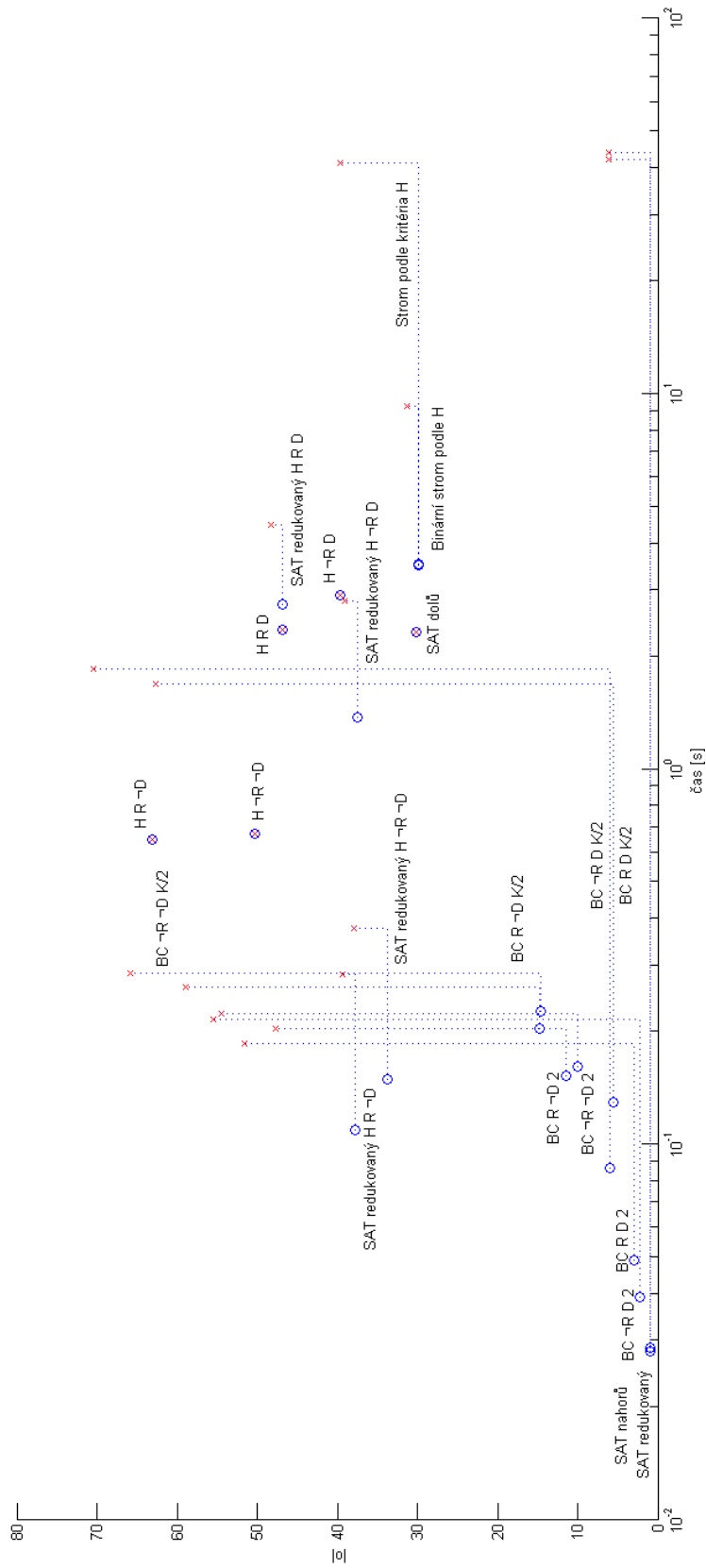
V této příloze předkládáme některé grafy a obrázky, na které se sice v textu odkazujeme, ale nebyly kvůli své velikosti použity přímo v textu.



**Obrázek B.1:** Příklad G09 konvergence k řešení.



Obrázek B.2: Výsledky pro ve ejná data.



Obrázek B.3: Výsledky pro nevejná data.





## Příloha C

### Obsah CD

V přiloženém CD se nachází zdrojové kódy v jazyce C++. Část kódu je převzata z dizertace Radomíra Šernocho [14] jako například kódy pro lock-chart a výpočet binomické vlny. Tímto souborem *eigen.cpp* obsahuje všechny uvedené algoritmy z této práce. Instrukce ke spuštění algoritmu jsou uvedené v přiloženém README na CD. Dále jsou na CD všechny uvedené grafy, naměřená data spolu s diplomovou prací ve formátu *.pdf* a matlabový skript k vygenerování uvedených tabulek z textu.





